Proceedings 12th IEEE Int. Conf. on Data Engineering ICDE'96, New-Orleans (USA),
February 26 - March 1, 1996, IEEE Press, pp. 218-227.

Towards the Reverse Engineering of Denormalized Relational Databases

J-M. Petit, F. Toumani, J-F. Boulicaut, J. Kouloumdjian
Laboratoire d'Ingénierie des Systèmes d'Information
INSA Lyon, 20 av. Albert Einstein, Bât. 501
F-69621 Villeurbanne cedex
e-mail: jean-marc.petit@lisi.insa-lyon.fr

Abstract

This paper describes a method to cope with denormalized relational schemas in a database reverse engineering process. We propose two main steps to improve the understanding of data semantics. Firstly we extract inclusion dependencies by analyzing the equi-join queries embedded in application programs and by querying the database extension. Secondly we show how to discover only functional dependencies which influence the way attributes should be restructured. The method is interactive since an expert user has to validate the presumptions on the elicited dependencies. Moreover, a restructuring phase leads to a relational schema in third normal form provided with key constraints and referential integrity constraints. Finally, we sketch how an Entity-Relationship schema can be derived from such information.

1 Introduction

The aim of a Database Reverse Engineering (DBRE) process is to improve the understanding of the data semantics. Many aspects of database evolution, especially for old databases where data semantics has been lost for years, require a DBRE process [7]. Such current situations are the re-engineering of the so-called legacy systems or the federation of distributed databases. Many works have already been done where a conceptual schema (often based on an extension of the Entity-Relationship (ER) model [4]) is derived from a hierarchical database [15, 2], a network database [2] or a relational database [3, 15, 13, 2, 21, 5]. A DBRE process is naturally split into two major steps [18]:

- Eliciting the data semantics from the existing system

  Various sources of information can be relevant for tackling this task, e.g., the physical schema, the database extension, the application programs, but especially expert users.

- Expressing the extracted semantics with a high level data model

  This task consists in a schema translation activity and gives rise to several difficulties since the concepts of the original model do not overlap those of the target model.

In the context of relational databases, most of the DBRE methods [15, 13, 21] focus only on the schema translation task since they assume that the constraints (e.g., functional dependencies or foreign keys) are available at the beginning of the process. However, to cope with real-life situations, such strong assumptions are not realistic since old versions of DataBase Management Systems (DBMSs) do not support such declarations.

Some recent works [19, 22, 1, 16] have proposed independently to alleviate the assumptions on the knowledge available a priori. Given a schema in third Normal Form (3NF), the key idea is to fetch the needed information from the data manipulation statements embedded in application programs. We have already interesting results in this direction [16, 17, 18]. Unlike [5], we do not constrain the relational schema with a *consistent naming of key attributes* and unlike [13, 21, 10, 9], we do not need to have all the structural constraints before applying the method.

A current assumption in existing DBRE methods, including our previous results, is to impose the relational schema to be in 3NF to ensure that each relation corresponds to a unique object of the application domain. Nevertheless, Johannesson has shown that several objects, the so-called *hidden objects*, can be encoded in a 3NF relation [10]. He introduces a formal framework to handle such cases in a DBRE process. Unlike Johannesson who still has strong assumptions

on the a priori knowledge, we propose in [18] a full method to cope with 3NF schemas while eliciting the needed knowledge from the application programs.

However, the 3NF requirement remains one of the major limits of current DBRE methods. Indeed, during the database design process, the relational schemas are often either directly produced in 1NF or in 2NF, or denormalized at the end of the design process. The denormalization occurs mainly:

- during the physical database implementation,
- during the maintenance phase when attributes are added.

The aim is generally to reduce the access time and to provide efficiency to end-users [6, 20].

This paper deals with the reverse engineering of relational schemas for which only 1NF is required, arguing that it is a major step towards real-life DBRE. Coping with such denormalized schemas in the DBRE context is, up to our knowledge, an open problem. Shoval and Shreiber [21] have investigated this problem, but with all the needed constraints at hand, whereas Anderson [1] has briefly addressed it.
We focus on the elicitation of the dependencies that enable to derive a new relational schema in 3NF with key constraints and referential integrity constraints. The translation of such a 3NF relational schema into Extended Entity-Relationship (EER) structures has been widely studied, e.g., in [13, 2, 5]. In the spirit of Markowitz and Makowsky [13], we consider the relational schemas that can be translated into conceptual schemas, by looking into the method which has been used to design them.
The key problems to apply DBRE techniques to a denormalized relational schema can be resumed as follows: identifying the relevant objects of the application domain, recovering the structure of each of these objects and eliciting the links (or relationships) between these objects.
To tackle these problems we propose to decompose the denormalized schemas into 3NF schemas where each relation maps exactly one object of the application domain. To achieve this restructuring task, we need to extract the functional dependencies which are meaningful for the application domain while they have not been conceptualized as relations. Hence the first difficulty is to find out the non-key attributes that correspond to *identifiers* of objects of the application domain. These attributes constitute the left hand side of relevant functional dependencies and are involved in interrelation dependencies.

These interrelation dependencies can be recovered by analyzing the equi-join queries. Indeed, given a relational database, the practitioners are not completely free from the *navigation problem* since they must specify the access paths among relations to define queries [11]. The thesis of this work is that understanding the *logical navigation* in a relational schema by analyzing the set of equi-join queries defined in application programs, enables to elicit the interrelation dependencies. Such interrelation dependencies will become inclusion dependencies regarding to the database extension.

