

On the Mapping of NIAM Schemas into SQL

Alberto H.F. Laender¹

Donal J. Flynn²

Marco A. Casanova³

Abstract

The design of relational databases typically starts with the definition of a conceptual schema which describes the conceptual structures of the database in terms of a high-level data model, such as the ER and NIAM models, and then proceeds by mapping this conceptual schema into a relational representation. However, this mapping has been usually described in terms of a basic “canonical” relational model, which means that most of the integrity constraints specified at the conceptual level are not fully captured in the final relational representation. The recent SQL standard proposal has provided a new framework for discussing the design of relational databases, since it introduces a number of facilities which capture most of the integrity constraints specified in a conceptual schema. In this paper, we discuss the general aspects involved in the mapping of NIAM schemas into SQL. We present a mapping procedure which handles the basic NIAM constructs and then discuss how to extend it to cope with the more complex ones. We also address the problem of optimizing the SQL representation of NIAM schemas to reduce the number of referential constraint definitions in the final SQL representation. We point out that, besides reducing the number of referential constraint definitions, the mapping strategy applied also simplifies the mapping of NIAM inter-relationship constraints.

1 Introduction

The design of relational databases typically starts with the definition of a conceptual schema which describes the conceptual structures of the database in terms of a high-level data model. The design proceeds by mapping the conceptual schema into a relational representation which is then implemented using the features of a specific relational database management system [?, ?].

Over the past decade, several data models have been proposed for conceptual database design [?, ?, ?]. Among them, the most popular is the entity-relationship model (ER model) [?], and the mapping of ER schemas into the relational model has received a considerable attention from the database community [?, ?, ?, ?, ?, ?].

More recently, the NIAM model [?, ?] has been pointed out as an alternative approach to conceptual database design. NIAM is a very powerful data model [?] which provides a fact-oriented approach to data modelling (also called object-role modelling). The design of relational databases, based on the NIAM model, was first addressed in [?] and is extensively discussed in [?].

However, the mapping of both, ER and NIAM schemas, into the relational model has been usually described in terms of a basic “canonical” model, which means that most of the integrity constraints specified at the conceptual level are not fully captured in the final relational representation. These constraints have then to be handled at the implementation level and coded in the application programs, which has been a major cause for frequent problems in many database applications.

The recent SQL standard proposal [?, ?] has provided a new framework for discussing the design of relational databases. It introduces a number of facilities which are able to capture most of the integrity constraints specified in a conceptual schema [?, ?, ?, ?]. These facilities include a more general form of the referential integrity constraint and assertion mechanisms for the definition of general application-oriented constraints.

In this paper, we discuss the general aspects involved in the mapping of NIAM schemas into SQL. We present a mapping procedure which handles the basic NIAM constructs and then discuss how to extend it to cope with the more complex ones. We also address the problem of optimizing the SQL representation of NIAM schemas by adapting the optimization strategy proposed in [?, ?] for ER schemas. This optimization strategy aims to reducing the number of referential constraint definitions in the final SQL representation. We point out that, besides reducing the number of referential constraint definitions, it also simplifies the mapping of NIAM inter-relationship constraints. We recall that the versions of the NIAM model and the SQL standard used in this paper are those described in [?] and [?] respectively.

The remaining sections of the paper are organized as follows. Section 2 summarizes the characteristics of the SQL features which are relevant to this work. Section 3 presents an overview of the NIAM constructs using a common

terminology framework. Section 4 presents the basic mapping procedure and discusses the mapping of aggregate objects and inter-relationship constraints. Section 5 addresses the optimization of the SQL representations. Finally, Section 6 presents the conclusions of the work.

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* [?, sec. 11.4, 11.6 to 11.9], which are of interest to the mappings discussed in this paper.

We first recall that the SQL standard disclosure [?, 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.

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 [?, 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 [?, p. 29].

Each referential constraint definition 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.

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.

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;
- at a table definition, when C specifies conditions that the rows of the table must satisfy;
- as a separate *assertion definition*, when C specifies conditions that the tables must satisfy.

