Static Detection of Sources of Dynamic Anomalies in a Network of Referential Integrity Restrictions

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

Jorge H. Doorn
ISISTAN - UNCentro,
B.A. Argentina
Phone: +54 2293 440363
jdoorn@exa.unicen.edu.ar

Daniel Loureiro
UNCentro,
B.A. Argentina
Phone: +54 2293 440363

ABSTRACT

Under certain circumstances, basic operations over tables in a relational database, where integrity restrictions such as referential and null restrictions have been specified, may produce unpredictable results, not detectable by means of a static analysis of the schema. When the design includes redundancies or when the set of restrictions is contradictory it is easy to detect and prevent future errors, but there are situations that require a dynamic analysis. In this paper, the properties of networks of referencial integrity restrictions that contain irregularities are analyzed, and the anomalies that may appear when data actualization in such environment is done are studied in order to define criteria and develop an algorithm to generate rules for proper handling of inconsistencies.

Keywords: integrity restrictions, referential integrity, database updates, anomalous updates.

1. INTRODUCTION

Semantic data control ensures the maintenance of database consistency, by rejecting update transactions that lead to inconsistent states or by activating specific actions on the database state to compensate the effect of the previous transaction. Unpredictable results may be obtained in a relational particular database state, when basic operations are applied. This inconvenience is produced by redundancies in the schema design, by contradictory referential integrity restrictions or simply because the designer established complex restrictions having tuple dependent semantic. From a procedural point of view, it means that the result may be unpredictable because it is affected by the order in which individual restrictions are applied or by the order in which the constraints are enforced.

SAC'00 March 19-21 Como, Italy
(c) 2000 ACM 1-58113-239-5/00/003>...>$5.00

Those problems are well-known and there are research reports in relation with some aspects of them (see e.g. [Markowitz90], [Markowitz91], [Casanova88], [Rivero96]) but the formalization of conditions and the implementation of mechanisms for the automatic prevention of those anomalies is a recent investigation field. In [Markowitz94] the uncertainties produced by specific data manipulation are described, especially in delete operations. In [Casanova89] special cases of updates propagation are described.

This research paper may be divided in three well-defined parts. The first is developed in sections 3 and 4 and is devoted to refine Markowitz study extending it to all operations. The second is dedicated to the specification of algorithms able to detect potential sources of anomalies in a static way (Section 5) [Rivero98]. The last part (Section 6.) presents an algorithm that automatically generates rules that allows the integrity verification.

2. RELATIONAL CONCEPTS

A relational schema is $\boldsymbol{R} \equiv <\mathbf{R}, \mathbf{D}>$, with $\mathbf{R}=\{\mathbf{R}_1, \mathbf{R}_2, ..., \mathbf{R}_m\}$ and $\mathbf{D}=\{\mathbf{FD}, \mathbf{ID}, \mathbf{NR}\}$ set of relations, **FD** and **ID** set of functional dependencies (***fd***) and inclusion dependencies (***id***) respectively and **NR** set of null restrictions (null constraint and null not allowed). A database state for **R** is denoted by $\boldsymbol{r} = \{r_1, r_2, ..., r_m\}$; $\mathbf{sch(R_i)}$ represents the set of attributes of R_i, K_i stands for a candidate key over Ri; and FK represents a foreign key for R_i. A database state $\boldsymbol{r}$ associated with **R** is consistent if it satisfies all restrictions in **D**. Two attribute sets overlap <u>**iff**</u> they share attributes $(Y \cap Z \neq \emptyset)$, and strictly overlap <u>**iff**</u> they overlap but they are not equal $((Y \cap Z \neq \emptyset) \wedge (Y \neq Z))$.

A functional dependency (***fd***) over a set of attributes U is one expression of the form $\mathbf{X \rightarrow Y}$, where $X, Y \subseteq U$. If $R_i \varepsilon \boldsymbol{R}$, ***fd***'s over R_i will be indicated by $\mathbf{R_i{:}X \rightarrow Y}$.

A null constraint (***nna***) may be expressed by $\mathbf{R_i{:}L_i} \neq \lambda$. It is satisfied by R_i <u>**iff**</u> for every tuple t of R_i the subtuple t.Li has only not null values. There is at least one ***nna*** in R_i, that is R_i: $K_i \neq \lambda$.

One inclusion dependency (***id***) in $\boldsymbol{R}$ is an expression $\mathbf{R_i[X] << R_j[Z]{:}(\alpha,\beta,\mu_i,\mu_d)}$, where R_i and R_j are relation names (possibly the same); $X,Z \subseteq sch(R_j)$ are compatible attributes; α, β, μ_i and μ_d are the strategies to perform inserts, deletes and updates over the left and right side respectively, and the strategies may be c (*Cascades*), r (*Restricted*) or n (*Nullifies*). All combinations are studied in this article, however inserts and updates over the left side are generally done using α y μ_i specified with modality 'r'. If Z is the primary key of the relation R_j, then it is a ***key-based-id*** and X constitutes a foreign key for R_i; this sort of ***id***'s are named referencial integrity restrictions (***rir***'s).

The referential integrity directed graph **G=(V,H)**, associated with **R** may be defined with **V=R** and **H** = $\{(R_i,R_j,L{:}(\alpha,\beta,\mu_i,\mu_d))\ /R_i[L]<<R_j[K_j]{:}\{\alpha,\beta,\mu_i,\mu_d] \in ID\}$. **H** is composed by elements $(R_i,R_j,\ L{:}\ (\alpha,\ \beta,\ \mu_i,\ \mu_d))$, where the edge goes from R_i to R_j and $L{:}(\alpha,\beta,\mu_i,\mu_d)$ is the label of the edge.