The paper is organized as follows: Section 2 provides the basic relational concepts and the notations we use. We briefly recall in Section 3 the two main design approaches for relational databases and their influences on a DBRE process. The working assumptions are stated in Section 4. An example that is used throughout the paper is introduced in Section 5. Section 6 deals with data semantics elicitation: An operational process to discover a set of inclusion dependencies from the relational database is proposed; Next, an algorithm to get the left hand sides of candidate functional dependencies is given; Then the right hand sides of these candidate functional dependencies are found out. Given the original schema and the elicited data semantics, we present a schema restructuring process in Section 7. It facilitates the translation of the relational schema into an ER schema. Finally, conclusion and perspectives are given in Section 8.

2 Preliminaries

We give the basic relational concepts that are used throughout the paper. We define a relational database as $(R, \mathcal{E}, \Delta)$ with a set of relations R, a database extension $\mathcal{E}$ and a set of dependencies Δ over $(R, \mathcal{E})$.

A *relation* $R_i(X_i)$ belongs to R and is defined with a relation name R_i and a set of *attributes* X_i. At the semantic level, each relation $R_i(X_i)$ is associated with a *table* r_i and each attribute $x_i \in X_i$ is associated with a *domain* D_i. Each table is made of a set of *tuples* and each tuple belongs to $\mathcal{E}$. Thus, the database extension $\mathcal{E}$ represents the set of tables r_i. $r_i[Y]$ is the *projection* of the table r_i on a subset Y of X_i and $t[Y]$ is the projection of the tuple t following Y.

The *dependencies* over $(R, \mathcal{E})$ are denoted by $\Delta = (\mathcal{F} \cup \mathcal{IND})$ where $\mathcal{F}$ is the set of functional dependencies and $\mathcal{IND}$ the set of inclusion dependencies. Let $R_i(X_i)$ be a relation associated with the table r_i and let Y and Z be two subsets of X_i. A *functional dependency* denoted by $R_i : Y \rightarrow Z$ on $R_i(X_i)$ is satisfied by r_i iff $\forall t, t' \in r_i \quad t[Y] = t'[Y] \Rightarrow t[Z] = t'[Z]$.
Let $R_i(X_i)$ and $R_j(X_j)$ be two relations associated with tables r_i and r_j respectively. Let Y (resp. Z) be

a subset of attributes of X_i (resp. X_j). An *inclusion dependency* on $R_i(X_i)$ and $R_j(X_j)$ between Y and Z denoted by $R_i[Y] \ll R_j[Z]$ is satisfied by r_i and r_j iff $r_i[Y] \subseteq r_j[Z]$.
An element of $\mathcal{IND}$ whose right hand side is a key is called a *key-based* inclusion dependency or a *referential integrity constraint*. A *key* constraint K_i ($K_i \subseteq X_i$) on $R_i(X_i)$ denoted as $R_i : K_i \rightarrow X_i$ is a functional dependency whose right hand side is equal to X_i and no strict subset of K_i is a key. A *Not Null* constraint on Z denoted by not null Z is satisfied by r_i iff $\forall t \in r_i$, none of the values of $t[Z]$ is *null*.

Notations

A set of attributes will be denoted by an upper case letter (e.g., X or R.X to indicate the associated relation) whereas a single attribute will be denoted by a lower case letter (e.g., a or R.a). We write XY for $X \cup Y$, $X - Y$ for $X \backslash Y$ and the singleton set {a} is written a.
Let $\|.\|$ be a function from $\mathcal{E}$ to natural integers which counts the number of distinct tuples in a table regarding a set of attributes.
This function can be computed in any SQL-like language as follows:

$$\|r_i[X]\| \equiv \begin{array}{ll} \textsf{select} & \textsf{count distinct } X \\ \textsf{from} & R_i \end{array}$$

An *equi-join* between $R_i[Y]$ and $R_j[Z]$ is denoted by $R_i[Y] \bowtie R_j[Z]$.

3 The Problem

To highlight the difficulties when tackling a denormalized relational schema in a DBRE context, we look at the database design process and the normalization activity to evaluate how they influence each other.

Two main approaches to design a relational database exist. The first one, the *Universal Relation* (UR) approach, assumes that all the semantics is captured through various dependencies expressed over a *universal set* of attributes [11]. The normalization process of the universal relation is guided by the *functional dependencies* and can lead to a relational schema that does not match the intuition about how information should be organized in relations [24]. Batini et al. [2, p.161] argue that the dependencies in general are inappropriate to capture the requirements in the application domain. The main reason is that the functional dependencies used in a normalization process do not always express a relationship that is worth conceptualizing but can represent only an integrity constraint with no influence on the data organization [13].
It also implies that the relational schema obtained by the UR approach cannot be always translated by any DBRE method into a correct conceptual schema [13]. Fortunately, this is not a limit in practice since the UR approach is not frequently used to design real-life databases [6, p.435].

The second approach to design a relational database uses semantic data models [8] to describe the application domain at the conceptual level. This enables to reduce the *conceptual distance* between the application domain and its implementation in a relational DBMS. One of the most popular semantic models is the ER model [4]. Various algorithms [23, 14, 2] have been proposed to map an ER (or an EER) schema into a relational schema. Markowitz and Shoshani [14] have shown that the dependencies that are directly derivable from the EER schemas are key constraints and referential integrity constraints.
The resulting relational schemas match a representation of the information systems in terms of objects expressed by the relations, and interactions between these objects expressed by the interrelation dependencies. Therefore, DBRE methods can be applied on such relational schemas.

