数理解析研究所講究録
第 495 巻 1983 年 186-220

VALIDATING DATABASE UPDATES

Isamu KOBAYASHI

SANNO Institute of Business Administration
School of Management and Informatics
1573 Kamaikasuya, Isehara, Kanagawa 259-11, Japan

Feburuary 18,1983

ABSTRACT: Individual facts in the real world are represented by tuples in database relations (instances), while universal (time-independent) facts are treated as semantic constraints regarding database relation schemata. One important role of these semantic constraints is their use as integrity constraints that must not be violated by update operations. Among these, static constraints are represented by assertions, which are extended relational calculi in which every tuple variable is bound over a relation, and become true in a consistent database. When an update has been made on a consistent database, it is necessary to ascertain if the updated database is still consistent. It can be done by evaluating all the assertions in the updated database, but this is very time-consuming. If a given assertion is in one of some classes, it is possible to devise an efficient validation procedure, which before the update is actually applied determines if the update violates the given assertion. In many cases a simplified form can be found, by examining whose value the properness of the given update is determined. The existence of such an efficient procedure and simplified form depends on what class the given assertion belongs, and also on what type of the update is to be made on what relation. This paper presents a method of finding such a procedure and simplified form using several simple syntactical transformation rules regarding extended relational calculi. This method is based on several basic properties in propositional logic and many-sorted predicate logic.

KEYWORDS AND PHRASES: assertion, aggregate function, consistency, database updates, extended relational calculus, integrity, validation

1 INTRODUCTION

The two major enhancements expected in future database systems are the integration of advanced integrity checking function and that of inference execution. These two are different uses of the same semantic constraints. In such an environment, it is very desirable to develop an integrity checking mechanism which is applicable to constraints of any type described in a standard form - probably in the form of predicate calculus. This paper first introduces the notion of extended relational calculus which is appropriate to represent integrity constraints of some type (static constraints).

A database is consistent when all assertions, which are extended relational calculi expressing integrity constraints, are true in it. When an update has been made on a consistent database, it is necessary to ascertainn that the updated database is still consistent. This can be achieved by evaluating all assertions in the updated database, but this is too much time-consuming in many cases.

If the given assertion is in one of some assertion classes, it is possible to devise an efficient procedure which before the database is actually updated determines if the update violates this assertion. In many cases a simplified form can be found, by examinig whose value (instead of the value of the given assertion) the properness of the update is determined.

The existence of such an efficient procedure and simplified form depends on what class the assertion belongs to, and also on what type of update is to be applied to what relation. Sometimes the simplified form becomes (constantly) true or false. Sometimes a simplified form acting a necessary and sufficient condition for the update being proper exists, but sometimes only a simplified form acting a necessary condition or a sufficient condition is found. Also there are the cases where no simplified forms exist.

This paper presents a method of finding such an efficient procedure and simplified form using several simple transformation rules in many-sorted predicate calculus.

2 SEMANTIC CONSTRAINTS ON DATABASE RELATIONS

The real world can be thought as consisting of various objects, and various facts which are some kinds of relationships among objects. These facts can be described by natural language sentences, which can in turn be transcribed into predicates of the form

$$p(a_1,a_2,\dots,a_n),$$

where each a_k is an individual constant corresponding to a real world object and p is a predicate symbol. Each predicate is given a truth value with regard to the real world state "at a certain time."

A database "relation" R_p^τ is defined by

$$R_p^\tau=\{(a_1,a_2,\dots,a_n)|p(a_1,a_2,\dots,a_n) \text{ is true at the time } \tau\}.$$

An element t in a relation is called a "tuple." The individual constant a_k is called the k-th "attribute value" of the tuple t, and denoted by

$$a_k=A_k(t).$$

Here A_k can be regarded as a function of tuples and called the k-th "attribute."

A family $\mathcal{R}_p(\{A_1,A_2,\dots,A_n\})$ of relations with a predicate symbol p and attributes $A_1,A_2,\dots,A_n$, each realizable at a certain time, is called a "relation schema."

What a tuple in a database relation can represent is an individual fact in the real world. It can be seen that there exist somewhat universal facts such as "a person is less than 120 years old" and "a father is male." Such universal facts are obtained inductively from a historical observation of the real world, or deductively from the basic properties of the real world. To express such universal facts, it is necessary to introduce (bound) variables and logical operators into the predicate. The above two facts can be represented by

$$(\forall z)((\exists x)(\exists y)\text{person}(x,y,z)\supset z<120)$$

and

$$(\forall x)((\exists w)\text{father}(x,w)\supset(\exists z)\text{person}(x,\text{'male'},z))$$

where person(x,y,z) is the predicate representing "x (name) is a person whose sex is y and age is z," z<120 is the predicate representing that "z is less than 120," and father(x,w) is the predicate representing "x (name) is w's (name) father."

If individual facts regarding predicate symbols person and father are represented by tuples in relations R^τ_{person} and R^τ_{father}, and if the truth value of a<b can be computed whenever two individual constants a and b are given, the above two facts can be represented by

$$(\forall\tau)(\forall t/R^{\tau}_{person})age(t)<120$$

and

$$(\forall\tau)(\forall t_1/R^{\tau}_{father})(\exists t_2/R^{\tau'}_{person})(father\_name(t_1)=name(t_2)\wedge sex(t_2)='male')$$

which are predicates regarding database relations. Predicates of this type are used throughout this paper. The necessity of time variables could be understood when one wishes to express some facts like "the age does not decrease," which is written as

$$(\forall\tau)(\forall\tau')(\tau<\tau'\supset (\forall t/R^{\tau}_{person})(\forall t'/R^{\tau'}_{person})(name(t)=name(t')\supset age(t)<age(t'))).$$

These universal facts are called "semantic constraints." This is because they can be regarded as representing some part of meanings behind individual facts, which are represented by database relations.

Note that predicates regarding time variables, if exist, must be explicitly described. That is, notations like R^{60n}_{p} and $R^{\tau+10}_{p}$ are not permitted. A semantic constraint is said to be "static" if it has just one universally quantified time variable and no restrictions (predicates) regarding this time variable. Otherwise it is said to be "dynamic."

Semantic constraints as well as individual facts can be regarded as axioms. Together with axioms in formal logic, these two types of facts constitute an axiom system. Using inference rules in formal logic, it is possible to infer some theorems from this axiom system. That is, some other (individual or universal) facts can be deduced.

In the database environment, semantic constraints may be registered somewhere in the database (probably in the conceptual schema) in a certain form. Some tuples or semantic constraints which are implicit in the database can be deduced from those explicit in the database using a certain built-in inference mechanism. One more important role of these semantic constraints in the database environment is their being used for validating update operations.

A database D^{τ} is said to be in a "consistent state" if all the registered static constraints hold in D^{τ}. That is, all relations R^{τ}_{k} in D^{τ} satisfy the static constraints. A database update $D^{\tau}\rightarrow D^{\tau'}$ is said to be "proper" if it transforms a consistent database D^{τ} into another consistent database $D^{\tau'}$ without violating the registered dynamic constraints.

In this paper, the author tries to find update validation procedures to preserve given "static" constraints in database relations. Although it is necessary to validate updates against dynamic constraints in order to assure the properness of database updates, validation procedures for dynamic constraints are left to future study. Let us say a database update is "S-proper" if it transforms a consistent database into another consistent database.

3 RELATION SCHEMA CALCULI REPRESENTING STATIC CONSTRAINTS

For expressing static constraints, it is better to introduce a little simpler notation. Since every static constraint has a single universally quantified time variable, it is possible to omit the time variable description. Instead, it is appropriate to replace the R_p^τ notation in the quantification of tuple variables by the name R_p of the relation schema to which R_p^τ belongs. The two examples of static constraints shown previously can then be written as

$(\forall t/R_{person})age(t)<120$

and

$(\forall t_1/R_{father})(\exists t_2/R_{person})(\ father\_name(t_1)=name(t_2)\wedge sex(t_2)='male')$.

This simplification is reasonable because a static constraint is a statement regarding some relation schema rather than regarding relations (instances at a specific time). Let us call this form the "relation schema calculus." It must be distinguished from the relational calculus because the latter is a statement regarding relations (instances at a specific time). That is, the value of relational calculus is time-dependent in contrast to the relation schema calculus whose value is time-independent. One more major difference is that the relational calculus can contain free tuple variables, which the relation schema calculus cannot contain.

The relation schema calculus is a many-sorted logic, whose sorts are relation schemata. Its variables are tuple variables bound over some relation schemata. It contains functions of tuples. Formal definition of the relational schema calculus is as follows.

First it is necessary to provide an enumerably infinite number of tuple variables $t_1,t_2,\ldots,$ relation schema symbols $R_1,R_2,\ldots,$ attribute symbols $A_1,A_2,\ldots,$ value sets $V_1,V_2,\ldots,$ and individual constants. Also assume that each attribute is associated to a value sets, and each individual constant is associated to the value set to which it belongs.

(1) Domain terms: For a tuple variable t and a relation schema symbol R, the form t/R is called a domain term.

(2) Domain-coupled quantifiers: If Γ is a domain term, then the form $(\varepsilon\Gamma)$ is called a domain-coupled quantifier, where ε is either $\exists$ or $\forall$.

(3) FOTs (function of tuples): Following three are FOTs.

3-1 An individual constant is an FOT with any number (can be zero) of tuple variables. There must be two special individual constants 'true' and 'false'.

3-2 An attribute A is an FOT with one tuple variable t. This FOT is denoted by A(t).

3-3 Let $V_1, V_2, \ldots, V_m$ and V_ϕ be m+1 value sets where a function (operator)

$\phi: V_1 \times V_2 \times \ldots \times V_m \to V_\phi$

is defined. If f_k is an FOT with m(k) tuple variables $t_{k1}, t_{k2}, \ldots, t_{km(k)}$ whose range is V_k, then

$\phi(f_1, f_2, \ldots, f_m)$

is an FOT. Its range is V_ϕ. As a special case, k(m) can be 0, that is, $\{t_{k1}, t_{k2}, \ldots, t_{km(k)}\}$ can be empty when f_k is an individual constant. The sets of tuple variables of FOTs $f_1, f_2, \ldots, f_m$ are not necessarily mutually disjoint. Tuple variables of the function constructed above are those in

$\bigcup_{k=1}^{m}\{t_{k1}, t_{k2}, \ldots, t_{km(k)}\}$.

This definition is very general. It is possible to use any function (operator defined in various value sets. For example, it is possible to use unary and binary arithemetic operators to combine arithemetic FOTs, which are FOTS whos range is a arithmetic set. In combining string FOTs, various string operators can be used.

An FOT is called a logical FOT if its range is the truth value set {'true','false'}. Relational operators can be used to construct a logical FOT by combining two FOTs with a common range. One more operator used to construct a logical FOT is ϵV. The FOT $f \epsilon V$ becomes a logical FOT. Logical operators can be used to construct another logical FOT from one or two logical FOTs. However, these are left to a little more expanded definition of logical expressions shown below.

(4) LEXs (logical expressions): Following three are LEXs.

4-1 An FOT (defined by (3)) is an LEX if its range is the truth value set.

4-2 If both f and g are LEXs, then all $f\wedge g$, $f\vee g$, $f\supset g$ and $\sim f$ are LEXs.

4-3 If f is an LEX and $(\varepsilon t/R)$ is a domain-coupled quantifier, then $(\varepsilon t/R)f$ is an LEX. The LEX f is called the scope of the domain-coupled quantifier. The tuple variable t is said to be quantified over R in the scope f.

One more type of FOTs can be added to the definition (3).

