Local Verification of Global Integrity Constraints in Distributed Databases

Ashish Gupta*
Department of Computer Science
Stanford University
Stanford, CA 94305-2140
agupta@cs.stanford.edu

Jennifer Widom
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099
widom@almaden.ibm.com

Abstract

We present an optimization for integrity constraint verification in distributed databases. The optimization allows a global constraint, i.e. a constraint spanning multiple databases, to be verified by accessing data at a single database, eliminating the cost of accessing remote data. The optimization is based on an algorithm that takes as input a global constraint and data to be inserted into a local database. The algorithm produces a local condition such that if the local data satisfies this condition then, based on the previous satisfaction of the global constraint, the global constraint is still satisfied. If the local data does not satisfy the condition, then a conventional global verification procedure is required.

1 Introduction

A clear trend in information systems technology is the distribution of related data across multiple sites. Such systems may vary from tightly-coupled parallel databases to federated information systems. In all cases, one benefit of data distribution is that each site processes its own data with some degree of autonomy. At the same time, however, since data across sites may be related, there may be certain global consistency requirements, or *integrity constraints*, on the distributed data. Integrity constraints specify those configurations of the data that are considered semantically correct. Integrity constraints have been discussed in many forms and are well accepted as a useful mechanism for monitoring and enforcing data semantics.

*Work was supported by NSF grants IRI-91-16646 and IRI-90-16358, and ARO DAAL-03-G-0177.

A database that is initially consistent with respect to a set of integrity constraints can become inconsistent if data is modified. When an integrity constraint involves data from multiple sites, verifying consistency could involve a distributed transaction and the expenses associated with it: two phase commit protocols, distributed concurrency control, network communication costs, and multiple interface layers if the databases are heterogeneous [CP84, OV91]. A useful optimization in these environments is to be able to verify global constraints by only accessing local data. We present an algorithm that takes a global constraint and data to be inserted into a local database and produces a local test condition. If the local data satisfies the test condition then, based on the previous satisfaction of the global constraint, the global constraint is still satisfied. We call this approach *Local Verification of Global Integrity Constraints.* The following two examples illustrate our general approach (we re-use the examples throughout this paper).

EXAMPLE 1.1 Consider an employee-department relational database with two relations:

`emp`$(E,\ D,\ S)$
% employee number E in department D has salary S
`dept`$(D,\ MS)$
% some manager in department D has salary MS

Say integrity constraint $I1$ requires that every department number referenced by a tuple in the `emp` relation exists in the `dept` relation. This is often referred to as a referential integrity constraint. If a tuple `emp`$(e,\ d1,\ 50)$ is inserted, we need to verify the existence of tuple `dept`$(d1, MS)$, where MS could be any constant. If the two relations `emp` and `dept` reside on different sites, this check can be done by reading the `dept` relation remotely. Suppose, however, that the local relation `emp` contains a tuple `emp`$(e1,\ d1,\ s1)$, and the database satisfies integrity constraint $I1$ before the tuple is inserted. The `dept` relation must therefore contain a tuple `dept`$(d1, MS)$. Consequently when the tuple `emp`$(e, d1, 50)$ is inserted, a remote read is not required to confirm the existence of department $d1$. □

EXAMPLE 1.2 Consider another integrity constraint $I2$ on the employee-department database. $I2$ requires that every employee earn less than every manager in the same department. When the tuple $\mathtt{emp}(e, d1, 50)$ is inserted in the `emp` relation, constraint $I2$ needs to be checked. Suppose, however, that department $d1$ already has an employee $e1$ whose salary is 100. Given that constraint $I2$ is not violated by the existing tuple, we know that the salary of every manager of department $d1$ is more than 100. Therefore, the salary of every manager of department $d1$ is more than 50 and inserting $\mathtt{emp}(e, d1, 50)$ does not violate constraint $I2$. □

In this paper we consider the relational model [Ull88], although our results can also be applied to other models. We give an algorithm that takes:

1. A tuple to be inserted into a local relation L,
2. An integrity constraint C over local and remote relations, and
3. An assumption that the constraint is not violated before the insertion is made to L.

