OPTIMIZATION OF RELATIONAL SCHEMAS CONTAINING INCLUSION DEPENDENCIES

Marco A. Casanova[1], Luiz Tucherman[1], Antonio L. Furtado[2], Anelise P. Braga[1]

[1]Rio Scientific Center - IBM Brazil - P.O Box 4624 - 22.071, Rio de Janeiro, RJ - Brazil

[2]Dept. de Informática - Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de S. Vicente, 225 - 22.453, Rio de Janeiro, RJ - Brazil

ABSTRACT

A two-step optimization strategy for relational schemas that contains a class of inclusion dependencies is described. Both steps take into account additional information that indicates how to preserve each inclusion dependency in the presence of insertions and deletions. The first step eliminates inclusion dependencies which are redundant with respect to both the semantics of the data and the behavior of the transactions. The second step discards dependencies through a structural transformation that again preserves the semantics of the data and of the transactions and that applies both to INF and to NF^2 relational schemas.

1. INTRODUCTION

Among the classes of integrity constraints considered for the relational model, we find the *inclusion dependencies* [CFP], or INDs. For example, one may declare an IND between tables SECRETARY and EMPLOYEE to capture that all secretaries are employees. The specification of an IND is often complemented with insertion/deletion options [CFT,Da] that indicate how to preserve the dependency in the presence of insertions and deletions. For example, one specify that the deletion of an entry of table EMPLOYEE must propagate to the deletion of the corresponding entry of SECRETARY, if any, and that the insertion of an entry of SECRETARY must be blocked if there is no corresponding entry of EMPLOYEE.

Proceedings of the Fifteenth International Conference on Very Large Data Bases

This paper describes an optimization process that minimizes the set of INDs of a relational schema by eliminating redundant dependencies and by redesigning the schema, without modifying the basic semantics of the data and the behavior of the transactions. The process assumes that all INDs are *key-based* and that are qualified with insertion/deletion options. A key-based IND, or K-IND, is just an IND whose right-hand side is a key. This restricted class of INDs is sufficiently powerful to capture important semantic properties, such as hierarchies of sets of objects or relationships between sets of objects, in an information model.

The process of eliminating redundant inclusion dependencies is based on the inference rules for INDs described in [CFP], adapted to K-INDs and to the insertion/deletion options.

The process of discarding K-INDs through a structural transformation may work in the context of 1NF schemas or in the context of NF^2 schemas. For 1NF schemas, the transformation eliminates a K-IND by collapsing the two tables the dependency relates into a single table. For NF^2 schemas, the transformation also eliminates an inclusion dependency by nesting the two tables related. The transformation is used only when the dependency meets certain criteria that guarantee that the new table will not contain undue redundancies and that the old tables can be redefined as views of the new table. Dependencies satisfying these criteria are called *trivializable*.

From another perspective, the optimization process can be viewed as a design discipline for schemas containing a class of K-INDs, which is defined both for the traditional and the NF^2 variations of the relational model. In this aspect, the process differs substantially from the well-known design disciplines for the relational model [Ma], as well as from those defined for the NF^2 variation [AMM,FG,OY,RKS], since it is based on inclusion dependencies whereas the vast

Amsterdam, 1989

majority of published disciplines are based on functional or multivalued dependencies or their generalizations.

The organization of the paper is as follows. Section 2 discusses the elimination of redundant K-INDs. Section 3 describes the structural transformation assuming that all tables must be in first-normal form. Section 4 shows how to adapt the transformation for the NF^2 version of the relational model. Finally, section 5 contains the conclusions.

The reader is referred to [CTFB] for a more detailed presentation.

2. ELIMINATING REDUNDANT KEY-BASED INCLUSION DEPENDENCIES

2.1 Preliminary Definitions

We assume a scenario where the database designer specifies the conceptual schema of a database within the traditional relational model, that is, using relation schemes in first-normal form. We will be interested in three classes of integrity constraints, the null constraints, indicating when attributes do not admit null values, keys and a class of inclusion dependencies which is sufficiently powerful to capture hierarchies of sets of objects and to help define relationships over sets of objects.

More precisely, a *relation scheme* is an expression of the form $R[A_1,...,A_n]$ where R is the *name* and $A_1,...,A_n$ is the list of *attribute names* of the scheme, taken from a given set of identifiers. We will frequently use the term "attribute" instead of "attribute name".

Let $\boldsymbol{E}$ be a set of relation schemes with distinct names in the rest of this subsection. A *state* for $\boldsymbol{E}$ is a function σ that associates an *n*-ary relation $\sigma(R)$ to each name R of a scheme $R[A_1,...,A_n]$ in $\boldsymbol{E}$.

We assume familiarity with the relational algebra and denote the value of a relational expression Q in a state σ of $\boldsymbol{E}$ by $\sigma(Q)$.