3-4 Let ∇ be an aggregate operator defined in a value set V and g be an FOT with tuple variables $t_1,t_2,\dots,t_p,t_{p+1},\dots,t_q$ whose range is V. Let f be an LEX with tuple variable $t_1,t_2,\dots,t_p,t_{q+1},\dots,t_r$ among which $t_1,t_2,\dots,t_p$ are not quantified in f. Finally let $t_1/R_1,t_2/R_2,\dots,t_p/R_p$ be domain terms for tuple variables $t_1,t_2,\dots,t_p$. Then

$$\nabla[t_1/R_1,t_2/R_2,\dots,t_p/R_p;f]g$$

is an FOT. Tuple variables t_k $(1\le k\le p)$ are said to be bound over R_k by the aggregate operator ∇. Here $\{t_{p+1},\dots,t_q\}$ and $\{t_{q+1},\dots,t_r\}$ are not necessarily disjoint. Tuple varaibles of this aggregate function are those in $\{t_{p+1},\dots,t_q\}\cup\{t_{q+1},\dots,t_r\}$, which are said to be free with respect to ∇.

The interpretation of aggregate function defined above is that values of function g are aggregated according to the aggregate operator ∇ for the ordered sets $(t_1,t_2,\dots,t_p)$ for which the LEX f becomes 'true', where each t_k belongs to the relation R_k in R_k. If t/R is regarded as an LEX whose value is 'true' when t belongs to the relation R in R, the condition of aggregation can be written as

$$t_1/R_1\wedge t_2/R_2\wedge\dots\wedge t_p/R_p\wedge f.$$

(It is a somewaht extended relational calculus.) By this means it becomes possible to use aggregate operators such as Σ, Π, max, min, average, standard deviation and so forth.

Aggregate operators $\bigwedge$ and $\bigvee$ which correspond to $\wedge$ and $\vee$ in the truth value set can be considered just like Σ corresponds to +. Then it is possible to construct an aggregate function

$$\nabla[t/R;f]g$$

where ∇ is $\bigwedge$ or $\bigvee$. Unlike other aggregate functions, both f and g are

LEXs in this function. If ∇ is $\bigwedge$, then this function is equivalent to

$(\forall t/R)f\supset g$,

while if ∇ is $\bigvee$, then this function is equivalent to

$(\exists t/R)f\wedge g$.

Conversely, the domain-coupled quantifier $(\varepsilon t/R)g$ can be rewritten as

$\nabla[t/R;]g$

where the default of the function f specifies that f is a constant (function) 'true'. This gives an interpretation of domain-coupled qunatifiers. Obviously quantified variables are bound variables.

(5) Relation schema calculus: A relation schema calculus is an LEX in which no free variables exists, and every attribute A(t) appearing in it is an attribute defined for the relation schema over which t is bound.

4 ASSERTIONS AS EXTENDED RELATIONAL CALCULI

One universal method of validating database updates is to use a query evaluation procedure that processes a given extended relational calculus. Definition of the "extended relational calculus" is obtained by applying some modifications to that of the relation schema calculus.

First it is necessary to provide an enumerably infinite number of relation symbols $R_1, R_2, \ldots$ instead of relation schema symbols. Definition of domain terms must be

(1) Domain terms: For a tuple t and a relation symbol R, the form t/R is a domain term.

Definition of domain-coupled quantifiers and FOTs are left unchanged. To the definition of LEX,

4-4 A domain term is an LEX.

must be added. Finally the extended relational calculus is defined by

(5) Extended relational calculus: An extended relational calculus is an LEX in which every free tuple variable has one corresponding domain term in it, every bound tuple variable is bound by one aggregate function (can be quantifier), and every attribute A(t) is defined in the relation in which the tuple variable t moves.

In particular, it is possible to obtain an extended relational calculus without free variables by replacing each relation schema symbol in a relation schema calculus representing a static constraint by a relation symbol. Such an extended relational calculus is called an "assertion." To examine if a database is consistent, it is sufficient to show that all assertions become 'true' in this database.

In general, when an extended relational calculus with m free tuple variables is given, it is possible to extract the set of ordered sets of m tuples qualified for this extended relational calculus from the database. If the given relational calculus has no free tuple variables, the answer is 'true' or 'false'. An optimal algorithm of evaluating queries based on the extended relational calculus was presented in [1]. Such an algorithm can be applied to examining if a database is consistent with respect to a set of static constraints.

If an update has been applied to a consistent database, and if the updated database is still consistent, then this update is S-proper. One can assure an assertion being 'true' in the updated database by evaluating its value using a universal query evaluation algorithm. If at least one assertion has been found to be 'false' in the updated database, the update is not S-proper, and the database must be restored.

Obviously, however, this validation procedure is unnecessarily time-consuming in many cases. First it is not necessary to examine any assertion whose value is not affected by the given update. Then what assertion would be violated by the update? Let us first consider only the "unit updates" each adds, deletes or replaces just one tuple in a database relation. An assertion is said to be "relevant" to a unit update if it contains one or more tuple variables bound over the relation to be updated. Only the assertions relevant to the given update could be violated by this update. Tuple variables bound over the relation to be updated are called "update-relevant" variables.

There are various types of assertions. Some contains only one tuple variable, while others contain two or more tuple variables. In the latter case, tuple variables can be all bound over a single relation or over different relations. Tuple variables can be universally quantified, existentially quantified or bound by some other aggregate operators.

Assertions can be classified according to these characteristics [2]. Some assertion classes can be examined if they would be violated by an update before this update being actually applied to the database. Let us next consider if a validation procedure which is more efficient than eveluating assertions in the updated database exists for a given assertion class. Assume that all assertions are written in prenex form.

5 ASSERTIONS WITH ONE UPDATE-RELEVANT VARIABLES

First several basic properties of domain-coupled quantifiers must be examined. The following two lemmas are used in proving all theorems.

LEMMA 1 (Substitution rule): Let Q be an LEX without domain terms for a tuple variable t, and $\bar{t}$ be a tuple (constant), then

$$(t/\{\bar{t}\}\supset Q)\equiv(t/\{\bar{t}\}\wedge Q)\equiv Q[\bar{t}\to t]$$

where $Q[\bar{t}\to t]$ is the LEX obtained by substituting $\bar{t}$ for all instances of t in Q.

This is directly deduced from the interpretation of domain terms. This lemma can be generalized as follows:

LEMMA 1': Let Q be an LEX without domain terms for a tuple variable t_n, and let $\bar{t}$ be a tuple (constant), then

$$t_n/\{\bar{t}\}\supset Q[\bar{t}\to t_1,t_2,\ldots,t_{n-1}]\equiv t_n/\{\bar{t}\}\wedge Q[\bar{t}\to t_1,t_2,\ldots,t_{n-1}]$$
$$\equiv Q[\bar{t}\to t_1,t_2,\ldots,t_n]$$

where $Q[\bar{t}\to t_1,t_2,\ldots,t_n]$ is a LEX obtained by substituting $\bar{t}$ for all instances of $t_1,t_2,\ldots,t_n$ in Q.

The next lemma is in many-sorted logic.

LEMMA 2: Let R^+ be the relation obtained by adding a tuple $\bar{t}$ to a relation R, that is, $R^+=R\cup\{\bar{t}\}$. Then

(1) $(^\forall t/R^+)Q\equiv(^\forall t/R)Q\wedge Q[\bar{t}\to t]$

(2) $(^\exists t/R^+)Q\equiv(^\exists t/R)Q\vee Q[\bar{t}\to t]$

PROOF: The proof is straightforward.

(1) $(^\forall t/R^+)Q\equiv(^\forall t)(t/R^+\supset Q)\equiv(^\forall t)(\sim t/R^+\vee Q)\equiv(^\forall t)(\sim(t/R\vee t/\{\bar{t}\})\vee Q)$
$\equiv(^\forall t)((\sim t/R\wedge\sim t/\{\bar{t}\})\vee Q)\equiv(^\forall t)((\sim t/R\vee Q)\wedge(\sim t/\{\bar{t}\}\vee Q))$
$\equiv(^\forall t)(t/R\supset Q)\wedge(t/\{\bar{t}\}\supset Q))\equiv(^\forall t)((t/R\supset Q)\wedge Q[\bar{t}\to t])$
$\equiv(^\forall t)(t/R\supset Q)\wedge Q[\bar{t}\to t]\equiv(^\forall t/R)Q\wedge Q[\bar{t}\to t]$

(2) $(^\exists t/R^+)Q\equiv(^\exists t)(t/R^+\wedge Q)\equiv(^\exists t)((t/R\vee t/\{\bar{t}\})\wedge Q)$
$\equiv(^\exists t)((t/R\wedge Q)\vee(t/\{\bar{t}\}\wedge Q))\equiv(^\exists t)((t/R\wedge Q)\vee Q[\bar{t}\to t])$
$\equiv(^\exists t)(t/R\wedge Q)\vee Q[\bar{t}\to t]\equiv(^\exists t/R)Q\vee Q[\bar{t}\to t]$

Let t be only one update-relevant variable in the given assertion P. Let us denote $R\cup\{\bar{t}\}$ by R^{+}, $R-\{\bar{t}'\}$ by R^{-}, and $R-\{\bar{t}'\}\cup\{\bar{t}\}$ by R^{*}.

THEOREM ∀: If the assertion is of the form

$P\equiv(\forall t/R)Q$

and Q contains no other update-relevant variables than t, then

(1) Adding a tuple $\bar{t}$ to R is proper with respect to P if and only if $Q[\bar{t}\rightarrow t]$ is 'true' in the current database.

(2) Deleting a tuple $\bar{t}'$ from R is always proper with respect to P.

(3) Replacing a tuple $\bar{t}'$ in R by $\bar{t}$ is proper with respect to P if and only if $Q[\bar{t}\rightarrow t]$ is 'true' in the current database.

The following lemma in propositional logic must be used in proving this theorem.

LEMMA 3∧: Let P, P_1 and P_2 are propositions, then

(1) $(P_1\equiv P\wedge P_2)\supset(P\supset(P_1\equiv P_2))$

(2) $(P\supset(P_1\wedge P_2))\supset(P\supset P_1)$

(3) $((P\supset(P_1'\wedge P_2'))\wedge(P_1\equiv P_1'\wedge P_2))\supset(P\supset(P_1\equiv P_2))$

PROOF: Cases (1) and (2) should be obvious.

(3) $((P\supset(P_1'\wedge P_2'))\wedge(P_1\equiv P_1'\wedge P_2))\supset((P\supset P_1')\wedge(P_1'\supset(P_1\equiv P_2))\supset(P\supset(P_1\equiv P_2))$ □

PROOF of theorem ∀: From lemma 2,

$(\forall t/R^{+})Q\equiv(\forall t/R)Q\wedge Q[\bar{t}\rightarrow t]$,

$(\forall t/R)Q\equiv(\forall t/R^{-})Q\wedge Q[\bar{t}'\rightarrow t]$

and

$(\forall t/R^{*})Q\equiv(\forall t/R^{-})Q\wedge Q[\bar{t}\rightarrow t]$.

Since $(\forall t/R)Q$ is 'true' in the current (consistent) database, it is possible to apply lemma 3∧ (1), (2) and (3) for adding, deleting and replacing a tuple, respectively. □

THEOREM ∃: If the assertion is of the form

$P\equiv(\exists t/R)Q$

and Q contains no other update-relevant varaibles than t, then

(1) Adding a tuple $\bar{t}$ to R is always proper with respect to P.

(2) Deleting a tuple $\bar{t}'$ from R is proper with respect to P if $Q[\bar{t}'\rightarrow t]$ is 'false'. (If $Q[\bar{t}'\rightarrow t]$ is 'true'. it can be either proper or improper with respect to P.)

(3) Replacing a tuple $\bar{t}'$ in R by $\bar{t}$ is proper with respect to P if $Q[\bar{t}'\rightarrow t]$ is 'false' or $Q[\bar{t}\rightarrow t]$ is 'true'. Otherwise, it is proper

with respect to P if and only if $(\exists t/R^{-})Q$ is evaluated to be 'true'.

The following lemma in propositional logic must be used in proving this theorem.

LEMMA 3v: Let P, P_1 and P_2 be propositions. Then

(1) $((P \vee P_2) \supset P_1) \supset (P \supset P_1)$

(2) $(P \supset (P_1 \vee P_2)) \supset (P \supset (\sim P_2 \supset P_1))$

(3) $((P \supset (P_1' \vee P_2')) \wedge (P_1 \equiv P_1' \vee P_2)) \supset ((P \supset (\sim P_2' \supset P_1)) \wedge (P_2 \supset P_1) \wedge (\sim P_2 \supset (P_1 \equiv P_1')))$

PROOF: Cases (1) and (2) should be obvious.