Characteristics of Denormalized Schemas

In a 1NF schema, a relation can represent many independent objects of the application domain. Consequently, some identifiers of these objects are represented by non key attributes in the denormalized schema. The main difficulty is to find out the relevant dependencies for both the normalization and the schema translation process, i.e., the inclusion dependencies that represent objects interactions and the functional dependencies that have a meaning in the application domain. The interrelation dependencies express interaction among objects and are not reduced to the referential integrity constraints in a 1NF schema since all *identifiers* of objects are not compulsorily mapped into *keys* in the relational schema.

4 Assumptions

We define the kind of relational database that is considered in this paper.

The purpose is to apply a DBRE method to a relational database $(R, \mathcal{E}, \emptyset)$, given its associated set $\mathcal{P}$ of application programs. The data manipulation statements that perform accesses to $\mathcal{E}$ are embedded into these application programs.
We classify the assumptions on each of these components and we make the user involvement as clear as possible by identifying when his/her knowledge is optional or mandatory.

On the relational schema

The relational schema is at least in 1NF and without any restriction on the naming of attributes.

On the available constraints

The dependencies known a priori on the data are those available in most of the DBMSs. So the available constraints on attributes are assumed to be unique and not null. Furthermore, these constraints are easily understood by the practitioners. As in standard SQL, a unique constraint implies a not null constraint (on each attribute involved).

We compute the set $\mathcal{K}$ of the key attributes and the set $\mathcal{N}$ of the null not allowed attributes as follows:

$\mathcal{K} = \{R.X$ such that X is declared unique$\}$

$\mathcal{N} = \{R.a$ such that a is declared not null$\} \cup \{R.a \in R.X$ such that $R.X \in \mathcal{K}\}$

The set $\mathcal{K}$ enables to compute directly the key constraints occurring in the schema. The expert user is not required to provide this information since it can be extracted from the data dictionary.

On the database extension

No assumption is made on the database extension. Even if this extension is not always a faithful snapshot of all the constraints that must hold in the database, we noticed that this prevents the expert user from tedious manual tasks [5, 18].

On the application programs

The set $\mathcal{P}$ represents the application part of the relational database in operation. For our current purpose, we extract from $\mathcal{P}$ only the equi-join queries. It gives a set denoted by $\mathcal{Q}$.

Extracting automatically the equi-joins from a set of files is not a trivial task. For instance, an equi-join can be performed in different ways, with nested or unnested queries, with a where clause or with an intersect operator.

Let us illustrate how the set $\mathcal{Q}$ is augmented when using the following unnested query involving a where clause:

$$\left|\begin{array}{ll} \ldots & \\ \text{from} & R_k, R_l \\ \text{where} & a^k_{i_1} = a^l_{j_1} \text{ and } \ldots \text{and } a^k_{i_n} = a^l_{j_n} \end{array}\right.$$

$A_k = \{a^k_{i_1} \ldots a^k_{i_n}\}$ and $A_l = \{a^l_{j_1} \ldots a^l_{j_n}\}$

$\Rightarrow \mathcal{Q} = \mathcal{Q} \cup \{R_k[A_k] \bowtie R_l[A_l]\}$

While the extraction of equi-joins between single attributes remains simple, it becomes complex when equi-joins between sets of attributes have to be discovered. However, it is out of the scope of this paper and we assume that such a set is available, i.e., it has been computed.

To sum up, the application of the method requires only a relational database $(R, \mathcal{E}, \emptyset)$ and the three sets $\mathcal{K}, \mathcal{N}$ and $\mathcal{Q}$.

5 An Introductory Example

We propose the following example to demonstrate how the method works. This database manages the employees of an organization who work in projects assigned in different departments. Each relation name begins with an upper-case letter, each key constraint on attribute(s) is underlined and each not null constraint on attribute(s) is emphasized. A possible relational schema could be defined as follows in the data dictionary of the DBMS:

Person(id,name, street, number, zip-code, state)	2NF
HEmployee(no, date, salary)	3NF
Department(dep, emp, skill, *location*, proj)	2NF
Assignment(emp, dep, proj, date, project-name)	1NF

The normal form of each relation is given as comment for clarity purpose. So the sets of constraints $\mathcal{K}$ and $\mathcal{N}$ are computed:

$\mathcal{K}$ = {Person.{id}, HEmployee.{no,date}, Department.{dep}, Assignment.{emp,dep,proj}}

$\mathcal{N}$ = {Department.location, Person.id, HEmployee.no, HEmployee.date, Department.dep, Assignment.dep, Assignment.emp, Assignment.proj} .

Let us assume that the following set $\mathcal{Q}$ of equi-joins has been extracted from the application programs (e.g., forms, reports, batch files).

$$\mathcal{Q} = \left\{\begin{array}{lcl} \text{HEmployee[no]} & \bowtie & \text{Person[id]} \\ \text{Department[emp]} & \bowtie & \text{HEmployee[no]} \\ \text{Assignment[emp]} & \bowtie & \text{HEmployee[no]} \\ \text{Assignment[dep]} & \bowtie & \text{Department[dep]} \\ \text{Department[proj]} & \bowtie & \text{Assignment[proj]} \end{array}\right\}$$

The database extension $\mathcal{E}$ is assumed to be correct with respect to the constraints defined in the data dictionary.