3. CONFLICTIVE MANIPULATIONS

In order to characterize the problem that may arise when some updates are performed over data constrained by ***rir***'s the examples in Figures 1 and 2 will be used.

Example 1:

- **Relations** (keys are underlined)

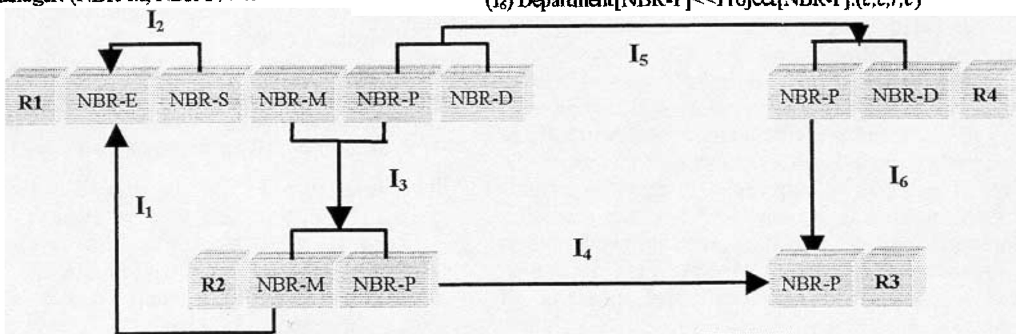
(R_1) Employee (<u>NBR-E</u>, NBR-S, NBR-M, NBR-D, NBR-P)
(R_2) Manager (<u>NBR-M</u>, NBR-P)
(R_3) Project (<u>NBR-P</u>)
(R_4) Department (<u>NBR-D, NBR-P</u>)

- **Null restrictions**

(N_1) Employee: NBR-E $\neq \lambda$
(N_2) Manager: (NBR-M, NBR-P) $\neq \lambda$
(N_3) Project: NBR-P $\neq \lambda$
(N_4) Department: (NBP-D, NBR-P) $\neq \lambda$

- **Referential Integrity Restrictions**

(I_1) Manager[NBR-M]<<Employee[NBR-E]:(*c,c,c,c*)
(I_2) Employee[NBR-S]<<Employee[NBR-E]:(*n,r,n,r*)
(I_3) Employee[NBR-M, NBR-P]<<Manager[NBR-M, NBR-P]:(*c,c,c,n*)
(I_4) Manager[NBR-P]<<Project[NBR-P]:(*r,r,c,c*)
(I_5) Employee[NBR-P, NBR-D]<<Department[NBR-P, NBR-D]:(*c,c,c,c*)
(I_6) Department[NBR-P]<<Project[NBR-P]:(*c,c,r,c*)



R1: r_1	R2: r_2	R3: r_3	R4: r_4
1 4 4 a x	2 b	a	a x
2 λ λ b λ	4 a	b	b y
3 2 2 b y		c	c y
4 λ λ a x			

Figure 1: Restriction Graph, State and Restrictions for Example 1 (adapted from [Markowitz94])

Example 2:

- **Relations** (keys are underlined)

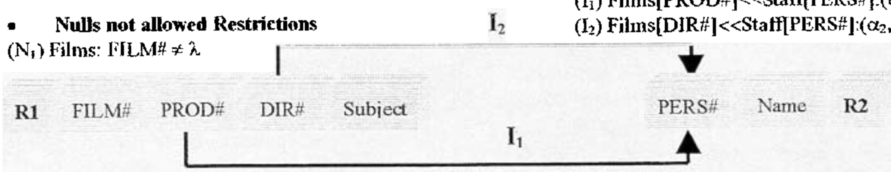
(R_1) Films (<u>FILM#</u>, PROD#, DIR#, Subject)
(R_2) Staff (<u>PERS#</u>, NAME)

- **Nulls not allowed Restrictions**

(N_1) Films: FILM# $\neq \lambda$
(N_2) Staff: PERS# $\neq \lambda$

- **Referential Integrity Restrictions**

(I_1) Films[PROD#]<<Staff[PERS#]:($\alpha_1,\beta_1,\mu_{i1},\mu_{d1}$)
(I_2) Films[DIR#]<<Staff[PERS#]:($\alpha_2,\beta_2,\mu_{i2},\mu_{d2}$)



R1: r_1	R2: r_2	R1: r_1'
1 1 4 x	1 xx	1 1 4 x
2 1 2 q	2 yy	2 1 2 q
3 2 3 s	3 zz	3 2 2 s
4 4 5 v	4 tt	4 4 3 v
	5 pp	

Figure 2: Restrictions Graph, State and Restrictions for Example 2.

The data manipulations considered are insertions, deletions or updates of one or several tuples; a manipulation involves only one kind of operation, it refers to an unique relation and entails a set of tuples that do not change during the execution of the manipulation. The specified constraints will be verified after each single tuple operation. The manipulation will be successful if all of its single tuple operations are carried out, otherwise it fails (is revoked). The above two examples present more than just one anomalous behavior. In the following sections, simplified subschemes of them are used to illustrate different problems.

3.1. Problems in Insert Operations

The effect of insertions is examined in this section.

Example 1-1: Consider the ***rir***'s and the database state depicted in Figure 1, excluding in this case I_1 and I_2. The operation I inserts the tuple (5 4 4 *d x*) in the relation r_1. I may cause the insertion (via I_5 and I_6) of the tuple (*d*) in the relation r_3, or it may block the insert operation (via I_3 and I_4). The result of I depends on the order in which the ***rir***'s that involve R_1 are enforced: (i) if I_5 is first considered, the tuple (*d x*) is inserted in r_4, this triggers the enforcing of I_6, which provokes the insertion of the tuple (*d*) in r_3, then I_3 will cause the insertion of the tuple (*4 d*) in r_4; or (ii) if I_3 is taken into account in the first place, the tuple (4 *d*) is inserted in r_2, and this triggers the enforcing of I_4 which blocks the insert operation since the tuple (*d*) does not exist in the relation r_3.

