Entity-Relationship Approach to
Systems Analysis and Design, P.P. Chen (ed.)
North-Holland Publishing Company, 1980

LOGICAL DESIGN AND SCHEMA CONVERSION FOR RELATIONAL AND DBTG DATABASES

Eugene Wong and Randy H. Katz

Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, California 94720
U.S.A.

The objective of this paper is three-fold: first, a variant of the Entity-Relationship model is proposed as the data model for logical design. In terms of the constructs of the model "update anomalies" are precisely defined. Second, with the prevention of "update anomalies" as the goal for logical design, mapping rules will be proposed for transforming a design schema into a corresponding schema of either relational or DBTG type. Third, it is shown that with relatively minor semantic augmentation of DBTG and relational data models, the mapping is reversible. As a consequence, reversible schema conversion between DBTG and relational models becomes possible.

1. Introduction

The semantic spareness 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], and the problem as given rise to a theory of "normal forms" [BERN76,FAGI77a,FAGI77b]. With a substantial and growing literature, the theory of relational normal forms has acquired a life of its own, and most of the recent results do not directly address problems of update difficulties. While logical design for databases of the CODASYL-DBTG type has received comparatively little attention, problems of update anomalies are no less important there. The first motivation for the work reported in this paper is to provide a unified framework for designing relational and DBTG schemas that are free of update anomalies.

Our work is also motivated by the realization that coexistence of DBTG and relational data models in the same system may well be necessary in many instances. A distributed database system, for example, may result from the integration of existing databases of both types. Consequently, we believe that algorithms for DBTG-Relational schema conversion that are semantics-preserving are needed. We propose that a test for semantics-preservation be reversibility of the conversion process.

Our approach to both of these problems is to introduce as an intermediary a design data model which is a variant of the Entity-Relationship model. Specifically, we shall do three things: define "update anomalies" in terms of the constructs of the design model; present mapping rules for producing anomaly-free schemas of both DBTG and relational types, and by semantic augmentation make these mappings reversible and consequently render a reversible schema conversion between DBTG and relational data models possible.

Research supported by the U.S. Army Research Office Grant DAAG-76-6-0245, the U.S. Air Force Office of Scientific Research Grant 78-3596, the Honeywell Corporation, and an I.B.M. Fellowship for the second author.

2. A Design Model

We shall use the term "data model" in a generic sense to mean a collection of data object types, and a "schema" to mean a specific choice of data objects to represent a database. For example, the relational data model consists of the data object types domains and relations, while a relational schema would comprise a specific collection of relations each with specified domains.

Our point of departure is to choose a basic semantic data model to be used to specify a design schema. The model as such is not new, being a simplified version of both the Entity-Relationship model [CHEN76] and the semantic data model introduced in [SCHM75].

For each instant 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 one-parameter 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 function 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 $E_1(t)$ X $E_2(t)$ X ... X $E_n(t)$. 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 single-valued 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 association. At each t an association $R_t(E_1(t), E_2(t))$ is a function 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, then mgr(dept,emp) is an association. If a dept can be temporarily without a manager, then mgr(dept,emp) is a binary relationship, 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, while the "set" construct provides a natural support for "single-valued" relationships.

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 superfluous. A single-valued relationship cannot enjoy automatic integrity support

through the set construct if it has a property defined on it.

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

associations
works-in(emp,dept)
assignment(emp,job)

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

3. Design for Anomaly-Free Updates

Roughly speaking, "update anomalies" take the form of either "fragmentation of an atomic operation" or "uncontrolled side effects". A data model used for logical design must make these issues clear. We think that the design model we have introduced is particularly well suited for this purpose.

We shall define an atomic update operation as one of the following:

(1) inserting or deleting an entity

(2) inserting or deleting an instance in a relationship

(3) changing the value of a function (property or association) of an entity.

An atomic update operation may cause inconsistencies in the database. For example, suppose the Job-entity "Engineer" is deleted, what happens to employees who are engineers? For consistency to be maintained, one or more additional atomic updates may have to be undertaken. These induced updates will be referred to as side-effects.

Observe that only updates of type (1) have side-effects, and these can be described as follows:

(1) Deleting an entity e causes

(a) the deletion of any instance of a relationship in which e participates, and

(b) the deletion of any entity that has e as its range value in an association.

