# ON THE SEMANTICS OF THE RELATIONAL DATA MODEL

Hans Albrecht Schmid
J. Richard Swenson

Department of Computer Science
University of Toronto

The relational model for data base organization introduced clearly defined basic algebraic concepts whose properties are well understood. As a consequence, questions of a semantic nature arise. Due to the mathematical nature of the relational model, these questions cannot be answered completely by it. Therefore, a data model is discussed that takes into account basic semantic properties that object types and relationships among them apparently have in the real world. The model permits a data base to be partitioned into independent object types which are natural insertion and deletion units, and associations among them. Independent object types are described by characteristics which are meaningful modification units.

This data model permits us to analyze the ideas behind the relational theory more precisely, in particular the meaning of functional dependency and normalization. Insertion-deletion rules for relations in third normal form are developed, and proposals are made to avoid some drawbacks that exist even when relations in third normal form are used as the user interface to a data base.

Category: Data Structures

Keywords: Data Base Design, Data Structures
Relational Data Base Management Systems
Users View of Data

## 1. INTRODUCTION

In the relational model for data base organization, the basic concepts are mathematical relations and a family of operators on them, augmented with the concept of functional dependency.

The syntactic properties of these concepts have been investigated intensively and are rather well understood, in contrast to many other data models. On the other hand, the semantic properties of the relational model are not so well understood.

We are using the word 'semantics' in this paper in an a priori sense, i.e., notions in a formal system will be developed and used in a way that corresponds to pre-existing intuitive ideas.

Two formal notions are central in the relational model: normalization and functional dependency. Informal counterparts for these notions have not been offered; only the effects of their application have been shown. For example, normalization eliminates certain kinds of redundancy, and unpleasant insertion-deletion anomalies from a collection of relations [3], [4].

In section 2 we show in some detail that there are open questions concerning the semantic nature of the concepts "relation", "normalization", and "functional dependency". In order to answer these questions a basic semantic data model is introduced in section 3. As this model takes into account certain basic properties of the real world, it provides us with informal counterparts for its formal notions. The principal idea is that there are two kinds of relationships among primitive object types that are essentially different. Making the distinction between characteristics and associations allows us to partition a data base into independent object types which are natural insertion and deletion units, and associations among them. Independent object types are described by characteristics which are meaningful modification units.

The model is applied to relational theory in section 4. In particular the meaning of normalization is discussed. It is shown that five different types of relations can be distinguished among those in third normal form.

Section 5 analyzes some drawbacks that exist even when relations in third normal form are used as the user interface of a data base. Some means to avoid these drawbacks are proposed.

---

This work has been supported by the National Research Council of Canada and the Department of Computer Science of the University of Toronto.

## 2. THE RELATIONAL DATA MODEL

The relational data model proposed by Codd [2] provides a collection of relations as the user interface to a data base. The user is thereby protected from having to know about the physical real organization of the data.

A relation, R, is defined as a subset of the Cartesian product over n domains, $D_1 x D_2 x \ldots D_n$. The domains, $D_i$, i=1,...,n are given as sets. In normalized relations, the elements of a domain are not permitted to be relations or sets themselves.

A data base is a finite collection of time-varying relations defined on a finite collection of domains $\{D_i\}$. As the domains in a relation need not necessarily be distinct, (distinct) attribute names must be given to them.

An attribute, B, of a relation, R, is called functionally dependent on attribute A of R, if at every instant of time, each value of A has no more than one value in B associated with it under R. The same definition applies to a collection of attributes. We write R.A -> R.B if R.B is functionally dependent on R.A, and R.A. $\not\rightarrow$ R.B if it is not. In the sequel we will not make a distinction between intra- and interrelational functional dependencies, as we will see that there is no difference from the semantic point of view.

One of the major advantages of the relational model is that relations, their properties and possible operations on them (e.g. join, projection) are mathematically well defined. This means that when a part of the world is represented by a collection of relations, then relational theory provides mathematical tools (e.g., relational algebra) to derive different representations of the world from the given one. For example, a join can be performed if two attributes of different relations are of interest at one time.

But the relational theory gives no indication about the way in which the world is to be represented by a collection of relations. Moreover, important properties of the (derived) relations depend on what relations were intially chosen in order to represent the world. For example, consider Codd's example [2] where the difference between a collection of relations R(SUPPLIER, PART), S(PART, PROJECT) and T(SUPPLIER, PART, PROJECT) is shown.

Thus the question arises: What does it mean to collect attributes into a relation? Suppose that there are materials {A,B}, and customers, {1,2}, which are not directly connected, but which are both described in a company's data base. Is it possible to describe them in one relation that has the attributes MAT and CUST? If so, which of the following ways corresponds most satisfactorily to our view of the world?

| R (MAT | CUST) |
|---|---|
| A | 1 |
| B | 2 |

| R' (MAT | CUST) |
|---|---|
| A | ∅ |
| B | ∅ |
| ∅ | 1 |
| ∅ | 2 |

| R'' (MAT | CUST) |
|---|---|
| A | 1 |
| A | 2 |
| B | 1 |
| B | 2 |

This is one of the rather important questions that can hardly be answered from the mathematical point of view of the relational model. (Though it might very well be answered by the intuition somebody has acquired from frequent use of the relational model.)

The next question concerns the differences between different forms of a collection of relations which contain the same information and where these differences come from. In particular, one of the objectives of normalization is "to free the collection of relations from undesirable insertion, update and deletion dependencies."[3] This objective is clearly of a semantic nature, and it it not easy to see why it is completely attained by the syntactic process of normalization.

