

ON THE RELATIONAL REPRESENTATION OF COMPLEX  
SPECIALIZATION STRUCTURES<sup>†</sup>ALTIGRAN S. DA SILVA<sup>1</sup>, ALBERTO H.F. LAENDER<sup>1</sup>, and MARCO A. CASANOVA<sup>2</sup><sup>1</sup>Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Av. Antônio Carlos 6627  
31270-010, Belo Horizonte MG, Brasil<sup>2</sup>Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rua Marquês de São Vicente  
255 22453-900, Rio de Janeiro RJ, Brasil*(Received 24 April 1999; in final revised form 16 May 2000)*

**Abstract** — The mapping of entity-relationship schemas (ER schemas) that contain complex specialization structures into the relational model requires the use of specific strategies to avoid inconsistent states in the final relational database. In this paper, we show that for this mapping to be correct it is required to enforce a special kind of integrity constraint, the *key pairing constraint (KPC)*. We present a mapping strategy that use simple inclusion dependencies to enforce KPC and show that this strategy can be used to correctly map specialization structures that are more general than the simple specialization structures considered by previous strategies.  
©2000 Elsevier Science Ltd. All rights reserved

*Key words:* ER Model, Relational Database Design, Specialization Structures, Integrity Constraints

## 1. INTRODUCTION

One of the classical problems in databases is the logical representation of entity-relationship schemas (ER schemas) [10] in terms of the relational model [3, 13, 28]. The interest in this particular problem is due to the specific features of each model, i.e., the ER model is adequate to directly model the real world, whereas the relational model is supported by a wide range of commercial DBMSs.

Over the last two decades, many methods have been proposed to generate relational schemas from ER schemas [8, 9, 12, 19, 21, 28, 29]. This process is called logical database design or simply *logical design* [3, 13]. Some of these methods incorporate integrity constraints in the resulting relational schema to enforce the semantic properties of the ER model [8, 21]. In particular, the method presented in [8] generates optimized relational schemas where the number of inclusion dependencies between relations is reduced. These methods have also served as the basis for a number of automated database design tools [9, 17, 16, 22].

A particular problem related to the logical design of relational databases is the representation of specialization structures, i.e., the mapping of semantic structures denoting is-a relationships into relational model constructs. Although this is an important problem, it has been almost neglected in the literature and, in general, the proposed mapping strategies only address simple cases [3, 13, 28, 29].

One of the first works to address the mapping of complex specialization structures, i.e., specialization structures which are not pure hierarchies, is reported in [20]. In that paper, the authors present examples of ER schemas containing complex specialization structures and outlines a strategy for correctly representing them in terms of relational model constructs as specified by the SQL standard [23]. In this paper, we present a generalization of the mapping strategy outlined in [20] and argument its correctness based on results presented in [11]. We also introduce a special type of key constraint, called *key pairing constraint (KPC)*, which must be enforced to guarantee the correctness of the mapping. Moreover, we define a structural constraint that characterizes the specialization structures for which this generalized strategy can be implemented using usual relational constraints.

---

<sup>†</sup>Recommended by Maurizio Lenzerini

It is important to notice that, although in this paper we discuss this mapping strategy based on the framework of the ER model, it equally applies to other semantic data models [24] and to object-oriented modeling techniques such as OMT [4, 25], OOAD [5], and OSM [14] which have very similar semantic structures.

It should also be pointed out that the study of mapping strategies for the correct relational representation of specialization structures becomes even more important nowadays when the first commercial versions of the so-called object-relational DBMSs [27] are appearing in the market. As the facilities provided by these systems for handling specialization structures are still very limited, new strategies for the representation of such structures can be very useful not only for DBMS implementors but also for database designers who are faced with situations not fully supported by them.

The paper is organized as follows. Section 2 discusses related work. Section 3 presents the terminology and the notation used throughout the paper. Section 4 revises the commonly adopted strategy to map simple specialization structures into relational constructs. Section 5 discusses the problem of mappings complex specialization structures and introduces the concept of *key pairing constraint*. Section 6 presents a strategy that enforces the key pairing constraint by means of simple inclusion dependencies. This section also defines a structural constraint that characterizes the specialization structures for which this strategy can be applied. Finally, Section 7 concludes the paper.

## 2. RELATED WORK

Over the last two decades, many methods have been described in the literature that address the problem of generating relational schemas from ER schemas. Despite this, there are relatively few works where the mapping of specialization structures is deeply discussed. In widely adopted text books, such as [3, 13, 28], the authors usually focus the discussion on simple cases (mostly non-constrained hierarchies) where only the structural part of the mappings (i.e., the definition of the relation schemes) is considered. For instance, in [13], the authors present four different strategies for representing specialization structures using relational concepts but do not discuss how to enforce the semantics of such structures by means of appropriate integrity constraints. The mapping approach described in [3] involves a high-level logical design phase in which specialization structures are removed and modeled using just the basic ER model constructs (i.e., entities and relationships), avoiding the problem of directly representing such structures in terms of the relational model.

A very deep discussion on the relational representation of ER schemas is presented in [21]. The method described in that paper is particularly accurate in dealing with relationships constraints (e.g., cardinality constraints [13]), a matter poorly addressed by most traditional design methods. However, regarding the representation of specialization structures, the authors assume in their (extended) ER model some strong hypothesis that simplify the is-a relationships that can appear in the ER schemas and therefore the relational mapping. For example, their model does not allow specialization structures whose components (entity sets) have more than one source thus avoiding the mapping problems discussed in Section 5 of this paper.

In fact, the mapping of ER schemas containing more complex specialization structures, such as those constrained with respect to totality and disjointness and those that are not pure hierarchies, has received very little attention in the literature. Two papers that address distinct aspects of this problem are [18] and [20].

In [18], the authors discuss the mapping of specialization structures constrained with respect to totality and disjointness, but that are pure hierarchies. The paper characterizes the semantics of such structures and presents a number of techniques to map them into SQL standard constructs (including check clauses and trigger mechanisms [23]).

The work presented in [20] also addresses some aspects of such constrained specialization structures (which are termed *generalizations*) as part of an analysis of SQL integrity constraints. In addition, the paper raises the problem of mapping complex specialization structures (i.e., those that are not pure hierarchies) by presenting examples of ER schemas containing such structures and outlining a strategy for correctly representing them in terms of SQL standard constructs.

In the present paper, we generalize such a strategy, argument its correctness, and characterize, by means of a structural constraint, the specialization structures for which this strategy can be implemented by using usual relational constraints.

Specialization structures also occur in the context of object oriented modeling through the mechanism of inheritance (as discussed in [26]). Discussions on the relational representation of such a mechanism appear in [1], [2], and [4] but again with little attention to more complex cases (e.g., multiple inheritance, class partition, etc.).

### 3. TERMINOLOGY AND NOTATION

#### 3.1. Entity-Relationship Schemas

An *Entity-Relationship schema* (ER schema) is a pair  $S_E = \langle \mathcal{E}, \mathfrak{R} \rangle$ , where  $\mathcal{E}$  is a set of entity schemes and  $\mathfrak{R}$  is a set of relationship schemes.

An *entity scheme* has a name, a list of attributes with their respective domains and, optionally, a list of keys. A *key* is a subset of the entity scheme attributes whose values uniquely identify the instances of the corresponding entity set. By convention, we assume that the first key in the list is considered as the primary key. The other ones, if any, are called alternate keys. At least one of the attributes of the entity scheme must be a *discriminating attribute* [11, 12], an attribute whose value is never null. We use this feature to indicate the existence of an entity (instance). To enforce the principle of entity integrity [13], primary key attributes must always be discriminating attributes.

A *relationship scheme* has a name, a list of participant entity schemes and, optionally, a list of attributes with their respective domains. The instances of a relationship scheme are elements from the Cartesian product of the sets of instances of their participant entity schemes. In this paper, for the sake of simplicity, we will only consider ER schemas where  $\mathfrak{R} = \emptyset$ , i.e., ER schemas that do not include relationship schemes. Nevertheless, the entire discussion remains valid for ER schemas that include relationship schemes [11].

An entity scheme may be *specialized* into one or more entity schemes and may also specialize one or more entity schemes. We assume that if an entity scheme  $E$  specializes an entity scheme  $G$ , then every instance of  $E$  is also a instance of  $G$ . We also say that  $G$  is a *generic* of  $E$  and  $E$  is a *specialization* of  $G$ .