(2) Insertion of an entity e requires any entity that is the value of any association of e to already exist.

We note that side-effects of the type (1b) propagate, so that deleting one entity may spawn a large number of additional entity deletions. The type "association" exist precisely to allow automatic propagation of deletions. Whether one designates a single-valued binary relationship as an association or not is a design choice used to control side-effects. The concept of "dependent entities" can be modelled by an association in this way.

We also note that the insertion side-effect (2) means that the order of insertion of entities may be constrained by associations, and no cycle of associations can exist.

Our approach to logical design can now be stated. First, one specifies a schema (design schema) in terms of the design data model. In so doing, one completely determines the atomicity and side-effects of all update operations. To design schemas of the relational and DBTG types, one needs mapping rules that would transform the design schema into one of the appropriate type, while preserving both atomicity and side-effects of updates. What we mean by this is expressed by the following goals for the mapping rules:

Preservation of Atomicity

One atomic update affects a single tuple in the corresponding relational schema and a single record (plus possibly some set membership changes) in the DBTG schema.

Preservation of Side-Effects

The additional atomic operations induced are exactly the same as in the design model case.

The DBTG data model provides the possibility of automatic integrity support for the constructs "association" and "single-valued relationship". We wish to take advantage of this possibility. Hence, the mapping rules for the DBTG case should provide:

Automatic Integrity Support (for "association" and single-valued relationships)

This support will be achieved through the feature of set-membership-option on insert/delete provided in the DBTG definition facility.

4. Mapping Rule for a Relational Schema

We define an identifier as a one-to-one property of an entity set, designated to globally represent the entities. As such, an identifier is the same as a surrogate proposed in [CODD79], and the value of the identifier for a given entity cannot be changed. 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) 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, usually it takes more than one association and/or property to uniquely identify an entity set. For global representation, such a combination is too verbose.

(2) 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.

(3) 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.

(4) 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, dno for dept, and jid for job. The primitive objects for this example together with their functions are given as follows:

primitive objects	functions
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,edept)
DEPT(dno,location,mgr)
JOB(jid,title,salary)
ALLOC(dno,jid,number)
QUAL(jid,eno)

Some renaming of associations has obviously taken place.

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 violations can be classified as follows:

A. Putting two primitive objects together with 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 example 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, and 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 also 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 relation cannot be functions of the key, contradicting (4.2). QED

Theorem 5.1 does not assert that every 4NF relational schema must come from a design schema via our mapping rules. Indeed, there is not enough semantics in the 4NF relational model to make such an assertion meaningful.

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:

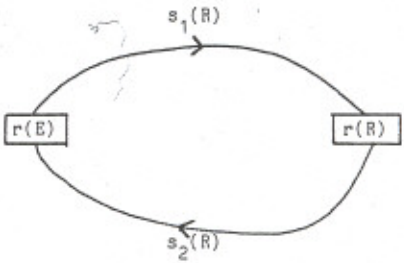
(6.1) Each entity set has an explicit identifier.

(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.

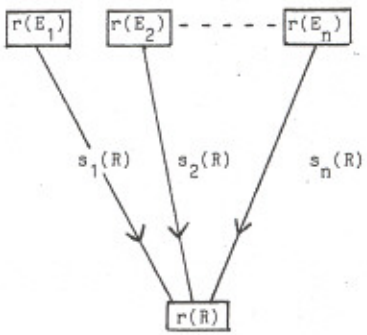
(6.3) For an association or single-valued relationship $R(E_1, E_2)$ where $E_1 \neq E_2$, define a set type s(R) with $r(E_2)$ as the owner record type and $r(E_1)$ as the member record type.

(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 is a total member of a set type if every record occurrence is a member of a set occurrence. A member that is not total is said to be partial. The membership of a link record type in any set type should be total. The membership of a self-identified record type is 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 related concepts appear to have been suggested by [NIJI75].

