DESIGN GOALS FOR RELATIONAL AND DBTG DATABASES

by

E. Wong and R.H. Katz

Memorandum No. UCB/ERL M78/89

15 December 1978

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Design Goals for Relational and DBTG Databases

E. Wong and R. H. Katz
University of California, Berkeley

Abstract

In this paper the goals of logical design for relational and DBTG databases are stated in terms of the operational requirements of database maintenance and redesign. A set of mapping rules for attaining these goals are then secified. The relational schemas that result from an application of the mapping rules are shown to be in the fourth normal form.

1. Introduction

The semantic sparseness of the relational data model, while responsible for much of its power in retrieval and data manipulation, works against it in update operations. This defect has long been recognized [CODD71]. One approach to its correction is to augment the semantics of the relational model with _functional dependencies_ and _multivalued dependencies_, and through these structures define normal forms for relations.

The idea of normalization in database design was proposed by Codd [CODD71], and subsequently two distinct variations to this approach have emerged. One, called

Research supported by the U.S. Army research Office Grant DAAG29-76-6-0245 and the U.S. Air Force Office of Scientific Research Grant 78-3596.

decomposition, is a refinement of Codd's normalization [FAGI77b]. Here, one starts with a collection of relations having atomic values (i.e., in first normal form) and a specified list of multivalued dependencies among the attributes of these relations. The design proceeds by decomposing the relations in the initial collection, until every relation is in the fourth normal form (4NF). It is important to note that a multivalued dependency is defined in the context of a specific relation. As such, it is a property of the attributes of a relation, not their underlying domains.

An alternative to decomposition is synthesis [BERN76]. Here, one begins with a set of functional dependencies, somewhat modified in the definition so as to be independent of the context of specific relations. The design then proceeds by synthesizing a collection of relations in the third normal form (3NF).

Fagin [FAGI77b] has elaborated on the difficulties of synthesis as a design procedure. Yet, there is much about it that we like. It seems to us unsatisfactory that the principal semantic objects of the design model should depend on the context of the specific relations, which after all are only convenient ways of grouping data.

The difficulty with synthesis, we believe is due to the fact that even augmented with functional dependencies, the relational model is still semantically inadequate for design. Semantic objects that should be distinguished are not, with the result that simple ideas become difficult to explicate. This fact was clearly recognized in [SCHM75] and they proposed a "basic semantic model" in order to distinguish between "characteristics" and "associations" among primitive object types. They showed that violations of 3NF are results of combining in the same relation different semantic objects which are independent.

Normal forms for DBTG data models do not appear to have been considered, but clearly such models are also not immune from update anomalies.

In this paper our primary objective is to state some explicit and operational design goals for both relational and DBTG data models and to propose a procedure by which such goals can be realized. In the process both normal forms and the relational-DBTG schema translation problem are illuminated. While we believe that a data model used for design must have sufficient semantics to allow the goals of design to be clearly stated, we also believe that excessive semantics is a burden, and every semantic object introduced should be justified by a specific operational purpose. Our concern is with the operational effects of semantics and not semantics per se.

2. A Design Model

Our point of departure is to choose a basic semantic data model which is to be used to specify a design schema.

The model as such is not new, being essentially a simplification of both the entity-relationship model [CHEN76] and the semantic data model introduced in [SCHM75]. BY a "data model" we shall mean a collection of data object types introduced to represent data, while by a "schema" we shall mean a specific choice of data objects to represent a database. Thus, for example, the relational data model consists of: rlation, domain, tuple, attribute, etc.,while a relational schema consists of a specific collection of relations.

For each instance of time t, let $E_1(t)$, $E_2(t)$, ... , $E_n(t)$ be n distinct sets, which we shall call entity sets. An entity set, usually referenced by name, is in reality a family of sets which change as members are inserted and deleted.

A property of an entity set E(t) is a one-parameter family of functions f_t, mapping at each t E(t) into some set V of values. Observe that implicit in this definition are the requirements that: (a) at each t f_t is defined on all of E(t), and (b) for every e in E(t) the value $f_t(e)$ is unique.

