Differential Relational Calculus for Integrity Maintenance

Levent V. Orman

Abstract—A differential calculus for first-order logic is developed to enforce database integrity. Formal differentiation of first-order sentences is useful in maintaining database integrity, since once a database constraint is expressed as a first-order sentence, its derivative with respect to a transaction provides the necessary and sufficient condition for maintaining integrity. The derivative is often much simpler to test than the original constraint since it maintains integrity differentially by assuming integrity before the transaction, and testing only for new violations. The formal differentiation requires no resolution search, but only substitution. It is more efficient than resolution-based approaches; and it provides a considerably more general solution than previous substitution-based methods since it is valid for all first-order sentences and with all transactions involving arbitrary collections of atomic changes to the database. It also produces a large number of sufficient conditions that are often less strict than those of the previous approaches; and it can be extended to accommodate many dynamic constraints.

Index Terms—Database integrity, integrity maintenance, relational database, first-order logic, formal differentiation.

1 INTRODUCTION

DATABASE integrity constraints are logical statements that define valid states of the database. They must be obeyed by the data at all times to ensure the validity of data. They are critical in maintaining the quality of data and preventing a variety of errors and abuses. Efficient maintenance of integrity is a critical problem since testing the validity of a large number of constraints, against a large database, and after each transaction that may change the database state is an expensive undertaking. Considerable efficiency can be gained by maintaining integrity differentially, by assuming integrity before the transaction, and testing for new violations only [10], [11]. The critical problem in differential integrity maintenance is deriving the test conditions for these new violations. These conditions are often considerably easier to test than the original constraint, but they are also considerably different from the original constraint, and their derivation is not trivial, adding to the cost of integrity maintenance. Moreover, these test conditions are different for every transaction and, although they could be derived and compiled ahead of time for anticipated transactions, ad hoc transactions have even more stringent efficiency requirements—since their test conditions have to be derived in real time after the transaction is received and before it can be authorized [8], [13].

There have been two general approaches to the derivation of differential integrity test conditions. The first approach involves studying the quantifier structure of the constraint, and substituting the update values of the transaction for some constraint variables [1], [6], [10], [18]. This approach is very efficient in deriving the test conditions, but it is severely restrictive. The general problem is the difficulty of finding a procedure for analyzing the structure of arbitrarily complex constraints and arbitrarily complex transactions. Consequently, each implementation has its own severe restrictions. Some fail to derive the test conditions for constraints with complex quantifier structures or for constraints with multiple occurrences of a relation. Some find only one test condition when there are alternative conditions that may be easier to test. For complex transactions with multiple updates, this approach either fails or requires repeated application of the substitution algorithm. It also often derives conditions that need to be tested after the execution of the transaction, requiring an expensive rollback if the integrity test fails. The second approach involves resolution theorem proving where, given the constraint is satisfied before the transaction and the database state is changed by a transaction, one needs to prove that the constraint continues to be satisfied [8], [16], [19]. This approach is very general. It derives a necessary and sufficient test condition for all first-order sentences that satisfy the domain independence requirement, and it can be used with arbitrary nonatomic transactions. This approach also finds some sufficient conditions that may be easier to test; and its tests can be applied before the transaction, eliminating the need for a rollback. The disadvantages of this approach are the need for resolution search that makes it less efficient than substitution, and the need to write complex transition axioms to describe transactions. Both of these problems are partially alleviated for anticipated transactions by performing these tasks ahead of time and compiling the results into update programs; but for ad hoc transactions, real time integrity maintenance remains a problem. A similar approach was described in [15] where a differential relational algebra was developed to compute the changes to a relational algebra expression. This approach is also very general since it...
applies to all first-order sentences satisfying the domain independence requirement, and its tests can be applied before the transaction is executed. However, it fails to take advantage of the incremental enforcement principle; i.e., no violations before the transaction. Consequently, some manual simplification is necessary to eliminate the subexpressions that return null; and no sufficient conditions can be derived utilizing the incremental violations assumption. Dynamic constraints and domain-dependent constraints are also excluded in this approach.

This article develops a differential relational calculus for integrity maintenance that combines the advantages of both approaches. It is based on substitution, requires no resolution search, and hence it is very efficient. It requires no complex transition axioms to describe the transactions, and combined with its efficiency, this property makes it an ideal method for ad hoc transactions. It is also very general since it derives necessary and sufficient conditions for all first-order sentences satisfying domain independence requirement; it can be used with arbitrary transactions involving multiple atomic updates; and its tests can be applied before the transaction is executed. It also finds a large number of sufficient conditions that are often less restrictive than the previous approaches. Moreover, it can be extended to deal with dynamic constraints and domain-dependent constraints, which are excluded from the previous approaches.

The article is organized in seven sections. Formal differentiation of first-order sentences is introduced in Section 2. In Section 3, the rules of differentiation are proved correct and some simplification rules are introduced since derivatives may contain redundancies and/or trivial tests. Section 4 provides detailed examples of the method developed. Section 5 derives sufficient conditions for integrity that are often easier to test than the necessary and sufficient conditions and can be used to quickly accept some transactions. Section 6 provides two major extensions by dropping the domain independence requirement so that all first-order sentences can be accommodated and by demonstrating how some dynamic constraints can be easily incorporated into the system. Section 7 summarizes the results, and provides a detailed comparison to previous work.

## 2 Differential Calculus

A relational database is a collection of relations $R_1, \ldots, R_n$ with degrees $d_1, \ldots, d_n$, where each relation $R_i$ is a collection of $d_i$-tuples [4]. A database transaction is a collection of insertions into and deletions from the database. It is represented as a collection of differential relations $\Delta R_1, \Delta R_2, \ldots, \Delta R_n$, where $\Delta R_i$ contains all the tuples to be inserted into $R_i$, and $\Delta R_i$ contains all the tuples to be deleted from $R_i$. For each database relation $R_i$ and its differential relations $\Delta R_i$ and $\Delta R_i$, the predicates $R_i, \Delta R_i$, and $\Delta R_i$ are defined as $d_i$-place predicates where $R_i(x), \Delta R_i(x)$, or $\Delta R_i(x)$ are true if and only if the $d_i$-tuple $x$ is in $R_i, \Delta R_i$, or $\Delta R_i$, respectively. A derived predicate $V(x)$, defined by an arbitrary first-order sentence $S$ with free variables $x$, is denoted by $V(x) \equiv S$. $V(x)$ is true if and only if $x$ satisfies $S$. Database relations are modified by transactions. Given a transaction, for each database relation $R_i$, let $R'_i$ be the new state of the relation $R_i$ after the transaction. Let $R'_i$ also denote the predicate corresponding to the relation $R'_i$. Let $S'$ be the sentence $S$ where each occurrence of the predicate $R_i$ is replaced with $R'_i$. For each derived predicate $V(x)$ defined by a sentence $S$, $V'(x)$ is defined by $S'(V'(x) \equiv S')$. $V'(x)$ corresponds to the value of the predicate $V(x)$ after the execution of the given transaction.

**Definition 2.1.** The derivative $\Delta V(x)$ of an arbitrary predicate $V(x)$ is defined as $\nabla(x) \land V'(x)$. In other words, for all $x$, $\Delta V(x)$ is true if and only if $V(x)$ is false and $V'(x)$ is true, i.e., $x$ satisfies the derivative if and only if $x$ does not satisfy the predicate before the transaction, but satisfies it after the transaction.

**Definition 2.2.** The deletion derivative $\Delta V(x)$ of an arbitrary predicate $V(x)$ is defined as $V(x) \land \nabla(x)$. In other words, for all $x$, $\Delta V(x)$ is true if and only if $V(x)$ is true and $V'(x)$ is false; i.e., $x$ satisfies the deletion derivative if and only if it satisfies the predicate before the transaction, but does not satisfy it after the transaction.