Example 2-1: Consider the example in Figure 2 and the corresponding state of the database $\boldsymbol{r} = \{r_1, r_2\}$. If α_1: r and α_2: *c* and the operation I inserts the tuple (5 6 6 *v*) in the relation r_1, it may trigger the insertion (via I_2) of the tuple (6 λ) in r_2, or the insert operation may be blocked (via I_1). The result of I depends on the order in which the ***rir***'s that involve a R_1 are applied: (i) if I_2 is enforced in first place then the tuple (5 6 6 *v*) is inserted in r_1 and the tuple (6 λ) in r_2; or (ii) if I_1 is first taken into account, I is blocked since there is not a value 6 for PERSON-ID in R_2. If α_1: *n* and α_2: *c* and it is required that PROD# and DIR# foreign keys associated with R_1 can not hold null values, the set null modality for the insert operation becomes restricted, showing a behavior similar to the previous.

3.2. Problems in Delete Operations

A deletion D may trigger the typical actions defined by the strategy, triggering other operations or, by the contrary, blocking the manipulation. The outcome of D, may be unpredictable when the enforcement of the ***rir***'s promoted by D implies the trigger of updates or deletions of tuples, that in turn can blockade D if another path of **G** is first considered.

Example 1-2: Consider the Example 1, excluding in this case I_1 and I_2. Suppose that D involves the tuple (*a*) of relation r_3. The tuple (4 *a*) of r_2 can block D via I_4, while D can trigger the deletion of that tuple via I_6, I_5, and I_1. The result of D depends on the order of the enforcement of the ***rir***'s that involve R_3: (i) if I_6 is verified in the first time, then (*a x*) is deleted from r_4, then I_5 is enforced, triggering the deletion of tuples (1 4 *x a*) and (4 λ *x a*) of r_1; finally I_1 promotes the suppression of tuple (4 *a*) of r_2; on the other side (ii) if I_3 is first enforced, D is blocked by tuple (4 *a*) of r_2.

Example 2-2: Consider Example 2; with β_1: *r*, β_2: *c*; and $\boldsymbol{r}=\{r_1', r_2\}$. If D involves the tuple (2 yy) of r_2, the tuple (3 2 2 s) of r_1' may block D via I_1, or may trigger the deletion of this tuple via I_2. The outcome of the operation depends on the order of ***rir's*** involving R_2 enforcement: (i) If I_2 is taken into account in first place (3 2 2 s) and (2 1 2 q) are deleted from r_1'; or (ii) if I_1 is first considered, D is blocked by the tuple (3 2 2 s) of r_1'.
The undesirable effects illustrated above increase when there is overlapping of the foreign key attributes [Clair98].

3.3. Problems in Updates

The study of all possible updates may be divided in three cases: i) **left update**: the foreign key FK_i of R_i is updated, and $FK_i \cap K_i = \varnothing$; ii) **right update:** the primary key K_j of R_j is updated, and $FK_j \cap K_j = \varnothing$; iii) **both sides update:** the primary key K_j of R_j is updated, and $FK_j \cap K_j \neq \varnothing$.

3.3.1. Right Updates

Only the update of values belonging to primary keys in the relations R_j will be taken into account. Let ***Ru*** be the update of one or more tuples of one relation. ***Ru*** may promote actions like those seen for deletions: i) the update of tuples referencing tuples involved in ***Ru*** via ***rir***'s with *Cascades* modality; or ii) the update of foreign key values in tuples referencing tuples involved in ***Ru*** via ***rir***'s with *Nullified* modality. On the other hand, if one tuple t references one tuple involved in ***Ru*** via one ***rir*** with a *Restricted* modality, t blocks the execution of ***Ru***.

Example 1-3: Consider the Example 1, excluding I_1 and I_2. Suppose that ***Ru*** changes the tuple (*a*) by (*d*) in r_3. The result of this update depends on the order in which the ***rir***'s involving R_3 are enforced: (i) if I_6 is enforced in first place, the tuple (*a x*) in r_4 is modified to (*d x*), this leads to the enforcement of I_5, which results in a failed attempt to modify the value of the attribute PROJ# in (1 4 4 *a x*) and (4 λ λ *a x*) of r_1 to (*d*), where the failure is due to the conflict of the new value by I_3; (ii) if I_4 is enforced in first place, the tuple (4 *a*) in r_2 is modified to (4 *d*), leading to the enforcement of I_3 which in turns assigns null values to the attributes DIR# and PROJ# in the tuple (1 4 4 *a x*) of r_1; then as a result of enforcing I_6, (*a x*) in r_4 is modified to (*d* x), and enforcing I_5 would provoke to modify the value of the attribute PROJ# in (4 - - *a* x) of the relation r_1 to (*d*), and (1 4 - - x) in r_1 does not hold any reference to r_4.

3.3.2. Left Updates

Only updates of values belonging to foreign keys will be taken into account. Let ***Lu*** be the update of one or more tuples of Ri. ***Lu*** may promote actions such as: i) the insertion of the tuples referenced by the tuples involved in ***Lu*** via ***rir***'s with *Cascades* modality; or ii) the update of foreign key values in tuples involved in ***Lu*** referencing non existing tuples in a relation linked via one ***rir*** with *Nullified* modality. On the opposite if one tuple t references a non existing tuple in a relation referenced via one ***rir*** with a *Restricted* modality, t blocks the execution of ***Lu***. Problematic cases in left updates are the same that those for insertions.

