

# On the design and maintenance of optimized relational representations of entity-relationship schemas

Marco A. Casanova<sup>a\*</sup>, Luiz Tucherman<sup>a</sup> and Alberto H.F. Laender<sup>b</sup>

<sup>a</sup>Rio Scientific Center, IBM Brazil, Av. Pres. Vargas, 824/22 andar, 20.071-001 Rio de Janeiro RJ, Brazil

<sup>b</sup>Computer Science Department, Federal University of Minas Gerais, P.O. Box 702, 30.161-970 Belo Horizonte MG, Brazil

Received 1 April 1992

Accepted 28 July 1992

## *Abstract*

A method for obtaining optimized relational representations of database conceptual schemas in an extended entity-relationship model is first proposed. The method incorporates and generalizes a familiar heuristics to obtain good relational representations and also produces, for each relational structure, an explanation indicating which concepts it represents. Then, a redesign method that, given changes to the conceptual schema, generates a plan to modify the original representation and to organize the database state is described.

*Keywords.* Conceptual database design; extended entity-relationship model; relational schema optimization; database schema evolution.

## 1. Introduction

A large share of the database systems in commercial use today are based on the relational technology, and this scenario is likely to continue throughout the rest of the decade. This implies that the problem of designing, implementing and maintaining relational databases will retain considerable importance in the near future.

The design of a relational database typically starts with the definition of an entity-relationship schema describing the conceptual structures of the database. Then, the design proceeds by mapping the conceptual schema into a relational representation. This second step of the process is usually manual and, possibly, improves the representation, guided by simple empirical heuristics.

The first contribution of this paper is to define a systematic design method that accepts as input an entity relationship conceptual schema and generates an optimized relational representation for the schema. We use the term optimized in the sense that the number of dependencies of the relational schema is minimized [11]. The representation includes explanations indicating which objects of the conceptual schema each relation scheme represents and why. The method generalizes an optimization heuristics that is commonly adopted and it operates based exclusively on the structure of the conceptual schema.

\* Corresponding author.

The second contribution is a redesign method that accepts as input a conceptual schema, the relational representation for the schema produced by the design method and a sequence of changes on the schema, and produces as output the new conceptual schema and a plan to create an optimized relational representation for the new schema and to restructure the relational database state accordingly.

To understand the redesign method, consider the following scenario (the numbers between parenthesis below relate each step of the process with a mapping shown in *Fig. 1*). Suppose that the database designer defines a conceptual schema  $S_E$ , which the design method mapped (1) into a relational representation  $S_R$ . Assume now that the designer defines a set of changes to  $S_E$  and that the relational database currently is in a state  $\sigma$  that conforms (2) to the relational representation  $S_R$ . The redesign method will then process the changes to map (3)  $S_E$  into the new conceptual schema  $S'_E$  and to produce a redesign plan that will take (4)  $S_R$  into the new relational representation  $S'_R$  and that will restructure (5)  $\sigma$  into the new relational state  $\sigma'$ . The redesign method guarantees that the redesign plan is correct in the sense that

- $\sigma'$  conforms (7) to the new representation  $S'_R$ ;
- $S'_R$  correctly and optimally represents (6)  $S'_E$ . In particular, note that some changes, when applied to  $S_E$ , may invalidate previous optimizations that  $S_R$  reflects, while others may create opportunities for new optimizations that  $S'_R$  must take into account.

A broad perspective of logical database design is given in [4]. The use of variations of the entity-relationship model [13] for database conceptual design has been extensively investigated (see e.g. [27] for a survey). The idea of transforming an extended entity-relationship schema into a relational schema is not new [2, 3, 5, 15, 21, 25]. The difference between most of the proposals lies in the way the structures of the entity-relationship model are translated into structures of the relational model based on the scope of the extensions adopted. Some of the methods proposed are performance-driven [6]. More recently, the NIAM model has been pointed out as an alternative approach to conceptual database design. The design of relational databases, based on the NIAM model, was first addressed in [20] and is extensively discussed in [22]. A design method for object-oriented databases is described in [7]. In fact, this reference maintains the tradition of starting the design process with a description of the database schema in a high level language based on a semantic data model.

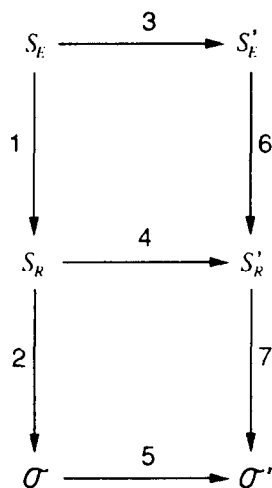


Fig. 1. The redesign process.

Research on expert tools for database design can be found in [8, 11, 23, 24, 28]. In particular, the design method we proposed in [11] uses the structure of the relational representation as a guide to the optimization phase, a strategy we abandoned because it proved inadequate when we considered the redesign problem. The optimization we describe in this paper uses the structure of the conceptual schema instead. The work reported here is part of a larger project focusing on software tools for automated database design, described in part in [10–12, 17, 25, 29].

This paper is divided as follows. Section 2 describes the variation of the entity-relationship model adopted. Section 3 reviews some concepts of the relational model. Section 4 discusses the design method, whereas Section 5 addresses the redesign method. Finally, Section 6 contains the conclusions.

## 2. The extended entity-relationship model adopted

We summarize in Section 2.1 the variation of the entity-relationship model adopted. A careful description of the syntax and semantics of a more complete variation can be found in [12]. We then present in Section 2.2 a simple example, that we will expand throughout the paper, illustrating the concepts introduced in Section 2.1.

### 2.1. Description of the model

An entity set has a name and, optionally, a set of attributes, a primary key and a set of alternate keys. If the primary key is neither specified nor inherited from another scheme (see below), it is taken by default as the list of all attributes.

An entity set is defined through an *entity scheme* of the form (the expressions between brackets are optional):

$$\text{define entity } E \text{ [attributes } A_1 D_1, \dots, A_n D_n \text{] [key } K_0 \text{ [ } \dots \text{ key } K_p \text{ ]]}$$

where  $K_i$  is a sublist of  $A_1, \dots, A_n$ , for  $i=0, \dots, p$ . We say that  $E$  is the *name*,  $A_1, \dots, A_n$  is the list of *attribute names*,  $K_0$  is the *primary key* and  $K_1, \dots, K_p$  are the *alternate keys* of the scheme. We also say that  $D_j$  is the domain of  $A_j$  and that  $A_j$  *accepts null values*, if  $D_j$  does, for  $j=1, \dots, n$ .

A relationship set is basically a subset of the Cartesian product of a collection of entity sets. We let an entity set participate in a relationship set more than once, provided that a distinct role is assigned to each occurrence. A relationship set may optionally be defined as total with respect to the role of an entity set. We also introduce the notion of identifier as the counterpart of keys for relationship sets. Thus, for example, if we indicate that *ROOM* and *SCHEDULE* form the identifier of the relationship set named *ALLOCATED* over the entity sets *ROOM*, *SCHEDULE* and *EVENT*, we are saying that each room in each time slot can be allocated to just one event. We chose the term ‘*identifier*’ to call attention to the fact that we now have a list of participants of the relationship set, possibly identified by their roles, not a list of attributes. Lastly, we let a relationship set be equipped with more than one identifier, which is necessary, for example, to define 1–1 binary relationship sets. As for entity schemes, the primary identifier, the alternate identifiers and the list of attributes can all be omitted. If the primary identifier is omitted, it is taken by default as the list of the roles of all participants.