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.

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.

3 Overview of NIAM

In this section we present an overview of the NIAM [?] constructs and illustrate them using a number of examples from a conference application. The constructs are introduced using a common terminology [?] which provides a basic framework for understanding their semantics.

3.1 Basic Constructs

3.1.1 Entities and Entity Classes

Entities are represented in NIAM in terms of *entity types*. An entity type defines a class of similar entities in the application and is represented graphically by an ellipse containing its name. An entity type may also be specialized in subtypes producing a type hierarchy in which an instance of any subtype is an instance of its supertype, as in Figure 1. In general, a type hierarchy is a directed acyclic graph rather than just a tree since a type may have many subtypes and a subtype may have many supertypes [?].

Figure 1: Subtype-supertype representation in NIAM

3.1.2 Relationships and Relationship classes

In NIAM, a relationship between entities is termed a *fact* and a *fact type* represents a class of relationships involving entities of the same type. Within a fact, each related entity plays a particular *role*, a role name expressing the semantics of the direction of the relationship from the viewpoint of an entity. Figure 2 shows that a fact type is represented by boxes which contain the role names, with line segments joining each role name to the relevant entity type; the entity types AUTHOR and PAPER participate in an AUTHORSHIP relationship, where AUTHOR may play the role WRITES and PAPER may play the role WRITTEN-BY.

Figure 2: Relationship representation in NIAM

Relationship Constraints. Within any given relationship, there are usually constraints which limit the possible ways in which entity instances may associate to form relationship instances. We identify two commonly occurring types of constraint which apply to an entity in a relationship [?, ?]: the *cardinality constraint* and the *participation constraint*.

The cardinality constraint defines the number of instances of an entity class which may associate with one instance of another entity class in a relationship. NIAM uses the *uniqueness constraint* to express the concept of cardinality constraint. A uniqueness constraint applied to a role restricts each entity instance in that role to be unique. An arrowed bar is placed over the role to which a uniqueness constraint applies, its absence indicating that no uniqueness constraint applies. Where no uniqueness constraint applies to a role separately (e.g. the binary many-to-many situation) NIAM applies it to a role combination. Figure 3 shows the diagrammatic representation of the cardinality constraint in NIAM.

The participation constraint specifies whether the instances of an entity class must participate (total participation) or only may participate (partial participation) in a relationship with instances of another entity class. NIAM uses the terms mandatory and optional role, so that a mandatory role is one which all instances of the relevant entity type must play. Figure 3 shows the diagrammatic representation of the total/partial participation constraint in NIAM, where a dot indicates that the participation of entity instances in a relationship is total (mandatory).

Figure 3: Cardinality and participation constraint representation in NIAM

3.1.3 Attributes and Attribute Classes

An *attribute* represents a descriptor or property of an entity instance, such as the age of a person, and an *attribute class* is a set of such entity descriptors, although the term attribute is often used to refer to both concepts.

In NIAM, attribute and attribute class are termed *label* and *label type* respectively. An association between an entity type and a label type is termed a *reference type*. Like entities in a relationship, attributes are also characterized by their cardinality and the way they are associated to entities, and NIAM is consistent in its approach over fact and reference types. Figure 4 shows how attributes are represented in NIAM, where a label type is represented by a broken ellipse, containing its name, with line segments joining it via role boxes to its related entity type, similar to the fact type.

Figure 4: Attribute representation in NIAM

3.1.4 Identifiers

In its simplest form, an *entity identifier* is a single attribute, each value of which uniquely identifies an instance of an entity class. An identifier may also be composite, in which case it may be made up of two or more attributes. Figure 5 shows the identifier representation in NIAM, where the external uniqueness constraint (also called *inter-fact-type uniqueness constraint* [?]) is used to represent a composite identifier.

Figure 5: Identifier representation in NIAM

A composite identifier may also be made of entities, for example as in Figure 6 where the PERSON, SUBJECT and PANEL entity types jointly identify the PANELLING entity type. These “externally” identified entity types usually come up when we use a binary relationship approach to model an n-ary relationship [?] or when we need to model a relationship which has attributes.

