

An Approach to Maintaining Optimized Relational Representations of Entity-Relationship Schemas*

Altigran S. da Silva¹ Alberto H. F. Laender² Marco A. Casanova³

¹ Departamento de Ciência da Computação
Universidade do Amazonas
69077-000 Manaus AM, Brasil
alti@dcc.fua.br

² Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
31270-901 Belo Horizonte MG, Brasil
laender@dcc.ufmg.br

³ LAHESC
IBM Brasil
22960-900 Rio de Janeiro RJ, Brasil
casanova@vnet.ibm.com

Abstract. This paper presents a redesign method for maintaining optimized relational representations of entity-relationship schemas. This method is based on the generation of a transient virtual database state that is used to construct the new database state and that can be obtained without modifying the current relational representation. It also provides means to collect additional data and to integrate them to the the current database state, as well as to check for new integrity constraints.

1 Introduction

The traditional process of designing a relational database usually starts with the definition of an entity-relationship schema (ER schema) from requirements of the real world and proceeds by mapping this schema into a relational representation which will be implemented by a relational DBMS. This process has been extensively discussed in the literature over the past years and a number of design methodologies has been proposed to support it (see, for example, [1, 7, 11, 13]).

Now let us consider the following situation. Suppose that, due to changes in the real world and after the database has been created, the ER schema is modified. This implies that the corresponding relational representation must also be modified to accommodate such changes and that the current database state must be adapted to this new situation. This problem, called the *redesign problem*, is illustrated in Fig. 1 and is described in what follows.

* This work was partially funded by the Brazilian Ministry of Education agency CAPES and by the Brazilian National Research Council (CNPq) under grant 300959/85-0.

Let S_E be an ER schema. In the logical design phase, this schema is mapped (1) into a relational representation S_R . The database is then populated (2) to create a consistent database state σ . Suppose now that, according to changes in the real world, the ER schema S_E is modified generating (3) a new ER schema S'_E . The redesign process consists in modifying S_R to generate (4) a new relational representation S'_R and in mapping (5) σ to a new database state σ' that must be consistent with S'_R (7).

In [3, 4], a *redesign method* was proposed to address this problem. The method accepts as input (1) a list of redesign commands, specifying changes to the original ER schema, (2) the relational representation of the original ER schema, and (3) the current database state, and produces as output a *redesign plan*, which consists of commands written in a relational DDL/DML, to restructure the original database accordingly.

In this paper, we review this redesign method and present a new approach to restructuring the database [12]. This new approach, which addresses some open problems left in [3, 4], is based on the generation of a transient virtual database state that is used to construct the new database state and that can be obtained without modifying the current relational representation. It also provides means to collect additional data and to integrate them to the the current database state, as well as to check if all the new integrity constraints are satisfied.

In the context of this paper, database redesign can be seen as the problem of how to reflect in the relational representation evolutions occurred in the conceptual schema. Conceptual schema evolution has been the subject of many works in the literature (see, for example, [5, 8]), but these works have focused on how to specify the evolution requirements rather than on how to reflect the changes on the database. Proposals for maintaining the logical representation of a database and to adapt the corresponding database state accordingly have also been presented in [9] and [10], but their approaches differ substantially from ours since the former is based on a diagrammatic notation used to specify schema derivations and to support the construction of views for the schema versions whereas the later uses a calculus-oriented ER notation to express database state mappings.

The rest of the paper is organized as follows. Section 2 presents some basic concepts and the terminology used throughout the paper. Section 3 presents an overview of the redesign method. Section 4 discusses the generation of the new ER schema. Section 5 addresses the generation of the new relational representation. Section 6 describes the aspects related to the generation of the new database state. Section 7 discusses the generation of the redesign plan. Finally, Section 8 discusses our main results and presents some conclusions.

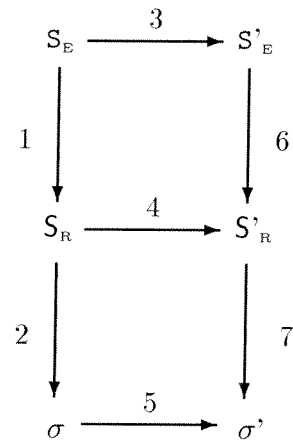


Fig. 1. Redesign process

2 Basic Definitions

2.1 Entity-Relationship Schemas

In this work, we use the concepts of ER schema, entity schemes, relationship schemes and specialization. We also use the standard integrity constraints usually associated with the notion of relationship and specialization [6, 7, 12].

An *entity scheme* has a name, a set of attributes with their respective domains and, optionally, a primary key. An instance of an entity scheme \mathbf{E} is called an \mathbf{E} -instance. A key is a subset of the entity scheme attributes whose values uniquely identify the \mathbf{E} -instances in a set. At least one of the attributes of the entity scheme must be a *discriminating attribute* [12], an attribute whose value is never null. We use this feature to indicate the existence of an entity (instance). The primary key attributes are always discriminating attributes.

An entity scheme may be *specialized* into one or more entity schemes and may also *specialize* one or more entity schemes. We assume that if an entity scheme \mathbf{E} specializes an entity scheme \mathbf{F} , then every \mathbf{E} -instance is also an \mathbf{F} -instance. We also say that \mathbf{F} is a generic of \mathbf{E} .

If a given entity scheme does not specialize any other scheme, the definition of a primary key for this scheme is mandatory. We say that an entity scheme *inherits* the attributes and keys of its generics and we distinguish these attributes and keys, called *inherited* attributes and keys, from those defined in the scheme itself, called *native* attributes and keys. Also, we can define alternate keys to an entity scheme.

A *relationship scheme* has a name and may optionally have an attribute list. The instances of a relationship scheme \mathbf{R} , also called \mathbf{R} -instances, are elements from the Cartesian product of the set of instances of the entity schemes that are participants of \mathbf{R} . To each participant is given a *role* that must be unique in each relationship scheme.