Let λ denote both the null value and tuples, with arbitrary length, of null values. A *null constraint* over $\boldsymbol{E}$ is an expression of the form "R: $X \neq \lambda$" where R is the name of a relation scheme in $\boldsymbol{E}$ and X is a sequence of distinct attributes of R. We say that each attribute A in X *does not admit null values* in R. A state σ for $\boldsymbol{E}$ *satisfies* "R: $X \neq \lambda$" iff $t_A \neq \lambda$, for each tuple t in $\sigma(R)$ and for each attribute A in X.

A *key* over $\boldsymbol{E}$ is an expression of the form "R: K" where R is the name of a relation scheme in $\boldsymbol{E}$ and K is a sequence of distinct attributes of R. A state σ for $\boldsymbol{E}$ *satisfies* "R: K" iff, for each pair of tuples t,u in $\sigma(R)$, if $t_K = u_K$ then $t = u$ and, for each tuple v in $\sigma(R)$, $v_A \neq \lambda$, for each attribute A in K. Therefore, by definition, "R: K" implies "R: $K \neq \lambda$" and we also say that every attribute in a key "R: K" *does not admit null values* in R.

Let $\boldsymbol{K}$ be a set of keys over $\boldsymbol{E}$. We say that a key "R: K" in $\boldsymbol{K}$ is *minimal* iff there is no other key in $\boldsymbol{K}$ of the form "R: L" such that all attributes in L also occur in K. In the definition of a relational schema, to be soon introduced, we will indeed require that every key be minimal.

Let $\boldsymbol{K}$ be a set of keys and $\boldsymbol{N}$ be a set of null constraints over $\boldsymbol{E}$. A *key-based inclusion dependency* over $\boldsymbol{E}$, $\boldsymbol{K}$ and $\boldsymbol{N}$, or a *K-IND*, is an expression of the form $R_1[X_1] \subseteq R_2[X_2]:(\gamma,\delta)$ where, for i = 1,2,

- R_i is the name of a relation scheme in $\boldsymbol{E}$;
- X_i is a sequence of distinct attributes of R_i such that X_1 and X_2 have the same length and X_2 is defined as a key of R_2 in $\boldsymbol{K}$;
- γ is the *insertion option* and δ is the *deletion option* of the K-IND, taking values from the set $\{b^i, b^d, p^i, p^d\}$, with the following intended interpretation:

  b^i block immediately
  p^i propagate immediately
  b^d block deferredly
  p^d propagate deferredly

  with the restriction that γ can be equal to p^i or p^d only if all attributes of R_2, excluding those in X_2, admit null values, that is, no null constraint in $\boldsymbol{N}$ says the contrary.

The restrictions imposed on γ just avoid indeterminacies when propagating insertions into R_1 to insertions into R_2. Indeed, an insertion into R_1 determines only the value of the attributes in X_2. Hence, if some attribute A of R_2 did not admit a null value, the propagation of the insertion would have to arbitrate some non-null value for A, which is not reasonable.

The semantics of K-INDs has a *static* and a *dynamic* perspective. The static perspective is reflected by

defining that a state σ for E *satisfies* $R_1[X_1] \subseteq R_2[X_2]$:$(\gamma,\delta)$ iff $\sigma(R_1[X_1]) \subseteq \sigma(R_2[X_2])$.

The dynamic perspective refers to the fact that the insertion and deletion options affect the behavior of the operations over R_1 and R_2 as defined in [CFT,Da]. For example, if the deletion option is b^i, for block immediately, then there is a test that rejects the deletion of a tuple t from R_2 in a state σ, if $t_{X_2} \in \sigma(R_1[X_1])$. Note that, since X_2 is a key of R_2, no other tuple t' in $\sigma(R_2)$ is such that $t'_{X_2} = t_{X_2}$. Moreover, the test need not consider updates on X_2 since we do not permit updates on keys. If the deletion option is b^d, for block deferredly, there is a test that aborts the transaction if the state φ at commit time is such that $\varphi(R_1[X_1]) \not\subseteq \varphi(R_2[X_2])$.

A *relational schema* is a pair $S=(E,I)$ where E is a set of relation schemes with distinct names and I is a set of null constraints, keys and K-INDs over E such that every key in I is minimal. A *state* of S is a state of E. A state of S is *consistent* iff it satisfies all constraints in I.

We recall that, given a set D of dependencies and a dependency d over a set of relation schemes E, we say that d is a *logical consequence* of D iff any state σ of E that satisfies all dependencies in D also satisfies d.

Let $S=(E,I)$ be a relational schema and T be a transaction over S. The *behavior* of T in S is the set of pairs (σ,σ') of states of E such that there is an execution of T, starting in σ and terminating in σ', that correctly fires all possible triggers and assertions, immediate or deferred, associated with the insertion/deletion options of K-INDs in I.

Let $S=(E,I)$ be a relational schema and d be a K-IND in I. We say that d is *redundant* in I iff

- d is a logical consequence of $I-\{d\}$;
- every transaction T over S has the same behavior in S and in $S'=(E,I-\{d\})$.