A database constraint violation is a derived predicate $V$ defined by an arbitrary closed first-order sentence $S$ [5]. $V$ implies a violation of database integrity. It is the negation of the common database constraint concept, and it will be used throughout this article. Given a transaction, the derivative $\Delta V$ of $V$ with respect to that transaction is a closed first-order sentence that is the necessary and sufficient condition for a violation of integrity after the transaction, assuming that no violation exists before the transaction. In other words, $V \leftrightarrow (\Delta V \leftrightarrow V')$. The proof follows immediately from the definition of derivatives. Since $\Delta V \leftrightarrow (\nabla(x) \land V')$ from Definition 2.1, then $\nabla(x) \leftrightarrow (\Delta V \leftrightarrow V')$. $\Delta V$ is useful in maintaining integrity incrementally, since given no violations before the transaction, $\Delta V$ is the necessary and sufficient condition for a violation after the transaction, and its negation $\Delta V$ is the necessary and sufficient condition for no violations after the transaction, i.e., integrity maintenance.

Formal differentiation of first-order sentences will be done in two steps:

1) The first step involves rewriting the sentence in a new normal form called “differential normal form,” containing no explicit quantifiers, with all variables assumed to be existentially quantified, and capturing quantifier scopes explicitly through intermediate derived predicates.

2) The second step involves applying the formal rules of differentiation to the normalized sentences.

Normalization ensures a minimal set of differentiation rules and minimal need for further simplification of the derivatives.
2.1 Differential Normal Form

Differential normal form of first-order logic relies on intermediate derived predicates to capture quantifier scopes explicitly, since the quantifiers scopes are a critical factor in the formal differentiation process. Moreover, all universal quantifiers are eliminated and replaced with negated existential quantifiers, and all existential quantifiers are eliminated since all nonfree variables are assumed to be existentially quantified. The remaining expressions consist only of atoms connected with AND and OR, with no quantifiers and no function symbols as in Datalog [4]. The scopes of negations and quantifiers, and the free variables are captured through intermediate derived predicates. The intermediate predicates are defined using the \( \equiv \) operator. Finally, the notation is simplified by denoting AND by “ \( \land \) ”, OR by “ \( \lor \) ”, negation by “ \( \neg \) ”. Formally, \( V(x) = P(x, y) \) where \( P(x, y) \) is an arbitrary first-order sentence in differential normal form with variables \( x, y \). It is interpreted to mean \( \forall x(V(x) \iff 3yP(x, y)) \) in standard notation where \( \forall x \) and \( \exists y \) are shorthand for \( \forall x_1, \ldots, \forall x_n \) and \( \exists y_1, \ldots, \exists y_n \) if \( x = (x_1, \ldots, x_n) \) and \( y = (y_1, \ldots, y_n) \). \( P(x, y) \) is in differential normal form and, hence, it consists of only atoms connected with “ \( \land \) ” for AND and “ \( \lor \) ” for OR.

Example 2.1. Given the predicate SUPPLY(\( x, y \)) indicating that the supplier \( x \) supplies the item \( y \); a constraint violation \( V \) prohibits a supplier from supplying both \( B \) and \( C \):

\[ V \equiv \text{SUPPLY}(x, B) \land \text{SUPPLY}(x, C) \]

which is equivalent to the first-order sentence

\[ V \iff 3x(\text{SUPPLY}(x, B) \land \text{SUPPLY}(x, C)) \]

where \( V \) indicates an integrity violation if a supplier \( x \) supplies both of items \( B \) and \( C \).

Given \( \Delta \text{SUPPLY}(A, B) \) i.e., \( (A, B) \) is inserted into SUPPLY; the derivative \( \Delta V \) will be computed through differentiation as FALSE, which implies that inserting \( (A, B) \) into SUPPLY cannot violate this constraint.

2.2 Rules of Differentiation

Given a derived predicate \( V(x) \), its derivatives \( \Delta V(x) \) and \( \nabla V(x) \) are computed recursively as follows assuming that no tuple is both inserted into and deleted from the same database relation (formal proofs will be given in Section 3):

a) if \( V(x) \equiv S \) for an arbitrary positive atom \( S \), then

\[ \Delta V(x) = \Delta S \quad \text{if } S \text{ has no variables other than } x, \]

\[ \nabla V(x) = \nabla \Delta S \quad \text{otherwise.} \]

b) if \( V(x) = \exists T \) for an arbitrary sentence \( T \), then

\[ \Delta V(x) = \nabla T \quad \text{if } T \text{ has no variables other than } x, \]

\[ \nabla V(x) = \nabla \exists T \quad \text{otherwise.} \]

c) if \( V(x) = ST \) for arbitrary sentences \( S \) and \( T \), then

\[ \Delta V(x) = S \nabla T \Delta S, T \nabla T \Delta S, \Delta S \Delta T \]

if \( S, T \) has no variables other than \( x \),

\[ \nabla V(x) = \Delta T \nabla S, \Delta S \nabla T \]

otherwise.

d) if \( V(x) \equiv S, T \) for arbitrary sentences \( S \) and \( T \), then

\[ \Delta V(x) = \nabla \exists S, \nabla \exists T \]

if \( S, T \) has no variables other than \( x \),

\[ \nabla V(x) = \Delta S \nabla T, \Delta T \nabla S \]

otherwise.

e) Finally, for all database predicates \( S \),

\[ S' \equiv S \nabla S, \Delta S \]

Example 2.2. Given the predicates SUPPLY(\( x, y \)), as above, and SUPPLIER(\( x \)) indicating that \( x \) is a supplier, a constraint violation prohibits suppliers from supplying nothing:

\[ V \equiv \text{SUPPLIER}(x) \land \overline{M(x)} \]

where \( V \) indicates an integrity violation if a supplier \( x \) supplies nothing.

Given \( \Delta \text{SUPPLY}(A, B) \) i.e., \( (A, B) \) is inserted into SUPPLY; the derivative \( \Delta V \) will be computed through differentiation as FALSE, which implies that inserting \( (A, B) \) into SUPPLY cannot violate this constraint.

Example 2.3. Given the predicate SUPPLY(\( x, y \)) indicating that the supplier \( x \) supplies the item \( y \); a transaction inserting \( (A, B) \) into SUPPLY; and a constraint \( V \) prohibiting a supplier from supplying both \( B \) and \( C \):

\[ V \equiv \text{SUPPLY}(x, B) \land \text{SUPPLY}(x, C) \]

from the premise
\[ \Delta \text{SUPPLY}(x, y) \equiv (x, y) = (A, B) \]

from the premise

\[ \Delta V = \overline{V} (\text{SUPPLY}(x, B)) \overline{V} \text{SUPPLY}(x, B) \Delta \text{SUPPLY}(x, C), \]
\[ \text{SUPPLY}(x, C) \overline{V} \text{SUPPLY}(x, C) \Delta \text{SUPPLY}(x, B), \]
\[ \Delta \text{SUPPLY}(x, B) \Delta \text{SUPPLY}(x, C) \]

by using Rule c

\[ = \text{TRUE} (\text{SUPPLY}(x, B)) \text{TRUE} \text{FALSE}, \]
\[ \text{SUPPLY}(x, C) \text{TRUE} (x, B) = (A, B), \]
\[ (x, B) = (A, B) \text{FALSE} \]
\[ = \text{TRUE} (\text{FALSE}, \text{SUPPLY}(x, C)) (x, B) = (A, B), \text{FALSE} \]
\[ = (\text{SUPPLY}(x, C)) x = A \]
\[ = \text{SUPPLY}(A, C) \]

since \( \overline{V} \text{SUPPLY}(x, B) \) and \( \overline{V} \text{SUPPLY}(x, C) \) are FALSE when no deletions are indicated, and their negations are TRUE. \( \Delta \text{SUPPLY}(x, C) \) is FALSE since there are no insertions involving the item C. \( \overline{V} \) is true from the assumption of no violations before the transaction. The derivative suggests that a violation is introduced by the transaction if and only if \((A, C)\) is in SUPPLY.