Let $\{\mathbf{E}_1, \dots, \mathbf{E}_n\}$ be a subset of participants of \mathbf{R} with roles $\mathbf{N}_1, \dots, \mathbf{N}_n$, respectively. The set $\{\mathbf{N}_1, \dots, \mathbf{N}_n\}$ is called an identifier of \mathbf{R} if every combination of instances of $\mathbf{E}_1, \dots, \mathbf{E}_n$ is unique in the set of \mathbf{R} -instances. If $n = 1$ we say that \mathbf{R} is *functional* on \mathbf{E}_1 . We also say that \mathbf{R} is *total* on an entity scheme \mathbf{E} with role \mathbf{N} if every \mathbf{E} -instance participates with role \mathbf{N} in at least one \mathbf{R} -instance.

Entity schemes and relationship schemes are also called *ER-object (ERO)* schemes in this paper.

An *ER schema* is a pair $\mathbf{S}_E = \langle \mathcal{E}, \mathcal{R} \rangle$, where \mathcal{E} is a set of entity schemes and \mathcal{R} is a set of relationship schemes. A state σ_E of an ER schema \mathbf{S}_E assigns to each ER-object \mathbf{O} in \mathbf{S}_E a set of \mathbf{O} -instances. The state is *consistent* if all associated semantic constraints are satisfied by the instances of its entity and relationship schemes. A more precise definition may be found in [12]. We assume in this paper that every attribute has a unique name in an ER schema. This assumption is not actually needed, as explained in [12], but it will be used here to simplify the examples.

Fig. 2 shows the definition of the ER schema **Company 1** which will be used as our running example in this paper. Since the schemas that we will present

here are quite simple, we will omit the description of the syntax used to define them.

- define entity **E** (**Employee**)
attributes **Id** char(4) not null,
 Salary decimal(8,2)
key **Id**
- define entity **S** (**Stock-Holder**)
attributes **Code** char(4) not null,
 Stocks integer
key **Code**
- define entity **P** (**Project**)
attributes **Name** char(20) not null,
 Contractor char(20)
key **Name**
- define entity **R** (**Researcher**)
attributes **Degree** char(4) not null
specialization of **E**
- define entity **A** (**Administrative**)
attributes **Job_Desc** char(40) not null
specialization of **E**
- define entity **M** (**Manager**)
attributes **Title** char(40) not null
specialization of **S, A**
- define relationship **W** (**Work**) over **R, P**
attributes **Hours** integer not null,
 identifier **R**

Fig. 2. ER schema *Company 1*

A graph $g = (V, A, l)$ of an ER schema S_E (or simply an *ER graph*) is a directed multi-graph, partially labeled by l , where every vertex in V corresponds to an ERO scheme of S_E and an arc $(S, T) \in A$ iff S specializes T or T participates in S with role \mathbf{N} , being $l((S, T)) = \mathbf{N}$. We say that (S, T) is a *total arc* if S is total on T . Fig. 3(a) shows the ER graph for the ER schema *Company 1*.

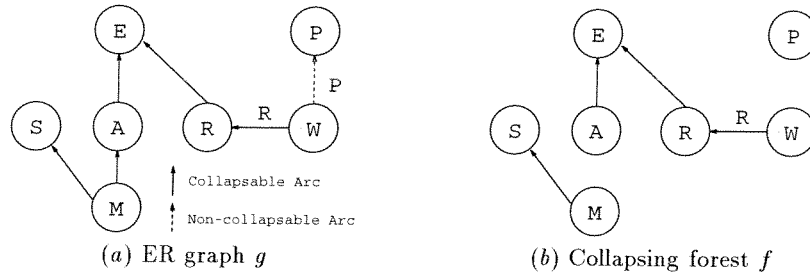


Fig. 3. ER graph and collapsing forest of the ER schema *Company 1*

2.2 Relational Schemas

A *relational schema* is a triple $S_R = \langle T, \mathcal{V}, \mathcal{C} \rangle$, where T is a set of relation schemes, \mathcal{V} is a set of view schemes and \mathcal{C} is a set of integrity constraints. A *state* σ for S_R assigns to each relation scheme R in T a relation r in the usual way. To stress that r is associated with R , we sometimes use the notation $r(R)$ in what follows.

A *relation scheme* has a name, an attribute list (each attribute has a respective domain), a primary key and, optionally, one or more alternate keys. We can also specify that a certain attribute may not assume null values. A null value is

represented here by the symbol λ . A *primary key* is a list of attributes that must not assume null values and an *alternate key* is any list of attributes. Any view scheme in \mathcal{V} is defined over relation schemes in \mathcal{T} or others view schemes in \mathcal{V} .

The integrity constraints in \mathcal{C} may be of two types: inclusion dependencies [2] and null dependencies [12].

Let \mathcal{T} be a set of relation schemes. An *inclusion dependency (IND)* is an expression of the form $T_1[X_1] \subseteq T_2[X_2]$ where, for $j = 1, 2$, T_j is a relation scheme in \mathcal{T} and X_j is a sequence of distinct attribute names of T_j such that X_1 and X_2 have the same number of attributes and that the k -th attribute of X_1 and the k -th attribute of X_2 have the same domain. An inclusion dependency is satisfied by the relations $t_1(T_1)$ and $t_2(T_2)$ of the same state σ iff, for each tuple t of $t_1(T_1)$, there is a tuple u of $t_2(T_2)$ such that $t[X_1] = u[X_2]$. We also allow the definition of INDs over view schemes.