Our goal is to transform this schema so that each relation maps exactly one object of the application domain. So we need to select only the relevant functional dependencies for the schema restructuring. For instance, let us assume that the two following functional dependencies Assignment: proj $\rightarrow$ project-name and Person: zip-code $\rightarrow$ state occur in the schema.

Since programmers have referred to the attribute Assignment[proj] by the equi-join Department[proj] $\bowtie$ Assignment[proj], the first functional dependency will be discovered by our method and will be relevant during the schema restructuring process.

On the other hand, the second functional dependency is only an integrity constraint which will not be considered further (programmers do not refer to Person[zip-code] in an equi-join). It is worth noting that keeping the relation Person in 2NF does not imply *update*

anomalies [2]. Moreover, the use of this kind of functional dependency during a normalization process can lead to an erroneous design [13].

We use this example throughout the paper by pointing out the various steps that lead to a conceptual schema (Section 7).

6 Data Semantics Elicitation

We now focus on the two main steps required to carry out a DBRE process on a denormalized relational schema: 1) how to recover interrelation dependencies from a relational database in operation and 2) how functional dependencies, which influence the way data could be structured, are elicited.

6.1 Inclusion Dependency Elicitation

An algorithm that discovers the interrelation dependencies between attributes of the relational schema is defined. Such interrelation dependencies are either inclusion dependencies or non-empty intersections between the two sets of values of the attributes. We elicit them by scanning the equi-joins of $\mathcal{Q}$ and by accessing the database extension $\mathcal{E}$. Roughly speaking, an equi-join $A \bowtie B$ is a mean to say that attributes A and B *share something* and thus it allows to express interrelation dependencies.

The idea is to establish against the database extension $\mathcal{E}$ whether the related attributes are in inclusion dependencies or not. To cope with corrupted database extensions, the expert user is involved to decide if non empty intersections can not be reduced to inclusion dependencies.

The equi-join analysis focuses on relevant attributes enforcing the efficiency of the inclusion dependencies elicitation.

IND-Discovery Algorithm

- Input: $R, \mathcal{E}, \mathcal{Q}$
- Output: $\mathcal{IND}$ a set of inclusion dependencies, $\mathcal{S}$ a set of new relations

```
Begin
IND = ∅;
S = ∅;
for each q ∈ Q do
  Let q = R_k[A_k] ⋈ R_l[A_l];
  N_k = ||r_k[A_k]||;
  N_l = ||r_l[A_l]||;
  N_kl = ||r_k[A_k] ⋈ r_l[A_l]||;
  if N_kl = 0 then
    IND left unchanged                          (i)
  elsif N_kl = N_k or N_kl = N_l then
    if N_k ≤ N_l then
      IND = IND ⊔ {R_k[A_k] ≪ R_l[A_l]};        (ii)
    fi
    if N_l ≤ N_k then
      IND = IND ⊔ {R_l[A_l] ≪ R_k[A_k]};        (iii)
    fi
  elsif N_kl ≠ N_k and N_kl ≠ N_l then
    /* A Non-Empty Intersection (NEI) between the
    sets of values of A_k and A_l is discovered */
    if the expert user conceptualizes this NEI
    with R_p(A_p) then
      Add the new relation R_p(A_p) to S;       (iv)
      IND = IND ⊔ {R_p[A_p] ≪ R_k[A_k]}
                ⊔ {R_p[A_p] ≪ R_l[A_l]}
    else the expert user can choose among these
    three statements:
      IND = IND ⊔ {R_l[A_l] ≪ R_k[A_k]};        (v)
      IND = IND ⊔ {R_k[A_k] ≪ R_l[A_l]});       (vi)
      IND left unchanged;                       (vii)
    fi
  fi
od
End
```

When the intersection between the two sets of values of the attributes is empty (i), a data integrity problem can occur and no interrelation dependency can be elicited.

Otherwise, when this intersection is equal to one of the two sets of values, an inclusion dependency is elicited ((ii) or (iii)).

Finally when a non empty intersection, which is distinct from the two previous sets of values, exists, the expert user is involved. Regarding the amount of data implied in this intersection in comparison with these two sets of values, the expert user has to decide whether it is worth adding a new relation into S to conceptualize this intersection. In this case (iv), he/she thinks that the database extension is a faithful snapshot of the constraints for these attributes. If he/she decides not to create a new relation, he/she disregards the database extension (data integrity problems can occur) and we give to him/her an alternative:

- the considered non empty intersection becomes an inclusion dependency, i.e., $R_l[A_l] \ll R_k[A_k]$ (v) or the inverse (vi),
- the non empty intersection is ignored (vii).

The former takes into account the interrelation dependencies as inclusion dependencies whereas the latter ignores it. The expert user is warned about the risk to give up a non empty intersection.

Whatever the choice made by the expert user, the obtained data structure no longer matches the database extension.

On the example of Section 5, we illustrate how the interrelation dependencies can be recovered using the IND-Discovery algorithm.
Let us detail how the equi-joins HEmployee[no] $\bowtie$ Person[id] and Assignment[dep] $\bowtie$ Department[dep] are processed. In the first case, we assume that querying the database extension returns the following valuations of the relation HEmployee and Person:

$$||\text{Person[id]}|| = 2200$$
$$||\text{HEmployee[no]}|| = 1550$$
$$||\text{Person[id]} \bowtie \text{HEmployee[no]}|| = 1550$$

Hence, a new inclusion dependency (HEmployee[no] $\ll$ Person[id]) is elicited and added to the set $\mathcal{IND}$.