As an example, consider the following entity sets and properties:

entity set	properties
emp	ename,birthyr
dept	dname,location
job	title,status,salary

A relationship R_t among entity sets $E_1(t)$, ... , $E_n(t)$ is a time-varying relation, i.e., at each t R_t is a subset of the cartesian product $E_1(t) \times E_2(t) \times \ldots \times E_n(t)$. For example, the following two relationships specify respectively the employees qualified to hold each job, and the jobs allocated to each department:

```
qualified (job,emp)
allocation (dept,job)
```

A relationship may optionally have a property defined on it. For example, "number allocated" is such a property for "allocation." We assume that the relationships specified in a design schema are independent and indecomposable. Independence means that no relationship is derivable from other relationships, and indecomposability means that no relationship is equal to the join of two of its projections for all time.

We shall say that a binary relationship R_t on entity sets $E_1(t)$ and $E_2(t)$ is <u>single-valued</u> in $E_1(t)$ if each entity of $E_1(t)$ occurs in at most one instance of R_t. If each entity in $E_1(t)$ occurs in exactly one instance of R_t, we shall call R_t an <u>association</u>. At each t an association $R_t(E_1(t), E_2(t))$ is a <u>function</u> which maps $E_1(t)$ into $E_2(t)$. For example, consider the binary relationship mgr(dept,emp). If each dept is required to have one and only one manager at all times, the mgr(dept,emp) is an association. If a dept can be temporarily without a manager, then mgr(dept,emp) is single-valued in dept, but not an association. As we shall see, the distinction between an association and other relationships is important for both relational and DBTG schemas. The distinction between single-valued relationships and other relationships is important in the DBTG data model, where the set construct provides a natural support for the single-value property.

We shall assume that neither an association nor a single-valued relationship has a property defined on it. A property of an association is necessarily a property of the domain entity set of the association so that the concept is superflous. To provide automatic integrity support for a single-valued relationship, we shall represent such a relationship by a single DBTG set, and doing so precludes the relationship from having a property.

The design model that we have outlined distinguishes among the following semantic objects:

(a) entity set
(b) properties of entity sets
(c) associations
(d) single-valued binary relationships
(e) relationships
(f) properties of relationships

An example of a design schema is the following:

Example 2.1

entity set	properties
emp	ename,birthyr
dept	dname,location
job	title,salary

<u>associations</u>
works-in(emp,dept)
assignment(emp,job)

<u>relationship</u>	<u>status</u>	<u>properties</u>
mgr(dept,emp)	single-valued	---
qualified(emp,job)	general	---
allocation(dept,job)	general	number

Our task now is to state the design goals and to specify the mapping rules by which these goals can be realized.

3. Operational Goals for Design

Our goals are related to database redesign and updates. We define a database <u>redesign</u> operation as the addition or removal of one of the following semantic object types:

(a) entity set
(b) property
(c) association
(d) relationship

The removal of an entity set causes the automatic removal of all its properties and associations, and all the relationships in which it participates.

<u>Design Goal #1</u> A redesign shall have no impact on programs that reference only those data objects which survive the redesign.

For example, if the relationship "qualified" is eliminated, a program to find all the employees assigned to a given job would be unaffected.

We define an _update operation_ as one of the following:

(a) inserting or deleting an element of an entity set
(b) adding or removing an instance of a relationship
(c) changing the value of a property or an association of a single entity

When an entity e is deleted, any entity having e as its value in an association must also be deleted to preserve the integrity of the association. The induced deletions are considered additional update operations, and not as a part of the deletion of e. Similarly, the removal of any instances of relationships in which e participates is also considered to be additional updates. The remaining design goals pertain to update operations.

Design Goal #2 An update of type (b) cannot spawn additional update operations.

In particular, no entity can be caused to disappear as the result of the removal of an instance of a relationship, and the insertion of an entity does not require that it participate in any relationship.

Design Goal #3 A single update affects a single tuple in the relational case and a single record occurrence for DBTG.