Similarly, an insertion of \((A, B)\) and \((A, C)\) into SUPPLY (i.e., \( \Delta \text{SUPPLY}(A, B), \Delta \text{SUPPLY}(A, C) \)) for the same constraint leads to:

\[ \Delta V = \overline{V} (\text{SUPPLY}(x, B)) \overline{V} \text{SUPPLY}(x, B) \]
\[ \overline{V} \text{SUPPLY}(x, C) \Delta \text{SUPPLY}(x, C), \]
\[ \Delta \text{SUPPLY}(x, C) \overline{V} \text{SUPPLY}(x, C) \]

from Rule c

\[ = \text{TRUE} (\text{SUPPLY}(x, B)) \text{TRUE} (x = A), \]
\[ \text{SUPPLY}(x, C) \text{TRUE} (x, A), \]
\[ (x = A)(x = A) \]
\[ = \text{TRUE} (\text{SUPPLY}(A, B), \text{SUPPLY}(A, C), \text{TRUE}) \]
\[ = \text{TRUE} \]

since \( \Delta \text{SUPPLY}(x, C) \) is TRUE for \( x = A \), \( \Delta \text{SUPPLY}(x, B) \) is TRUE for \( x = A \), \( \overline{V} \text{SUPPLY}(x, B) \) and \( \overline{V} \text{SUPPLY}(x, C) \) are FALSE since no deletions are indicated, and their negations are TRUE, \( \overline{V} \) is TRUE from the assumption of no violation before the transaction, and finally \((x = A)\) is TRUE since there is always an \( x \) equal to A (namely A). The derivative \( \Delta V = \text{TRUE} \) indicates that the transaction always leads to a violation unconditionally.

EXAMPLE 2.4. Given the predicates \( \text{SUPPLY}(x, y) \) as above, and \( \text{SUPPLIER}(x) \) indicating that \( x \) is a supplier; also, given a transaction inserting \((A, B)\) into SUPPLY and a constraint prohibiting suppliers from supplying nothing:

\[ V = \text{SUPPLIER}(x) \overline{M}(x) \]

a supplier \( x \) not in \( M \) causes a violation.

\[ M(x) = \text{SUPPLY} (x, y) \]

\[ x \text{ is in } M \text{ if supplies something.} \]
\[ \Delta \text{SUPPLY}(x, y) \equiv (x, y) = (A, B) \]

\[ \Delta V = \overline{V} (\text{SUPPLIER}(x)) \overline{V} \text{SUPPLIER} (x) \text{VM}(x), \]
\[ \overline{M}(x) \Delta \overline{M}(x) \Delta \text{SUPPLIER}(x), \]
\[ \text{SUPPLIER}(x) \text{VM}(x) \]

using Rule c,

and since \( \Delta \overline{M}(x) \equiv \Delta A(x) = \overline{V}M(x) \)

where \( A(x) = \overline{M}(x) \)

using Rule b.

\[ = \text{TRUE} (\text{SUPPLIER}(x)) \text{VM}(x), \]
\[ \text{FALSE, FALSE} \]

Since \( \Delta \text{SUPPLIER}(x) \) is FALSE, i.e., no insertions into SUPPLIER, and \( \overline{V} \) is true from the assumption of no violations before the transaction,

\[ = \text{SUPPLIER}(x) \overline{M}(x) \overline{V} \text{SUPPLY}(x, y) \]

using Rule a,

\[ = \text{SUPPLIER}(x) \overline{M}(x) \text{FALSE} \]

since \( \overline{V} \text{SUPPLY}(x, y) \) is FALSE, i.e., no deletions from SUPPLY,

\[ = \text{FALSE} \]

No violations can occur since \( \Delta V \) is always FALSE.

3 FORMAL DEVELOPMENT

This section will formally introduce the differential normal form of first-order logic, and the conversion algorithm from prenex disjunctive normal form to the differential normal form. The conversion algorithm will be proved correct with respect to the definitions. The formal rules of differentiation will be proved correct utilizing the differential normal form and the definitions. Finally, some optional simplification rules will be introduced to further simplify the necessary and sufficient conditions obtained by differentiation. These latter rules follow easily from first-order logic, and they are not essential to the differentiation process.

3.1 Differential Normal Form

The basic building block of an expression in differential normal form is:

\[ V(x) \equiv S \]

where \( V \) is a derived \( n \)-place predicate, \( x \) is a \( n \)-tuple of variables, and \( S \) is an arbitrary collection of atoms connected with \( \land \) for AND, and \( \lor \) for OR. \( S \) may contain other derived predicates but not \( V \), or any other predicate derived using \( V \). Its semantics is formally defined as:

\[ \forall x V(x) \text{ iff } \exists y S \]

where \( y \) are the variables of \( S \) that are not in \( x \). An expression in differential normal form is merely a collection of these building blocks that define derived predicates. Universal quantifiers are captured as negated derived predicates since all variables are assumed to be existentially quantified and scope information is captured in the nesting of predicate definitions. In other words, differential normal form expres-
sions are logic programs with disjunction, but with no recursion or cycles [4], [5]. They are used to express constraints, not to extend the database with deductive capabilities. The following algorithm is used to convert prenex disjunctive normal form sentences to the differential normal form.

**Algorithm 3.1.** Given an arbitrary prenex disjunctive normal form sentence $S$ except all $\forall$ have been converted to $\neg\exists$:

a) Define a derived predicate $V(x)$ as:

$$V(x) \equiv S$$

where $x$ are the free variables of $S$.

b) For every derived predicate $M(x)$, convert

$$M(x) \equiv \neg P$$

to

$$M(x) \equiv \neg N(x)$$

and

$$M(x) \equiv \exists y P$$

to

$$M(x) \equiv N(x, y)$$

where $P$ is an arbitrary expression, and $N(x)$ and $N(x, y)$ are new predicates.

c) Simplify notation by converting $\land$ to space, $\lor$ to $\neg \land$, and negation to bar. Simplify expressions by eliminating all parentheses since the scopes will be captured by the nesting of derived predicates, and AND will have precedence over OR. Eliminate redundant derived predicates by converting:

$$M(x)N(x, y)$$

$$N(x, y) \equiv S$$

to

$$M(x) \equiv S$$

for derived predicates $M$ and $N$, and an arbitrary expression $S$. Optionally, further simplify using the simplification rules of Section 3.3.

**Example 3.1.** Given the predicate $SUPPLY(x, y)$ indicating that the supplier $x$ supplies the item $y$; and a constraint prohibiting a supplier from supplying both items $B$ and $C$; applying the algorithm to convert to the differential normal form:

a) $V \equiv \exists x \ (SUPPLY(x, B) \land SUPPLY(x, C))$

b) $V \equiv M(x)$

$$M(x) \equiv SUPPLY(x, B) \land SUPPLY(x, C)$$

c) $V \equiv SUPPLY(x, B) \land SUPPLY(x, C)$

**Example 3.2.** Given the predicate $SUPPLIER(x)$ indicating $x$ is a supplier, $SUPPLY(x, y)$ as above, and a constraint prohibiting a supplier from supplying nothing; applying the algorithm, an expression in differential normal form is obtained as follows:

a) $V \equiv \exists x \ \forall y (SUPPLIER(x) \land \neg SUPPLY(x, y))$

$$V \equiv \exists x \ \exists y (SUPPLIER(x) \lor SUPPLY(x, y))$$

b) $V \equiv M(x)$

$$M(x) \equiv \neg N(x)$$

$$N(x) \equiv P(x, y)$$

$$P(x, y) \equiv \neg SUPPLIER(x) \lor SUPPLY(x, y)$$

c) $V \equiv N(x)$

$$N(x) \equiv SUPPLIER(x), SUPPLY(x, y)$$

or equivalently

$$V \equiv SUPPLIER(x) \land \neg N(x)$$

$$N(x) \equiv SUPPLY(x, y)$$

(see simplification Rule d1 of Section 3.3 for this last step).

**Proposition 3.1.** Conversion algorithm is correct. Proof follows from the formal semantics of differential normal form expressions.

### 3.2 Differentiation Rules

Transactions modify the extensions of database relations. Given a transaction, for each database relation $R$, let $R' \equiv S$ be the new state of the relation $R$ after the transaction. The changes in the state of $R$ can only result from insertions into and deletions from $R(\Delta R \lor VR)$. The new state $R'$ of $R$ contains all tuples that were in $R$ before the transaction and not deleted, and all tuples that were inserted by the transaction. $R'$ can then be defined as $R \land \neg \Delta R \lor VR$ using the differential relations. Similarly, given an arbitrary predicate $V$ defined by a sentence $S (V \equiv S)$, let the new value of the predicate after the transaction be $V'$, and let $S'$ be the sentence $S$ with each occurrence of a database relation $R$ replaced with $R' (V' \equiv S')$. $\Delta V$ is defined as $\neg V \lor V'$, and $\Delta VV$ is defined as $\neg \Delta V \lor \Delta V$.