If a given entity scheme does not specialize any other scheme, the definition of a primary key for this scheme is mandatory. We say that an entity scheme *inherits* the attributes and keys of its generics and we distinguish these attributes and keys, called *inherited* attributes and keys, from those defined in the scheme itself, called *native* attributes and keys. We note that, for the sake of simplicity and without loss of generality, the examples presented in this paper only include primary keys.

The semantics of specialization structures is defined below.

**Definition 1** Let  $S_E = \langle \mathcal{E}, \mathfrak{R} \rangle$  be an ER schema where  $\mathfrak{R} = \emptyset$ . A consistent state for  $S_E$  is a triple  $(\mathcal{E}, \mathcal{I}, \mathcal{V})$  for which the following properties are valid:

1.  $\mathcal{E}$  is a set of *entity instances*;
2.  $\mathcal{I}$  maps each entity scheme  $E$  of  $S_E$  into a set  $\mathcal{I}(E) \subseteq \mathcal{E}$ , the set of *instances* of  $E$  defined by  $\mathcal{I}$ ;
3. If  $E$  is an entity scheme that specializes another entity scheme  $G$  in  $S_E$  then  $\mathcal{I}(E) \subseteq \mathcal{I}(G)$ ;
4. Let  $E$  be an entity scheme of  $S_E$ .  $\mathcal{V}$  maps each entity instance  $e \in \mathcal{I}(E)$  into a value  $\mathcal{V}(e, E.A)$  such that:
  - 4.1 If  $A$  is a native attribute of  $e$ , then  $\mathcal{V}(e, E.A)$  belongs to the domain of  $A$ ; and
  - 4.2 If  $A$  is an attribute of  $E$  inherited from  $G$ , then  $\mathcal{V}(e, E.A) = \mathcal{V}(e, G.A)$ ;
5. Let  $E$  be an entity scheme of  $S_E$  and  $K$  be a key of  $E$  (native or inherited). For every  $e_1$  and  $e_2 \in \mathcal{I}(E)$ , if  $\mathcal{V}(e_1, E.K) = \mathcal{V}(e_2, E.K)$  then  $e_1 = e_2$ .

By property 4, we then have that, if  $E$  specializes  $G$  and  $A$  is an attribute of  $G$ , then  $\mathcal{V}(e, E.A)$  is defined for every  $e \in \mathcal{I}(E)$ . Indeed, given  $e \in \mathcal{I}(E)$ , we know that  $e \in \mathcal{I}(G)$ , since  $E$  specializes  $G$ , and hence  $\mathcal{V}(e, F.A)$  is defined, since  $A$  is an attribute of  $G$ .

To simplify the notation, we will write  $\mathcal{V}(e, E.A)$  instead of  $\mathcal{V}(e, E.A_1), \dots, \mathcal{V}(e, E.A_n)$ , where  $A = \{A_1, \dots, A_n\}$  is the set of attributes of an instance  $e$ .

We define a *specialization graph* as a directed graph  $g = (V_g, A_g)$ , such that  $V_g$  is a set of entity scheme names in a ER schema  $S_E$  and  $A_g$  is a set of arcs such that an arc  $(E, G)$  is in  $A_g$  if, and only if,  $E$  is a specialization of  $G$  in  $S_E$ . We say that a vertex in  $V_g$  is a *root* of  $g$  if and only if there is no arc in  $A_g$  that starts from it. If we have a path in  $g$  from a vertex  $V$  to a vertex  $U$ , we say that  $V$  is a *descendant* of  $U$  and that  $U$  is an *ancestor* of  $V$ . We also say that a vertex is an ancestor and a descendent of itself.

We also say that a specialization structure is *complex* when its corresponding specialization graph has more than one distinct path between any two vertices. Otherwise, we say that the specialization structure is *simple*.

In this paper, we will use the following notation to define entity schemes in an ER schema:

```

define entity E
  attributes A1 D1 [ not null ], ..., An Dn [ not null ],
  [ key K1, ..., key Kp ]
  [ specialization of G1, ..., Gq ]

```

where  $E$  is the entity scheme name,  $A_1, \dots, A_n$  ( $n \geq 1$ ) are attribute names,  $D_1, \dots, D_n$  ( $n \geq 1$ ) are valid domain names (e.g., `integer`, `char`, etc.),  $K_1, \dots, K_p$  ( $p \geq 0$ ) are lists of attributes from  $\{A_1, \dots, A_n\}$ , and  $G_1, \dots, G_q$  ( $q \geq 0$ ) are entity scheme names. The clauses between square brackets are optional. However, at least one of the attributes  $A_1, \dots, A_n$  must be specified as “not null” and one of the keys must be composed of attributes specified as “not null”. The first of these keys is assumed to be the primary key.

### 3.2. Relational Schemas

A *relational schema* is a pair  $S_R = (\mathfrak{R}, \mathfrak{C})$ , where  $\mathfrak{R}$  is a set of relation schemes and  $\mathfrak{C}$  is a set of integrity constraints. We assume that  $\mathfrak{C}$  is only composed of inclusion dependencies [7, 6]. A *state*  $\sigma$  for  $S_R$  assigns to each relation scheme  $R$  in  $\mathfrak{R}$  a relation instance in the usual way. We will refer to the instance of a relation scheme  $R$  as *relation*  $R$ .

A *relation scheme* has a name, an attribute list (each attribute has a respective domain), a *primary key* and, optionally, one or more *alternate keys*. The list of attributes that form a primary key must be composed by attributes that cannot assume null values, whereas the list of attributes used to define alternate keys may include attributes that can assume null values. A null value is represented here by the symbol  $\lambda$ . In this work, we will use a syntax close to the SQL standard described in [23] to define relation schemes.

An *inclusion dependency (IND)* is an expression of the form  $T_1[X_1] \subseteq T_2[X_2]$  where, for  $j = 1, 2$ ,  $T_j$  is a relation scheme in  $\mathfrak{R}$  and  $X_j$  is a sequence of distinct attribute names of  $T_j$  such that  $X_1$  and  $X_2$  have the same number of attributes and that the  $k$ -th attribute of  $X_1$  and the  $k$ -th attribute of  $X_2$  have the same domain. An inclusion dependency is satisfied by the relations  $T_1$  and  $T_2$  of the same state  $\sigma$  iff, for each tuple  $t$  of  $T_1$ , there is a tuple  $u$  of  $T_2$  such that  $t[X_1] = u[X_2]$ .

### 3.3. Relational Representations of ER Schemas

A relational schema  $S_R$  is a *relational representation* of an ER schema  $S_E$  if the definition of  $S_R$  is such that every consistent state  $\sigma_E$  of  $S_E$  may be represented by a consistent state  $\sigma_R$  of  $S_R$  and every consistent state  $\sigma_R$  represents a consistent state of  $S_E$  [11, 12].

One method to obtain a relational representation of an ER schema is to generate a relation scheme (with appropriate attributes) to represent each entity and relationship scheme, using INDs to capture the semantics of the ER model. A relational representation obtained this way is usually called an *one-to-one relational representation*. Despite its simplicity, this kind of representation may require a large number of INDs, which can be expensive to check for violation [8].

**Algorithm 1**BeginInput: A specialization graph  $g$  of an ER schema  $S_E$ ;Output: A relational schema  $S_R = \langle \mathcal{R}, \mathcal{C} \rangle$ ;For each vertex of  $g$  corresponding to an entity scheme  $E$ , generate a relation scheme  $T_E$  as follows:Map each attribute of  $E$  directly to an attribute of  $T_E$ ;For each arc  $(E, G)$  of  $g$ If the entity scheme  $G$  has a primary keyThen let  $L$  be this primary keyElse let  $L$  be the primary key of its nearest ancestor which has one;

End-If;

Map each attribute of  $L$  to an attribute of  $T_E$ ;Create an alternate key of  $T_E$  corresponding to  $L$ ;

End-For;

If  $E$  has a primary key  $K$ Then map  $K$  to the primary key of  $T_E$ Else transform one of the alternate keys of  $T_E$  into its primary key;

End-If;

Add  $T_E$  to  $\mathcal{R}$ 

End-For;

For each arc  $(E, G)$  of  $g$ Let  $T_E$  and  $T_G$  be the relation schemes that map, respectively, the entity scheme  $E$  and  $G$ ;Let  $K$  be the primary key of  $G$ ;Add the inclusion dependency  $T_E[K] \subseteq T_G[K]$  to  $\mathcal{C}$ ;

End-For;

End.