The change in the value of one function (property or association) of one entity should reflect in the change of a single tuple for the relational data model and in a single record occurrence for the DBTG data model.

The insertion/deletion of an entity should correspond to the insertion/deletion of a single tuple in the relational data model and a single record occurrence in the DBTG data model.

The addition/removal of a single instance of a relationship should correspond to the insertion/deletion of a single tuple in the relational data model, and for the DBTG data model it should correspond to the insertion/deletion of not more than one record occurrence.

The goal of minimality is not merely one of minimizing storage. Indeed, that is not the primary issue. Instead, the main issues are: (a) ease of maintaining consistency through a one-fact-one-place rule, and (b) an economy of expression in the update programs.

Our next design goal is achievable only for the DBTG data model, and it pertains to automatic propagation of entity deletions. When an entity e is deleted, any entity having e as its value in an association must also be deleted. For example, in the design schema of example 2.1, if a job with title "accountant" is deleted, then all employees having job assignment as accountants are also deleted. This is a necessary consequence of "assignment" being an association. Similarly, when e is deleted any instances of relationships in which e participates must be removed. The structures of a DBTG schema can be used to support automatic propagation of these induced deletions.

Design Goal #4 For a DBTG schema the command to delete one entity suffices to effect all induced updates.

For the DBTG data model the deletion of an entity should cause all induced deletions and removals to be effected without explicit instructions to do so in the applications program.

Our final design goal is also restricted to the DBTG data model.

Design Goal #5 The integrity of being single-valued of any relationship is to be automatically preserved on updates for a DBTG schema.

Here, we want to take maximum advantage of the one-to-many nature of DBTG sets to obviate explicit integrity support for single-valued relationships.

4. Mapping Rule for a Relational Schema

We define an identifier as a one-to-one property of an entity set, designated to represent the entities. As such, the value of the identifier for a given entity cannot be changed[HALL76]. A primary function is a property or an association specified in the design schema. A primitive object is either a relationship, or an entity set in its role as the domain of a primary function. We propose the following rules for mapping a design schema into a relational schema.

(4.1) Each entity set has an explicit identifier which represents it globally in the relational model.

(4.2) The identifier(s) of a primitive object together with all the primary functions of the primitive object are grouped in the same relation of the relational schema.

(4.3) There is one and only one primitive object per relation of the relational schema.

Comments:

(1) it is clear that an application of (4.1)-(4.3) yields a relational schema consisting of one relation for each entity set, and one relation for each relationship. The domains of a relation representing an entity set consist of: its identifier, the identifiers of its associations, and its properties. a relation representing a relationship has as its domains; the identifiers of the participating entity sets, and any properties of the relationship.

(2) Often, an entity can be identified by a combination of associations and/or properties. For example, (dname,mgr) may well identify dept uniquely. We shall assume that even in such cases, an explicit identifier is assigned. Arguments in favor of an explicit identifier are many and to us persuasive: (a) Values of properties and associations can change, and such changes can propagate if they are used in identifiers, thus violating the minimal update design

objective. (b) An identifier is an incarnation of an entity and as such should only be inserted and deleted, not changed. (c) Finally, it usually takes more than one association and/or property to uniquely identify an entity set. For global representation, such a combination is too verbose.

(3) The concept of an "identifier" is not the same as that of a "key" in the framework of "normalization." A key is defined in terms of the attributes of a specific relation, while we have defined an identifier in terms of an entity set and nothing else. An identifier is more than a one-to-one function, its role is to stand for the corresponding entity.

(4) Mapping rule (4.2) is clearly designed to minimize updates. By choosing to represent an association as a relationship, a designer can circumvent the automatic deletion property of an association, but at the price of sacrificing some economy in expression for updates.

(5) We believe that mapping rule (4.3) by itself captures the essence of normal forms. A violation of any one of the normal forms can be interpreted as a violation of rule (4.3).

