

# Designing and Maintaining Optimized DB2 Representations of Entity-Relationship Schemas

Marco A. Casanova and Luiz Tucheran

Rio Scientific Center  
IBM Brazil  
P.O. Box 4624  
20.001 Rio de Janeiro RJ, Brazil

## Abstract

An algorithm for obtaining optimized relational representations of database conceptual schemas in an extended entity-relationship model is first proposed. The algorithm 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 algorithm that, given changes to the conceptual schema, generates a plan to modify the original representation and to organize the database state is described.

## 1. INTRODUCTION

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 design algorithm that accepts as input an entity relationship conceptual schema and generates an optimized relational representation for the schema (optimized in the sense that the number of dependencies of the relational schema is minimized [8]). The representation includes explanations indicating which objects of the conceptual schema each relation scheme represents and why. The algorithm generalizes an optimization heuristics that is commonly adopted and it operates based exclusively on the structure of the conceptual schema.

The second contribution is a redesign algorithm that accepts as input a conceptual schema, the relational representation for the schema produced by the design algorithm 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 database state accordingly.

To understand the redesign algorithm, consider the following scenario (the numbers between parenthesis below relate each step of the process with a mapping shown in Figure 1). Suppose that the database designer defines a conceptual schema  $E$ , which the design algorithm maps (1) into a relational representation  $R$ . Assume now that the designer defines a set of changes to  $E$  and that the database currently is in a state  $\sigma$  that conforms (2) to the relational representation  $R$ . The redesign algorithm will then process the changes to map (3)  $E$  into the new conceptual schema  $E'$  and to produce a redesign plan that will take (4)  $R$  into the new relational representation  $R'$  and that will restructure (5)  $\sigma$  into the new state  $\sigma'$ . The difficulties lie in that the redesign plan must be correct, in the sense that

- $\sigma'$  must conform (7) to the new representation  $R'$ ;
- $R'$  must correctly and optimally represent (6)  $E'$ . In particular, note that some changes, when applied to  $E$ , may invalidate previous optimizations that  $R$  reflects, while others may create opportunities for new optimizations that  $R'$  must take into account.

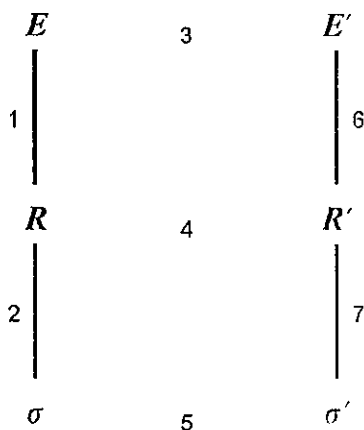


Figure 1. The Redesign Process

The use of variations of the entity-relationship model [9] for database conceptual design has been extensively investigated (see, e.g., [19] for a survey). The idea of transforming an extended entity-relationship schema into a relational schema is not new

[1,2,3,12,15,18]. 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 [4]. Research on expert tools for database design can be found in [5,8,16,17,20]. In particular, the design algorithm we proposed in [8] 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 [6,7,8,14,18,21].

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 algorithm, whereas section 5 addresses the redesign algorithm. 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 [6]. 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. However, 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):

```
define entity E [attributes  $A_1 D_1, \dots, A_n D_n$ ] [key  $K_0$  [... key  $K_p$ ]]
```

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  $D_i$  is the domain of  $A_i$  and that  $A_i$  *accepts null values*, if  $D_i$  does, for  $i=1, \dots, n$ .

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 frame 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], ...,  $O_m$  [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  $i^{\text{th}}$  *participant* is  $O_i$  and the  $i^{\text{th}}$  *role* is  $N_i$ , if specified, otherwise it is  $O_i$ , by default. Therefore, the  $i^{\text{th}}$  role acts as an alias for the  $i^{\text{th}}$  participant. When "total" is specified for " $O_i$  [as  $N_i$ ]", we say that scheme is *total* on the  $i^{\text{th}}$  participant (or role).