Fig. 1: Algorithm for Mapping Simple Specialization Structures

To reduce the number of INDs of a relational representation, a fairly common heuristic is to collapse into an entity scheme  $E$  the relationship schemes that are functional on a role that  $E$  plays, as well as the entity schemes that specialize  $E$ , and to represent them as a single relation scheme. We call a relational representation obtained using this strategy an *optimized relational representation* [8, 11, 12].

Although we have addressed both representations in our previous works [8, 11, 12], in this paper we focus our discussion on one-to-one relational representations. The reasons are twofold: first, the adequate representation of complex specialization structures requires additional integrity constraints (such as *null dependencies* [12]) not available in commercial relational DBMSs and, second, one-to-one relational representations are structurally nearer to their corresponding ER schemas than optimized relational representations, which makes them more suitable for the discussion presented in this paper.

#### 4. MAPPING OF SIMPLE SPECIALIZATION STRUCTURES

Algorithm 1 (Figure 1) describes a commonly used strategy [3, 13, 28] to map specialization structures into a set of relation schemes. However, as we will see later, this algorithm can only be used with simple specialization structures or, otherwise, the resulting relational schema will not satisfy the semantics of such structures.

Note that this algorithm assumes that an entity scheme which has no primary key must necessarily specialize another entity scheme. Thus, the primary key of its corresponding relation scheme is “inherited” from one of its ancestors. Also note that we use inclusion dependencies, instead of referential integrity constraints [13], since in the context of this work the two types of integrity constraint are equivalent [11]. These two observations will remain valid for the rest of this paper.

In this mapping, inclusion dependencies are used to enforce property 3 of Definition 1, which

```

define entity A
  attributes KA char(4) not null,
             NA char(4) not null
  key       KA

define entity B
  attributes NB char(4) not null
  specialization of A

```

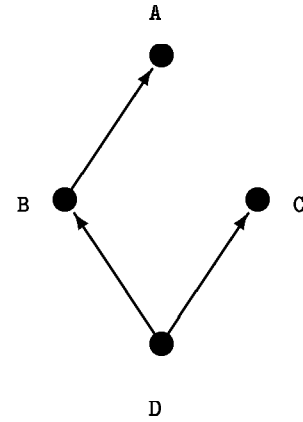
```

define entity C
  attributes KC char(4) not null,
             NC char(4) not null
  key       KC

define entity D
  attributes ND char(4) not null
  specialization of B,C

```

(a) Schema Definition



(b) Graph

```

CREATE TABLE TA
( KA CHAR(4) NOT NULL,
  NA CHAR(4) NOT NULL,
  PRIMARY KEY (KA)

CREATE TABLE TC
( KC CHAR(4) NOT NULL,
  NC CHAR(4) NOT NULL,
  PRIMARY KEY (KC)

```

```

CREATE TABLE TB
( KA CHAR(4) NOT NULL,
  NB CHAR(4) NOT NULL,
  PRIMARY KEY (KA)

CREATE TABLE TD
( KC CHAR(4) NOT NULL,
  KA CHAR(4) NOT NULL,
  ND CHAR(4) NOT NULL,
  PRIMARY KEY (KA),
  UNIQUE (KC)

```

```

TB[KA] ⊆ TA[KA]
TD[KA] ⊆ TB[KA]
TD[KC] ⊆ TC[KC]

```

(c) Relation Schema SS

Fig. 2: ER Schema SS

states that if an entity scheme  $E$  specializes an entity scheme  $G$ , then for every instance  $e \in \mathcal{I}(E)$  there must be an instance  $g \in \mathcal{I}(G)$  such that  $e=g$ .

From this property we can state the following corollary:

**Corollary 1** *Let  $G$  and  $E$  be entity schemes such that  $E$  specializes  $G$ . Let  $K_G$  and  $K_E$  be the set of keys (native and inherited) of  $G$  and  $E$ , respectively, and  $K = K_G \cap K_E$ . For every instance  $e \in \mathcal{I}(E)$  there must be an instance  $g \in \mathcal{I}(G)$  such that  $\mathcal{V}(g, G.K) = \mathcal{V}(e, E.K)$ .*

This means that if we use a relation scheme  $T_G$  to represent  $G$  and a relation scheme  $T_E$  to represent  $E$ , we may guarantee property 3 of Definition 1, provided that the following conditions apply:

1.  $T_G$  includes all the attributes of  $K_G$  and  $T_E$  includes all the attributes of  $K_E$ ;
2. For any tuple  $u$  of relation  $T_E$  there must be a tuple  $t$  of relation  $T_G$  such that  $t[K] = u[K]$ , where  $K = K_G \cap K_E$ . This condition can be restated as:

$$T_E[K] \subseteq T_G[K] \quad (1)$$

As an example of a simple specialization structure, consider the ER schema  $SS$  defined in Figure 2(a), whose specialization graph is shown in Figure 2(b). Algorithm 1 will then generate a relation scheme for each entity scheme, resulting in the relational schema shown in Figure 2(c). For an ER schema  $S_E$  such as that shown in Figure 2(a), Algorithm 1 generates a relational schema that correctly represents the semantics of the specialization structure of  $S_E$ . Note that the inclusion dependencies generated enforce property 3 of Definition 1 based on Corollary 1.

```

define entity A
  attributes KA char(4) not null,
            NA char(4) not null
  key      KA

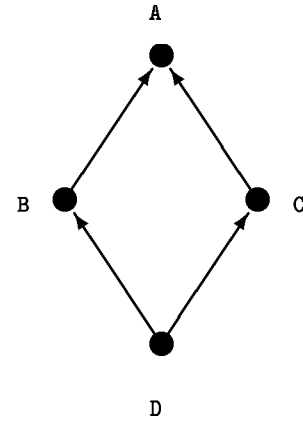
define entity B
  attributes NB char(4) not null
  specialization of A
    
```

```

define entity C
  attributes KC char(4) not null,
            NC char(4) not null
  key      KC
  specialization of A

define entity D
  attributes ND char(4) not null
  specialization of B,C
    
```

(a) Schema Definition



(b) Graph

```

CREATE TABLE TA
( KA CHAR(4) NOT NULL,
  NA CHAR(4) NOT NULL,
  PRIMARY KEY (KA)

CREATE TABLE TC
( KC CHAR(4) NOT NULL,
  KA CHAR(4) NOT NULL,
  NC CHAR(4) NOT NULL,
  PRIMARY KEY (KC),
  UNIQUE (KA)
    
```

```

CREATE TABLE TB
( KA CHAR(4) NOT NULL,
  NB CHAR(4) NOT NULL,
  PRIMARY KEY (KA)

CREATE TABLE TD
( KC CHAR(4) NOT NULL,
  KA CHAR(4) NOT NULL,
  ND CHAR(4) NOT NULL,
  PRIMARY KEY (KA),
  UNIQUE (KC)
    
```

```

TB[KA] ⊆ TA[KA]
TC[KA] ⊆ TA[KA]
TD[KA] ⊆ TB[KA]
TD[KC] ⊆ TC[KC]
    
```

(c) Relational Schema SD1

Fig. 3: ER Schema SD

## 5. MAPPING OF COMPLEX SPECIALIZATION STRUCTURES

In this section, we address the problem of correctly mapping complex specialization structures into relational model constructs. We begin by presenting some motivating examples and proceed by discussing the generalization of the mapping strategy (proposed in [20]) applied to them.

### 5.1. Motivating Examples

Consider the ER schema SD defined in Figure 3(a), which represents a complex specialization structure and whose specialization graph is shown in Figure 3(b). For this ER schema, Algorithm 1 will generate the relational schema shown in Figure 3(c). Figure 4 shows a state of this relational schema which is consistent with the defined integrity constraints, and yet it does not correctly reflect the semantics of the specialization structures of SD. Indeed, observe that:

1. Relation T<sub>A</sub> has two tuples that represent two entities of A, say, e<sub>1</sub> and e<sub>2</sub>. They indicate that the values of the primary key K<sub>A</sub> of A for e<sub>1</sub> and e<sub>2</sub> are, respectively, a1 and a2.
2. Relation T<sub>C</sub> has two tuples that represent these same entities through the values a1 and a2. They also indicate that the values of the key K<sub>C</sub> of C for e<sub>1</sub> and e<sub>2</sub> are c1 and c2, respectively.
3. Relation T<sub>D</sub> has two tuples that represent these same entities. However, they incorrectly indicate that the values of K<sub>C</sub> for e<sub>1</sub> and e<sub>2</sub> are c2 and c1, respectively.