Example 4.1

Consider the design schema given in example 2.1. Let us introduce the following identifiers for the entity sets: eno for emp, <u>dno</u> for dept, and <u>jid</u> for job. The primitive objects for this example together with their functions are given as follows:

<u>primitive objects</u>	<u>functions</u>
emp	ename,birthyr,works-in,assignment
dept	dname,location
job	title,salary
mgr	---
allocation	number
qualified	---

These are mapped into five relations according to the mapping rules as follows:

```
EMP(eno,ename,birthyr,assignment,works-in)
DEPT(dno,dname,location,mgr)
JOB(jid,title,salary)
ALLOC(dno,jid,number)
QUAL(jid,eno)
```

5. Normal Forms

A violation of one of the normal forms can always be interpreted as a violation of mapping rule (4.3). Different ways in which (4.3) are violated correspond to different normal forms, and these can be classified as follows:

A. Putting two primitive objects which have no entity set in common in the same relation. This violation of (4.3) results in a relation not in 2NF.

   Example: Cartesian product of JOB and DEPT

B. Putting a function of an entity set and a relationship involving it in the same relation. This too results in a relation not in 2NF.

Example: Equijoin of JOB and QUAL on jid

C. Putting two functions with different entity sets as their domains in the same relation. If this violation is not one of category (A) then it must involve function f_1 and f_2 of the form

$$E_1 \xrightarrow{f_1} E_2 \xrightarrow{f_2} S$$

This results in a relation not in 3NF.

Example: The equijoin EMP(assignment=jid)JOB

D. Putting two relationships with a common entity set together in the same relation. This violation results in a relation not in 4NF.

Consider the equijoin AQ of ALLOC with QUAL. As it stands, AQ is not in 2NF because of the partial dependence of "number" on the key (dno,jid,eno) of AQ. The projection AQ[dno,jid,eno] is in 3NF (and hence also 2NF), but not 4NF. There are two multivalued dependencies: "eno on jid" and "dno on jid" in AQ[dno,jid,eno].

Theorem 5.1 A relational schema resulting from applying the mapping rules (4.1) - (4.3) to a design schema is in 4NF.

proof: By rule (4.3), there is one and only one primitive object per relation. One possibility is that the primitive object is an entity set E serving as the domain of a collection of primary functions. In this case its identifier is a key of the relation, and every attribute, being a representation of a primary function of E, is a full function of the identifier. There can be no multivalued dependency in such a relation.

The other possibility is that the primitive object is a relationship. The identifiers of the entity sets making up the relationship comprise a key of the relation, and every non-key attribute is a function of the key. Suppose that contrary to the assertion of the theorem the relation is not in 4NF, then it is equal to the join of two relations [FAGI77a]. Either the identifiers making up the key are split between these two relations or they are not. If they are split then the relationship must be decomposible, contradicting the assumption that each relationship in the design schema is indecomposable. If the key resides entirely in one of the component relations, then attributes of the other component relation cannot be functions of the key, contradicting (4.2). QED

6. Mapping Rules for a DBTG Schema

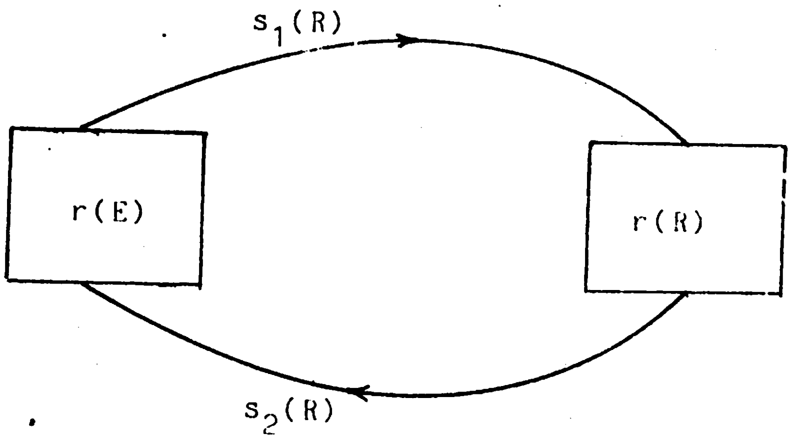
The following mapping rules are introduced to convert a design schema into a DBTG schema so as to achieve our design goals:

(6.1) Each entity set has an explicit identifier.