A relationship set is defined through a *relationship scheme* of the form:

define relationship  $R$  over  $O_1$  [as  $N_1$ ] [total],  $\dots$ ,  $O_m$  [as  $N_m$ ]  
 [total]  
 [attributes  $A_1 D_1, \dots, A_n D_n$ ]  
 [identifier  $I_0$  [... identifier  $I_p$ ]]

We say that  $R$  is the *name* and that  $A_1, \dots, A_n$  is the list of *attribute names* of the scheme. For each  $i=1, \dots, m$ , the  $i$ th *participant* is  $O_i$  and the  $i$ th *role* is  $N_i$ , if specified, otherwise it is  $O_i$ , by default. Therefore, the  $i$ th role acts as an alias for the  $i$ th participant. When ‘total’ is specified for ‘ $O_i$  [as  $N_i$ ]’, we say that scheme is *total* on the  $i$ th participant (or role).

For each  $j=0, \dots, p$ ,  $I_j$  must be a list of roles. We say that  $I_0$  is the *primary identifier* and  $I_1, \dots, I_p$  are the *alternate identifiers* of the scheme. Furthermore, we say that  $R$  is *functional* on the  $j$ th participant (or role) iff the  $j$ th role is an identifier of  $R$ .

The entity sets may be organized as an acyclic specialization graph. That is, an entity set may be declared as a specialization of more than one entity set in our model. If an entity set  $F$  is defined as a specialization of  $E$ , then  $F$  must always be a subset of  $E$ . Lastly, an entity set  $F$  inherits the attributes and keys of all sets it specializes, directly or transitively (to avoid conflicts, when inherited, attributes with the same name are qualified with the name of the set they originate from).

More precisely, we introduce a *specialization declaration* as an expression of the form:

specialize  $E$  into  $F_1, \dots, F_m$

where  $E, F_1, \dots, F_m$  are names of entity schemes. We say that  $F_1, \dots, F_m$  are *specializations* of  $E$  and that  $E$  is a *generalization* of  $F_1, \dots, F_m$ .

An *EER conceptual schema* or, simply, an *EER schema*, is a triple  $\mathcal{S}_E = (\mathbf{E}, \mathbf{R}, \mathbf{S})$  where  $\mathbf{E}$  is a set of entity schemes,  $\mathbf{R}$  is a set of relationship schemes and  $\mathbf{S}$  is a set of specialization declarations.

The *graph* of an EER schema  $\mathcal{S}_E$  is a labelled directed multigraph,  $\mathbf{g}(\mathcal{S}_E) = (\mathbf{V}, \mathbf{A}, \mathbf{I})$ , allowing more than one arc between two nodes and with the arcs partially labelled by  $\mathbf{I}$ , such that  $\mathbf{V}$  is the set of the names of all entity or relationship schemes of  $\mathcal{S}_E$  and an arc  $(O, P)$  is in  $\mathbf{A}$  iff

- $O$  is a specialization of  $P$  in  $\mathcal{S}_E$ , in which case  $\mathbf{I}((O, P))$  is not defined, or
- $O$  is a relationship scheme of  $\mathcal{S}_E$  such that  $P$  participates with role  $N$ , in which case  $\mathbf{I}((O, P)) = N$ , or
- $P$  is a relationship scheme of  $\mathcal{S}_E$  such that  $O$  participates with role  $N$  and  $P$  is total on  $N$ , in which case  $\mathbf{I}((O, P)) = N$ .

This concludes the description of the syntax of the model. The semantics is standard and can be found in [29]. In particular, let  $\sigma$  be a database state for an EER schema  $\mathcal{S}_E$ . We say that  $e$  is an *E-entity* in  $\sigma$  iff  $e$  is in the entity set that  $\sigma$  associates with the entity scheme  $E$ ; likewise, we say that  $r$  is a *R-relationship* in  $\sigma$  iff  $r$  is in the relationship set that  $\sigma$  associates with the relationship scheme  $R$ . If  $\sigma$  is understood from the context, then the reference to  $\sigma$  is omitted.

We now discuss in detail how the semantic constraints associated with relationship schemes and specialization declarations modify the operations because this point directly affects the design method described in Section 4.

Let  $R$  be a relationship scheme and  $E$  be the entity scheme on role  $N$  of  $R$ . If  $R$  is not total on  $N$ , we fix that deletions from  $E$  block immediately with respect to (w.r.t.)  $R$  and insertions into  $R$  block immediately w.r.t.  $E$ . Deletions from  $R$  and insertions into  $E$  suffer no restriction from this point of view (they may do so for other reasons). However, if  $R$  is

total on  $N$ , we fix that deletions from  $E$  (or  $R$ ) propagate immediately w.r.t.  $R$  (or  $E$ ) and insertions into  $R$  (or  $E$ ) block deferredly w.r.t.  $E$  (or  $R$ ).

To block the deletion of an  $E$ -entity  $e$  immediately w.r.t.  $R$  means to reject the deletion when submitted, if there is an  $R$ -relationship  $r$  in which  $e$  participates. To propagate the deletion of an  $E$ -entity  $e$  immediately w.r.t.  $R$  means to delete all  $R$ -relationships  $r$  in which  $e$  participates. To block the insertion of a  $R$ -relationship  $r$  immediately w.r.t.  $E$  (respectively, deferredly w.r.t.  $E$ ) means to reject the insertion when submitted (respectively, at the end of the transaction), if the  $E$ -entity  $e$  that  $r$  refers to does not exist. To propagate the deletion of an  $R$ -relationship  $r$  immediately w.r.t.  $E$  means to delete the  $E$ -entity  $e$  that participates in  $r$ , if there is no other  $R$ -relationship in which  $e$  participates. The other options mentioned above are similarly defined.

Note that totality severely restricts the alternatives for governing insertions and deletions. If we opted to block insertions or deletions immediately, then such operations would be impossible to execute. On the other hand, propagation of insertions is not feasible because one insertion does not determine the other. Indeed, the insertion of an  $R$ -relationship  $r$  does not determine the attributes of the participant entities. Likewise, the insertion of an  $E$ -entity  $e$  does not determine the other participants of the required relationship. Therefore, the only other alternative for the options selected above would be to propagate deletions from  $E$  deferredly w.r.t.  $R$ , which we discard in favor of immediate propagation of deletions.

Let  $F$  be a specialization of  $E$ . We also fix that deletions from  $E$  and insertions into  $F$  block immediately. That is, the deletion of an  $E$ -entity  $e$  is possible only if  $e$  does not occur in  $F$  and the insertion of an  $F$ -entity  $f$  is accepted only if it already occurs in  $E$ . The general problem of updating specialization hierarchies is discussed in [29].

## 2.2. Example of an EER schema

We define in this section the EER schema, called *COMPANY*, we will use in the examples throughout the paper. Briefly, the entity scheme *EMPLOYEE* describes the set of employees; *RESEARCHER*, the set of employees that are researchers; *ADMINISTRATIVE*, the set of employees that form the administrative staff; *STOCK\_HOLDER*, the set of stock holders of the company; *MANAGER*, the set of managers, which are always classified as administrative staff and which are always stock holders; and *PROJECT*, the set of research projects. The relationship scheme *WORKS* describes a set of relationships that associate each researcher to the single major research project he is assigned to, a restriction that is captured by defining *RESEARCHER* as the identifier of *WORKS*.

To simplify the development of the example in later sections, we will use just the first letter of the name to refer to an entity or relationship scheme from now on.

The definition of the EER schema *COMPANY*, in the EER data definition language introduced in Section 2.1, is shown in Fig. 2, the EER diagram, in Fig. 3, and the graph, in Fig. 4. Note that the arc  $(W, R)$  is labelled with  $R$ , since  $R$  participates in  $W$  with role  $R$ , by default, and similarly for  $(W, P)$ . This labelling is not strictly necessary in this example, but it becomes crucial when an entity scheme participates in a relationship scheme in more than one role.

## 3. Review of the relational model

We summarize in Section 3.1 the concepts of the relational model we will need in the next sections, leaving the examples to Section 3.2.