Normalization is governed by functional dependencies which either may be explicitly declared or be derived by algebraic rules from the declared functional dependencies. But are functional dependencies adequate for expressing knowledge about the world? (Some methods for data base design that are based only on functional dependencies might suggest this. [1], [10])

Recall that the definition of functional dependency is usually only applied to relations in first normal form. Therefore, if a set of values in a domain B is associated to one value in a domain A, then domain B is not considered to be functionally dependent on domain A. As a consequence, the knowledge (say) that an employee has one manager is represented by a functional dependency, but the knowledge that an employee has children is not. Why? Is there a difference between these kinds of knowledge?

For the same reason, set valued facts can not be derived either. Suppose an employee works in several departments, then EMPL $\not\rightarrow$ DEPT, and suppose that each department has only one contract type, DEPT -> CTYPE. Clearly there is knowledge about employees and contract types on which they are working. But it cannot be expressed as a functional dependency.

Another problem is related to the transitivity of functional dependencies. Suppose that an employee has one manager, and the manager earns one salary, and an employee has one job description, and each job description contains one job name. We have then:

EMPL# -> MAN# MAN# -> SAL, and
EMPL# -> JOBDESCR# JOBDESCR# -> JOBNAME

Now we can derive the dependencies

EMPL# -> SAL, and EMPL# -> JOBNAME.

The meaning of these derived dependencies is quite different. The first one denotes the salary of the manager of the employee, whereas the second one denotes the jobname of the employee. This semantic difference is not taken into account by the functional dependencies above.

These observations demonstrates that the functional dependencies are not completely adequate for expressing some knowledge about the world. Therefore, a data model that expresses a low level of semantic knowledge is discussed in the next section.

## 3. A BASIC SEMANTIC DATA MODEL

In this section, we start our discussion with a data model that represents the world by object types and relationships among these object types. This model shows inadequate structure, however. For when mapping from the world into such a model an important difference which exists in the real world is lost, namely, the difference as to whether one object is used only in order to describe another object, or whether there is a relationship among objects each of which exists on its own. We will make this distinction and call the first kind of relationship a characteristic, the second kind an association. As a consequence, the resulting model shows a structure that corresponds to our 'semantic' view of the world. In this paper the model will be explained informally. A detailed discussion of the model, its relation to other models, and precise mathematical definitions are given in [9].

### 3.1 Assumptions and Definitions

Let us assume that the real world can be represented by a set of objects, $\{obj_i\}$, (also called instances of object types) which represent the things existing in the world, and by a set of instances of relationships, $\{ir_j(obj_1,...obj_n)\}$, which represent interrelations or interactions among the objects.

Instances of object types and instances of relationships can be classified according to properties which they have in common. These classes will be called object type and relationship, respectively. A relationship can only be formed of instances whose corresponding objects belong to the same object type. Instances of object types and of relationships will be denoted by small letters, while object types and relationships will be denoted by capital letters.

For example, assume that a car "a" has the colour "maise yellow", another car "b" has the colour "metallic red", and that there are other colours, "green", "red", .... An object type CAR is formed by {a,b}, an object type COLOUR is formed by {red, green,..}, and an object type CAR_COLOUR is formed by {maise yellow, metallic red}. A relationship, CAR_HAS_COLOUR will only be defined between the object types CAR and CAR_COLOUR.

Note that there is a difference between a relationship and a relation in the sense of the relational model. Relationships represent fundamental facts about object types in our model. They are not to be further decomposed, and are supposed to have a direct semantic meaning. For example, relationships are that an employee "has" an employee number and a name, that he "lives at" an address, that he "earns" a salary, and that he "is a friend of" other employees. On the other hand, a relation EMPLOYEE(EMPLOYEE NUMBER, NAME, ADDRESS, SALARY, FRIEND) is a priori without semantic meaning. For example, what meaning is there in the fact that the domains FRIEND and SALARY appear together?

According to our view of the world, we can distinguish two kinds of relationships. Let us first explain the difference between them by an example. Suppose that an employee has an address and an eye-colour. The address and eye-colour characterize an employee, and their instances are only of interest as long as the employee whom they characterize exists (in the data base). Such a relationship will be called a characteristic. On the other hand, suppose that an employee is a friend of another employee. Both employees exist independently of one another whether or not they are friends. Such a relationship will be called an association.

The following definitions provide precise criteria for the distinction between characteristics and associations.

Definition 1: An object has exactly one occurrence when a modification, insertion or deletion of the object in one instance of a relationship in which it occurs implies its modification, insertion or deletion in all instances of all relationships in which it occurs. Otherwise, there are different occurrences of the object. If there exists different occurrences of an object such that its modification,

insertion or deletion in an instance of a relationship implies the modification, insertion or deletion in all instances of some other relationships, then it is the same occurrence of this object which occurs in these relationships.

Definition 2: An object, $obj_2$, is called a depending part with respect to an instance of a relationship, ir, if its existence implies the existence of ir and, consequently, the existence of an object, $obj_1$, such that $ir(obj_1,obj_2)$. The object, $obj_1$, is called a ruling part, and ir is called an instance of a depending relationship.

Definition 3: An instance of a relationship, $ir(obj_1,obj_2)$, is called an instance of a characteristic (relationship) if:

1. ir is a depending relationship, and $obj_1$ is the ruling part and $obj_2$ is the depending part of ir.
2. The same occurrence of $obj_2$ can only be the ruling part in other characteristics, but not occur otherwise in instances of relationships.
3. $obj_1$ and $obj_2$ are not the same object, nor the same occurrence of an object.