For each  $i=0, \dots, p$ ,  $I_i$  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  $i^{\text{th}}$  participant (or role) iff the  $i^{\text{th}}$  role is an identifier of *R*.

The entity schemes may be organized as an acyclic specialization graph. Hence, the concept of specialization is not restricted to hierarchies in our model. If an entity scheme *F* is defined as a specialization of *E*, then *F* must always denote a subset of the set of entities associated with *E*. Lastly, an entity scheme *F* inherits the attributes and keys of all schemes it specializes, directly or transitively (to avoid conflicts, when inherited, attributes with the same name are qualified with the name of the scheme they originate from).

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

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

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

- $O$  is a specialization of  $P$  in  $E$ , in which case  $I((O, P))$  is not defined, or
- $O$  is a relationship scheme of  $E$  such that  $P$  participates with role  $N$ , in which case  $I((O, P)) = N$ , or
- $P$  is a relationship scheme of  $E$  such that  $O$  participates with role  $N$  and  $P$  is total on  $N$ , in which case  $I((O, P)) = N$ .

This concludes the description of the syntax of the model. The semantics is standard and can be found in [6]. In particular, let  $\sigma$  be a state of an EER schema  $E$ . 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 algorithm 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 a  $R$ -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.  $E$  (resp. deferredly w.r.t.  $E$ ) means to reject the insertion when submitted (resp. at the end of the transaction), if the  $E$ -entity  $e$  that  $r$  refers to does not exist. To propagate the deletion of a  $R$ -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 a relationship  $r$  into  $R$  does not determine the attributes of the participant entities. Likewise, the insertion of an entity  $e$  into  $E$  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 entity  $e$  from  $E$  is possible only if  $e$  does not occur in  $F$  and the insertion of an entity  $f$  into  $F$  is accepted only if it already occurs in  $E$ . The general problem of updating specialization hierarchies is discussed in [21].

## 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* defines 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* associates each researcher to the single major research project he is assigned to, which is indicated by defining *RESEARCHER* as the identifier of *WORKS*.

### The EER Schema COMPANY

```

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

```

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 graph of the EER schema *COMPANY* is shown in Figure 2 below. 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 participates in a relationship in more than one role.

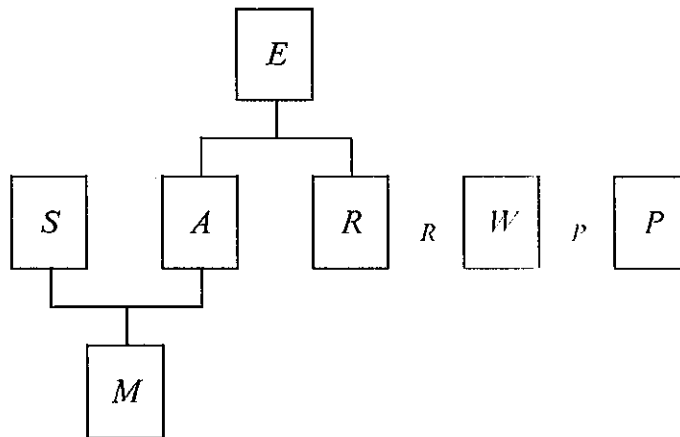


Figure 2: The graph of the EER schema *COMPANY*

### 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. Whenever necessary, we will use SQL-like data manipulation language statements [11], without formally defining their syntax or semantics.

#### 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  $D_i$  is the *domain* of  $A_i$  and that  $A_i$  *accepts null values* if  $D_i$  does, for  $i=1, \dots, n$ . The attributes of the primary key must not accept null values, but those of a 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  $D_i$  is the *domain* of  $A_i$  and that  $A_i$  *accepts null values* if  $D_i$  does, for  $i=1, \dots, n$ . 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 [13] for the operations over  $V$  into operations over the schemes in  $\mathbf{R}$ . We refer the reader to [8] for a detailed discussion about the need to consider view opera-

tion translations.

