Pergamon

Information Systems Vol. 19, No. 4, pp. 331–358, 1994
0306-4379(94)00016-6

0306-4379/94 $7.00 + 0.00

AN ANALYSIS OF SQL INTEGRITY CONSTRAINTS FROM AN ENTITY-RELATIONSHIP MODEL PERSPECTIVE†

A.H.F. LAENDER[1], M.A. CASANOVA[2], A.P. DE CARVALHO[2] and L.F.G.G.M. RIDOLFI[2]

[1]Departamento de Ciência da Computação , Universidade Federal de Minas Gerais
Caixa Postal 702 , 30161-970 Belo Horizonte MG, Brasil

[2]Centro Científico Rio , IBM Brasil
Caixa Postal 4624 , 20071-001 Rio de Janeiro RJ, Brasil

(Received 14 October 1993; in final revised form 25 April 1994)

Abstract — This paper presents an analysis of the integrity constraints defined in the SQL ISO standard disclosure in the light of the entity-relationship model. It points out what features of integrity constraints in SQL support which features of the entity-relationship model by discussing how to map an entity-relationship schema into a SQL schema. In order to organize the analysis, the paper distinguishes three levels of the entity-relationship model. The first level corresponds to the basic model, augmented with simple specializations. The second level considers totality and more complex specializations, and allows deferred and immediate propagation of deletions. Finally, the third level introduces generalization hierarchies and considers some types of inter-relationship constraints. For the first and second levels, the analysis indicates that the more complex features of the SQL referential integrity construct are necessary only when optimization is considered, but one of the features - propagation of nulls - should be slightly enhanced to easy the optimization task. However, for the third level, the analysis shows that, in most cases, the SQL referential integrity construct can efficiently model subsetting, but that the direct use of SQL integrity constraints to account for mutual exclusion may be very expensive.

1. INTRODUCTION

We analyze in this paper the integrity constraints included in the SQL language in the light of the entity-relationship (ER) model [10]. We base our analysis on the standard disclosure published by ISO in April 1991 [15], but most of the discussion also applies to the ANSI/ISO working draft for SQL2 [1] that reflects the ANSI X3H2 meeting held in Portland, ME on June 1990 and the ISO SQL2 meeting in Seoul on May 1990 (see [9]).

The SQL standard disclosure describes three classes of integrity constraints that cover primary and alternate keys, referential integrity and restrictions in general for the row values. Referential integrity is especially flexible and offers three varieties of triggered actions. Integrity constraints may also be checked immediately, after the execution of each SQL statement, or deferredly, roughly at the end of the transaction. The standard disclosure also distinguishes two subsets of the language, called Entry SQL and Intermediate SQL, of increasing complexity.

The first contribution of this paper lies in pointing out what features of integrity constraints in SQL support which features of the ER model by discussing how to map an ER schema into a SQL schema. The second contribution is to suggest a minor extension to SQL that greatly extends the ability to obtain optimized SQL representations of ER schemas.

In order to organize the analysis, we distinguish three levels of the ER model and make explicit some key restrictions of each level. The levels permit the design of applications of increasing sophistication, measured in terms of the complexity of the specialization/generalization hierarchies and the properties of the relationship sets.

The first level corresponds to the basic ER model, augmented with simple specializations. We argue that any schema that falls within the limits of this model can be mapped into Entry SQL, even without support for many of the enhancements introduced, like references to alternate keys and check constraint definitions, provided that updates to keys or identifiers are not allowed.

The second level considers totality and more complex specializations [22], and allows deferred and immediate propagation of deletions. In this case, we show that Intermediate SQL suffices for

†Recommended by N. Bidoit.

most cases, even without support for alternate keys with nulls, match type or update rule. We have to resort to full SQL if we allow updates to keys or identifiers, or the ER schema has a relationship which is total on a participant, but not functional on it. The handling of specializations that are not hierarchies also unveils several interesting features of SQL. The most interesting analysis is associated with the problem of optimizing the mapping of ER schemas into SQL by combining the representation of more than one ER scheme into a single table. We point out that the more complex features of the referential integrity mechanism introduced in SQL are necessary only when optimization is considered, but one of the features - propagation of nulls - should be slightly enhanced to easy the optimization task.

The third level introduces generalization hierarchies [4, 12, 21, 22] and permits the definition of some types of inter-relationship constraints [13, 14, 16, 18, 23]. The mapping of this kind of ER schema into the relational model requires mechanisms to capture subsetting and mutual exclusion. Our analysis shows that, in most cases, subsetting can be efficiently handled by referential integrity constraints, but that mutual exclusion may be very expensive to be handled directly by the SQL integrity constraint mechanisms.

The analysis described in this paper is broadly based on the experience gained during the design and prototyping of a database design tool, called FeST [20]. The description of the ER model is inspired on the data definition language introduced in [6]. The paper also specializes to SQL the optimization of relational representations of ER schemas outlined in [5, 7, 8].

Finally, it should be pointed out that the use of variations of the ER model for database conceptual design has been extensively investigated (see, e.g., [22] for a survey), as well as the mapping of extended ER schemas to the relational model [2, 3, 4, 12, 17, 19, 21]. The difference between most of these mappings lies in the way the structures of the ER model are translated into structures of the relational model based on the scope of the extensions adopted. Broadly speaking, this paper conveys similar ideas, but the focus is on the role of various forms of integrity constraints described in the standard of a concrete relational database language.

This paper is organized as follows. Section 2 summarizes the relevant aspects of the integrity constraint features of SQL. Sections 3, 4 and 5 describe, respectively, the three levels of the ER model we consider and discuss their mappings to SQL. Finally, Section 6 contains the conclusions.

2. SUMMARY OF THE RELEVANT CHARACTERISTICS OF SQL

We summarize in this section the relevant characteristics of three types of SQL constraint definitions, called *unique constraint definitions*, *referential constraint definitions* and *check constraint definitions*. For the sake of precision, we will repeatedly refer to specific sections of the SQL standard disclosure [15] in what follows. We also outline the restrictions that apply to Intermediate SQL and Entry SQL.

We first recall that the SQL standard disclosure [15, sec. 4.10] defines that the checking of an integrity constraint depends on its constraint mode within the current transaction. If the mode is *immediate*, then the constraint is checked at the end of each SQL statement. If the mode is *deferred*, the checking is deferred until the end of the transaction or until explicitly enacted by the execution of a set constraints mode statement. The default is immediate. When an integrity constraint is checked, if it is not satisfied, then an exception condition is raised and the SQL statement that caused the constraint to be checked has no other effect. For Entry SQL and Intermediate SQL, all constraints are always immediately checked [15, sec. 11.6, p.199].

The database designer may define at most one *primary key* and several *alternate keys* for a table by including unique constraint definitions in the table definition [15, sec. 11.7]. Primary keys are defined with the help of the keyword PRIMARY KEY and an alternate key is defined using the keyword UNIQUE. All keys of a table must be distinct, but they need not be minimal. NOT NULL is implicit for the column names that form the primary key, but the definition of the column names of an alternate key may admit null values. That is, an alternate key is satisfied iff no two rows in the table have the same non-null values for the key [15, p.29]. For Entry SQL, the definition of each column whose name occurs in a primary or alternate key must specify NOT NULL [15, sec. 11.7, p.201].

Each referential constraint definition [15, sec. 11.8] included in the definition of a table T specifies a *foreign key* K of T and relates it to a key L of a table U; the key L is called the *referenced key* and T and U are called the *referencing table* and the *referenced table*, respectively. The referenced key may either be the primary key or one of the alternate keys of U. If L is omitted, then it is taken by default to be the primary key of U.

Each referential constraint definition may also include a *match type* that influences the semantics of the constraint. Let T, K, U and L be as above. There are three cases to consider [15, sec. 8.10 and 11.8, p.204]:

Case 1 the match type is omitted. Then, the referential constraint is *satisfied* iff, for each row t of T, either some value in t[K] is the null value or no value in t[K] is the null value and there is a row u of U such that t[K]=u[L].

Case 2 the match type is PARTIAL. Then, the referential constraint is *satisfied* iff, for each row t of T, either all values in t[K] are the null value or some value in t[K] is not the null value and there is a row u of U such that t[K]=u[L].

Case 3 the match type is FULL. Then, the referential constraint is *satisfied* iff, for each row t of T, either all values in t[K] are the null value or no value in t[K] is the null value and there is a row u of U such that t[K]=u[L].

Intermediate SQL and Entry SQL do not support the concept of match type FULL.

Additionally, each referential constraint definition may also include a *delete rule* and an *update rule* that determine how the system should maintain the constraint. If specified, both rules may be CASCADE, SET NULL or SET DEFAULT and affect the semantics of the transactions as follows.