We adopt a neutral and concise syntax for relation schemes, view declarations and

---

```

define entity EMPLOYEE
  attributes Id      char(4) not null,
             Salary  decimal(8,2)
  key Id

define entity RESEARCHER
  attributes Degree  char(4) not null

define entity ADMINISTRATIVE
  attributes Job_Desc char(40) not null

define entity MANAGER
  attributes Title   char(40) not null

define entity STOCK_HOLDER
  attributes Code    char(4) not null,
             Stocks  integer
  key Code

define entity PROJECT
  attributes P_Name  char(20) not null,
             Contractor char(20)
  key P_Name

define relationship WORKS
  over RESEARCHER, PROJECT
  identifier RESEARCHER

specialize EMPLOYEE into RESEARCHER, ADMINISTRATIVE
specialize ADMINISTRATIVE into MANAGER
specialize STOCK_HOLDER into MANAGER

```

---

Fig. 2. The EER schema *COMPANY*.

integrity constraints that can be easily translated to the standard SQL language, as specified in [1] or [18], for example. We refer the reader to [9] for a full discussion about the use of standard SQL to capture entity-relationship concepts. Whenever necessary, we will also use SQL-like data manipulation language statements, without formally defining their syntax or semantics.

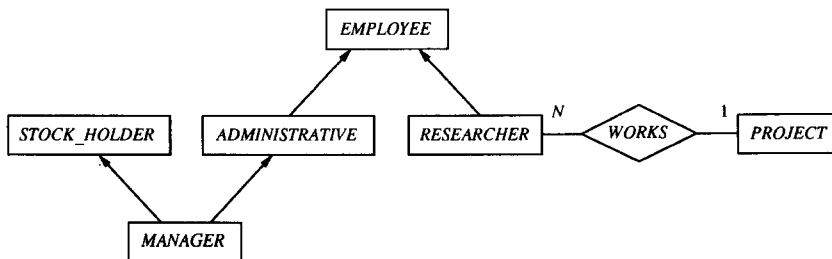
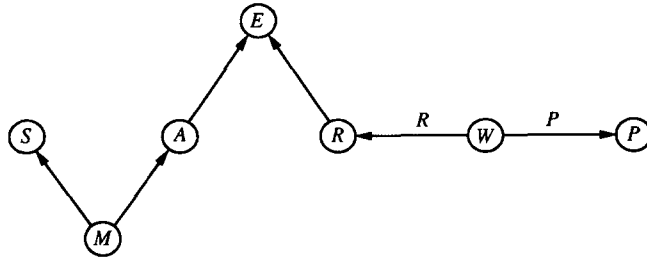


Fig. 3. The diagram of the schema *COMPANY*.

Fig. 4. The graph of the schema *COMPANY*.

### 3.1. Basic concepts

A *relation scheme* is an expression of the form:

```
define relation T [attributes A1 D1, . . . , An Dn]
                  [key K0] . . . [key Kp]
```

where  $K_i$  is a sublist of  $A_1, \dots, A_n$ , for  $i=0, \dots, p$ . We say that  $T$  is the *name*,  $A_1, \dots, A_n$  is the list of *attribute names*,  $K_0$  is the *primary key* and  $K_1, \dots, K_p$  are the *alternate keys* of the scheme. We also say that  $D_j$  is the *domain* of  $A_j$  and that  $A_j$  *accepts null values* if  $D_j$  does, for  $j=1, \dots, n$ . The attributes of the primary key must not accept null values, but those of an alternate key may accept null values.

Let  $\mathbf{R}$  be a set of relation schemes. A *view scheme* over  $\mathbf{R}$  is an expression of the form:

```
define view V [attributes A1 D1, . . . , An Dn]
              [key K0] . . . [key Kp]
as Q
```

where  $V$  is distinct from the names of the schemes in  $\mathbf{R}$ ,  $K_i$  is a sublist of  $A_1, \dots, A_n$ , for  $i=0, \dots, p$ , and  $Q$  is a SQL query over  $\mathbf{R}$  defining an  $n$ -ary relation (to match the number of attributes of  $V$ ). We say that  $V$  is the *name*,  $A_1, \dots, A_n$  is the list of *attribute names*,  $K_0$  is the *primary key*,  $K_1, \dots, K_p$  are the *alternate keys* and  $Q$  is the *defining expression* of the view. We also say that  $D_j$  is the *domain* of  $A_j$  and that  $A_j$  *accepts null values* if  $D_j$  does, for  $j=1, \dots, n$ . Note that, for simplicity, we do not let a view be defined over another view since  $Q$  must be an expression over  $\mathbf{R}$ .

The keys defined for  $V$  must be a logical consequence of the keys defined for the relation schemes in  $\mathbf{R}$ . The domains of the view attributes must also be derived from the domains of the relation schemes in  $\mathbf{R}$  via the defining expression  $Q$ .

A *view definition* over  $\mathbf{R}$  is a pair  $(V, E)$  where  $V$  is a view scheme over  $\mathbf{R}$  and  $E$  specifies correct translations [16] for the operations over  $V$  into operations over the schemes in  $\mathbf{R}$ . We refer the reader to [11] for a detailed discussion about the need to consider view operation translations.

We also introduce a class of inclusion dependencies [11] that is sufficiently powerful to capture the referential integrity constraints between tables that represent specializations and relationships, including the way these two concepts affect the semantics of the operations. Let  $\mathbf{R}$  be a set of relation schemes and  $\mathbf{V}$  be a set of view definitions over  $\mathbf{R}$ . An *inclusion dependency with propagation options*, or an *INDP*, over  $\mathbf{R}$  and  $\mathbf{V}$  is an expression of the form  $T_1[X_1] \subseteq T_2[X_2] : (\gamma, \delta)$  where, for  $i=1, 2$ :

- $T_i$  is the name of a relation scheme in  $\mathbf{R}$  or of a view definition in  $\mathbf{V}$ ;
- $X_i$  is a sequence of distinct attributes of  $T_i$  such that  $X_1$  and  $X_2$  are domain compatible;

- $\gamma$  is the *insertion option* and  $\delta$  is the *deletion option* of the INDP, with  $\gamma$  taking values from the set  $\{b^i, b^d\}$ , and  $\delta$  from the set  $\{b^i, b^d, p^i, p^d\}$ , with the following intended interpretation:

$b^i$	block immediately	$p^i$	propagate immediately
$b^d$	block deferredly	$p^d$	propagate deferredly

The restriction imposed on  $\gamma$  just avoids the indeterminacy that would appear if one tried to propagate an insertion into  $T_1$  to an insertion into  $T_2$ , since the insertion into  $T_1$  determines only the values of the attributes of  $T_2$  that appear in  $X_2$ .

A *relational schema* is a triple  $\mathcal{S}_R = (\mathbf{R}, \mathbf{V}, \mathbf{I})$  where  $\mathbf{R}$  is a set of relation schemes with distinct names,  $\mathbf{V}$  is a set of view definitions over  $\mathbf{R}$  and  $\mathbf{I}$  is a set of INDPs over  $\mathbf{R} \cup \mathbf{V}$ .

A *relational database state*, or simply a *state*,  $\sigma$  for  $\mathcal{S}_R$  assigns a relation  $\sigma(R)$  to each relation scheme  $R$  in  $\mathbf{R}$ . We also extend the state  $\sigma$  to assign values to the relational expressions over  $\mathbf{R}$  and to the schemes of the views definitions via their defining expressions.

Let  $\lambda$  denote both the null value or tuples of null values of arbitrary length. A state  $\sigma$  *satisfies* the primary key  $K$  of a relation scheme  $R$  iff, for any  $t, u \in \sigma(R)$ , if  $t[K] = u[K]$  then  $t = u$ . Now, a state  $\sigma$  *satisfies* an alternate key  $L$  of  $R$  iff, for any  $t, u \in \sigma(R)$ , if  $t[K] = u[K] \neq \lambda$ , then  $t = u$  and, for any  $t \in \sigma(R)$ , if  $t[L_i] = \lambda$ , for some attribute  $L_i$  of  $L$ , then  $t[L_j] = \lambda$ , for any other attribute  $L_j$  of  $L$ . Hence, unlike primary keys, we allow the attributes of an alternate key to be simultaneously null. This alternative semantics is sufficient for the purposes of Section 4. A state  $\sigma$  *satisfies*  $T_1[X_1] \subseteq T_2[X_2] : (\gamma, \delta)$  iff  $\sigma(T_1[X_1]) \subseteq \sigma(T_2[X_2])$ . Finally, we say  $\sigma$  is *consistent* iff  $\sigma$  satisfies all keys and INDPs of  $\mathcal{S}_R$ .