We also introduce a class of inclusion dependencies [8] 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  $\mathbf{R}$ ;
- $X_i$  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  $\mathcal{S} = (\mathbf{R}, \mathbf{V}, \mathbf{I})$  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  $\mathbf{RUV}$ .  $\mathbf{I}$  is a set of INDs over  $\mathbf{RUV}$ .

A *state*  $\sigma$  for  $\mathcal{S}$  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  $t = u$ ;  $\sigma$  *satisfies* a 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 a 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  $S$ .

The semantics of  $T_1[X_1] \sqsubseteq T_2[X_2]:(\gamma, \delta)$  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 [7]. 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  $\sigma(T_1[X_1]) \not\sqsubseteq \sigma(T_2[X_2])$ . 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  $\sigma(T_1[X_1]) \not\sqsubseteq \sigma(T_2[X_2])$ . 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 keys in the view schemes below 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 state of 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,  $t = u$ .

Furthermore, observe that the view attribute domains follow from the domains of the underlying relation schemes via the defining expressions.

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

```

                                Relational Schema C
/*
  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)
*/
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)

```

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 $\neq \lambda$

#### 4. AUTOMATIC SYSTHESIS OF RELATIONAL REPRESENTATIONS

We describe in section 4.1 the basic idea behind the design algorithm. In section 4.2, we present the overall structure of the algorithm 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  $E$  [18] iff  $R$  contains a distinct relation scheme (with the appropriate attributes) representing each entity or relationship scheme of  $E$  and a distinct IND 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 INDs that are expensive to check for violations.

To reduce the number of INDs of a relational representation, a fairly common heuristics 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 heuristics applies as well when  $F$  is an entity scheme that specializes  $E$ , possibly restricted to the case where  $F$  has few attributes. The design algorithm we describe in this section is essentially a systematization of this heuristics. For example, when applied to the EER schema *COMPANY* of section 2, the design algorithm will produce 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 a *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 a *M*-entity if  $t[Id] \neq \lambda$ ; otherwise *t* represents just a *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 a *S*-entity and the corresponding *M*-entity. Likewise, the translation of

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

is

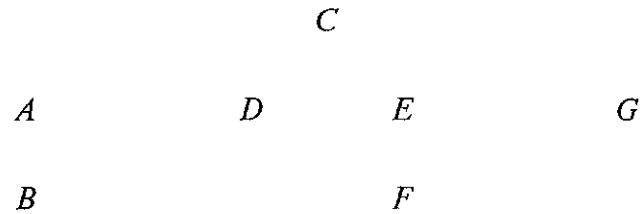
```
update S* set Id=i, Title=t where Code=c and Id= $\lambda$ 
```

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

To describe the design algorithm, we first introduce the concept of collapsable arc. Let *E* be an EER schema and let  $g(E) = (V, A, I)$  be its graph. An arc  $(F, E)$  in *A* is *collapsable* 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  $I((F, E)) = N$ . A node *F* is *collapsable* iff there is a collapsable arc in  $g(E)$  leaving *F*. If  $(F, G)$  is a collapsable 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 *E* iff the nodes of *f* are those of  $g(E)$  and *F* is a child of *E* iff the arc  $(F, E)$  in  $g(E)$  is collapsable. The forest is *complete* iff none of its roots is a collapsable node. Thus, a complete collapsable forest for *E* indicates a form of collapsing schemes of *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



## 4.2 Design Algorithm

The design algorithm is the following:

**Input:** an EER schema  $E$

**Output:** a relational representation  $R$  for  $E$

**Step 1:**

- construct a complete collapsing forest  $f$  for  $E$ .

**Step 2:**

- create, using  $f$ , a relational representation  $R$  for  $E$  according to the following general rules. For each tree  $g$  of  $f$  with root  $G$ :
  - if the tree has just the root node then:
    - generate a relation scheme  $G$  containing the attributes and keys of  $G$ ; consider " $G$ " as the explanation for  $G$ ;
  - if the tree has more than one node then:
    - for each node  $H$  of  $g$  (including the root), generate a view  $H$  over  $G^*$  containing all the attributes of  $H$ ; consider the scheme name  $H$  as the explanation for  $H$ ;
- for each arc of the graph of  $E$  that is not an arc of  $f$ , create an IND representing the arc.