Facts (2) and (3) can be concisely stated by

$$T_C \bowtie_{T_C.K_A=T_D.K_A} T_D \neq T_C \bowtie_{T_C.K_C=T_D.K_C} T_D$$

| $T_A$ | <table border="1"><tr><th><math>K_A</math></th><th><math>N_A</math></th></tr><tr><td><math>a1</math></td><td><math>x</math></td></tr><tr><td><math>a2</math></td><td><math>y</math></td></tr></table> | $K_A$ | $N_A$ | $a1$ | $x$ | $a2$ | $y$ |
|-------|---|-------|-------|------|-----|------|-----|
| $K_A$ | $N_A$   |       |       |      |     |      |     |
| $a1$  | $x$   |       |       |      |     |      |     |
| $a2$  | $y$   |       |       |      |     |      |     |

| $T_B$ | <table border="1"><tr><th><math>K_A</math></th><th><math>N_B</math></th></tr><tr><td><math>a1</math></td><td><math>u</math></td></tr><tr><td><math>a2</math></td><td><math>w</math></td></tr></table> | $K_A$ | $N_B$ | $a1$ | $u$ | $a2$ | $w$ |
|-------|---|-------|-------|------|-----|------|-----|
| $K_A$ | $N_B$   |       |       |      |     |      |     |
| $a1$  | $u$   |       |       |      |     |      |     |
| $a2$  | $w$   |       |       |      |     |      |     |

| $T_C$ | <table border="1"><tr><th><math>K_A</math></th><th><math>K_C</math></th><th><math>N_C</math></th></tr><tr><td><math>a1</math></td><td><math>c1</math></td><td><math>p</math></td></tr><tr><td><math>a2</math></td><td><math>c2</math></td><td><math>q</math></td></tr></table> | $K_A$ | $K_C$ | $N_C$ | $a1$ | $c1$ | $p$ | $a2$ | $c2$ | $q$ |
|-------|--|-------|-------|-------|------|------|-----|------|------|-----|
| $K_A$ | $K_C$  | $N_C$ |       |       |      |      |     |      |      |     |
| $a1$  | $c1$   | $p$   |       |       |      |      |     |      |      |     |
| $a2$  | $c2$   | $q$   |       |       |      |      |     |      |      |     |

| $T_D$ | <table border="1"><tr><th><math>K_A</math></th><th><math>K_C</math></th><th><math>N_D</math></th></tr><tr><td><math>a2</math></td><td><math>c1</math></td><td><math>f</math></td></tr><tr><td><math>a1</math></td><td><math>c2</math></td><td><math>g</math></td></tr></table> | $K_A$ | $K_C$ | $N_D$ | $a2$ | $c1$ | $f$ | $a1$ | $c2$ | $g$ |
|-------|--|-------|-------|-------|------|------|-----|------|------|-----|
| $K_A$ | $K_C$  | $N_D$ |       |       |      |      |     |      |      |     |
| $a2$  | $c1$   | $f$   |       |       |      |      |     |      |      |     |
| $a1$  | $c2$   | $g$   |       |       |      |      |     |      |      |     |

Fig. 4: Inconsistent State of SD1

```
CREATE TABLE T_A
(K_A CHAR(4) NOT NULL,
 N_A CHAR(4) NOT NULL,
 PRIMARY KEY (K_A))

CREATE TABLE T_C
(K_C CHAR(4) NOT NULL,
 K_A CHAR(4) NOT NULL,
 N_C CHAR(4) NOT NULL,
 PRIMARY KEY (K_C),
 UNIQUE (K_A))
```

```
CREATE TABLE T_B
(K_A CHAR(4) NOT NULL,
 N_B CHAR(4) NOT NULL,
 PRIMARY KEY (K_A))

CREATE TABLE T_D
(K_C CHAR(4) NOT NULL,
 K_A CHAR(4) NOT NULL,
 N_D CHAR(4) NOT NULL,
 PRIMARY KEY (K_A),
 UNIQUE K_C)
```

```
T_B[K_A] ⊆ T_A[K_A]
T_C[K_A] ⊆ T_A[K_A]
T_D[K_A] ⊆ T_B[K_A]
T_D[K_A, K_C] ⊆ T_C[K_A, K_C]
```

Fig. 5: Relational Schema SD2

when we consider the state of Figure 4. This means that the tuples from  $T_C$  and  $T_D$  that have the same values for the key  $K_A$  and those that have same values for the key  $K_C$  do not represent the same entity instances.

For the ER schema SD in Figure 3, a relational schema that correctly captures its semantics is shown in Figure 5. To generate this relational schema, the primary key  $K_A$  of A was “propagated” throughout the specialization structure and an inclusion dependency was generated for the pair of keys  $K_A$  and  $K_C$ , avoiding the problem found in the relational schema SD1. This inclusion dependency not only enforces property 3 of Definition 1, but also guarantees that

$$T_C \bowtie_{T_C.K_A=T_D.K_A} T_D = T_C \bowtie_{T_C.K_C=T_D.K_C} T_D \quad (2)$$

which means that the tuples in  $T_C$  and  $T_D$  that have the same values for the pair of keys  $K_A$  and  $K_C$  represent the same entity instances.

Consider now the ER schema DD and its graph described, respectively, in Figures 6(a) and 6(b). Using this same strategy to propagate the primary keys of the roots of the specialization graph, we will have the relational schema shown in Figure 7. In this example, we generated the inclusion dependencies

$$T_F[K_A, K_B] \subseteq T_D[K_A, K_B] \text{ and } T_G[K_A, K_B] \subseteq T_D[K_A, K_B], \quad (3)$$

containing both keys  $K_A$  and  $K_B$ , instead of simpler inclusion dependencies that would be generated if we had used Algorithm 1.

Indeed, these inclusion dependencies are essential to guarantee the correctness of the mapping. The argument is very similar to that presented for ER schema SD (Figure 3). Since F and G have  $K_A$  and  $K_B$  as inherited keys, it must be guaranteed that if a tuple in relation  $T_F$  and a tuple in relation  $T_G$  represent the same entity instance they must have the same values for the pair of keys  $K_A$  and  $K_B$  or, in other words,

$$T_F \bowtie_{T_F.K_A=T_G.K_A} T_G = T_F \bowtie_{T_F.K_B=T_G.K_B} T_G \quad (4)$$

It should be noted that the situation of vertices F and G in the ER schema DD is not the same as the one of vertices C and D in the ER schema SD. In schema SD, we have D specializing C while in schema DD the association between F and G is not so strong since property 3 of Definition 1 does not apply. Nevertheless, in both situations the entity schemes inherit keys from common ancestors. Also note that condition (4) is indirectly enforced by the inclusion dependencies generated due to the fact that F and G specialize the same vertex D, i.e., there is no need for explicitly generating integrity constraints between relation schemes  $T_F$  and  $T_G$  to enforce condition (4).