Figure 6: An externally identified entity type

3.2 Aggregate Object Classes

NIAM allows one to “objectify” a relationship, producing a *nested fact type* [?]. A nested fact type corresponds to an *aggregate object class* which is an entity class that is made up from a number of objects which describe the entity class in more detail. The term arises from the aggregation abstraction [?] which establishes the “part-of” relationship between the component objects and the aggregated object. We illustrate this construct in Figure 6, where the objectified REFEREE-PAPER relationship is nested in the compound object named REQUEST, indicated by the ellipse surrounding the relevant roles, and then associated to JUDGMENT.

3.3 Inter-relationship Constraints

NIAM provides additionally three types of inter-relationship constraint: *equality*, *exclusion* and *subset*, all of which are set constraints that may be specified on entity roles. These constraints, shown in Figure 8, are described below.

Equality constraint. This constraint restricts a set of instances of an entity type playing a role R1 to be equal to the set of instances of the (same) entity type playing another role R2. In Figure 8a, the set of CONFERENCE instances playing the role STARTING must be equal to the set of CONFERENCE instances playing the role ENDING. In other words, if a conference has a date, it must have both a start and an end date.

Exclusion constraint. This constraint restricts a set of instances of an entity type playing a role R1 to be disjoint from the set of instances of the (same) entity type playing another role R2. In Figure 8b, the set of PERSON instances playing the role CHAIRS in the relationship with CONFERENCE and the set of PERSON instances playing the role CHAIRS in the relationship with SESSION are mutually exclusive. That is, a person cannot be both a session and a conference chairperson.

Figure 7: NIAM objectified relationship (nested fact type)

Figure 8: Inter-relationship constraints representation in NIAM

Subset constraint. This constraint restricts the population of one or two roles R1 and R2 to be a subset of the population of one or two other roles R3 and R4, originating from similar object types. In Figure 8c, the population of the roles PRESENTS/PRESENTED-BY (joint instances of AUTHOR and PAPER) must be a subset of the population of the roles WRITES/WRITTEN-BY (also joint instances of AUTHOR and PAPER). In other words, the presenter of a paper must be an author of that paper.

4 Mapping into SQL

We discuss in this section the mapping of NIAM schemas into SQL. We start the discussion by outlining the mapping of the basic NIAM constructs, and then, with the help of examples, we elaborate on the mapping of aggregate objects and inter-relationship constraints.

4.1 Mapping of Basic Constructs

In order to simplify the discussion, we consider initially NIAM schemas which include only internally identified entity types and simple specializations. For such NIAM schemas, the mapping into SQL can be outlined as follows:

1. Each entity type E is mapped to a table definition E such that:
 - the attributes of E correspond to the label types related to E by reference types on which a uniqueness constraint applies to the role of E; each attribute must include a NOT NULL clause if the role of E is mandatory with respect to the corresponding reference type;
 - table E includes a unique constraint definition for each identifier of E. If E has only one identifier, then the corresponding unique constraint definition indicates the table primary key; otherwise, one of the unique constraint definitions indicates the primary key and the other ones alternate keys;
 - for each label type L related to E by a reference type on which a uniqueness constraint does not apply to the role of E, a new table L is defined such that its attributes are the attributes of the primary key K_E of table E and the label type L itself. Table L also includes a unique constraint definition indicating that the concatenation of all its attributes is its primary key. In addition, table L includes a referential constraint definition indicating that K_E in L is a foreign key referencing K_E in E; the delete and update rules are CASCADE, and the constraint mode is immediate;
 - if E is a subtype of G, then table E also imports the attributes of the primary key K_G declared in table G. It also has a unique constraint definition indicating that K_G is its primary key, if E has no primary key, or that K_G is one of its alternate keys, otherwise. Moreover, E includes a referential constraint definition indicating that K_G in E is a foreign key referencing K_G in G; the delete and update rules are CASCADE, and the constraint mode is immediate.
2. Each fact type F is mapped to a table definition F such that:
 - for each participating entity type P in F, table F imports the attributes of the primary key K_P defined for the corresponding table P, each of them including a SET NULL clause. Moreover, table F includes a referential constraint definition indicating that K_P in F is a foreign key referencing K_P in P; the delete and update rules are CASCADE, and the constraint mode is immediate. If the role of P in F is mandatory (total participation), then table F must also include a check constraint definition indicating that, for each row p in table P, there must be a row f in table F such that $p[K_P]=f[K_P]$ (such check constraint definition will involve a subquery to table P);
 - the uniqueness constraints applied to F are mapped into unique constraint definitions in table F involving the attributes of the imported primary keys. If only one uniqueness constraint applies to F, then the corresponding unique constraint definition indicates the table primary key; otherwise, one of the unique constraint definitions indicates the primary key and the other ones alternate keys.