(3) $((P \supset (P_1' \vee P_2')) \wedge (P_1 \equiv P_1' \vee P_2))$

$\supset ((P \supset (\sim P_2' \supset P_1')) \wedge (P_1' \supset P_1) \wedge (P_2 \supset P_1) \wedge (\sim P_2 \supset (P_1 \equiv P_1')))$

$\supset ((P \supset ((\sim P_2' \supset P_1') \wedge (P_1' \supset P_1))) \wedge (P_2 \supset P_1) \wedge (\sim P_2 \supset (P_1 \equiv P_1')))$

$\supset ((P \supset (\sim P_2' \supset P_1)) \wedge (P_2 \supset P_1) \wedge (\sim P_2 \supset (P_1 \equiv P_1')))$ □

PROOF of theorem $\exists$: From lemma 2,

$(\exists t/R^{+})Q \equiv (\exists t/R)Q \vee Q[\bar{t} \to t]$,

$(\exists t/R)Q \equiv (\exists t/R^{-})Q \vee Q[\bar{t}' \to t]$

and

$(\exists t/R^{*})Q \equiv (\exists t/R^{-})Q \vee Q[\bar{t} \to t]$.

Since $(\exists t/R)Q$ is 'true' in the current (consistent) database, it is possible to apply lemmas 3v (1), (2) and (3) for adding, deleting and replacing a tuple, respectively. □

This theorem shows that if the update involves deleting a tuple and $Q[\bar{t}' \to t]$ is 'true', or if it involves replacing a tuple, $Q[\bar{t}' \to t]$ is 'true' and $Q[\bar{t} \to t]$ is 'false', then the $(\exists t/R^{-})Q$ value must be evaluated directly. However, it can be noted that the $(\exists t/R^{-})Q$ value can be evaluated before the update is actually applied to R, for it can be transformed into

$(\exists t/R)(A_p(t) \neq A_p(\bar{t}) \wedge Q)$

where A_p is the concatenation of attributes constituting a candidate key.

In the above two theorems, Q may contain any number of update-irrelevant variables. Let Π be a series of domain-coupled quantifiers for update-irrelevant variables. If all quantifiers in Π are universal, then

$\Pi(\forall t/R)Q \equiv (\forall t/R)\Pi Q$,

and the theorem $\forall$ is still valid. If all quantifiers in Π are existential, then

$\Pi(\exists t/R)Q \equiv (\exists t/R)\Pi Q$,

and the theorem $\exists$ is still valid.

The following lemma is necessary for the further discussion.

LEMMA 4: Let Π be a series of domain-coupled quantifiers. and Q_1 and Q_2 be LEXs. Then

(1) $\Pi(Q_1 \wedge Q_2) \supset \Pi Q_1 \wedge \Pi Q_2$

(2) $\Pi Q_1 \vee \Pi Q_2 \supset \Pi(Q_1 \vee Q_2)$

PROOF: These two are obtained by combining the following four properties in (many-sorted) predicate logic:

(1) $(\forall t/R)Q_1 \wedge (\forall t/R)Q_2 \equiv (\forall t/R)(Q_1 \wedge Q_2)$

(2) $(\forall t/R)Q_1 \vee (\forall t/R)Q_2 \supset (\forall t/R)(Q_1 \vee Q_2)$

(3) $(\exists t/R)(Q_1 \wedge Q_2) \supset (\exists t/R)Q_1 \wedge (\exists t/R)Q_2$

(4) $(\exists t/R)(Q_1 \vee Q_2) \equiv (\exists t/R)Q_1 \vee (\exists t/R)Q_2$

THEOREM $\Pi\forall$: If the assertion is of the form

$$P \equiv \Pi(\forall t/R)Q$$

and Q contains no other update-relevant variables than t, where Π is a series of domain-coupled quantifiers for update-irrelevant variables, of which the last one is existentoal, then

(1) Adding a tuple $\bar{t}$ to R is improper with respect to P if $\Pi Q[\bar{t} \to t]$ is 'false'.

(2) Deleting a tuple $\bar{t}'$ from R is always proper with respect to P.

(3) Replacing a tuple $\bar{t}'$ by $\bar{t}$ is improper with respect to P if $\Pi Q[\bar{t} \to t]$ is 'false'.

One more lemma in propositional logic is necessary.

LEMMA 5: Let P_1, P_2 and P_3 be propositions. Then

$$(P_1 \supset (P_2 \wedge P_3)) \supset ((\sim P_2 \supset \sim P_1) \wedge (\sim P_3 \supset \sim P_1)).$$

PROOF of theorem $\Pi\forall$: From lemmas 3 and 4,

$$\Pi(\forall t/R^+)Q \equiv \Pi((\forall t/R)Q \wedge Q[\bar{t} \to t]) \supset \Pi(\forall t/R)Q \wedge \Pi Q[\bar{t} \to \bar{t}],$$

$$\Pi(\forall t/R)Q \equiv \Pi((\forall t/R^-)Q \wedge Q[\bar{t}' \to t]) \supset \Pi(\forall t/R^-)Q \wedge \Pi Q[\bar{t}' \to t]$$

and

$$\Pi(\forall t/R^*)Q \equiv \Pi((\forall t/R^-)Q \wedge Q[\bar{t} \to t]) \supset \Pi(\forall t/R^-)Q \wedge \Pi Q[\bar{t} \to t].$$

Cases (1) and (3) are obtained using lemma 5, while (2) using lemma 3$\wedge$. □

Here (1) and (3) give only a sufficient condition for adding and replacing a tuple being improper with respect to P. There are no other cases where $\Pi(\forall t/R^+)Q$ or $\Pi(\forall t/R^*)Q$ can be evaluated indirectly except in very special cases such as $Q[\bar{t} \to t]$ becomes a tautology (where adding and replacing a tuple are always proper with respect to P).

THEOREM $\Pi\exists$: If the assertion is of the form

$P\equiv\Pi(\exists t/R)Q$

and Q contains no update-relevant varaibles than t, where Π is a series of domain-coupled quantifiers for update-irrelevant variables, of which the last one is universal, then

(1) Adding a tuple $\bar{t}$ to R is always proper with respect to P.

(2) Deleting a tuple $\bar{t}'$ by $\bar{t}$ is proper with respect to P if $\Pi^u\sim Q[\bar{t}'\to t]$ is 'true' or, equivalently, $\Pi^e Q[\bar{t}'\to t]$ is 'false', where Π^u is a series of domain-coupled quantifiers obtained by replacing every existential quantifier in Π by a universal quantifier coupled with the same domain, while Π^e is that obtained by replacing every universal quantifier in Π by an existential quantifier coupled with the same domain.

(3) Replacing a tuple $\bar{t}'$ by $\bar{t}$ is proper with respect to P if $\Pi(\exists t/R^-)Q$ or $\Pi Q[\bar{t}\to t]$ is 'true'. The former is 'true' if $\Pi^u\sim Q[\bar{t}'\to t]$ is 'false'.

To prove this, it is necessary to introduce an additional lemma in (many-sorted) predicate logic.

LEMMA 6: Let P be a proposition, Π be a series of domain-coupled quantifiers, and Q_1 and Q_2 be LEXs. Then

$$(P\supset\Pi(Q_1\vee Q_2))\supset(P\supset(\Pi^u\sim Q_2\supset\Pi Q_1)).$$

PROOF: It will be easy to see

$$(P\supset(\forall t/R)(Q_1\vee Q_2))\supset(P\supset((\forall t/R)\sim Q_2\supset(\forall t/R)Q_1))$$

and

$$(P\supset(\exists t/R)(Q_1\vee Q_2))\supset(P\supset((\forall t/R)\sim Q_2\supset(\exists t/R)Q_1)).$$

The lemma can be obtained by combining the above two. $\boxed{}$

PROOF of theorem $\Pi\exists$: From lemmas 2 and 4,

$$\Pi(\exists t/R^+)Q\equiv\Pi((\exists t/R)Q\vee Q[\bar{t}\to t])\subset\Pi(\exists t/R)Q\vee\Pi Q[\bar{t}\to t].$$

Case (1) is obtained by applying lemma 3v to this.

Also we have

$$\Pi(\exists t/R)Q\equiv\Pi(\exists t/R^-)Q\vee Q[\bar{t}'\to t]).$$

The case (2) is obtained by using lemma 6. Finally by combining

$$\Pi(\exists t/R^*)Q\equiv\Pi(\exists t/R^-)Q\vee Q[\bar{t}\to t])\subset\Pi(\exists t/R^-)Q\vee\Pi Q[\bar{t}\to t]$$

and the result in case (2), case (3) is obtained.

Cases (2) and (3) present sufficient conditions only. It can be noted, however, that all $\Pi^u\sim Q[\bar{t}'\to t]$ (or $\Pi^e Q[\bar{t}'\to t]$), $\Pi Q[\bar{t}\to t]$ and

$$\Pi(\exists t/R^-)Q\equiv\Pi(\exists t/R)(A_p(t)\neq A_p(\bar{t}')\wedge Q)$$

can be evaluated in the current database (before applying the update).

6 ASSERTIONS WITH MORE THAN ONE UPDATE-RELEVANT VARIABLE

Let us next examine assertions with two or more update-relevant variables. First let us assume that the given assertion has two update-relevant variables t_1 and t_2.

THEOREM Ⅶ: If the assertion is of the form

$$P\equiv(\forall t_1/R)(\forall t_2/R)Q$$

and Q contains no other update-relevant variables than t_1 and t_2, then

(1) Adding a tuple $\bar{t}$ to R is proper with respect to P if and only if all $(\forall t_1/R)Q[\bar{t}\to t_2]$, $(\forall t_2/R)Q[\bar{t}\to t_1]$ and $Q[\bar{t}\to t_1,t_2]$ are 'true'.

(2) Deleting a tuple $\bar{t}'$ from R is always proper with respect to P.

(3) Repalacing a tuple $\bar{t}'$ in R by $\bar{t}$ is proper with respect to P if and only if all $(\forall t_1/R^-)Q[\bar{t}\to t_2]$, $(\forall t_2/R^-)Q[\bar{t}\to t_1]$ and $Q[\bar{t}\to t_1,t_2]$ are 'true'.

PROOF: This theorem is a simple extention of theorem Ⅴ. It is obtained from:

$$\begin{aligned}(\forall t_1/R^+)(\forall t_2/R^+)Q&\equiv(\forall t_1/R^+)((\forall t_2/R)Q\wedge Q[\bar{t}\to t_2])\\&\equiv(\forall t_1/R^+)(\forall t_2/R)Q\wedge(\forall t_1/R^+)Q[\bar{t}\to t_2]\\&\equiv(\forall t_1/R)(\forall t_2/R)Q\wedge(\forall t_2/R)Q[\bar{t}\to t_1]\\&\quad\wedge(\forall t_1/R)Q[\bar{t}\to t_2]\wedge Q[\bar{t}\to t_1,t_2]\\(\forall t_1/R)(\forall t_2/R)Q&\equiv(\forall t_1/R^-)(\forall t_2/R^-)Q\wedge(\forall t_1/R^-)Q[\bar{t}'\to t_2]\\&\quad\wedge(\forall t_2/R^-)Q[\bar{t}'\to t_1]\wedge Q[\bar{t}'\to t_1,t_2]\\(\forall t_1/R^*)(\forall t_2/R^*)Q&\equiv(\forall t_1/R^-)(\forall t_2/R^-)Q\wedge(\forall t_1/R^-)Q[\bar{t}\to t_2]\\&\quad\wedge(\forall t_2/R^-)Q[\bar{t}\to t_1]\wedge Q[\bar{t}\to t_1,t_2]\end{aligned}$$

□

It should be noticed that

$$(\forall t_1/R)Q[\bar{t}\to t_2]\wedge(\forall t_2/R)Q[\bar{t}\to t_1]\wedge Q[\bar{t}\to t_1,t_2]$$

being 'true' is a sufficient condition for replacing a tuple being proper with respect to P. However, it is not a necessary condition. In fact,

$$(\forall t_1/R)Q[\bar{t}\to t_2]\equiv(\forall t_1/R^-)Q[\bar{t}\to t_2]\wedge Q[\bar{t}'\to t_1,\bar{t}\to t_2]$$

and

$$(\forall t_2/R)Q[\bar{t}\to t_1]\equiv(\forall t_2/R^-)A[\bar{t}\to t_1]\wedge Q[\bar{t}\to t_1,\bar{t}'\to t_2]$$

hold, where $Q[\bar{t}_1\to t_1,\bar{t}_2\to t_2]$ is an LEX obtained by substituting $\bar{t}_1$ for t_1 and $\bar{t}_2$ for t_2 in Q. Therefore, $(\forall t_1/R^-)Q[\bar{t}\to t_2]$ is equivalent to $(\forall t_1/R)Q[\bar{t}\to t_2]$ only when $Q[\bar{t}'\to t_1,\bar{t}\to t_2]$ is 'true', and $(\forall t_2/R^-)Q[\bar{t}\to t_1]$ is equivalent to $(\forall t_2/R)Q[\bar{t}\to t_1]$ only when $Q[\bar{t}\to t_1,\bar{t}'\to t_2]$ is 'true'.