A *null dependency (NUD)* is an expression of the form

$$R : A_1, A_2, \dots, A_n \rightsquigarrow B_1, B_2, \dots, B_m,$$

where $A = \{A_1, A_2, \dots, A_n\}$ and $B = \{B_1, B_2, \dots, B_m\}$ are two subsets of the set of attributes of a relation scheme R in \mathcal{T} . A null dependency is satisfied by a relation $r(R)$ iff, for every tuple u of $r(R)$, if some attribute A_i is null, then all B_j must necessarily be null ($i = 1, \dots, n, j = 1, \dots, m$). We say that there exists a null dependency from B to A . In the DDL of many relational DBMS, it is possible to specify that an attribute will never assume a null value. In the context of null dependencies, this can be expressed by the notation $A_k \rightsquigarrow \square$, for an specific attribute A_k .

A *mutual null dependency* is an expression of the form $R:[C_1, C_2, \dots, C_n]$, where $C = \{C_1, C_2, \dots, C_n\}$ is a subset of the set of attributes of a relation scheme R in \mathcal{T} . A mutual null dependency is satisfied by a relation $r(R)$ iff, for every tuple u of $r(R)$, if some attribute C_i is null, then all the attributes in C are also null. We say that there is a mutual null dependency among the attributes of C , that is, the value of all the attributes of C must be either null or not null in a tuple.

An example of an relational schema is presented in Fig. 4. The syntax used to define this schema is also quite simple and will not be described.

2.3 Optimized Relational Representations of ER Schemas

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

A straightforward method to obtain a relational representation of an ER schema is to generate a relation scheme (with appropriate attributes) to represent each ERO scheme and to use INDs to capture the semantics of the arcs of the corresponding ER graph. Despite its simplicity, this method potentially produces a large number of INDs that are expensive to check for violation [3, 4].

- define relation E^*
attributes Id char(4) not null,
Salary decimal(8,2),
Job_Desc char(40),
Degree char(4),
Hours integer,
Name char(20)
key Id
- define view E
attributes Id char(4) not null,
Salary decimal(8,2)
key Id
as $E^*[Id,Salary]$
- define view A
attributes Id char(4) not null,
Job_Desc char(40) not null
key Id
as $E^*[Job_Desc \neq \lambda][Job_Desc,Id]$
- define view R
attributes Id char(4) not null,
Degree char(4) not null
key Id
as $E^*[Degree \neq \lambda][Degree,Id]$
- define relation S^*
attributes $Code$ char(4) not null,
Stocks integer,
Title char(40),
 Id char(4)
key $Code$
key Id
- define view S
attributes $Code$ char(4) not null,
Stocks integer
key $Code$
as $S^*[Code,Stocks]$
- define view M
attributes $Code$ char(4) not null,
 Id char(4) not null,
Title char(40) not null
key $Code$
key Id
as $S^*[Title \neq \lambda][Title,Code,Id]$
- define relation P^*
attributes $Name$ integer not null,
Contractor char(40)
key $Name$
- define view P
attributes $Name$ char(20) not null,
Contractor char(40)
key $Name$
as $P^*[Name,Contractor]$
- define view W
attributes Id char(4) not null,
 $Name$ char(20) not null,
Hours integer not null
key Id
as $E^*[Hours \neq \lambda][Hours,Id,Name]$
- INDs: $M[Id] \subseteq A[Id]$, $W[Name] \subseteq P[Name]$
- NUDs: $S^*:[Title,Id]$, $E^*:[Hours,Name]$, $E^*:Degree \sim Hours,Name$

Fig. 4. Relational representation of the ER schema *Company 1*

To reduce the number of INDs of a relational representation, a fairly common heuristic is to collapse into an entity scheme \mathbf{E} the relationship schemes that are functional on a role that \mathbf{E} plays, as well as the entity schemes that specialize \mathbf{E} , and to represent them as a single relation scheme. In this way, several INDs can be removed and some may be replaced by NUDs, which are cheaper to check. We call a relational representation obtained using this strategy an *optimized relational representation* [3, 4, 12].

An optimized relational representation is constructed based on a structure called a *collapsing forest* [3, 4]. A collapsing forest f of an ER schema \mathbf{S}_E is a set of trees containing the same vertices as the ER graph of \mathbf{S}_E , but in which only the *collapsible arcs* of g are present. An arc (S, T) in g is collapsible when it has no label or it is labeled with a label \mathbf{N} and S is functional on its participant T

with role \mathbf{N} .

In an optimized relational representation, for each root \mathbf{R} in the collapsing forest f , we generate a relation scheme \mathbf{R}^* , called a *collapsing scheme*, such that \mathbf{R}^* represents all the ERO schemes corresponding to the vertices in the tree whose root is \mathbf{R} . The generation of this scheme may be seen as the result of a function $\mu(f, \mathbf{R})$. Additionally, we generate a view scheme for each vertex \mathbf{V} of the collapsing forest f . Each of these view schemes is used to individually represent an ERO scheme in the forest and is defined over the collapsing scheme into which the ERO scheme has been collapsed. We call these view schemes *representation schemes*. The generation of a representation scheme of an ERO scheme \mathbf{V} may be seen as the result of a function $\nu(f, \mathbf{V})$. With respect to the integrity constraints, for each arc (\mathbf{S}, \mathbf{T}) of g that is not in f , we generate an IND from the representation of \mathbf{T} to the representation of \mathbf{S} . If (\mathbf{S}, \mathbf{T}) is a total arc, we also generate an IND in the inverse direction. For each arc (\mathbf{S}, \mathbf{T}) that is both in f and g , we generate a NUD from the attributes of \mathbf{T} to the attributes of \mathbf{S} . If (\mathbf{S}, \mathbf{T}) is a total arc, we also generate a NUD in the inverse direction. An exception to the former rule occurs when \mathbf{T} is a root in the collapsing forest. A NUD is additionally generated to define a mutual null dependency involving the discriminating attributes of every ERO scheme that does not correspond to a root in the collapsing forest.

The design method outlined above is discussed in [3, 4]. In [12] we prove that it generates correct relational representations of ER schemas.

The relational schema **Company 1** (Fig. 4) is an optimized relational representation of the ER schema **Company 1** (Fig. 2).