**Assumption 3.1.** No data tuple is both inserted into and deleted from the same database relation within the same transaction. This assumption is not essential for differentiation, however, it leads to some simplification in the rules, and it is relatively easy to check within a transaction. Moreover, the semantics of a simultaneous insertion and deletion of the same data tuple within the same relation is complex. It is not clear whether the insertion or the deletion has the precedence without further clarification. The assumption implies $\Delta VR \lor R = FALSE$ for all database relations. It also implies that $\Delta VR = \Delta R \lor \Delta VR \land VR \equiv VR$. The proof follows immediately from the assumption since:

$$\Delta VR = \Delta VR \land VR \equiv FALSE$$

$$\Delta VR = \Delta VR \lor VR \equiv \Delta R \land VR \equiv \Delta R$$

$$VR \Delta R = VR \Delta R \lor VR \equiv FALSE$$

$$VR \Delta R = VR \Delta R \lor VR \equiv VR \Delta R \land VR \equiv VR \land VR \equiv VR$$

**Assumption 3.2.** The integrity is maintained incrementally. It is assumed that no violations exist before a transaction takes place (i.e., $\neg V$ is TRUE). A violation may or may not result from the transaction (i.e., $V'$ may or may not be TRUE).

**Lemma 3.1.** Given $V(x) \equiv S(y)$ where $S(y)$ is an arbitrary sentence with variables $y$, then

$$\nabla(x) \equiv S(y)$$

iff $y \subseteq x$

$$\nabla(x) \equiv \nabla(x) \land S(y)$$

otherwise.
PROOF. 
\[ V(x) \equiv S(y) \]
\[ \forall x(V(x) \iff \exists pS(y)) \]
\[ \forall x(\nabla(x) \iff \forall pS(y)) \]
\[ \forall x(\nabla(x) \iff \exists pS(y)) \]
\[ \nabla(x) \equiv S(y) \]

If \( y - x \) is not empty,
\[ \forall x(\nabla(x) \iff \forall pS(y)) \]
\[ \forall x(\nabla(x) \iff \exists pS(y)) \]
\[ \nabla(x) \equiv S(y) \]

Similar conditions hold for and \( V(x) \) and \( S(y) \). \( \square \)

Lemmma 3.2. \( V(x) \iff V(x) \nabla V(x) \), \( \Delta V \) for all predicates \( V \).

PROOF. 
\[ V \iff V \nabla V \]
\[ \nabla V \iff V(x) \nabla \nabla V(x) \], \( \nabla V \iff V(x) \nabla V(x) \), \( \nabla V \iff V(x) \nabla V(x) \)
by using the definitions of \( \Delta V \) and \( \nabla V \) from Definitions 2.1 and 2.2, the identities \( (V, \nabla V) = \text{TRUE} \) and \( \nabla V \)
\( G \equiv \text{FALSE} \), and the law of negation \( (AB) = A, B \).

Theorem 3.1. Differentiation rules are correct. They compute the
derivatives correctly as defined in Definition 2.1 and 2.2
in terms of database predicates and transactions.

PROOF. 

a) \( V(x) = S \)

for an arbitrary positive
atom \( S \),
\[ \Delta V(x) = \nabla V(x) = \nabla S \equiv \Delta S \]
if \( S \) has no variables other
than \( x \), from Definition
2.1 and Lemma 3.1.
\[ \Delta V(x) = \nabla V(x) = \nabla \nabla S \equiv \Delta S \]
otherwise, from Definition
2.1 and Lemma 3.1.
\[ \nabla V(x) = \nabla V(x) \nabla V(x) \equiv \nabla V(x) \nabla V(x) \]
if \( S \) has no variables other
than \( x \), from Definition
2.2 and Lemma 3.1.
\[ \nabla V(x) = \nabla V(x) \nabla V(x) \equiv \nabla V(x) \nabla V(x) \]
otherwise, from Definition
2.2 and Lemma 3.1.

b) \( V(x) = S \)

for an arbitrary sentence \( S \),
\[ \Delta V(x) = \nabla V(x) = \nabla S \equiv \nabla S \]
if \( S \) has no variables other
than \( x \), from Definitions
2.1, 2.2, and Lemma 3.1.
\[ \Delta V(x) = \nabla V(x) = \nabla \nabla S \equiv \nabla V(x) \nabla V(x) \]
otherwise, from Definitions
2.1 and 2.2, and Lemma 3.1.
\[ \nabla'(x) (T \Delta T \forall S, \Delta \forall S \forall T, \forall S \forall T) \]
otherwise, from the previous result and Lemma 3.1.
\[ S' = S \forall S, \Delta S \]
for all database predicates. It follows immediately from Lemma 3.2. \( \square \)

**Corollary 3.1.** For a given database relation \( R \), \( \Delta R \) can contain tuples that are in \( R \), and \( \nabla R \) can contain tuples that are not in \( R \), without affecting the value of \( V \) for any derived predicate \( V \). The corollary is useful since it eliminates the need to check that the tuples to be inserted do not already exist in the database, or the tuples to be deleted are already nonexistent. The proof follows immediately from Definition 2.1 since \( \Delta V \) contains only new violations, and the updates that do not change the database state cannot cause new violations.

### 3.3 Optional Simplification Rules

The expressions obtained through differentiation are not always in the simplest form. They can be further simplified using the equivalences of first-order logic but this simplification is not essential to the process of differentiation, and it can and should ordinarily be left to the query processor. The following is a list of simplification rules that are most useful within the notation of differential normal form. All the rules follow easily from the propositional logic except the rules of negation that require predicate logic \([3]\). The \( \leftrightarrow \) symbol is used to indicate logical equivalence. Formally, \( A \leftrightarrow B \) means \( V \equiv A \) is equivalent to \( V \equiv B \) for any derived predicate \( V \).

a) **Identity, TRUE (T), and FALSE, (F) rules**

\[
\begin{align*}
a1) \quad & AA \leftrightarrow A \\
a2) \quad & A, A \leftrightarrow A \\
a3) \quad & \overline{T} \leftrightarrow F \\
& AF \leftrightarrow F \\
& A, F \leftrightarrow A \\
& \overline{A} \leftrightarrow A \\
& A \overline{A} \leftrightarrow F \\
& A, \overline{A} \leftrightarrow T
\end{align*}
\]

They follow directly from the definitions of AND, OR, and NOT in propositional logic where \( A \) is an arbitrary expression.

b) **Subsumption Rules:**

\[
\begin{align*}
b1) \quad & A, AB \leftrightarrow A \\
b2) \quad & \overline{A} \overline{B} \leftrightarrow \overline{A} \quad \text{if } B \leftrightarrow AC \text{ for arbitrary sentences } A, B, C.
\end{align*}
\]

**Proof.** This follows immediately from the negation laws of propositional logic and since \( A \land B \) implies \( A \), for arbitrary \( A \) and \( B \). \( \square \)

c) **Resolution Rules:**

\[
\begin{align*}
c1) \quad & AB, \overline{A}C \leftrightarrow AB, \overline{A}C, BC \\
c2) \quad & \overline{A} \overline{B} \leftrightarrow \overline{C} \quad \text{if } A \leftrightarrow CD \text{ and } B \leftrightarrow C \overline{D}
\end{align*}
\]

**Proof.** This follows from the negation and distribution laws of propositional logic for any \( A, B, \) and \( C \). \( \square \)

d) **Negation Rules:**

\[
\begin{align*}
d1) \quad & V(x) \equiv A(y), B(z) \text{ is equivalent to } \overline{V}(x) \equiv \overline{M}(x) \overline{N}(x) \text{ where } M(x) \equiv A(y) \text{ and } N(x) \equiv B(z). \\
d2) \quad & V(x) \equiv A(y)B(z) \text{ is equivalent to } \overline{V}(x) \equiv \overline{M}(x) \overline{N}(x) \text{ where } M(x) \equiv A(y) \text{ and } N(x) \equiv B(z) \text{ if } x \supseteq y \land z.
\end{align*}
\]