As we can see from the above mapping procedure, the semantics of the referential constraint definition and the fact that the defined foreign keys do not admit null values (either because they are primary keys imported from another table, in the case of an entity subtype, or because they are declared as such, in the case of a fact type) guarantee that each row in a subtype table corresponds to exactly one row in the supertype table and that each row in a fact type table corresponds to a fact that relates existing entities. Furthermore, the delete and update rules have been specified as **CASCADE** in order to propagate such operations and enforce the correct semantics of the NIAM constructs. The update rule, however, may be omitted if updates that modify primary keys are not allowed [?].

Finally, as already pointed out in [?] for the mapping of ER schemas, we note that the basic mapping just outlined generates a primary key for all table definitions, which is essential for capturing the semantics of subtypes and fact types via referential constraint definitions. However, this property of the mapping depends on the following requirements of the NIAM schemas:

- each entity type has at least one internal identifier, defined directly in terms of its attributes or inherited from a supertype;
- the subtypes/supertypes defined induce a hierarchical organization for the entity types.

When a NIAM schema includes externally identified entity types, the above mapping procedure requires an additional step to cope with such constructs. Let us discuss this case with the help of an example. Consider a NIAM schema including the externally identified entity type shown in Figure 6. If the table definitions corresponding to the entity types PERSON, SUBJECT and PANEL include, respectively, the primary keys PERSON#, SUBJECT# and PANEL#, defined as attributes of type char(3), then the entity type PANELLING is mapped to the following SQL table definition:

```
create table PANELLING
(PERSON# char(3) not null,
SUBJECT# char(3) not null,
PANEL# char(3) not null,
primary key (PERSON#,SUBJECT#,PANEL#),
foreign key (PERSON#) references PERSON on delete cascade,
foreign key (SUBJECT#) references SUBJECT on delete cascade,
foreign key (PANEL#) references PANEL on delete cascade)
```

Note that this mapping is similar to the mapping of a fact type in the sense the resulting table imports the attributes of the primary keys of the tables which represent the identifying entity types.

In order to handle the case of more complex specializations, where the type hierarchy results in a graph rather than just a tree, we have to revise the mapping of subtypes into referential constraint definitions, or otherwise we may obtain table definitions which allow table instances that do not correctly capture the semantics of specializations. Briefly, the general idea is to replicate the primary key of each table *S*, which represents an entity type *S* that is a sink of the specialization graph, to each table *E* such that there is a path from *E* to *S* in the graph, and use such keys to generate referential constraint definitions. However, this requires supporting references to alternate keys, since these tables may already have a primary key derived directly from their corresponding entity types. A detailed discussion of the mapping of complex specializations is beyond the scope of this paper and can be found in [?].

4.2 Mapping of Aggregate Object Classes

As discussed in [?], the mapping of an aggregate object class, expressed by a NIAM nested fact type, is similar to the mapping of an ordinary fact type, except that the resulting table definition may have additional attributes, derived from reference types associated to that object class, and that it may be referenced by other table definitions representing entity types.

