# Elicitation and Conversion of Hidden Objects and Restrictions in a Database Schema

Laura C. Rivero
INTIA – U.N.Centro
LINTI – U.N.La Plata
B.A. Argentina
Phone: +54 2293 440363
lrivero@exa.unicen.edu.ar

Jorge H. Doorn
INTIA – U.N.Centro
B.A. Argentina
Phone: +54 2293 440363
jdoorn@exa.unicen.edu.ar

Viviana E. Ferraggine
INTIA – U.N.Centro
B.A. Argentina
Phone: +54 2293 440363
vferra@exa.unicen.edu.ar

## ABSTRACT

Mapping a database schema from one model into another, with a higher semantic capacity, is a current research subject with application in several development fields, such as schema integration and translation, migration from legacy systems and reengineering of poor quality or no-longer accurate data models. Inclusion dependencies are one of the most important concepts in relational databases and they are the key to perform some reengineering of database schemas. Referential integrity restrictions (*rir*), a particular case of an inclusion constraint, requires that the set of distinct values occurring in some specified column, simple or composite (foreign key), must be a subset of the set of distinct primary key values drawn from the same domain. Pure inclusion dependencies (*id*), however, may apply between other pairs of attributes also (alternate keys or non-keys). Database schemas containing *ids* frequently reveal the presence of hidden objects and misrepresented relationships and, as a consequence, increase the effort to develop program applications and maintain the integrity. This work presents a heuristics for the conversion of schemas with *ids* into equivalent schemas with only *rirs*. In case some irreducible *ids* remain, a semantic interpretation of their necessity and maintenance is given.

## Keywords

Database conceptual schema reengineering, pure inclusion dependencies, denormalization.

## 1. INTRODUCTION

In many organizations, there are databases that have evolved over the last years. In those systems, the exact understanding of data has been often lost or it is no longer accurate thus restraining their effective utilization by the organization because of the poor semantic quality of the schemas. In other circumstances some dependencies were been detected or foreseen at design time and then they were forgotten or misrepresented.

The reengineering of available software systems is absolutely necessary in these situations. Software tools, guides, heuristics, etc. could make an important difference in the process of quality improvement.

Conceptual schemas of actual databases holding these design flaws frequently contain non-key-based inclusion dependencies which reveal the presence of hidden objects and obscure interobject dependencies.

As it can be read in [4], "... referential integrity is a special case of inclusion dependencies. These constraints require that the set of distinct values occurring in some specified column, simple or composite, must be a subset of the values occurring in some other specified column (simple or composite, respectively). In the case of referential integrity, the set of distinct foreign key values should be a subset of the set of primary key values drawn from the same domain."

Formally, an inclusion dependency (*id*) is an expression $R_l[X] \subseteq R_r[Z]:(\alpha,\beta,\mu_l,\mu_r)$. $R_l$ and $R_r$ are relation names (possibly the same); $R_l[X]$ and $R_r[Z]$ are named the inclusion dependency left and right sides respectively. X and Z are lists of compatible attributes. $\alpha$, $\beta$, $\mu_l$ and $\mu_r$ are the referential actions for insertions (into $R_l$), deletions (from $R_r$) and updates (over $R_l$ and $R_r$, respectively). Referential actions are strategies to cope with violations of referential integrity. The following actions have been suggested: *Cascade*, *Restrict*, *No Action*, *Set Default* and *Set Null*. When Z is the primary key of $R_r$, the *id* is key-based (also named a referential integrity restriction, *rir*). In this case, X constitutes a foreign key for $R_l$. Thus a *rir* is denoted as $R_l[FK_l] << R_r[K_r]:(\alpha,\beta,\mu_l,\mu_r)$. $K_i$ stands for a candidate key over $R_i$ and $FK_i$ represents a foreign key for $R_i$ [1], [4]. From now on *id* is used only for the non-key-based inclusion dependencies and *rir* for the key-based ones.

Nevertheless, inclusion constraints can be defined over other pairs of attributes or sets of attributes (e.g. non keys). When declared and enforced, they may reflect business rules. However, they frequently appear either implementing relationships between misrepresented objects or revealing somehow undefined business policies. Nonetheless, the schema can be restored moving it back to a well-designed and supported schema.

Key-based inclusion dependencies can be declaratively defined (FOREIGN KEY ... REFERENCES ...) and enforced by most current database systems. Moreover, even though systems in compliant with this standard provide the FOREIGN KEY clause, they do not support all possible referential actions. Referential actions for insertions and updates over the left table are *Restrict*


SAC 2002, Madrid, Spain

by default in all DBMSs. On the other hand, non-key inclusion dependencies, when they have been recognized, are usually defined with CHECK statements or triggers. Since *ids* cannot be defined in the same way *rirs* are, their presence frequently promote an extra effort for the development of application programs and integrity maintenance.