Let T, K, U and L be as before. e say that a row t of T *matches* a row u of U (with respect to the referential constraint in question) iff t[K]=u[L].

The possible cases of the deletion rule are (see [15, sec. 11.8, p.205] for the details):

1. if the delete rule is omitted, then the deletion of a row u from U is rejected if there is a row t of T that matches u;
2. if the delete rule specifies CASCADE then the deletion of a row u from U propagates to the deletion of all rows t of T that match u;
3. if the delete rule specifies SET NULL then the deletion of a row u from U causes all columns of K of all rows t of T that match u to be set to the null value;
4. if the delete rule specifies SET DEFAULT then the deletion of a row u from U causes each column of K of each row t of T that matches u to be set to its default value.

Let **u** be an update of the value of a column N in L of a row u from U. Let M be the column of K corresponding to N in L. Then, **u** causes the following action (see [15, sec. 11.8, p.206] for the details):

1. if the update rule is omitted, **u** is rejected if there is a row t of T that matches u;
2. if the update rule specifies CASCADE, all rows t of T that match u are updated in such a way that t[M] remains equal to u[N];
3. if the update rule specifies SET NULL:
   (a) in all rows t of T that match u, t[M] is set to the null value, if the match type is omitted;
   (b) in all rows t of T that match u, t[A] is set to the null value, for all columns A in K, if the match type is FULL;
4. if the update rule specifies SET DEFAULT, in all rows t of T that match u, t[M] is set to the default value associated with M or the domain of M.

Intermediate SQL does not support update rules and Entry SQL supports neither delete nor update rules [15, sec. 11.8, p.207]. This means that, in Entry SQL, any deletion or update that would cause an immediate violation of a referential constraint is rejected. The delete and update rules of distinct referential constraint definitions must be *consistent*, as defined in detail in [15, sec. 11.8] and omitted in this summary for brevity.

Finally, the schema definition may include a *check constraint definition* C at three different points:

- at a column definition, when C specifies conditions that the values of that column must satisfy [15, sec. 11.4];
- at a table definition, when C specifies conditions that the rows of the table must satisfy [15, sec. 11.9];
- as a separate *assertion definition*, when C specifies conditions that the tables must satisfy [15, sec. 11.34].

For example, in the table definition

```
create table T
    (A char(1) check (A='Y' or A='N'),
     B char(1) check (B match (select B from U where C='M')))
```

the check constraint definition associated with column A specifies that the domain of A is actually {Y,N}, while the check constraint definition associated with column B dictates that every row of T must have the value of column B equal to the value of column B of some tuple in U that has the value of column C equal to M.

Now, in the table definition

```
create table T
    (A char(1),
     B char(1),
     check ((A='Y' and B='N') or (A='N' and B='Y')))
```

the check constraint definition associated with the table itself imposes that every tuple of T must have certain combinations of values for columns A and B.

Restr.	*Feature*	*Page*	*Entry SQL*	*Intermediate SQL*
R1	constraint checking	199	always immediate	always immediate
R2	unique constraint	201	not null for both prim./alt. keys	not null only for primary keys
R3	referential constraint	207	MATCH not supported	MATCH not supported
R4	referential constraint	207	triggered action not supported	update rule not supported
R5	check constraint	209	subqueries not allowed	subqueries not allowed
R6	assertions	245	not supported	not supported

Table 1. Summary of Entry and Intermediate SQL restrictions

As our final example here, the assertion definition

```
create assertion AS
    check (not exists
            select K from T
                where K in (select K from U))
```

defines that the K column of T and U must have disjoint values.

For Intermediate and Entry SQL, assertion definitions are not allowed [15, sec. 11.34, p.245]. Also, for Intermediate and Entry SQL, the condition in a check constraint definition attached to a table or to a column must be a simple restriction [15, sec. 11.9, p.209, and sec. 11.9, p.192] and cannot involve subqueries to the same table or to other tables. In general, all restrictions on table constraint definitions also apply to column constraint definitions.

Table 1 summarizes the restrictions imposed on Entry and Intermediate SQL that directly influence the discussion in subsequent sections (the page numbers indicate where to find the restrictions in [15]).

The major consequences of each of these restrictions are:

R1. Each SQL statement must preserve all table constraints, which means that no inconsistent states occur during the processing of a transaction.

R2. Alternate keys may have null values only for Intermediate SQL.

R3. A foreign key that has a null value matches any primary key.

R4. Deletions and updates that violate a referential constraint have no effect other than raising a condition, that is, they *block.*

R5. A check constraint for a table T can only contain simple conditions that involve only the attributes of T. Therefore, a check constraint can be tested by observing one-by-one the tuples in T.

3. MAPPING OF LEVEL 0 ENTITY-RELATIONSHIP SCHEMAS

3.1. Level 0 entity-relationship schemas

A *level 0 entity-relationship schema* (or *level 0 schema*) is a list of entity schemes, relationship schemes and simple specialization declarations with the following characteristics.

An *entity scheme* contains a name E, a set of attributes with their domains and optionally a primary key and a list of alternate keys. The attributes belonging to any key must not allow null values.

The database designer may also indicate that an entity scheme E is a *specialization* of at most one other entity scheme F. In this case, E *inherits* all attributes and keys of F. The primary key is required for E only when E is not a specialization of any other scheme.

For each level 0 schema $\mathbf{A}$, we may define a *specialization graph* $\mathbf{G_A} = (\mathbf{N}, \mathbf{E})$) such that $\mathbf{N}$ is the set of entity schemes of $\mathbf{A}$ and (E,F)$\in\mathbf{E}$ iff E specializes F in $\mathbf{A}$. The *sinks* of $\mathbf{G_A}$ are then the entity schemes that do not specialize any other scheme. Since in level 0 schemas an entity scheme may specialize at most one other scheme, $\mathbf{G_A}$ is actually a forest, that is, a collection of trees.

Likewise, a *relationship scheme* has a name R and a set of attributes with their domains, and indicates a list $P_1, ..., P_m$ of *participants*, with $m \geq 2$, which are names of entity schemes or other relationship schemes. We say that R is *over* $P_1, ..., P_m$. However, there cannot be a chain of relationship schemes $R_0, ..., R_n$ such that R_i participates in R_{i+1} and R_n participates in R_0, as otherwise the semantics of $R_0, ..., R_n$ would be ill-defined. The same entity or relationship scheme may occur more than once in the list $P_1, ..., P_m$, that is, $P_i = P_j$, for some i and j in [1,m]. When this is the case, the participants should be distinguished by their position in the list. For example, one may define a relationship scheme REPORTS-TO over the list EMP,EMP, in which case the entity scheme EMP is both the first and the second participant.

Optionally, the relationship scheme may also define one or more *identifiers*, which are lists of participants that are used to capture the cardinalities of the relationship set. For example, a relationship scheme WORK over EMP and DEPT, with EMP as identifier, defines a binary **n:1** relationship set, with EMP "on the n side". But identifiers are much more flexible and permit capturing complex situations, such as that described in [22].

A *database state* σ for a level 0 schema **A** is a function that maps each entity scheme E of **A** into an *entity set* $\sigma(\mathsf{E})$ and each relationship scheme R over $\mathsf{P}_1, ..., \mathsf{P}_m$ into a relationship set $\sigma(\mathsf{R}) \subseteq \sigma(\mathsf{P}_1) \times ... \times \sigma(\mathsf{P}_m)$. The state σ is *consistent* [6] iff it respects all keys defined for entity schemes, all identifiers defined for relationship schemes, and $\sigma(\mathsf{E}) \subseteq \sigma(\mathsf{F})$, for each pair of entity schemes E and F such that E specializes F. Given a relationship scheme R over $\mathsf{P}_1, ..., \mathsf{P}_m$ and a state σ, we will use $\Pi_{\mathsf{P}_i}(\sigma(\mathsf{R}))$ to denote the i-th projection of $\sigma(\mathsf{R})$.

By convention, we will use capital letters for the names of the entity or relationship schemes and, when the state is immaterial, the same letter in italics for the associated entity or relationship set.

For level 0 schemas, if an entity scheme E specializes F, the insertion of an entity in E will be blocked if it does not exist in F, and the deletion of an entity from F will be blocked if it exists in E. Likewise, if R has a participant P, the insertion of a relationship r in R will be blocked if the projection of r on P does not exist in P, and the deletion of an instance p from P will be blocked if there is a relationship r in R such that the projection of r on P is p.