**Proof.**

\[
\begin{align*}
d1) \quad & \forall x \left( V(x) \text{ iff } \exists q(A(y) \lor B(z)) \right) \\
& \quad \text{where } q = y \lor z - x \text{ from definition of derived predicates,} \\
& \forall x \left( V(x) \text{ iff } \exists q(A(y) \lor B(z)) \right) \\
& \quad \text{by distributing } \exists \text{ over } \lor, \\
& \forall x \left( \neg V(x) \text{ iff } \neg \exists q(A(y) \land \neg B(z)) \right) \\
& \quad \text{by negating both sides}
\end{align*}
\]

\[
\begin{align*}
d2) \quad & V(x) \equiv A(y)B(z) \\
& \quad \text{is given} \\
& \forall x \left( V(x) \text{ iff } \exists q(A(y) \lor B(z)) \right) \\
& \quad \text{where } q = y \lor z - x \text{ by eliminating irrelevant quantifiers,} \\
& \forall x \left( \neg V(x) \text{ iff } \neg M(x) \land \neg N(x) \right) \\
& \quad \text{where } M(x) \equiv \exists q_1 A(y) \text{ and } N(x) \equiv \exists q_2 B(z), \\
& \overline{V}(x) \equiv \overline{M}(x) \overline{N}(x) \\
& \quad \text{where } M(x) \equiv A(y) \text{ and } N(x) \equiv B(z) \text{ from definitions and notation.}
\end{align*}
\]

\[
\begin{align*}
& \forall x \left( V(x) \text{ iff } \exists q(A(y) \land \exists q(z)(B(z)) \right) \\
& \quad \text{by distributing } \exists \text{ over } \land, \\
& \forall x \left( V(x) \text{ iff } \exists q(A(y) \land \exists q(z)(B(z)) \right) \\
& \quad \text{where } q_1 = y - x \text{ and } q_2 = z - x, \text{ by eliminating irrelevant quantifiers,} \\
& \forall x \left( \neg V(x) \text{ iff } \neg M(x) \lor \neg N(x) \right) \\
& \quad \text{by negating both sides} \\
& \forall x \left( \neg V(x) \text{ iff } \neg M(x) \lor \neg N(x) \right) \\
& \quad \text{where } M(x) \equiv \exists q_1 A(y) \text{ and } N(x) \equiv \exists q_2 B(z),
\end{align*}
\]

\[ \nabla (x) = M(x), \nabla (x) \]

where \( M(x) = A(y) \) and \( N(x) = B(z) \) from definitions.

e) Substitution Rules:

e1) \( V(x) = S(x) \) is equivalent to \( V(y) = S(x/y) \)

where \( S(x/y) \) is the sentence \( S \) with \( y \) substituted for \( x \).

e2) \( V(x) = S \) is equivalent to \( V(x) = S, S(x/A) \)

where \( S(x/A) \) is the sentence \( S \) with \( A \) substituted for \( x \).

**Proof.** This follows immediately from definitions, and since \( S(x/A) \) implies \( 3x5S \) for an arbitrary sentence \( S \) with free variables \( x \).

### 4 Examples

Given the predicates \( SALE(x, y) \) indicating that the department \( x \) sells the item \( y \), \( SUPPLY(z, x, y) \) indicating that supplier \( z \) supplies the department \( x \) with the item \( y \), and \( CLASS(y, w) \) indicating that the item \( y \) belongs to the class \( w \); the following constraints and transactions are used to demonstrate the formal differentiation process to generate necessary and sufficient conditions for integrity maintenance. The derivative \( \Delta V \) is the necessary and sufficient condition for a violation of the integrity constraint \( V \), and its negation \( \nabla V \) is the necessary and sufficient condition for continued maintenance of integrity. The results are often not in the simplest form, and they are simplified using the simplification rules of Section 3, but simplification of expressions is not an essential part of the differentiation process. Simplification is shown merely to compare the results to the previous work. All that is required for integrity maintenance is direct substitution into differentiation rules, and execution of the derivative \( \Delta V \) (or \( \nabla \Delta V \)) using a query processor. Any data returned by the query processor in response to \( \Delta V \) is a violation of integrity. Null return is a certification of integrity. More efficiently, any data returned in response to \( \nabla \Delta V \) is a certification of integrity. Null return indicates a violation. Throughout the examples, variables are shown in small letters and constants in capital letters.

Note that the substitution of constants is not essential to the differentiation process, the derivatives can be generated using variables for transactions and compiled into update programs where the substitution of transaction constants can be done at the execution time. Alternatively, the complete differentiation process can be done at the execution time with constants substituted since the process is very efficient, involving only substitution, using the rules of differentiation:

a) A constraint prohibits departments from selling items that are not supplied, and a transaction inserts \((A, B)\) into the relation \( SALE: \)

\[ V \equiv SALE(x, y) \quad T(x, y) \equiv SUPPLY(z, x, y) \]

is the given constraint

\[ \Delta SALE(x, y) \equiv (x, y) = (A, B) \]

is the given transaction

\[ \Delta V = V(SALE(x, y) \nabla SALE(x, y) \nabla T(x, y), \]

\[ T(x, y) \nabla T(x, y) \nabla \Delta Sale(x, y), \]

\[ \Delta (SALE(x, y) \nabla T(x, y)) \]

from Differentiation Rule c

from Assumption 3.2, and the given transaction

\[ \nabla V = T(A, B) = \]

indicating that integrity is maintained if there is a supplier supplying \( A \) with \( B \).

b) The same constraint as in a, and a transaction deletes \((A, B, C)\) from \( SUPPLY: \)

\[ V \equiv SALE(x, y) \quad T(x, y) \equiv SUPPLY(z, x, y) \]

is the given constraint

\[ \nabla SUPPLY(z, x, y) \equiv (z, x, y) = (A, B, C) \]

is the given constraint

\[ \Delta V = \nabla (SALE(x, y) \nabla SALE(x, y) \nabla VT(x, y), \]

\[ T(x, y) \nabla T(x, y) \nabla \Delta Sale(x, y), \]

\[ \Delta (SALE(x, y) \nabla VT(x, y)) \]

from Differentiation Rules b and c

\[ \equiv SALE(x, y) \nabla VT(x, y) \]

from Assumption 3.2, and the given transaction

\[ \equiv SALE(x, y) \nabla T(x, y) \nabla SUPPLY(z, x, y) \]

from Differentiation Rule a

\[ \equiv SALE(B, C) \nabla T(B, C) \]

from the given constraint

\[ \equiv \nabla SALE(B, C), \]

\[ \nabla SUPPLY(z, B, C) \nabla SUPPLY(z, B, C), \]

\[ \nabla SUPPLY(z, B, C), \]

From Rule e,

\[ \equiv SALE(B, C), \]

\[ \nabla SUPPLY(z, B, C)(z, B, C) \neq (A, B, C) \]

\[ \equiv SALE(B, C), \]

\[ \nabla SUPPLY(z, B, C)z \neq A \]

indicating that integrity is maintained if \((B, C)\) is not in \( SALE \), or there is a supplier other than \( A \) supplying \( B \) to \( C \).

c) The same constraint as in a; and a transaction deletes \((A, B)\) and inserts \((A, C)\) into \( SALE \) (i.e., modifies \((A, B)\) to \((A, C)\)):

\[ V \equiv SALE(x, y) \quad T(x, y) \equiv SUPPLY(z, x, y) \]

the given constraint

\[ \nabla SALE(x, y) \equiv (x, y) = (A, B) \]

\[ \equiv SALE(x, y) \quad T(x, y) \equiv SUPPLY(z, x, y) \]

is the given constraint

\[ \Delta SALE(x, y) \equiv (x, y) = (A, B) \]

is the given transaction

\[ \Delta V = \nabla (SALE(x, y) \nabla SALE(x, y) \nabla VT(x, y), \]

\[ T(x, y) \nabla T(x, y) \nabla \Delta Sale(x, y), \]

\[ \Delta (SALE(x, y) \nabla VT(x, y)) \]

from Differentiation Rule c

from Assumption 3.2, and the given transaction

\[ \nabla V = T(A, B) = \]

indicating that integrity is maintained if there is a supplier supplying \( A \) with \( B \).
\[ \Delta \text{SALE}(x, y) = (x, y) = (A, C) \]  
the given transaction

\[ \Delta V \equiv \nabla (\text{SALE}(x, y) \nabla \text{SALE}(x, y)) \nabla T(x, y) \Delta T(x, y) \Delta \text{SALE}(x, y), \]

\[ \Delta \text{SALE}(x, y) \nabla T(x, y) \]  
from Differentiation

Rules b, c,

\[ \Delta \text{V} \equiv T(A, C) \equiv \text{SUPPLY}(z, A, C) \]  
indicating that the deletion has no effect on integrity and hence gives the same result as example a.