The notion of redundancy just extends the usual notion of logical consequence for dependencies to accommodate the behavior of transactions. For this reason, it was phrased in a non-traditional way.

2.2 A Motivating Example

Consider a database storing information about software modules, where a module can be in three stages: planned, specified and implemented. Assume that every implemented module must have been first specified and that every specified module must have been first planned. Then, a simplified relational schema for this database would contain the relation schemes

P[NAME,DESC]
S[NAME,SPEC]
I[NAME,CODE]

where P, S and I stands for 'planned module', 'specified module' and 'implemented module', respectively, and the following K-INDs:

(d^0_1) S[NAME]$\subseteq$P[NAME]: (b^i,p^i)
(d^0_2) I[NAME]$\subseteq$S[NAME]: (b^i,p^i)
(d^0_3) I[NAME]$\subseteq$P[NAME]: (b^i,b^i)

Let γ_i and δ_i be the insertion and deletion options of d^0_i. We intuitively prove that d^0_3 is only partially redundant or, more precisely, we argue that: (1) d^0_3 is a logical consequence of d^0_1 and d^0_2; (2) γ_3 is redundant; but (3) δ_3 is not redundant.

Indeed, (1) follows directly from the semantics of K-INDs. To prove (2), note that γ_3 is a consequence of γ_2 and the definition of d^0_1. Indeed, since γ_3 is b^i, γ_3 will block the insertion of a new implemented module m iff m is not yet planned. But if m is not yet planned, by d^0_1, then m is not yet specified. Hence, since γ_2 is b^i, γ_2 will also block the insertion of m, making it unnecessary to define γ_3. Finally, to prove (3), observe that, as δ_3 is b^i, δ_3 will reject the deletion of a planned module m, if m is already implemented, independently of any other assertion and trigger. On the other hand, if δ_3 were not specified, δ_1 and δ_2 alone would propagate the deletion of a planned module m to the deletion of its specification and implementation. Hence, d^0_3 is not redundant because its deletion option affects the behavior of transactions.

Suppose now that we replace d^0_1 by:

(d^0_1') S[NAME]$\subseteq$P[NAME]: (b^i,b^i)

Then, d^0_3 becomes redundant for d^0_1' and d^0_2, which indicates that we may drop d^0_3 without affecting the semantics of the data and the behavior of the transactions. Intuitively, by d^0_2, all implemented modules must have been specified. Hence, if the deletion of a planned module m is blocked because m has already been implemented, the deletion will also be blocked because m must have already been specified.

In general, the database designer may specify a relational schema that contains any set of K-INDs. The goal of this section therefore is to analyse which of the K-INDs specified are redundant.

2.3 Conditions for Redundancy of K-INDs

To analyse when a K-IND is redundant in the presence of a set of K-INDs, we will adapt the inference rules for INDs given in [CFP] to take into account insertion and deletion options. In particular, one of the rules will use regular expressions to define sets of sequences of options. Thus, for example, the regular expression $(p^x \cup b^x)^+ b^x(p^x \cup b^x)^*$ denotes the set containing the sequences p^xb^x, b^xb^x, $p^xb^xp^x$, $p^xb^xb^x$, $b^xb^xp^x$, $b^xb^xb^x$, ...

The modified inference rules are:

(Extended Reflexivity)

$R[X] \subseteq R[X]:(p^i,p^i)$, if X is a key of R

(Extended Permutation)

if $S[B_1,...,B_m] \subseteq R[A_1,...,A_m]:(\gamma,\delta)$
then $S[B_{i_1},...,B_{i_m}] \subseteq R[A_{i_1},...,A_{i_m}]:(\gamma,\delta)$,
where $i_1,...,i_m$ is a permutation of 1,...,m

(Extended Transitivity)

if $R_{i-1}[X_{i-1}] \subseteq R_i[X_i]:(\gamma_i,\delta_i)$, for $i = 1,...,k$,
then $R_0[X_0] \subseteq R_k[X_k]:(\gamma,\delta)$
where γ and δ are as follows (with $x \in \{i,d\}$):

$\gamma_1...\gamma_k$	γ
$(p^x)^+$	p^x
$(b^x \cup p^x)^*b^x(b^x \cup p^x)^+$	p^x,b^x
$(p^x)^*b^x$	b^x

$\delta_1...\delta_k$	δ
$(p^x)^+$	p^x
$(b^x \cup p^x)^+ b^x(b^x \cup p^x)^*$	p^x,b^x
$b^x(p^x)^*$	b^x

Intuitively, the antecedents of the extended transitivity rule are ordered. Line 1 of the first table says that if all insertion options are of the form p^x then the insertion option of the conclusion must also be p^x; line 2 says that if all insertion options are immediate or all deferred (i.e., $x = i$ or $x = p$) and at least one insertion option *before the last one* is b^x then the insertion option of the conclusion can either be p^x or b^x; line 3 says that if all insertion options are p^x, except the last insertion which is b^x, then the insertion option of the conclusion must be b^x. The second table has a similar interpretation for deletion options.