Now let us assume that the processing of the second equi-join reveals a non-empty intersection between the values of Assignment[dep] and Department[dep]:

$$||\text{Assignment[dep]}|| = 30$$
$$||\text{Department[dep]}|| = 28$$
$$||\text{Department[dep]} \bowtie \text{Assignment[dep]}|| = 20$$

Then, let us consider that the expert user wants to conceptualize the departments which are assigned to both projects and employees in the relation Assignment. Thus, a new relation Ass-Dept(dep) is created and added to $\mathcal{S}$. Note that the choice of names for the new relations must be significant with respect to the application domain. The inclusion dependencies Ass-Dept[dep] $\ll$ Assignment[dep] and Ass-Dept[dep] $\ll$ Department[dep] are added to the set $\mathcal{IND}$.
Finally, at the end of the IND-Discovery processing, the set of inclusion dependencies $\mathcal{IND}$ is equal to:

$$\left\{\begin{array}{lcl} \text{HEmployee[no]} & \ll & \text{Person[id]} \\ \text{Department[emp]} & \ll & \text{HEmployee[no]} \\ \text{Assignment[emp]} & \ll & \text{HEmployee[no]} \\ \text{Ass-Dept[dep]} & \ll & \text{Assignment[dep]} \\ \text{Ass-Dept[dep]} & \ll & \text{Department[dep]} \\ \text{Department[proj]} & \ll & \text{Assignment[proj]} \end{array}\right\}$$

When a key occurs in an inclusion dependency, it is underlined. The set of relations $\mathcal{S}$ is equal to {Ass-Dept(dep)}.

6.2 Functional Dependency Elicitation

The elicitation of the functional dependencies allows to cope with the relations that are not explicitly represented in the current denormalized schema. Given a denormalized relational database $(R + \mathcal{S}, \mathcal{E}, \mathcal{IND})$ and the sets $\mathcal{K}$ and $\mathcal{N}$, we build the set $\mathcal{H}$ of hidden objects and the set $\mathcal{F}$ of functional dependencies which are not directly derivable from $\mathcal{K}$. The set $\mathcal{H}$ is intended to capture the relevant functional dependencies which have an empty right hand side.
There are two main steps to achieve this elicitation: 1) studying the set of inclusion dependencies to elicit candidate left hand sides of the relevant functional dependencies, 2) for each of them, recovering its right hand side.

6.2.1 Extracting the Candidate Left Hand Sides of Functional Dependencies

The following LHS-Discovery algorithm computes the set $\mathcal{LHS}$ of candidate left hand sides of functional dependencies and the set $\mathcal{H}$ of hidden objects. It starts by scanning the set $\mathcal{IND}$ to find out all the non key attributes implied in an inclusion dependency. Indeed, such non key attributes could be conceptualized as elements of $\mathcal{LHS}$ or $\mathcal{H}$.

LHS-Discovery Algorithm

- Input: $R + \mathcal{S}, K, \mathcal{IND}$
- Output: $\mathcal{LHS}, \mathcal{H}$

```
Begin
LHS = ∅; H = ∅;
for each I ∈ IND do
 let I = (Rk[Ak] ≪ Rl[Al]);
 if Rk(Ak) ∈ S then
  if Rl.Al ∉ K then H = H ∪ {Rl.Al}; fi        (i)
 elsif
  let Kk and Kl be the keys of Rk and Rl;
  if (Ak ≠ Kk) then
   LHS = LHS ∪ {Rk.Ak}; fi                     (ii)
  if (Al ≠ Kl) then
   LHS = LHS ∪ {Rl.Al}; fi                     (iii)
 fi
od
End
```

For each inclusion dependency, two cases arise:

- A relation of $\mathcal{S}$ is involved[1]: if the right hand side of this inclusion dependency is not a key (i), then it is added to the set $\mathcal{H}$ of hidden objects. These attributes must be conceptualized since the expert user has already decided to conceptualize a subset of their values (represented by this relation of $\mathcal{S}$).
- No relation of $\mathcal{S}$ is involved: if non key attributes occur (ii) (iii) then they become elements of $\mathcal{LHS}$. These attributes are candidate identifiers of the objects which are not explicitly represented by relations into the schema.

[1] By construction, this relation is compulsorily in the left hand side of this inclusion dependency

Let us illustrate on the example of Section 5, how the sets $\mathcal{H}$ and $\mathcal{LHS}$ are computed. For instance, Assignment.{dep} involved in the inclusion dependency Ass-Dept[dep] $\ll$ Assignment[dep] becomes an element of $\mathcal{H}$ since Ass-Dept belongs to $\mathcal{S}$ whereas HEmployee.{no} becomes an element of $\mathcal{LHS}$ due to the inclusion dependency HEmployee[no] $\ll$ Person[id]. Given the set $\mathcal{IND}$ of Section 6.1, the complete execution of the LHS-Discovery algorithm provides the two following sets:
$\mathcal{LHS}$ = {HEmployee.{no}, Department.{emp}, Assignment.{emp}, Assignment.{proj}, Department.{proj}}
$\mathcal{H}$ = {Assignment.{dep}}

6.2.2 Extracting the Right Hand Side of Functional Dependencies

The discovery of the right hand side of a functional dependency can be carried out in several ways, i.e., by relying on the expert user, by querying the extension $\mathcal{E}$ [12] or by extracting clues from application programs which is in the spirit of our approach.