d) A constraint prohibits companies from supplying a type D item without supplying a type E item, and a transaction inserts (A, B, C) into SUPPLY:

\[ V \equiv \text{SUPPLY}(x, y, z) \text{CLASS}(z, D) \nabla S(x), \]

\[ S(x) \equiv \text{SUPPLY}(x, v, u) \text{CLASS}(u, E), \]

\[ \Delta \text{SUPPLY}(x, y, z) \equiv (x, y, z) = A, B, C, \]

\[ \Delta V \equiv \nabla \text{SUPPLY}(x, y, z) \nabla \text{SUPPLY}(x, y, z) \nabla \text{M}(x, z), \]

\[ \Delta \text{SUPPLY}(x, y, z) \nabla \text{M}(x, z), \]

\[ \Delta \text{SUPPLY}(x, y, z) \]  
from Rule c, where

\[ \text{M}(x, z) = \text{CLASS}(z, D) \nabla \text{S}(x), \]

\[ \Delta \text{M}(x, z) = \text{CLASS}(z, D) \nabla \text{CLASS}(z, D) \nabla \text{S}(x), \]

\[ \Delta \text{CLASS}(z, D) \nabla \text{S}(x) = \]

\[ \text{FALSE} \]  
from the Rules b, c, and the given transaction,

\[ \text{VM}(x, z) = \text{CLASS}(z, D) \nabla \text{S}(x), \]

\[ \text{S}(x) \nabla \text{CLASS}(z, D) \nabla \text{VM}(x, z), \]

\[ \text{CLASS}(u, E) \nabla \text{CLASS}(u, E) \nabla \text{SUPPLY}(x, v, u), \]

\[ \Delta \text{SUPPLY}(x, v, u) \nabla \text{CLASS}(u, E), \]

\[ \text{CLASS}(z, D) \nabla \text{CLASS}(C, E) \nabla A \]  
from Rule c and the given transaction,

\[ \Delta V \equiv \text{CLASS}(z, D) \nabla \text{S}(x), \]

\[ \text{CLASS}(z, D) \text{CLASS}(C, E) \nabla A = A, \]

\[ \text{CLASS}(C, D), \text{CLASS}(C, E), A = A \]  
by substitution and negation Rule d2.

\[ = \text{CLASS}(C, D) \nabla \text{CLASS}(C, E), \]

\[ = \text{CLASS}(C, D) \nabla \text{CLASS}(C, E), \]

\[ \text{from identify Rule a1,} \]

\[ \Delta V \equiv \text{CLASS}(C, D), S(A), \text{CLASS}(C, E) \]  
where

\[ S(A) \equiv \text{SUPPLY}(A, v, u) \text{CLASS}(u, E) \]  
indicating that integrity is maintained if class of C is not D, or class of C is E, or A supplies an item of class E.

e) A constraint prohibits a company from supplying two different departments with item D; and a transaction inserts (A, B, C) into SUPPLY:

\[ V \equiv \text{SUPPLY}(x, y, z) \text{SUPPLY}(x, z, D) \nabla y \neq z, \]

\[ \Delta \text{SUPPLY}(x, y, z) \equiv (x, y, z) = (A, B, C), \]

\[ \Delta V \equiv \nabla \text{SUPPLY}(x, y, z) \nabla \text{SUPPLY}(x, y, z) \nabla \text{M}(x, z), \]

\[ \Delta \text{SUPPLY}(x, y, z) \nabla \text{M}(x, z), \]

\[ \Delta \text{SUPPLY}(x, y, z) \]  
from Rule c, where

\[ \text{M}(x, z) = \text{CLASS}(z, D) \nabla \text{S}(x), \]

\[ \Delta \text{M}(x, z) = \text{CLASS}(z, D) \nabla \text{CLASS}(z, D) \nabla \text{S}(x), \]

\[ \Delta \text{CLASS}(z, D) \nabla \text{S}(x) = \]

\[ \text{FALSE} \]  
from the Rules b, c, and the given transaction,

\[ \text{VM}(x, z) = \text{CLASS}(z, D) \nabla \text{S}(x), \]

\[ \text{S}(x) \nabla \text{CLASS}(z, D) \nabla \text{VM}(x, z), \]

\[ \text{CLASS}(u, E) \nabla \text{CLASS}(u, E) \nabla \text{SUPPLY}(x, v, u), \]

\[ \Delta \text{SUPPLY}(x, v, u) \nabla \text{CLASS}(u, E), \]

\[ \text{CLASS}(z, D) \nabla \text{CLASS}(C, E) \nabla A \]  
from Rule c and the given transaction,

\[ \Delta V \equiv \text{CLASS}(z, D) \nabla \text{S}(x), \]

\[ \text{CLASS}(z, D) \text{CLASS}(C, E) \nabla A = A, \]

\[ \text{CLASS}(C, D), \text{CLASS}(C, E), A = A \]  
by substitution and negation Rule d2.

\[ = \text{CLASS}(C, D) \nabla \text{CLASS}(C, E), \]

\[ = \text{CLASS}(C, D) \nabla \text{CLASS}(C, E), \]

\[ \text{from identify Rule a1,} \]

\[ \Delta V \equiv \text{CLASS}(C, D), S(A), \text{CLASS}(C, E) \]  
where

\[ S(A) \equiv \text{SUPPLY}(A, v, u) \text{CLASS}(u, E) \]  
indicating that integrity is maintained if class of C is not D, or class of C is E, or A supplies an item of class E.

f) A constraint requires the existence of a type D item supplied by all companies, and a transaction deletes (A, B) from CLASS:

\[ V \equiv \text{R}, \]

\[ \text{R} = \text{CLASS}(u, D) \nabla \text{S}(u), \]

\[ \text{S}(u) \equiv \text{SUPPLY}(x, y, z) \nabla T(x, u), \]

\[ T(x, u) \equiv \text{SUPPLY}(x, u), \]

\[ \text{VM}(x, u) = \text{CLASS}(u, E) \nabla \text{S}(u) \nabla \text{CLASS}(u, D), \]

\[ \Delta \text{SUPPLY}(x, u) \nabla \text{CLASS}(u, E), \]

\[ \Delta \text{SUPPLY}(x, u) \nabla \text{CLASS}(u, E), \]

\[ \text{CLASS}(z, D) \nabla \text{CLASS}(C, E) \nabla A = A \]  
from Rule c and the given transaction,

\[ \Delta V \equiv \text{R}, \text{S}(A), B \neq D \]  
by negation

\[ \text{R} = \text{CLASS}(u, D) \nabla \text{S}(u), \]

\[ \text{CLASS}(u, D) \nabla \text{CLASS}(u, D), \]

\[ \text{CLASS}(u, D) \nabla \text{S}(u) \]  
from Rule 2 and the given transaction. \( \Delta V \) indicates that integrity is maintained if B and D are different types, or a type D item is supplied by all companies \( \text{R}' \), or there is a company that does not supply A.