Note that step 1 is non-deterministic since there will be more than one complete collapsing forest for  $E$ , if there is a scheme  $F$  of  $E$  with more than one collapsable arc 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 illustrate the processing of the algorithm through an example. Suppose that the EER schema  $COMPANY$  defined in section 2 is given as input to the algorithm. Then, step 1 constructs a complete collapsing forest for  $COMPANY$  as follows. The forest initially contains all schemes of  $COMPANY$  as roots. Then, the algorithm selects each of the nodes, in an arbitrary order, and collapses 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

Step 2 of the algorithm then generates a relational representation for *COMPANY* from  $f_3$ , which is the relational schema  $C$  described in section 3. The following paragraphs contain observations about the structure of  $C$  that in fact apply to all relational representations that the design algorithm generates.

Observe that  $C$  contains just two INDs, whereas a one-to-one representation of *COMPANY* would contain six INDs (the number of arcs of the graph of *COMPANY*, shown at the end of section 2). This reduction in the number of INDs to check 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 INDs of  $C$  involve views, but the complexity of checking for violations of such INDs is no more complex than regular INDs 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 translation tables.

The generation of keys and INDs, 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 INDs 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  $G$  to  $S$  and from  $G$  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 IND is generated to capture that  $M$  specialize  $S$ . However,  $M[Id] \subseteq A[Id]:(b^i, b^i)$  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.

There are just two types of views in the relational representation the algorithm produces. The first type of views includes those, 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 their translation tables will be exactly like Table 1 in section 3.2. The other type are the views, such as  $M$ ,  $A$  and  $R$ , that correspond to interior nodes of the collapsing forest. Their defining expressions will be a restriction followed by a projection and their translation tables will be like Table 2 in section 3.2.

Step 2 also defines the tree " $E(A,R(W))$ " as the explanation for the relation scheme  $E^*$ , " $S(M)$ " that for  $S^*$  and " $P$ " 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 algorithm. We may modify step 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 algorithm 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.

## 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 algorithm together with a detailed example.

## 5.1 Redesign Commands

The redesign algorithm accepts as input an EER schema  $E$ , its relational representation  $R$  and a sequence  $s$  of redesign commands. It will apply  $s$  to  $E$ , producing the new EER schema  $E'$ , and a redesign plan to map  $R$  into a relational representation  $R'$  for  $E'$  and to rearrange the current database state to become a state for  $R'$ . The sequence  $s$  must be such that, after each command, the resulting EER schema 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 to request the loading of new data and to 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 a scheme; a key to/from an entity scheme; an identifier to/from a relationship 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.

We now briefly outline the redesign commands:

### **add/remove**

- an attribute to/from a scheme
- a key to/from an entity scheme
- an identifier to/from a relationship 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

## 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.

## 5.2 Redesign Algorithm

The redesign algorithm is, in outline, the following:

### Input:

- an EER schema  $E$
- a relational representation  $R$  for  $E$  (produced by the design algorithm)
- a sequence  $s$  of redesign commands for  $E$

### Output:

- the new EER schema  $E'$
- a relational representation  $R'$  for  $E'$
- a *redesign plan* to create  $R'$  starting from  $R$  and to reorganize the current state  $\sigma$  of  $R$  (supposed consistent) into a consistent state  $\sigma'$  of  $R'$

### Step 1:

- analyse the redesign commands in  $s$ , verifying if the new EER schema they produce is correct;
- produce the new EER schema  $E'$ ;
- reorganize the collapsing forest associated with  $R$  to reflect the redesign commands;
- initiate the redesign plan with operations to:
  - modify the current relation schemes and view definitions of  $R$  to temporarily accommodate the proposed changes;
  - request the loading of the new data required to fulfill the proposed changes;
  - verify if the the current state, augmented with the new data, is consistent with the proposed changes.
- reprocess step 1 of the design algorithm to continue the reorganization of the collapsing forest.