3 The Redesign Method

Let $\mathbf{S}'_{\mathbf{E}}$ be an ER schema that is the result of modifying an ER schema $\mathbf{S}_{\mathbf{E}}$. If a relational schema $\mathbf{S}_{\mathbf{R}}$ is a relational representation of $\mathbf{S}_{\mathbf{E}}$, we must modify $\mathbf{S}_{\mathbf{R}}$ in such a way that it becomes a new relational schema $\mathbf{S}'_{\mathbf{R}}$ which is a relational representation of $\mathbf{S}'_{\mathbf{E}}$.

In general, the collapsing forest f of $\mathbf{S}_{\mathbf{E}}$ is not an adequate collapsing forest for $\mathbf{S}'_{\mathbf{E}}$ and we have to generate a new collapsing forest f' . By comparing the differences between f and f' (the new roots, internal vertices, arcs, etc) and the differences between the schemes associated with the vertices in both forests, we can produce $\mathbf{S}'_{\mathbf{R}}$.

We must also consider that $\mathbf{S}_{\mathbf{R}}$ is possibly populated with data, that is, its corresponding relations form a consistent database state σ . Therefore we must map σ into a new database state σ' consistent with $\mathbf{S}'_{\mathbf{R}}$. In some cases, this goal can only be achieved by introducing additional data to satisfy new constraints added to the new relational representation, although sometimes even the addition of new data does not make the mapping possible.

The redesign method we propose accepts as input a sequence of *redesign commands* that specify changes to the original ER schema $\mathbf{S}_{\mathbf{E}}$ to produce a new ER schema $\mathbf{S}'_{\mathbf{E}}$. The aim of the method is to generate a sequence of commands

in a relational DDL/DML that, when executed, will transform the original relational representation S_R into a new relational representation S'_R and will map the original database state σ into a new database state σ' , which must be consistent with S'_R . The method also checks σ to determine whether or not this mapping is possible. We call this sequence of commands the *redesign plan*.

In our redesign method, we have three main tasks to perform: (1) **Generation of the new ER schema** – In this task, the redesign commands are applied to the original ER schema producing a new ER schema and generating the corresponding graph and collapsing forest; (2) **Generation of the new relational representation** – This task addresses the modification of the original relational representation based on the comparison of the original graph and collapsing forest with the new ones; (3) **Generation of the new database state** – This task maps the original database state into a new one, possibly collecting some additional data to satisfy new integrity constraints and checking whether or not the mapping is possible.

Our method executes task (1) directly and generates a redesign plan whose commands must be executed in order to perform tasks (2) e (3). In the next sections, we describe, with the help of an example based on the ER schema of Fig. 2, how these three tasks are performed according to our redesign method. A more detailed discussion can be found in [12] where the algorithms for generating the commands that compose a redesign plan are described.

4 Generation of the New ER Schema

Given an ER schema, its graph, and its collapsing forest, the method applies a number of redesign commands, issued by the database designer, to transform the original ER schema into a new one. The redesign commands must be such that, after applying all of them, the resulting ER schema is a correct one. For example, we cannot issue a command to remove an entity scheme E if we do not also issue another command to remove the participation of E in all relationship schemes where E is a participant. We also assume that the list of participants in a relationship scheme cannot be changed.

Consider the ER schema **Company 1** in Fig. 2 and the list of redesign commands presented in Fig. 5. If we apply this list of commands to the ER schema **Company 1**, we generate a new ER schema **Company 2**, whose ER graph and collapsing forest are shown in Fig. 6. Fig. 7 shows the new definition of the ERO schemes affected by these commands.

Although in this paper we use a simple set of redesign commands, which are defined in [12], we realize that a more elaborated and powerful set of transformation commands is needed to offer more facilities to the database designer. We refer the reader to [5, 8] where this topic is more deeply discussed.

- add "S specialization of E"
- remove "M specialization of S"
- remove "identifier R from relationship W"
- add "total P on relationship W"
- remove "key {Name} from P"
- add "attribute Num integer not null to P"
- add "key {Num} to P"

Fig. 5. Redesign commands to transform the ER schema Company 1

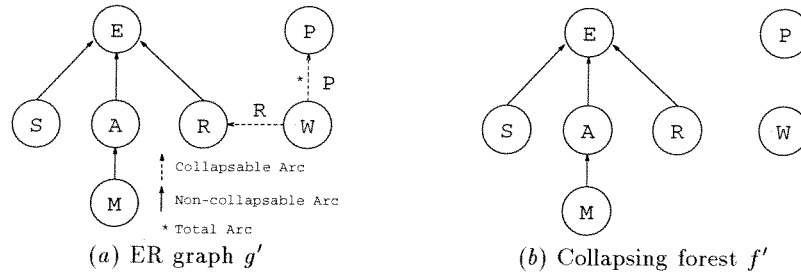


Fig. 6. ER graph and collapsing forest of the ER schema Company 2

5 Generation of the New Relational Representation

The transformation of the original relational representation into a new one is achieved by comparing the original and the new collapsing forests and the schemes of the corresponding vertices of both forests. Based on this comparison, the method issues commands that can be grouped into three types [12]: (1) **Expansion commands** – create collapsing schemes, representation schemes and new integrity constraints, and add new attributes and keys to relation schemes; (2) **Contraction commands** – remove collapsing schemes, representation schemes, integrity constraints, attributes and keys; (3) **Refreshing commands** – alter the representation schemes of ERO schemes when they are modified or their position in the collapsing forest is changed.