As a consequence of definition 3, if $obj_2$ is also a depending part in other instances of relationships then there are different occurrences of $obj_2$ occurring in them.

The object, $obj_2$, in definition 3 is called a characteristic object. An object that is not a depending part in any instance of a characteristic relationship (i.e., not a characteristic object) is called an independent object.

All instances of relationships that are not instances of characteristics are called instances of associations.

Now classes of characteristic and independent objects, respectively, can be formed. A class of instances of relationships all of which are instances of characteristics (resp. associations) is called a characteristic (resp. association).

In some cases, the same relationships may be regarded as either a characteristic or an association. Suppose that a PERSON POSSESSES a CAR. If we want to describe only the property of persons in a data base, then POSSESSES can be considered as a characteristic. But if we want to describe also that a CAR is REGISTERED BY a VEHICLE BUREAU, then POSSESSES must be considered as an association. That means that the same part of the world can be modelled from different points of view. But our precise definitions will help the data base designer to know exactly what effects his choice has.

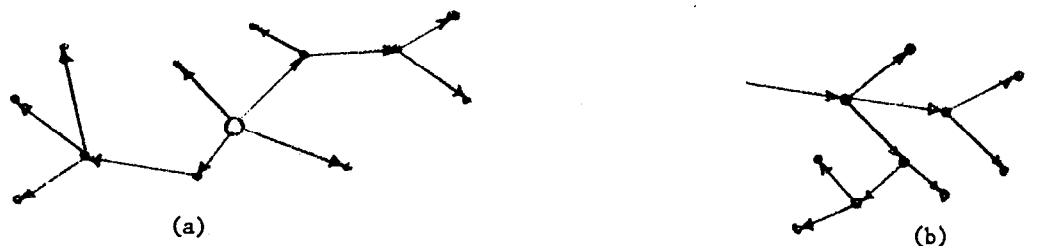
### 3.2 Structuring the Data Model

A data base consists of complex independent object types, and associations among them. A complex independent object type is formed by a kernel that is an independent object type, and by all its characteristics, and the characteristics of these characteristics, and so on. A simple independent object type is an independent object type that has no characteristics.

Correspondingly, complex characteristic object types are characteristic object types together with all their characteristics, and simple characteristics are characteristics that have no characteristics themselves.

A complex independent object type and a complex characteristic object type is formed by all independent objects and characteristic objects, respectively, which are from the same object type and have characteristics out of the same characteristic object type. We can distinguish between non-repeating characteristics, where a given object has only one characteristic object out of a characteristic object type, and repeating characteristics, where several characteristics out of one object type may exist.

A graphical representation of these definitions can be given as follows. Independent object types are represented by a "o", characteristic object types by a ".", associations by labelled undirected (thick) edges, and characteristic relationships by edges directed toward the characteristic object types. For the sake of simplicity, we only label the associations; we do not label the characteristic relationships. Instead, we name the characteristic relationships by using the name of the corresponding characteristic object type.



(a) (b)

Figure 1

Figure 1a represents a complex independent object type. The kernel is characterized by characteristics, some of which are complex and some of which are simple. All the leaves are simple characteristics. Figure 1b represents a complex characteristic object type.

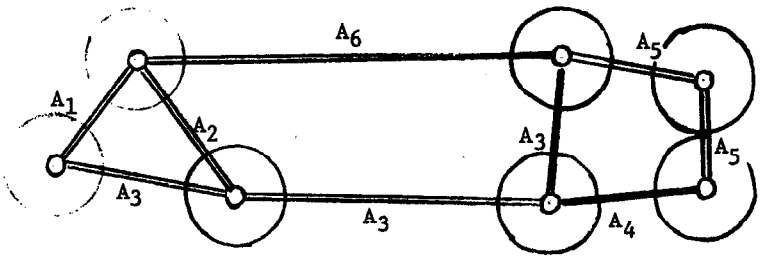
The only connections among complex independent object types are by associations among the kernels of these object types. Instances of associations can be set up only between those objects which already exist, but they can be deleted at any time provided no additional rules are given with the association.

Complex independent objects are the natural units of insertion and deletion. On the one hand, they represent in our model the things that we regard as real objects in the world. On the other hand, it is proven in [9] that the insertion or deletion of a complex independent object does not affect other ones. But complex independent objects must only be deleted when they are in no instance of an association.

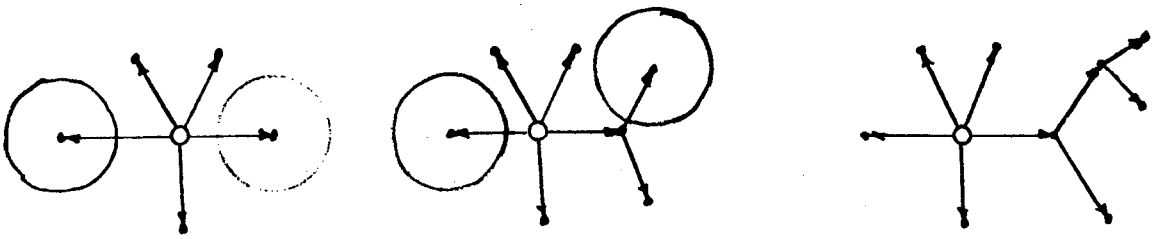
A characteristic is a unit that can be independently modified, e.g. an address, or a house number of an address. But a characteristic cannot be deleted in the same way as an independent object because the object that it characterizes continues to exist. Deleting a characteristic means that it is undefined or unknown, which is different from the fact that it does not exist.

As characteristics describe a complex independent object, they are determined by the object, i.e., there is a semantic dependency among a complex independent object and its characteristics. For example, if a book is given then it determines the key words that describe it, and a given employee determines his employee number and his address.