Let us assume that $R_i.A \in (\mathcal{LHS} \cup \mathcal{H})$ and that we are looking for functional dependencies in the relation $R_i(X_i)$. The candidate attributes for the right hand side of $R_i.A$ are included in X_i.
The first step of the RHS-Discovery algorithm is to decrease the number of candidate attributes for the right hand side (e.g., the keys are deleted from the candidate attributes since we want to meet the 3NF requirement only). The second step consists of testing for each candidate attribute whether a functional dependency exists. The effective computation of each possible functional dependency (cf. (i)) is not detailed.

```
RHS-Discovery algorithm
  • Input:  R, E, LHS, H
  • Output: F, H
Begin
F = ∅;
for each R_i.A ∈ (LHS ∪ H) do
  let K_i be the key of R_i(X_i);
  /* Decreasing the number of right hand side
  attributes */
  T = X_i − AK_i;
  if A ∉ N then T = T − (N ∩ X_i); fi;
  /* Computing the possible functional
  dependencies */
  B = ∅;
  for each attribute b ∈ T do
     if A → b holds in r_i then B = Bb          (i)
     else the expert user can enforce B = Bb;(ii)
     fi
  od
  if B ≠ ∅ then                                 (iii)
     F = F ∪ {R_i : A → B};
     if R_i.A ∈ H then H = H \{R_i.A}; fi
  elsif R_i.A ∉ H then
     /* A Hidden Object (HO) can be elicited */
     if the expert user conceptualizes R_i.A then
       H = H ∪ {R_i.A}; fi                      (iv)
  else R_i.A is not considered;                 (v)
  fi
od
End
```

To account for data integrity problems of the database extension $\mathcal{E}$ and regarding the amount of data which are implied, the expert user still can enforce a functional dependency (ii).
Once a presumption of functional dependency (the sets $\mathcal{LHS}$ and $\mathcal{H}$) has been obtained from an inclusion dependency, we have to find out, in interaction with the expert user, if this functional dependency truly occurs. Three cases exist: 1) if a functional dependency is elicited (iii) indicating the existence of a denormalized relation, then it becomes an element of the set $\mathcal{F}$ of functional dependencies. If the left hand side of the elicited functional dependency has been assigned to $\mathcal{H}$ during the LHS-discovery algorithm, then it has to be removed from $\mathcal{H}$ since it is now conceptualized in $\mathcal{F}$. 2) If an empty right hand side occurs and if the user decides to conceptualize this hidden object (iv), then it is added to $\mathcal{H}$. 3) Otherwise, the expert user decides to ignore this information (v).

On the example of Section 5, assume that the RHS-Discovery algorithm is applied to find out if a right hand side exists for each element of $\mathcal{LHS} \cup \mathcal{H}$.
Let us assume that the element Department{emp} from $\mathcal{LHS}$ is processed. The candidate attributes for the right hand side are dep, skill, location and proj. The attributes dep and location are then removed from the candidate attributes since dep is a key and location is not null (emp having null values). The remaining attributes are skill and proj. Assume that the functional dependency Department: emp $\rightarrow$ skill, proj is found out and, after being validated by the expert user, it is added to $\mathcal{F}$.

At the end, if the expert user has decided to conceptualize the element HEmployee{no} of $\mathcal{LHS}$ which represents the object Employee[2] of the application domain, the following sets are obtained:

$$\mathrm{F}=\left\{\begin{array}{lll} \text{Department: emp} & \rightarrow & \text{skill, proj} \\ \text{Assignment: proj} & \rightarrow & \text{project-name} \end{array}\right\}$$

$\mathcal{H}$ = {HEmployee{no}, Assignment{dep}}.

[2]This is an object embedded in the 3NF relation HEmployee [10]

The remaining attributes of $\mathcal{LHS}$ are not conceptualized. It indicates that the attributes Assignment{emp} and Department{proj} of $\mathcal{LHS}$ have been given up by the expert user.

7 Towards a Conceptual Schema

At this point, the end of the knowledge elicitation phase is reached since the needed knowledge to perform a reverse engineering process on a 1NF relational schema is available. Indeed, the inclusion dependencies, the functional dependencies and the hidden objects elicited by IND-Discovery, LHS-Discovery and RHS-Discovery algorithms enable to normalize the 1NF schema in order to get a 3NF schema. This is a usual requirement to translate relational schemas into conceptual structures [13, 10].

The Restruct algorithm, introduced below, gives the main steps to restructure a 1NF relational schema given the set $\mathcal{IND}$ of inclusion dependencies, and the set $\mathcal{F}$ of functional dependencies. This restructuring provides a 3NF schema that can be represented by an EER schema.

Restruct Algorithm

- Input: $R \cup \mathcal{S}, \mathcal{K}, \mathcal{F}, \mathcal{H}, \mathcal{IND}$
- Output: $R \cup \mathcal{S}, \mathcal{K}, \mathcal{RIC}$

```
Begin
RI = ∅;
/* Eliciting the hidden objects */
for each R_i.A_i ∈ H do
   add the new relation R_p(A_i) to S;
   add R_p.A_i to K;
   IND = IND ∪ {R_i[A_i] ≪ R_p[A_i]};
   replace R_i[A_i] by R_p[A_i] in IND;
od
/* Splitting the schema using FDs */
for each f_i ∈ F do
   let f_i = R_i : A_i → B_i;
   add the new relation R_p(A_iB_i) to S;
   add R_i.A_i to K;
   remove B_i from R_i(X_i);
   IND = IND ∪ {R_i[A_i] ≪ R_p[A_i]};
   replace R_i[A_i] by R_p[A_i] and R_i[B_i] by R_p[B_i]
   in IND;
od
/* Computing RIC */
RIC = {R_i[A_i] ≪ R_j[A_j] ∈ IND such that
R_j.A_j ∈ K};
End
```

On the example of Section 5, the elements HEmployee[no] and Assignment[dep] of $\mathcal{H}$ become two relations Employee(no) and Other-Dep(dep) respectively.