For example, from

$R_0[X_0] \subseteq R_1[X_1]:(p^i,p^i)$
$R_1[X_1] \subseteq R_2[X_2]:(p^i,p^i)$
$R_2[X_2] \subseteq R_3[X_3]:(b^i,p^i)$

we may derive just $R_0[X_0] \subseteq R_3[X_3]:(b^i,p^i)$. Indeed, the insertion options of the K-INDs form the sequence $p^ip^ib^i$, which is in the set denoted by $(p^i)^*b^i$, but not in the sets denoted by $(p^i)^+$ or $(b^i \cup p^i)^*b^i(b^i \cup p^i)^+$, and hence the insertion option of the derived K-IND must be b^i. Moreover, the deletion options form the sequence $p^ip^ip^i$, which is in the set denoted by $(p^i)^+$, but not in the sets denoted by $(b^i \cup p^i)^+ b^i(b^i \cup p^i)^*$ or $b^i(p^i)^*$, and hence the deletion option of the derived K-IND must be p^i. Therefore, $R_0[X_0] \subseteq R_3[X_3]:(b^i,p^i)$ follows from the other three EK-INDs by the extended transitivity rule.

These rules are sound, which means that whenever a relational schema $\boldsymbol{S} = (\boldsymbol{E},\boldsymbol{I})$ has a K-IND $\boldsymbol{d}$ in $\boldsymbol{I}$ which can be obtained by the extended rules from the other dependencies in $\boldsymbol{I}$, we may optimize the database design by replacing $\boldsymbol{S}$ by a new schema $\boldsymbol{S'} = (\boldsymbol{E},\boldsymbol{I'})$, where $\boldsymbol{I'} = \boldsymbol{I} - \{\boldsymbol{d}\}$. This transformation will affect neither the semantics of the data nor the semantics of the transactions since a state σ of $\boldsymbol{E}$ satisfies $\boldsymbol{I}$ iff it satisfies $\boldsymbol{I'}$ and any transaction $\boldsymbol{T}$ has the same behavior in $\boldsymbol{S}$ and in $\boldsymbol{S'}$.

The reader is referred to [CTFB] for a fast algorithm testing redundancy for a special class of K-INDs.

3. ELIMINATING K-INDS IN THE 1NF RELATIONAL MODEL BY STRUCTURAL TRANSFORMATIONS

3.1 Preliminary Definitions

We now address an optimization process that minimizes the set of K-INDs of a relational schema by a structural transformation that preserves first-normal form. The transformation eliminates each K-IND that satisfies certain criteria by collapsing the two tables related by the dependency into a single table. The transformation is made transparent to users by redefining the original tables as views over the new table and by mapping their operations onto operations on the new table. The transformation is advantageous

exactly because the view operations are much faster in most cases than the corresponding operations on the original relational schema, that is, on the schema contained the non-trivialized K-IND.

The definition of the transformation requires the following additional concepts. Let $\boldsymbol{E}$ be a set of relation schemes with distinct names. A *view definition* over $\boldsymbol{E}$ is a triple $(V[A_1,...,A_n],Q,T)$ where

- $V[A_1,...,A_n]$ is a relation scheme whose name is distinct from the names of the schemes in $\boldsymbol{E}$;
- Q is an n-ary relational expression over $\boldsymbol{E}$;
- T is a specification of correct translations for the operations over V into operations over the schemes in $\boldsymbol{E}$ (see [FC]).

When the specification of T is not of interest, we will denote the view definition by $V[A_1,...,A_n] = Q$. We also say that V is the *name* of the view and that Q is the *defining expression* of the view.

Let $\boldsymbol{V}$ be a set of view definitions over $\boldsymbol{E}$. Suppose that the schemes in $\boldsymbol{E}$ and those defined in $\boldsymbol{V}$ have distinct names. We extend a state σ of $\boldsymbol{E}$ to $\boldsymbol{V}$ by assigning to each view V in $\boldsymbol{V}$, whose defining expression is Q, the value $\sigma(Q)$.

Although a defining expression may be any valid relational expression, the structural transformation will produce only λ-restrictions, defined below, followed by projections.

Let λ again denote both the null value and tuples, with arbitrary length, of null values. We define a *λ-restriction* over a relation scheme R in $\boldsymbol{E}$ as a restriction of the form $R[A_{i_1} \neq \lambda]...[A_{i_k} \neq \lambda]$, where $A_{i_1},...,A_{i_k}$ are attributes of R. We will frequently use the following equivalences for relational expressions:

Original Expression	*Equivalent Expression*
$R[X][Y]$	$R[Y]$, if $Y \subseteq X$
$R[X \neq \lambda][Y \neq \lambda]$	$R[XY \neq \lambda]$
$R[X][Y \neq \lambda]$	$R[Y \neq \lambda][X]$, if $Y \subseteq X$

The transformation also requires extending the classes of integrity constraints previously introduced to relations defined by λ-restrictions.