Indeed, conditions (2) and (4) are examples of a constraint, called *Key Pairing Constraint*, that will be discussed next.

```

define entity A
  attributes KA char(4) not null,
           NA char(4) not null
  key      KA

define entity B
  attributes KB char(4) not null,
           NB char(4) not null
  key      KB

define entity C
  attributes NC char(4) not null
  specialization of A

define entity D
  attributes ND char(4) not null
  specialization of A,B

define entity E
  attributes NE char(4) not null
  specialization of B

define entity F
  attributes NF char(4) not null
  specialization of C,D

define entity G
  attributes NG char(4) not null
  specialization of D,E
    
```

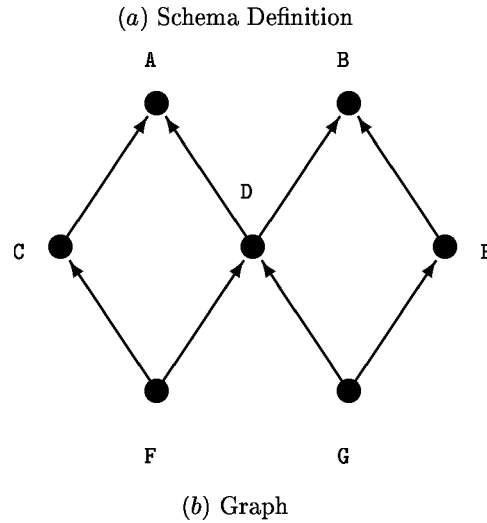


Fig. 6: ER Schema DD

### 5.2. Key Pairing Constraint

In order to generalize the mapping strategy presented above, let E and F be entity schemes in a specialization structure of an ER schema and let  $K_E$  and  $K_F$  be the set of keys (native and inherited) of E and F respectively (from now on, we denote the set of keys, native and inherited, of any entity scheme A by  $K_A$ ). In any consistent state of the ER schema we have the following property:

**Property 1** For every instance  $e \in \mathcal{I}(E) \cap \mathcal{I}(F)$ ,  $\mathcal{V}(e, E.K) = \mathcal{V}(e, F.K)$  for every key  $K \in K_E \cap K_F$ .

From Property 1 we can state the following corollary:

**Corollary 2** For any key  $K \in K_E \cap K_F$ ,  $\mathcal{V}(e, E.K) = \mathcal{V}(e, F.K)$  iff  $\mathcal{V}(e, E.L) = \mathcal{V}(e, F.L)$  for every key  $L \in K_E \cap K_F - \{K\}$ .

This means that if we use a relation scheme  $T_E$  to represent E and a relation scheme  $T_F$  to represent F, we guarantee Property 1, provided that the following conditions apply:

1.  $T_E$  includes all the attributes of  $K_E$  and  $T_F$  includes all the attributes of  $K_F$ ;

|  |  |   |
|--|--|---|
| CREATE TABLE T <sub>A</sub><br>( K <sub>A</sub> CHAR(4) NOT NULL,<br>N <sub>A</sub> CHAR(4) NOT NULL,<br>PRIMARY KEY (K <sub>A</sub> ))  | CREATE TABLE T <sub>B</sub><br>( K <sub>B</sub> CHAR(4) NOT NULL,<br>N <sub>B</sub> CHAR(4) NOT NULL,<br>PRIMARY KEY (K <sub>B</sub> ))  | CREATE TABLE T <sub>C</sub><br>( K <sub>A</sub> CHAR(4) NOT NULL,<br>N <sub>C</sub> CHAR(4) NOT NULL,<br>PRIMARY KEY (K <sub>A</sub> ))   |
| CREATE TABLE T <sub>D</sub><br>( K <sub>A</sub> CHAR(4) NOT NULL,<br>K <sub>B</sub> CHAR(4) NOT NULL,<br>N <sub>D</sub> CHAR(4) NOT NULL,<br>PRIMARY KEY (K <sub>A</sub> ),<br>UNIQUE (K <sub>B</sub> )) | CREATE TABLE T <sub>E</sub><br>( K <sub>B</sub> CHAR(4) NOT NULL,<br>N <sub>E</sub> CHAR(4) NOT NULL,<br>PRIMARY KEY (K <sub>B</sub> ))  | T <sub>C</sub> [K <sub>A</sub> ] ⊆ T <sub>A</sub> [K <sub>A</sub> ]<br>T <sub>D</sub> [K <sub>A</sub> ] ⊆ T <sub>A</sub> [K <sub>A</sub> ]<br>T <sub>D</sub> [K <sub>B</sub> ] ⊆ T <sub>B</sub> [K <sub>B</sub> ]<br>T <sub>E</sub> [K <sub>B</sub> ] ⊆ T <sub>B</sub> [K <sub>B</sub> ]<br>T <sub>F</sub> [K <sub>A</sub> ] ⊆ T <sub>C</sub> [K <sub>A</sub> ] |
| CREATE TABLE T <sub>F</sub><br>( K <sub>A</sub> CHAR(4) NOT NULL,<br>K <sub>B</sub> CHAR(4) NOT NULL,<br>N <sub>F</sub> CHAR(4) NOT NULL,<br>PRIMARY KEY (K <sub>A</sub> ),<br>UNIQUE (K <sub>B</sub> )) | CREATE TABLE T <sub>G</sub><br>( K <sub>A</sub> CHAR(4) NOT NULL,<br>K <sub>B</sub> CHAR(4) NOT NULL,<br>N <sub>G</sub> CHAR(4) NOT NULL,<br>PRIMARY KEY (K <sub>B</sub> ),<br>UNIQUE (K <sub>A</sub> )) | T <sub>F</sub> [K <sub>A</sub> , K <sub>B</sub> ] ⊆ T <sub>D</sub> [K <sub>A</sub> , K <sub>B</sub> ]<br>T <sub>G</sub> [K <sub>B</sub> ] ⊆ T <sub>E</sub> [K <sub>B</sub> ]<br>T <sub>G</sub> [K <sub>A</sub> , K <sub>B</sub> ] ⊆ T <sub>D</sub> [K <sub>A</sub> , K <sub>B</sub> ]   |

Fig. 7: Relational Schema DD

- For any tuple  $t$  of relation T<sub>E</sub> if there is a tuple  $u$  in relation T<sub>F</sub> such that  $t[K] = u[K]$  for a key  $K$  that corresponds to some key  $K \in K_E \cap K_F$  then  $t[L] = u[L]$  for every key  $L$  that corresponds to a key  $L \in K_E \cap K_F - \{K\}$ . This condition can be restated as:

$$T_E \underset{T_E.K_i=T_F.K_i}{\bowtie} T_F = T_E \underset{T_E.K_j=T_F.K_j}{\bowtie} T_F \quad (5)$$

for every  $K_i, K_j$  corresponding to distinct keys in  $K_E \cap K_F$ .

Note that condition (5) is the general expression of conditions (2) and (4). We will refer to condition (5) as the *Key Pairing Constraint* or *KPC* for short.

The main conclusion of our discussion so far is that the correct mapping of a specialization structure requires the enforcement of KPC for every pair of relation schemes that represent entity schemes sharing common inherited keys. However, enforcing KPC using current relational DBMSs mechanisms is not usually a simple task, since, in general, it requires the use of check rules or triggers [23]. This strategy has two major shortcomings: (1) coding this type of constraint is not trivial and (2) the cost of checking it may be prohibitive since it requires the join of the involved relations on the corresponding key pairs.

Note that, in the examples discussed in Section 5.1 (ER schemas SD and DD), we indirectly enforced KPCs by means of INDs, since this type of constraint can be expressed as referential integrity constraints [13] which are fairly standard mechanisms in most commercial relational DBMSs. So, whenever possible, we enforce KPCs by means of INDs. Motivated by this, in the next section we generalize the strategy used in the mapping of the ER schemas SD and DD (Figures 5 and 7).

## 6. USING INCLUSION DEPENDENCIES TO ENFORCE KEY PAIRING CONSTRAINTS

In this section, we will show that the strategy used to map the ER schemas SD and DD can be generalized to some cases of specialization structures. First, we will discuss the inheritance of keys in specialization structures with more detail in order to identify such cases accordingly.