5 SUFFICIENT CONDITIONS
A number of sufficient conditions can be derived by applying the substitution and resolution rules to the necessary and sufficient conditions of Section 4 and the original constraint. These sufficient conditions are often easier to test than the necessary and sufficient conditions, and only one
of them needs to be tested for integrity maintenance. The selection of the condition to test depends on the database physical structure and will not be pursued here. Moreover, many other sufficient conditions and also necessary conditions can be derived similarly by considering the set of all constraints. The analysis here will be restricted to the sufficient conditions derived from each constraint individually. Given a constraint violation \( V \), let negation of its derivative be \( \Delta V = S \) providing the necessary and sufficient condition for integrity maintenance. \( \Delta V = S, V \) from identity rules, since \( V \) is FALSE from the assumption of integrity before the transaction. A number of very useful sufficient conditions are obtained by applying the substitution and resolution rules to \( S, V \). The following sufficient conditions are derived using the same examples as in Section 4.

\[
\begin{align*}
\text{a) } V & \equiv SALE(x, y)T(x, y) \\
T(x, y) & = SUPPLY(z, x, y) \\
\Delta V & = T(A, B) = SUPPLY(A, x, B) \\
\Delta V & = T(A, B), SALE(x, y)T(x, y),
\end{align*}
\]

assuming \( V \) is FALSE, using the substitution rule,

\[
\begin{align*}
\text{SALE}(A, B) & T(A, B), \\
\text{SALE}(A, B) & \text{using resolution and subsumption where} \\
\text{SALE}(A, B) & \text{is the new sufficient condition for integrity.}
\end{align*}
\]

In fact, every disjunct of \( \Delta V \) is a sufficient condition for integrity, since all variables are assumed to be existentially quantified and distributable over OR, the satisfaction of any disjunct alone is sufficient for integrity. \( T(A, B) \) is the necessary and sufficient condition, \( SALE(x, y) T(x, y) \) is the original constraint and assumed FALSE, \( SALE(A, B) T(A, B) \) is subsumed; hence, \( SALE(A, B) \) is the only new sufficient condition. We will place the new sufficient conditions on the last line of each example for quick recognition.

\[
\begin{align*}
\text{b) } V & \equiv SALE(x, y)T(x, y) \\
T(x, y) & = SUPPLY(z, x, y) \\
\Delta V & = SALE(B, C), \\
\text{SUPPLY}(z, B, C)z & \neq A, \\
\text{SALE}(x, y)T(x, y), \\
T(B, C), \text{SUPPLY}(A, B, C)
\end{align*}
\]

using substitution and resolution, and since \( \text{SUPPLY}(z, B, C) \equiv \text{SUPPLY}(z, B, C)z \neq A, \text{SUPPLY}(z, B, C)z = A \) from the resolution rule.

\[
\begin{align*}
\text{c) } V & \equiv SALE(x, y)T(x, y) \\
T(x, y) & = SUPPLY(z, x, y) \\
\Delta V & = T(A, C), \text{SALE}(A, C)
\end{align*}
\]

using the resolution and subsumption rules.

\[
\begin{align*}
\text{d) } V & \equiv SUPPLY(x, y, z)CLASS(z, D)S(x) \\
S(x) & = SUPPLY(x, v, u)CLASS(u, E)
\end{align*}
\]

\( \Delta V = \text{CLASS}(C, D), \text{CLASS}(C, E), S(A), \text{SUPPLY}(x, y, z)\text{CLASS}(z, D)S(x), \text{SUPPLY}(A, y, z)\text{CLASS}(z, D), \text{SUPPLY}(y, x, C)S(x), \text{SUPPLY}(A, y, C) \)

It is important to note that sufficient conditions are derived by utilizing the resolution and other simplification rules, and hence the process is not as efficient as the derivation of the necessary and sufficient condition that requires only substitution. On the positive side, a large number of sufficient conditions are obtained by considering each constraint individually and resolving it with the necessary and sufficient condition already derived. Most sufficient conditions are derived with only one application of the resolution rule. The selection of the most efficient sufficient condition for execution is dependent on the physical database structure. However, a useful heuristic is to pick the condition with minimum number of joins. Estimating and comparing the execution cost of alternative sufficient conditions is beyond the scope of this article.

6 Extensions

6.1 Domain Independence

The preceding sections assumed that the transactions would not affect the underlying domain; in other words, they would not insert into or delete from the universe of all database constants. They are called “domain invariant” transactions. Alternatively, this assumption can be interpreted to restrict the constraints to those that are not affected by the changes in the underlying domain. Most useful constraints fall into this category, and this “domain independence” assumption is present in most previous work in integrity maintenance in one form or another. In differential normal form, this assumption requires that each variable in the constraint to appear at least once in a positive database predicate in each disjunct of a derivation, and the resulting derived predicate is never negated if it involves the variable in question. Consequently,

\[
\begin{align*}
\Delta V & = \text{CLASS}(C, D), \text{CLASS}(C, E), S(A), \text{SUPPLY}(x, y, z)\text{CLASS}(z, D)S(x), \\
\text{SUPPLY}(A, y, z)\text{CLASS}(z, D), \text{SUPPLY}(y, x, C)S(x), \text{SUPPLY}(A, y, C) \\
\end{align*}
\]

\[
\begin{align*}
V & \equiv SUPPLY(x, y, D)\text{SUPPLY}(x, z, D) y \neq z \\
\Delta V & = D \neq C, T \\
T & = \text{SUPPLY}(A, z, D)z \neq B \\
\Delta V & = SUPPLY(x, y, D)\text{SUPPLY}(x, z, D) y \neq z, \\
\text{SUPPLY}(A, B, D), \text{SUPPLY}(A, B, C) \\
\end{align*}
\]

Domain independence is not a critical assumption in this article, since formal differentiation can easily be extended to include all first-order sentences including the domain dependent sentences by introducing a predicate \( U \) for the universe of all database constants. Given a database consisting
of relations \( R_1, \ldots, R_n; U(x) \equiv R_i(y_1), \ldots, R_n(y_n) \) where \( x \in y_i \) for some \( i \). The derivatives \( \Delta U \) and \( \nabla U \) can be derived using the differentiation rules. For example, \( \Delta U(x) = \Delta R_i(y_1), \ldots, \Delta R_n(y_n) \) for some \( i \). Once the \( U \) predicate and its derivatives are defined, the constraints are modified slightly to transform domain dependent constraints to equivalent domain independent constraints by restricting the domain to \( U \), and hence extending the differentiation rules to all first-order sentences.

**Axiom 6.1.** A constraint violation can only be caused by the data in \( U \), or the nonexistence of it in \( U \); i.e., the data outside the universe of database constants is irrelevant.

**Observation 6.1.** A domain dependent constraint violation \( V \equiv S \) can be reformulated as \( V \equiv SU(x_1) \ldots U(x_n) \) where \( x_1, \ldots, x_n \) are the variables in \( S \). This reformulation is sufficient to transform a domain dependent constraint to an equivalent domain independent constraint, so that the differentiation rules of Section 2 can be supplied. Proof of equivalence follows from Axiom 6.1 since the reformulation only restricts the variables to the relevant universe. Proof of sufficiency follows from the observation that the reformulation restricts all variables explicitly to the universe of database constants \( U \), and hence no variable fails to appear in a positive database predicate.

**Simplification 6.1.** \( P(y)U(x) \) is equivalent to \( P(y) \) where \( P \) is a database predicate and \( x \in y \).

Proof follows from the definition of \( U \) predicate, and the subsumption rule.

\[
U(x) = R_1(y_1), \ldots, P(y), \ldots, R_n(y_n) \\
P(y)U(x) = P(y)R_1(y_1), \ldots, P(y), \ldots, P(y)R_n(y_n)
\]

from the subsumption rule.

This equivalence leads to considerable simplification of derivatives since only the variables that do not appear in positive database predicates need to be considered in \( U \) predicates.

**Example 6.1.** Given the predicates \( SALE(x, y) \) \( SUPPLY(z, x, y) \) and \( CLASS(y, w) \); a constraint requires that if a department \( x \) sells an item \( y \), then everything must supply \( x \) to \( y \); and a transaction inserts \( (A, B) \) into \( CLASS \).

\[
V \equiv SALE(x, y) \quad SUPPLY(z, x, y) \\
\Delta CLASS(y, w) \equiv (y, w) = (A, B) \\
\Delta U(x) \equiv (x = A), (x = B) \\
V \equiv SALE(x, y) \quad SUPPLY(z, x, y)U(x) \equiv \quad \Delta U(y) \equiv \quad \Delta U(z)
\]

from simplification 6.1.

\[
\Delta V = SALE(x, y) \quad SUPPLY(z, x, y) \Delta U(z)
\]

for \( x = A, z = B \)

\[
\Delta V = SALE(x, y) \quad SUPPLY(A, x, y), \quad \Delta V = SALE(x, y) \quad SUPPLY(B, x, y)
\]

**6.2 Dynamic Constraints**

Dynamic constraints are constraints that involve multiple states of the database [2], [17]. We will restrict ourselves to those constraints involving only two states of the database, namely the state before and the state after a transaction. This is by far the largest and most useful class of dynamic database constraints, although it is possible to envision constraints involving more than two states and multiple transactions.

**Observation 6.2.** Dynamic constraints involving the database states before and after the transaction can be expressed by specifying the derivatives directly.