Low quality designs having *ids* are common, thus it is necessary to transform those hard-to-maintain denormalized schemes into schemas with only key-based references, preserving the same information content.

This research work is devoted to give support to the reengineering of the actual system recovering the specification of the implicit (not explicitly declared) and explicit (declared via the DDL) structures and constraints of the persistent data of information systems. In this proposal, the configurations of the left and right sides of the inclusion dependencies are considered, at first, from a syntactic viewpoint.

After that, a set of hypotheses related to the presence of non-key inclusion dependencies is developed, highlighting their possible origin from a semantic point of view. Next, a heuristics based on these hypotheses is presented.

It allows the conversion of the conceptual schema into a normalized one by showing omitted objects and hidden relationships. Non-key inclusion dependencies are converted into key-based inclusion dependencies until a point of irreducibility is reached. To conclude, the context conditions that specify whether the remaining non-key inclusion dependencies (*rids*) can be ignored or maintained are established and an alternative declarative specification of *rids* is given.

To obtain a better conceptual schema two main phases must be completed:

a) Get the hidden, poorly specified knowledge recorded in the database, highlighting embedded and omitted objects and missing interobject relationships.

b) Augment the base of metadata with new and reformulated relation-schemes and transformed constraints with the obtained knowledge.

Figure 1 shows this process. The top box corresponds to the first phase and the following two to the second phase.

It should be noticed that a database design "strictly" adhering to a design methodology only produces referential integrity restrictions, but an ad-hoc refinement of the logical design without concerning the corresponding conceptual design usually leads to the modeling of non-key inclusion dependencies. While simple schemes may be treated with a few transformation rules, complex designs need formal or semiformal methods to carry out the conversion process.

Traditionally, database dependencies have been considered to be part of the data model, however, in a scenario such as the ones described at the beginning of this section, they may (and frequently must) also be retrieved from the extensional data. From now on it is supposed that those non-key based functional dependencies revealing the presence of hidden objects in the right table have been detected, and that the whole schema has some degree of normalization.

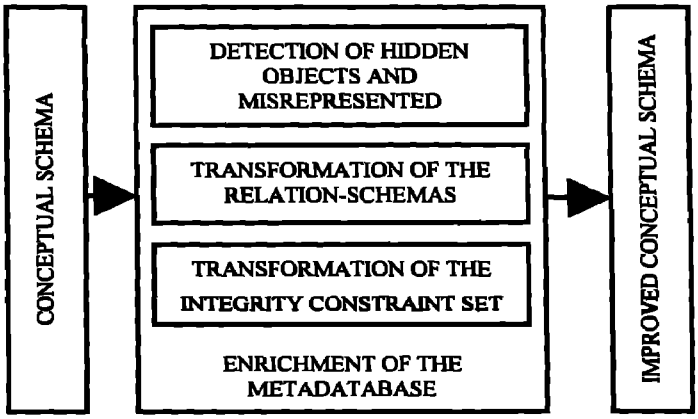


**Figure 1. Improvement of the Metadatabase.**

## 2. REFERENTIAL INTEGRITY

Referential integrity is one of the most important concepts in databases. The study of the left and right terms of referential constraints helps to comprehend all issues of this subject. There are five possible placements of a set of attributes in relation with the key in a table. Being Z such set of attributes, sch(R) the set of all attributes of R, and K the primary key of R, the five placements are (Figure 2): I) $Z \equiv K$; II) $Z \cap K = \emptyset$, i.e. $Z \equiv X$, being X a subset of non-key attributes; III) $Z \subset K$ i.e. $Z \equiv X'$ being X' a proper subset of K; IV) $Z \supset K$ i.e. $Z \equiv K \cup X$; and finally V) $Z \cap K \neq \emptyset$, $Z - K \neq \emptyset$ and $K - Z \neq \emptyset$., i.e. $Z \equiv X \cup X'$. Then it follows that there are 25 possible pairs $\langle R_i[Z_i], R_r[Z_r] \rangle$ (Table 1). Those cases numbered 1 to 5 in Table 1 correspond to referential integrity restrictions. Referential constraints of type IV and V – i.e. with the left term of type IV or V– are not typical and they do not appear in a normalized schema but in an ad-hoc refinement of it. All these cases have been developed in [7], [14], [15]. The remaining cases correspond to *ids*. The study of these latter cases is the subject of this paper, though it is worth mentioning that cases 16 to 25 are less frequent than the rest[1]. In [11] there is a complete analysis of the update anomalies and the trouble of integrity maintenance in the presence of denormalized tables of those types in the right hand side of *ids*. Notice that for the study of referential integrity restrictions the left term must be analyzed whereas the study of inclusion dependencies must be focused on the right term.