Fig. 8 presents expansion, contraction and refreshing commands (written in a informal syntax) that, if applied in this order to the relational schema of

- define entity **S** (**Stock-Holder**)
attributes **Code** char(4) not null,
 Stocks integer
key **Code**
specialization of **E**
- define entity **M** (**Manager**)
attributes **Title** char(40) not null
specialization of **A**
- define entity **P** (**Project**)
attributes **Num** integer not null,
 Name char(20) not null,
 Contractor char(20)
key **Num**
- define relationship **W** (**Work**) over **R**, total **P**
attributes **Hours** integer not null

Fig. 7. New definition of the ERO schemes affected by the redesign commands

Fig. 4, will generate the relational schema *Company 2*, shown in Fig. 9, which is the relational representation of the ER Schema *Company 2*. Note from Fig. 6(b) that, in our example, a new root *W* was created and that *E* now has *S* as a new child. Also note that the representation schemes of *P* and *W* now have new attributes. Thus, the commands shown in Fig. 8 reflect exactly these changes.

EX1. create W^* according to $\mu(f', \mathbb{W})$
EX2. add Code, Stocks and Title to E^*
EX3. add an alternate key {Code} to E^*
EX4. create NUD $E^*:\text{Job_Desc} \sim \text{Title}$
EX5. add Num to the primary key of P^*
EX6. create IND $W[\text{Num}] \subseteq P[\text{Num}]$
EX7. create IND $P[\text{Num}] \subseteq W[\text{Num}]$
EX8. create IND $W[\text{Id}] \subseteq R[\text{Id}]$

(a) Expansion commands

CT1. remove collapsing scheme S^*
CT2. remove IND $W[\text{Name}] \subseteq P[\text{Name}]$
CT3. remove Hours, Name from E^*
CT4. remove Name from the primary key of P^*

(b) Contraction commands

RF1. rebuild *S* according to $\nu(f', S)$
RF2. rebuild *M* according to $\nu(f', \mathbb{M})$
RF3. rebuild *W* according to $\nu(f', \mathbb{W})$
RF4. rebuild *P* according to $\nu(f', P)$

(c) Refreshing commands

Fig. 8. Commands to transform the original relational representation

6 Generation of the New Database State

In this section, we describe how the method generates a database state σ' , consistent with the new relational representation S'_R , from the original database state σ . This is performed by mapping σ into σ' , that is, by adapting σ to S'_R . However, the new state will be consistent only if all integrity constraints in S'_R are satisfied by the parts of σ that remain in σ' . Thus, for any integrity constraint, we have three situations: (1) the integrity constraint is satisfied; (2) the integrity constraint is not satisfied; and (3) the integrity constraint cannot be checked because the values of the attributes to which it refers are not present in σ .

The third situation occurs, for example, when a new discriminating attribute is created in the new ER schema. To deal with this situation, the method provides means to: (1) collect additional data and integrate them to the database; and (2) verify if all integrity constraints in S'_R are satisfied in the *transient state*, i.e., the database state that corresponds to the data in σ integrated with new collected data. The additional data is collected through relations created by the method and that must be populated accordingly. This type of relation is called a Δ -relation. To integrate the new collected data to the database, the method

- define relation E^*
attributes Id char(4) not null,
Salary decimal(8,2),
Job_Desc char(40),
Degree char(4),
Code char(4),
Stocks integer,
Title char(40),
key Id
key $Code$
- define view E
attributes Id char(4) not null,
Salary decimal(8,2)
key Id
as $E^*[Id,Salary]$
- define view A
attributes Id char(4) not null,
Job_Desc char(40) not null
key Id
as $E^*[Job_Desc \neq \lambda][Job_Desc,Id]$
- define view R
attributes Id char(4) not null,
Degree char(4) not null
key Id
as $E^*[Degree \neq \lambda][Degree,Id]$
- define view M
attributes Id char(4) not null,
Title char(40) not null
key Id
as $E^*[Title \neq \lambda][Title,Id]$
- NUD: $E^*:Job_Desc \sim Title$
- INDs: $W[Num] \subseteq P[Num]$, $P[Num] \subseteq W[Num]$, $W[Id] \subseteq R[Id]$
- define view S
attributes $Code$ char(4) not null,
Stocks integer
 Id char(4) not null
key $Code$
key Id
as $E^*[Code \neq \lambda][Code,Stocks]$
- define relation P^*
attributes Num integer not null,
Name char(4) not null,
Contractor char(40)
key Num
- define view P
attributes Num integer not null,
Name char(20) not null,
Contractor char(40)
key Num
as $P^*[Name,Num,Contractor]$
- define relation W^*
attributes Id char(4), not null,
Num integer, not null,
Hours integer, not null
key Id,Num
- define view W
attributes Id char(4) not null,
Num integer not null,
Hours integer not null
key Id,Num
as $W^*[Id,Num,Hours]$

Fig. 9. Relational representation of the ER schema Company 2

creates a set of view schemes, called *transient representation schemes*, which will be used to generate the new database state σ' .

Note that some integrity constraints may not be satisfied by the new state. For example, let A be an entity scheme that participates in a relationship scheme B , and suppose that in the state σ we have an A -instance participating in more than one B -instance. If, in the new ER schema S'_E , A is functional on B , we have no means of adapting this state to S'_E without loss of information, since only one of these B -instances would be allowed to exist in the new database state. In this case, we say that σ is not adequate to S'_E ; otherwise, we say that σ is adequate to S'_E .

We divide the operations to generate the new database state into two groups: (1) operations to determine whether the original database state is adequate to

the new ER schema; and (2) operations to map the original database state (possibly integrated with additional data) into a new state. The method adds these operations to the redesign plan by examining the current database state and the new constraints added to correctly represent the new ER schema.

Although some integrity constraints can be verified before data collection, in our method all the new constraints are checked after new data is collected and integrated to compose the transient state.

6.1 Adjusting the Original Database State to the New ER Schema

We will now illustrate how the redesign method proceeds to adjust the original database state, i.e., to adjust the current state of the original relational representation to the the new ER schema. Our aim is to show how additional data are collected and integrated to the current database state and how the transient state is used to check if all the new integrity constraints are satisfied.