Thus, an *extended null constraint* over $\boldsymbol{E}$ is an expression of the form "$P: X \neq \lambda$" and an *extended key* over $\boldsymbol{E}$ is an expression of the form "$P: X$" where P either is equal to the name R of a relation scheme in $\boldsymbol{E}$ or is a λ-restriction over R and X is a sequence of distinct attributes of R. The semantics of both classes of constraints is a direct generalization of the semantics of the original classes.

Let $\boldsymbol{K}$ and $\boldsymbol{N}$ be sets of extended keys and null constraints. An *extended key-based inclusion dependency* over $\boldsymbol{E}$, $\boldsymbol{K}$ and $\boldsymbol{N}$, or an *EK-IND*, is an expression of one of the forms

1) $R_1[X_1] \subseteq R_2[X_2]:(\gamma,\delta)$
2) $R_1[Y_1 \neq \lambda][X_1] \subseteq R_2[X_2]:(\gamma,\delta)$,
3) $R_1[X_1] \subseteq R_2[Y_2 \neq \lambda][X_2]:(\gamma,\delta)$ or
4) $R_1[Y_1 \neq \lambda][X_1] \subseteq R_2[Y_2 \neq \lambda][X_2]:(\gamma,\delta)$

where, for $i = 1,2$,

- R_i is the name of a relation scheme in $\boldsymbol{E}$;
- Y_i is a sequence of distinct attributes of R_i such that $Y_2 \cap X_2 = \emptyset$;
- X_i is a sequence of distinct attributes of R_i such that X_1 and X_2 have the same length and X_2 is defined as a key of R_2 in $\boldsymbol{K}$;
- γ is the *insertion option*, taking values from the set $\{b^i,b^d,p^i,p^d\}$. However, γ can be equal to p^i or p^d only if the dependency is of the forms (1) or (2) and all attributes of R_2, excluding those in X_2, admit null values, that is, no null constraint in $\boldsymbol{N}$ says the contrary.
- δ is the *deletion option*, taking values from the set $\{b^i,b^d,p^i,p^d,n^i,n^d\}$, where n^i stands for "propagate immediately by nullifying" and n^d stands for "propagate deferredly by nullifying". However, δ can be equal to n^i or n^d only if the dependency is of the forms (2) and (4) and all attributes in Y_1 admit null values in R_1.

The restrictions imposed on γ again just avoid indeterminacies when propagating insertions and the restrictions on δ just reflect when consistency can be restored, after a deletion from R_2, by nullifying attributes values.

We say that the EK-IND is *simple on the right-hand side* in cases (1) and (2) and that it is *simple on the left-hand side* in cases (1) and (3). We also say that the EK-IND is *from* R_1 *to* R_2.

The semantics of EK-INDs is a direct extension of the semantics of K-INDs and the new deletion options have the following interpretation. Consider a EK-IND of the forms (2) or (4). If the deletion option δ is n^i, there is a trigger associated with the

EK-IND that is fired right after each deletion affecting R_2 terminates. Let o be one such deletion and suppose that σ is the state right after o terminates and that D is the set of tuples that o deletes. The trigger sets to null the Y_1-values of every tuple u in $\sigma(R_1[Y_1 \neq \lambda])$ such that there is a tuple t in D such that $u_{X_1} = t_{X_2}$ and $t_{X_2} \in \sigma(R_1[Y_1 \neq \lambda][X_1])$. With option n^d, we associate a trigger, fired when each transaction commits data, that sets to null the Y_1-values of u, for each tuple u in $\varphi(R_1[Y_1 \neq \lambda])$ such that $u_{X_1} \notin \varphi(R_2[X_2])$, where φ is the commit state of the transaction.

The optimization method will work with more complex schemas, defined as follows. An *extended relational schema* is a pair $SS = (IS, VS)$ where

- $IS = (BS, VD, IC)$, the *internal component* of SS, is such that
  - BS is a set of relation schemes with distinct names, called *base schemes* of SS;
  - VD is a set of view definitions over BS whose schemes have distinct names;
  - IC is a set of integrity constraints over the schemes in BS, called *active constraints* of SS, such that every key in IC is minimal.
- $VS = (E, I)$, the *visible component* of SS, is a relational schema such that $E \subseteq BS \cup e(VD)$, where $e(VD)$ is the set of relation schemes introduced in view definitions in VD.

An extended relational schema $SS = ((BS, VD, IC), (E, I))$ is *correct* iff:

- every consistent state of (BS, IC) induces, via VD, a consistent state of (E, I);
- the deletion and insertion options qualifying the inclusion dependencies in IC guarantee, again via VD, the deletion and insertion options qualifying the inclusion dependencies in I.