3.3.3. Both-Side Updates

Only the update of values belonging to both, primary keys and foreign keys in a relation belonging to two ***rir***'s as the right side and the left side respectively, will be taken into account. The problematic cases for both-side update operations are the same that those detected for left and right updates.

4. STATIC DETECTION OF ANOMALIES

In section 3, different anomalies in the manipulation of data, has been examined.

In this section the mechanisms needed to detect the potential presence of anomalies will be detailed. This will be done adhering to the [Markowitz94] and [Casanova89] approaches. Safeness conditions must ensure:

1 - Data manipulations must produce only one result regardless the order in which the ***rir***'s are enforced and the order in which the tuples are accessed.

2 - A data manipulation, must map a consistent database state r to another consistent database state r', this is in concordance with the immediate mode verification used in this article.

The present study is driven by the immediate mode in what refers to integrity verification. On the other hand, deferred mode is an advantageous and even a mandatory well-known strategy for integrity maintenance. However, the full understanding of the immediate mode is required for the analysis of deferred mode that is under development in this project.

The relations in whom the anomalies may appear during the execution of operations over the database can be determined through safeness conditions. To accomplish that, the following sets of relations will be defined: ***C(Ri), R(Ri), N(Ri), CDir(Ri), RDir(Ri),*** and ***NDir(Ri)***. These sets are formed by elements **(R_j, FK)** including R_i, where $R_j \in$ **R**, and FK is one foreign key associated with R_j.

4.1. Insert Operations

The sets needed to detect sources of inconsistencies are:

- ***CDir(Ri)*** contains elements **(R_j,FK)**, where R_j is a relation connected in **G** to R_i by one edge corresponding to one ***rir*** with *Cascades* modality for insertions.

- ***C(Ri)*** contains elements **(R_j,FK)**, where one relation R_t of ***C(Ri)*** or ***CDir(Ri)*** is connected in **G** to R_j by an edge corresponding to a ***rir*** with a *Cascades* modality for insertions, of the form $R_t[FK]<<R_j[K_j]:(c,\beta,\mu_i,\mu_d)$, where: $FK \subseteq K_t$, and K_t is primary key of R_t;

- ***NDir(Ri)*** contains elements **(R_j,FK)**, where R_j is one relation connected in **G** to R_i by one edge corresponding to a ***rir*** with *Nullified* modality for insertions and for every $X \subseteq Y$, there does not exist any ***nna*** $R_i:X \neq \lambda$.

- ***RDir(Ri)*** contains elements of the form **(R_j,FK)**, where R_j is one relation connected in **G** to R_i by one arc corresponding to one of the followings: (1) one ***rir*** $R_i[FK]<<R_j[K_j]:(\mathbf{r},\beta,\mu_i,\mu_d)$; (2) one ***rir*** $R_i[FK]<<R_j[K_j]:(\mathbf{n},\beta,\mu_i,\mu_d)$, where there are at least one $X \subseteq FK$, with one ***nna*** of the form $R_i:X \neq \lambda$.

- ***R(Ri)*** contains elements of the form **(R_j,FK)**, where one relation R_t of ***C(Ri)*** or ***CDir(Ri)*** is connected in G to R_j by one edge corresponding to: (1) one ***rir*** $R_t[FK]<<R_j[K_j]:(\mathbf{r},\beta,\mu_i,\mu_d)$ and $FK \subseteq K_t$, where K_t is the primary key of R_t; (2) one ***rir*** $R_t[FK]<<R_j[K_j]:(\mathbf{n},\beta,\mu_i,\mu_d)$, where $FK \subseteq K_t$, where K_t is the primary key of R_t and there exists $X \subseteq FK$, with at least one ***nna*** $R_t:X \neq \lambda$.

Note that in delete or right update operations, the relation that enforces the ***rir*** by *Nullified* does not suffer any modification in its attributes since it propagates the operation's effect to the left relation. By the contrary, insertions or left updates by *Nullified*, sets null their own involved attributes, voiding any reference to another relation. This is why the definition of the sets ***RDir***, ***CDir*** and ***NDir*** is needed. It may be said that the relational scheme ***R*** is safe in insert operations <u>**iff**</u> for every relation R_i of **R**:

I1) there is not any relation R_j of **R** belonging to ***C(Ri)*** or ***CDir(Ri)*** and ***R(Ri)*** or ***RDir(Ri)*** at the same time;

I2) there is not any pair of elements (R_j,Y) and (R_k,Y') belonging to ***CDir(Ri)*** and ***NDir(Ri)*** respectively, where: (i) R_j=R_k or (ii) Y and Y' overlaps;

I3) there is not any relation R_j of **R** belonging to ***C(Ri)*** and ***NDir(Ri)*** at the same time;

I4) there is not exist any pair of elements (R_j,Y) and (R_k,Y') belonging to ***NDir(Ri)*** respectively, where Y and Y' strictly overlaps;

I5) there is not any pair of elements (R_j,Y) and (R_k,Y') belonging to ***RDir(Ri)*** and ***NDir(Ri)*** respectively, and Y and Y' overlaps.

Example: The relational scheme of the Example 1-1 of section 3.1 does not satisfy **I1**, since the relation $\mathbf{R_3}$ is involved in the sets ***R(R1)*** and ***C(R1)***, then $\mathbf{R_1}$ is a possible source of unpredictable results during inserts.