3.2. Mapping into SQL

We argue that the mapping of level 0 entity-relationship schemas does not require:

- deferred checking;
- unique constraint definitions with support for:
  - alternate keys that allow null values;
- referential constraint definitions with support for:
  - references to alternate keys;
  - delete rules;
  - SET NULL and SET DEFAULT update rules;
  - match type;
- check constraint definitions.

If we permit updates on keys or identifiers, the mapping will require:

- the CASCADE update rule.

Therefore, when we restrict ourselves to level 0 entity-relationship schemas and we do not permit updates on keys or identifiers, Entry SQL suffices, even without support for references to alternate keys and check constraint definitions.

Indeed, assume for simplicity that attributes of entity-relationship schemas directly map into SQL and ignore the renaming of attributes and similar details. We outline in what follows a mapping from level 0 schemas into SQL schemas that associates each entity or relationship scheme with a distinct table definition.

Let **A** be a level 0 entity-relationship schema. The corresponding SQL schema $\mathbf{S_A}$ is constructed as follows. Each entity scheme E in **A** is mapped into a table definition $\mathsf{T_E}$ of $\mathbf{S_A}$ such that:

- the attributes of E are directly mapped into attributes of $\mathsf{T_E}$;
- the primary key defined in E is mapped into a unique constraint definition included in $\mathsf{T_E}$, using the PRIMARY KEY clause;

- the alternate keys defined in E are also mapped into unique constraint definitions included in T_E, using the UNIQUE clause;

- if E specializes an entity scheme F, then T_E also contains the attributes of the primary key K_F declared in T_F and a unique constraint definition indicating that K_F is its primary key. All user defined keys of E will then be mapped into alternate keys of T_E. Moreover, T_E includes a referential constraint definition indicating that K_F in T_E is a foreign key referencing K_F in T_F; the match type and the delete rule are omitted; the update rule is CASCADE; and the constraint mode is immediate.

The semantics of the referential constraint definition and the fact that the foreign key K_F of T_E does not admit null values (because K_F in T_E reflects the primary key of T_F) guarantee that each row in T_E references a row in T_F. Moreover, since K_F is a key of T_E, each row in T_F is referenced by at most one row in T_E. Therefore, each row in T_E corresponds to exactly one row in T_F and each row in T_F corresponds to at most one row in T_E. This suffices to characterize that the referential constraint definition correctly captures that, in any consistent database state σ of the ER schema $\mathbf{A}$, $\sigma(E) \subseteq \sigma(F)$. The inheritance of attributes from F to E can be captured by defining a view over T_E and T_F.

Furthermore note that the match type need not be specified as FULL essentially because the foreign key K_F of T_E does not admit null values; the delete rule is omitted and the constraint mode cannot be specified as deferrable because the way deletions involving specializations are treated in level 0 entity-relationship schemas. The update rule is CASCADE just to propagate to T_E any update to T_F that affects K_F, if updates that modify primary key values are allowed.

Each relationship scheme R in $\mathbf{A}$ is similarly mapped into a table definition T_R of $\mathbf{S_A}$:

- the attributes of R are directly mapped into attributes of T_R;

- for each participant P_i defined in R, T_R imports the attributes of the primary key K_{P_i} declared for T_{P_i} (if $P_i = P_j$, for some i and j, then the attributes will naturally have to be renamed when imported more than once). Moreover, T_R includes a referential constraint definition indicating that K_{P_i} in T_R is a foreign key referencing K_{P_i} in T_{P_i}; the match type and the delete rule are omitted; the update rule is CASCADE; and the constraint mode is immediate;

- the identifiers defined in R are mapped into unique constraint definitions included in T_R and involving the attributes of the imported primary keys. If R includes no identifier, then T_R must have a unique constraint definition indicating that the concatenation of the primary keys of the participants of R is the primary key of T_R.

Exactly as argued for entity schemes, the referential constraint definitions correctly capture that the entities participating in a relationship must exist in the corresponding sets. Again, the treatment of deletions and updates involving relationship sets in level 0 entity-relationship schemas requires that the match type and the delete rule be omitted and that the constraint mode be immediate. The update rule is CASCADE just to propagate to T_R any update to T_{P_i} that affects K_{P_i}, if updates that modify primary key values are allowed.

Finally, we note that the mapping just outlined generates a primary key for each table definition, which is essential for capturing the semantics of specializations and relationship sets via referential constraint definitions. However, this property of the mapping depends on the following requirements of level 0 schemas:

- each entity scheme has a primary key, defined directly or inherited from the scheme it specializes;

- the specialization declarations induce a hierarchical organization for the entity schemes;

- a relationship scheme may participate in other relationship schemes, as long as no circular definition is created.

The first two requirements are not mandatory for ER schemas and were included just to simplify the mapping. We will show in the next section how to relax the second requirement and modify the mapping accordingly, but we will maintain the first one because SQL does not provide surrogates (internally generated primary keys) [11]. The third requirement cannot be dropped on the account of the semantics of relationship schemes, as already pointed out.

As a simple example, consider a level 0 schema **A** with:

- an entity scheme EMP, whose attributes are ID, with domain CHAR(5), and NAME, with domain CHAR(25), and whose primary key is ID;
- an entity scheme DEPT, whose attributes are D#, with domain CHAR(3), and LOCATION, with domain CHAR(20), and whose primary key is D#;
- a relationship scheme WORK over EMP and DEPT, with identifier EMP (by definition of level 0 schemes, deletions from EMP and DEPT block immediately with respect to WORK).

When applied to schema **A**, the mapping outlined in this section produces three tables:

```
create table T_EMP
     (ID char(5) not null,
      NAME char(25),
      primary key (ID))

create table T_DEPT
     (D# char(3) not null,
      LOCATION char(20),
      primary key (D#))

create table T_WORK
     (ID char(5) not null,
      D# char(3) not null,
      foreign key (ID) references T_EMP
      foreign key (D#) references T_DEPT)
```

4. MAPPING OF LEVEL 1 ENTITY-RELATIONSHIP SCHEMAS

4.1. Level 1 entity-relationship schemas

A *level 1 entity-relationship schema* (or *level 1 schema*) is defined exactly as a level 0 schema, except for the following additional facilities.

First, the definition of relationship schemes is enhanced as follows. A relationship scheme R may be declared as *total* with respect to a participant P_i. Furthermore, the definition of R may indicate, for each participant P_i, whether deletions from P_i should propagate to R. Conversely, when R is total on P_i, the definition of R may also indicate whether deletions from R should propagate to P_i.

The semantics of these extensions are as follows. Let σ be a consistent database state of the ER schema. If R is total with respect to P_i, then the projection of $\sigma(R)$ on P_i must be equal to $\sigma(P_i)$. Propagating deletions from P_i to R means that the deletion of an instance p of $\sigma(P_i)$ will force the deletion of all relationships in $\sigma(R)$ where p occurs. When R is total on P_i, propagating deletions from R to P_i means the deletion of a relationship r from $\sigma(R)$ should cause the deletion of the instance p in $\sigma(P_i)$ such that p is the projection of r on P_i and there is no relationship s in $\sigma(R)$ such that $s \neq r$ and p is also the projection of s on P_i.

Second, specializations are relaxed as follows. An entity scheme E may specialize more than one entity scheme, as long as E does not transitively specialize itself. When E specializes F, the database designer may also declare that deletions from F should propagate to deletions from E. The blocking or propagation of deletions may also be declared as immediate or deferred.

Finally, insertions into entity or relationship sets behave as for level 0 schemas, except that the blocking of insertions may be declared as immediate or deferred.

The possibility of deferring the blocking of insertions is essential when totality is considered. Indeed, suppose that R is total with respect to a participant P_i. Let σ be a database state, r be a relationship to be inserted into $\sigma(R)$ and p be the projection of r on P_i. Suppose that p does not exist in $\sigma(P_i)$. Then, any transaction inserting r into $\sigma(R)$ must also insert p into $\sigma(P_i)$. Hence, either insertions into $\sigma(R)$ must block deferredly or insertions into $\sigma(P_i)$ must do so, as otherwise such insertions would be impossible.

The only alternative to blocking insertions deferredly would be to map R and P_i into a single table T and have the user insert a single row into T to represent both the insertion into R and into P_i. However, T would be normalized in this case only when P_i is an identifier of R, that is, when each relationship in $\sigma(R)$ is associated with at most one instance in $\sigma(P_i)$, for any consistente database state σ.

4.2. Mapping into SQL

The mapping of level 1 entity-relationship schemas to SQL still does not require:

- unique constraint definitions with support for
  - alternate keys that allow null values;
- referential constraint definitions with support for:
  - match type;
  - the SET DEFAULT and SET NULL update and delete rules;

but it requires:

- deferred checking;
- referential constraint definitions with support for:
  - references to alternate keys;
  - the CASCADE delete rule;
  - the CASCADE update rule, if we permit updates on keys or identifiers;
- check constraint definitions.