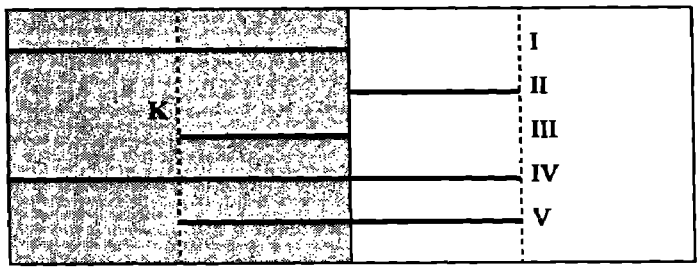


**Figure 2: Placements of a set of attributes in relation with the key**

[1] This assertion is founded on a study of several real cases, pertaining to different organizations and enterprises.

**Table 1. Different types of general inclusion dependencies.**

| Left Term \ Right Term | I) Key (K) | II) Non Key (X) | III) Part of a Key (X') | IV) Key + Non Key (K ∪ X) | V) Part of a Key + Non Key (X ∪ X') |
|---|---|---|---|---|---|
| I) Key (K) | 1. K << K | 6. K ⊆ X | 11. K ⊆ X' | 16. K ⊆ K ∪ X | 21. K ⊆ X ∪ X' |
| II) Non Key (X) | 2. X << K | 7. X ⊆ X | 12. X ⊆ X' | 17. X ⊆ K ∪ X | 22. X ⊆ X ∪ X' |
| III) Part of a Key (X') | 3. X' << K | 8. X' ⊆ X | 13. X' ⊆ X' | 18. X' ⊆ K ∪ X | 23. X' ⊆ X ∪ X' |
| IV) Key + Non Key (K ∪ X) | 4. K ∪ X << K | 9. K ∪ X ⊆ X | 14. K ∪ X ⊆ X' | 19. K ∪ X ⊆ K ∪ X | 14. K ∪ X ⊆ X ∪ X' |
| V) Part of a Key + Non Key (X ∪ X') | 5. X ∪ X' << K | 10. X ∪ X' ⊆ X | 15. X ∪ X' ⊆ X' | 20. X ∪ X' ⊆ K ∪ X | 25. X ∪ X' ⊆ X ∪ X' |

## 2.1 Right Relation Structure

As the analysis of *ids* must be focused on the right relation, in this section all possible types are considered. Types I, II and III are completely developed and exemplified. The remaining cases refer to the previous ones. Since the right term can be seen as the virtual join of two tables through a *rir* type 't', from now on it will be named "right term of type 't' ". $R_l$ and $R_r$ are respectively the left and the right term of the referential constraint. $Z_l$ is the set of referencing attributes in $R_l$; $K_r$ is $R_r$ primary key. $R_r$ comes from $R_{r1}$ and $R_{r2}$ which in turn stand for the virtual components of $R_r$; $K_{r1}$ and $K_{r2}$ are the keys of $R_{r1}$ and $R_{r2}$ respectively; $X'_{r1}$ is a proper subset of $K_{r1}$; $X_{r1}$ is a set of secondary attributes in $R_{r1}$. Referential actions are not relevant for the following analysis.

Cases 6. to 10. can be later on grouped into two subcases: a) the non-key attributes are an alternative key and b) the non-key attributes are not an alternative key. Subcase a) has been completely developed in [10] and it will be not treated in this work.

i) *Right term of Type I* (cases 1 to 5 in Table 1; *rir*s are syntactically included in this case):

*Id*s may look like *rir*s but perhaps they are not. Consider the relations (primary keys are underlined) EMPLOYEE (Employee-id, Employee-name, Technical-Degree, Area-of-Expertise) and SUPERVISOR (Employee-id, Responsibility); and the *id* SUPERVISOR [Employee-id] ⊆ EMPLOYEE [Employee-id].

Suppose Technical-Degree and Area-of-Expertise are inapplicable attributes, since they are 'not null' only when an employee is a Technical one. Intuitively, this case may be redesigned as EMPLOYEE (Employee-id, Employee-name); TECHNICAL-EMPLOYEE (Employee-id, Technical-Degree, Area-of-Expertise) and SUPERVISOR (Employee-id, Responsibility).

Consider the diagram in Figure 3. If $R_r$ is a view of $R_{r1}$ and $R_{r2}$, the relationship between $R_l$ and $R_r$ through Employee-id looks like a *rir*, but it could mask a new constraint. $R_r$ holds two different objects, avoiding any specific reference to one of them. The designer has no choice when referencing either a single employee or a technical one as the structure only allows references to the unified table.