THEOREM Ⅷ: If the assertion is of the form

$P\equiv(\exists t_1/R)(\exists t_2/R)Q$

and Q contains no other update-relevant varaibles than t_1 and t_2, then

(1) Adding a tuple $\bar{t}$ to R is always proper with respect to P.

(2) Deleting a tuple $\bar{t}'$ from R is proper with respect to P if all $(\exists t_1/R^-)Q[\bar{t}'\rightarrow t_2]$, $(\exists t_2/R^-)Q[\bar{t}'\rightarrow t_1]$ and $Q[\bar{t}'\rightarrow t_1,t_2]$ are 'false'.

(3) Replacing a tuple $\bar{t}'$ in R by $\bar{t}$ is proper with respect to P if all $(\exists t_1/R^-)Q[\bar{t}'\rightarrow t_2]$, $(\exists t_2/R^-)Q[\bar{t}'\rightarrow t_1]$ and $Q[\bar{t}'\rightarrow t_1,t_2]$ are 'false' or if at least one of $(\exists t_1/R^-)Q[\bar{t}\rightarrow t_2]$, $(\exists t_2/R^-)Q[\bar{t}\rightarrow t_1]$ and $Q[\bar{t}\rightarrow t_1,t_2]$ is 'true'. Otherwise it is proper with respect to P if and only if $(\exists t_1/R^-)(\exists t_2/R^-)Q$ is 'true'.

PROOF: This theorem is a simple extention of theorem $\exists$. It is obtained from

$$\begin{aligned}(\exists t_1/R^+)(\exists t_2/R^+)Q &\equiv(\exists t_1/R^+)((\exists t_2/R)Q\vee Q[\bar{t}\rightarrow t_2])\\ &\equiv(\exists t_1/R^+)(\exists t_2/R)Q\vee(\exists t_1/R^+)Q[\bar{t}\rightarrow t_2]\\ &\equiv(\exists t_1/R)(\exists t_2/R)Q\vee(\exists t_2/R)Q[\bar{t}\rightarrow t_1]\\ &\quad\vee(\exists t_1/R)Q[\bar{t}\rightarrow t_2]\vee Q[\bar{t}\rightarrow t_1,t_2]\end{aligned}$$

$$\begin{aligned}(\exists t_1/R)(\exists t_2/R)Q &\equiv(\exists t_1/R^-)(\exists t_2/R^-)Q\vee(\exists t_1/R^-)Q[\bar{t}'\rightarrow t_2]\\ &\quad\vee(\exists t_2/R^-)Q[\bar{t}'\rightarrow t_1]\vee Q[\bar{t}'\rightarrow t_1,t_2]\end{aligned}$$

and

$$\begin{aligned}(\exists t_1/R^*)(\exists t_2/R^*)Q &\equiv(\exists t_1/R^-)(\exists t_2/R^-)Q\vee(\exists t_1/R^-)Q[\bar{t}\rightarrow t_2]\\ &\quad\vee(\exists t_2/R^-)Q[\bar{t}\rightarrow t_1]\vee Q[\bar{t}\rightarrow t_1,t_2]. \qquad \square\end{aligned}$$

THEOREM $\forall\exists$: If the assertion is of the form

$P\equiv(\forall t_1/R)(\exists t_2/R)Q$

and Q contains no other update-relevant variables than t_1 and t_2, then

(1) Adding a tuple $\bar{t}$ to R is proper with respect to P if and only if $(\exists t_2/R)Q[\bar{t}\rightarrow t_1]$ or $Q[\bar{t}\rightarrow t_1,t_2]$ is 'true'.

(2) Deleting a tuple $\bar{t}'$ from R is proper with respect to P if $(\exists t_1/R^-)Q[\bar{t}'\rightarrow t_2]$ is 'false'.

(3) Replacing a tuple $\bar{t}'$ in R by $\bar{t}$ is proper with respect to P if and only if $(\forall t_1/R^-)((\exists t_2/R^-)Q\vee Q[\bar{t}\rightarrow t_2])$ is 'true' and at least one of $(\exists t_2/R^-)Q[\bar{t}\rightarrow t_1]$ and $Q[\bar{t}\rightarrow t_1,t_2]$ is 'true'. The former is 'true' if $(\forall t_1/R^-)(\exists t_2/R^-)Q$ or $(\forall t_1/R^-)Q[\bar{t}\rightarrow t_2]$ is 'true'. If $(\exists t_1/R^-)Q[\bar{t}'\rightarrow t_2]$ is 'false', $(\forall t_1/R^-)(\exists t_2/R^-)Q$ is 'true'.

PROOF: It can be seen that

$$(\forall t_1/R^+)(\exists t_2/R^+)Q \equiv (\forall t_1/R^+)((\exists t_2/R)Q \vee Q[\bar{t}\to t_2])$$
$$\equiv (\forall t_1/R)((\exists t_2/R)Q \vee Q[\bar{t}\to t_2])$$
$$\wedge((\exists t_2/R)Q[\bar{t}\to t_1] \vee Q[\bar{t}\to t_1,t_2]).$$

Since

$$(\forall t_1)(\exists t_2/R)Q \vee (\forall t_1/R)Q[\bar{t}\to t_2] \supset (\forall t_1/R)((\exists t_2/R)Q \vee Q[\bar{t}\to t_2])$$

the first term is always 'true'. This is case (1).

Since

$$(\forall t_1/R)(\exists t_2/R)Q \equiv (\forall t_1/R^-)((\exists t_2/R^-)Q \vee Q[\bar{t}'\to t])$$
$$\wedge((\exists t_2/R^-)Q[\bar{t}'\to t_1] \vee Q[\bar{t}'\to t_1,t_2]),$$
$$(\forall t_1/R^-)((\exists t_2/R^-)Q \vee Q[\bar{t}'\to t_2])$$

is always 'true'. From lemma 6, if $(\forall t_1/R^-)\sim Q[\bar{t}'\to t_2]$ is 'true', that is, if $(\exists t_1/R^-)Q[\bar{t}'\to t_2]$ is 'false', then $(\forall t_1/R^-)(\exists t_2/R^-)Q$ must be 'true'. This is case (2). Finally, case (3) can be obtained from

$$(\forall t_1/R^*)(\exists t_2/R^*)Q \equiv (\forall t_1/R^-)((\exists t_2/R^-)Q \vee Q[\bar{t}\to t_2])$$
$$\wedge((\exists t_2/R^-)Q[\bar{t}\to t_1] \vee Q[\bar{t}\to t_1,t_2]) \qquad \Box$$

THEOREM $\exists\forall$: If the assertion is of the form

$$P \equiv (\exists t_1/R)(\forall t_2/R)Q$$

and Q contains no other update-relevant variables than t_1 and t_2, then

(1) Adding a tuple $\bar{t}$ to R is proper with respect to P if and only if $(\exists t_1/R)((\forall t_2/R)Q \wedge Q[\bar{t}\to t_2])$ is 'true' or both $(\forall t_2/R)Q[\bar{t}\to t_1]$ and $Q[\bar{t}\to t_1,t_2]$ are 'true'. Here $(\exists t_1/R)((\forall t_2/R)Q \wedge Q[\bar{t}\to t_2])$ is 'false' if $(\exists t_1/R)Q[\bar{t}\to t_2]$ is 'false'.

(2) Deleting a tuple $\bar{t}'$ from R is proper with respect to P if $(\forall t_2/R^-)Q[\bar{t}'\to t_1]$ or $Q[\bar{t}'\to t_1,t_2]$ is 'false'.

(3) Replacing a tuple $\bar{t}'$ in R by $\bar{t}$ is proper with respect to P if and only if $(\exists t_1/R^-)((\forall t_2/R^-)Q \wedge Q[\bar{t}\to t_2])$ is 'true' or both $(\forall t_2/R^-)Q[\bar{t}\to t_1]$ and $Q[\bar{t}\to t_1,t_2]$ are 'true'. Here $(\exists t_1/R^-)((\forall t_2/R^-)Q \wedge Q[\bar{t}\to t_2])$ is 'false' if $(\exists t_1/R^-)(\forall t_2/R^-)Q$ or $(\exists t_1/R^-)Q[\bar{t}\to t_2]$ is 'false'.

PROOF: It can be seen that

$$(\exists t_1/R^+)(\forall t_2/R^+)Q \equiv (\exists t_1/R^+)((\forall t_2/R)Q \wedge Q[\bar{t}\to t_2])$$
$$\equiv (\exists t_1/R)((\forall t_2/R)Q \wedge Q[\bar{t}\to t_2])$$
$$\vee((\forall t_2/R)Q[\bar{t}\to t_1] \wedge Q[\bar{t}\to t_1,t_2])$$

It implies that $(\exists t_1/R^+)(\forall t_2/R^+)Q$ is 'true' if and only if $(\exists t_1/R)((\forall t_2/R)Q \wedge Q[\bar{t}\to t_2])$ or $(\forall t_2/R)Q[\bar{t}\to t_1] \wedge Q[\bar{t}\to t_1,t_2]$ is 'true'. For the former, lemmas 4 and 5 can be applied to obtain case (1).

Next, in

$$(^{\exists}t_1/R)(^{\forall}t_2/R)Q \equiv (^{\exists}t_1/R^-)((^{\forall}t_2/R^-)Q \wedge Q[\bar{t}' \rightarrow t_2]) \vee ((^{\forall}t_2/R^-)Q[\bar{t}' \rightarrow t_1] \wedge Q[\bar{t}' \rightarrow t_1, t_2]),$$

the first term must be 'true' if the second term is 'false'. Then lemmas 4 and 3$\wedge$ can be applied to obtain case (2).

Finally, it is seen that

$$(^{\forall}t_1/R^*)(^{\exists}t_2/R^*)Q \equiv (^{\exists}t_1/R^-)((^{\forall}t_2/R^-)Q \wedge Q[\bar{t} \rightarrow t_2]) \vee ((^{\forall}t_2/R^-)Q[\bar{t} \rightarrow t_1] \wedge Q[\bar{t} \rightarrow t_1, t_2]),$$

and hence $(^{\exists}t_1/R^*)(^{\forall}t_2/R^*)Q$ is 'true' if and only if one of $(^{\exists}t_1/R^-)((^{\forall}t_2/R^-)Q \wedge Q[\bar{t} \rightarrow t_2])$ and $(^{\forall}t_2/R^-)Q[\bar{t} \rightarrow t_1] \wedge Q[\bar{t} \rightarrow t_1, t_2])$ is 'true'. For the former, lemmas 4 and 5 can be applied to get case (3). □

Let us next consider assertions with three update-relevant variables. Let t_1, t_2 and t_3 be such tuple variables. It will be very easy to deduce theorems $\forall\forall\forall$ and $\exists\exists\exists$ because they are simple extentions of theorems $\forall\forall$ and $\exists\exists$. Only a theorem corresponding to assertions, which frequently appear in practical applications, is shown below.