Because of deferred checking, we must use Full SQL in this case. However, if we do not permit updates on keys or identifiers as well as the use of totality, Intermediate SQL, even without support for alternate keys with nulls, match type and update rules, suffices to handle the mapping of level 1 entity-relationship schemas. But Entry SQL does not suffice, if the level 1 schema requires deferred checking and delete rules.

We will discuss in what follows how to modify the mapping suggested in Section 3.2 to account for level 1 schemas. In general, all referential constraint definitions must include delete rules and constraint modes whenever appropriate, since level 1 entity-relationship schemas consider the possibility of propagating deletions and of deferring the checking of certain implicit constraints, as already pointed out.

Totality can be handled as follows. Consider a relationship scheme R which is total with respect to a participant P_i. Suppose first that P_i is an identifier for R, in which case the primary key K_{P_i} of T_{P_i} is also a primary or alternate key of T_R. Then, the definition of table T_{P_i} must include

a referential constraint definition indicating that K_{P_i} in T_{P_i} is a foreign key referencing the key K_{P_i} in T_R; the match type and the update rule are omitted; the delete rule is either specified as CASCADE or omitted, depending on whether deletions from P_i propagate to R or not; finally the constraint modes DEFERRABLE or NOT DEFERRABLE are included depending on whether the scheme R specifies that the blocking of insertions or the blocking or propagation of deletions from P_i should be made deferredly or immediately.

Suppose now that P_i is not an identifier for R and, hence, the primary key K_{P_i} of T_{P_i} is not a key of T_R. Therefore, the previous solution does not apply because the referential constraint definition requires that K_{P_i} be a key of T_R. In this case, the suggested mapping is to include a check constraint definition in the definition of table T_{P_i} indicating that, for each row p in T_{P_i}, there must be a row r in T_R such that $p[K_{P_i}] = r[K_{P_i}]$. Note that the mapping will be correct only when the relationship scheme R does not indicate that deletions from R should propagate to P_i , since the check constraint definition does not offer triggered actions. But the mapping still permits specifying that the blocking of deletions should be deferred. Also note that such check constraint definition will involve a subquery to T_{P_i}.

The rest of this section analyses, with the help of examples, how to handle the case of an entity set that is a specialization of more than one entity set. We shall argue that supporting references to alternate keys is required in this case, as otherwise the mapping becomes somewhat complex and inefficient.

Consider a level 1 schema **A** that contains four entity schemes, A, B, C and D, such that the primary key of A is K, the primary key of C is L, the schemes B and C specialize A, the scheme D specializes B and C and the schemes contain no other attributes, except those implied by their keys, and that the domain of K and L is char(20), say. For all specializations, assume that insertions and deletions block immediately. The specialization graph of schema **A** is shown in Figure 1.

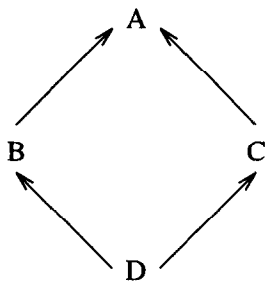

Fig. 1. Specialization graph of schema

If we apply the mapping outlined in Section 3.2 to this schema, we obtain the following table definitions:

```
create table T_A
     (K char(20) not null,
      primary key (K))

create table T_B
     (K char(20) not null,
      primary key (K),
      foreign key (K) references T_A
           on update cascade)
```

```
create table T_C
     (K char(20) not null,
      L char(20) not null,
      primary key (L),
      unique (K),
      foreign key (K) references T_A
           on update cascade)

create table T_D
     (K char(20) not null,
      L char(20) not null,
      primary key (L),
      unique (K),
      foreign key (K) references T_B
           on update cascade,
      foreign key (L) references T_C
           on update cascade)
```

The choice of K or L as the primary key of table T_D is immaterial. The problem of the above mapping lies in that it allows table values that do not correctly capture the semantics of specializations. For example, the tables shown in Figure 2 are consistent with the unique and referential constraint definitions included in the table definitions for T_A, T_B, T_C and T_D. Table T_A represents two entities, say, e_1 and e_2, identified by the key values k_1 and k_2, respectively. Table T_C also represents the entities e_1 and e_2. Moreover, it indicates that entity e_1 can also be identified by the key value l_1 and entity e_2 by the key value l_2. Likewise, table T_D also represents the entities e_1 and e_2. However, it incorrectly associates the key value l_2 with entity e_1 and the key value l_1 with entity e_2.

T_A

K
k_1
k_2

T_B

K
k_1
k_2

T_C

K	L
k_1	l_1
k_2	l_2

T_D

K	L
k_2	l_1
k_1	l_2

Fig. 2. Table instances violating the semantics of specializations

To solve the problem raised when we relax the assumption that each entity set specializes at most one entity set, we have to revise the mapping of specialization declarations into referential constraint definitions. Briefly, the general idea is to replicate the primary key of each sink A of the specialization graph to each scheme E such that there is a path from E to A in the graph and use such keys to generate referential constraint definitions. In our running example, the specialization graph has just one sink, which is the entity scheme A. Therefore, the primary key K of T_A is replicated in T_B, T_C and T_D, and used to generate referential constraint definitions as follows:

```
create table T_A
     (K char(20) not null,
      primary key (K))

create table T_B
     (K char(20) not null,
      primary key (K),
      foreign key (K) references T_A
           on update cascade)
```

```
create table T_C
    (K char(20) not null,
     L char(20) not null,
     primary key (K),
     unique (L),
     foreign key (K) references T_A
          on update cascade)

create table T_D
    (K char(20) not null,
     primary key (K),
     foreign key (K) references T_B
          on update cascade,
     foreign key (K) references T_C
          on update cascade)
```

Note that the above solution uses only primary keys as the referencing keys. However, more complex specialization configurations require the use of alternate keys as the referencing keys. Consider, for example, a second level 1 schema **B** that contains seven entity schemes, named A to G, such that the primary key of A is K, the primary key of B is L, schemes C and D specialize A, schemes D and E specialize B, scheme F specializes C and D, and scheme G specializes D and E. As in the previous example, assume that the schemes contain no other attributes, except those implied by their keys, and that the domain of K and L is again char(20). For all specializations, assume that insertions and deletions block immediately. The specialization graph of schema **B** is shown in Figure 3.

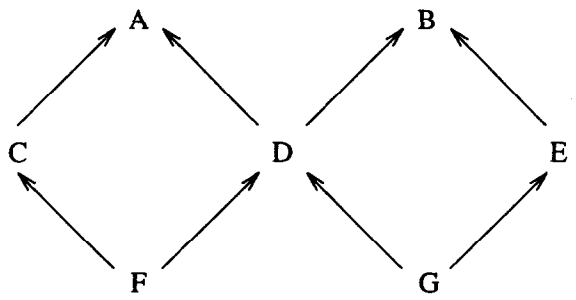

Fig. 3. Specialization graph of schema

Note that both K and L are candidates for the primary key of T_D. Suppose that we choose K (choosing L is symmetric). Then, to avoid a problem similar to that raised by mapping the schema **A** into SQL, we have to include in the table definition for T_G a referential constraint definition indicating that L is a foreign key of T_G, with L in T_D as the referenced key. Hence, using the alternate key L of T_D is essential to correctly capture the semantics of specializations. The definitions of tables T_D, T_F and T_G would then be (the definitions of the other tables are omitted for brevity):

```
create table T_D
    (K char(20) not null,
     L char(20) not null,
     primary key (K),
     unique (L),
     foreign key (K) references T_A
          on update cascade,
     foreign key (L) references T_B
          on update cascade)
```

```
create table T_F
     (K char(20) not null,
      primary key (K),
      foreign key (K) references T_C
           on update cascade,
      foreign key (K) references T_D
           on update cascade)

create table T_G
     (L char(20) not null,
      primary key (L),
      foreign key (L) references T_D(L)
           on update cascade,
      foreign key (L) references T_E
           on update cascade)
```

If referenced keys were limited to primary keys only, the above mapping would not be valid. In this case, an alternative mapping, which is not so elegant, could be defined by redundantly choosing the concatenation of K and L as the primary key of T_D and adjusting the referential constraint definitions in T_F and T_G accordingly. The definitions of tables T_D, T_F and T_G would now be (the definition of the other tables is again omitted for brevity):

```
create table T_D
     (K char(20) not null,
      L char(20) not null,
      primary key (K,L)
      unique (K),
      unique (L),
      foreign key (K) references T_A
           on update cascade,
      foreign key (L) references T_B
           on update cascade)

create table T_F
     (K char(20) not null,
      L char(20) not null,
      primary key (K),
      unique (L),
      foreign key (K) references T_C
           on update cascade,
      foreign key (K,L) references T_D
           on update cascade)

create table T_G
     (K char(20) not null,
      L char(20) not null,
      primary key (L),
      unique (K),
      foreign key (K,L) references T_D
           on update cascade,
      foreign key (L) references T_E
           on update cascade)
```

To summarize, the mapping outlined in Section 3.2 can be extended to level 1 schemas, provided that:

- a referential constraint or a check constraint (with subqueries) definition is included in table T_{P_i}, if P_i participates in a relationship scheme R and R is total on P_i;
- the unique constraint and the referential constraint definitions that capture specializations are adjusted as outlined above.