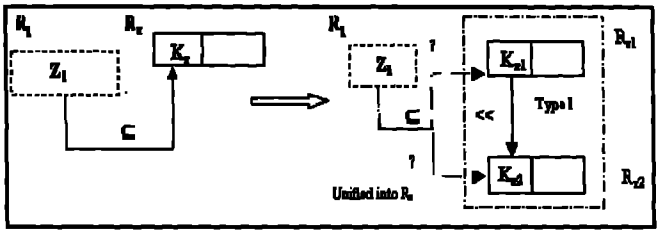


**Figure 3. Right term of type I**

Consequently, the restriction SUPERVISOR [Employee-id] ⊆ EMPLOYEE [Employee-id] is seen as a *rir*, but when "only technical employees may become supervisors", that restriction is not a *rir*. The only way the designer could preserve the semantics of the data would be by introducing an artificial business rule. Thus, if any employee may become a supervisor, the following constraints must be specified: TECHNICAL-EMPLOYEE[Employee-id]<<EMPLOYEE[Employee-id] and SUPERVISOR[Employee-id]<<EMPLOYEE[Employee-id].

However, if the reference must be only based on technical employees, the correct referential constraints are TECHNICAL-EMPLOYEE [Employee-id] << EMPLOYEE [Employee-id] and

SUPERVISOR [Employee-id] << TECHNICAL-EMPLOYEE [Employee-id].

Since embedded objects come from the referencing table in the unified $R_r$, case I is slightly different from the following four cases where embedding objects play this role.

ii) *Right term of Type II* (cases 6 to 10 in Table 1).

Consider the relations (primary keys are underlined) PROJECT (Project#, Project-name, Department-name, Start-date) and BUDGET (Proceeding#, Department-name, Amount); and the *id* BUDGET [Department-name] ⊆ PROJECT [Department-name]. In this design, the omitted class of entities is DEPARTMENT. This case is depicted in Figure 4.

In this case, - which design did the designer intend? If (s)he wanted to reference DEPARTMENT, the hidden object should be

directly modeled. The resulting redesigned schema is: PROJECT (Project#, Project-name, Department-name, Start-date); BUDGET (Proceeding#, Department-name, Amount) and DEPARTMENT (Department-name).

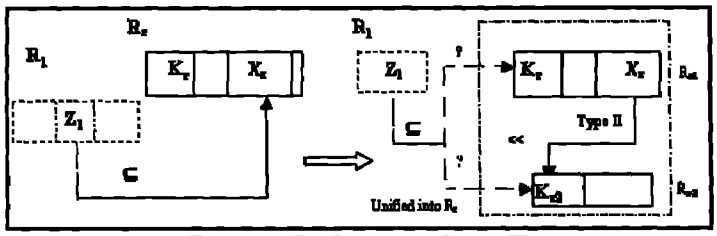


**Figure 4. Right term of type II**

Moreover, replacing the original *id* with the following *rirs* modifies the set of restrictions:

BUDGET [Department-name] << DEPARTMENT[Department-name] and PROJECT [Department-name] << DEPARTMENT [Department-name].

On the contrary, if the designer wanted to establish that "only those departments having projects" have a budget, the transformation of the set of relations is the same but it is necessary to represent the restriction in quotation marks as the following *rid*: DEPARTMENT [Department-name] ⊆ PROJECT [Department-name].

This is actually a *hidden business rule* (a domain constraint). From the last two restrictions, it comes out that PROJECT.Department-name and DEPARTMENT.Department-name always have the same values.

iii) *The right term comes from a unification of type III* (cases 11 to 15 in Table 1).

Consider the relations (keys are underlined) DELEGATE-STUDENT (Student#) and ATTENDANCE (Course#, Student#, Mark); and the *id* DELEGATE-STUDENT [Student#] ⊆ ATTENDANCE [Student#].

This *id* points that there is a missing entity of the real world: STUDENT. The diagram in Figure 5 shows this problem.

Again, two original designs may be possible in relation with the designer intention. If the requirement was just to establish the semantic link among DELEGATE-STUDENT, ATTENDANCE and STUDENT, then the references should have been the following: DELEGATE-STUDENT [Student#] << STUDENT [Student#] and ATTENDANCE [Student#] << STUDENT [Student#].

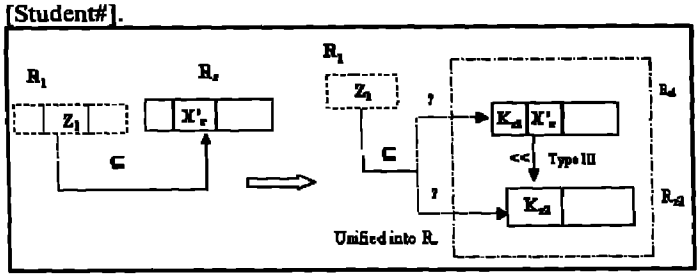


**Figure 5. Right term of type III**

On the other hand, if the designer actually wanted to specify a constraint such as "only those students with a Mark in at least a course may be delegates", the following reference should be added: STUDENT [Student#] ⊆ ATTENDANCE [Student#].