THEOREM $\forall\forall\exists$: If the assertion is of the form

$$P \equiv (^{\forall}t_1/R)(^{\forall}t_2/R)(^{\exists}t_3/R)Q$$

and Q contains no other update-relevant varaibles than t_1, t_2 and t_3, then

(1) Adding a tuple $\bar{t}$ to R is proper with respect to P if and only if all

$$(\forall t_1/R)((\exists t_3/R)Q[\bar{t} \rightarrow t_2] \vee Q[\bar{t} \rightarrow t_2, t_3]),$$
$$(\forall t_2/R)((\exists t_3/R)Q[\bar{t} \rightarrow t_1] \vee Q[\bar{t} \rightarrow t_1, t_3])$$

and

$$(\exists t_3)Q[\bar{t} \rightarrow t_1, t_2] \vee Q[\bar{t} \rightarrow t_1, t_2, t_3]$$

are 'true'.

(2) Deleting a tuple $\bar{t}'$ from R is proper with respect to P if $(\exists t_1/R^-)(\exists t_2/R^-)Q[\bar{t}' \rightarrow t_3]$ is 'false'.

(3) Replacing a tuple $\bar{t}'$ in R by $\bar{t}$ is proper with respect to P if and only if all

$$(\forall t_1/R^-)(\forall t_2/R^-)((\exists t_3/R^-)Q \vee Q[\bar{t} \rightarrow t_3]),$$
$$(\forall t_1/R^-)((\exists t_3/R^-)Q[\bar{t} \rightarrow t_2] \vee Q[\bar{t} \rightarrow t_2, t_3]),$$
$$(\forall t_2/R^-)((\exists t_3/R^-)Q[\bar{t} \rightarrow t_1] \vee Q[\bar{t} \rightarrow t_1, t_3]),$$

and

$(\exists t_3/R)Q[\bar{t}\to t_1,t_2] \vee Q[\bar{t}\to t_1,t_2,t_3]$

are 'true'. The first term is 'true' if $(\exists t_1/R^-)(\exists t_2/R^-)Q[\bar{t}'\to t_3]$ is 'false'.

This theorem can be proven in a similar manner to proving the theorem $\exists\forall$. If necessary, it is possible to deduce theorems correspondina tp any other cases such as $\Pi\forall\forall$, $\Pi\exists\exists$, $\forall\exists\exists$, $\exists\exists\forall$ and so forth. Deduction of each theorem is composed of two steps, (1) transformation of three extended relational calculi corresponding to adding, deleting and replacing a tuple, and (2) application of several lemmas as deductive rules. In step (1) 2^N terms are generated, where N is the number of update-relevant variables. Step (2) uses only a small number of deductive rules in a limited number of combinations. Therefore, the time necessary to deduce a theorem is estimated to be less than $O(2^N)$.

From the two transformation rules presented as lemma 2, it can be shown that theorems which are useful for finding certain efficient validation procedures can be deduced only in the cases corresponding to $\mathcal{V}$, $\mathcal{E}$, $\mathcal{V}\mathcal{E}$, $\mathcal{E}\mathcal{V}$, $\Pi\mathcal{V}$, $\Pi\mathcal{E}$, $\mathcal{V}\Pi\mathcal{V}$, $\mathcal{E}\Pi\mathcal{E}$, $\mathcal{V}\Pi\mathcal{E}$ and $\mathcal{E}\Pi\mathcal{V}$ where $\mathcal{V}$ is a series of domain-coupled universal quantifiers regarding update-relevant variables, $\mathcal{E}$ is a series of domain-coupled existential quantifiers regarding update-relevant variables, and Π is a series of domain-coupled (universal and/or existential) qunatifiers regarding update-irrelevant variables. From the practical point of view, only a small limitted number of prefix patterns appear. Therefore, one may prefabricate a limitted number of theorems instead of deducing a theorem each time an assertion and an update are specified.

7 EFFECTS OF SUBSTITUTIONS

In the theorems presented so far, the following three cases exists:

(1) An assertion P becomes 'true' in the updated database if an extended relational calculus P' is 'true' in the current database (P' is a sufficient condition for P).

(2) P becomes 'false' in the updated database if P' is 'false' in the

current database (P' is a necessary condition for P).

(3) P becomes 'true' in the updated database if and only if P' is 'true' in the current database (P' is a necessary and sufficient condition for P).

(4) P is always 'true' in the updated database.

These cases are useful because it can be evaluated in the current database. Besides, in most cases, P' contains less domain-coupled quantifiers than P itself.

One more important effect is caused by substitutions applied in obtaining P'. Substituting a tuple (constant) for a tuple variable decreases the number of tuple variables by 1. Application of n substitutions decreases the number of tuple variables by n.

The extended relational calculus P' may consist of several component LEXs combined by relational operators. If P is symmetric with respect to two or more tuple variables, some of these component LEXs become equivalent with each other.

As the result of substitutions, some of these LEXs become simply 'true', and some others simply 'false'. Such LEXs can be evaluated very quickly.

In particular, if P contains an LEX of the form $A_i(t_1)\theta A_j(t_2)$ where θ is a relational operator (join term), it is transformed into an LEX of the form $A_i(t_1)\theta A_j(\bar{t})$ (=const) (selection term) in P'.when substituting $\bar{t}$ for t_2. The database search algorithm [1] takes advantage of this property, which is named "complacency" in [3].

What a simple extended relational calculus P' is obtained depends on the form of the matrix part Q of the given assertion P. Let us next examine several typical assertions.

TUPLE CONSTRAINTS: Assertions corresponding to tuple constraints [2] are of the form

$P \equiv (^{\forall}t/R)Q$

where Q contains no other tuple variables than t. Theorem $\forall$ states that adding and replacing a tuple is proper with respect to P if and only if $P' \equiv Q[\bar{t} \rightarrow t]$ is 'true', while deleting a tuple is always proper with respect to P. The P' value can be quickly evaluated since it now contains no tuple variables.

FUNCTIONAL DEPENDENCIES: Assertions for FDs are of the form

$$P \equiv (^{\forall}t_1/R)(^{\forall}t_2/R)Q$$

where

$$Q \equiv (A_1(t_1)=A_1(t_2) \supset A_2(t_1)=A_2(t_2))$$

with A_1 and A_2 being two sets of attributes of the relation R. Theorem ∀∀ states that adding and replacing a tuple is proper with respect to P if and only if

$$(^{\forall}t_1/R)Q[\bar{t}\rightarrow t_2] \wedge (^{\forall}t_2/R)Q[\bar{t}\rightarrow t_1] \wedge Q[\bar{t}\rightarrow t_1,t_2]$$

is 'true', while deleting a tuple is always proper with respect to P. The assertion P is symmetric with respect to t_1 and t_2. This implies

$$(\forall t_1/R)Q[\bar{t}\rightarrow t_2] \equiv (\forall t_2/R)Q[\bar{t}\rightarrow t_1].$$

Furthermore, whatever $\bar{t}$ might be,

$$Q[\bar{t}\rightarrow t_1,t_2] \equiv (A_1(\bar{t})=A_1(\bar{t}) \supset A_2(\bar{t})=A_2(\bar{t})$$

is 'true' Therefore, to validate adding and replacing a tuple, it is sufficient to ascertain that

$$P' \equiv (\forall t_1/R)(A_1(t)=A_1(\bar{t}) \supset A_2(t_1)=A_2(\bar{t})).$$

is 'true'.

(EMBEDDED) MULTIVALUED DEPENDENCIES: Assertions for EMVDs are of the form

$$P \equiv (\forall t_1/R)(\forall t_2/R)(\exists t_3/R)Q$$

where

$$Q \equiv (A_1(t_1)=A_1(t_2) \supset (A_2(t_3)=A_2(t_1) \wedge A_3(t_3)=A_3(t_2))$$

with A_1, A_2 and A_3 being three sets of attributes of the relation R. If there are no attributes of R which are not included in $A_1 \cup A_2 \cup A_3$, the assertions are for MVDs. Theorem ∀∀∃ can now be applied. For adding a tuple, all

$$(^{\forall}t_1/R)((^{\exists}t_3/R)Q[\bar{t}\rightarrow t_2] \vee Q[\bar{t}\rightarrow t_2,t_3]),$$
$$(^{\forall}t_2/R)((^{\exists}t_3/R)Q[\bar{t}\rightarrow t_1] \vee Q[\bar{t}\rightarrow t_1,t_3])$$

and

$$((^{\exists}t_3/R)Q[\bar{t}\rightarrow t_1,t_2] \vee Q[\bar{t}\rightarrow t_1,t_2,t_3])$$

must be 'true'. In this case,

$$Q[\bar{t}\rightarrow t_1,t_2,t_3] \equiv (A_1(\bar{t})=A_1(\bar{t}) \supset (A_2(\bar{t})=A_2(\bar{t}) \wedge A_3(\bar{t})=A_3(\bar{t})))$$

is 'true' whatever $\bar{t}$ might be, and hence the last term is always 'true'. Therefore, it is necessary and sufficient to examine whether both

$$(^{\forall}t_1/R)((^{\exists}t_3/R)Q[\bar{t}\rightarrow t_2] \vee Q[\bar{t}\rightarrow t_2,t_3])$$
$$\equiv (^{\forall}t_1/R)((^{\exists}t_3/R)(A_1(t_1)=A_1(\bar{t}) \supset (A_2(t_3)=A_2(t_1) \wedge A_3(t_3)=A_3(\bar{t})))$$
$$\vee (A_1(t_1)=A_1(\bar{t}) \supset (A_2(\bar{t})=A_2(t_1) \wedge A_3(\bar{t})=A_3(\bar{t}))))$$

$$\begin{aligned}&\equiv(\forall t_1/R)((A_1(t_1)=A_1(\bar{t})\supset(\exists t_3/R)(A_2(t_3)=A_2(t_1)\wedge A_3(t_3)=A_3(\bar{t})))\\&\qquad\qquad\vee(A_1(t_1)=A_1(\bar{t})\supset A_2(t_1)=A_2(\bar{t})))\\&\equiv(\forall t_1/R)(A_1(t_1)=A_1(\bar{t})\supset(A_2(t_1)=A_2(\bar{t})\\&\qquad\vee(\exists t_3/R)(A_2(t_3)=A_2(t_1)\wedge A_3(t_3)=A_3(\bar{t}))))\\&\equiv(\forall t_1/R)(A_1(t_1)=A_1(\bar{t})\supset(A_2(t_1)\neq A_2(\bar{t})\supset\\&\qquad\supset(\exists t_3/R)(A_2(t_3)=A_2(t_1)\wedge A_3(t_3)=A_3(t_3))))\\&\equiv(\forall t_1/R)((A_1(t_1)=A_1(\bar{t})\wedge A_2(t_1)\neq A_2(\bar{t}))\\&\qquad\supset(\exists t_3/R)(A_2(t_3)=A_2(t_1)\wedge A_3(t_3)=A_3(\bar{t})))\end{aligned}$$

and

$$\begin{aligned}&(\forall t_2/R)((\exists t_3/R)Q[\bar{t}\to t_1]\vee Q[\bar{t}\to t_1,t_3])\\&\equiv(\forall t_2/R)((A_1(t_2)=A_1(\bar{t})\wedge A_3(t_2)\neq A_3(\bar{t}))\\&\qquad\supset(\exists t_3/R)(A_2(t_3)=A_2(\bar{t})\wedge A_3(t_3)=A_3(t_2)))\end{aligned}$$

are 'true'. There are no simpler extended relational calculus which act$\dot{s}$ as a necessary and sufficient condition for deleting a tuple to be proper with respect to P. The sufficient condition