In both cases, the mapping may require the use of references to alternate keys.

4.3. Optimization

For each ER schema **A**, the mappings outlined so far produce a one-to-one relational representation $\mathbf{S_A}$, in the sense that $\mathbf{S_A}$ contains a distinct table definition for each entity or relationship scheme of **A**. One-to-one representations are fairly straightforward to obtain, but they contain a potentially large number of referential constraint definitions that are expensive to check for violations.

To reduce the number of referential constraint definitions of a relational representation, a fairly common heuristics is to collapse a relationship scheme R into an entity scheme E and to represent both as a single table definition T, if E is an identifier of R. The most frequent case is when R is binary and **n:1**, with E "on the **n** side". The same heuristics applies as well when F is an entity scheme that specializes E, possibly restricted to the case where F has few attributes. The reader is referred to [7, 8] for a complete discussion of this topic.

We analyze in this section, with the help of examples, the impact this type of optimization has on the mapping of level 1 schemas into SQL. We argue that the mapping will require:

- deferred checking;
- unique constraint definitions with support for:
  - alternate keys that allow null values;
- referential constraint definitions with support for:
  - references to alternate keys;
  - the CASCADE and SET NULL delete rules;
  - the CASCADE update rule, if we permit updates on keys or identifiers;
- check constraint definitions.

Moreover, except for the most trivial forms of optimization, SQL would have to be extended to cover a more sophisticated form of propagation of nulls. However, we still do not need match type, update rules and the SET DEFAULT delete rule.

To facilitate the presentation of the examples, we will use T_P to denote the table representing only the entity or relationship scheme P and T_P* to denote the table representing P and possibly some other scheme.

We begin with an example that illustrates that the most common form of optimization requires the use of the SET NULL delete rule. In fact, it provides a very reasonable explanation for this delete rule, which otherwise has no justification from the conceptual design point of view. Consider the same schema **A** introduced at the end of Section 3.2, but suppose now that deletions from EMP and DEPT propagate immediately to WORK. This makes **A** a level 1 schema.

When applied to schema **A**, the mapping outlined in the previous section produces three tables, one for each scheme. However, since EMP is an identifier of WORK and since deletions from EMP propagate immediately to WORK, we may use a single table to represent both EMP and WORK. The final SQL schema $\mathbf{S_A}$ will contain the following table definitions:

```
create table T_EMP*
     (ID char(5) not null,
      NAME char(25),
      D# char(3),
      primary key (ID),
      foreign key (D#) references T_DEPT
           on delete set null)

create table T_DEPT
     (D# char(3) not null,
      LOCATION char(20),
      primary key (D#))
```

Note that the delete rule of the referential constraint definition in the definition of T_EMP* specifies SET NULL. Intuitively, this is required because a row t in T_EMP* represents an employee e and, if t[D#] $= d$ and $d \neq$ null, the relationship between e and the department d. Hence, to propagate the deletion of d from T_DEPT, t[D#] must be set to null, which is exactly the semantics of the SET NULL delete rule. Furthermore, note that, at the entity-relationship level, deletions from EMP must propagate immediately to WORK since the deletion of a row t from T_EMP* represents the deletion of both an employee e and the relationship between e and the department e works for.

To recover the property that each entity or relationship scheme corresponds to a unique table, we may define two views as follows:

```
create view T_EMP (ID, NAME)
     as select ID, NAME
     from T_EMP*

create view T_WORK (ID, D#)
     as select ID, D#
     from T_EMP*
     where D# is not null
```

Our second example illustrates another use of the check constraint definition. Consider a level 1 schema **B** obtained by adding to the schema **A**:

- an entity scheme PROJ, whose attributes are P#, with domain CHAR(3), and DURATION, with domain CHAR(6), and whose primary key is P#;
- a relationship scheme PAY over PROJ and WORK whose identifier is WORK.

Suppose that deletions from PROJ block immediately with respect to PAY and that deletions from WORK block immediately with respect to PAY.

Then, the SQL schema $\mathbf{S_A}$ can be extended to represent the ER schema **B** by including the following table definitions:

```
create table T_PROJ
     (P# char(3) not null,
      DURATION char(6),
      primary key (P#))

create table T_PAY
     (ID char(5) not null,
      P# char(3) not null
      primary key (ID),
      foreign key (P#) references T_PROJ
      check ID match (select ID from T_EMP* where D# is not null))
```

Note that the check constraint definition cannot be replaced by a referential constraint definition indicating that ID in T_PAY is a foreign key referencing ID in T_EMP*. Indeed, a row in T_PAY can only reference a row in T_EMP* with a non null value for D# because only these rows represent a WORK relationship. A clearer way of capturing the semantics of the relationship scheme PAY with respect to WORK would in fact be through a referential constraint definition indicating that ID in T_PAY is a foreign key referencing ID in the view T_WORK, which is currently not supported by SQL. This observation is quite general and applies to many situations raised by representing more than one entity or relationship scheme in a single table.

Our third example illustrates that an optimized mapping may sometimes require the facility, provided by SQL, of defining an alternate key K for a table T over attributes that allow null values. In this case, the semantics of SQL dictates that, for any two rows t and u in T, if nulls do not occur in t[K] and u[K] and t[K] = u[K] then $t = u$.

Consider a level 1 schema **C**, defined exactly as the schema **A** above, except that DEPT is also an identifier of WORK. That is, assume that WORK defines a **1:1** relationship set. Then, the table T_EMP* has to be redefined as follows:

```
create table T_EMP*
    (ID char(5) not null,
    NAME char(25),
    D# char(3),
    primary key (ID),
    unique (D#),
    foreign key (D#) references T_DEPT
        on delete set null)
```

We stress that this mapping of the relationship scheme WORK and the entity scheme EMP into the same table T_EMP* correctly captures that the relationship scheme WORK defines a **1:1** relationship set because the unique constraint definition for D# filters out null values. That is, table T_EMP* may contain two rows t and u such that t[D#] and u[D#] are both null.

Again another way of explaining this mapping would be to observe that the view T_WORK over T_EMP* must have ID and D# as keys. Since ID is already a key of T_EMP*, it is naturally a key of T_WORK. The unique constraint definition indicating that D# is an alternate key of T_EMP* captures exactly the fact that ID is a key of the view T_WORK.

Our final example illustrates an optimization that creates a situation not supported by SQL. Let **D** be a level 1 schema with:

- entity schemes EMP and DEPT defined as for the schema **A**;
- an entity scheme PROJ defined as for the schema **B**;
- a relationship scheme WORK over EMP, DEPT and PROJ, with identifier EMP.

Suppose that deletions from EMP and DEPT propagate immediately to WORK, but deletions from PROJ block immediately with respect to WORK. Now, if we collapse WORK in EMP as in the mapping of schema **A**, we obtain the following table definitions:

```
create table T_EMP*
    (ID char(5) not null,
    NAME char(25),
    D# char(3),
    P# char(3),
    primary key (ID),
    foreign key (D#) references T_DEPT
        on delete set null,
    foreign key (P#) references T_PROJ)
```

```
create table T_DEPT
    (D# char(3) not null,
    LOCATION char(20),
    primary key (D#))

create table T_PROJ
    (P# char(3) not null,
    DURATION char(20),
    primary key (P#))
```

The definition of table T_EMP* is incorrect though. Indeed, the deletion of a row d from T_DEPT causes e[D#] to be set to null, for all rows e of T_EMP* that reference d. But the deletion does not force e[P#] to be set to null for all such rows. Hence, if the user wants to delete a row p from T_PROJ that is referenced by one of these rows e, the deletion will be incorrectly blocked.