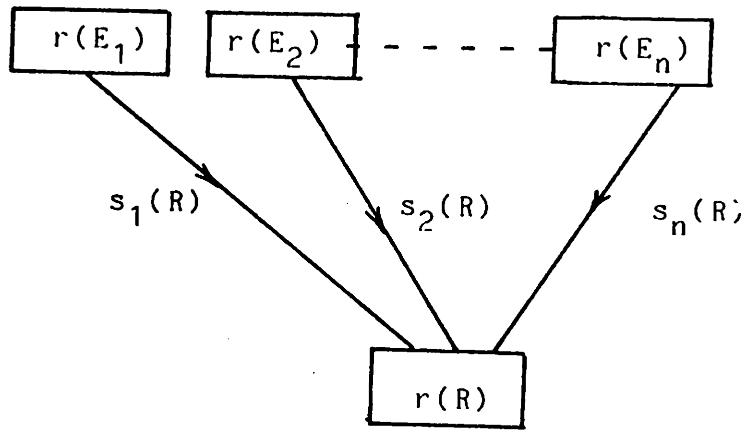
(6.2) For each entity set E define a record type r(E). The data items of r(E) are made up of the identifier of E and the properties of E.

(6.3) For an association or single-valued relationship $R(E_1, E_2)$ where $E_1 \neq E_2$, define a set type s(R) with $r(E_2)$ as the owner record type and $r(E_1)$ as the member record type.

(6.4) For an association or single-valued relationship R(E,E) define a record type r(R) having no data item, and a pair of set types $s_1(R)$ and $s_2(R)$ forming a cycle between r(E) and r(R). The assignment is depicted below:



(6.5) For a general relationship $R(E_1, E_2, \ldots, E_n)$ define a confluent hierarchy, consisting of a record type r(R) with only the properties of R as its data items, and n set types $s_1(R)$, $s_2(R)$, ... , $s_n(R)$ as shown below.



All the object types in the design model have now been mapped into object types in the DBTG data model. Two kinds of record types have resulted from the mapping: ones which contain an identifier data item and those which do not. We shall call the former self-identified record types, and the latter link record types. Self-identified record types represent entity sets while link record types represent relationships and possibly associations.

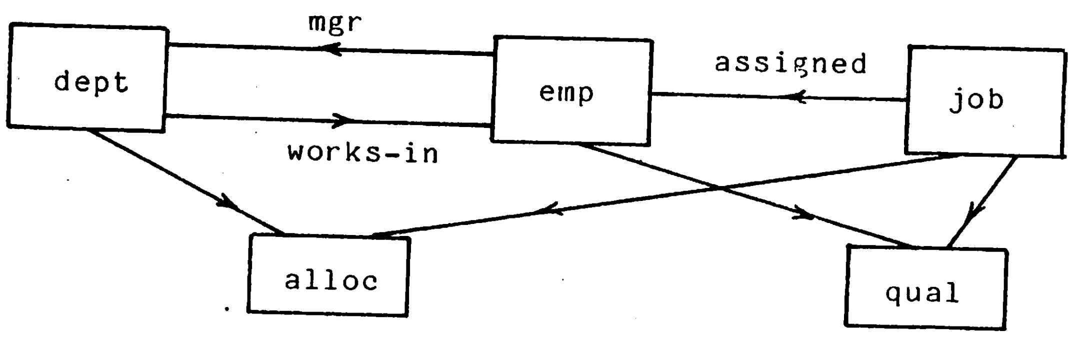
For set types the logical concept of total membership would be useful in our context. A record type r is a total member of a set type s if every occurrence of r is a member of an occurrence of s. A member that is not total is said to be partial. The membership of a link record type in any set type should always be total. The membership of a self-identified record type should be total in any set which represents an association but not otherwise. The concept of "total" membership in sets does not exist in the current version of the DBTG model although a related concept appears to have been suggested by [NIJI75].