Assume the functional dependency Department: emp $\rightarrow$ skill proj of $\mathcal{F}$ is currently processed. Consider that the expert user chooses to call this new relation Manager, its structure being Manager(emp, skill, proj). The structure of Department becomes Department(dep, emp, *location*) since the attributes skill and proj have been removed.
Therefore, the modifications of the set $\mathcal{IND}$ are the following: 1) a new inclusion dependency Department[emp] $\ll$ Manager[emp] is created;
2) assume the functional dependency : Assignment: proj $\rightarrow$ project-name has already been processed and has given rise to the relation Project(proj, project-name), then the modification on $\mathcal{IND}$ leads to replace Assignment[proj] $\ll$ Project[proj] by Manager[proj] $\ll$ Project[proj].

The application of the Restruct algorithm on the schema R provided with the dependencies $(\mathcal{F}, \mathcal{IND})$ leads to the following restructured schema:

Person(id, name, street, number, zip-code, city)
HEmployee(no, date, salary)
Department(dep, emp, *location*)
Assignment(emp, dep, proj, date)
Employee(no)
Ass-Dept(dep)
Other-Dept(dep)
Manager(emp, skill, proj)
Project(proj, project-name)

The set $\mathcal{K}$ contains the underlined sets of attributes and we obtain the following set of referential integrity constraints:

$$\mathcal{RIC} = \left\{ \begin{array}{lcl} \text{Employee[no]} & \ll & \text{Person[id]} \\ \text{Manager[emp]} & \ll & \text{Employee[no]} \\ \text{Assignment[emp]} & \ll & \text{Employee[no]} \\ \text{Ass-Dept[dep]} & \ll & \text{Other-Dept[dep]} \\ \text{Assignment[dep]} & \ll & \text{Other-Dept[dep]} \\ \text{Ass-Dept[dep]} & \ll & \text{Department[dep]} \\ \text{Manager[proj]} & \ll & \text{Project[proj]} \\ \text{Hemployee[no]} & \ll & \text{Employee[no]} \\ \text{Department[emp]} & \ll & \text{Manager[emp]} \\ \text{Assignment[proj]} & \ll & \text{Project[proj]} \end{array} \right\}$$

This restructured schema can be translated into EER structures [21, 5, 9].

Finally, we sketch the Translate algorithm which allows to achieve the mapping of the restructured relational schema into EER structures. We only give a flavor (e.g., the treatment of cyclic inclusion dependencies is not considered here) of how EER constructs are obtained without going into details (see [13]).

The target model is the ER model extended to the *Specialization/Generalization* of object-types[3].

Translate Algorithm (sketch)

- **Input:** $R \sqcup S$, $\mathcal{K}$, $\mathcal{RIC}$
- **Output:** an EER schema

Begin

/ Mapping the relational schema into the EER structures */*

- map each relation of $R \sqcup S$ into an object-type;

/ Identifying EER object-types */*

- for each $R_l[A_l] \ll R_k[A_k] \in \mathcal{RIC}$ do

  a) if $A_l \in \mathcal{K}$ then an is-a link is elicited;
  b) if A_l forms a partition each element of which appears as left hand side of an element of $\mathcal{RIC}$ then a many-to-many relationship-type is elicited else a weak entity-type is elicited;
  c) if $A_l \notin \mathcal{K}$ then a binary relationship-type is elicited;

End

The resulting EER schema is depicted in Figure 1 where the entity-types are denoted by rectangles; relationships by diamond shaped boxes; weak entity-types by double boxes and is-a links by arrows with two pointers at their head.

8 Discussion

We have investigated the open problem of reverse engineering of denormalized relational databases into conceptual schemas, assuming only weak assumptions on the knowledge available a priori.
We have proposed a method to elicit the needed knowledge to restructure a 1NF schema into a 3NF schema by an analysis of equi-join queries embedded in application programs. In this sense, this work is a significant extension of our previous results [16, 18]. The retained constraints at the beginning of the method are neither supplied by an expert user, nor derivable from strong naming conventions on attributes. Here, the expert user is involved only for validation purposes.
It is worth noting that our method can be integrated as a front-end of all the existing relational DBRE methods.

[3] Object-type denotes either entity-type or relationship-type.

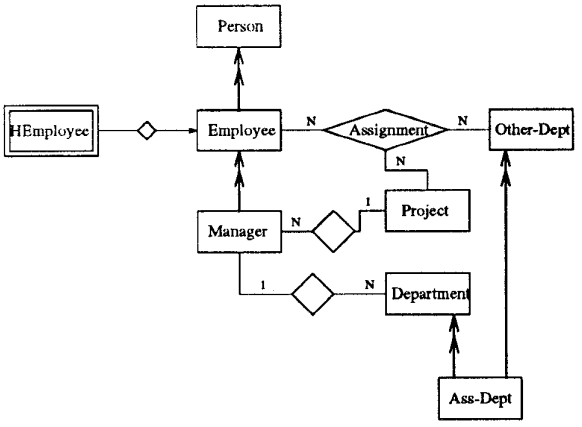

Figure 1: The final EER schema

This ongoing research is part of the project DREAM (Database Reverse Engineering Analysis Method) that defines an operational method to reverse-engineer real-life relational databases [16, 17, 18].