Again, the expression in quotation marks stands for a *hidden business rule*.

iv) *Right term of types IV or V* (cases 16 to 20, and 21 to 25 in Table 1, respectively)

Once more, it is impossible to know which relation the designer wanted to reference. If the designer's intention was to reference $R_{r2}$, the proper set of relations and restrictions should be reformulated in a similar way as in the previous case. On the other hand, if the restriction "$R_i$ references just only those instances in $Z_r$ ($Z_r \equiv K_r \cup X'_r$ and $Z_r \equiv X'_r \cup X_r$, respectively)"[2] has to be maintained, it must be specified via a *rid*, again as in the previous cases.

Notice that through the analysis of the possible origin of Rr, the normalized design can be figured out. However, it is insufficient because the designer's intention must be guessed. In other words, syntactic aspects guide the conversion but, previously, the designer must elucidate the semantic aspects.

# 3. MEANING AND ORIGIN OF THE EMBEDDED OBJECTS

The purpose of the enrichment process is to obtain improved descriptions of the relevant objects, through the incorporation of the semantics supplied by their relationships with other objects. As the entities were not properly designed in the schema, the relationships among them and other objects were also misrepresented. Hence, reflecting on all possible origins of the hidden objects allows for the precise definition of the nature of the connections among them. With basis on the study of several real cases, the following different origins can be identified:

1. *Intentionally embedded object*: it appears when $R_r$ is a virtual view of tables through *rirs* of type I to V. The presence of a referential constraint of type IV always produces *ids* [7], [15].
2. *Dropped object:* in this case, the subset of referenced attributes also includes an identifier and descriptive attributes which stand for an embedded object. However, its origin is not an intentional virtual join but a poor design.
3. *Intentionally dropped object*: this kind is similar to the previous one but only its identifiers represent the embedded object.
4. *Duplicated data:* this is a special situation of Case 1. The relation $R_r$ is a view of virtual components obtained via projections, selections and joins through a referential integrity restriction of one of the types I to V [14], [15].

Cases 2, 3 and 4 are syntactically included in Case 1. However, they have been highlighted as they hold semantic differences.

An obvious question immediately arises at this point: if the problem is related to denormalized schemes, - why do not just normalize them? In different contexts, this question has different answers depending on how well the schema is currently understood. In a fully documented and well-designed database schema the best solution is to normalize it, reaching the higher level of normalization allowed by performance and storage

[2] Again, Xr is a subset of sch(Rr)-Kr and X'r is a proper subset of Kr

considerations[3]. However, when the user cannot afford the response time required by queries including join operations, (s)he has the alternative to denormalize the schema.

In a poorly understood database schema, the heuristics described in Section 4 eases the discovery of implicit knowledge. Notice that the complete determination of the semantic aspects is required in order to reach decisions related to the schema reengineering process.

# 4. IMPROVEMENT OF THE METADATABASE

This section presents a heuristics to convert the real schema into one with a higher semantic level. Let the original relational schema be $\mathbf{R}=<\mathbf{R},\mathbf{D}>$, being $\mathbf{R}=\{R_1, R_2, ..., R_m\}$ and $\mathbf{D}=\{FD, ID, NR\}$ the set of relations and the set of constraints respectively. FD, ID and NR are the sets of functional dependencies, inclusion dependencies and null restrictions respectively. A functional dependency held in $R_i$ is expressed as: $R_i:X \rightarrow Y$. If X is the key of $R_i$, the functional dependency is a primary one. Otherwise, it is a secondary one. The schema of a relation $R_i$ is the set of its attributes and it is expressed as $sch(R_i)$. In what follows, consider the references detailed in Table 2.

Particularly ID = { ...; $R_{ij}[Z_{ij}] \subseteq R_r[Z_r]$; ...} j =1, 2, .... and FD = {..., $R_i:K_i \rightarrow sch(R_i)-K_i$; $R_r:K_r \rightarrow sch(R_r)-K_r$; $R_r: Z_r \rightarrow X_2$; ...}[4] In cases II to V, the last one is a secondary functional dependency. In case I this dependency reveals that $X_2$ is a set of inapplicable attributes. Let the enhanced schema be defined as $R^e=<R^e,D^e>$ with its components similarly expressed as before. Once the missing entities have been detected and their classes identified, the following steps permit the schema reengineering:

1. For each hidden or omitted entity detected, create a new relation (NEW). Define its identifier as the right term of the *id* ($K_{NEW} \equiv Z_r$). Consider all descriptive attributes for the hidden object (if there is any) as descriptive attributes in NEW, and drop them from $R_r$. Thus sch(NEW) = $Z_r$, $X_2$. From now on $R'_r$ denotes $R_r$ without $X_2$ attributes.
2. Relocate all *ids* whose left term is the set of attributes identifying the missing object in NEW (i.e. $R_{ij}[Z_{ij}] \subseteq R_r[Z_r] \rightarrow\rightarrow R_{ij}[Z_{ij}] << NEW[K_{NEW}]$, j =1, 2, ... ). In case I, only when this constraint is a *hidden business rule*; otherwise: $R_{ij}[Z_{ij}] << R'_r[K_r]$.
3. Include a new *rir*, $R'_r[Z_r] << NEW[K_{NEW}]$ in $ID^e$. In case I, only if NEW and $R'_r$ relationship is *1:1 mandatory*
4. Add a *rid* $NEW[K_{NEW}] \subseteq R'_r[Z_r]$ to $ID^e$ only when this constraint is a *hidden business rule*. Always in case I.

The application of these steps results in: $R^e = R \cup \{NEW; R'_r\} - \{R_r\}$. For cases II-V, $ID^e$ = ID $\cup\{R'_r[Z_r]<<NEW[K_{NEW}]\} \cup\{R_{ij}[Z_{ij}] << NEW[K_{NEW}]\}j=1,2,.... - \{R_i[Z_{ij}] \subseteq R_r[Z_r]\}j=1,2,....$ For the case pointed in 4. $ID^e$ = ID $\cup\{R'_r[Z_r]<<NEW[K_{NEW}]$; $NEW[K_{NEW}] \subseteq R'_r[Z_r]\} \cup\{R_{ij}[Z_{ij}] << NEW[K_{NEW}]\}j=1,2,.... - \{R_{ij}[Z_{ij}] \subseteq R_r[Z_r]\}j=1,2,....$ $FD^e$ = FD $\cup$ $\{NEW:K_{NEW} \rightarrow X_2\} - \{R_r:K_r \rightarrow X_2, R_r:Z_r \rightarrow X_2\}$.

For the case I, $ID^e$ and $FD^e$ can be obtained in a similar way. Figures 6 and 7 show this heuristics schematically.

In order to make the examples informally developed in section 2.1 clearer, the second one is reconsidered. The relations are PROJECT (Project#, Project-name, Department-name, Start-year) and BUDGET (Proceeding#, Department-name, Amount). The referential restriction is BUDGET [Department-name] $\subseteq$ PROJECT [Department-name]. Then $Z_r \equiv X$ = PROJECT.Department-name; $X_1$= PROJECT.Project-name, PROJECT. Start-year and $X_2 = \emptyset$.

**Table 2: $sch(R_r)$ components. $Z_r$ is the referenced attribute (simple or composite). X' and X" are proper disjoint subsets of $K_r$. X, $X_1$ and $X_2$ are disjoint subsets of $sch(R_r) - K_r$. $X_1$ and $X_2$ may be empty.**

| TYPE | Sch($R_r$) | | $Z_r$ |
|---|---|---|---|
| | NON-KEY TTRIBUTES | $K_r$ | |
| I | $X_1 \cup X_2$ | $K_r$ | $K_r$ |
| II | $X \cup X_1 \cup X_2$ | $K_r$ | X |
| III | $X_1 \cup X_2$ | X' $\cup$ X" | X' |
| IV | $X \cup X_1 \cup X_2$ | $K_r$ | $K_r \cup X$ |
| V | $X \cup X_1 \cup X_2$ | X' $\cup$ X" | X' $\cup$ X |

Step 1) These *id* indicate DEPARTMENT as the real world missing entity (NEW), then $R^e$ = {BUDGET, PROJECT, DEPARTMENT}.

Step 2) The *id* must be reformulated as the following *rir*:

BUDGET[Department-name]<< DEPARTMENT [Department-name]

Step 3) The following *rir* must be added to $ID^e$:

PROJECT[Department-name]<<DEPARTMENT [Department-name]

Step 4) At this point, two scenarios may be possible. If the requirement is just to establish the semantic link among BUDGET, DEPARTMENT, and PROJECT, then the reengineering process is done. In contrast, if the designer actually wants to specify a constraint such as "only those departments with at least a project may have a budget", the next *rid* must be included into $ID^e$: DEPARTMENT [Department-name] $\subseteq$ PROJECT[Department-name] (dotted arrow).

## 4.1 Materialization of *rids*

The expression in quotation marks in the previous section is just a particular case of a business rule: a *domain constraint*. This restriction controls the values of one attribute in a relation against the elements of a set defined by intension. Update operations may change this set. A truly conceptual implementation should calculate the permitted values before using them, although performance reasons naturally suggest that a better solution should be to build this set incrementally. It is a pragmatically wise approach if the designer does not consider this set as a real table.