### 6.1. Multiple Inheritance of Keys

Let E and F be entity schemes in a specialization structure of an ER schema. Assuming that E and F have no common native attributes, we can say that  $K_E \cap K_F \neq \{\}$  iff they have at least one common ancestor (we recall that any entity scheme is its own ancestor). In Figure 3(b), for example, D and C have two common ancestors, A and C, which means that  $K_A$  and  $K_C$  belong to

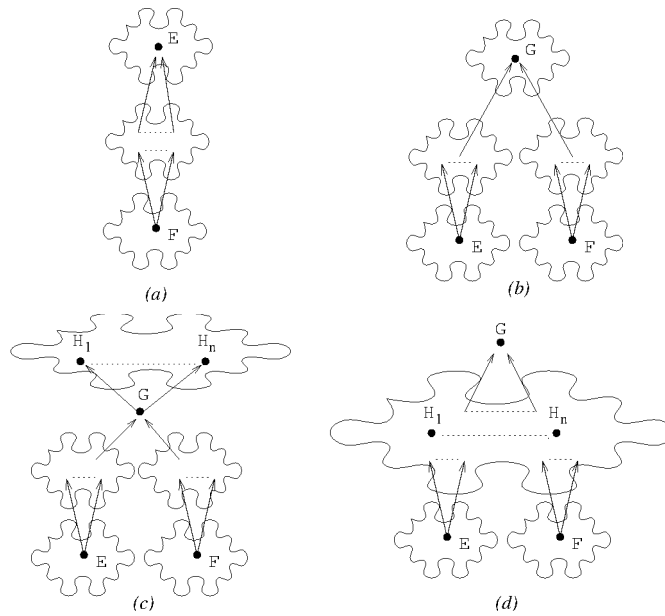


Fig. 8: Cases of Key Inheritance

**Algorithm 2**BeginInput: A specialization graph  $g$  of an ER schema  $S_E$ ;Output: A set  $\mathcal{I}$  of inclusion dependencies;Let A and B be entity schemes in a specialization structure of  $S_E$ ;Let  $K_A$  and  $K_B$  be the set of keys (native and inherited) of A and B respectively;For each arc (B,A) of  $g$ Let  $K = K_A \cap K_B$ ;Add the inclusion dependency  $T_B[K] \subseteq T_A[K]$  to  $\mathcal{I}$ ;End-For;End.

Fig. 9: Algorithm for Generating Inclusion Dependencies to Enforce KPC

the set of keys of D and C. To guide our discussion, we distinguish the following four cases of key inheritance, as illustrated in Figure 8:

- **Case 1:** E is an ancestor of F (Figure 8(a));
- **Case 2:** E and F are distinct entity schemes which have only one common ancestor G (Figure 8(b));
- **Case 3:** E and F are distinct entity schemes which have more than one common ancestor  $G, H_1, \dots, H_n$  ( $n \geq 2$ ), such that there is one and only one vertex G that is descendent of  $H_1, \dots, H_n$  (Figure 8(c)).
- **Case 4:** E and F are distinct entity schemes which have more than one common ancestor  $G, H_1, \dots, H_n$  ( $n \geq 2$ ), such that there is one and only one vertex G that is an ancestor of  $H_1, \dots, H_n$  (Figure 8(d)).

We shall see, for Cases 1 to 4, that we may enforce KPC by using Algorithm 2 (Figure 9) to generate the appropriate inclusion dependencies. For this discussion, we will assume that E and F

are entity schemes and that  $\mathsf{T}_E$  and  $\mathsf{T}_F$  are, respectively, the relation schemes that represent them. We also assume that  $\mathsf{T}_E$  includes all the attributes in  $K_E$  and that  $\mathsf{T}_F$  includes all the attributes in  $K_F$ .

*Case 1: E is an ancestor of F.*

For each path  $\langle F, G_1, G_2, \dots, G_n, E \rangle$ , Algorithm 2 will generate the inclusion dependencies

$$\mathsf{T}_F[K_0] \subseteq \mathsf{T}_{G_1}[K_0], \mathsf{T}_{G_1}[K_1] \subseteq \mathsf{T}_{G_2}[K_1], \mathsf{T}_{G_2}[K_2] \subseteq \mathsf{T}_{G_3}[K_2], \dots, \mathsf{T}_{G_n}[K_n] \subseteq \mathsf{T}_E[K_n]$$

where

$$K_0 = K_F \cap K_{G_1}, K_i = K_{G_i} \cap K_{G_{i+1}} \quad (1 < i < n - 1) \text{ and } K_n = K_{G_n} \cap K_E.$$

Let  $K = K_E \cap K_F$ . Then  $K \subseteq K_j$  ( $0 \leq j \leq n$ ), and thus

$$\mathsf{T}_F[K] \subseteq \mathsf{T}_{G_1}[K] \subseteq \mathsf{T}_{G_2}[K], \dots, \mathsf{T}_{G_n}[K] \subseteq \mathsf{T}_E[K]$$

which implies that

$$\mathsf{T}_F[K] \subseteq \mathsf{T}_E[K].$$

Let  $K_i$  and  $K_j$  be two distinct keys in  $K$ . Thus, for every tuple  $t$  of relation  $\mathsf{T}_F$ , if there is a tuple  $u$  of relation  $\mathsf{T}_E$  such that  $t[K_i] = u[K_i]$  then  $t[K_j] = u[K_j]$  and, as a consequence, KPC will be enforced.

*Case 2: E and F have only one common ancestor.*

If  $E$  and  $F$  have only one common ancestor  $G$ , the problem of enforcing KPC does not exist since  $K = K_E \cap K_F = K_G$ .

*Case 3: E and F are distinct entity schemes which have more than one common ancestor, but there is one and only one common ancestor G that is descendent of all the other ones.*

In this case, from the discussion of Case 1, we can say that

$$\mathsf{T}_E[L] \subseteq \mathsf{T}_G[L] \text{ and } \mathsf{T}_F[M] \subseteq \mathsf{T}_G[M],$$

where  $L = K_E \cap K_G$  and  $M = K_F \cap K_G$ .

Let  $K$  be the set of all keys of the common ancestors of  $E$  and  $F$ . We note that this is the set  $L$  of common keys of  $E$  and  $G$  and that it is also the set  $M$  of common keys of  $F$  and  $G$ , since every common ancestor of  $E$  and  $F$  is also an ancestor of  $G$ . We can say that this is also the case for the set of common keys of  $E$  and  $F$ , that is,  $K = K_E \cap K_F = L = M$ , and the inclusion dependencies generated are

$$\mathsf{T}_E[K] \subseteq \mathsf{T}_G[K] \text{ and } \mathsf{T}_F[K] \subseteq \mathsf{T}_G[K].$$

Let  $K_i$  and  $K_j$  be any two distinct keys in  $K$ . Thus, for every tuple  $t$  of relation  $\mathsf{T}_F$ , if there is a tuple  $u$  of relation  $\mathsf{T}_E$  such that  $t[K_i] = u[K_i]$  then there must be a tuple  $v$  of relation  $\mathsf{T}_G$  such that  $t[K_i] = v[K_i]$  and  $u[K_i] = v[K_i]$ . This implies that  $t[K_j] = v[K_j]$  and  $u[K_j] = v[K_j]$  and thus  $t[K_j] = u[K_j]$ . As a consequence, KPC will be enforced.

*Case 4: E and F are distinct entity schemes which have more than one common ancestor, but there is one and only one common ancestor G that is ancestor of all the other ones.*

Let  $H_1, H_2, \dots, H_n$  be the set of common ancestors of  $E$  and  $F$ , except  $G$ . For any two distinct vertices  $H_i$  and  $H_j$  ( $1 \leq i, j \leq n$ ) we will have

$$\mathsf{T}_E[K_{H_i}, K_G] \subseteq \mathsf{T}_{H_i}[K_{H_i}, K_G], \mathsf{T}_F[K_{H_i}, K_G] \subseteq \mathsf{T}_{H_i}[K_{H_i}, K_G] \quad (6)$$

and

$$\mathsf{T}_E[K_{H_i}, K_G] \subseteq \mathsf{T}_{H_j}[K_{H_i}, K_G], \mathsf{T}_F[K_{H_i}, K_G] \subseteq \mathsf{T}_{H_j}[K_{H_i}, K_G], \quad (7)$$

```

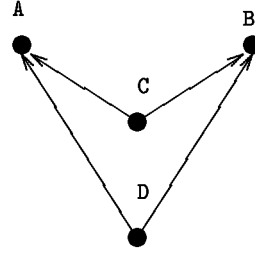
define entity A
  attributes KA char(4) not null,
           NA char(4) not null
  key      KA

define entity B
  attributes KB char(4) not null,
           NB char(4) not null
  key      KB

define entity C
  attributes NC char(4) not null
  specialization of A,B

define entity D
  attributes ND char(4) not null
  specialization of A,B
    
```