In a gross view of the data base, the characteristics of complex independent object types can be disregarded.




Furthermore, an object type can be regarded in different levels of detail.



As a consequence, we have the possibility of regarding a data base in different levels of abstraction. But even on the coarsest level all possible connections among complex object types are visible.

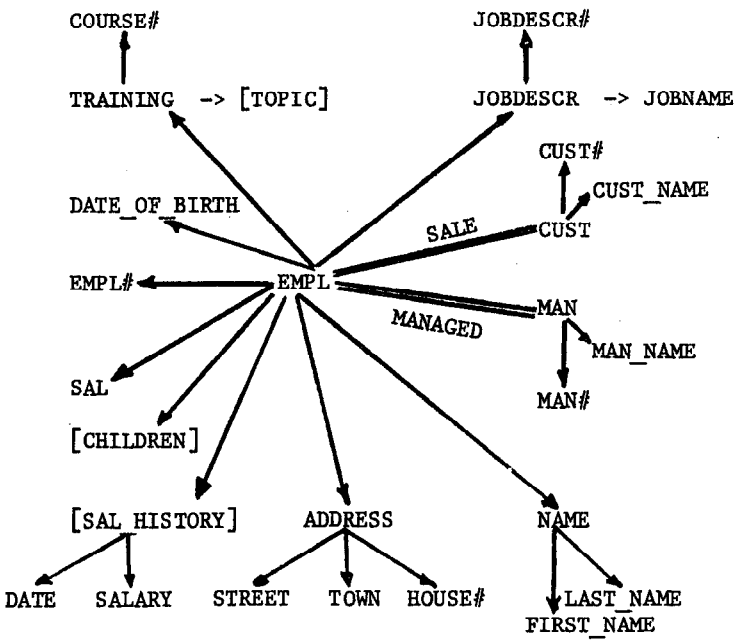
Until now we did not discuss how a specific independent object is identified in its object type. One generally uses the characteristics of an object in order to identify an object, and calls a (set of) characteristic object types that identify an object in its type, a key. When an object cannot be uniquely identified in its type by its characteristics then we will identify it by a key formed by concatenating keys of other objects with which it is related.

A theory based on our semantic model will classify different kinds of associations, such as "part of", "subconcept" of", etc., and different kinds of characteristics. (See [6]). This further classification will permit us to derive more detailed properties of object types, associations, and characteristics. For example, when an object type is the subconcept of another object type, then characteristics of this latter type are characteristics of the subconcept. This rule does not hold generally for associations, only for subconcept-associations.

A further classification of the concepts of our basic model will derive properties that do not replace those which have been discussed in this section, but that hold in addition to them. That means that when a theory is based on the distinction among object types, characteristics, and associations, then it can build upon the fundamental properties of these concepts.

## 4. ON THE SEMANTICS OF RELATIONAL NORMALIZATION

In this section, the data model and the results developed in section 3 will be applied to Codd's relational model. All examples will refer to figure 2. For the sake of simplicity, we will use "characteristic" instead of "characteristic object type" when no misunderstanding is possible.




Repeating characteristic object types are indicated by [..].

Basic semantic data model

EMPLOYEE (EMPL#, FIRST_NAME, LAST_NAME, STREET, TOWN, HOUSE#, DATE_OF_BIRTH, SAL, JOBDESCR#, COURSE#)

JOBDESCR (JOBDESCR#, JOBNAME)

SALHISTORY (EMPL#, DATE, SALARY)

MAN (MAN#, MAN_NAME)

CUST (CUST#, CUST_NAME)

SALE (CUST#, EMPL#)

TRAINING (COURSE#, TOPIC)

CHILDREN (EMPL#, CHILD_NAME)

MANAGED (EMPL#, MAN#)

Relational model in third normal form

Figure 2. Contrasting data base representations

### 4.1 Semantic Dependencies in the Relational Model

In the following we will analyze how semantic dependencies are represented by functional dependencies in the relational model. Recall that domains in normalized relations correspond to simple object types in the basic semantic model, and each value in the domain corresponds to a simple object.

Let us consider first associations among complex independent object types. There are no semantic dependencies and, as a consequence, in general no functional dependencies. Neither is it the case that all objects of one object type need to participate in an association, e.g. not every employee must be associated with a customer. That means that a relation representing an association is not non-loss joinable with other relations on a domain formed by the key of a complex independent object type.

But in the relational model, it is possible that additional time independent rules resulting in a functional dependency or in non-loss joinability are imposed on an association. For example, a rule can indicate that every employee must have exactly one manager.

Next let us consider characteristics within complex independent objects types.

An instance of a complex independent object type, e.g. EMPL in figure 2, determines necessarily all its characteristic (objects), independently of whether they are repeating ones or not. It is identified by one or more candidate keys, each of which may consist of instances of either one simple characteristic object type, e.g. EMPL#, or a set of them e.g., (NAME, ADDRESS, DATE OF BIRTH). As a consequence, the keys determine the characteristics of the object, albeit in an indirect way.

It is also possible for a complex characteristic object type to have its own key. For example, JOBDESCR# is a key for a JOBDESCR. Such a key will be called a characteristic key in the sequel. Note that the characteristics of which the complex characteristic is composed are semantically determined by both the characteristic key and the candidate keys of the object possessing the characteristics, e.g. JOBNAME by JOBDESR# and EMPL#.