Safe conditions may be used to place the affected relations during insert operations: i) if **I1** is not satisfied, the affected relation is R_j which is involved in ***C(Ri)*** or ***CDir(Ri)***) and ***R(Ri)*** or ***RDir(Ri)*** at the same time; ii) if **I3** is not satisfied, the affected relation is R_j which is involved in ***C(Ri)*** and ***NDir(Ri)*** at the same time; iii) if **I2**, **I4** or **I5** are not satisfied, the place where different results may appear is R_i.

Proposition: For every relation R_i of **R**, for every database state r associated with ***R***, and for every insertion I involving one or more tuples of the relation r_i of r associated with the relation R_i, I maps r into an unique state of the database <u>**iff**</u> ***R*** satisfies the previous conditions. In [Rivero99] the proof of that proposition is depicted.

4.2. Delete Operations

For delete operations, sets that help to detect conflictive nodes are:

- ***C(Ri)*** contains the element **(R_i, -)**, and elements **(R_j, FK)**, where R_j is a relation connected to R_i in **G** for an oriented path formed by edges corresponding to ***rir***'s with *Cascades* mode for deletions, such that the first edge is labeled $FK:(\alpha,\mathbf{c},\mu_i,\mu_d)$.

- ***N(Ri)*** contains elements **(R_j, FK)**, where R_j is a relation connected to R_m of ***C(Ri)*** in **G** by an edge corresponding to a ***rir*** $R_j[FK]<<R_m[K_m]:(\alpha, \mathbf{n}, \mu_i, \mu_d)$, and for each $X \subseteq FK$, X is allowed to have null values.

- ***R(Ri)*** contains elements **(R_j, FK)**, where R_j is a relation connected to R_m of ***C(Ri)*** in **G** by an edge corresponding to: (1) a ***rir*** with *Restricted* mode and labeled with the foreign key FK; (2) a ***rir*** $R_j[FK]<<R_m[K_m]:(\alpha, \mathbf{n}, \mu_i, \mu_d)$, such that there exists $X \subseteq FK$, and may not take null values.

By involving those operations that may be rejected because they try to nullify attributes associated to a ***nna*** to the set

$\mathbf{R(R_j)}$, the sets $\overline{Casc}$ and $\overline{Restr}$ defined by Markowitz (1994), are no longer required [Rivero98].

The relational schema $\boldsymbol{R}$ is sure **iff** for each R_i in **R**:

D1) there is not any relation R_j of **R** involved in both $C(R_i)$ and $R(R_i)$; this condition avoids the deletion of tuples that block the operation when another path is followed;

D2) there is not any pair of elements (R_j, Y) and (R_j, Y') involved in $C(R_i)$ and $N(R_i)$ respectively, such that Y and Y' overlap. It avoids the updating or deletion according to the path of ***rir***'s that is enforced in the first place;

D3) there is not any pair of elements (R_j, Y) and (R_j, Y') belonging to $R(R_i)$ and $N(R_i)$ respectively, such that Y and Y' overlap. This condition prevents that the tuples affected by a delete operation were modified or stay unaltered blocking the operation, according to the order in which the restrictions are verified;

D4) there is not any pair of elements (R_j, Y) and (R_j, Y') belonging to the set $N(R_i)$, such that Y and Y' strictly overlap. In such a way different results when updates with *Nullifies* strategy are performed, are avoided.

Other combinations of sets either are symmetric to those previously exposed or do not produce anomalies. The relational schema of Example 1 is unpredictable since $\mathbf{R_2}$ is involved in $R(R_i)$ and $C(R_i)$. $\mathbf{R_1}$ is a possible source of unpredictable results when a delete operation is performed since it not satisfies **D1**. The relational schema of Example 2 does not satisfies **D1** because $\mathbf{R_1}$ is involved in both $R(R_2)$ and $C(R_2)$; in such a way, $\mathbf{R_2}$ is a source of potential anomalies.

Safety conditions permit to establish that the source of anomalies is the relation R_i and the places where anomalies occur is: the R_j that is involved in $C(R_i)$ and $R(R_i)$, when **D1** is not satisfied; the R_j that is involved in $C(R_i)$ and $N(R_i)$ with the elements (R_j,Y) and (R_j,Y') respectively in such a way that Y and Y' overlap, when **D2** is not satisfied; the R_j that is involved in $R(R_i)$ and $N(R_i)$ with the elements (R_j,Y) and (R_j,Y') respectively in such a way that Y and Y' overlap, if **D3** is not satisfied; the R_j that is involved in $N(R_i)$ with the elements (R_j,Y) and (R_j,Y') respectively in such a way that Y and Y' strictly overlap if **D4** is not satisfied. In Markowitz (1994) the proof of necessity and sufficiency of that conditions, is sketched.

4.3. Update Operations

In the same way as previous operations, sets of relations are built in order to find sources of potential anomalies.