The semantics of  $T_1[X_1] \subseteq T_2[X_2] : (\gamma, \delta)$  also has a dynamic perspective capturing how the insertion and deletion options affect the behavior of the operations over  $T_1$  and  $T_2$ , as defined in [10]. For example, if the deletion option is  $b^i$ , for block immediately, then there is a test that rejects a deletion from  $T_2$ , if the state  $\sigma$  after the deletion is such that  $\sigma(T_1[X_1]) \not\subseteq \sigma(T_2[X_2])$ . If the deletion option is  $b^d$ , for block deferredly, then there is a test that aborts the transaction if the state  $\sigma$  at commit time is such that  $\sigma(T_1[X_1]) \not\subseteq \sigma(T_2[X_2])$ . The definition of the other options follows similarly.

### 3.2. Example of a relational schema

In this section we define a relational schema, called  $C$ , with three relation schemes,  $E^*$ ,  $S^*$  and  $P$ , six view definitions,  $E$ ,  $R$ ,  $A$ ,  $W$ ,  $S$  and  $M$ , and two INDPs.

Observe that the view attribute domains follow from the domains of the underlying relation schemes via the defining expressions. Also note that, for simplicity, we did not include in the view schemes those attributes inherited through the specialization hierarchy, but rather we assumed that users can recover them through joins. However, the design tool we implemented [26] includes an elaborated algorithm to add the inherited attributes to the view schemes and to resolve attribute name conflicts.

Furthermore, observe that the keys included in the view schemes are indeed a logical consequence of those defined for the relation schemes. For example, the primary key  $Code$  of  $S$  and  $M$  is obviously a consequence of the primary key  $Code$  of  $S^*$  due to the defining expression of  $S$  and  $M$ . Moreover, the alternate key  $Id$  of  $M$  is a logical consequence of the alternate key  $Id$  of  $S^*$ . Indeed, let  $\sigma$  be a consistent database state for the relational schema  $C$  and let  $t, u \in \sigma(M)$ . Assume that  $t[Id] = u[Id]$ . Then, there are tuples  $t', u' \in \sigma(S^*)$  such that  $t'[Code, Id, Title] = t$  with  $t'[Id] \neq \lambda$  and  $u'[Code, Id, Title] = u$  with  $u'[Id] \neq \lambda$ . But, since  $Id$  is an alternate key of  $S^*$ ,  $t' = u'$ . Hence,  $t = u$ .

However, we postpone to Section 4 an explanation for the translations of the view operations described in *Tables 1* and *2* below.

Table 1.  
Translation of the operations for  $S$

Operation	Translation
insert into $S$ (Code = $c$ , Stocks = $s$ )	insert into $S^*$ (Code = $c$ , Stocks = $s$ , Id = $\lambda$ , Title = $\lambda$ )
delete from $S$ where $Q$	delete from $S^*$ where $Q$ and Id = $\lambda$
update $S$ set Stocks = $s$ where $Q$	update $S^*$ set Stocks = $s$ where $Q$

Table 2.  
Translation of the operations for  $M$

Operation	Translation
insert into $M$ (Code = $c$ , Id = $i$ , Title = $t$ )	update $S^*$ set Id = $i$ , Title = $t$ where Code = $c$ and Id = $\lambda$
delete from $M$ where $Q$	update $S^*$ set Id = $\lambda$ , Title = $\lambda$ where $Q$ and Id $\neq \lambda$
update $M$ set Title = $t$ where $Q$	update $S^*$ set Title = $t$ where $Q$ and Id $\neq \lambda$

```

/*
  Relation Schemes
*/
define relation E*
  attributes Id      char(4) not null,
             Salary  decimal(8,2),
             Degree  char(4),
             Job_Desc char(40),
             P_Name  char(20)

  key Id

define relation P
  attributes P_Name  char(20) not null,
             Contractor char(20)

  key P_Name

define relation S*
  attributes Code    char(4) not null,
             Stocks  integer,
             Id      char(4),
             Title   char(40)

  key Code
  key Id

/*
  View S (see Table 1 below for the translation of the operations)
*/
define view S
  attributes Code    char(4) not null,
             Stocks  integer

  key Code

  as select Code, Stocks
     from S*

/*
  View M (see Table 2 below for the translation of the operations)

```

Fig. 5. The relational schema  $C$  (continued on next page).

```

*/
define view M
  attributes Code      char(4) not null,
             Id        char(4) not null,
             Title     char(40) not null
  key Code
  key Id
  as select Code, Id, Title
     from S*
     where Id ≠ λ
/*
View E (the translation of the operations follows like that in Table 1)
*/
define view E
  attributes Id        char(4) not null,
             Salary    decimal(8,2)
  key Id
  as select Id, Salary
     from E*
/*
View R (the translation of the operations follows like that in Table 2)
*/
define view R
  attributes Id        char(4) not null,
             Degree    char(4) not null
  key Id
  as select Id, Degree
     from E*
     where Degree ≠ λ
/*
View A (the translation of the operations follows like that in Table 2)
*/
define view A
  attributes Id        char(4) not null,
             Job_Desc  char(40) not null
  key Id
  as select Id, Job_Desc
     from E*
     where Job_Desc ≠ λ
/*
View W (the translation of the operations follows like that in Table 2)
*/
define view W
  attribute Id        char(4) not null,
           P_Name     char(20) not null
  key Id
  as select Id, P_Name
     from E*
     where P_Name ≠ λ
/*
INDPs
*/
M[Id]≡A[Id]: (bi, bi)
W[P_Name]≡P[P_Name]: (bi, bi)

```

Fig. 5 (cont.).

#### 4. Automatic synthesis of relational representations

We describe in Section 4.1 the basic idea behind the design method. In Section 4.2, we present the overall structure of the method and a brief example illustrating how it operates.

##### 4.1. Schema collapsing

A relational schema  $R$  is a *one-to-one relational representation* of an EER schema  $E$  [25] iff  $R$  contains a distinct relation scheme (with the appropriate attributes) representing each entity or relationship scheme of  $E$  and a distinct INDP capturing the semantics of each arc of the graph of  $E$ . A one-to-one representation is straightforward to obtain, but it contains a potentially large number of INDPs that are expensive to check for violations.

To reduce the number of INDPs of a relational representation, a fairly common heuristic is to collapse a relationship scheme  $F$  into an entity scheme  $E$  and to represent both as a single relation scheme  $T$ , if  $F$  is functional on a role that  $E$  plays. The most frequent case is when  $F$  is binary and  $n - 1$ , with  $E$  ‘on the  $n$  side’. The same heuristic applies as well when  $F$  is an entity scheme that specializes  $E$ , possibly restricted to the case where  $F$  has few attributes. Whenever  $F$  is collapsed into  $E$ , we may extend the heuristics to also produce a view that recovers  $F$  exactly as this scheme was before collapsing. Therefore, users still see  $E$  and  $F$  separately, except that  $F$  is now virtual. However, we stress, the underlying database system will have to check one less INDP, that which previously related  $F$  to  $E$ . For example, when applied to the EER schema *COMPANY* of Section 2, this heuristic produces the relational schema  $C$  given in Section 3.