**Proof.** Given a database with relations \( R_1, \ldots, R_n \) let \( R_i', \ldots, R_n' \) be the new states of these relations, respectively. A constraint involving both states can be expressed in terms of \( R_i, \ldots, R_n, R_i', \ldots, R_n' \). \( R \) is defined as \( R \neg \neg R \). Consequently, the dynamic constraint can be expressed in terms of \( R_i, \ldots, R_n, \neg \neg R_i, \ldots, \neg \neg R_n \). A constraint violation expressed in terms of these differential relations, assuming that no violation existed before the transaction, was called a derivative. Dynamic constraint violations involve both states before and after the transaction, and hence cannot have violations before the transaction. Consequently, derivatives and database relations are sufficient to completely specify dynamic constraints.

**Example 6.2.**

a) Given the predicate \( CL \)

b) \( ASS(x, y) \) indicating that the class of item \( x \) is \( y \); a constraint prohibits modifying the class of item \( A \) from \( B \) to \( C \):

\[
\Delta V = CLASS(A, B) \neg CLASS(A, B) \Delta CLASS(A, C)
\]

b) Given the predicate \( SALE(x, y, z) \) indicating that the department \( x \) sells item \( y \) for the price \( z \); and a constraint prohibits departments from reducing prices:

\[
\Delta V = SALE(x, y, z) \neg SALE(x, y, z) \Delta SALE(x, y, w) w < z
\]

**7 Conclusions**

Formal differentiation of first-order sentences has been shown to be useful in maintaining database integrity. Once a database constraint violation is expressed as a first-order sentence, its derivative with respect to a transaction provides the necessary and sufficient condition for a new integrity violation to be caused by the transaction. The derivative is often much simpler to test than the original constraint since it maintains integrity differentially by assuming integrity before the transaction, and testing only for new violations. The formal differentiation approach has a number of distinct advantages over previous approaches to integrity maintenance:
1) Formal differentiation requires no resolution search, but only substitution. It is considerably more efficient than resolution-based approaches, and hence it is appropriate for real time integrity checking with ad-hoc transactions. Resolution-based approaches are computation intensive, and they are appropriate only for predefined transactions where the simplified tests can be derived ahead of time and compiled into update programs [8], [16], [19].

2) Formal differentiation is considerably more general than the previous substitution-based approaches. It is even more general than the resolution-based approaches since it does not require a domain independence assumption, and it is valid for all first-order sentences. Previous substitution-based approaches are considerably more restrictive. They analyze the quantifier structure of the constraint and perform substitutions depending on the quantifier structure. Kobayashi [6] and Nicolas [10] both fail for constraints where a universal quantifier is within the scope of an existential quantifier. Example f shows how formal differentiation can handle a constraint where a universal quantifier is within the scope of an existential quantifier. Example f demonstrates how formal differentiation works with a constraint with more than one occurrence of an updated relation. Kobayashi [6] can accommodate constraints with more than one occurrence of an updated relation. Both [6] and [10] generate some tests that need to be executed after the execution of the transaction. All of our tests are performed before the transaction is executed, avoiding an expensive rollback operation in case of failure.

3) Formal differentiation produces a large number of sufficient conditions in addition to the necessary and sufficient condition for integrity maintenance. Substitution based methods often find only one sufficient condition for each constraint by substituting some constants of the transaction for some variables of the constraint. Resolution based approaches find many more sufficient conditions. Formal differentiation method employs a limited use of resolution to obtain sufficient conditions, often restricted to a single application of the resolution rule to the already derived necessary and sufficient condition and the original constraint. It often finds more sufficient conditions than the full resolution-based approach with a single application of the resolution rule, possibly because of the simplicity of the transaction definitions. Examples b, and e demonstrate the derivation of additional sufficient conditions than those of [8], [10]. Differential relational algebra [15] finds no sufficient conditions.

4) The sufficient conditions derived by formal differentiation are often less strict than the previous approaches. Example d demonstrates the derivation of sufficient conditions that are less strict than previous approaches [8], [10].

5) Formal differentiation can be extended quite naturally to express a large class of dynamic constraints. Dynamic constraints are difficult to represent and maintain, and often higher order logics are used to study their semantics [2]. Formal differentiation method extends to accommodate dynamic constraints by merely expressing them as derivatives in terms of differential relations. Once expressed as derivatives, they can be enforced directly as queries against differential relations, without further simplification.

Future work is suggested in evaluating and comparing the execution cost of alternative sufficient conditions to aid in the selection of one as the optimum. Also the differential calculus is likely to have applications in transaction repair and view maintenance that are left for future work. Finally, the extension to recursive constraints, and constraints with statistical aggregate functions pose significant theoretical problems.
APPENDIX

Using standard first-order logic, a more detailed proof of Theorem 3.1 for some difficult cases is shown below, since the shorthand notation of differential calculus is not as explicit as the standard notation.

The following first-order tautologies are used throughout the proof:

A1: \((\forall y S(y)) \rightarrow (\forall z S(z))\) by renaming the variable.

A2: \(\forall z(\forall y S(y)) \rightarrow S(z)\) from A1, by collecting quantifier \(\forall z\).

A3: \(\forall z((\exists y S(y)) \leftrightarrow ((\forall y S(y)) \land S(z)))\) from A2, since \(A \rightarrow B\) if \(A \leftrightarrow AB\) for all \(A, B\).

a) Given \(V(x) \equiv S(x, y)\) with \(y\) nonempty,

\[\Delta V(x) \iff \nabla (V(x)) \iff (\exists y S(x, y)) \land (\exists z S(x, z))\]

from Definition 2.1 and the premise, by collecting the quantifier \(\exists y\),

\[i\f(\forall y S(x, y)) \land S(x, z)\]

from A3 by substituting \(S(x, y)\) for \(S(y)\), by distributing \(\exists y\),

\[\nabla V(x) \Delta x\]

from Definition 2.1 and the premise.

Similarly:

\[\nabla V(x) \iff V(x) \land V'(x) \iff (\forall y S(x, y)) \land (\exists z S(x, z))\]

from Definition 2.1 and the premise, by collecting the quantifier \(\exists z\),

\[i\f(\forall y S(x, y)) \land S(x, z)\]

from A3 by substituting \(S(x, y)\) for \(S(y)\), by distributing \(\exists z\),

\[\nabla V(x) \forall S\]

from Definition 2.1 and the premise.

c) Given \(V(x) \equiv S(x, y)T(x, z)\) with \(y, z\) nonempty,

\[\Delta V(x) \iff \nabla (V(x)) \iff (\exists y S(x, y)) \land (\exists z S(x, z))\]

from Definition 2.1 and the premise, with \(y, z\) nonempty,

\[i\f(\exists y S(x, y) \lor T(x, z)) \land (\exists z S(x, z))\]

by collecting the quantifier \(\exists z\),

\[\nabla V(x) \land \exists w ((\forall y S(x, y) \lor T(x, z)) \land (\exists S(x, z) \land S(x, v)) \land (S(x, v) \land T(x, w)) \land \Delta (S(x, v) \land T(x, w)))\]

from A3 by substituting \((S(x, y) \lor T(x, z))\) for \(S(y)\),

\[\nabla V(x) \land \exists w ((\Delta S(x, v) \land T(x, w)) \lor (\Delta T(x, w) \land S(x, v)))\]

from the premise and by distributing \(\lor\),

\[i\f(\forall (\Delta S(x, v) \land T(x, w)) \lor (\Delta T(x, w) \land S(x, v)))\] by Definition 2.1,

\[\nabla V(x) \land \exists w (\Delta S(x, v) \land \Delta T(x, w)) \lor (\Delta S(x, v) \land \Delta T(x, w)) \lor (\Delta S(x, v) \land \Delta T(x, w))\]

by distributing \(\exists w\) and using differential calculus notation.

Similarly:

\[\nabla V(x) \equiv \nabla (T, S, S \nabla T)\]

by following the same steps.
REFERENCES


Levent V. Orman received the MM and PhD degrees from Northwestern University, Evanston, Illinois. He currently is an associate professor of information systems at Cornell University, Ithaca, New York. His professional interests include design, specification, and evaluation of information systems, with particular emphasis on database and knowledge-base architectures and languages. The most recent articles he published appeared in Information Systems, Intelligent Information Systems, Applied Artificial Intelligence, Acta Informatica, and IEEE Transactions on Software Engineering.