$$\begin{aligned}&\sim(\exists t_1/R^-)(\exists t_2/R^-)Q[\bar{t}\to t_3]\equiv(\forall t_1/R^-)(\forall t_2/R^-)\sim Q[\bar{t}\to t_3]\\&\equiv(\forall t_1/R^-)(\forall t_2/R^-)\sim(A_1(t_1)=A_1(t_2)\supset(A_2(t_1)=A_2(\bar{t})\wedge A_3(t_2)=A_3(\bar{t})))\\&\equiv(\forall t_1/R^-)(\forall t_2/R^-)\sim(A_1(t_1)\neq A_1(t_2)\vee(A_2(t_1)=A_2(\bar{t})\wedge A_3(t_2)=A_3(\bar{t})))\\&\equiv(\forall t_1/R^-)(\forall t_2/R^-)(A_1(t_1)=A_1(t_2)\wedge(A_2(t_1)\neq A_2(\bar{t})\vee A_3(t_2)\neq A_3(\bar{t})))\end{aligned}$$

is useless in this case because it becomes 'false' whenever R^- contains more than one tuple. For replacing a tuple, it is necessary to examine all

$$\begin{aligned}&(\forall t_1/R^-)(\forall t_2/R^-)((\exists t_3/R^-)Q\vee Q[\bar{t}\to t_3])\\&\equiv(\forall t_1/R^-)(\forall t_2/R^-)((A_1(t_1)=A_1(t_2)\wedge(A_2(t_1)\neq A_2(\bar{t})\vee A_3(t_2)\neq A_3(\bar{t})))\\&\qquad\qquad\supset(\exists t_3/R^-)(A_2(t_3)=A_2(t_1)\wedge A_3(t_3)=A_3(t_2))),\\&(\forall t_1/R^-)((\exists t_3/R^-)Q[\bar{t}\to t_2]\vee Q[\bar{t}\to t_2,t_3]\\&\equiv(\forall t_1/R^-)((A_1(t_1)=A_1(\bar{t})\wedge A_2(t_1)\neq A_2(\bar{t}))\\&\qquad\supset(\exists t_3/R^-)(A_2(t_3)=A_2(t_1)\wedge A_3(t_3)=A_3(\bar{t})))\end{aligned}$$

and

$$\begin{aligned}&(\forall t_2/R^-)((\exists t_3/R^-)Q[\bar{t}\to t_1]\vee Q[\bar{t}\to t_1,t_3])\\&\equiv(\forall t_2/R^-)((A_1(t_2)=A_1(\bar{t})\wedge A_3(t_2)\neq A_3(\bar{t}))\\&\qquad\supset(\exists t_3/R^-)(A_2(t_3)=A_2(\bar{t})\wedge A_3(t_3)=A_3(t_2))).\end{aligned}$$

As seen above, validation of updates for MVDs (and EMVDs) is much less easier than that for FDs, in particular, if the update involes deleting or replacing a tuple. It may be concluded from this point of view that the non-first normal form [2,4] is rather desirable than the first normal form, and if the first normal form is employed then the fourth normal form decomposition is mandatory.

Note that in calculating P' for an (E)MVD, several properties in propositional logic and (many-sorted) predicate logic are used. They are:

LEMMA 7; Let P_1, P_2 and P_3 be propositions. Then,

(1) $(P_1 \vee P_2) \equiv (\sim P_1 \supset P_2)$

(2) $((P_1 \supset P_2) \vee (P_1 \supset P_3)) \equiv (P_1 \supset (P_2 \vee P_3))$

(3) $(P_1 \supset (P_2 \supset P_3)) \equiv ((P_1 \wedge P_2) \supset P_3)$

LEMMA 8: If Π is a series of domain-couple quantifiers for tuple variables $t_1, t_2, \ldots, t_n$, and Q_1 contains none of tuple variables $t_1, t_2, \ldots, t_n$ then

(1) $\Pi(Q_1 \wedge Q_2) \equiv Q_1 \wedge \Pi Q_2$

(2) $\Pi(Q_1 \vee Q_2) \equiv Q_1 \vee \Pi Q_2$

(3) $\Pi(Q_1 \supset Q_2) \equiv Q_1 \supset \Pi Q_2$.

One more important notice must be made on the evaluation of P or components of P' for FDs and (E)MVDs. An extended relational calculus of the form

$$(\forall t_1/R_1)(\forall t_2/R_2)\ldots(\forall t_n/R_n)(Q_1 \supset Q_2)$$

is said to be a "Horn calculus" if Q_1 does not contain tuple variables other than $t_1, t_2, \ldots, t_n$. Here $R_1, R_2, \ldots, R_n$ are not necessarily mutually distinct. Since the database search algorithm [1] was devised mainly for dealing with extended relational calculi with free tuple variables, and it first obtains the set of ordered sets of tuples (satisfiers) each satisfies the matrix part of the given calculus, it does not make any special treatment for Horn calculi. However, Horn Calculi can be evaluated in a little more efficient manner. Since the given calculus is 'true' whenever $(\forall t_1/R_1)(\forall t_2/R_2)\ldots(\forall t_n/R_n)Q$ is 'false', One may first obtain

$$L = \{(t_1, t_2, \ldots, t_n) \mid t_1/R_1 \wedge t_2/R_2 \wedge \ldots \wedge t_n/R_n \wedge Q_1\}.$$

Then the given Horn calculus becomes 'true' if and only if

$$(\forall (t_1, t_2, \ldots, t_n)/L) Q_2$$

is 'true'.

In validating adding and replacing a tuple for an FD, it is sufficient to examine whether $A_2(t_1) = A_2(\bar{t})$ is 'true' for all tuples t_1 in R that satisfy $A_1(t_1) = A_1(\bar{t})$. Such tuples can be quickly obtained if the database file representing R is organized as a sequential, tree-structured or direct file using the A_1 value, or is indexed by the A_1 value.

In validating adding a tuple for (E)MVD, a similar procedure can be applied. Deleting a tuple must be validated by evaluating the $(t_1/R^-)(t_2/R^-)(t_3/R^-)Q$ value directly. However, since

$$\begin{aligned}&(\forall t_1/R^-)(\forall t_2/R^-)(\exists t_3/R^-)Q\\ &\equiv(\forall t_1/R^-)(\forall t_2/R^-)(A_1(t_1)=A_1(t_2)\\ &\qquad\supset(\exists t_3/R^-)(A_2(t_3)=A_2(t_1)\wedge A_3(t_3)=A_3(t_2)))\\ &\equiv(\forall t_1/R)(\forall t_2/R)((A_p(t_1)\neq A_p(\bar{t}')\wedge A_p(t_2)\neq A_p(\bar{t}')\wedge A_1(t_1)=A_1(t_2))\\ &\qquad\supset(\exists t_3/R)(A_p(t_3)\neq A_p(\bar{t}')\wedge A_2(t_3)=A_2(t_1)\wedge A_3(t_3)=A_3(t_2))),\end{aligned}$$

one may first obtain the set of tuple pairs (t_1,t_2) in $R\times R$ all satisfy $A_p(t_1)\neq A_p(\bar{t}')\wedge A_p(t_2)\neq A_p(\bar{t}')\wedge A_1(t_1)=A_1(t_2)$, and then for each pair in this set examine whether

$$(\exists t_3/R)(A_p(t_3)\neq A_p(\bar{t}')\wedge A_2(t_3)=A_2(t_1)\wedge A_3(t_3)=A_3(t_2))$$

is 'true'.

All three terms to be examined for validating replacing a tuple are also Horn calculus. Similar procedures can be used for examining these.

All (embedded) mutual dependencies, subset dependencies, join dependencies, and, in geberal, template dependencies [5] are Horn calculi. The above procedure can be applied in validating database updates for these dependencies, although the resulting procedures can still be time-consuming.

Tuple constraints, functional dependencies and other (template) dependencies are intrarelation constraints [2]. All tuple variables are bound over the same relation. Let us next consider some interrelation constraints, which have two or more tuple variables bound over distinct relations.

INCLUSION CONSTRAINTS: Assertions corresponding to inclusion constraints are of the form

$$P\equiv(\forall t_1/R_1)(\exists t_2/R_2)A_1(t_1)=A_2(t_2).$$

If A_2 is a candidate key of the relation R_2, the constraint is an into constraint, while if A_1 is a candidate key of the relation R_1, it is an onto constraint, both from R_1 to R_2 [2]. Two distinct cases must be discussed: the case where t_1 is update-relevant, and the case where t_2 is update-relevant. For the former case, theorem V is applicable. Therefore, both adding a tuple $\bar{t}$ to R_1 and replacing a tuple $\bar{t}'$ in R_1 by $\bar{t}$ are proper with respect to P if and only if

$$Q[\bar{t}\rightarrow t_1]\equiv(\exists t_2/R_2)A_2(t_2)=A_1(\bar{t})$$

is 'true'. Deleting a tuple $\bar{t}'$ from R_1 is always proper with respect to P. For the latter case, theorem $\Pi\exists$ is applicable. Therefore, adding a tuple $\bar{t}$ to R_1 is always proper with respect to P. Deleting a tuple $\bar{t}'$ from R_2 is proper with respect to P if

$(\forall t_1/R_1)A_1(t_1)\neq A_2(\bar{t}')$

is 'true', or equivalently

$(\exists t_1/R_1)A_1(t_1)=A_2(\bar{t}')$

is 'false'. This is only a sufficient condition but is effective because it can be evaluated easily. Replacing a tuple $\bar{t}'$ in R_2 by $\bar{t}$ is proper with respect to P if

$(\forall t_1/R_1)(\exists t_2/R_2)A_1(t_1)=A_2(t_2)$

or

$(\forall t_1/R_1)A_1(t_1)=A_2(\bar{t})$

is 'true'. The latter is useless because it becomes 'false' whenever R_1 contains a tuple t_1 for which $A_1(t_1)\neq A_2(\bar{t})$. However, the former can be used because it is 'true' if

$(\exists t_1/R_1)A_1(t_1)=A_2(\bar{t}')$

is 'false', although it is only a sufficient condition.

RELATIONSHIP CONSTRAINTS: A relationship constraint [2] is a conjunct of two or more into constraints , and hence the validating procedure for inclusion constraints can be applied.

After deducing an appropriate theorem, necessary substitutions must be applied and, if possible, the obtained relational calculus must be somehow simplified. The time required for this process can be estimated to be $O(\ell)$ where ℓ is the length of the calculus.

8 NICOLAS' EXAMPLES

Nicolas in [6] presented a method of improving validation procedures, which is based on a similar idea to that used in developing the method presented in this paper but was developed using notions in (ordinary) first-order logic. The method presented here is superior to Nicolas' in several points. Among them two major advantages are: (1) it deals with existential quantifiers and unversal quantifiers in the scope of an existential quantifier as well as other universal quantifiers, and (2) it presents a necessary or a sufficient condition sometimes even when no necessary and sufficient condition for the unit update being proper exists. These makes this method much more powerful than Nicolas'. Here let us demonstrate these points by applying this method to the examples that Nicolas used in [6]. (Extended) relational calculi are used instead of (one-sorted) first-order predicates to represent assertions.

There are four relation schemata

(a) R_{supply} ({comp,dept,item}): Each tuple $t_1=(C,D,I)$ in a relation R_1 belonging to this relation schema represents that "company C supplies depertment D with item I."

(b) R_{class}({item, type}): Each tuple $t_2=(I,T)$ in a relation R_2 belonging to this relation schema represents that "I is a type T item."

(c) R_{sale}({dept,item}): Each tuple $t_3=(D,I)$ in a relation R_3 belonging to this relation schema represents that "department D sells item I."

(d) R_{subord}({emp,mng}): each tuple $t_4=(E_1,E_2)$ in relation R_4 belonging to this relation schema represents that "employee E_1 is asubordinate of E_2."

ASSERTION 1: "When a department sells an item then there is a company which supplies it with this item." can be written as

$$(\forall t_3/R_3)(\exists t_1/R_1)(dept(t_3)=dept(t_1)\wedge item(t_3)=item(t_1)).$$

If t_3 is update relevant, adding a tuple $\bar{t}$ to R_1 can be validated by examining whether

$$(\exists t_1/R_1)(dept(t_1)=dept(\bar{t})\wedge item(t_1)=item(\bar{t}))$$

is 'true'. while deleting a tuple from R_1 is always proper with respect to this assertion (inclusion constraint).