Collecting and Integrating Additional Data. In our redesign method, data collection is performed basically to satisfy new integrity constraints. Thus, we have only to determine values for the discriminating attributes because all integrity constraints considered, i.e., key constraints, INDs and NUDs, are defined on such attributes.

Consider the transformation of the relational representation of the ER schema **Company 1** into the relational representation of ER the schema **Company 2**. We notice that new discriminating attributes have been introduced to the representations of **S**, **P** and **W**, as we can see by comparing Fig. 4 and Fig. 9.

The attribute **Num** is a new native discriminating attribute of **P**, so it must have a not null value for all **P**-instances. To deal with this situation, the method creates the relation **P_Num(Name, Num)**, which is a Δ -relation that must be populated to associate a value of **Num** with each **P**-instance identified by the value of the attribute **Name**, the original key of **P**. Fig. 10(b) shows a possible instance for this relation. To associate the defined values for this new native discriminating attribute with the values of pre-existing attributes, the method generates the view $\overline{\mathbf{P}}$, shown in Fig. 10(c), which is obtained by joining **P_Num** and the current instance of the original representation scheme **P** of **P**, shown in Fig. 10(a).

P	Name	Contractor
1	<i>X</i>	<i>Comp.A</i>
2	<i>Y</i>	λ
3	<i>Z</i>	<i>Comp.C</i>

(a)

P_Num	Name	Num
1	<i>X</i>	<i>1</i>
2	<i>Y</i>	<i>2</i>
3	<i>Z</i>	<i>3</i>

(b)

$\overline{\mathbf{P}}$	Num	Contractor	Name
1	<i>1</i>	<i>Comp.A</i>	<i>X</i>
2	<i>2</i>	λ	<i>Y</i>
3	<i>3</i>	<i>Comp.C</i>	<i>Z</i>

(c)

Fig. 10. Collecting data for **P**

To determine the values of **Num** for **W**-instances, we must verify the values of **Num** in the corresponding **P**-instances. This is necessary because we have a previous association of **P**-instances and **W**-instances in the original ER schema. These values are determined by the relational algebra expression $(W[\text{Id}, \text{Name}] \bowtie \overline{\text{P}})[\text{Id}, \text{Num}]$. If we consider the relation **W** shown in Fig. 11(a) as the current state of the representation scheme of **W**, the result of this expression is the relation **WxP**, shown in Fig. 11(b).

W	Id	Name	Hours
1	E4	X	10
2	E6	Y	20
3	E5	Z	15

(a)

WxP	Id	Num
1	E4	1
2	E6	2
3	E5	3

(b)

S.Id	Code	Id
1	S2	E1
2	S5	E6

(c)

SxE	Id	Code
1	E3	S1
2	E1	S2
3	E4	S3
4	E5	S4
5	E6	S5

(d)

Fig. 11. Collecting data for **W** and **S**

In the representation of **S**, the new attribute **Id** is introduced to associate **S**-instances with **E**-instances, since **S** is now a specialization of **E**. Looking at the ER graph g in Fig. 3(a), we can see that some **S**-instances may be already associated with **E**-instances, because **S** and **E** have the entity scheme **M** as a common descendent. Based on this, the method generates the relational algebra expression $S^*[\text{Title} \neq \lambda][\text{Code}, \text{Id}]$ to determine the values of **Id** that are already associated with some **S**-instance. In the relation that results from this expression, each tuple represents, by the value of the key **Code** of **S**, an **S**-instance. Note that, for **S**-instances that are not represented in this relation, the corresponding values of the attribute **Id** must be collected. For this purpose, the method creates a Δ -relation **S_Id**(**Code**, **Id**). A possible instance of this relation is shown in Fig. 11(c). Note that there will be an IND to be checked afterwards and that guarantees that this relation instance is correct. The method also generates the relational algebra expression $S^*[\text{Title} \neq \lambda][\text{Code}, \text{Id}] \cup \text{S_Id}$, whose resulting relation **SxE** is shown in Fig. 11(d). This relation integrates the collected values of **Id** to those in the current database state σ . We assume that tuples 1, 3 and 4 show the pre-existing associations between **S**-instances and **E**-instances.

The sequence of commands generated by the method for performing this task is shown in Fig. 12. Note that commands *VL2* and *VL3* create INDs that guarantee that a value must be supplied for the attribute **Num** for every **P**-instance.

This simple example illustrates the basic aspects of the data collection strategy of our redesign method. A more detailed discussion of this topic can be found in [12].

Checking the New Integrity Constraints. Before we discuss how the new integrity constraints are checked, we introduce the concept of a *transient representation scheme*. A transient representation scheme of an ERO scheme **O** is a view scheme $\tilde{\text{O}}$ defined over its original representation scheme **O** and whose tuples correspond

VL1. create relation P_Num(Name,Num), key Name
VL2. add IND P_Num[Name] \subseteq P[Name]
VL3. add IND P[Name] \subseteq P_Num[Name]
VL4. include tuples in P_Num
VL5. create relation S_Id(Code,Id), key Code;
VL6. include tuples in S_Id

Fig. 12. Commands to collect data

to the \mathbf{O} -instances that will compose the new database state according to its new representation scheme [12]. If new discriminating attributes have been added to \mathbf{O} , the defining expression of $\tilde{\mathbf{O}}$ will involve the Δ -relations and additional relations (such as $W \times P$ and $S \times E$ shown in Fig. 11) generated to associate existing instances in the current database state. Otherwise, the defining expression of $\tilde{\mathbf{O}}$ will be a trivial relational expression on the original representation scheme \mathbf{O} of \mathbf{O} . Note that the tuples of $\tilde{\mathbf{O}}$ include at least the values of the discriminating attributes in the new relational representation and that such values are associated with the key of \mathbf{O} in the original representation.