A reasonable solution to this problem would be to include a delete rule that explicitly specifies which attributes should be set to null. In the definitions of table T_EMP* we would then have the following referential constraint definition:

```
foreign key (D#) references T_DEPT
    on delete set null (D#,P#)
```

5. MAPPING OF LEVEL 2 ENTITY-RELATIONSHIP SCHEMAS

5.1. Level 2 entity-relationship schemas

A *level 2 entity-relationship schema* (or *level 2 schema*) is defined exactly as a level 1 schema, except that it also permits defining generalization hierarchies, as opposed to specialization hierarchies, and certain types of inter-relationship constraints.

We consider two types of generalization in a level 2 schema: *non-total generalization* and *total generalization* [12, 16].

If an entity scheme G is declared as a *non-total generalization* of a list $E_1, ..., E_n$ of other entity schemes, then any consistent database state σ will be such that:

(1) $\sigma(E_i)$ is a subset of $\sigma(G)$;

(2) $\sigma(E_i)$ and $\sigma(E_j)$ are disjoint, for every i,j in [1,n] with $i \neq j$.

Furthermore, the dynamic behavior is such that:

(3) a deletion from G cascades immediately to E_i;

(4) an insertion into E_i blocks immediately, if **(1)** or **(2)** are violated, for every i in [1,n].

On the other hand, if an entity scheme G is declared as a *total generalization* of a list $E_1, ..., E_n$ of other entity schemes, then any consistent database state σ will be such that:

(5) $\sigma(G_i)$ is the union of $\sigma(E_1), ..., \sigma(E_n)$;

(6) $\sigma(E_i)$ and $\sigma(E_j)$ are disjoint, for every i,j in [1,n] with $i \neq j$.

Furthermore, the dynamic behavior is such that:

(7) a deletion from G cascades immediately to E_i and a deletion from E_i blocks immediately with respect to G;

(8) insertions into G and E_i must come together in the same transaction, as otherwise they will be blocked deferredly.

The inter-relationship contraints that can be defined in a level 2 schema are: *subset*, *equality*, *exclusion* and *total union*. These constraints are derived from the NIAM model [13, 16, 18, 23] and have been proposed as extensions to the ER model in [14, 21]. Their formal semantics is described below.

Let R be a relationship scheme over the entity schemes $E_1, ..., E_m$ and let S be a relationship scheme over $F_1, ..., F_n$. Recall that $\Pi_{E_i}(\sigma(R))$ denotes the i-th projection of $\sigma(R)$ (and similarly for S).

The first variation of the subset, equality and exclusion constraints impose restrictions on R and S only with respect to one specific participant of R, say, E_i, and one participant of S, say, F_j. More precisely, these constraints express that the database state σ will be consistent with each of them iff:

(9) $\Pi_{F_j}(\sigma(S)) \subseteq \Pi_{E_i}(\sigma(R))$ – subset constraint

(10) $\Pi_{F_j}(\sigma(S)) = \Pi_{E_i}(\sigma(R))$ – equality constraint

(11) $\Pi_{F_j}(\sigma(S)) \cap \Pi_{E_i}(\sigma(R)) = \emptyset$ – exclusion constraint

Assuming now that m=n, the second variation of the subset, equality and exclusion constraints imposes restrictions on R and S as a whole, in contrast with the previous definition. In detail, a database state σ will now be consistent iff:

(12) $\sigma(S) \subseteq \sigma(R)$ – subset constraint

(13) $\sigma(S) = \sigma(R)$ – equality constraint

(14) $\sigma(S) \cap \sigma(R) = \emptyset$ – exclusion constraint

Finally, the total union constraint expresses that a database state σ will be consistent with it iff:

(15) $\Pi_{F_j}(\sigma(S)) \cup \Pi_{E_i}(\sigma(R)) = \sigma(E_i)$ – total union constraint

where E_i is a participant of R and F_j is a participant of S.

5.2. Mapping into SQL

The mapping of level 2 entity-relationship schemas into SQL still does not require:

- referential constraint definitions with support for:
  - match type;
  - the SET DEFAULT and SET NULL update rules.

However, it additionally requires:

- assertions and check constraint for column definitions, both involving subqueries, when no optimization is done;
- check constraint for column definitions, without subqueries, when optimization is done.

Hence, as we can see from the above observations, the mapping of level 2 schemas requires Full SQL.

In the next sections we discuss with the help of examples the mappings of generalization hierarchies and inter-relationship constraints into SQL.

5.2.1. Mapping of generalization hierarchies

Non-total generalizations

A non-optimized mapping of non-total generalizations into SQL will use foreign keys to capture condition **(1)** and assertions to capture condition **(2)** of Section 5.1, while optimized mappings may use *discriminating attributes* [22], as the next set of examples illustrates.

Consider a level 2 schema **A** with:

- an entity scheme EMP, whose attributes are ID, with domain CHAR(5), and NAME, with domain CHAR(25), and whose primary key is ID;
- an entity scheme MGR, whose attribute is POS, with domain CHAR(25);
- an entity scheme ADM, whose attribute is SPEC, with domain CHAR(25);
- EMP defined as a non-total generalization of the other two schemes.

A non-optimized mapping, similar to those considered in the previous sections, would produce three tables and one assertion:

```
create table T_EMP
     (ID char(5) not null,
      NAME char(25) not null,
      primary key (ID))

create table T_MGR
     (ID char(5) not null,
      POS char(25) not null,
      primary key (ID),
      foreign key (ID) references T_EMP
           on delete cascade)

create table T_ADM
     (ID char(5) not null,
      SPEC char(25) not null,
      primary key (ID),
      foreign key (ID) references T_EMP
           on delete cascade)

create assertion A1
     check (not exists
               select ID from T_MGR
                    where ID in (select ID from T_ADM))
```

(by default, the assertion will be checked immediately).

Even without collapsing tables, we may optimize the above mapping by adding a discriminating attribute CAT to the generalized entity scheme. In our current example, the new mapping would then become:

```
create table T_EMP
     (ID char(5) not null,
      NAME char(25) not null,
      CAT char(1) check (CAT='M' or CAT='A' or CAT is null),
      primary key (ID),
      unique key (ID,CAT))
```

```
create table T_MGR
    (ID char(5) not null,
     POS char(25) not null,
     CAT char(1) check (CAT='M'),
     primary key (ID),
     foreign key (ID,CAT) references T_EMP(ID,CAT)
         on delete cascade)

create table T_ADM
    (ID char(5) not null,
     SPEC char(25) not null,
     CAT char(1) check (CAT='A'),
     primary key (ID),
     foreign key (ID,CAT) references T_EMP(ID,CAT)
         on delete cascade)
```

Note that this mapping eliminates assertion A1, that captured the mutual exclusion of T_MGR and T_ADM. However, it introduces an extra column, CAT, in all three tables. The check constraint definition associated with CAT in table T_EMP effectively enforces the mutual exclusion of T_MGR and T_ADM. However, the appropriated column values must be inserted by the user and redundantly repeated in the rows of tables T_MGR and T_ADM. The foreign keys of these two tables must include CAT to avoid mistaken references, in the sense that a row in T_MGR must indeed reference a row in T_EMP that is categorized as a manager, and symmetrically for rows of T_ADM.

As an alternative mapping, we would not repeat the discriminating column CAT in tables T_MGR and T_ADM and replace the referential constraint definitions of such tables by check constraint definitions that capture the appropriate references:

```
create table T_EMP
    (ID char(5) not null,
     NAME char(25) not null,
     CAT char(1) check (CAT='M' or CAT='A' or CAT is null),
     primary key (ID))

create table T_MGR
    (ID char(5) not null,
     POS char(25) not null,
     primary key (ID),
     check (ID in select ID from T_EMP where CAT='M'))

create table T_ADM
    (ID char(5) not null,
     SPEC char(25) not null,
     primary key (ID),
     check (ID in select ID from T_EMP where CAT='A'))
```

However, this second mapping may not be as efficient as the first one, if the underlying relational database management system has an optimized implementation of referential integrity. Furthermore, it requires changing condition **(3)** of the semantics of non-total generalizations to specify that deletions must block, if they violate condition **(1)**, since check constraint definitions may not specify the propagation of deletions (see section 5.1 for these conditions).

The best optimized mapping would, however, collapse all three tables into just one as follows:

```
create table T_EMP*
    (ID char(5) not null,
     NAME char(25) not null,
     CAT char(1),
     POS char(25),
     SPEC char(25),
     primary key (ID),
     check ((CAT='M' and POS is not null and SPEC is null) or
            (CAT='A' and POS is null and SPEC is not null) or
           ((CAT, POS, SPEC) is null)))
```

This mapping eliminates all foreign keys and introduces a simple check constraint definition that restricts the rows of T_EMP*. However, all references to T_MGR and to T_ADM have to be replaced by assertions or check constraint definitions that point to the appropriate tuples of T_EMP*. Moreover, users have to be aware that deletions, insertions and updates originally directed to T_MGR and T_ADM have to be replaced by updates to T_EMP*. These two problems may reduce the attractiveness of the optimization proposed [5].