ASSERTION 2: "No other companies than company C supplies type T_4 items." can be written as

$$(\forall t_1/R_1)(\forall t_2/R_2)((item(t_1)=item(t_2)\wedge type(t_2)=T_4)\supset comp(t_1)=C)).$$

If t_2 is update-relevant, (since two adjacent universal quantifiers) are commutative) adding a tuple $\bar{t}$ to R_2 can be validated by examining whether

$$(\forall t_1/R_1)((item(t_1)=item(\bar{t})\wedge item(\bar{t})=T_4)\supset comp(t_1)=C)$$

is 'true'. As this is a Horn calculus, one may first obtain

$$\{t_1|t_1/R_1\wedge item(t_1)=item(\bar{t})\wedge type(\bar{t})=T_4\}.$$

This becomes empty whenever type $(\bar{t})\neq T_4$. Otherwise it is necessary to examine whether $comp(t_1)=C$ for all tuples t_1 in this set, or equivalently there are no tuples t_1 for which $\mathrm{comp}(t_1)\neq C$. Deleting a tuple $\bar{t}'$ from R_2 is always proper with respect to this assertion.

ASSERTION 3: "Any company that supplies I_1 also supplies I_2" can be written as

$$(\forall t_1/R_1)(\exists t_1'/R_1)(\mathrm{item}(t_1)=I_1\supset(\mathrm{comp}(t_1')=\mathrm{comp}(t_1)\wedge\mathrm{item}(t_1')=I_2)).$$

Adding a tuple $\bar{t}$ to R_1 can be validated by examining whether

$$(\exists t_1'/R_1)(\mathrm{item}(\bar{t})=I_1\supset(\mathrm{comp}(t_1')=\mathrm{comp}(\bar{t})\wedge\mathrm{item}(t_1')=I_2))$$

or

$$\mathrm{item}(\bar{t})=I_1\supset(\mathrm{comp}(\bar{t})=\mathrm{comp}(\bar{t})\wedge\mathrm{item}(\bar{t})=I_2)$$

is 'true'. The both becomes 'true' whenever $\mathrm{item}(\bar{t})\neq I_1$. If $\mathrm{item}(\bar{t})=I_1$, the first term becomes 'true' if

$$(\exists t_1'/R_1)(\mathrm{comp}(t_1')=\mathrm{comp}(\bar{t})\wedge\mathrm{item}(t_1')=I_2).$$

is 'true', while the second term is always 'false'. For deleting a tuple $\bar{t}'$ from R_1, no simpler form acting a necessary and sufficient condition exists. However, there is a sufficient condition that

$$(\exists t_1/R_1^-)(\mathrm{item}(t_1)\neq I_1\wedge\mathrm{comp}(t_1)=\mathrm{comp}(\bar{t}')\wedge\mathrm{item}(\bar{t}')=I_2))$$

to be 'false'. This becomes 'false' whenever $\mathrm{item}(\bar{t}')\neq I_2$.

ASSERTION 4: "Any company that supplies type T_1 items also supplies type T_2 items" can be written as

$$(\forall t_1/R_1^-)(\forall t_2/R_2)(\exists t_1'/R_1)(\exists t_2'/R_2)$$
$$((\mathrm{item}(t_1)=\mathrm{item}(t_2)\wedge\mathrm{type}(t_2)=T_1)$$
$$\supset(\mathrm{comp}(t_1)=\mathrm{comp}(t_1')\wedge\mathrm{item}(t_1')=\mathrm{item}(t_2')\wedge\mathrm{type}(t_2')=T_2)).$$

Let t_1 and t_1' be update-relevant. In this case theorem $\forall\Pi\exists$ becomes necessary. Applying transformation rules in lemma 2,

$$(\forall t_1/R_1^+)\Pi(\exists t_1'/R_1^+)Q$$
$$\equiv(\forall t_1/R)\Pi((\exists t_1'/R_1)Q\vee Q[\bar{t}\to t_1'])\wedge\Pi((\exists t_1'/R_1)Q[\bar{t}\to t_1]\vee Q[\bar{t}\to t_1,t_1']).$$

is obtained. From the lemma 4, it can be seen that

$$((\forall t_1/R_1)\Pi(\exists t_1'/R_1)Q\vee(\forall t_1/R_1)\Pi Q[\bar{t}\to t_1'])\supset(\forall t_1/R_1)\Pi((\exists t_1'/R_1)Q\vee Q[\bar{t}\to t_1']),$$

and hence the first term is always 'true'. Therefore, adding a tuple $\bar{t}$ to R_1 is proper with respect to this assertion if and only if

$$\Pi((\exists t_1'/R_1)Q[\bar{t}\to t_1]\vee Q[\bar{t}\to t_1,t_1'])$$
$$\equiv(\forall t_2/R_2)((\exists t_1'/R_1)(\exists t_2'/R_2)((\mathrm{item}(t_2)=\mathrm{item}(\bar{t})\wedge\mathrm{type}(t_2)=T_1)$$
$$\supset((\mathrm{comp}(t_1')=\mathrm{comp}(\bar{t})\wedge\mathrm{item}(t_1')=\mathrm{item}(t_2')\wedge\mathrm{type}(t_2')=T_2))$$
$$\vee(\exists t_2'/R_2)((\mathrm{item}(t_2)=\mathrm{item}(\bar{t})\wedge\mathrm{type}(t_2)=T_1)$$
$$\supset((\mathrm{comp}(\bar{t})=\mathrm{comp}(\bar{t})\wedge\mathrm{item}(t_2')=\mathrm{item}(\bar{t})\wedge\mathrm{type}(t_2'=T_2))$$

$$\equiv(\forall t_2/R_2)((\mathrm{item}(t_2)=\mathrm{item}(\bar{t})\wedge\mathrm{type}(t_2)=T_1)$$
$$\supset(\exists t_2'/R_2)(\mathrm{type}(t_2')=T_2$$
$$\wedge((\exists t_1'/R_1)(\mathrm{comp}(t_1')=\mathrm{comp}(\bar{t})\wedge\mathrm{item}(t_1')=\mathrm{item}(t_2'))\vee\mathrm{item}(t_2')=\mathrm{item}(\bar{t}))))$$

is 'true'. There is no simpler form acting a necessary and sufficient condition for the properness of deleting a tuple $\bar{t}'$ from R. The sufficient condition,

$$(\forall t_1/R_1^-)\Pi^u\sim Q[\bar{t}'\rightarrow t_1']$$
$$\equiv(\forall t_1/R_1^-)(\forall t_2/R_2)\sim Q[\bar{t}'\rightarrow t_1']\equiv\sim(\exists t_1/R_1^-)(\exists t_2/R_2)Q[\bar{t}'\rightarrow t_1']$$
$$\equiv\sim(\ t_1/R_1^-)(\ t_2/R_2)(\mathrm{item}(t_1)\neq\mathrm{item}(t_2)\vee\mathrm{type}(t_2)\neq T_1$$
$$\vee(\mathrm{comp}(t_1)=\mathrm{comp}(\bar{t}')\wedge(\ t_2'/R_2)(\mathrm{item}(t_2')=\mathrm{item}(\bar{t}')\wedge\mathrm{type}(t_2')=T_2)))$$

being 'true' is useless in this case because it becomes 'false' whenever there exist one or more companies supplying different items, or there exists an item of any other type than T_1.

ASSERTION 5: "No company must supply two different departments with item I" can be written as

$$(\forall t_1/R_1)(\forall t_1'/R_1)((\mathrm{comp}(t_1)=\mathrm{comp}(t_1')\wedge\mathrm{item}(t_1)=I\wedge\mathrm{item}(t_1')=I)$$
$$\supset\mathrm{dept}(t_1)=\mathrm{dept}(t_1')).$$

There is only a slight difference from FDs. Adding a tuple $\bar{t}$ to R_1 can be validated by examinig whether

$$(\forall t_1/R_1)((\mathrm{comp}(t_1)=\mathrm{comp}(\bar{t})\wedge\mathrm{item}(t_1)=I\wedge\mathrm{item}(\bar{t})=I)$$
$$\supset\mathrm{dept}(t_1)=\mathrm{dept}(\bar{t}))$$

is 'true'. This becomes 'true' whenever $\mathrm{item}(\bar{t})\neq I$. Deleting a tuple $\bar{t}'$ from R_1 is always proper with respect to this assertion.

ASSERTION 6: "Wnenever an employee is a subordinate of another employee which is itself a subordinate of a third one then the first one is a subordinate of the latter one" can be written as

$$(\forall t_4/R_4)(\forall t_4'/R_4)(\exists t_4''/R_4)(\mathrm{mng}(t_4)=\mathrm{emp}(t_4')$$
$$\supset(\mathrm{emp}(t_4)=\mathrm{emp}(t_4'')\wedge\mathrm{mng}(t_4')=\mathrm{mng}(t_4''))).$$

Adding a tuple $\bar{t}$ to R_4 can be validated by examining whether all

$$(\forall t_4/R_4)((\exists t_4''/R_4)Q[\bar{t}\rightarrow t_4']\vee Q[\bar{t}\rightarrow t_4',t_4'']),$$
$$(\forall t_4'/R_4)((\exists t_4''/R_4)Q[\bar{t}\rightarrow t_4]\vee Q[\bar{t}\rightarrow t_4,t_4''])$$

and

$$((\exists t_4''/R_4)Q[\bar{t}\rightarrow t_4,t_4']\vee Q[\bar{t}\rightarrow t_4,t_4',t_4''])$$

are 'true'. If there exists a plausible assertion that

$$(\forall t_4/R_4)\mathrm{emp}(t_4)\neq\mathrm{mng}(t_4).$$

then the first and second terms become

$(\forall t_4/R_4)(mng(t_4)=emp(\bar{t})\supset(\exists t''_4/R_4)(emp(t''_4)=emp(t_4)\wedge mng(t''_4)=mng(\bar{t})))$

and

$(\forall t'_4/R_4)(emp(t'_4)=mng(\bar{t})\supset(\exists t''_4/R_4)(emp(t''_4)=emp(\bar{t})\wedge mng(t''_4)=mng(t'_4)))$

respectively. The third term is always 'true'. There is no simpler form acting a necessary and sufficient condition for deleting a tuple $\bar{t}'$ from R_4 being proper. It is obvious that the sufficient condition

$(\exists t_4/R_4^-)(\exists t'_4/R_4^-)Q[\bar{t}'\to t''_4]$

$\equiv(\exists t_4/R_4)(\exists t'_4/R_4)(emp(t_4)\neq mng(t'_4)\vee(emp(t'_4)=emp(\bar{t})\wedge mng(t'_4)=mng(\bar{t})))$

being 'false' is useless in this case.

ASSERTION 7: "There is at least one type T item which is supplied by every company" can be written as

$(\exists t_2/R_2)(\forall t_1/R_1)(\exists t'_1/R_1)$

$(type(t_2)=T\wedge comp(t_1)=comp(t'_1)\wedge item(t'_1)=item(t_2))$.

If t_1 and t'_1 are update-relevant, we must deduce theorem $\Pi\forall\exists$. We have

$\Pi(\forall t_1/R_1^+)(\exists t'_1/R_1^+)Q$

$\equiv\Pi((\forall t_1/R_1)((\exists t'_1/R_1)Q\vee Q[\bar{t}\to t'_1])\wedge((\exists t'_1/R_1)Q[\bar{t}\to t_1]\vee Q[\bar{t}\to t_1,t'_1]))$.

In this case, however, no simpler form acting a necessary condition, sufficient condition, or necessary and sufficient condition can be obtained for either adding a tuple $\bar{t}$ to R_1 or deleting a tuple $\bar{t}'$ from R_1 being proper with respect to this assertion.

As seen above a result equivalent to that obtained by Nicolas' method has been obtained for each assertion. Besides much more information has been obtained, in particular, regarding necessary conditions and sufficient conditions.

9 AGGREGATE CONSTRAINTS

In contrast to Nicolas' method which is based on the first order logic, the method presented here takes advantage of properties of domain-coupled quantifiers as aggregate functions. Let us remember the lemma 2 that states

$(\forall t/R^+)\equiv(\forall t/R)Q\wedge Q[\bar{t}\to t]$

and

$(\exists t/R^+)\equiv(\exists t/R)Q\vee Q[\bar{t}\to t]$.