4.3.1. Right Updates

In this case the following sets must be built:

- $C(R_i)$ has the element ($\mathbf{R_i}$, -), and elements ($\mathbf{R_j}$, **FK**), such that there exists an element (R_k, S_k) belonging to $C(R_i)$, where R_j is a relation of **R** linked to R_k in **G** by an edge corresponding to a ***rir*** $R_j[FK]<<R_k[K_k]:(\alpha,\beta,\mu_i,c)$ with a *Cascades* option for updates, and: (a) $S_k = \emptyset$ or (b) $S_k \cap K_k \neq \emptyset$.
- $N(R_i)$ contains elements ($\mathbf{R_j}$,**FK**), such that there exists an element (R_k,S_k) belonging to $C(R_i)$, where R_j is linked to R_k in **G** by an edge corresponding to a ***rir*** $R_j[FK]<<R_k[K_k]:(\alpha,\beta,\mu_i,\mathbf{n})$ such that for each $X \subseteq Y$, there does not exist any ***rnn*** R_j:$X \neq \lambda$ and: (a) $S_k = \emptyset$ or (b) $S_k \cap K_k \neq \emptyset$.
- $R(R_i)$ is formed by elements ($\mathbf{R_j}$, **FK**), such that there exists an element (R_k,S_k) belonging to $C(R_i)$, where R_j is a **R**'s relation connected to R_k in **G** by an edge corresponding to: (1) a ***rir*** $R_j[FK] << R_k[K_k]$: (α, β,μ_i, **b**) such that: (a) $S_k = \emptyset$ or (b) $S_k \cap K_k \neq \emptyset$; (2) a ***rir*** $R_j[FK] << R_k[K_k]$: (α,β,μ_i,**n**) such that there exists $X \subseteq FK$, associated to a ***rnn*** R_j:$X \neq \lambda$ and: (a) $S_k = \emptyset$; or (b) $S_k \cap K_k \neq \emptyset$.

It may be stated that a relational schema $\boldsymbol{R}$ is safe under right updates **iff** for each relation R_i of **R**:

U_r1) there is not any relation R_j of **R**, such that there exists a pair of elements (R_j,Y) and (R_j,Y') belonging to $C(R_i)$ and $R(R_i)$ respectively and Y and Y' overlap;

U_r2) there is not any relation R_j of **R**, such that there exists a pair of elements (R_j,Y) and (R_j,Y') belonging to $C(R_i)$ and $N(R_i)$ respectively and Y overlaps Y';

U_r3) there is not any relation R_j of **R**, such that there exists a pair of elements (R_j,Y) and (R_j,Y') belonging to $R(R_i)$ and $N(R_i)$ respectively and Y overlaps Y';

U_r4) there is not any relation R_j of **R**, such that there exists a pair of elements (R_j,Y) and (R_j,Y') belonging to the set $N(R_i)$, and Y and Y' strictly overlap.

Using those four conditions the sources of anomalies when updates to the right side relation are performed, may be determined. For all cases, the source is the relation R_i. If **U_r1**, is not satisfied the irregularity will occur in a R_j contained in both $C(R_i)$ and $R(R_i)$ with the elements (R_j,Y) and (R_j,Y') respectively, where Y and Y' overlap. Analogously, when **U_r2** is not satisfied the anomalies will be placed in a R_j involved in both $C(R_i)$ and $N(R_i)$ with the elements (R_j,Y) and (R_j,Y') respectively, where Y overlaps Y'. The same situation occurs when **U_r3** is not attained. If **U_r4**, is violated the place of anomalies will be R_j contained in $N(R_i)$ with the elements (R_j,Y) and (R_j,Y') respectively where Y and Y' strictly overlap.

4.3.2. Left Updates

Insecure cases for referential integrity when left updates are performed are the same as those studied for insertions, if their modalities agree. The composition of the set changes because the first edge must be considered with the update option but the following ones must be seen as the ones corresponding to insertions. Safety conditions are the same.

4.3.3. Both-Side Updates

In this case, problematic cases are the same as those studied under left and right updates, then their analysis may be summarized according to what was indicated for those operations.

5. GENERATION OF RULES

For each one of the relations that appeared as potential sources of anomalies during the static analysis of the schema, rules were built. They will permit to determine if anomalies are present in a given database state. Such rules are expressed as serial combinations using the logical operations **not**, **and** and the composition operator **o**. A serial combination from a specific relation, is an expression representing a directed path in **G**, and links nodes with an incidence degree (delete and right updates) or divergence degree (insertion and left updates) equal to 1.

The partial divergence (incidence) degree (**pdd** and **pid** respectively) of a vertex is defined as the number of significant edges that leave (reach) it. A significant edge is defined according to the operation: i) for deletions and left updates, every edge is a significant one; ii) for insertions a significant edge is one representing a ***rir*** $R_i[W] << R_j[K_j]:(\alpha, \beta, \mu_i, \mu_d)$, with $W \subseteq K_j$; iii) for right updates a significant edge is one representing a ***rir*** $R_i[W] << R_j[K_j]:(\alpha, \beta, \mu_i, \mu_d)$, with $W \cap K_j \neq \emptyset$.

In the restriction graph, seven types of nodes will be distinguished according to their partial incidence or divergence degrees: 1) **source node (pid**=0); 2) **sink node (pdd**=0); 3) **unifier node (pid**≥2 and **pdd**=1); **branch node (pdd**≥2 and **pid**=1); **passing node** (**pid**=1 and **pdd**=1); **multiple node (pid**≥2 and **pdd**≥2); **isolated node (pid**=0 and **pdd**=0).