Natural enforcement for "total" membership is provided by the mandatory/automatic option for delete/insert in DBTG, except when a sequence of set types form a cycle. In that case one set type in the cycle is required to be "manual" on insertion. Under our mapping rules a cycle of set types each having a total member can arise in only two ways: (i) a two-set cycle representing a self-association, (ii) a cycle of associations. In the first case we shall make the set type $s_2(R)$ mandatory/manual. In the latter case we choose any one of the set types to be mandatory/manual.

Summarizing our discussion on set membership, we have the final mapping rule for DBTG:

(6.6) The membership of a link record type in any set type is total. The membership of a self-identified record type in any set that represents an association (or is a part of the representation of a self-association) is total, but not otherwise. All set membership that are not total are optional/manual. All total memberships are mandatory/automatic, except for a self-association or a cycle of associations. For the exceptions one of the set types in the cycle must have a mandatory/manual membership option, and the property of being "total" must then be supported procedurally.

Application of rules (6.1)-(6.6) to example (2.1) yields the following DBTG schema:



Except for the partial membership of "dept" in "mgr," all memberships are total.

7. Schema Translation and Equivalence

In section 6 we introduced a distinction between self-identified records and link records, and between total and partial memberships in a set. These distinctions can be deduced from existing constructs of the DDL, but it may well be desirable to make them explicit in the DBTG schema. With this bit of semantic enhancement in the DBTG data model, the process of mapping a design schema into a DBTG schema using rules (6.1) - (6.6) becomes reversible, as is shown in the following correspondences:

self-identified record type	-->	entity set
data items except id	-->	properties
set type between two self-identified record types	-->	association or single-valued relationship, according to whether membership is total or partial
link record type (member of two or more set types)	-->	relationship
a two-set cycle between a self-identified record type and a link	-->	association or single-valued relationship between an entity set and itself, differentiated

by set membership

It follows that a DBTG schema derived from a design schema by following our mapping rules can be translated into a relational schema which is the same as what would be obtained by directly mapping the design schema. The rules of translation are rather simple and given below.

For a self-identified record define its key to be its identifier. For a link record define its key to be the collection of the keys of the owners of all sets in which the link record is a member. For a DBTG schema obtained from using our mapping rules link records can only be owned by self-identified records, so that the definition of key is not circular. We propose the following rules for DBTG to relational schema translation.

(7.1) For each self-identified record type r define a relation R(r). Each data item of r is a domain of R(r). The key of the owner of every set in which r is a total member is also a domain of R(r).

(7.2) For each link record type k that is the member of two or more set types, define a relation R(k). The domains of R(k) consist of the data items of k plus the keys of the owners of the sets in which k is a member.

(7.3) For each set type s which has a partial member, define a binary relation R(s). The domains of R(s) are the keys of the owner and the member of s.

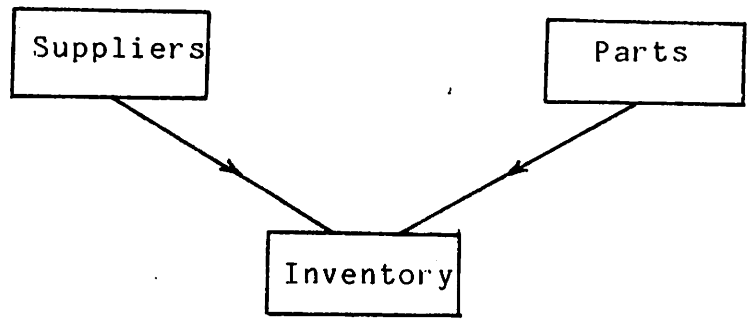
In terms of the constructs of the design model, (7.1) identifies an entity set, (7.2) a relationship, and (7.3) a single-valued relationship.

We have identified a collection of DBTG schemas that can be mapped into corresponding relational schemas while preserving the design goals. The translation procedure can be applied in general, but in doing so, we impose a semantic interpretation on the schema to be translated. The success of the translation depends on the extent to which the interpretation is correct.