Apart from these two cases, another case must be considered. If one of the candidate keys is composite, then a characteristic may determine one of the key's constituents. For example, imagine a post office file where municipality and street address form a composite key and postal code is a characteristic. If postal codes are only assigned within municipalities (as in Canada), then the code determines the municipality. In the following it is supposed that no semantic dependencies of that kind exists.

Let us now consider non-repeating simple characteristics, S. They are obviously functionally dependent on candidate keys, P-KEY, as for every key there is only one simple characteristic in its object type. We write

P-KEY -> S.

In our example, SAL and DATE_OF_BIRTH are such characteristics.

A complex characteristic is represented in a normalized relation by splitting it up into single characteristics.

If a non-repeating complex characteristic has no characteristic key, then it can be split up into non-repeating simple characteristics, $S_i$. In our example, NAME can be split into FIRST_NAME, LAST_NAME. These simple characteristics are functionally dependent on the primary key(s)

P-KEY -> $S_i$.

If a complex characteristic itself has a characteristic key, C-KEY, e.g., JOBDESCR# in the complex characteristic JOBDESCR, then this key is functionally dependent on the candidate key(s)

P-KEY -> C-KEY.

The same considerations as were applied to a complex independent object type can now be applied to a complex characteristic, C, with a characteristic key, C-KEY. When the characteristics of C can be split up into non-repeating simple characteristics, e.g., JOBNAME in JOBDESCR, or have a non-repeating characteristic with a key, $C\text{-}KEY_j$, then

C-KEY -> $S_j$, C-KEY -> $C\text{-}KEY_j$.

Furthermore, as we remarked earlier,

P-KEY -> $S_j$, P-KEY -> $C\text{-}KEY_j$.

Clearly this process can be repeated.

Characteristics, S, $S_i$, and C-KEY are called directly determined characteristics. That means that S, $S_i$, and C-KEY are not determined by any other key than P-KEY.

According to our assumptions, there are no other non-trivial[1] functional dependencies within complex independent object types.

Repeating characteristics do not result in functional dependencies as for every key there may be more than one simple characteristic in its object type. Otherwise the same considerations apply as in the case of non-repeating characteristics. If we denote the corresponding sets by a *, then for repeating characteristics,

P-KEY $\not\rightarrow$ S*
P-KEY $\not\rightarrow$ $S_i$*
P-KEY $\not\rightarrow$ C*-KEY
C-KEY $\not\rightarrow$ $S_j$*
C-KEY $\not\rightarrow$ $C\text{*-}KEY_j$.

Perhaps other kinds of semantic dependencies not covered by the cases discussed above could be constructed. Such possibilities will be disregarded because they do not seem to be of practical importance. There is then no difference between the two third normal form definitions (compare [3], [5]). We will use the Boyce-Codd third normal form in a proof because it is simpler to deal with.

Let us restate briefly the results of this section. By the transition to functional dependencies, some knowledge of the semantic dependencies among objects has been lost. Instead, purely syntactic functional dependencies resulting from rules additionally imposed on associations have been added.

## 4.2 Semantics of Third Normal Form

Each subgraph in the graph representation of our semantic data model can be considered as a relation in the relational model. (Some of these subgraphs may not be very meaningful, though.)

When a complex independent object type is represented entirely by one relation (which may contain associations also) then the transition to third normal form will perhaps split up the relation. On the other hand, when a complex independent object type is represented by several relations then each such relation must contain at least one of the object's candidate keys or one of the characteristic keys. By normalization to the third normal form one will obtain (as is described later in detail) subrelations each of which contains the candidate key(s) or characteristic key(s) and which describes the object partially. Those subrelations that describe the same object type are non-loss joinable. By the transition to optimal third normal form, they will be collected into one relation. This leads to Theorem 1.

Theorem 1: Relations that are in optimal third normal form can be subdivided into five different types:

1. Each relation represents an independent object type. The relation then contains all candidate keys {P-KEY} of the type, all non-repeating simple characteristics, {S} and {$S_i$}, which are directly determined by {P-KEY}, and all non-repeating characteristic keys, {C-KEY}, directly determined by {P-KEY}. (See remark 2 below.)

[1] Trivial functional dependencies are those which arise from projections, e.g., (FIRST,LAST) -> LAST.

2. Each relation represents a repeating characteristic object type. It contains a candidate key, P-KEY, and one repeating simple characteristic, S*, or repeating simple characteristics, $S_i$*, or one repeating characteristic key, C*-KEY, of a complex characteristic object type. (See remark 3 below.)
3. Each relation represents a complex characteristic object type with a key. It contains one or more equivalent characteristic keys, {C-KEY}, and all non-repeating simple characteristics, $\{S_j\}$, and characteristic keys, $\{C\text{-}KEY_i\}$.
4. Each relation represents a repeating complex characteristic object type with a key. It contains a candidate characteristic key, C-KEY, and repeating simple characteristics, $S_j$*, or one repeating characteristic key, $C^*\text{-}KEY_j$.
5. Each relation represents an association. When the association is functional, then the relation has a candidate key, P-KEY, as a key. (See remark 2 below.) When it is non-functional, then the key is formed as a composite of all the candidate keys of the complex independent object types that are in the association.

The proof will be given in the Appendix.

Remarks: 1. The relations of the third and fourth type correspond in their structure to those of the first and second type. They simply describe complex characteristic object types as opposed to complex independent object types.

2. The optimality criterion requires that the number of relations be as small as possible in third normal form. According to it, non-loss joinable functional associations would be combined on a P-KEY with descriptions of a complex independent object type having the same P-KEY. That is, a relation of type 1 would be joined with a corresponding relation of type 5 resulting in a relation which would contain additionally one candidate key of each of the associated object types. However, we believe that it is important to distinguish clearly between associations and characteristics. We recommend that the optimality criterion, which is important for collecting together as many characteristics as possible into a relation, not be applied to non-loss joinable functional associations.