5.1. Insertion Rules

In order to build the rules corresponding to potentially anomalous insert operations, the following serial combinations must be considered:

- $\mathbf{C^+}$ ($\mathbf{I_j}$) ≡ representing a **G**'s directed path from a non-sink node (R_i) to a non-source node (R_{jn}), where the edges of the path represent the following ***rir***'s: $R_i[FK_i] << R_{j1}[K_{j1}]: (\mathbf{c}, \beta, \mu_i, \mu_d)$; $R_{j1}[FK_1] << R_{j2}[K_{j2}]:(\mathbf{c}, \beta, \mu_i, \mu_d)$; ...; $R_{jn-1}[FK_{n-1}] << R_{jn}[K_{jn}]:(\mathbf{c}, \beta, \mu_i, \mu_d)$; with $FK_1 \subseteq K_{j1}$; $FK_2 \subseteq K_{j2}$; ..; $FK_{n-1} \subseteq K_{jn-1}$ and the first edge in the path is I_j.
- $\mathbf{N}$ ($\mathbf{I_j}$) ≡ representing a directed path in **G**, composed by an only edge, that corresponds to the ***rir*** $I_j:R_i[FK] << R_m[K_m]:(\mathbf{n}, \beta, \mu_i, \mu_d)$, such that for each $X \subseteq FK$, X is allowed to be null.
- $\mathbf{R}$ ($\mathbf{I_j}$) ≡ representing a directed path in **G**, composed by a unique edge that corresponds to: (1) the ***rir*** $I_j:R_i[FK] << R_m[K_m]:(\mathbf{r}, \beta, \mu_i, \mu_d)$; or (2) the ***rir*** $I_j:R_i[FK] << R_m[K_m]:(\mathbf{n}, \beta, \mu_i, \mu_d)$ such that $X \subseteq FK$ exists and X is restricted by a ***rnn***.
- $\mathbf{C_r}$ ($\mathbf{I_j}$) ≡ representing a directed path in **G** leaving from a non-sink node (R_i) and reaching a non-source node (R_j), where the edges correspond to the following ***rir***'s: (1) $R_i[FK_i] << R_{j1}[K_{j1}]:(\mathbf{c}, \beta, \mu_i, \mu_d)$;$R_{j1}[FK1] << R_{j2}[K_{j2}]:(\mathbf{c}, \beta, \mu_i, \mu_d)$;...;$R_{jn-1}[FK_{n-1}] << R_{jn}[K_{jn}]:(\mathbf{c}, \beta, \mu_i, \mu_d)$;$R_{jn}[FK_n] << R_j[K_{jj}]:(\mathbf{r}, \beta, \mu_i, \mu_d)$; with $FK_1 \subseteq K_{j1}$; $FK_2 \subseteq K_{j2}$;...; $FK_n \subseteq K_{jn}$ and the first edge corresponds to I_j or (2) $R_i[FK_i] << R_{j1}[K_{j1}]:(\mathbf{c}, \beta, \mu_i, \mu_d)$; $R_{j1}[FK1] << R_{j2}[K_{j2}]:(\mathbf{c}, \beta, \mu_i, \mu_d)$;...; $R_{jn-1}[FK_{n-1}] << R_{jn}[K_{jn}]:(\mathbf{c}, \beta, \mu_i, \mu_d)$; $R_{jn}[FK_n] << R_j[K_j]:(\mathbf{n}, \beta, \mu_i, \mu_d)$; with $FK_1 \subseteq K_{j1}$; $FK_2 \subseteq K_{j2}$;...; $FK_n \subseteq K_{jn}$, where the first edge is associated to I_j; $X \subseteq FK_n$ exists and X is not allowed to have null values.

Algorithm: For an insertion operation over a table that is a source of anomalies, a set of trees will be built in order to support the rules generation, applying the following algorithm:

1. Set the source of anomalies table as the root of the tree. Set its straight descendents in the graph as their children in the tree (the number of children of the root node will be equal to the partial divergence degree of the node in the graph).

*2. In each one of the branches of the tree (each of the internal nodes have only one child), combine serially all sequences of nodes with a partial divergence degree equal to 1, until a node with an ancestor reaching it with an option not equal to **Cascades** or a node with a partial divergence degree not equal to 1 is reached.*

For each tree, a rule is built. Each one of the branches of the trees will be a serial combination since each internal node has a unique child. They will be assembled by means of the **and** operator. If the branch represents a serial combination C_r **o** R it will be preceded by the **not** operator. If a branch ends in a node that is the root of another tree, the serial combination of that branch (C^+) is composed (**o**) with the rule corresponding to that relation. Each one of the paths is considered in such a way that the treatment of the same anomaly twice or more times is avoided. If a relation has no associated rule, it is because it never produces anomalies when insert operations are performed in it.

5.2. Delete Rules

The analysis of the different paths in the restriction graph is analogous to that exposed for insertions, but in this case the graph is scanned in the reverse direction. Besides, the algorithm is quite similar to the one already depicted in the previous section.

Example: The rule for the source of anomalies R_3, according to the graph of Figure 1, is Rule R_3: $C^+(I_6)$ **and** (**not** $R(I_4)$). Figure 3 shows their construction. For the evaluation of a rule the knowledge of the database state, is essential. The mechanisms that perform the evaluation should be refined in order to obtain an acceptable level of efficiency; on the contrary the proposed strategy will not be applicable.

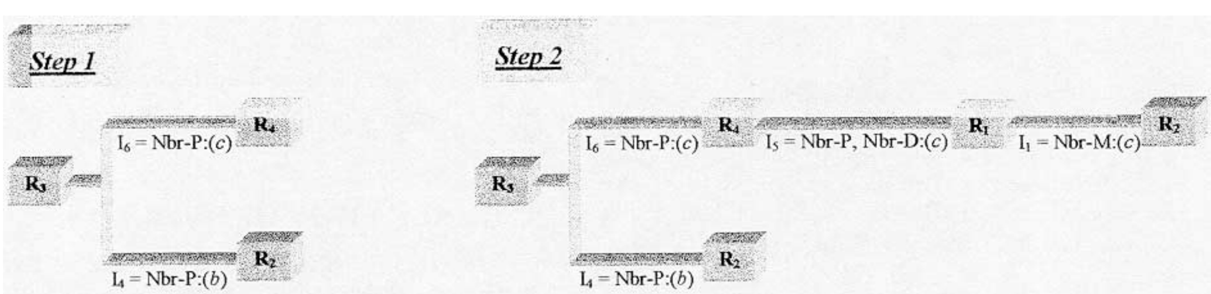


Figure 3: Rule for R_3: $C^+(I_6)$ and (not $R(I_4)$)