(a) Schema Definition



(b) Graph

```

CREATE TABLE TA
( KA CHAR(4) NOT NULL,
  NA CHAR(4) NOT NULL,
  PRIMARY KEY (KA)

CREATE TABLE TB
( KB CHAR(4) NOT NULL,
  NB CHAR(4) NOT NULL,
  PRIMARY KEY (KB)

CREATE TABLE TC
( KA CHAR(4) NOT NULL,
  KB CHAR(4) NOT NULL,
  NC CHAR(4) NOT NULL,
  PRIMARY KEY (KA),
  UNIQUE (KB)

CREATE TABLE TD
( KA CHAR(4) NOT NULL,
  KB CHAR(4) NOT NULL,
  ND CHAR(4) NOT NULL,
  PRIMARY KEY (KA),
  UNIQUE (KB)
    
```

```

TC[KA] ⊆ TA[KA]
TC[KB] ⊆ TB[KB]
TD[KA] ⊆ TA[KA]
TD[KB] ⊆ TB[KB]
    
```

(c) Relation Schema ND

Fig. 10: ER Schema ND

| T <sub>A</sub> | K <sub>A</sub> | N <sub>A</sub> |
|----------------|----------------|----------------|
|                | a1             | x              |
|                | a2             | y              |

| T <sub>B</sub> | K <sub>B</sub> | N <sub>B</sub> |
|----------------|----------------|----------------|
|                | b1             | u              |
|                | b2             | w              |

| T <sub>C</sub> | K <sub>A</sub> | K <sub>B</sub> | N <sub>C</sub> |
|----------------|----------------|----------------|----------------|
|                | a1             | b1             | p              |
|                | a2             | b2             | q              |

| T <sub>D</sub> | K <sub>A</sub> | K <sub>B</sub> | N <sub>D</sub> |
|----------------|----------------|----------------|----------------|
|                | a1             | b2             | f              |
|                | a2             | b1             | g              |

Fig. 11: Inconsistent State of ND

which implies that

$$T_E[K_H, K_G] \cup T_F[K_H, K_G] \subseteq T_{H_i}[K_H, K_G] \quad (8)$$

and

$$T_E[K_{H_i}, K_G] \cup T_F[K_{H_i}, K_G] \subseteq T_{H_j}[K_{H_i}, K_G]. \quad (9)$$

Let  $t$  be a tuple in relation  $T_E$  and  $u$  be a tuple in relation  $T_F$ . If  $t[K_{H_i}] = u[K_{H_i}]$  then, by (8),  $t[K_G] = u[K_G]$  and, if  $t[K_G] = u[K_G]$  then, by (9),  $t[K_{H_j}] = u[K_{H_j}]$ . Thus we conclude that KPC will be enforced.

If a specialization structure cannot be classified as one of the Cases from 1 to 4, then it is not possible to enforce KPC by using Algorithm 2 to generate the required inclusion dependencies. Thus, we define a constraint to the ER schemas for which it is possible, according to our approach, to generate a correct representation of specialization structures. Such a constraint, called *structural constraint*, is defined in what follows.

## 6.2. Structural Constraint

Consider the ER schema ND in Figure 10(a) and whose specialization graph is shown in Figure 10(b). Figure 10(c) presents the relational schema that results from ND when we use the mapping strategy discussed earlier and generate the INDs according to Algorithm 2. Figure 11 shows a state of this relation scheme where all the inclusion dependencies are satisfied, but which is inconsistent with the semantics of a specialization structure.

As we can see from Figure 11, this example clearly violates KPC since

$$T_C \bowtie_{T_C.K_A=T_D.K_A} T_D \neq T_C \bowtie_{T_C.K_B=T_D.K_B} T_D$$

Note that the existence of an arc (D,C) (or (C,D)) would lead to the generation of an inclusion dependency  $T_D[K_A, K_B] \subseteq T_C[K_A, K_B]$  (or  $T_C[K_A, K_B] \subseteq T_D[K_A, K_B]$ ), and that this would guarantee KPC (i.e., condition (5)). We notice that this would correspond to Case 1 discussed earlier.

In fact, looking at Schema ND in Figure 10, we know that there may exist some association between the instances of C and D. However, since there is no arc that explicitly represents this association, no integrity constraint is generated and, therefore, we cannot enforce KPC.

In such cases, it is not possible to guarantee the correctness of the mapping carried out according to the same strategy used for Cases 1 to 4. Thus, to avoid this problem, we impose a *structural constraint* on the specialization structures in order to generate a correct mapping according to this strategy [11]. This structural constraint synthesizes the cases discussed in Section 6.1 for which we have shown to be possible to apply the mapping strategy presented.

Thus, we say that a specialization structure with a graph  $g$  satisfies the structural constraint if, and only if:

*For any two vertices E and F of  $g$ , given the set H of their common ancestors, if E and F have more than one common ancestor there must be one and only one of them, say G  $\in$  H, such that:*

- a) G is a common descendent of the vertices in  $H - \{G\}$ , or
- b) G is a common ancestor of the vertices in  $H - \{G\}$ .

Note that the ER schema of Figure 10 is not valid under the structural constraint. However we can turn it into a valid schema by, for example, adding a new entity scheme G as the generic of both A and B. Another possibility to attain a similar effect is to include an arc connecting C and D, as discussed earlier in this section.

### 6.3. Dominant Keys

Considering the relational representation of the ER schema SD described in Figure 5, we can see that if we change the inclusion dependency

$$T_D[K_A, K_C] \subseteq T_C[K_A, K_C]$$

to

$$T_D[K_A] \subseteq T_C[K_A]$$

and redefine relation scheme  $T_D$  as

```
CREATE TABLE T_D
( K_C CHAR(4) NOT NULL,
  K_A CHAR(4) NOT NULL,
  N_D CHAR(4) NOT NULL,
  PRIMARY KEY (K_A))
```

i.e., if we just propagate the key from the root A, the resulting relation scheme  $T_D$  still corresponds to a correct representation of the entity scheme D, eliminating the problem of having to guarantee condition (2).

In what follows we will show that this strategy can also be generalized. First, we introduce the notion of dominant key [11] of an entity scheme.

**Definition 2** A set of attributes  $K$  is a *dominant key* of an entity scheme E when:

- a) E has no generic and  $K$  is its primary key, or
- b)  $K$  is a dominant key of some generic of E.

**Algorithm 3**BeginInput: *A specialization graph  $g$  of an ER schema  $S_E$ ;*Output: *A relational schema  $S_R = \langle \mathcal{T}, \mathcal{C} \rangle$ ;*For *each vertex  $g$  corresponding to an entity scheme  $E$ , generate a relation scheme  $T_E$  as follows:*Let  *$L$  be the set of non-native dominant keys of  $E$ ;*Let  *$A$  be the set of native attributes of  $E$ ;**Map each attribute of the elements of  $L \cup \{A\}$  to an attribute of  $T_E$ ;*Let  *$L$  be the set of dominant keys of  $E$* *Map each key in  $L$  to an alternate key of  $T_E$ ;*If  *$E$  has a primary key  $K$* Then *map  $K$  to the primary key of  $T_E$* Else *transform one of the alternate keys of  $T_E$  into its primary key;*End-If;*Add  $T_E$  to  $\mathcal{T}$ ;*End-For;For *each arc  $(E, G)$  of  $g$* Let  *$T_E$  and  $T_G$  be the relation schemes that map  $E$  and  $G$  respectively;*Let  *$K$  be the set of attributes that compose the dominant keys of  $G$ ;**Add the inclusion dependency  $T_E[K] \subseteq T_G[K]$  to  $\mathcal{C}$ ;*End-For;End.

Fig. 12: Algorithm for Mapping Complex Specialization Structures

We will use the notation  $\mathcal{D}(E)$  to denote the set of dominant keys of an entity scheme  $E$ .

The main idea of this strategy is to include as few attributes as possible in the relation scheme that represents each entity scheme in a specialization structure, but still enforcing KPC. To achieve this, we will restate condition (1) as

$$T_F[D] \subseteq T_E[D], \quad (10)$$

where  $D$  is the set of keys derived from  $\mathcal{D}(E) \cap \mathcal{D}(F)$ , and condition (5) as

$$T_E \bowtie_{T_E.D_i=T_F.D_i} T_F = T_E \bowtie_{T_E.D_j=T_F.D_j} T_F \quad (11)$$

for every  $D_i, D_j \in D$ .

This can be done since, for any two entity schemes  $E$  and  $F$ , only the keys derived from the minimum set of common keys of  $E$  and  $F$  ( $\mathcal{D}(E) \cap \mathcal{D}(F)$ ) are required to guarantee that the tuples in relations  $T_E$  and  $T_F$  that have the same values for these keys represent the same entity instance.