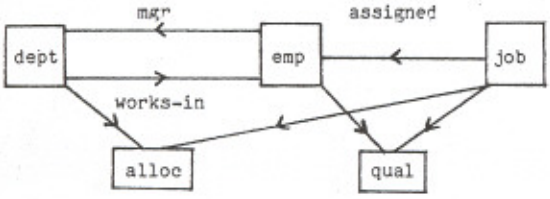
Natural enforcement for "total" membership is provided by the mandatory/automatic option for delete/insert, except when a sequence of set types form a cycle. In that case one set type in the cycle is required to be "manual" on insertion. Under our mapping rules a cycle of set types each having a total 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:



7. Schema Translation and Equivalence

The mapping rules indicate which objects of the target model schema have been derived from design model constructs. If the schema is augmented with additional semantic information that distinguishes among similar objects derived from different design model objects, the mapping rules of sections 4 and 6 can be used to formulate inverse mappings.

For example, in section 6 we introduced a distinction between self-identified records and link records, and between total and partial memberships in a set in a DBTG schema. Although these distinctions can be deduced from existing constructs of the DDL, it is desirable to make them explicit in the DBTG schema. Similarly, the relational schema can be augmented with identification of key and foreign key domains (ID domains).

The equivalent relational and DBTG constructs for each design model con-

struct are shown below:

Design	DBTG	Relational
Entity Set	Record Type(SI)	Relation(E)
Identifier	Data Item(ID)	Key Domain(ID)
Property	Data Item	Domain(value)
Association	Set Type(Total)	Domain(ID)
S.V. Relationship	Set Type(Partial)	Relation(S)
E_1 (S.V.)	Member	Key Domain(ID)
E_2	Owner	Domain(ID)
Relationship	Record Type(L)	Relation(R)
Property	Data Item + Set Types(Total)	Domain(value)
$E_1, E_2, \ldots, E_n$	Owners	Domains(ID)

We note that there is a certain difficulty in distinguishing between a single-valued relationship and a general binary relationship within a derived relational schema without distinguishing between the types of relations. This is why we have introduced E,S, and R relations.

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, and vice versa. The rules of the transformation 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 l that is the member of two or more set types, define a relation R(l). The domains of R(l) consist of the data items of l plus the keys of the owners of the sets in which l 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.

Similarly, we can formulate the rules for mapping a relational schema directly into a DBTG schema. Identifier domains are drawn from the key values of the associated E-relation. We propose the following rules for relational to DBTG schema translation:

(7.4) For each E-relation R define a self-identified record type r whose identifier is the key domain of R. Each value domain of R is a data item of r. The representation of the E-relations associated with the identifier domains of R are owners of sets for which r is a total member.

(7.5) For each R-relation R define a link record type l. Each value domain of R is a data item of l. The representation of the E-relations associated with the identifier domains of R are owners of sets for which l is a total member.

(7.6) For each S-relation R, the representation of the E-relation associated with the key domain of R is the partial member of a set owned by the representation of the E-relation associated with the identifier domain of R.

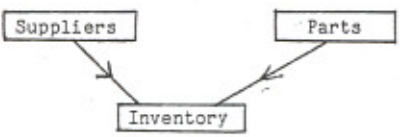
(7.4) identifies an entity set, (7.5) a relationship, and (7.6) a single-valued relationship.

We have identified a class of DBTG schemas that can be mapped into corresponding relational schemas, and vice versa, while preserving the design goals. The translation procedure is based on a semantic interpretation derived from the design model. The translation procedure can be applied in general, but the quality of the translation depends on the extent to which the interpretation is correct.

8. Conclusion

The focus of this paper is semantics of data models, but semantics with a purpose. The purpose is two-fold: logical design to avoid update anomalies, and semantic comparison between data models to achieve schema translation that is reversible. We believe that in the process the semantics of both relational and DBTG models are elucidated. 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. There are constraints as to how sets are implemented, so that logical design bears an impact 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-supplies", 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, because a part is no longer an entity. Hence, atomicity, though preserved, no longer guarantees that only a single data-item value be changed. In short, one can change the DBTG design by manipulating the design schema, but there are

consequences of doing so and these are explicated by a careful examination of the design goals.

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.

[CODD79] Codd, E. F., "Extending the Database Relational Model to Capture More Meaning." ACM SIGMOD Conf. Supplement, to appear in ACM Transactions on Database Systems, (1979).

[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.

[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.