5.3. Update Rules

Regarding update operations, different situations may occur: i) in case of right updates, the rules and the algorithm are similar to the one for deletions, taking into account the overlapping of the attributes; ii) in case of left updates, the algorithm is quite similar to the one for insertions, but in the first step, the update strategy must be considered. From that point, this operation becomes an insertion. For that reason, if in the generated tree there is a leaf reached by the root, with a left update strategy *'Cascades'*, then the serial combination of that branch (C^+) will be composed (**o**) with the insertion rule of the relation represented by that node (Clair, 1998); iii) in case of both sides updates, rules and algorithms for one-side updates may be combined: **Rule R_i:** $Rule_{left}(R_i)$ and $Rule_{right}(R_i)$

6. GENERAL PROCEDURE

In order to analyze the schema, the algorithm of Figure 4 must be followed:

```
(Variables  G:Graph;  Vi: Vertex;  Problem: Boolean;
 N, C, R, NDir, Cdir, RDir: ListSure;  Rule: String )
  Init_Graph(G)
  Call Built_Graph(G)
  Foreach Vertex (Vi) in G Do
     Call FillSetDelete(G, Vi, N, C, R)
      Problem = False
     Call AnalyzeSetDelete(N, C, R, Problem)
     If Problem Then
         Call BuildRuleDelete(G, Vi, Rule)
            Call InsertRule (Vi, "d", Rule)
     End If
     Call FillSetUpdateRight(G, Vi, N, C, R)
      Problem = False
     Call AnalyzeSetUpdateRight(N, C, R, Problem)
     If Problem Then
         Call BuildRuleUpdateRight(G, Vi, Rule)
            Call InsertRule(Vi, "ur", Rule)
     End If
     Call FillSetInsert(G, Vi, NDir, CDir, RDir, C, R)
      Problem = False
     Call AnalyzeSetInsert(NDir, CDir, RDir, C, R, Problem)
     If Problem Then
         Call BuildRuleInsert(G, Vi, Rule)
            Call InsertRule(Vi, "i", Rule)
     End If
     Call FillSetUpdateLeft(G, Vi, NDir, CDir, RDir, C, R)
      Problem = False
     Call AnalyzeSetUpdateLeft (NDir, CDir, RDir, C, R, Problem)
     If Problem Then
         Call BuildRuleUpdateLeft(G, Vi, Rule)
            Call InsertRule(Vi, "ul", Rule)
     End If
  End Foreach
  End Main
```

Figure 4: General Procedure

7. CONCLUSIONS AND FUTURE WORK.

The effects of basic operations over relations in a conceptual schema with referential integrity constraints and null restrictions were studied. A minor simplification of the static analysis of deletions, developed by Markowitz (1994), was made. The analysis was extended to all update operations, in despite of the fact that insertions and updates over the left-hand side relation are generally performed with a *Restricted* modality.

In order to determine if a conceptual schema is sure with respect to all basic operations, algorithms related with each one of them are presented, extending by this way current research. A software tool was designed and implemented (DepuSem), for the analysis of the ***rir*** and ***nna***'s graph.

Important points of symmetry have been detected: i) problematic nodes for insertions and left updates are the same when the options are the same (obviously, the generated rules will be the same); ii) problematic nodes related to right updates are also problematic with respect to delete operations whenever their options are the same; iii) on the contrary, unsure nodes for delete operations are not inevitably unsure with respect to right updates, even if they have the same options. This is true because the propagation of right update operations requires the overlapping of primary and foreign keys.

The design of strategies for the efficient evaluation of the rules must be faced since the proposed monitoring process may become inapplicable if it slows down when the number of tables and relationships increases.

8. REFERENCES

[1] Casanova, M. et al.. Optimization of Relational schemes containing inclusion dependencies. *Proceedings of 15 VLDB Conference*. Amsterdam, pp.315-325. (1989)

[2] Casanova, M.; Tucherman, L.: Enforcing Inclusion Dependencies and Referential Integrity". *Proceedings 14th. VLDB Conference*. L.A. Calif. (1988).

[3] Clair, D.; Loureiro, D. y Vizcaino, M., Depurador Semántico de una Red de Restricciones. *Graduate Thesis*. Facultad Ciencias Exactas. Universidad Nacional del Centro. BA. Argentina. (1998)

[4] Markowitz, V., Safe Referential Integrity and Null Constraint Structures in Relational Databases. Personal communication. (1994)

[5] Markowitz, V.: Referential Integrity Revisited: An Object-Oriented Perspective. *Proccedings of the 16th VLDB Conference*. Brisbane. Australia. (1990)

[6] Markowitz, V.: Safe Referential Integrity Structures in Relational Databases. *Proceedings 17th. VLDB Conference*. Barcelona, España (1991).

[7] Rivero, L, Doorn, J and Loureiro, D., Semantic Integrity Constraint Network Validation. *Research report RR 1999-001* ISISTAN. Facultad Ciencias Exactas. Universidad Nacional del Centro. BA. Argentina. (1999)

[8] Rivero, L. and Doorn, J., Semantic Integrity in a Relational Database: Properties and Rules. *In Proc. XI International Conference of Systems Engineering*, ICSE-96, Las Vegas, 170-175 (1996)

[9] Rivero, L. et al., Un Analizador de una Red de Restricciones Semánticas. In Actas de Intercon'98. IEEE Perú. Tacna. Perú. (1998).