

|ER90|1  
|072|

# Algorithms for designing and maintaining optimized relational representations of entity-relationship schemas

Marco A. Casanova<sup>1</sup>, Luiz Tucherman<sup>1</sup> and Alberto H.F. Laender<sup>2</sup>

<sup>1</sup>Rio Scientific Center, IBM Brazil,  
P.O. Box 4624, 20.001 Rio de Janeiro RJ, Brazil

<sup>2</sup>Computer Science Department, Federal University of Minas Gerais,  
P.O. Box 702, 30.161 Belo Horizonte MG, 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 [7]). 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 rela-

tional representation for the new schema and to restructure the database state accordingly.

The use of variations of the entity-relationship model [8] for database conceptual design has been extensively investigated (see, e.g., [15] for a survey). The idea of transforming an extended entity-relationship schema into a relational schema is not new [1,2,10,12]. 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 [3]. Research on expert tools for database design can be found in [4,7,13,14,16]. In particular, the design algorithm we proposed in [7] 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 [5,6,7,17].

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 this section 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 [5].

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 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 defined through *specialization declarations* 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 [5]. 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 conclude with the definition of the EER schema *COMPANY*, used 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*.

```

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

```

Figure 1: The EER Schema COMPANY

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.

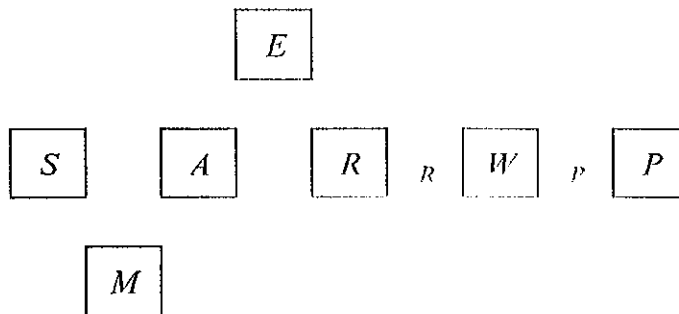


Figure 2: The graph of the EER schema *COMPANY*

### 3. REVIEW OF THE RELATIONAL MODEL

We summarize in this section the concepts of the relational model we will need in the next sections and illustrate them with an example.

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 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  $D_i$  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 [11] for the operations over  $V$  into operations over the schemes in  $\mathbf{R}$ . We refer the reader to [7] for a detailed discussion about the need to consider view operation translations.

We also introduce a class of inclusion dependencies [7] 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*, or an *IND*, over  $\mathbf{R}$  and  $\mathbf{V}$  is an expression of the form  $T_1[X_1] \subseteq T_2[X_2]$  where, for

$i=1,2$ ,  $T_i$  is the name of a relation scheme in  $R$  or of a view definition in  $R$  and  $X_i$  is a sequence of distinct attributes of  $T_i$  such that  $X_1$  and  $X_2$  are domain compatible.

A *relational schema* is a triple  $S=(R,V,I)$  where  $R$  is a set of relation schemes with distinct names,  $V$  is a set of view definitions over  $R$  and  $I$  is a set of INDPs over  $RUV$ .  $I$  is a set of INDs over  $RUV$ .

A *state*  $\sigma$  for  $S$  assigns a relation  $\sigma(R)$  to each relation scheme  $R$  in  $R$ . We also extend the state  $\sigma$  to assign values to the relational expressions over  $R$  and to the schemes of the views definitions via their defining expressions. We also say that  $\sigma$  is *consistent* iff  $\sigma$  satisfies all keys and INDs of  $S$ .

We conclude by defining the relational schema  $C$  that will be used in later examples. In this and the following examples, we omit the translation of the view operations for the sake of brevity and use  $\lambda$  to denote both the null value or a tuple of null values of arbitrary length.

---

```

/*
  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

/*
  Views
*/
define view S
  attributes Code char(4) not null, Stocks integer
  key Code
  as select Code, Stocks from S*

define view M
  attributes Id      char(4) not null, Code char(4) not null,
             Title  char(40) not null
  key Code key Id
  as select Code, Id, Title from S* where Id ≠ λ

```

```

define view E
  attributes Id char(4) not null, Salary decimal(8,2)
  key Id
  as select Id, Salary from E*

define view R
  attributes Id char(4) not null, Degree char(4) not null
  key Id
  as select Id, Degree from E* where Degree ≠ 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 ≠ 1

define view W
  attributes Id char(4) not null, P_Name char(20) not null
  key Id
  as select Id, P_Name from E* where P_Name ≠ 2
/*
  INDs
*/
M[Id] ⊆ A[Id]
W[P_Name] ⊆ P[P_Name]

```

---

Figure 3: The Relational Schema C

#### 4. THE DESIGN ALGORITHM

A relational schema  $R$  is a *one-to-one relational representation* of an EER schema  $E$  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.