3.2 Description of the Transformation

This section describes a structural transformation for relational schemas, in first-normal form, that minimizes the number of EK-INDs. The transformation is called *trivialization.*

Let $SS = ((BS, VD, IC), (E, I))$ be an extended schema. An active EK-IND d in IC is *trivializable* iff d is of the form $S[L] \subseteq R[K]:(\gamma,\delta)$ or of the form $S[L] \subseteq R[Z \neq \lambda][K]:(\gamma,\delta)$ where

1) S is a base scheme of SS with key L that has an attribute N that does not occur in L and does not admit null values;
2) R is a base scheme of SS with key K and Z is a list of attributes of R;
3) $\gamma, \delta \in \{b^i, p^i\}$;
4) there is no active EK-IND in IC from a scheme T to S such that the insertion option is p^i or p^d.

Condition (2) is a direct consequence of the definition of EK-IND and is repeated here just to highlight that K must be a key of R. Condition (3) indicates that the insertion and deletion options of d are either b^i or p^i, that is, they must be both immediate, but it does not require that the options be equal.

Conditions (1) and (2) permit, via the keys K and L, to collapse schemes S and R into a single scheme G without creating redundancies. Each tuple t in G always encodes exactly one tuple of R and, if $t_N \neq \lambda$, also encodes exactly one tuple of S. Condition (3) clarifies which are the insertion and deletion options compatible with the creation of G. Without any of these three conditions, collapsing R and S into G would produce a new relational schema that does not preserve the semantics of the original schema. Finally, condition (4) reflects the restrictions imposed on the insertion options that may qualify a EK-IND. Without it, step (7) of the trivialization process, described in what follows, would produce an invalid insertion option.

The trivialization algorithm takes as input a relational schema SS_0 and produces as output an extended schema SS_n. Each step of the algorithm transforms an extended schema SS_i into a new extended schema SS_{i+1} by trivializing an EK-IND d_i of SS_i as follows. Suppose that d_i is of the form $S[L] \subseteq R[K]:(\gamma,\delta)$ or of the form $S[L] \subseteq R[Z \neq \lambda][K]:(\gamma,\delta)$, where R[K,X] and S[L,Y] are base schemes of SS_i. Suppose that N is an attribute of S that does not belong to L and that does not admit null values. The new extended schema SS_{i+1} is obtained from SS_i through the following transformations:

1) Remove d_i from the set of active integrity constraints;
2) Add G[K,X,Y′] to the set of base schemes, where Y′ is a renaming of the attributes in Y to avoid conflicts with the attributes of K and X (we will denote by A′ the attribute of Y′ corresponding to an attribute A of Y).

3) Remove R[K,X] from the set of base schemes transforming it into a view by adding to the set of view definitions the triple (R[K,X],G[K,X],**T**), where **T** defines the translation of the operations over R to G as follows:

Operation	*Translation*	*Note*
insert into R (t)	insert into G ($K=t_K, X=t_X, Y'=\lambda$)	
delete from R where Q	delete from G where Q	(1)
delete from R where Q	delete from G where Q and $N'=\lambda$	(2,3)
update R set X=t where Q	update G set X=t where Q	

Notes:

(1) translation when $\delta = p^i$, that is, when deletions from R propagate immediately;

(2) translation when $\delta = b^i$, that is, when deletions from R block immediately;

(3) $N'=\lambda$ indicates that a tuple of G does not encode a tuple of S.

4) Remove S[L,Y] from the set of base schemes transforming it into a view by adding to the set of view definitions the triple (S[L,Y],G[$N'\neq\lambda$][K,Y'],**U**) where **U** is defined as:

Operation	*Translation*	*Note*
insert into S ($L=u_L, Y=u_Y$)	insert into G ($K=u_L, Y'=u_Y, X=\lambda$) *if* insert fails *then* update G set $Y'=u_Y$ where $K=u_L$ and $N'=\lambda$	(1)
insert into S ($L=u_L, Y=u_Y$)	update G set $Y'=u_Y$ where $K=u_L$ and $N'=\lambda$	(2)
insert into S ($L=u_L, Y=u_Y$)	update G set $Y'=u_Y$ where $K=u_L$ and $N'=\lambda$ and $Z\neq\lambda$	(3)
delete from S where Q	update G set $Y'=\lambda$ where Q'	(4)
update S set $Y=u_Y$ where Q	update G set $Y'=u_Y$ where Q' and $N'\neq\lambda$	(5)

Notes:

(1) translation when $\gamma = p^i$, that is, when insertions into S propagate immediately. Note that, in this case, the trivialized EK-IND can only be of the form $S[L]\subseteq R[K]:(\gamma,\delta)$ because γ specifies propagation;

(2) translation when $\gamma = b^i$, that is, when insertions into S block immediately, and the EK-IND is of the form $S[L]\subseteq R[K]:(\gamma,\delta)$;

(3) translation when $\gamma = b^i$ and the EK-IND is of the form $S[L]\subseteq R[Z\neq\lambda][K]:(\gamma,\delta)$;

(4) Q' is a renaming of Q reflecting the renaming of Y to Y';

(5) the term $N'\neq\lambda$ avoids that the update becomes an insertion.