Then, if a relation scheme is used to represent an entity scheme, it should include at least its native attributes and the attributes of its dominant key. Thus, Algorithm 2 must be modified to generate INDs only over the set of common dominant keys of the two entity schemes in order to enforce KPC.

This strategy implies that a primary key  $K$  of an entity schema  $E$  will be included in the relation scheme  $T_F$  of one of its descendent  $F$  only if  $E$  is a root of the specialization graph. However, if it is not the case, the value of  $K$  for tuples of relation  $T_F$  may be obtained by joining  $T_F$  and the relation that represents  $E$  (say  $T_E$ ) on the attributes derived from  $\mathcal{D}(E) \cap \mathcal{D}(F)$ .

Thus we can now generalize the strategy used in the mapping of the ER schema  $SD$ , in which we only propagate the primary keys of the roots, i.e., the set of dominant keys. This strategy is described by Algorithm 3 in Figure 12 which incorporates a modified version of Algorithm 2 to enforce KPC by generating INDs over dominant keys.

## 7. CONCLUSIONS

In this paper, we discussed the problem of generating relational representations of complex specialization structures. We showed that, for such specialization structures, it is necessary to use a mapping strategy that is different from those applied by traditional relational database logical design methods. In particular, we showed that, for correctly mapping such structures into relational model constructs, it is necessary to enforce a specific integrity constraint, called the key pairing constraint (KPC).

Motivated by the shortcomings for directly enforcing KPCs, we showed that, for some cases of specialization structures, it is possible to enforce them by means of simple INDs, which are a type of constraint that can be expressed in commercial relational DBMSs as referential integrity constraints. Moreover, we showed that every specialization structure for which it is possible to enforce KPCs by means of INDs must satisfy the structural constraint defined in Section 6.2. It is worth noting that such specialization structures generalize those that are simple hierarchies and those having a single root as discussed in Section 6.1.

Although not addressed in this paper, it is also possible to show that the proposed mapping strategy can be adapted to generate optimized relational representations, i.e., relational schemas obtained by collapsing several entity and relationship schemes into a single relation scheme (as discussed in [8]), thus reducing the number of inclusion dependencies to be enforced.

Finally, it should be pointed out that the mapping strategy presented in this paper is part of the design/redesign method described in [12] which has been fully implemented for a commercial relational DBMS [15].

*Acknowledgements* — The work of the first two authors was partially supported by grants from the Brazilian government agencies CAPES and CNPq.

## REFERENCES

- [1] S. Agrawal, C. Keene, and A.M. Keller. Architecting object applications for high performance relational databases. Presented at the *OOPSLA'95 Workshop on Object Database Behavior, Benchmarks and Performance*, <ftp://ftp.cs.colorado.edu/pub/misc/oopsla95/agarwal.ps.gz> (1995).
- [2] S.W. Ambler. *Mapping Objects to Relational Databases*. AmbySoft Inc. White Paper, <http://www.ambysoft.com/mappingObjects.pdf> (2000).
- [3] C. Batini, S. Ceri, and S. Navathe. *Conceptual Database Design: An Entity-Relationship Approach*. Benjamin/Cummings, Redwood City, California (1992).
- [4] M. Blaha and W. Premerlani. *Object-Oriented Modeling and Design for Database Applications*. Prentice-Hall, Upper Saddle River, New Jersey (1998).
- [5] G. Booch. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, Menlo Park, California (1994).
- [6] M.A. Casanova, R. Fagin, and K. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. In *Proceedings of the 1st ACM Symposium on Principles of Database Systems*, Los Angeles, California, pp. 171–176, ACM Press (1982).
- [7] M.A. Casanova, L. Tucherman, A.L. Furtado, and A. Pacheco. Optimization of relational schemas containing inclusion dependencies. In *Proceedings of the 15th International Conference on Very Large Data Bases*, Amsterdam, The Netherlands, pp. 317–325 (1989).
- [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] S. Ceri, editor. *Methodology and Tools for Database Design*. North-Holland, Amsterdam, The Netherlands (1983).
- [10] P.P. Chen. The entity-relationship model: towards a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36 (1996).
- [11] A.S. da Silva. *A Contribution to the Problem of Maintaining Optimized Relational Representations of Entity-Relationship Schemas*. Master's thesis, Department of Computer Science, UFMG, Belo Horizonte, Brazil (in Portuguese) (1995).
- [12] A.S. da Silva, A.H.F. Laender, and M.A. Casanova. An approach to maintaining optimized relational representations of entity-relationship schemas. In B. Thalheim, editor, *Conceptual Modeling - ER'96, 15th International Conference on Conceptual Modeling, Proceedings*, Cottbus, Germany, Lecture Notes in Computer Science 1157, pp. 292–308, Springer-Verlag, Berlin, Germany (1996).

- [13] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, Menlo Park, California, 3rd. edition (2000).
- [14] D.W. Embley. *Object Database Development: Concepts and Principles*. Addison-Wesley, Reading, Massachusetts (1988).
- [15] A.A. Ferreira, A.H.F. Laender, and A.S. da Silva. A tool for the design/redesign of relational databases. In *Proceedings of the 17th Brazilian Symposium on Databases*, Fortaleza, Brazil, pp. 169–182, (in Portuguese) (1997).
- [16] M. Frank. The evolution of client/server case tools. *DBMS*, **1**(9):67–74 (1996).
- [17] J.-L. Hainaut, V. Englebert, J. Henrard, J.-M. Hick, and D. Roland. Database evolution: the DB-Main approach. In P. Loucopoulos, editor, *Entity-Relationship Approach - ER'94, Business Modelling and Re-Engineering, 13th International Conference on the Entity-Relationship Approach, Proceedings*, Manchester, U.K., Lecture Notes in Computer Science 881, pp. 112–131, Springer-Verlag, Berlin, Germany (1994).
- [18] J.-L. Hainaut, J.-M. Hick, V. Englebert, J. Henrard, and D. Roland. Understanding the implementation of is-a relations. In B. Thalheim, editor, *Conceptual Modeling - ER'96, 15th International Conference on Conceptual Modeling, Proceedings*, Cottbus, Germany, Lecture Notes in Computer Science 1157, pp. 42–57, Springer-Verlag, Berlin, Germany (1996).
- [19] J.-L. Hainaut, C. Tonneau, M. Joris, and M. Chandelon. Schema transformation techniques for database reverse engineering. In R.A. Elmasri, V. Kouramajian, and B. Thalheim, editors, *Entity-Relationship Approach - ER'93, 12th International Conference on the Entity-Relationship Approach, Proceedings*, Arlington, Texas, USA, Lecture Notes in Computer Science 823, pp. 364–375, Springer-Verlag, Berlin, Germany (1994).
- [20] A.H.F. Laender, M.A. Casanova, A.P. Carvalho, and L.F.G.G.M. Ridolfi. An analysis of SQL integrity constraints from an entity-relationship model perspective. *Information Systems*, **19**(4):331–258 (1994).
- [21] 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).
- [22] V.M. Markowitz and A. Shoshani. An overview of the Lawrence Berkeley Laboratory extended entity-relationship tools. In P. Loucopoulos, editor, *Entity-Relationship Approach - ER'94, Business Modelling and Re-Engineering, 13th International Conference on the Entity-Relationship Approach, Proceedings*, Manchester, U.K., Lecture Notes in Computer Science 881, pp. 333–350, Springer-Verlag, Berlin, Germany (1994).
- [23] J. Melton, J., and A.R. Simon. *Understanding the New SQL: A Complete Guide*. Morgan Kaufman, San Francisco, California (1993).
- [24] S.B. Navathe. Evolution of data modeling for databases. *Communications of the ACM*, **35**(9):112–123 (1992).
- [25] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenson. *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, New Jersey (1991).
- [26] M. Snoeck and G. Dedene. Generalization/specialization and role in object oriented conceptual modeling. *Data & Knowledge Engineering*, **1**(19):171–195 (1996).
- [27] M. Stonebraker, D. Moore, and P. Brown. *Object-Relational DBMS: Tracking the Next Great Wave*. Morgan Kaufman, San Francisco, California (1998).
- [28] T.J. Teorey. *Database Modeling and Design*. Morgan Kaufman, San Francisco, California (1998).
- [29] 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).