Total generalizations

The mapping of total generalizations into SQL follows as before, except that additional check constraint definitions must be added since condition **(5)** in Section 5.1 is now more complex.

In our running example, if EMP is defined as a total generalization of the other two schemes, then the non-optimized mapping would additionally include in the definition of table T_EMP the following check constraint definition:

```
create table T_EMP
    (...
     check ((ID in select ID from T_MGR) or (ID in select ID from T_ADM)))
```

By analogy with non-total generalizations, the optimized mappings that do not use collapsed tables would include a slightly different check constraint definition for the discriminating attribute:

```
create table T_EMP
    (...
     CAT char(1) check (CAT='M' or CAT='A'),
     ...)
```

However, this mapping is incorrect because it does not guarantee the mutual exclusion of the two tables.

On the other hand, the optimized mapping that uses collapsed tables do work correctly, albeit it would also include a similar check constraint definition for the discriminating attribute:

```
create table T_EMP*
    (...
     check ((CAT='M' and POS is not null and SPEC is null) or
            (CAT='A' and POS is null and SPEC is not null)))
```

5.2.2. Mapping of inter-relationship constraints

In order to discuss the mappings of inter-relationship constraints, we will use as a running example a level 2 schema **B** with:

- an entity scheme PROJ, whose attributes are P#, with domain CHAR(3), and PNAME, with domain CHAR(20), and whose primary key is P#;

- entity schemes MGR and ADM as in Section 5.2.1;
- a relationship scheme PM over PROJ and MGR;
- a relationship scheme PA over PROJ and ADM.

Subset and equality constraints

To capture the first form of subset constraint, a non-optimized mapping uses either an assertion definition involving the foreign keys of the tables that correspond to the relationship schemes R and S, or a check constraint definition included in the definition of the table that corresponds to the relationship scheme S. An optimized mapping can be produced if the relationship schemes R and S are identified by E_i and F_j respectively. In this case, S can be collapsed into R or, alternatively, S and R can be collapsed into E_i and all represented as a single table definition containing a check constraint definition.

Now assume the following subset constraint for our running example:

(SC) $\Pi_{PROJ}(\sigma(PM)) \subseteq \Pi_{PROJ}(\sigma(PA))$, for any consistent database state σ.

The first non-optimized mapping alternative will produce the following assertion definition to capture the constraint SC:

```
(SC1)   create assertion SC1
            check (select P# from PM in select P# from PA)
```

An equivalent mapping would include a check constraint in the table definition of PM:

```
(SC2)   create table PM
            (P# char(3) not null,
             ID char(5) not null,
             primary key (P#,ID),
             foreign key (P#) references PROJ
                  on delete cascade,
             foreign key (ID) references T_MGR
                  on delete cascade,
             check (P# match (select P# from PA)))
```

In the above table definition, note that the check constraint cannot be replaced by a referential constraint definition since P# is not a key of PA.

Suppose now that PA and PM are both **n:1** on PROJ. A first optimized mapping would collapse PM into PA, producing the following table definition:

```
(SC3)   create table PA*
            (P# char(3) not null,
             ADM_ID char(5) not null,
             MGR_ID char(5),
             primary key (P#),
             foreign key (P#) references PROJ
                  on delete cascade,
             foreign key (ADM_ID) references ADM
                  on delete cascade,
             foreign key (MGR_ID) references MGR
                  on delete set null)
```

The above table definition correctly captures SC since each row of PA* represents a relationship PA and a relationship PM involving the same project, when MGR_ID is not null.

A second optimized mapping would collapse both relationships on PROJ, producing the following table definition:

```
(SC4)   create table PROJ*
            (P# char(3) not null,
             PNAME char(20),
             ADM_ID char(5),
             MGR_ID char(5),
             primary key (P#),
             foreign key (ADM_ID) references ADM
                  on delete set null,
             foreign key (MGR_ID) references T_MGR
                  on delete set null,
             check (ADM_ID is not null or MGR_ID is null))
```

The check constraint in the above table definition guarantees that, if a project has a manager, then it also has some administrative staff, which intuitively is what SC says.

Let us now consider the mapping of the second form of the subset constraint. Suppose that in our example we have the following constraint:

(SC') $\sigma(\text{PM}) \subseteq \sigma(\text{PA})$, for any consistent database state σ.

Thus the previous mappings would be altered as follows:

```
(SC1')  create assertion SC1
            check (select P#, MGR_ID from PM in select P#, ADM_ID from PA)

(SC2')  create table PM
            (P# char(3) not null,
             ID char(5) not null,
             primary key (P#,ID),
             foreign key (P#) references PROJ
                  on delete cascade,
             foreign key (ID) references T_MGR
                  on delete cascade,
             foreign key (P#,ID) references PA
                  on delete cascade)

(SC3')  create table PA*
            (P# char(3) not null,
             ADM_ID char(5) not null,
             MGR_ID char(5),
             primary key (P#),
             foreign key (P#) references PROJ
                  on delete cascade,
             foreign key (ADM_ID) references ADM
                  on delete cascade,
             foreign key (MGR_ID) references T_MGR
                  on delete set null,
             check (MGR_ID is null or MGR_ID=ADM_ID))

(SC4')  create table PROJ*
            (P# char(3) not null,
             PNAME char(20),
```

```
    ADM_ID char(5),
    MGR_ID char(5),
    primary key (P#),
    foreign key (ADM_ID) references ADM
        on delete set null,
    foreign key (MGR_ID) references T_MGR
        on delete set null,
    check (MGR_ID is null or MGR_ID=ADM_ID))
```

Note that mapping SC3' works correctly even for **n:m** relationships whereas mapping SC4' assumes that PA and PM are both **n:1** on PROJ. The differences between this collection of mappings and the previous one just reflect the fact that SC' forces PM to be a subset of PA, whereas SC requires that only the projection of PM on PROJ be a subset of the projection of PA on PROJ.

Finally, the mappings of the subset constraint discussed above can be easily adapted to capture the equality constraint. For instance, in our running example, mapping SC4 would just require replacing the check constraint definition by the following one:

```
check ((MGR_ID,ADM_ID) is not null or (MGR_ID,ADM_ID) is null))
```

Exclusion constraint

The mapping of the exclusion constraint is also similar to that of the subset constraint. A non-optimized mapping uses an assertion definition that directly captures the constraint. An optimized mapping can be produced if the relationship schemes R and S are identified by E_i and F_j respectively. In this case, R and S can be collapsed into E_i and all represented as a single table definition containing a check constraint definition.

Assume that the following exclusion constraint holds in our example:

(EC) $\Pi_{PROJ}(\sigma(PM)) \cap \Pi_{PROJ}(\sigma(PA)) = \emptyset$, for any consistent database state σ.

The non-optimized mapping will produce the following assertion definition to capture EC:

```
(EC1)   create assertion EC1
            check (not exists
                    select * from PM
                        where P# in (select P# from PA))
```

Suppose now that PA and PM are both **n:1** on PROJ. An optimized mapping would collapse both relationships on PROJ, producing the following table definition:

```
(EC2)   create table PROJ*
            (P# char(3) not null,
             PNAME char(20),
             ADM_ID char(5),
             MGR_ID char(5),
             primary key (P#),
             foreign key (ADM_ID) references T_ADM
                 on delete set null,
             foreign key (MGR_ID) references T_MGR
                 on delete set null,
             check (ADM_ID is null or MGR_ID is null))
```

The check constraint in the above table definition guarantees that, if a project has a manager, then it does not have an administrative staff, and vice-versa, which intuitively is what EC says.

Let us consider now the mapping of the second form of the exclusion constraint. Suppose that in our example we have the following constraint:

(EC') $\sigma(\text{PM}) \cap \sigma(\text{PA}) = \emptyset$, for any consistent database state σ.

The mapping of EC' into an assertion definition is immediate. As for an optimized mapping, we could alter the mapping EC2 as follows:

```
(EC2')   create table PROJ*
           (P# char(3) not null,
            PNAME char(20),
            ADM_ID char(5),
            MGR_ID char(5),
            primary key (P#),
            foreign key (ADM_ID) references T_ADM
                 on delete set null,
            foreign key (MGR_ID) references T_MGR
                 on delete set null,
            check (MGR_ID is null or MGR_ID<>ADM_ID))
```

As for EC2, the mapping EC2' also assumes that PA and PM are both **n:1** on PROJ.