3. When the optimality criterion mentioned above is applied to relations of type 2 or 4, then one relation would contain all repeating characteristic object types that characterize one complex object type. As a consequence, the Cartesian product of all characteristic objects in the different object types has to be created. Instead, we recommend using one relation for each repeating characteristic object type. The following example shows the difference in the two cases:

| EMPL# | CHILD_NAME | DATE | SAL |
|---|---|---|---|
| 1 | Hans | 70 | 10000 |
| 1 | Dick | 70 | 10000 |
| 1 | Hans | 72 | 12000 |
| 1 | Dick | 72 | 12000 |
| 1 | Hans | 73 | 15000 |
| 1 | Dick | 73 | 15000 |
| 2 | Eva | 70 | 20000 |

| EMPL# | CHILD_NAME |
|---|---|
| 1 | Hans |
| 1 | Dick |
| 2 | Eva |

| EMPL# | DATE | SAL |
|---|---|---|
| 1 | 70 | 10000 |
| 1 | 72 | 12000 |
| 1 | 73 | 15000 |
| 2 | 70 | 20000 |

4. This classification of relations into different types is comprehensive with respect to our basic data model. Each concept introduced by the model is covered by a type of relation and vice versa.

But there are concepts which are not discussed in the data model, e.g., relationships among relationships. This will be discussed in [9]. It seems that characteristics of associations or characteristics themselves are of some interest. For example, consider a relationship

SUPPLY (PART, SUPPLIER, QUANTITY)

where the domain QUANTITY indicates the possible quantity in which a SUPPLIER can SUPPLY a PART. The object types PART and SUPPLIER are in the association SUPPLY which has the characteristic QUANTITY.

Though these additional concepts are not covered by the five types of relations, it should be pointed out that characteristics of a relationship may be represented within the same relation as that which represents the relationship. Thus, characteristics of characteristics would be represented within relations of types 1 - 4, while characteristics of an association would be represented within a relation of type 5.

We will use Figure 2 to illustrate the relation types of Theorem 1.

The complex independent object types in the example are

EMPL, CUST, and MANAGER

while all the remaining object types are characteristic object types. The complex independent object types EMPL and CUST are in an association SALE, and EMPL and MANAGER are in a functional non-loss joinable

association, MANAGED. Both of these associations lead to relations of type 5.

SAL, DATE_OF_BIRTH, and EMPL# are simple non-repeating characteristics with EMPL# being a candidate key. These and the simple characteristics which are grouped into the (keyless) complex characteristic types, NAME, and ADDRESS must be combined into a type 1 relation keyed on EMPL#. Furthermore, the keys of the complex characteristic types JOBDESCR and TRAINING will be included in the type 1 relation keyed on EMPL#. These two complex characteristic object types will form relations of a different type.

EMPL# and the repeating simple characteristic, CHILD_NAME, form a relation of type 2. EMPL# and the repeating complex characteristic, SAL_HISTORY, form also a relation of type 2 over the domains EMPL#, SAL, and SAL_DATE.

JOBDESCR being a non-repeating characteristic with a key, JOBDESCR#, and a simple characteristic, JOBNAME, forms a type 3 relation.

Finally, TRAINING, being a characteristic object type with characteristic key, COURSE#, and a repeating simple characteristic, TOPIC, forms a type 4 relation.

Theorem 1 shows from where the important benefits of normalization originate. The most important benefit is that normalization separates the descriptions of complex independent object types from each other, and from associations. Conversion to second normal form accomplishes this separation for object descriptions that are combined with non-functional associations. For example, the relation

(EMPL#, DATE_OF_BIRTH, SAL, CUST#, CUST_NAME)

is split up into the three relations:

(EMPL#, DATE_OF_BIRTH, SAL),
(CUST#, CUST_NAME),
(EMPL#, CUST#)

each of which represents either a complex independent object type or an association.

However, conversion to second normal form does not split up the relation

(EMPL#, DATE_OF_BIRTH, SAL, MAN#,MAN_NAME)

where EMPL# -> MAN# and every employee has a manager (i.e., non-loss joinable!). Only conversion to third normal form splits up functionally associated independent object types. The result is two relations:

(EMPL#, DATE_OF_BIRTH, SAL, MAN#)
(MAN#, MAN_NAME).

As we remarked earlier, our opinion is that the association between employee and manager should be separated from the description of the employee. The collection of relations then becomes:

(EMPL#, DATE_OF_BIRTH, SAL),
(EMPL#, MAN#)
(MAN#, MAN_NAME).

After the separation that is caused by normalization, every complex independent object and every instance of an association is represented only once in the collection of relations. This is the reason why normalization eliminates insertion and deletion dependencies resulting from multiple representations of a complex independent object or an association in a relation.

Other benefits with respect to characteristics result from normalization. Conversion to second normal form separates non-repeating from repeating characteristics, whereas conversion to third normal form separates characteristics that possess a characteristic key.

### 4.3 Insertion-Deletion Rules for Relations in Optimal Third Normal Form

In cases when a complex independent object type is described completely by non-repeating simple or complex characteristics without characteristic keys, then normalization delivers the complete description of an object type by a relation of type 1. Practically all examples given in the literature about relational normalization are of that kind. In such a case, a tuple in that relation is a meaningful unit for insertion or deletion, because it represents completely a complex independent object.