In order to illustrate this, let us discuss the mapping of the NIAM structure shown in Figure 5. Considering that the table definitions which represent the entity types REFEREE, PAPER and JUDGMENT include, respectively, the primary keys REFEREE#, PAPER# and JUDGMENT#, defined as attributes of type char(3), this structure is mapped to the following SQL table definitions:

```

create table REQUEST
  (REFEREE# char(3) not null,
   PAPER# char(3) not null,
   primary key (REFEREE#,PAPER#),
   foreign key (REFEREE#) references REFEREE on delete cascade,
   foreign key (PAPER#) references PAPER on delete cascade)

create table REQUEST-HAS
  (REFEREE# char(3) not null,
   PAPER# char(3) not null,
   JUDGMENT# char(3) not null,
   primary key (REFEREE#,PAPER#),
   alternate key (JUDGMENT#),
   foreign key (REFEREE#,PAPER#) references REQUEST on delete cascade,
   foreign key (JUDGMENT#) references JUDGMENT on delete cascade)

```

Note that table REQUEST-HAS, which represents the fact type that relates REQUEST and JUDGMENT, includes as foreign key the primary key of table REQUEST; that is, table REQUEST-HAS references table REQUEST exactly as if the nested fact type REQUEST were an entity type.

4.3 Mapping of Inter-relationship Constraints

Inter-relationship constraints are mapped into SQL using either general assertion definitions or check constraint definitions included in the definition of the tables that correspond to the fact or entity types involved. In what follows, we discuss the mapping of these constraints using the examples shown in Figure 8. For this discussion, we consider that entity and fact types are mapped according to the basic mapping procedure outlined in Section 4.1, and that each entity type has a single identifier which originates the primary key included in the definition of its corresponding table.

Equality constraint. The basic mapping of an equality constraint into SQL uses an assertion definition involving the foreign keys of the tables that correspond to the fact types on which the constraint holds. For the equality constraint shown in Figure 8a, this assertion definition is

```

create assertion EQC1
  check (select CONFERENCE# from CONFERENCE-STARTING in
         select CONFERENCE# from CONFERENCE-ENDING) and
         select CONFERENCE# from CONFERENCE-ENDING in
         select CONFERENCE# from CONFERENCE-STARTING)

```

where CONFERENCE-STARTING and CONFERENCE-ENDING are the tables which correspond to the fact types involved in the constraint. We note that both tables have CONFERENCE# as primary key and import the primary key of the table that represents the entity type DATE. Thus this assertion definition guarantees that the set of values of the primary key CONFERENCE# is the same in both tables, which means that if a conference has a date, it has both a start and an end date. This is exactly what the equality constraint in Figure 8a says.

Exclusion constraint. An exclusion constraint is also mapped to an assertion definition involving the foreign keys of the tables that correspond to the fact types on which the constraint holds. Considering that CONFERENCE-CHAired-BY and SESSION-CHAired-BY are the tables that represent the fact types involved in the exclusion constraint shown in Figure 8b, this constraint can be expressed by the following assertion definition:

```

create assertion EXC1
  check (not exists select PERSON# from CONFERENCE-CHAired-BY
         where PERSON# in (select PERSON# from SESSION-CHAired-BY))

```

This assertion definition involves the foreign key PERSON# of tables CONFERENCE-CHAired-BY and SESSION-CHAired-BY, and enforces that the values of these foreign keys are mutually exclusive; that is, a person can either chair a conference or a session, but not both, which is exactly what the exclusion constraint in Figure 8b says.