[3] Even more, it replicates the values of the attributes involved in the join, thus causing a serious integrity danger. If an update is performed, all copies must be updated, so the cost of this manipulation may grow significantly

[4] Without loss of generality, schemas with only one denormalized $R_r$ are considered.

If the designer persists in keeping such set as a table, different implementations for this subject can be imagined. Obviously, *ids* should be disregarded due to the difficulties inherent to their maintenance. General CHECK constraints or simply programming code driving the referential action needed to maintain that set in a consistent state, are some of the ways to obtain the desired behavior.

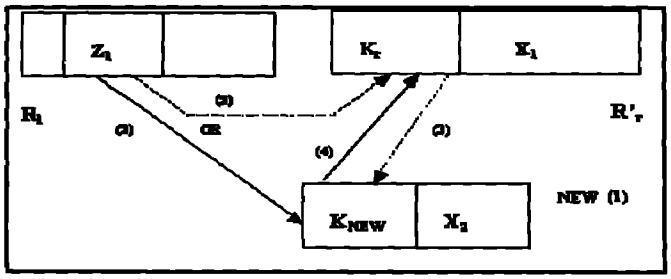


**Figure 6: Case I, conversion**

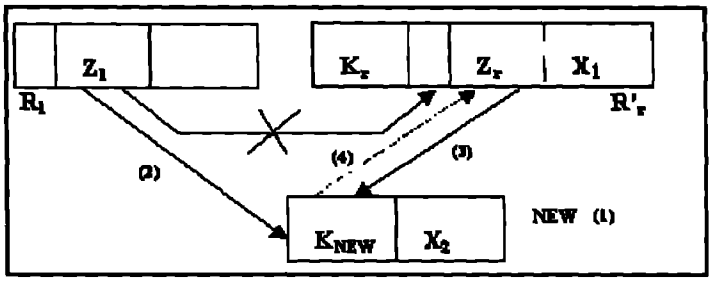


**Figure 7: Cases II to V, conversion**

The first case is the simplest one. As for the previous example, the constraint can be specified as:

CHECK (NOT EXISTS (SELECT Department-name FROM DEPARTMENT WHERE Department-name NOT IN (SELECT Department-name FROM PROJECT))).

Generally, for cases II to V:

CHECK (NOT EXISTS (SELECT $K_{NEW}$ FROM NEW WHERE NEW.$K_{NEW}$ NOT IN (SELECT $R_r.Z_r$ FROM $R_r$))).

The second option requires the programming of the following referential actions:

Insertions: an insertion over $R_r$ provokes an insertion in the table NEW (insertions with Cascade as the referential action) if and only if *the value of the concerning attribute (or a set of them) is a new one*. Insertions over the table NEW are prohibited in other cases.

Deletions: *when the deleted tuple of $R_r$ contains a last instance of the referred attribute* it must provoke the deletion of the tuple of table NEW containing the same value. Deletions over NEW are prohibited in other cases.

Updates: as with deletions, this operation has a special semantics because it must be understood in one of two ways: i) it represents the update of a particular $R_r$ tuple or ii) it represents the update of the embedded object.

Notice that, in this scenario, table NEW can never be modified by direct operations. In case I, this materialization is straightforward because all restrictions can be specified as *rirs*, even in the situation explained in step 3 although the mutual reference must be carefully treated.

When the declarative issues are not supported by the system or the intended restrictions cannot be expressed declaratively, triggers are very useful in supporting data integrity in a database. Triggers for referential integrity are an alternative to the use of foreign key constraints in commercial SQL92-relational products. Even so, foreign key constraints or check constraints are normally preferable to explicit triggers because they are declarative and then easier to manage [6].

# 5. RELATED WORK

The development of methodologies and heuristics to address the improvement of the expressiveness in relational conceptual schemas has been the subject of several research projects. Chapter 4 in [5] can be seen as a foundational analysis of referential integrity focusing only on key-based inclusion dependencies in a relational environment. On the other hand, [2] is a seminar paper on the theory of key-based inclusion dependencies. It describes a two-step optimization strategy for relational schemes, taking into account the referential actions for insertions and deletions.

Methodologies such as those introduced in [12], and [10], assume that the original conceptual schemas are normalized, i.e. at least in 3NF (BCNF in [12]). In [12] a formal method to capture the structural semantics of information systems is proposed. It can be used to analyze the semantics of existing relational databases and to convert conventional relational schemes into object-oriented database schemas. The method considers functional dependencies and key-based inclusion dependencies. Castellanos [3] presents a more general approach since not only 3NF cases are considered but also certain kinds of denormalized ones. It considers inclusion dependencies under a general viewpoint. They are analyzed according to a set of 25 cases based on the composition of their left and right hand sides. This analysis is the basis for a semi-automatic reengineering process that recognizes hidden structures in an interoperability environment. In this work, the problem is studied from a broader point of view including all Castellanos' cases (1, 2, 3, 11, 12, 13, 6, 7, 8 case numbers). Johanneson [10] developed a method that translates a schema in a traditional relational data model into a conceptual schema. His classification of the right term does not consider conformations such as "part-of-a-key+secondary-attributes", "key+secondary-attributes" and "part-of-a-key". [8] presents the analysis and comparison of several techniques for the elicitation of hidden structures and in [9] the authors treat the reconstruction of the DBMS-dependant (logical) schema, focusing on the reasonings and processes through which hidden structures and constraints can be elicited. In [14] and [15] the treatment of key-based inclusion dependencies is analyzed, extending the results of [12], [13] since two pathological cases are characterized and included in the analysis.