5) In each view defining expression, except those of R and S, replace R and S by their defining expressions, simplifying the result if possible.

6) In each translation of a view operation, except those associated with R and S, apply the translations specified in **T** and **U** to each operation over R and S.

7) Define K as a key of G. Each active integrity constraint C of $\boldsymbol{SS}_i$, except the keys K of R and L of S, generates an active integrity constraint C' of $\boldsymbol{SS}_{i+1}$ obtained by

   a) replacing R in C by its defining expression;
   b) replacing S in C by its defining expression and renaming the attributes of S to their new names in G;
   c) simplifying the result.

   Moreover, if C is a EK-IND, the deletion and insertion options of C' remain the same as those of C, except if C is of the form $S[X_1]\subseteq P[X_2]:(\gamma,\delta)$ and the deletion option δ is p^i or p^d, in which case the deletion option of C' becomes n^i or n^d, respectively.

8) Discard the view definitions whose schemes do not belong to the visible part.

The algorithm is correct in the sense that the visible component of the final extended schema $\boldsymbol{SS}_n$ is exactly the original relational schema $\boldsymbol{SS}_0$ and that $\boldsymbol{SS}_n$ is correct (as defined at the end of section 3.1). In other words, the optimization is transparent to users since they will still see the original schema $\boldsymbol{SS}_0$ and, moreover, the semantics derived from that of the internal component and from the view definitions of

SS_n is equivalent to the semantics originally specified. The correctness of the algorithm follows by induction on the number of iterations observing that, in each step, the dependencies over R and S, which are now views over G, become consequences of the dependencies over G. Indeed, by defining K as a key of G in step (2), we guarantee that K and L are keys of R and S and, through the generic transformation defined in step (7), we automatically map all null constraints, keys and EK-INDs involving R and S to equivalent dependencies over G.

As an example of the algorithm, consider a relational schema containing the relation schemes

P[NAME,DESC] S[NAME,SPEC]
I[NAME,CODE] C[E#]
M[NAME,E#]

standing for 'planned module', 'specified module', 'implemented module', 'chief-programmer' and 'manages', respectively, where keys are underlined and all attributes do not admit null values. Suppose also that the schema contains the following INDs:

(d^0_1) S[NAME]⊆P[NAME]: (b^i,p^i)

(d^0_2) I[NAME]⊆S[NAME]: (b^i,p^i)

(d^0_3) M[NAME]⊆P[NAME]: (b^i,p^i)

(d^0_4) M[E#]⊆C[E#]: (b^i,p^i)

Intuitively, in terms of the entity-relationship model, these dependencies say that I is a specialization of S, S is a specialization of P and M represents a relationship between P and C, which is n-1 with P on the "n" side.

The algorithm will transform this schema into a final extended relational schema with just two relation schemes and one EK-IND, thus trivializing the maintenance of three EK-INDs. For example, the trivialization of d^0_2 will produce one new scheme, SI[NAME,SPEC,CODE], and will transform S and I into views with defining expressions:

S[NAME,SPEC] = SI[NAME,SPEC]
I[NAME,CODE] = SI[CODE ≠ λ][NAME,CODE].

Furthermore, this trivialization will modified all constraints involving S and I. For example, d^0_1 becomes

SI[NAME]⊆P[NAME]: (b^i,p^i)

To conclude this section, we remark that the trivialization algorithm may be reinterpreted according to two opposite views. On one hand, if we understand a normalization process in the broad sense of a process that simplifies the treatment of dependencies by structural transformations, then the trivialization algorithm can be viewed as a normalization process. Indeed, define that an extended relational schema $SS_n = ((BS_n, VD_n, IC_n), (E,I))$ is in *extended key-based inclusion dependency normal form* (*EK-IND/NF*) iff IC_n has no trivializable EK-IND. We can then reinterpret the trivialization algorithm as a process to transform a given relational schema $S = (E,I)$ into an extended relational schema in EK-IND/NF. On the other hand, if we understand a denormalization process as any process that collapses into a single representation two logically separated concepts, then the trivialization algorithm can also be viewed as a denormalization process.

4. EXTENDING THE STRUCTURAL TRANSFORMATION TO THE NF2 RELATIONAL MODEL

This section briefly discusses, using an example, how to extend the structural transformation to NF^2 relational schemas. Without defining them explicitly, we shall use the basic notation of [Ja] and the counterpart for the NF^2 model of the concepts introduced in section 3. In particular, to indicate the operation that unnests an NF^2 relation, we shall use the symbol μ, subscripted with an expression E defining the part to be unnested. The unnest operation may be recursively applied to the result of other unnest operations. Note that μ automatically filters out the tuples whose unnested component is the empty set.