We argue that the collapsing of schemes produces a correct representation by resorting to an example. Recall that, in the EER schema, *COMPANY*, the scheme  $M$  is a specialization of  $S$  and that, in the relational representation  $C$ ,  $M$  is collapsed into  $S$  and stored as a single relation scheme  $S^*$ . The collapsing is correct essentially because each tuple  $t$  of  $S^*$  represents an  $S$ -entity  $s$  and the unique  $M$ -entity  $m$ , if it exists, that corresponds to  $s$  (since  $M$  is a specialization of  $S$ ). Moreover, since the attribute *Id* of  $M$ , inherited from  $E$  via  $A$ , does not admit null values,  $t$  will represent an  $M$ -entity if  $t[Id] \neq \lambda$ ; otherwise  $t$  represents just an  $S$ -entity  $s$ . The definition of view  $M$  in the relational schema  $C$  reflects this point. Furthermore, since deletions from  $S$  and insertions into  $M$  block immediately, by definition of specialization, the translation of the operations on the views  $M$  and  $S$  to operations on  $S^*$ , described in *Tables 1* and *2* of Section 3.2, is indeed correct. That is, the translation of

delete from  $S$  where  $Q$

is

delete from  $S^*$  where  $Q$  and  $Id = \lambda$

just to block the deletion of a tuple from  $S^*$  that represents an  $S$ -entity and the corresponding  $M$ -entity. Likewise, the translation of

insert into  $M$  ( $Code=c, Id=i, Title=t$ )

is

update  $S^*$  set  $Id=i, Title=t$  where  $Code=c$  and  $Id = \lambda$

to block the insertion of the representation of an  $M$ -entity whose corresponding  $S$ -entity is not already represented.

#### 4.2. Design method

To describe the design method, we first introduce the concept of collapsible arc. Let  $S_E$  be an EER schema and let  $g(S_E) = (\mathbf{V}, \mathbf{A}, \mathbf{I})$  be its graph. An arc  $(F, E)$  in  $\mathbf{A}$  is *collapsible* iff  $F$  is a specialization of  $E$  or  $F$  is a relationship scheme which is functional on a role  $N$  that  $E$  plays in  $F$  and  $\mathbf{I}((F, E)) = N$ . A node  $F$  is *collapsible* iff there is a collapsible arc in  $g(S_E)$  leaving  $F$ . If  $(F, G)$  is a collapsible arc then it is possible to represent  $F$  and  $G$  as a single relation scheme, as discussed above.

A forest  $f$  is a *collapsing forest* for  $S_E$  iff the nodes of  $f$  are those of  $g(S_E)$  and  $F$  is a child of  $E$  iff the arc  $(F, E)$  in  $g(S_E)$  is collapsible. The forest is *complete* iff none of its roots is a collapsible node. Thus, a complete collapsible forest for  $S_E$  indicates a form of collapsing schemes of  $S_E$  until no further collapsing is possible.

In what follows we will adopt a multilist representation for forests. For example, the multilist  $f = (A(B), C(D, E(F)), G)$  represents the forest in Fig. 6.

The design method is essentially a systematization of the heuristics outlined in Section 4.1. It adopts as optimization criteria the minimization of the number of inclusion dependencies (or foreign key [14, 15], in other terms) of the relational schema, since these constraints are very expensive to check. The only additional overhead is the generation of new views, which is absorbed at query compile time.

The design method is summarized as follows:

**Input:** an EER schema  $S_E$

**Output:** a relational representation  $S_R$  for  $S_E$  and the corresponding collapsing forest  $f$

**Phase 1:**

Construct a complete collapsing forest  $f$  for  $S_E$ .

**Phase 2:**

**2.1.** Create, using  $f$ , a relational representation  $S_R$  for  $S_E$  according to the following general rules. For each tree  $g$  of  $f$  with root  $G$ :

**2.1.1.** If the tree  $g$  has just the root node  $G$  (which is by definition the name of an entity or relationship scheme), then:

- Generate a relation scheme  $G$  containing the attributes and keys of the scheme named  $G$ ; consider  $G$  as the *explanation* for  $G$ .

**2.1.2.** If the tree  $g$  has more than one node, then:

- Generate a relation scheme  $G^*$  containing the attributes and keys of all the entity or relationship schemes whose names occur in the tree  $g$ ; consider  $g$  as the *explanation* for  $G^*$ ;
- For each node  $H$  of  $g$  (including the root), generate a view  $H$  over  $G^*$  containing all the attributes of the scheme named  $H$ ; consider  $H$  as the *explanation* for the view  $H$ .

**2.2.** For each arc of the graph of  $S_E$  that is not an arc of  $f$ , create an INDP representing the arc.

Note that phase 1 is non-deterministic since there will be more than one complete collapsing forest for  $S_E$ , if there is a scheme  $F$  of  $S_E$  with more than one collapsible arc

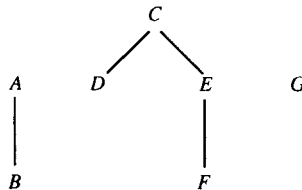


Fig. 6. A forest.

leaving it. This occurs when  $F$  is a specialization of more than one scheme or when  $F$  is a relationship scheme functional on more than one entity scheme. We refer the reader to [9] for a fuller discussion on how to correctly generate relation scheme keys from the entity keys and the relationship identifiers.

We illustrate the method through an example. Suppose that the EER schema *COMPANY* defined in Section 2 is given as input to the method. In phase 1, we first construct a complete collapsing forest for *COMPANY* as follows. The forest initially contains all schemes of *COMPANY* as roots. Then, we select each of the nodes, in an arbitrary order, and collapse in it all possible nodes:

Forest	Selected Node	Collapsed Nodes
$f_0 = (E, S, A, R, M, P, W)$	$E$	$A, R$
$f_1 = (E(A, R), S, M, P, W)$	$S$	$M$
$f_2 = (E(A, R), S(M), P, W)$	$P$	none
(same)	$W$	none
(same)	$A$	none
(same)	$R$	$W$
$f_3 = (E(A, R(W)), S(M), P)$	$M$	none

In phase 2, we generate a relational representation for *COMPANY* from  $f_3$ , which is the relational schema  $C$  described in Section 3. Observe that  $C$  contains just two INDPs, whereas a one-to-one representation of *COMPANY* would contain six INDPs (the number of arcs of the graph of *COMPANY*, shown in Fig. 4). This reduction in the number of INDPs represents a substantial gain in terms of the performance of the update operations. Also note that each entity or relationship scheme of *COMPANY* still corresponds to a view or a relation scheme of  $C$ . Therefore, from the user's point of view,  $C$  is as good as the one-to-one representation.

The additional complexity of  $C$  is twofold. First, the INDPs of  $C$  involve views, but the complexity of checking for violations of such INDPs is no more complex than regular INDPs since the views involved are quite simple. Second, the operations on the views of  $C$  require special translations, as shown in Tables 1 and 2 in Section 3.2. But, again, it is straightforward to implement such translations.

The generation of keys and INDPs, as well as of the defining expressions and the translation tables for views, deserve some comments. First observe that the graph of the EER schema *COMPANY* has three sink nodes,  $P$ ,  $S$  and  $E$ . The primary key of each of these three entity schemes will be inherited by all entity schemes that are connected to them by a path in the graph and used to synthesize the INDPs necessary to represent specializations and relationships. For example, the view scheme  $M$  has *Code* and *Id* as keys since, in the graph of *COMPANY*, there are paths from  $M$  to  $S$  and from  $M$  to  $E$  and since *Code* is the key of  $S$  and *Id* of  $E$ . As  $M$  was collapsed into  $S$ , *Code* is chosen as the primary key of  $M$  and no INDP is generated to capture that  $M$  specializes  $S$ . However,  $M[\text{Id}] \subseteq A[\text{Id}] : (b^i, b^i)$  is generated to capture that  $M$  specializes  $A$ . If  $M$  were collapsed neither in  $S$  nor in  $A$ , one of the keys, *Code* or *Id*, would be arbitrarily chosen as its primary key.

We note that the method produces just two types of views. The first one includes those views, such as  $S$  and  $E$ , that correspond to the roots of the trees of the collapsing forest. Their defining expressions will be simple projections and the translation of their operations will be exactly like in Table 1. The other type contains the views, such as  $M$ ,  $A$  and  $R$ , that correspond to interior nodes of the collapsing forest. Their defining expressions will be a

selection followed by a projection and the translation of their operations will be like in *Table 2*.