Subset constraint. The basic mapping of a subset constraint also uses an assertion definition similar to those discussed above for the equality and exclusion constraints. Alternatively, this type of constraint may also be expressed by a check constraint definition included in the definition of the table that represents the fact type from which the constraint originates. Considering the example shown in Figure 8c, the assertion definition which captures the subset constraint specified is

```
create assertion SSC1
  check (select PERSON#,PAPER# from PAPER-PRESENTED-BY in
        select PERSON#,PAPER# from PAPER-WRITTEN-BY)
```

where PAPER-PRESENTED-BY and PAPER-WRITTEN-BY are the tables which correspond to the fact types involved in the constraint. This assertion definition guarantees that any pair (PERSON#,PAPER#) in table PAPER-PRESENTED-BY must also appear in table PAPER-WRITTEN-BY; that is, the presenter of a paper must be one of its authors, which is exactly the meaning of the subset constraint in Figure 8c. An equivalent mapping for this subset constraint would include the following check constraint definition in the definition of table PAPER-PRESENTED-BY:

```
check ((PERSON#,PAPER#) match (select PERSON#,PAPER# from PAPER-WRITTEN-BY))
```

The mappings discussed above may be generalized as follows. Let F and G be two fact types relating entity types E_1, E_2, \dots, E_n , for $n \geq 2$. Considering that F and G are mapped into table definitions F and G according to the mapping outlined in Section 4.1, equality, exclusion and subset constraints specified on the roles played by E_1, E_2, \dots, E_n in F and G may be expressed, respectively, by the following SQL assertion definitions:

```
create assertion EQC
  check (select KE1, KE2, ..., KEm from F in select KE1, KE2, ..., KEm from G and
        select KE1, KE2, ..., KEm from G in select KE1, KE2, ..., KEm from F)
```

```
create assertion EXC
  check (not exists select KE1, KE2, ..., KEm from F
        where KE1, KE2, ..., KEm in (select KE1, KE2, ..., KEm from G))
```

```
create assertion SSC
  check (select KE1, KE2, ..., KEm from F in select KE1, KE2, ..., KEm from G)
```

where $K_{E_1}, K_{E_2}, \dots, K_{E_m}$, for $m \leq n$, are the primary keys imported from tables E_1, E_2, \dots, E_m and declared as foreign keys in the definition of tables F and G.

5 Optimization of SQL Representations

Given a NIAM schema, the mappings outlined so far produce a one-to-one SQL representation in the sense it contains a distinct table definition for each entity or fact type. However, one-to-one SQL representations 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 such SQL representations, we may adapt the heuristics usually applied to the mapping of ER schemas into the relational model [?, ?, ?] and collapse a fact type F into an entity type E and represent both as a single table definition, if E identifies F, i.e. a uniqueness constraint applies to the role of E in F. The same heuristics applies as well when G is a subtype of an entity type E, possibly restricted to the case where G has few attributes. The reader is referred to [?, ?] for a complete discussion of this topic.

Besides reducing the number of referential constraint definitions, this optimization strategy allows one to enforce a mandatory role (total participation) by just declaring the corresponding foreign key as not null, as well as to replace assertion definitions which express inter-relationship constraints by simpler check constraint definitions. In this section, we analyse, with the help of examples, the impact of this type of optimization on the mapping of NIAM schemas into SQL.

We begin with an example that illustrates that the most common form of optimization requires the use of the SET NULL delete rule. Consider the fact type, shown in Figure 8c, which relates the entity type AUTHOR with

role PRESENTS to the entity type PAPER with role PRESENTED-BY. If we apply the mapping procedure outlined in Section 4.1 to this fact type, we obtain a separate table definition which includes two referential constraint definitions. However, since now a uniqueness constraint applies to the role PRESENTED-BY, we may instead collapse this fact type on the entity type PAPER and represent both by the same table PAPER, which would be defined as follows:

```
create table PAPER
(PAPER# char(3) not null,
  ⋮ <other PAPER attribute definitions>
PERSON# char(3),
primary key (PAPER#),
foreign key (PERSON#) references AUTHOR on delete set null)
```

Note that the delete rule of the referential constraint definition specifies SET NULL. This is required because a row in table PAPER now represents a paper p and a fact that relates p to an author a , if the value of the foreign key PERSON# is not null. Thus, to propagate the deletion of a from table AUTHOR, the corresponding value of PERSON# in table PAPER must be set to null, which is exactly what the SET NULL delete rule does.