These are equivalent to the properties of aggregate functions $\bigwedge$ and $\bigvee$, that is,

$$\bigwedge[t/R^+;]Q \equiv \bigwedge[t/R;]Q \wedge Q[\bar{t}\to t]$$

and

$$\bigvee[t/R^+;]Q \equiv \bigvee[t/R;]Q \vee Q[\bar{t}\to t].$$

A similar property is possessed by every aggregate operator. For example,

$$\Sigma[t/R^+;]f \equiv \Sigma[t/R;]f + f(\bar{t}),$$

$$\Pi[t/R^+;]f \equiv \Pi[t/R;]f \times f(\bar{t})$$

and

$$\max[t/R^+;]f \equiv \max(\max[t/R;]f, f(\bar{t}))$$

are properties of aggregate operators Σ, Π and max. For the average and standard deviation, from the definition that

$$\text{avg}[t/R;]f \equiv \Sigma[t/R;]f / \Sigma[t/R;]\text{count}$$

and

$$\sigma[t/R;]f \equiv (\Sigma(\text{avg}[t/R;]f - f)^2 / \Sigma[t/R;]\text{count})^{1/2}$$
$$\equiv (\Sigma[t/R;]f^2 / \Sigma[t/R;]\text{count} - (\text{avg}[t/R;]f)^2)^{1/2},$$

where count is a constant function which assigns 1 to every tuple, it is possible to obtain

$$\text{avg}[t/R^+;]f \equiv (\Sigma[t/R;]f + f(\bar{t})) / (\Sigma[t/R;]\text{count}+1)$$

and

$$\sigma[t/R^+;]f \equiv ((\Sigma[t/R;]f^2 + (f(\bar{t}))^2) / (\Sigma[t/R;]\text{count}+1)$$
$$-((\Sigma[t/R;]f + f(\bar{t})) / ((\Sigma[t/R;]\text{count}+1))^2)^{1/2}.$$

If the values of aggregate functions defined by operators such as $\Sigma[t/R;]f$, $\Sigma[t/R;]f^2$, $\Pi[t/R;]f$, $\max[t/R;]f$ and $\Sigma[t/R;]\text{count}$ are known in the current database, values of aggregate functions such as $\Sigma[t/R;]f$, $\Pi[t/R;]f$, $\max[t/R;]f$, $\text{avg}[t/R;]f$ and $\sigma[t/R;]f$ in the updated database can be calculated quickly before the database is actually updated. Such a technique can be applied to validating database updates for aggregate constraints like

$$P \equiv \Sigma[t/R;]A(t) < K$$

and

$$P \equiv \max[t/R;]A_1(t) - \min[t/R;]A_2(t) < K.$$

Like domain-coupled quantifiers, aggregate functions can appear in combination. For example, in

$$P \equiv (\forall t_1/R)(\Sigma[t_2/R; A_1(t_2)=A_1(t_1)]A_2(t_2) < K)$$

and

$P \equiv \Sigma[t_1/R_1;]\Sigma[t_2/R_2;A_2(t_2)=A_1(t_1)]A_3(t_2)<K$,
two aggregate functions appear. In statistical databases, aggregate constraints such as

$P \equiv (\forall t_1/R_1)(A_2(t_1)=\Sigma[t_2/R_2;A_3(t_2)=A_1(t_1)]A_4(t_2))$
are very popular. In these three examples, an efficient validation is possible if the value of second aggregate function is known for each tuple t_1 in R or R_1 in the current database.

10 TEMPORARY INCONSISTENCY

Let us next consider a series of unit updates called a "transaction." When a single update woald violate some assertion, one can reject this update. However, sometimes when this update is in a transaction, it may be executed and the generated "temporary inconsistency" is removed by one or more other updates executed suceedingly.

Two different cases can be considered. One is the case where the generation of temporary inconsistencies can be avoided by an appropriate execution sequence of unit updates in the transaction. Let us assume, for example, that an assertion

$P \equiv (\forall t_1/R_1)(\exists t_2/R_2)A_1(t_1)=A_2(t_2)$
corresponding to an inclusion constraint is given. If a tuple $\bar{t}_1$ is added to R_1, for which no tuples t_2 satisfying $A_1(\bar{t}_1)=A_2(t_2)$ exist, this assertion is violated. This temporary inconsistency is removed by adding a tuple $\bar{t}_2$ to R_2, for which $A_1(\bar{t}_1)=A_2(\bar{t}_2)$ is satisfied. It is obvious, however, this temporary inconsistency can be avoided by adding $\bar{t}_2$ to R_2 followed by adding $\bar{t}_1$ to R_1.

The other is the case where no execution sequences of unit updates that do not generate a temporary inconsistency exist. If given both

$P_1 \equiv (\forall t_1/R_1)(\exists t_2/R_2)A_1(t_1)=A_2(t_2)$
and

$P_2 \equiv (\forall t_2/R_2)(\exists t_1/R_1)A_1(t_1)=A_2(t_2)$,
then any update sequence generates a temporary inconsistency. Sometimes a single assertion is violated by any unit update. For example, an assertion

$P \equiv (\forall t_1/R_1)A_2(t_1) = \Sigma[t_2/R_2;A_3(t_2)=A_1(t_1)]A_4(t_2)$

is violated by any one of adding a tuple to R_1, deleting a tuple from R_1, adding a tuple to R_2 and deleting a tuple from R_2.

There are two ways of dealing with the latter case. One is to allow the user only to invoke built-in standard "collective update procedures" which include several unit updates that may generate temporary inconsistencies but are assured to restore the database into a consistent state before the control is returned to the user (user's program). Replacing a tuple can be provided as a collective update procedure. Provision of such collective update procedures are particularly desirable when a relationship constraint is defined, that is, there is a relationship relation among several other relations [2]. According to the type of relationship relation, which is also characterized by several other constraints determining the relationship relation type, various collective update procedures should be provided.

The other way is to provide a device by which the user direct the database mahagement system to postpone consistency checking until a series of unit updates has been completely executed. This can be achieved by declaring start and end of transaction.

When the end of transaction is reached, the database must be ascertained to be consistent. This can be achieved by evaluating values of all the registered assertions in the updated database. However, this is again a time-consuming task. It is only necessary to ascertain all the assertions which were violated by some unit updates in the transaction have made 'true' again by some other unit updates in the transaction. Furthermore, there can be the cases where it is not necessary to reevaluate assertion values in the updated database, that is, there exists some simpler from acting a necessary condition, a sufficient condition or a necessary and sufficient condition for the given assertion becoming 'true' in the database updated by the transaction. Such a form must include two or more tuples (constants) that were added to or deleted from the database.

For example, assume that a transaction adds a tuple $\bar{t}_1$ to R_1 forming R_1^+ and also adds a tuple $\bar{t}_2$ to R_2 forming R_2^+. For an assertion

$$P \equiv (\forall t_1/R_1)(\exists t_2/R_2)A_1(t_1)=A_2(t_2)$$

corresponding to an inclusion constraint, it can be seen that

$$(\forall t_1/R_1^+)(\exists t_2/R_2^+)Q \equiv (\forall t_1/R)((\exists t_2/R)Q \vee Q[\bar{t}_2 \to t_2])$$
$$\wedge((\exists t_2/R_2)Q[\bar{t}_1 \to t_1] \vee Q[\bar{t}_1 \to t_1, \bar{t}_2 \to t_2]).$$

Since the first term is always 'true', this transaction is proper with respect to P if and only if

$$(\exists t_2/R_2)Q[\bar{t}_1 \to t_1]$$

or

$$Q[\bar{t}_1 \to t_1, \bar{t}_2 \to t_2] \equiv A_1(\bar{t}_1) = A_2(\bar{t}_2)$$

is 'true'. It is very easy to check the second sufficient condition.

A generalized form of the lemma 2

LEMMA 2':

(1) $(\forall t/R_1 \cup R_2)Q \equiv (\forall t/R_1)Q \wedge (\forall t/R_2)Q$

(2) $(\exists t/R_1 \cup R_2)Q \equiv (\exists t/R_1)Q \vee (\exists t/R_2)Q.$

becomes necessary to develop a method applicable to more general cases.

11 CONCLUSION

Database updates should not conflict with any integrity constraints. Two types of integrity constraints, static constraints and dynamic constraints, exist. Update validation procedures for integrity constraints of the former type have been discussed.

A static constraint can be expressed by a relation schema calculus defined on database relation schemata. To each relation schema calculus, an assertion which is an extended (in the sense that any function of tuples can be used in defining it) relational calculus corresponds. A database is consistent if all the registered assertions are true in it. An update is S-proper if it transforms a consistent database into another consistent database.

The S-properness of an update can be examined by evaluating all the registered assertions in the updated database. However, this procedure is unnecessarily time-consuming. It is sufficient to evaluate only update-relevant assertions that can be falsified by the given update in the updated database.

Furthermore, in many cases there exist some simplified extended relational calculus, which can be evaluated in the current database using tuples (constants) to be added to and/or deleted from the database, and which (being true) acts a necessary condition, a sufficient condition, or an necessary and sufficient (equivalent) condition for an assertion (being true) in the updated database.

A method of finding such simplified forms for a given assertion and a given unit update (adding, deleting or replacing a single tuple) was presented first. This method is based on several basic properties of propositional logic and (many-sorted) predicate logic. The latter properties regard those of domain-coupled quantifiers as aggregate functions.

The method consists of two major steps: (1) Finding simplified forms by processing the prefix part of the given assertion (in prenex form), and (2) Further simplification by substituting tuple constants for some tuple variables in the matrix part of the given assertion. The result varies according to what the assertion form is and what update is to be applied to what relation (what the update-relevant variables are).

The time required for finding an efficient validation procedure by this method is less than $O(2^N \ell)$ where N is the number of update-relevant variables in the given assertion and ℓ is the length of the assertion formula. For most assertions N=1. There are very few cases where N>3. On the other hand, the time saved by applying the obtained procedure instead of directly eveluating the assertion can be $O(log\,n)$ to $O(n^2)$ or more, where n is the number of tuples in the relations on which the assertion is defined. It varies according to what search procedures can be used for evaluating the assertion P and the obtained simplified form P'. Since n can be fairly large, it seems quite safe to say that the proposed method is practical.

Some parts of the method can be extensively applied to validating database updates against aggregate constraints. Also it is possible to generalize the method for improving procedures of validating transactions.

ACKNOWLEDGEMENT

The author is greatly indebted to H. Katsuno, Musashino Laboratory of Nippon Telephone and Telegraph, and T. Miura, Mitsui Ship-building Co., for their fruitful discussions and suggestions on the topics described in this paer. This paper was presented at the Information System Modelling Symposium held at Institute of Mathematical Analysis of University of Kyoto, February 17-19, 1983.

[REFERENCES]

1. I.Kobayashi, Evaluation of Queries based on the Extended Relational Calculi, *Intn'l Jour. of Computer and Information Sciences,* 10 (2), pp. 63-103, 1981.
2. I.Kobayashi. On the Semantic Constraints and Normal Forms of Database Relations, *Intn'l Jour. of Policy and Information,* 6 (1), pp. 107-118, 1982.
3. P.A.Bernstein, and B.T.Blaustein, Fast Methods for Testing Quantified Relational Calculus Assertions, *Proc. ACM SIGMOD '82,* pp. 39-50, 1982.
4. G.Jaeschke, and H.-J.Schek, Remarks on the Algebra of Non First Normal Form Relations, *Proc. ACM SIGMOD-SIGACT Principles of Database Systems '82,* pp. 124-136, 1982.
5. F.Sadri, and J.D.Ullman, A Complete Axiomatization for a Large Class of Dependencies in Relational Databases, *Proc. ACM Symp. Theory and Computing '79,* pp. 117-122, 1979.
6. J.-M.Nicolas, Logic for Improving Integrity Checking in Relational Data Bases, *Acta Informatica,* 18, pp. 227-253, 1982.