Total union constraint

To capture the total union constraint, a non-optimized mapping also uses an assertion definition involving the foreign keys of the tables that correspond to the relationship schemes R and S. Referring to condition **(15)** in Section 5.1, an optimized mapping can be produced as before, if the relationship schemes R and S are identified by E_i and F_j respectively, by collapsing R and S into E_i and representing them as a single table definition containing a check constraint.

Thus assume the following total union constraint for our running example

(TC) $\Pi_{\text{PROJ}}(\sigma(\text{PM})) \cup \Pi_{\text{PROJ}}(\sigma(\text{PA})) = \sigma(\text{PROJ})$, for any consistent state σ.

The non-optimized mapping will then produce:

```
(TC1)    create assertion TC1
           check (not exists
                    select * from PROJ
                         where P# not in (select P# from PA)
                         and   P# not in (select P# from PM)
```

This mapping is correct since we always have

$$\Pi_{\text{PROJ}}(\sigma(\text{PM})) \cup \Pi_{\text{PROJ}}(\sigma(\text{PA})) \subseteq \sigma(\text{PROJ}),$$

for any consistent state σ.

Suppose now that PA and PM are both **n:1** on PROJ. An optimized mapping would collapse both relationships on PROJ, producing the following table definition:

```
(TC2)    create table PROJ*
           (P# char(3) not null,
            PNAME char(20),
            ADM_ID char(5),
            MGR_ID char(5),
            primary key (P#),
            foreign key (ADM_ID) references T_ADM
```

```
    on delete set null,
foreign key (MGR_ID) references T_MGR
    on delete set null,
check (ADM_ID is not null or MGR_ID is not null))
```

The check constraint in the above table definition guarantees that a project always has either a manager or an administrative staff, or both, which intuitively is what TC says.

6. CONCLUSIONS

The analysis presented in this paper ultimately leads to a classification of some of the constraint features of SQL according to the level of support they offer for the ER model. It also suggests that one of the major drawbacks of SQL, as expected, is the lack of support for surrogates (i.e., internally defined keys). This forced us to introduce restrictions on the ER model to guarantee that every scheme has a primary key. The lack of surrogates also complicated the definition of the mappings, since we had to guarantee that every table has the "right" set of keys and referential constraint definitions. It also required cascading the updates on referenced keys.

The analysis also revealed that some features, such as SET DEFAULT and match type, are unnecessary as far as mapping ER schemas into SQL. (The match type is not needed essentially because foreign keys never overlap in the mappings defined in Sections 3, 4 and 5). On the other hand, it suggested that the SET NULL option should be generalized to allow the propagation of nulls to attributes outside the foreign key.

The mapping of level 2 schemas to the relational model requires mechanisms to capture subsetting and mutual exclusion. Referential integrity definitions, as specified in SQL, handles most cases of subsetting, which is important since some commercial database management systems have efficient implementations of this mechanism. However, the analysis presented in this paper reveals that check constraint definitions must be used to capture subsetting when the referenced columns do not form a key of the referenced table, or when some rows of the referenced table cannot be referenced. Since such check constraint definitions may not be handled efficiently and since the full generality of this type of statement is not actually needed, implementors of relational systems should try to extend the referential integrity mechanism to cover these cases directly.

Mutual exclusion is very expensive to handle directly, especially when the list of classes that must be mutually exclusive is not small. The ideal situation occurs when mutual exclusion reduces to comparing column values of the same row or to the use of discriminating columns, which can be captured by check constraint definitions with very simple conditions. But this ideal situation can sometimes be achieved only by introducing artificial columns or by collapsing tables. In both cases, users must be instructed on how to properly write updates, which is error prone. Therefore, language designers and implementors should devise new special classes of constraints that capture

ER schema	*SQL Level*	*Special Requirements*
Level 0	Entry	(none)
Level 1	Full	deferred checking, deletion rules and check constraint definitions
Level 1 (optimized)	Full	alternate keys with null values
Level 2	Full	assertions and check constraint for column definitions

Note: If totality is not used, then deferred checking is not needed, which implies that Intermediate SQL suffices to model level 1 ER schemas

Table 2. Summary of the analysis

mutual exclusion and whose implementation hides the details that the mappings outlined in this paper make visible to the users.

To conclude, Table 2 summarizes the results of our analysis showing what SQL subset is required to map which level of the ER model.

REFERENCES

[1] ANSI X3H2-90-264 / ISO IEC JTC1 SC21 WG3. (ISO working draft) Database Language SQL2 (J. Melton, ed.) (1990).

[2] A. Atri and D. Sacca. Equivalence and mapping of database schemes. *Proc. 10th Int. Conf. on Very Large Data Bases*, Singapore (1984).

[3] N. Azar and E. Pichat. Translation of an extended entity-relationship model into the universal relation with inclusion formalism. *Proc. 5th Int. Conf. on the Entity-Relationship Approach*, Dijon, France (1986).

[4] C. Batini, S. Ceri and S. Navathe. *Conceptual Database Design: An Entity-Relationship Approach.* Benjamin Cummings, Redwood City, California (1992).

[5] M.A. Casanova, L. Tucherman, A.L. Furtado and A. Pacheco. Optimization of relational schemas containing inclusion dependencies. *Proc. 15th Int. Conf. on Very Large Data Bases*, Amsterdam, Holland (1989).

[6] M.A. Casanova, L. Tucherman, P.M. Gualandi, A. Pacheco and M.R. Cavalcanti. A data definition language for an extended entity-relationship model. Technical Report CCR072, Rio Scientific Center, IBM Brazil (1989).

[7] M.A. Casanova, L. Tucherman and A.H.F. Laender. Algorithms for designing and maintaining optimized relational representations of entity-relationship schemas. *Proc. 9th Int. Conf. on Entity-Relationship Approach*, Lausanne, Switzerland (1990).

[8] M.A. Casanova, L. Tucherman and A.H.F. Laender. On the design and maintenance of optimized relational representations of entity-relationship schemas. *Data and Knowledge Engineering* **11**(1), 1-20 (1993).

[9] M.A. Casanova, A.P. Carvalho, L.F.G.G.M. Ridolfi and A.H.F. Laender. An analysis of table constraints in SQL2 based on the entity-relationship model. *Proc. 10th Int. Conf. on Entity-Relationship Approach*, San Mateo, California (1991).

[10] P.P. Chen. The entity-relationship model: toward a unified view of data. *ACM Transactions on Database Systems* **1**(1), 9-36 (1976).

[11] E.F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems* **4**(4), 397-434 (1979).

[12] R. Elmasri and S. Navathe. *Fundamentals of Database Systems.* Benjamin Cummings, Redwood City, California (1989).

[13] D.J. Flynn and A.H.F. Laender. Mapping from a conceptual schema to a target internal schema. *The Computer Journal* **28**(5), 508-517 (1985).

[14] G. Guido. Semantic modeling of an accounting universe of discourse - the usefulness of inter-relationship constraints. *Proc. 10th Int. Conf. on Entity-Relationship Approach*, San Mateo, California (1991).

[15] Joint Technical Committee ISO/IEC JCT1. International Standard ISO/IEC 9075:199x(E) - Database Language SQL (1991).

[16] A.H.F. Laender and D.J. Flynn. A semantic comparison of the modelling capabilities of the ER and NIAM models. *Proc 12th Int. Conf. on Entity-Relationship Approach*, Arlington, Texas (1993).

[17] V.M. Markowitz and A. Shoshani. Representing extended entity-relationship structures in relational databases: a modular approach. *ACM Transactions on Database Systems* **17**(3), 423-464 (1992).

[18] G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design: A Fact-Oriented Approach.* Prentice-Hall, Sydney, Australia (1989)

[19] A. Pacheco. Correct translation of entity-relationship schemas into relational schemas. *Proc. of the 9th Congress of the Brazilian Computer Society*, Uberlândia, Brazil, In Portuguese.(1989).

[20] L. Ridolfi, A.P. Carvalho and M.A. Casanova. A tool for conceptual database design based on the entity relationship model. Technical Report CCR130, Rio Scientific Center, IBM Brazil (Nov. 1991). In Portuguese.

[21] T.J. Teorey. *Database Modeling and Design: The Entity-Relationship Approach.* Morgan Kaufmann, San Mateo, California (1990).

[22] T.J. Teorey, D. Yang and J.P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Survey* **18**(2), 197-222 (1986).

[23] G. Verheijen and J. van Bekkum. NIAM: An information analysis method. In *Information Systems Design Methodologies: A Comparative Review* (T.W. Olle, H.G. Sol and A.A. Verrijn-Stuart, eds.). North-Holland, Amsterdam (1982).