Fig. 13(a) shows the definition of the transient representation schemes for \mathbf{S} , \mathbf{P} and \mathbf{W} ($\tilde{\mathbf{S}}$, $\tilde{\mathbf{P}}$ and $\tilde{\mathbf{W}}$, respectively). The corresponding views generated according to the relations and views previously presented are shown in Fig. 13(b).

<pre> define view $\tilde{\mathbf{S}}$ attributes Code char(4), Stocks integer, Id char(4) as S[Code, Stocks] \bowtie SxE define view $\tilde{\mathbf{P}}$ attributes Name char(20), Contractor char(40), Num integer as P[Name, Contractor] \bowtie $\overline{\mathbf{P}}$ define view $\tilde{\mathbf{W}}$ attributes Id char(4), Hours integer, Num integer as W[Id, Hours] \bowtie WxP </pre>	<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th>$\tilde{\mathbf{S}}$</th> <th>Code</th> <th>Stocks</th> <th>Id</th> </tr> </thead> <tbody> <tr><td>1</td><td>S1</td><td>10000</td><td>E3</td></tr> <tr><td>2</td><td>S2</td><td>5000</td><td>E1</td></tr> <tr><td>3</td><td>S3</td><td>7000</td><td>E4</td></tr> <tr><td>4</td><td>S4</td><td>8000</td><td>E5</td></tr> <tr><td>5</td><td>S5</td><td>-</td><td>E6</td></tr> </tbody> </table> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th>$\tilde{\mathbf{P}}$</th> <th>Name</th> <th>Contractor</th> <th>Num</th> </tr> </thead> <tbody> <tr><td>1</td><td>X</td><td>Comp.A</td><td>1</td></tr> <tr><td>2</td><td>Y</td><td>Comp.B</td><td>2</td></tr> <tr><td>3</td><td>Z</td><td>Comp.C</td><td>3</td></tr> </tbody> </table> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th>$\tilde{\mathbf{W}}$</th> <th>Id</th> <th>Num</th> <th>Hours</th> </tr> </thead> <tbody> <tr><td>1</td><td>E4</td><td>1</td><td>10</td></tr> <tr><td>2</td><td>E6</td><td>2</td><td>20</td></tr> <tr><td>3</td><td>E5</td><td>3</td><td>15</td></tr> </tbody> </table>	$\tilde{\mathbf{S}}$	Code	Stocks	Id	1	S1	10000	E3	2	S2	5000	E1	3	S3	7000	E4	4	S4	8000	E5	5	S5	-	E6	$\tilde{\mathbf{P}}$	Name	Contractor	Num	1	X	Comp.A	1	2	Y	Comp.B	2	3	Z	Comp.C	3	$\tilde{\mathbf{W}}$	Id	Num	Hours	1	E4	1	10	2	E6	2	20	3	E5	3	15
$\tilde{\mathbf{S}}$	Code	Stocks	Id																																																						
1	S1	10000	E3																																																						
2	S2	5000	E1																																																						
3	S3	7000	E4																																																						
4	S4	8000	E5																																																						
5	S5	-	E6																																																						
$\tilde{\mathbf{P}}$	Name	Contractor	Num																																																						
1	X	Comp.A	1																																																						
2	Y	Comp.B	2																																																						
3	Z	Comp.C	3																																																						
$\tilde{\mathbf{W}}$	Id	Num	Hours																																																						
1	E4	1	10																																																						
2	E6	2	20																																																						
3	E5	3	15																																																						
(a)	(b)																																																								

Fig. 13. Transient representation schemes $\tilde{\mathbf{S}}$, $\tilde{\mathbf{P}}$ and $\tilde{\mathbf{W}}$

The transient representation schemes aim at: (1) integrating the additional data with the pre-existing data; (2) facilitating the checking of new integrity constraints; and (3) facilitating the generation of the new database state. Aim (1) is obviously fulfilled. Aims (2) and (3) will be discussed in what follows.

We call attention to the fact that a transient state can be viewed as a set of instances of transient representation schemes and that this state is in fact the initial state of the new relational representation. Note that this state can be obtained without modifying the original relational representation, that is, we can “virtually” generate the new relational representation and the new database state.

In order to discuss how the new integrity constraints are checked, let us consider that the new integrity constraints added to the new relational representation in our example are the following: (1) Keys $\{\text{Id}\}$ of S , $\{\text{Num}\}$ of P , and $\{\text{Id}, \text{Num}\}$ of W ; and (2) INDs $S[\text{Id}] \subseteq E[\text{Id}]$, $P[\text{Num}] \subseteq W[\text{Num}]$ and $W[\text{Num}] \subseteq P[\text{Num}]$.

As we said, the transient state will be used as the initial state of the new relational representation. These integrity constraints will be satisfied by this initial state iff they are satisfied by the transient state. Thus, we must check the following situations in the transient state: (1) if $\{\text{Id}\}$, $\{\text{Num}\}$, and $\{\text{Id}, \text{Num}\}$ have unique values for the tuples of \tilde{S} , \tilde{P} and \tilde{W} , respectively; and (2) if the INDs $\tilde{S}[\text{Id}] \subseteq \tilde{E}[\text{Id}]$, $\tilde{P}[\text{Num}] \subseteq \tilde{W}[\text{Num}]$, and $\tilde{W}[\text{Num}] \subseteq \tilde{P}[\text{Num}]$ are satisfied.

If all new integrity constraints are satisfied in the transient state, we say that the current state of the original relational representation is *adequate* to the new ER schema. This means that we can generate a transient state, consistent with the relational representation of the new ER schema, from the current state of the original relational representation. Note that these two states co-exist in the database. Thus, we only have to check if the new integrity constraints are satisfied by the transient state, since all pre-existing integrity constraints are already satisfied and the current database state is considered consistent. The redesign method issues the commands to check these new integrity constraints.

6.2 Mapping the Transient Database State into the New State

Once the transient state is ready and checked, we must “materialize” it in order to generate the new database state. This is carried out by assigning values to the new relation attributes, i.e, by correctly filling the new attributes created by the expansion operations.