Finally, in [11] there is a complete analysis of the redundancy problem and the preservation of integrity in presence of *ids* interacting with functional dependencies.

# 6. CONCLUSIONS AND FURTHER WORK

This work extends previous results analyzing referential restrictions whose right sides are denormalised tables, i.e. particular views. The possible origins of embedded objects in the right side of these restrictions have been highlighted and described. Next, a heuristics for the complete reengineering of conventional relational schemas holding non-key inclusion

dependencies is detailed. Whenever an irreducible remaining *id* is detected, the contextual conditions for its maintenance are established, leading to the definition of a specific business rule. These special cases of masked business rules have been completely analyzed and a new conceptual perspective of its need was given.

Since the right terms are seen as denormalized tables, the first step of the proposed heuristics promotes the normalization by transferring the descriptive attributes to table NEW.

The concepts developed in this chapter are useful and portable to SQL3-based systems and object-oriented databases, since the poor quality problem in conceptual schemas may also be present in those systems too. These ideas are also relevant for other areas such as federated databases, database migration, and reengineering of legacy systems.

Currently, this work is being extended with the inference of the referential actions of the initial *id*s into the *rir*s and *rid*s. The design and implementation of an interactive computerized tool to facilitate the designer to obtain a better quality schema, with basis on the heuristics, will be faced in the near future.

## 7. REFERENCES


[1] Abiteboul, S., Hull, R., Vianu, V.: Foundations on Databases. Addison Wesley Publ. Co. (1995)

[2] Casanova, M., Tucherman, L., Furtado, A., Braga, A.: Optimization of Relational Schemes Containing Inclusion Dependencies. Proceedings of 15 VLDB Conference. Amsterdam. (1989) 317-325

[3] Castellanos, M.G.: A Methodology for Semantically Enriching Interoperable Databases. Proceedings of 11 th. British National Conference on Databases, Keele (1993)

[4] Codd, E.: The Relational Model for Database Management. Version 2. Addison Wesley Publ.Co (1990)

[5] Date, C.: Updating Views. Relational Databases, Selected Writings. Addison Wesley. Reprinted with corrections (1989)

[6] Date, C.: An Introduction to Database Systems. Addison Wesley (2000)

[7] Doorn, J., Rivero, L.: Normalization of Non-BCNF Relations Integrity Constraints. Proceedings XII International Conference of Systems Engineering, ICSE-97, Coventry, UK. (1997) 217-222

[8] Henrard, J., Hainaut, J.-L..: Data Dependency Elicitation in Database Reverse Engineering. Published CSMR 2001. Lisbon. Portugal. March 2001.

[9] Henrard, J., Hainaut, J.-L.; Hick, J.-M.; Roland, D.; Englebert, V.: Data Structure Extraction in Database Reverse Engineering. Proceedings of REIS'99 Workshop. Paris, November 1999. Research paper RP-99-007. Institut d'Informatique. University of Namur.

[10] Johanneson, P.: A Method for Transforming Relational Schemas into Conceptual Schemas. IEEE Trans. on Software Eng. (1994) 190-201

[11] Levene, M., Vincent, M.: Justification for Inclusion Dependency Normal Form. IEEE TKDE Vol 12 (2). March/April (2000) 281-291

[12] Markowitz, V., Makowsky, J.: Identifying extended entity-relationship object structures in relational schemas. IEEE Trans. on Software Eng. 16, 8, (1990) 777-790

[13] Markowitz, V.: Referential Integrity Revisited: an Object-oriented Perspective. Proceedings of the 16th VLDB Conference. Brisbane. Australia (1990)

[14] Rivero, L., Doorn, J.: Integridad Referencial y Actualizaciones en Relaciones Desnormalizadas. In Proceedings of XIV Conferencia Latinoamericana de Informática - CLEI Panel 98. Quito, Ecuador (1998) 911-921

[15] Rivero, L., Doorn, J.: Managing Referential Integrity and Non Key-based Dependencies in a Denormalized Context. Proceedings of 2000 IRMA International Conference. Anchorage, Alaska (2000) 883-886