Rules that concern the modification of one tuple in a relation have been discussed by Heath [8]. But even in the simple case when a complex independent object is completely represented by a tuple of a relation in third normal form two additional rules must be observed. They concern the (interrelational) influences of tuples in different relations. From section 3 we know that when independent objects are deleted then they must be in no associations. This leads to rule 1.

Rule 1: A tuple in a relation of type 1 can be deleted only if none of its candidate keys is contained in any other tuple of a relation of type 5.

In figure 2, an employee can only be deleted if he is not in an association with a customer in the relation SALE, or a manager can only be deleted if he is not managing an employee (in relation MANAGED).

An inverse rule is valid for the insertion of associations as associations can only be made between existing independent objects.

Rule 2: A tuple may be inserted into a relation of type 5 only if all foreign keys of the tuple exist already as candidate keys in relations of type 1.

In figure 2, a tuple can only be inserted into the relation SALE, if the corresponding employee and customer exist.

When a complex independent object type has complex characteristics with keys or repeating characteristics, then its description is distributed over more than one relation. From section 3 we know that a complex independent object must be inserted or deleted with all its characteristics at one time. In the case that it has non-repeating characteristics with keys, then for each key there is an additional relation of type 3. Compare in figure 2 the relation JOB_DESCRIPTION. Then the following rule applies:

Rule 3: If a relation of type 1 or 3 contains foreign keys that are candidate keys in a relation of type 3, then the corresponding tuples in all these relations must be inserted or deleted at the same time.

For repeating characteristics, it is also true that they are to be inserted or deleted at one time with the complex independent object which they characterize. But now several tuples that form a repeating characteristic are affected by an insertion or deletion. This leads to the following rule.

Rule 4: If a candidate key of a relation of type 1 (resp. 3) is contained in a relation of type 2 (resp. 4) then insertion of tuples in the type 2 (resp. 4) relation must be preceded by an insertion of a corresponding tuple in the type 1 (resp. 3) relation. The reverse is the case when a deletion is performed.

Note that modifications of a repeating characteristic, i.e. insertion or deletion of only some of the tuples that form the repeating characteristic, are not taken into account by rule 4.

Based on the semantic data model, an insertion-deletion theory for a multirelation data base has been developed. Modifications are to be regarded in the same framework. A modification of an association must be considered as a deletion followed by an insertion. In the case of characteristics, the modification of a foreign key in a relation of type 1 or 3 has implications for tuples with the same key in other relations of type 3, and possible further implications for tuples in relations of type 4. The underlying semantic principle is that the modification of a characteristic has influences on sub-characteristics, and vice versa. This will not be discussed in more detail here.

## 5. CONSEQUENCE FOR RELATIONAL DATA BASE DESCRIPTION

A relational data base is described on the schema level by the data model definition. It defines for each relation its name, all attribute names and the primary key, and furthermore, the underlying domains [7]. Additionally, information about time independent constraints such as functional dependencies, set inclusion declarations and dynamic integrity constraints is given.

As a consequence, the description of a data base by a relational schema is very flat. That means that the structure inherent to a certain data base is contained only implicitly in the schema; it cannot be recognized easily.

There are two types of structure which are hidden in a relational data model definition:

1. Intrarelational: Existing natural hierarchical structure of characteristics (e.g. NAME, ADDRESS) is lost because complex characteristics must be split up into attributes. It is hard to understand what the collection of attributes

STREET, LAST_NAME, HOUSE#, FIRST_NAME, TOWN

means.

2. Interrelational: As there is only one type of relational structure one cannot distinguish whether a relation describes a complex independent object type (as do relations of type 1 and 2), a complex characteristic object type (as do relations of type 3 and 4), or as associations between independent objects (as do relations of class 5).

This loss of structure has two consequences:

a) Without knowing at least intuitively to which type a relation belongs, no consistent modifications, insertions or deletions of tuples in the relation can be made. This is because the consequences of a modification on other relations are different according to their types. (Compare rules 1 - 4.)

b) Questions in a data base can be based either on a commonality of underlying domains, or on an explicit connection between domains. In the latter case, possible connections among domains of different relations must be recognized. But this may be rather difficult in the relational model. Consider the relational data model definition in Figure 2, and suppose that the domain TOPIC in the relation TRAINING and the domain JOBNAME in the relation JOBDESCR. How are they connected? Either these attributes could be foreign keys themselves, whence we must look for where they occur in other relations, or a connection could go through the primary keys, whence we must look for those occurrences. Furthermore, the connections could go through several relations.

In the basic semantic model more of the structure of the data base is shown explicitly and connections between attributes are seen more easily.

Based on the semantic model, two alternatives are proposed as to how a relational data model definition should be specified. These proposals maintain the advantages but avoid the disadvantages of the relational model.

In the the first approach, a two level network is superimposed onto a relational schema, one level representing the associations, the other representing separately each complex object. From both levels, there are references to corresponding relations in the relational schema and vice versa.

In this approach, the network shows clearly the possible connections among relations and allows hierarchical grouping of attributes. But all advantages of the relational model are preserved.