In phase 2, we also define the tree ' $E(A, R(W))$ ' as the explanation for the relation scheme  $E^*$ , the tree ' $S(M)$ ' as that for  $S^*$  and ' $P$ ' as that for the relation scheme  $P$ . The explanation for  $E^*$  indicates that it represents the entity schemes  $E$ ,  $A$  and  $R$  and the relationship scheme  $W$ , reflecting the collapsing of  $W$  into  $R$ , since  $W$  is functional on  $R$ , and the collapsing of  $R$  and  $A$  into  $E$ , since both are specializations of  $E$ . The explanation for  $S^*$  in turn indicates that it represents  $S$  and  $M$ , reflecting the collapsing of  $M$  into  $S$ . Note that  $M$  could have been alternatively collapsed into  $A$ . Finally, the explanation for  $P$  tells us that it represents just the entity scheme  $P$ .

We conclude this section with some observations about the implementation of the design method. We may modify phase 1 to enumerate all possible complete collapsing forests for the EER schema given as input and to choose the forest that leads to the optimal representation. However, depending on the EER schema, this approach will generate a large number of forests. Moreover, it requires solving the (serious) problem of defining a quality measure for relational representations. In addition to these problems, there is the objection that not all possible collapsings are in fact desirable. For example, if  $F$  specializes  $E$ , but  $F$  has a large number of attributes, it may not be adequate to collapse  $F$  into  $E$ .

For these reasons, we chose to implement the design method without an exhaustive enumeration of the forests, but permitting the designer to control the collapsing of schemes, if desired, through commands that indicate which collapsings to give preference, when there is more than one alternative, and which to ignore, if performance reasons so suggest. The implementation is part of the database design tool FeST, which generates SQL code for DB2, and is described in detail in [26].

## 5. Automatic modification of relational representations

We describe in Section 5.1 the redesign commands that can be applied to an EER schema and, in Section 5.2, we present the redesign method with the help of a detailed example.

### 5.1. Redesign commands

The redesign method accepts as input an EER schema  $S_E$ , its relational representation  $S_R$  and a sequence  $s$  of redesign commands. It applies  $s$  to  $S_E$ , producing the new EER schema  $S'_E$ , and creates a redesign plan, which is a sequence of operations that maps  $S_R$  into a relational representation  $S'_R$  for  $S'_E$  and which rearranges the current relational database state so that it becomes a state for  $S'_R$ . The sequence  $s$  must be such that, after each command, the resulting EER scheme is correct. For example, it is not possible to remove an entity schema before removing all relationship schemes in which it participates.

Some redesign commands require loading new data to the database as, for example, commands to add a new attribute that does not admit a null value. In such cases, the redesign plan will contain operations that request the loading of new data and that inform the integrity constraints the new data must satisfy. Likewise, some redesign commands require that the database be in a state that satisfies certain conditions to permit the generation of a consistent state for the new relational schema. For example, commands to add a key to an entity scheme fall in this category. The redesign plan will then contain tests to ensure the necessary conditions.

We consider redesign commands to add/remove: an attribute to/from an entity or relationship scheme; a key to/from an entity scheme; an identifier to/from a relationship

Table 3.  
Redesign commands

<b>add/remove</b>	an attribute to/from a scheme; a key to/from an entity scheme; an identifier to/from a relationship scheme; the totality specification to/from a role of a relationship scheme; an entity or relationship scheme to/from the EER schema; a specialization declaration to/from the EER schema.
<b>modify</b>	the name of an attribute; the domain of an attribute; the name of a scheme.

scheme; totality to/from a role of a relationship scheme; an entity or relationship scheme to/from the EER schema; a specialization declaration to/from the EER schema; and commands to modify: the name of an attribute; the domain of an attribute; the name of a scheme. We do not consider the possibility of adding or removing a participant of a relationship scheme since we consider that this destroys the meaning of the relationship and, hence, must be performed by removing the old relationship scheme and adding a new relationship scheme. *Table 3* summarizes the redesign commands.

## 5.2. Redesign method

The redesign method is outlined as follows:

- Input:** an EER schema  $S_E$ ;  
 a relational representation  $S_R$  for  $S_E$  and the corresponding collapsing forest  $f$  (produced by the design method);  
 a sequence  $s$  of redesign commands for  $S_E$ .
- Output:** the new EER schema  $S'_E$ ;  
 a relational representation  $S'_R$  for  $S'_E$  and the new collapsing forest  $g$ ;  
 a *redesign plan* to create  $S'_R$  starting from  $S_R$  and to reorganize the current state  $\sigma$  of  $S_R$  (supposed consistent) into a consistent state  $\sigma'$  of  $S'_R$ .

### Phase 1:

- 1.1. Apply the sequence  $s$  of redesign commands to  $S_E$  producing a new EER schema  $S'_E$ .
- 1.2. If  $S'_E$  is syntactically incorrect, reject the sequence  $s$ , discard  $S'_E$  and stop.
- 1.3. Reorganize the collapsing forest associated with  $S_R$  to reflect the redesign commands.
- 1.4. Initiate the redesign plan with operations to:
  - modify the current relation schemes and view definitions of  $S_R$  to temporarily accommodate the proposed changes;
  - request the loading of the new data required to fulfill the proposed changes;
  - verify if the current state, augmented with the new data, is consistent with the proposed changes.

### Phase 2:

- 2.1. Reorganize the collapsing forest (as in phase 1 of the design method), generating the new forest  $g$  for  $S'_E$ .
- 2.2. Compare the new collapsing forest  $g$  with the old one  $f$ , checking the movement of the nodes. The comparison is used to generate the redesign plan with operations to:
  - map the old representation  $S_R$  into the new representation  $S'_R$  for the new EER schema  $S'_E$ ;

- map the current database state  $\sigma$  associated with  $S_R$  into a consistent database state  $\sigma'$  associated with  $S'_R$ .

A detailed description of step 2.2 is beyond the scope of this paper, but we illustrate below, with the help of an example, how it proceeds.

Suppose that the input consists of the EER schema *COMPANY* defined in Section 2, its relational representation *C* defined in Section 3 and the following sequence  $s$  of redesign commands:

- $s_1$ . **remove** the declaration that  $M$  specializes  $S$
- $s_2$ . **add** a declaration indicating that  $S$  specializes  $E$
- $s_3$ . **remove**  $R$  as an identifier of  $W$

With these inputs, the redesign method will proceed as follows.

First, in phase 1, we apply the redesign commands and generate the new EER schema, whose graph is shown in Fig. 7.

Then, we reorganize the collapsing forest, generating the following sequence of forests:

$$f_0 = f = (E(A, R(W)), S(M), P)$$

$$f_1 = (E(A, R(W)), S, M, P)$$

$$f_2 = (E(A, R), W, S, M, P)$$

Indeed, after processing command  $s_1$ , we transform the initial collapsing forest  $f_0$  into forest  $f_1$  since  $M$  ceased to be a specialization of  $S$  and, hence,  $M$  cannot be collapsed into  $S$  anymore. Command  $s_2$  at this point does not provoke any modification to the forest. But, after processing command  $s_3$ , we transform the forest  $f_1$  into  $f_2$  since  $R$  is no longer an identifier of  $W$ , that is,  $W$  is no longer functional on  $R$ . Hence,  $W$  cannot be collapsed into  $R$  anymore.

Next, in phase 1, we initiate the redesign plan with operations to request the new data. The redesign commands  $s_1$  and  $s_3$  do not generate any operation. However, after processing  $s_2$ , since  $S$  becomes a specialization of  $E$ , it is necessary to identify each  $S$ -entity with an  $E$ -entity. However, as  $M$  specializes both  $S$  and  $A$  (and, so,  $E$ ) in the initial version of the EER schema, *COMPANY*, it is actually necessary to identify only those  $S$ -entities that are not also  $M$ -entities. This is reflected in operation **S2** below.