Moreover, this work can lead to original developments in the area of knowledge discovery in databases. Indeed, it seems to be quite interesting to study the influence of the application programs of legacy systems on traditional data mining processes. The application programs of databases could be considered as *oracles* that help to discover the relevant information into the data mines.

References

[1] M. Anderson. Extracting an ER Schema from a Relational Database Through Reverse Engineering. In *Proc. of the* 13^{th} *Int. Conf. on the ER Approach*, volume 881 of *LNCS*, pages 403–419, Manchester, Dec. 1994. Springer-Verlag.

[2] C. Batini, S. Ceri, and S. Navathe. *Conceptual Database Design: an Entity-Relationship Approach*. Benjamin Cummings, 1992.

[3] M.A. Casanova and J.E.A. de Sá. Designing Entity-Relationship Schemes for Conventional Information Systems. In *Proc. of the* 3^{td} *Int. Conf. on the ER Approach to Software Engineering*, pages 265–277, Anaheim, California, 1983. Elsevier Science Publishers.

[4] P.P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, Mar. 1976.

[5] R.H.L. Chiang, T.M. Barron, and V.C. Storey. Reverse Engineering of Relational Databases: Extraction of an EER Model from a Relational Database. *Data and Knowledge Engineering*, 10(12):107–142, 1994.

[6] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. Benjamin Cummings, second edition, 1994.

[7] J-L. Hainaut, V. Englebert, J. Henrard, J-M. Hick, and D. Roland. Requirements for Information System Reverse Engineering Support. In *Proc. of the IEEE Working Conference on Reverse Engineering*, Toronto, Canada, Jul. 1995. IEEE Computer Society.

[8] R. Hull and R. King. Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys*, 19(3):201–260, Sep. 1987.

[9] M. Jeusfeld and U. Johnen. An Executable Meta Model for Re-Engineering of Database Schemas. In *Proc. of the* 13^{th} *Int. Conf. on the ER Approach*, volume 881 of *LNCS*, pages 533–547, Manchester, Dec. 1994. Springer-Verlag.

[10] P. Johannesson. A Method for Transforming Relational Schemas into Conceptual Schemas. In *Proc. of the* 10^{th} *Int. Conf. on Data Engineering*, pages 190–201, Houston, Texas, Feb. 1994. IEEE Computer Society.

[11] D. Maier, J.D. Ullman, and M. Y. Vardi. On the Foundations of the Universal Relation Model. *ACM Transactions on Database Systems*, 9(2):283–308, Jun. 1984.

[12] H. Mannila and K-J. Räihä. Algorithms for Inferring Functional Dependencies from Relations. *Data and Knowledge Engineering*, 12:83–99, 1994.

[13] V.M. Markowitz and J.A. Makowsky. Identifying Extended Entity-Relationship Object Structures in Relational Schemas. *IEEE Transactions on Software Engineering*, 16(8):777–790, Aug. 1990.

[14] V.M. Markowitz and A. Shoshani. Representing Extended Entity-Relationship Structures in Relational Databases: A Modular Approach. *ACM Transactions on Database Systems*, 17(3):423–464, Sep. 1992.

[15] S. Navathe and A. Awong. Abstracting Relational and Hierarchical Data with a Semantic Data Model. In *Proc. of the* 6^{th} *Int. Conf. on the ER Approach*, pages 277–305, New-York, Nov. 1987.

[16] J-M. Petit, J. Kouloumdjian, J-F Boulicaut, and F. Toumani. Using Queries to Improve Database Reverse Engineering. In *Proc. of the* 13^{th} *Int. Conf. on the ER Approach*, volume 881 of *LNCS*, pages 369–386, Manchester, Oct. 1994. Springer-Verlag.

[17] J-M. Petit and F. Toumani. Taxonomic Reasoning in a Database Reverse Engineering Process. Research Report, 31 pages RR-94-45, LISI, Oct. 1994.

[18] J-M. Petit, F. Toumani, and J. Kouloumdjian. Relational Database Reverse Engineering: a Method Based on Query Analysis. *International Journal of Cooperative Information Systems*, 4(2,3):287–316, 1995.

[19] W.J. Premerlani and M. Blaha. An Approach for Reverse Engineering of Relational Databases. *Communications of the ACM*, 37(5):42–49, May 1994.

[20] U. Rogers. Denormalization: Why, What, and How? *Database Programming and Design*, 2(12):46–53, Dec. 1989.

[21] P. Shoval and N. Shreiber. Database Reverse Engineering: From the Relational to the Binary Relationship Model. *Data and Knowledge Engineering*, 10(10):293–315, 1993.

[22] O. Signore, M. Loffredo, M. Gregori, and M. Cima. Reconstruction of ER Schema from Database Applications: a Cognitive Approach. In *Proc. of the* 13^{th} *Int. Conf. on the ER Approach*, volume 881 of *LNCS*, pages 387–402, Manchester, Oct. 1994. Springer-Verlag.

[23] T.J. Teorey, Y. Dongqing, and J.P. Fry. A Logical Design Methodology for Relational Databases Using the Extented Entity-Relationship Model. *ACM Computing Surveys*, 18(2):197–222, Jun. 1986.

[24] J.D. Ullman. *Principles of Database Systems*. Computer Science Press, 1980.