**Step 2:**

- compare the new collapsing forest with the old one to continue the generation of the redesign plan with operations to:
  - map the old representation  $R$  into the new representation  $R'$  for the new EER schema  $E'$ ;
  - map the current state  $\sigma$  of  $R$  into a consistent state  $\sigma'$  of  $R'$ .

To illustrate the redesign algorithm, 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 algorithm proceeds as follows:

Step 1 first applies the redesign commands and generates the new EER schema, whose graph is shown in Figure 4.

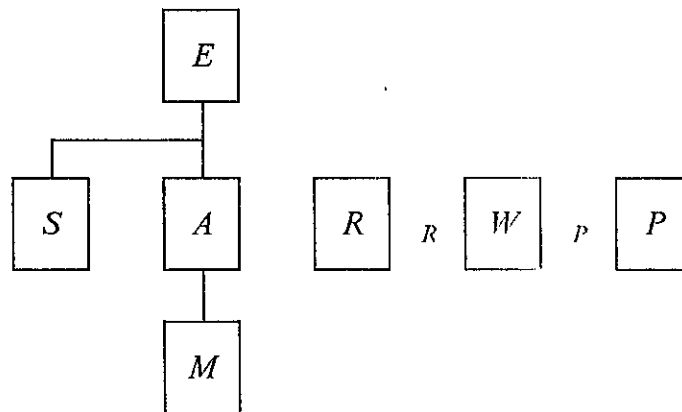


Figure 4: The graph of the new EER schema

Step 1 then reorganizes the collapsing forest, generating the following sequence of forests:

$$f_0 = (E(A, R(W)), S(M), P)$$
$$f_1 = (E(A, R(W)), S, M, P)$$
$$f_2 = (E(A, R), W, S, M, P)$$

Indeed, after processing command  $s_1$ , the algorithm transforms 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  $s_3$ , the algorithm transforms 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.

Step 1 next initiates 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,  $E$ ), it is actually necessary to identify only those  $S$ -entities that are not also  $M$ -entities. This is reflected in operation **S2** below.

Step 1 will then initiate the redesign plan with the following operations (recall that these operations are added to the plan, but not actually executed at this point):

**S1** add  $Id$  as a key of  $S^*$  and  $S^*[Id] \subseteq E^*[Id]$  as a new IND 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.

Finally, step 1 of the redesign algorithm continues to reorganize the forest (as in step 1 of the design algorithm), generating:

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

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

The algorithm transforms forest  $f_2$  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 first phase of step 2, the algorithm compares 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

$W$  - 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 algorithm simultaneously processes 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, the algorithm may place some of the operations generated by the processing of a node after the operations generated by other nodes it will later process, 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 algorithm 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:

**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 algorithms that, together, form the basis of a tool to help develop database applications. The algorithms cover the design and maintenance of conceptual schemas and their representations in terms of an implementation model. We adopted an extension of the entity-relationship model for conceptual modeling and the relational model as the implementation model.

The design algorithm is completely implemented, with the characteristics described at the end of section 4.2. The redesign algorithm is completely specified and its implementation is planned for the near future.

## ACKNOWLEDGEMENTS

This work is part of a joint research project signed between 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 Anelise Pacheco for the implementation of the design algorithm.