In detail, the redesign plan begins with the following operations (recall that these operations are not actually executed at this point):

**S0** remove the native attribute `Title` of  $M$  from  $S^*$ ;

**S1** add `Id` as a key of  $S^*$ ;

add  $S^*[Id] \subseteq E^*[Id]$  as a new INDP to the current relational representation;

**S2** for each tuple  $t$  of the relation currently associated with  $S^*$  such that  $t[Id] = \lambda$ , request to set  $t[Id]$  to a non-null value, respecting the dependencies defined in **S1**.

Note that the relation scheme  $S^*$  already contains `Id` as an attribute, hence it is not necessary to include an operation to expand  $S^*$  with this attribute.

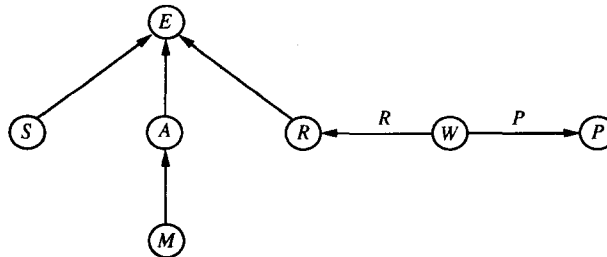


Fig. 7. The graph of the new EER schema.

Now, in phase 2, we reorganize the forest (as in phase 1 of the design method), generating:

$$f_3 = (E(S, A, R), W, M, P)$$

$$f_4 = (E(S, A(M), R), W, P) = g$$

As we can see, forest  $f_2$  was transformed into  $f_3$  since  $S$  became a specialization of  $E$  and, hence,  $S$  can now be collapsed into  $E$ . To justify  $f_4$ , observe that  $M$  can now be collapsed into  $A$  since  $M$  is now a specialization of  $A$  and  $M$  is not collapsed into  $S$  anymore.

During the second step of phase 2, we compare the initial forest  $f_0$  with the final forest  $f_4$ , detecting the following movement of the nodes:

$E, A, R, P$  – unaltered

$S$  – transformed from root to interior node

$S$  – transformed from interior node to root

$M$  – maintained as interior node, but in a different tree

We will now detail the generation of the rest of the redesign plan, analysing each of the nodes that was moved in the collapsing forest. In fact, the method simultaneously considers all nodes of each subtree that was moved. Thus, if a node  $N$  is a descendent of a node  $N'$  that was moved,  $N$  is processed together with  $N'$ . This more general situation will not be apparent in the example that follows. Furthermore, when applying the method, some of the operations generated by the processing of a node may be deferred to a later point in the plan, as illustrated below.

Let us begin with node  $S$ . Since  $S$  moved from being a root to being an interior node of the tree with root  $E$ , we have that  $S$  must be collapsed into  $E$  or, in terms of the relational representation,  $S$  must become a view over the relation scheme  $E^*$ . Therefore, the method adds to the redesign plan the following operations:

**S3** expand  $E^*$  with the attributes of  $S$ ;

**S4** update  $E^*$  as follows:

for each tuple  $t$  of the relation associated with  $E^*$ , if there is a tuple  $u$  of the relation associated with  $S^*$  such that  $t[\text{Id}] = u[\text{Id}]$ , then set  $t[\text{Code}] = u[\text{Code}]$  and  $t[\text{Stocks}] = u[\text{Stocks}]$ , else set  $t[\text{Code}] = \lambda$  and  $t[\text{Stocks}] = \lambda$ .

To complete the collapsing of  $S$  into  $E$  it is necessary to execute the following operations:

**S5** remove the relation associated with  $S^*$  from the database;

**S6** remove the relation scheme  $S^*$  and all dependencies over  $S^*$  from the relational schema;

**S7** remove  $S$  as a view over  $S^*$  from the relational schema;

**S8** add  $S$  as a view over  $E^*$  to the relational schema;

Operations **S5** and **S6** must be placed after the operations generated by the processing of  $M$  since  $M$  is also a view over  $S^*$ . Alternatively, these operations could be placed at the end of the plan by a process that eliminates all relation schemes that do not participate in view definitions.

Let us now consider  $M$ . Observe that  $M$  moved from the tree with root  $S$  to the tree with root  $E$ . This means that  $M$  must be removed from  $S$  and collapsed into  $E$  or, in terms of the relational representation,  $M$  must become a view over the relation scheme  $E^*$ . Hence, it is necessary to add to the redesign plan the following operations:

**M1** expand  $E^*$  with the attributes of  $M$ ;

**M2** update  $E^*$  as follows:

for each tuple  $t$  of the relation associated with  $E^*$ , if there is a tuple  $u$  of the relation associated with  $M$  such that  $t[\text{Id}] = u[\text{Id}]$ , then set  $t[\text{Title}] = u[\text{Title}]$  else set  $t[\text{Title}] = \lambda$ .

Note that, since  $M$  was already a specialization of  $A$ , and hence transitively of  $E$ , view  $M$  already contained the primary key  $\text{Id}$  of  $E$ , by definition of the relational representation.

To complete the collapsing of  $M$  into  $E$  it is also necessary to execute the following operations:

**M3** remove  $M$  as a view over  $S^*$  from the relational schema;

**M4** add  $M$  as a view over  $E^*$  to the relational schema;

Observe that the operations **S5** and **S6** may now be included in the redesign plan since no view over  $S^*$  remains.

Finally, consider node  $W$ , that moved from being an interior node of the tree with root  $E$  to being a root. This means that  $W$  must be removed from  $E$  or, in terms of the relational representation, view  $W$  must be materialized:

**W1** remove  $W$  as a view over  $E^*$  from the relational schema;

**W2** add  $W$  as a relation scheme to the relational schema;

**W3** populate the relation associated with  $W$  using the data stored in the relation associated with  $E^*$  and the original definition of  $W$  as a view over  $E^*$ ;

**W4** remove all attributes of  $W$  from the relation scheme  $E^*$  and drop the corresponding columns from the relation associated with  $E^*$ .

This concludes the redesign example. The final redesign plan will then be:

**S0;S1;S2;S3;S4;S7;S8;M1;M2;M3;M4;S5;S6;W1;W2;W3;W4**

We stress that these operations are not applied as they are generated. Rather, they are stored and passed to the database designer for inspection. When the designer decides to modify the database, he will probably stop operation of the system, at least in part, gradually load the required data and control the application of the operations in the plan. Finally, the designer may also use the redesign plan just to assess the impact the conceptual changes he is proposing will have on the operational data.

## 6. Conclusions

We outlined in this paper two methods that, together, provide a systematic way for the design and maintenance of conceptual schemas and their optimized representations in terms of an implementation model. The methods form the basis of a tool to help develop database applications. We adopted an extension of the entity-relationship model for conceptual modeling and the relational model as the implementation model.

As already observed in Section 3, there is a straightforward translation of the neutral syntax introduced for relation schemes, view declarations and integrity constraints into the standard SQL language, which means that the methods can be refined to algorithms that generate code for a large family of commercially available relational systems.

We can also generalize the design and redesign methods to cover other extensions of the entity-relationship model, if the target variation of the relational model is sufficiently expressive, as discussed in [9, 19].

An algorithm based on the design method was implemented, with the characteristics described at the end of Section 4.2. The implementation is part of the database design tool FeST, which generates SQL code for DB2, and is described in detail in [26]. This tool will now be extended to support the redesign method.

## Acknowledgements

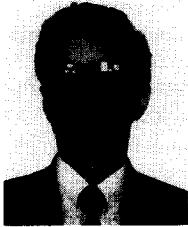
This work is part of a joint research project conducted by the Rio Scientific Center of IBM Brazil and the Computer Science Department of the Federal University of Minas Gerais. We

would also like to thank Alexandre Plastino Carvalho, Lorenzo Ridolfi, Maurício Roma Cavalcanti and José Carlos Magno for the implementation of the FeST design tool.