In the second approach, the structural information is provided by a grouping of relations in declarations with indications of their semantic type. We denote this in our example below by (#). Further syntactic means should permit attributes to be grouped in a relation. Let us demonstrate using the data model definition of Figure 2.

```
COMPLEX INDEPENDENT OBJECTS

EMPL            WITH KEY          EMPL#
                IN ASSOCIATIONS       SALE(5), MANAGED(5)
                DESCRIBED BY      EMPLOYEE(1)
                                  CHILDREN(2)
                                  SAL_HISTORY(2)
                                  JOB_DESCR(3)
                                  TRAINING(4)

CUST            WITH KEY          CUST#
                IN ASSOCIATIONS       SALE(5)
                DESCRIBED BY      CUST

MAN             WITH KEY          MAN#
                IN ASSOCIATIONS       MANAGED(5)
                DESCRIBED BY      MAN


ASSOCIATIONS

SALE(5)         WITH KEY          (EMPL#, CUST#)

MANAGED(5)      WITH KEY          EMPL#


RELATIONS

EMPLOYEE(1)     WITH KEY          EMPL#
                (EMPL#,
                 NAME
                     FIRST_NAME
                     LAST_NAME
                 ADDRESS
                     TOWN
                     STREET
                     HOUSE#
                 DATE_OF_BIRTH
                 SAL
                 JOBDESCR#
                 COURSE#)

CHILDREN(2)     WITH KEY          EMPL#, CHILD_NAME
                (EMPL#, CHILD_NAME)

SAL_HISTORY(2)  WITH KEY          EMPL#, DATE
                (EMPL#
                 SAL_HISTORY
                     DATE
                     SALARY)

.........

DOMAINS

EMPL#           INTEGER(10)
FIRST_NAME      CHARACTER(20)
......
```

The grouping of attributes may be shown in other ways, e.g. by using nested brackets in a linear notation.

## 6. CONCLUSIONS

It has been demonstrated that when the relational model is considered purely mathematically then some questions of a semantic nature are left open. In order to resolve these questions a basic semantic data model was introduced. Results on the structuring of data were then derived. They apparently corresponded to what we regard as "real world" structure.

Applying these results to the relational data model, the concepts of functional dependency and of normalization were analyzed from a semantic point of view. In particular, it was shown that normalization delivers apart from one exception, relations such that each of them represents one of the concepts developed in the basic semantic data model. This yielded five different types of relations in optimal third normal form. Insertion-deletion rules derived from the basic model were applied to these types.

Finally, a form was suggested for the specification of the relational data model which incorporated the knowledge about the basic semantic structure of a data base.

## 7. ACKNOWLEDGEMENTS

We would like to thank P.A. Bernstein, M. Brodie, F. Lochovsky and D. Tsichritzis of the Department of Computer Science at the University of Toronto for their many hours of conversation and friendly criticism.

In particularly, we would like to thank Dr. E.F. Codd of IBM San Jose Research Center for his encouragement and very helpful comments on an earlier draft of this paper.

## Appendix

Here we present a proof of Theorem 1 of section 4.2.

Proof: As a result of our earlier assumptions, we may use the Boyce-Codd definition [5] of third normal form in our proof.

a) If there is a relation from one of these types then it is in Boyce-Codd optimal third normal form. It can easily be seen that according to the functional dependencies derived in 4.1:

- in the first and third type simple characteristics and characteristic keys are functionally dependent only on the primary and characteristic keys, respectively.
- in the second, and fourth types there are no functional dependencies at all.
- in the fifth type there are no functional dependencies or there are functional associations.

b) If a relation is in Boyce-Codd optimal third normal form, then it is in one of these types.

- Suppose a relation contains a candidate key. Then it contains all candidate keys because of the optimality requirement and because keys are not transitively dependent on each other. It must contain all other non-repeating simple characteristics and characteristic keys for the same reason. It must not contain characteristics that are dependent both on the primary keys and a characteristic key (because of exclusion of strictly transitive dependence [3]). As there are no further functional dependencies on the primary keys this relation is in Boyce-Codd optimal third normal form, and of the first type. The same argument applies for relations of the third type.
- Suppose a relation contains the candidate key and one repeating characteristic or characteristic keys. As a consequence, it is of type 2 or 4, respectively.
- If a relation is not covered by the previous two cases, then it must be a relation containing a candidate key and either representing a non-functional association, or representing a functional association. This is precisely type 5.

## References

(1) P.A. Bernstein, J.R. Swenson and D. Tsichritzis, "A unified approach to functional dependencies and relations", Proc. of ACM SIGMOD Workshop, San Francisco, May 1975.

(2) E.F. Codd, "A relational model of data for large shared data banks", Comm. ACM, vol. 13, no. 6, June 1970, 377-387.

(3) E.F. Codd, "Further normalization of the data base relational model", Courant Computer Science Symposia 6, Data Base Systems, New York City, May 24-25, 1971, Prentice-Hall.

(4) E.F. Codd, "Normalized data base structure: a brief tutorial", Proc. of 1971 ACM SIGFIDET Workshop on Data Description, Access and Control, San Diego, Nov. 1971, 1-18.

(5) E.F. Codd, "Recent investigations in relational data base systems", Proc. of IFIP '74, North Holland Pub. Co., Amsterdam 1974, 1017-1021.

(6) P. Cohen, J. Mylopolcus and A. Borgida, "Some aspects of the representation of knowledge", Working Paper #3, TORUS Project, Department of Computer Science, University of Toronto, submitte to IJCAI, Tbilisi, Sept. 1975.

(7) C.J. Date and E.F. Codd, "The relational and network approaches: Comparison of the application programming interfaces", IBM Research Report RJ 1401, June 1974.

(8) I.J. Heath, "Unacceptable file operations in a relational data base", Proc. of 1971 ACM SIGFIDET Workshop on Data Description, Access and Control, San Diego, Nov. 1971, 19-33.

(9) H.A. Schmid and J.R. Swenson, "A basic semantic data model", to appear.

(10) C.P. Wang and H.H. Wedekind, "Sequent synthesis in logical data base design", IBM Journal Researcl and Development, vol. 19, no. 1, January 1975, pp. 71-77.