When an ERO scheme O is collapsed into a new collapsing scheme R^* in the new relational representation, the O -instances (if there is any) must be “moved” to the tuples of the relation R^* . This situation can be detected by examining the new and the original collapsing forests. Note that we guarantee that the state of a new relational representation is valid by taking the O -instances from the view defined by \tilde{O} , the transient representation scheme of O . If, instead, O remains collapsed in the same collapsing scheme, we must check if new attributes have been included in its representation and also get the values of these attributes from \tilde{O} .

In our example, the entity scheme S is a new child of E in the collapsing forest f' . This caused the expansion of E^* with the attributes that represent S -instances. Also, we created a new root W , as well as a new collapsing scheme

W^* , and expanded P^* with the new attribute of P . Fig. 14 shows the command sequence that must be executed to assign correct values to the new attributes.

```
ME1. update  $E^*$  set Code= $\lambda$ , Stocks= $\lambda$ 
ME2. update  $E^*$  set Code= $S[Code]$ , Stocks= $\tilde{S}[Stocks]$  where  $E^*[Id] = \tilde{S}[Id]$ 
ME3. update  $P^*$  set Num= $\lambda$ 
ME4. update  $P^*$  set Num= $P[Num]$  where  $P^*[Name] = \tilde{P}[Name]$ 
ME5. insert  $W^*(Id, Num, Hours)$  from  $W[Id, Num, Hours]$ 
```

Fig. 14. Commands to assign values to new attributes

7 Generation of the Redesign Plan

To correctly generate the new relational representation and the new database state, we must execute the commands required to restructure the database in such an order that each operation is successfully executed. For example, we cannot execute a command to assign values to a new attribute of a relation before expanding its scheme accordingly. Moreover, these values must be firstly collected before this operation can be executed. This requirement is trivially satisfied if we generate the commands in the following order: (1) commands to collect new data needed for the new database state; (2) commands to generate the transient representation views; (3) commands to verify if the transient state is adequate to the new ER schema; (4) commands to expand the current relational representation (expansion commands); (5) commands to map the transient state into the new database state; (6) commands to alter the current representation schemes (refreshing commands); (7) commands to contract the current relational representation (contraction commands).

Thus, any sequence of commands in this order is a possible redesign plan. Note that a plan generated in this way is not necessarily optimal with respect to execution time, number of operations or additional space required, what means that some optimization strategy may be applied to it in order to improve its execution.

8 Conclusions

In this paper, we reviewed the database redesign method proposed in [3, 4] and presented a new approach to maintaining optimized relational representations of ER schemas which addresses some open problems left in those papers. This new approach is based on the generation of a transient virtual database state that is used to construct the new database state and that can be obtained without modifying the current relational representation. This allows the database designer to assess the impact of the conceptual changes without actually restructuring the database.

At the moment, we are implementing this redesign method as part of a database design/redesign tool for a specific commercial relational database management system. This tool generates the redesign plan entirely in terms of commands of the DDL/DML of the target system.

As a future work, we intend to extend the set of redesign commands proposed in [12] in order to offer more facilities to the database designer. We also intend to address the problem of optimizing the generation of the redesign plan. A possible strategy to address this problem may be to adapt the approach proposed in [5], which considers the problem at the conceptual schema level, to the relational representation level.

References

1. Batini, C., Ceri, S. and Navathe, S. *Conceptual Database Design: An Entity-Relationship Approach*, Benjamin Cummings (1992).
2. Casanova, M.A., Tucherman, L., Furtado, A.L. and Pacheco, A. "Optimization of relational schemas containing inclusion dependencies". *Proc. 15th Int. Conf. on Very Large Data Bases*, Amsterdam, Holland (1989).
3. Casanova, M.A., Tucherman, L. and Laender, A.H.F. "Algorithms for designing and maintaining optimized relational representations of entity-relationship schemas", in Kangassalo, H. (ed.), *Entity-Relationship Approach: The Core of Conceptual Modelling*, North-Holland (1991).
4. Casanova, M.A., Tucherman, L. and Laender, A.H.F. "On the design and maintenance of optimized relational representations of entity-relationship schemas", *Data and Knowledge Engineering* 11,1 (1993).
5. Castilho, J.M. "A State-Space Approach for Database Redesign", in Elmasri, R.A., Kouramajian, V. and Thalheim, B. (eds.), *Entity-Relationship Approach - ER'93*, Springer-Verlag (1994).
6. Chen, P.P., "The entity-relationship model: toward a unified view of data", *ACM TODS* 1, 1 (1976).
7. Elmasri, R. and Navathe, S. *Fundamentals of Database Systems*, 2nd ed. Benjamin Cummings (1994).
8. Hainaut, J-L., Tonneau C., Joris M. and Chandelon M. "Schema Transformation Techniques for Database Reverse Engineering", in Elmasri, R.A., Kouramajian, V. and Thalheim, B. (eds.), *Entity-Relationship Approach - ER'93*, Springer-Verlag (1994).
9. Liu, C.T., Chang, S.K. and Chrysanthis, P.K. "Database Schema Evolution using EVER Diagrams", *Proc. of the Workshop on Advanced Visual Interfaces*, Bari, Italy (June 1994).
10. Markowitz, V.M. and Makowsky, J.A. "Incremental Reorganization of Relational Databases", *Proc. 13th Int. Conf. on Very Large Data Bases*, Brighton, England (Sept. 1987).
11. Markowitz, V.M. and Shoshani, A. "Representing extended entity-relationship structures in relational databases: a modular approach", *ACM TODS* 17, 3 (Sept. 1992).
12. Silva, A.S. A Contribution to the Problem of Maintaining Optimized Relational Representations of Entity-Relationship Schemas, MSc Dissertation, Department of Computer Science, UFMG (1995). (in Portuguese)
13. 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).