This same optimization strategy may also be applied when the role of an nested fact type identifies a fact type in which it participates. For example, in Figure 7, the fact type which relates REQUEST and JUDGMENT may be collapsed into REQUEST and represented by the same table REQUEST as follows:

```
create table REQUEST
(REFEREE# char(3) not null,
PAPER# char(3) not null,
JUDGMENT# char(3),
primary key (REFEREE#,PAPER#),
alternate key (JUDGMENT#),
foreign key (REFEREE#) references REFEREE on delete cascade,
foreign key (PAPER#) references PAPER on delete cascade,
foreign key (JUDGMENT#) references JUDGMENT on update cascade)
```

Suppose now that in the example of Figure 8c the role PRESENTED-BY of PAPER is mandatory. According to the basic mapping procedure, this situation requires a check constraint definition to enforce that, for every PAPER row, there must exist at least one associated row in the table that represents the corresponding fact type. In the optimized mapping, the mandatory role of PAPER can be easily captured by just declaring the foreign key PERSON# in table PAPER defined above as not null, and omitting the delete rule in the associated referential constraint definition in order to block deletions from table AUTHOR.

The optimization strategy discussed above also simplifies the mapping of inter-relationship constraints. To illustrate this, let us suppose that in the example of Figure 8c the uniqueness constraint applies only to the role WRITTEN-BY and not to the role combination WRITES/WRITTEN-BY, i.e. a paper may have just one author. Now both fact types may be collapsed on the entity type PAPER and represented in table PAPER defined below which includes a check constraint definition to enforce the subset constraint.

```
create table PAPER
(PAPER# char(3) not null,
  ⋮ <other PAPER attribute definitions>
PRESENTED-BY-PERSON# char(3),
WRITTEN-BY-PERSON# char(3),
primary key (PAPER#),
foreign key (PRESENTED-BY-PERSON#) references AUTHOR on delete set null,
foreign key (WRITTEN-BY-PERSON#) references AUTHOR on delete set null,
check (PRESENTED-BY-PERSON# is null or
PRESENTED-BY-PERSON# = WRITTEN-BY-PERSON#))
```

The check constraint definition above guarantees that, if a paper has a presenter, this must be its author, which is exactly what the new subset constraint says. We note this type of constraint is much simpler and cheaper to

enforce than the equivalent assertion definition that would result from the mapping discussed in Section 4.3.

The optimization discussed above for the subset constraint may be analogously applied to the equality and exclusion constraints. For example, in Figure 8a, if we collapse the fact types into CONFERENCE and represent them in table CONFERENCE, the subset constraint specified may be expressed by the following check constraint definition included in the definition of this table:

```
check ((STARTING-DATE is null and ENDING-DATE is null) or
       (STARTING-DATE is not null and ENDING-DATE is not null))
```

The reader is referred to [LCCR93] for a more detailed discussion of the optimization of such constraints.

6 Conclusions

We discussed in this paper the general aspects involved in the mapping of NIAM schemas into SQL. We presented a basic mapping procedure which handles the model basic constructs and then discussed how to map the more complex ones. We also addressed the problem of optimizing the SQL representation of a NIAM schema by applying an optimization strategy which reduces the number of referential constraint definitions in the final SQL schema. We pointed out that this optimization strategy also simplifies the mapping of inter-relationship constraints.

Finally, following [?], we point out that, although the SQL check constraint and assertion definitions are capable to fully capture inter-relationship constraints, they are generally very expensive to enforce, so that language designers and implementors should devise new mechanisms that capture these set-oriented constraints in a more efficient way and whose implementation hides some details that the optimizations discussed in this paper make visible to users.