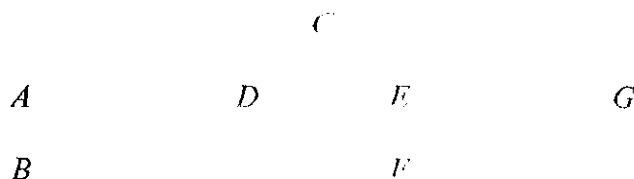
To reduce the number of INDs of a relational representation, a fairly common heuristic is to collapse a relationship scheme  $F$  into an entity scheme  $E$  and to represent both as a single relation scheme  $T$ , if  $F$  is functional on a role that  $E$  plays. The most frequent case is when  $F$  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. The design algorithm we describe in this section is essentially a systematization of this heuristic. 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.

To describe the design algorithm, we first introduce the concept of collapsible arc. Let  $E$  be an EER schema and let  $g(E) = (V, A, I)$  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  $I((F,E))=N$ . A node  $F$  is *collapsible* iff there is a collapsible arc in  $g(E)$  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  $E$  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 collapsible. The forest is *complete* iff none of its roots is a collapsible node. Thus, a complete collapsible 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



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.

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. 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.

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  $P$ .

## 5. THE REDESIGN ALGORITHM

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.

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  $F$
- $s_3$ . **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.

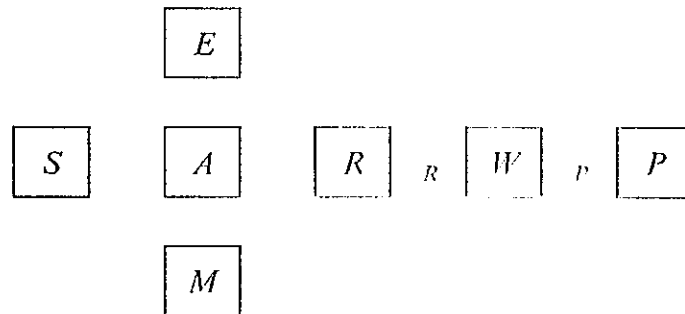


Figure 4: The graph of the new EER schema

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  $t[Id]=u[Id]$ , then set  $t[Code]=u[Code]$  and  $t[Stocks]=u[Stocks]$ , else set  $t[Code]=\lambda$  and  $t[Stocks]=\lambda$ .

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. The processing of nodes  $M$  and  $W$  follows likewise.

We conclude by observing that the operations that form the redesign plan 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, whereas 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] 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).

- [4] Briand, H., Habrias, H., Hue, J-F. and Simon, V., "Expert system for translating an E-R diagram into databases". Proc. 4th Int'l. Conf. on the Entity-Relationship Approach, Chicago (Oct. 1985).
- [5] 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).
- [6] 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).
- [7] 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).
- [8] Chen, P.P., "The entity-relationship model: toward a unified view of data", ACM Transactions on Database Systems 1:1 (1976).
- [9] Chung, I., Nakamura, F. and Chen, P.P., "A decomposition of relations using the entity-relationship approach", in *Entity-relationship approach to information modelling and analysis*. P.P. Chen (Ed.), North-Holland (1981).
- [10] Elmasri, R. and Navathe, S., *Fundamentals of Database Systems*, Benjamin Cummings (1989).
- [11] 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).
- [12] 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).
- [13] Reiner, D., Brodie, M., Brown G. et alli, "The database design and evaluation workbench (DDEW)", IEEE Database Engineering 7:4 (Dec. 1984).
- [14] Rolland, C. and Proix, C., "An expert system approach to information system design", in *Information Processing 86*. H.J. Kugler (ed.), North-Holland (1986).
- [15] 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).
- [16] Tucherman, L., Casanova, M.A., Furtado, A.L., "The CHRIS consultant - A tool for database design and rapid prototyping". Information Systems (to appear).
- [17] 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).