## References

- [1] ANSI X3H2-90-264/ISO IEC JTC1 SC21 WG3, (ISO working draft), J. Melton, ed., Database Language SQL2 (1990).
- [2] A. Atri and D. Sacca, Equivalence and mapping of database schemes, *Proc. 10th Internat. Conf. on Very Large Data Bases* (Singapore, Sep. 1984).
- [3] N. Azar and E. Pichat, Translation of an extended entity-relationship model into the universal relation with inclusion formalism, *Proc. 5th Internat. Conf. on the Entity-Relationship Approach* (Dijon, Nov. 1986).
- [4] C. Batini, S. Ceri and S.B. Navathe, *Conceptual Database Design – An Entity-Relationship Approach* (Benjamin Cummings, Menlo Park, CA, 1992).
- [5] M.N. Bert, G. Ciardo, B. Demo et al., The logical design in the DATAID project: The Easymap System, in A. Albano, V. De Antonellis and A. Di Leva, eds., *Computer-Aided Database Design: The DATAID Project* (North-Holland, Amsterdam, 1985).
- [6] P. Bertaina, A. Di Leva and P. Giolito, Logical design in Codasyl and relational environment, in S. Ceri, ed., *Methodology and Tools for Data Base Design* (North-Holland, Amsterdam, 1983).
- [7] M. Bouzeghoub and E. Métais, Semantic modeling of object-oriented databases, *Proc. 17th Internat. Conf. on Very Large Data Bases* (Barcelona, Sep. 1991).
- [8] H. Briand, H. Habrias, J-F. Hue and Y. Simon, Expert system for translating an E-R diagram into databases, *Proc. 4th Internat. Conf. on the Entity-Relationship Approach* (Chicago, Oct. 1985).
- [9] M.A. Casanova, A.P. de Carvalho, L. Ridolfi and A.H.F. Laender, An analysis of table constraints in SQL2 based on the entity-relationship model, *Proc. 10th Internat. Conf. on the Entity-Relationship Approach* (San Mateo, Oct. 1991).
- [10] M.A. Casanova, A.L. Furtado and L. Tucherman, Enforcing inclusion dependencies and referential integrity, *Proc. 14th Internat. Conf. on Very Large Data Bases* (Los Angeles, Aug. 1988).
- [11] M.A. Casanova, L. Tucherman, A.L. Furtado and A. Pacheco, Optimization of relational schemas containing inclusion dependencies, *Proc. 15th Internat. Conf. on Very Large Data Bases* (Amsterdam, Sep. 1989).
- [12] M.A. Casanova, L. Tucherman, P.M. Gualandi, A. Pacheco and M.R. Cavalcanti, A data definition language for an extended entity-relationship model, Technical Report CCR072, Rio Scientific Center, IBM Brazil, July 1989.
- [13] P.P. Chen, The entity-relationship model: Toward a unified view of data, *ACM Trans. Database Syst.* 1(1) (1976).
- [14] C.J. Date, *A Guide to the SQL Standard* (Addison-Wesley, Reading, MA, 1987).
- [15] R. Elmasri and S. Navathe, *Fundamentals of Database Systems* (Benjamin Cummings, Menlo Park, CA, 1989).
- [16] A.L. Furtado and M.A. Casanova, Updating relational views, in W. Kim, D.S. Reiner and D.S. Batory, eds., *Query Processing in Database Systems* (Springer, Berlin, 1985).
- [17] P.P. Gualandi, Maintenance of inclusion dependencies in relational databases, *Proc. 9th Congress of the Brazilian Computer Society* (Uberlândia, 1989) – in Portuguese.
- [18] International Standard ISO 9075: 1992, Database Language SQL, April 1991.
- [19] A.H.F. Laender, M.A. Casanova, A.P. de Carvalho and L.F. Ridolfi, An analysis of SQL integrity constraints from an entity-relationship model perspective, in preparation.
- [20] C.M.R. Leung and G.M. Nijssen, Relational database design using the NIAM conceptual schema, *Informat. Syst.* 13(2) (1988).
- [21] V.M. Markowitz and A. Shoshani, Representing extended entity-relationship structures in relational databases: a modular approach, *ACM Trans. Database Syst.* 17(3) (1992).
- [22] G.M. Nijssen and T. Halpin, *Conceptual Schema and Relational Database Design: A Fact-Oriented Approach* (Prentice Hall, Sidney, 1989).
- [23] D. Reiner, M. Brodie, G. Brown et al., The database design and evaluation workbench (DDEW), *IEEE Database Engrg.* 7(4) (Dec. 1984).
- [24] C. Rolland and C. Proix, An expert system approach to information system design, in H.J. Kugler, ed., *Information Processing 86* (North-Holland, Amsterdam, 1986).
- [25] A. Pacheco, Correct translation of entity-relationship schemas into relational schemas, *Proc. 9th Congress of the Brazilian Computer Society* (Uberlândia, July 1989) – in Portuguese.
- [26] L. Ridolfi, A.P. Carvalho and M.A. Casanova, A tool for conceptual database design based on the entity-relationship model, Technical Report CCR130, Rio Scientific Center, IBM Brazil, Nov. 1991, in Portuguese.

- [27] T.J. Teorey, D. Yang and J.P. Fry, A logical design methodology for relational databases using the extended entity-relationship model, *ACM Comput. Surv.* 18(2) (June 1986).
- [28] L. Tucherman, M.A. Casanova and A.L. Furtado, The CHRIS Consultant - A tool for database design and rapid prototyping, *Informat. Syst.* 15(2) (May 1990).
- [29] L. Tucherman, M.A. Casanova, P.M. Gualandi and A. Pacheco, Generalization and subset abstractions in the entity-relationship model, *Proc. 8th Internat. Conf. on Entity-Relationship Approach* (Toronto, Oct. 1989).



**Marco Antonio Casanova** has a Ph.D. in Applied Mathematics from Harvard University (1979), an M.Sc. in Computer Science from the Pontifical Catholic University of Rio de Janeiro (1976), and a B.Sc. in Electronic Engineering from the Military Institute of Engineering (1974). He joined the IBM Brazil Scientific Center in November 1982, where he is now Research Manager.

From 1980 to 1982, he was Assistant Professor at the Department of Informatics of the Catholic University in Rio, where he also acted as Graduate Program Coordinator during 1981-82.

He is author of the book *The Concurrency Control Problem for Database Systems*, published by Springer-Verlag, and co-authored the books (in Portuguese) *Principles of Distributed Database Systems*, *Logic Programming and Prolog* and *Fundamentals of Multimedia Systems*. He also published several technical papers in international scientific journals. His academic interests include database management systems and logic programming.



**Luiz Tucherman** is Operations Manager of the Education/Research/Health Marketing Sector of IBM Brazil. Until January 1992, he was Senior Researcher at the Rio Scientific Center. He graduated in Building and Electrotechnic Engineering at the Federal University of Juiz de Fora and obtained the D.Sc. and M.Sc. degrees, both in Computer Science, from the Pontifical

Catholic University of Rio de Janeiro in 1986 and 1983, respectively. His academic interests include database management systems and tools and methodologies for database design. He has also published several papers in both national and international conferences and technical journals.



**Alberto H.F. Laender** received the B.S. degree in Electrical Engineering and the M.Sc. degree in Computer Science from the Federal University of Minas Gerais, Belo Horizonte, Brazil, in 1974 and 1979, respectively, and the Ph.D. degree in Computing from the University of East Anglia, Norwich, England, in 1984. He is a Professor in the Computer Science Department at

the Federal University of Minas Gerais, where he got his first appointment in 1975. His research interests include database design methodologies, database end-user interfaces and parallel and distributed databases. Prof. Laender is a senior member of the Brazilian Computer Society and is currently a member of the Editorial Board of the Brazilian Computer Journal (*Revista Brasileira de Computacao*).