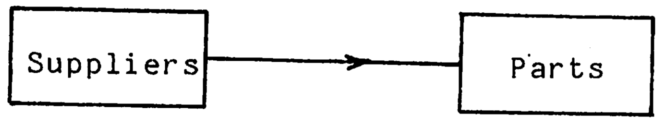
8. Discussion

The focus of the design methodology that we have described is on the preservation of basic semantic integrity so as to prevent unwanted side effects in redesign and updates. Issues of performance have not been addressed. For the relational case, issues of performance relate only to the storage schema and are rightly avoided in the design of the conceptual schema. The situation is somewhat more complex in the DBTG case. Constraints on set implementation and the possibility of clustering members cause the logical design to have an effect on performance. We shall try to explain this issue through an example.

Consider a design schema containing entity sets: suppliers and parts, and a relationship inventory (supplies, parts). Following the mapping rules of section 6 results in a DBTG schema depicted as follows:



where "inventory" is a link record type. It can be argued with some cogency that if the only accesses are through suppliers, then the following design would be better for performance:



The latter design can, in fact, be obtained with our procedure by modifying the design schema. Instead of "parts", introduce an entity set parts-of-suppliers, the same part supplied by different suppliers being separate entities. The "inventory" relationship is now single-valued in "parts-of-suppliers", so that an application of mapping rule (6.3) results in precisely the second design. The design goals are still satisfied but now have a different interpretation. For example, changing the property of a part is no longer an atomic update operation, so that the minimality goal, though attained, no longer guarantees that only a single data-item value is changed. In short, one can change the DBTG design by manipulating the design schema, but there are consequences of doing so on redesign and updates and these can be made explicit by a careful examination of the design goals.

The semantics of our design model are still quite sparse. A number of possible additional object types have been deliberately omitted. Our criterion for inclusion is, "does including it make a difference in operational terms?" Examples such as "property of property" and "relationship of relationships" fail on this ground. The answer in the case of "dependent entities" is less straightforward. It can be argued that there are natural examples, such as "children of employee", of entities whose existence in the database depends on that of others. Operationally, such dependence is already provided for in our design model by the construct "association". The entity set "children of employee" would have an association "parent". Deletion of the parent causes deletion of the child in conformity with our rules on automatic deletion. Insertion of a child cannot be made until the parent exists. The only difference that the semantic construct "dependent entities" might make on the design relates to the choice of an identifier. It might be argued that the identifier of a dependent entity should always be a concatenation of the parent-id and a local-id, and there is no provision in our rules for enforcing such a choice. However, it seems to us that the additional benefit is small and does not warrant adding yet another semantic object.

References

[BERN76] Bernstein, P. A. "Synthesizing third normal form relations from functional dependencies." Transactions on Database Systems 1,4 (Dec. 1976), pp. 277-298.

[CHEN76] Chen, P. P. "The entity-relationship model - towards a unified view of data." Transactions on Database Systems 1,1 (Mar. 1976), pp. 9-36.

[CODD71] Codd, E. F. "Further normalization of the data base relational model." Courant Computer Science Symposia 6, Data Base Systems, Prentice-Hall, New York, (May 1971), pp. 65-98.

[FAGI77a] Fagin, R. "Multivalued dependencies and a new normal form for relational databases." Transactions on Database Systems 2,3 (Sep. 1977), pp. 262-278.

[FAGI77b] Fagin, R. "The decomposition versus the synthetic approach to relational database design." Proceedings 1977 Very Large Data Bases Conference, 1977, pp. 441-446.

[HALL76] Hall, P., Owlett, J., and Todd, S., "Relations and entities", Proceedings IFIPTC-2 Working Conference on Modelling in Database Management Systems, North Holland, 1976.

[NIJI75] Nijssen, G. M. "Set and CODASYL set or coset." Data Base Description, B. C. M. Douque, G. M. Nijssen, eds., North-Holland, Amsterdam, 1975, pp. 1-70.

[SCHM75] Schmid, H. A., Swenson, J. R., "On the semantics of the relational data model." Proceedings ACM-Sigmod Conference, (May 1976), pp. 9-36.