References

- [1] ANSI X3H2-90-264 / ISO IEC JTC1 SC21 WG3. “(ISO working draft) Database Language SQL2”, J. Melton (ed.), July 1990.
- [2] Batini, C., Ceri, S. and Navathe, S. *Conceptual Database Design: An Entity-Relationship Approach*, Benjamin Cummings, 1992.
- [3] Casanova, M.A., Tucherman, L. and Laender, A.H.F. “Algorithms for designing and maintaining optimized relational representations of entity-relationship schemas”, *Proceedings of the 9th International Conference on Entity-Relationship Approach*, Lausanne, Switzerland, Oct. 1990.
- [4] Casanova, M.A., Tucherman, L. and Laender, A.H.F. “On the design and maintainance of optimized relational representations of entity-relationship schemas”, *Data and Knowledge Engineering 11*, (1993).
- [5] Casanova, M.A., Carvalho, A.P., Ridolfi, L.F.G.G.M. and Laender, A.H.F. “An analysis of table constraints in SQL2 based on the entity-relationship model”, *Proceedings of the 10th International Conference on Entity-Relationship Approach*, San Mateo, California, Oct. 1991.
- [6] Chen, P.P. “The entity-relationship model: toward a unified view of data”, *ACM Transactions on Database Systems 1*, 1 (March 1976).
- [7] Elmasri, R. and Navathe, S. *Fundamentals of Database Systems*, Benjamin Cummings, 1989.
- [8] Hull, R. and King, R. “Semantic database modelling: survey applications, and research issues”, *ACM Computing Surveys 19*, 3 (Sept. 1987).
- [9] ISO/TC97/SC5. “Concepts and terminology for the conceptual schema and the information base”, J.J. Van Griethuysen (ed.), ISO/TC97/SC5-N 695, March 1982.
- [10] Joint Technical Committee ISO/IEC JCT1. “International Standard ISO/IEC 9075:199x(E) - Database Language SQL”, April 1991.
- [11] Laender, A.H.F., Casanova, M.A., Carvalho, A.P. and Ridolfi, L.F.G.G.M. “An analysis of SQL integrity constraints from an entity-relationship model perspective”. Technical Report RT 018/93, DCC/UFMG, 1993 (submitted for publication).

- [12] Laender, A.H.L. and Flynn, D.J. "A semantic comparison of the modelling capabilities of the ER and NIAM models", *Proceedings of the 12th International Conference on Entity-Relationship Approach*, Arlington, Texas, Dec. 1993.
- [13] Leung, C.M.R. and Nijssen, G.M. "Relational database design using the NIAM conceptual schema", *Information Systems 13*, 2 (1988).
- [14] Markowitz, V.M. and Shoshani, A. "On the correctness of representing extended entity-relationship structures in the relational model", *Proc. ACM SIGMOD International Conference on the Management of Data*, Portland, Oregon, June 1989.
- [15] McCormack, J.I., Halpin, T.A. and Ritson, P.R. "Automated mapping of conceptual schemas to relational schemas", in C. Rolland et al. (eds.), *Advanced Information Systems Engineering*, Springer-Verlag, 1993.
- [16] Navathe, S.B. "Evolution of data modeling for databases", *Communications of ACM 35*, 9 (September 1992).
- [17] Nijssen, G.M. and Halpin, T.A. *Conceptual Schema and Relational Database Design: A Fact Oriented Approach*, Prentice-Hall, 1989.
- [18] Peckham, J. and Maryanski, F. "Semantic data models", *ACM Computing Surveys 20*, 3 (September 1988).
- [19] Ritson, P.R. and Halpin, T.A. "Mapping integrity constraints to a relational schema", *Proceedings of the 4th Australian Conference on Information Systems*, Brisbane, Australia, Sept. 1993.
- [20] Smith, J.M. and Smith, D.C. "Database abstractions: aggregation and generalization", *ACM Transactions on Database Systems 2*, 2 (June 1977).
- [21] Teorey, T.J. *Database Modeling and Design: The Entity-Relationship Approach*, Morgan Kaufmann, 1990.
- [22] Teorey, T.J., Yang, D. and Fry, J.P. "A logical design methodology for relational databases using the extended entity-relationship model", *ACM Computing Surveys 18*, 2 (June 1986).
- [23] Verheijen, G. and Van Bekkum, J. "NIAM: An information analysis method", in T.W. Olle et al. (eds.), *Information Systems Design Methodologies: A Comparative Review*, North-Holland (1982).