## REFERENCES

- [1] Atri, A. and Sacca, D., "Equivalence and mapping of database schemes", Proc. 10th Int'l. Conf. on Very Large Data Bases, Singapore (Sept. 1984).
- [2] Azar, N. and Pichat, E., "Translation of an extended entity-relationship model into the universal relation with inclusion formalism", Proc. 5th Int'l. Conf. on the Entity-Relationship Approach, Dijon (Nov. 1986).
- [3] Bert, M.N., Ciardo, G., Demo, B. et alli, "The logical design in the DATAID project: The Easymap System", in *Computer-aided database design: the DATAID project*, A. Albano, V. De Antonellis and A. Di Leva (eds.), North-Holland (1985).
- [4] Bertaina, P., Di Leva, A., Giolito, P., "Logical design in Codasyl and relational environment", in *Methodology and Tools for Data Base Design*, S. Ceri (Ed.), North-Holland (1983).
- [5] Briand, H., Habrias, H., Hue, J-F. and Simon, Y., "Expert system for translating an E-R diagram into databases", Proc. 4th Int'l. Conf. on the Entity-Relationship Approach, Chicago (Oct. 1985).
- [6] Casanova, M.A., Tucherman, L., Gualandi, P.M., Pacheco, A. and Cavalcanti, M.R., "A data definition language for an extended entity-relationship model", Technical Report CCR072, Rio Scientific Center, IBM Brazil (July 1989).

- [7] Casanova, M.A., Furtado, A.L. and Tucherman, L., "Enforcing inclusion dependencies and referential integrity", Proc. 14<sup>th</sup> Int'l. Conf. on Very Large Data Bases, Los Angeles (Aug. 1988).
- [8] Casanova, M.A., Tucherman, L., Furtado, A.L. and Pacheco, A., "Optimization of relational schemas containing inclusion dependencies", Proc. 15<sup>th</sup> Int'l. Conf. on Very Large Data Bases, Amsterdam (Sept. 1989).
- [9] Chen, P.P., "The entity-relationship model: toward a unified view of data", ACM Transactions on Database Systems 1:1 (1976).
- [10] Chung, I., Nakamura, F. and Chen, P.P., "A decomposition of relations using the entity-relationship approach", in *Entity-relationship approach to information modeling and analysis*, P.P. Chen (Ed.), North-Holland (1981).
- [11] Date, C.J., *A guide to the SQL standard*, Addison-Wesley Pub. Co. (1987).
- [12] Elmasri, R. and Navathe, S., *Fundamentals of Database Systems*, Benjamin Cummings (1989).
- [13] Furtado, A.L. and Casanova, M.A., "Updating relational views", in *Query Processing in Database Systems*, W. Kim, D.S. Reiner and D.S. Batory (eds.), Springer Verlag (1985).
- [14] Gualandi, P.M., "Manutencao de dependencias de inclusao em bancos de dados relacionais", Proc. of the 9<sup>th</sup> Congress of the Brazilian Computer Society, Uberlandia (July 1989).
- [15] Markowitz, V.M. and Shoshani, A., "On the correctness of representing extended entity-relationship structures in the relational model", Proc. ACM SIGMOD International Conference on the Management of Data, Portland, Oregon (June 1989).
- [16] Reiner, D., Brodie, M., Brown G. et alli, "The database design and evaluation workbench (DDEW)", IEEE Database Engineering 7:4 (Dec. 1984).
- [17] Rolland, C. and Proix, C., "An expert system approach to information system design", in *Information Processing 86*, H.J. Kugler (ed.), North-Holland (1986).
- [18] Pacheco, A., "Traducao correta de esquemas entidade-relacionamento em esquemas relacionais", Proc. of the 9<sup>th</sup> Congress of the Brazilian Computer Society, Uberlandia (July 1989).
- [19] Teorey, T.J., Yang, D. and Fry, J.P., "A logical design methodology for relational databases Using the extended entity-relationship model", ACM Computing Survey 18:2 (June 1986).
- [20] Tucherman, L., Casanova, M.A., Furtado, A.L., "The CHRIS consultant - A tool for database design and rapid prototyping", Information Systems (to appear).
- [21] Tucherman, L., Casanova, M.A., Gualandi, P.M. and Braga, A.P., "Generalization and subset abstractions in the entity-relationship model", Proc. 8<sup>th</sup> Int'l. Conf. on Entity-Relationship Approach, Toronto (Oct. 1989).