Let SS be an extended NF^2 relational schema. A EK-IND d in SS is *NF^2-trivializable* iff it is of the form S[L]⊆R[K] or of the form S[L]⊆μ_E(R)[K] where

1) S is a base scheme of SS with at least one attribute N not in L;
2,3,4) Identical to conditions (2),(3) and (4) of trivialization in the traditional model.

Note that, in the NF^2 model, L need not be a key of S and N may admit null values. The trivialization of d nests S into R, creating a single scheme G. Each tuple t of G always encodes the tuple of R with key t_K and the set of tuples u of S such that $u_L = t_K$. If this set is empty, the projection of t over the nested attributes of S will be simply the empty set, and not a tuple of nulls, as in the traditional model.

The trivialization algorithm can be extended to NF^2 relational schemas as the following example suggests. Let us consider a relational schema containing just

two relation schemes:

E(SSN,NAME,DESC)
D(SSN,DNAME)

where E and D stands for 'employee' and 'dependent', respectively, and where the keys are underlined. Suppose also that the schema has one K-IND,

(d^0_1) D[SSN] ⊆ E[SSN]: (b^i,p^i)

Then, we may trivialize d^0_1 producing a single NF^2 relation scheme:

ED(SSN,NAME,DESC,D(DNAME))

and transforming E and D into views with defining expressions:

E(SSN,NAME,DESC) = ED[SSN,NAME,DESC]

D(SSN,DNAME) =
$(\mu_{D;DNAME}(ED))$[SSN,DNAME]

In a general setting, it is convenient to carefully analyze the benefits of nesting a scheme S into another scheme when there are EK-INDs involving S that will not be trivialized because they will be transformed into EK-INDs over relations defined with the help of the unnest operator.

As for the traditional model, the definition of trivializable dependencies induces a normal form for the NF^2 model and the trivialization algorithm corresponds to a normalization process.

5. CONCLUSIONS

The optimization process described in this paper minimizes the set of K-INDs of a relational schema by eliminating redundant K-INDs and by trivializing EK-INDs without modifying the semantics of the application. As a conceptual design process, it differs from the strategies defined for the traditional as well as the NF^2 variations of the relational model because is based on inclusion dependencies and depends in a fundamental way on the insertion and deletion options to avoid altering the behavior of the transactions.

ACKNOWLEDGEMENT

We thank Prof. Claudia Bauzer Medeiros for the useful discussion concerning section 2.

REFERENCES

[AMM] H. Arisawa, K. Moriya, T. Miura, "Operations And The Properties On Non-First-Normal-Form Relational Databases", Proc. of the 9th International Conference on Very Large Data Bases, (1983), pp. 197-204.

[CFP] M.A. Casanova, R. Fagin, C. Papadimitriou, "Inclusion Dependencies and Their Interaction with Functional Dependencies", J. of Computer and System Sciences, Vol. 28, No. 1 (Feb. 1984), pp. 29-59.

[CFT] M.A. Casanova, A.L. Furtado and L. Tucherman, "Enforcing Inclusion Dependencies and Referential Integrity", Proc. 14th International Conference on Very Large Data Bases, Los Angeles (Aug. 1988), pp. 13-25.

[CTFB] M.A. Casanova, L. Tucherman, A.L. Furtado and A.P. Braga, "Optimization of Relational Schemas containing Inclusion Dependencies", Technical Report CCR078, Rio Scientific Center, IBM Brazil, Rio de Janeiro, Brazil (June 1989).

[Da] C.J. Date, "Referential Integrity" in *Relational Database: Selected Writings* Addison-Wesley Publishing Company, (1986), pp. 41-62.

[FC] A.L. Furtado and M.A. Casanova, "Updating Relational Views", in *Query Processing in Database Systems*, W. Kim, D.S. Reiner and D.S. Batory (eds.), Springer Verlag (1985), pp. 127-142.

[FG] P. Fisher, D. Van Gucht, "Determining When A Structure Is A Nested Relation", Proc. of the 11th International Conference on Very Large Data Bases, (1985), pp. 171-180.

[Ja] G. Jaeschke, "Recursive Algebra for Relations with Relation Valued Attributes", TR-85.03.002, IBM Heildeberg Scientific Center (Mar. 1985).

[Ma] D. Maier, *The Theory of Relational Databases*, Computer Science Press (1983).

[OY] Z. M. Ozsoyoglu, L. Yuan, "A New Normal Form for Nested Relations", ACM Trans. on Database Systems, Vol.12, No.1 (March 1987), pp. 111-136.

[RKS] M. A. Roth, H. F. Korth, A. Silberschatz, "Theory of non-first-normal form relational databases", TR-84-36, Dept. of Computer Science, Univ. of Texas at Austin, (1984).