The algorithm produces a test condition over local relation L such that if L satisfies the test, then the insertion into L does not violate the integrity constraint C. If on the other hand, L does not satisfy the test, a conventional integrity constraint checking method needs to be used. In Example 1.1 the local test condition requires finding an existing employee tuple with the same department number as the new employee. Similarly, in Example 1.2, the local test condition requires finding an existing employee whose salary is at least as much as the salary of the new employee, and who works in the same department. In both cases if the local test conditions are not satisfied then a conventional integrity checking mechanism has to be used to check if tuple $\mathtt{emp}(e, d1, 50)$ violates integrity constraints $I1$ and $I2$ respectively.

Although the test condition depends on the inserted tuple, for efficiency it can be generated once (at compile time) and then instantiated with the inserted tuple at run time. Even though we discuss only insertions in this paper, we have extended our local verification techniques to consider deletions and updates. We consider distributed databases as the motivation for our work. However, the idea of local integrity constraint checking is not restricted to distributed databases. The cost of accessing different relations on the same site may be different because, for example, some critical data may be expensive to lock, or some data may periodically be unavailable. Checking integrity constraints using only the cheaper data then becomes attractive. On the other hand, in some distributed databases it may be the case that remote data is not any more expensive to access than local data. In such cases our optimization is inappropriate.

For a given integrity constraint, our optimization can be used at every site involved in the constraint for which the optimization is applicable. There is no problem with concurrency control since the optimization generates conditions over local data only.

1.1 Related Work

Most previous work on integrity constraint checking is not tailored for the distributed database environment and addresses one of the following two problems:

1. How to use the integrity constraint assertion and the modification to avoid consulting the database at all, i.e. to infer that the modification is irrelevant with respect to the constraint [BCL89, Elk90]. This approach also produces a local test because the inference process does not use remote data; in fact the test is independent of the underlying database.
2. How to use the modification to reduce the amount of data read from the database during the checking phase. This can be done by deriving cheaper sufficient conditions than the naive exhaustive constraint check [Nic82, KSS87, LST87, BMM92], or by maintaining redundant derived information to avoid run time inference [BBC80, BGM92].

This paper presents a way to use a specific part of the original database in order to check integrity constraints; namely the data that is local to the site where the modification is made.

[BGM92] also considers the distributed database scenario, with a similar goal of checking global integrity constraints locally. Their solution, the *Demarcation Protocol*, relies on storing some extra information about data on remote sites and the solution applies only to a limited class of integrity constraints.

[Nic82, KSS87, LST87, BMM92] present ways to produce sufficient integrity constraint verification tests given a modification to the database and the entire original database. In this paper we present an algorithm to generate sufficient tests that refer to only a part of the database. We also consider the (nontrivial) issue of converting the test into a query on the local database. Our local approach gives a less general but more efficient test than the global checks of [Nic82, KSS87, LST87, BMM92].

[BBC80] uses extra aggregate information (maxima and minima) about the database to produce sufficient tests for constraints that are expressed by tuple calculus formulas involving two tuple variables. The techniques described in this paper avoid storing any extra information and use a more general constraint assertion language.

[Elk90] considers conjunctive queries with arithmetic inequalities, and structural constraints like functional dependencies. The queries he considers are similar to, but less expressive than, the integrity constraints in this paper. The results in [Elk90] (and also those in [BCL89]) determine when an integrity constraint is independent of a modification to the database, using only the constraint and the modification. Hence, the methods in [BCL89, Elk90] are even more efficient than ours (because they do not require accessing any data) but are less general in that our approach may succeed in verifying a constraint locally while the methods in [BCL89, Elk90] may not. Consequently, the methods in [BCL89, Elk90] might be applied first and, if they fail, our methods could be applied.

1.2 Outline of Paper

Section 2 provides intuition for our approach. Section 3 defines the integrity constraint assertion language used in the paper. Section 4 describes our method for local verification of global integrity constraints and proves its correctness. We also discuss evaluability issues involved in local verification. Section 5 discusses how our results would be used in practice, in particular how integrity constraints expressed in SQL and Datalog would be checked. Section 6 derives local tests for two extensions of the integrity constraint assertion language discussed in Section 3. Finally, further research problems and extensions of the work are mentioned in Section 7.

2 Intuition

The discussion in this section is kept informal in order to highlight the underlying concepts. The ideas are formalized later in the paper. We first define an assumption underlying all the results in this paper.

Definition 2.1 Initial Consistency Assumption: Given an integrity constraint C and a modification made to the database, the initial consistency assumption states that C is not violated by the database before the modification is made. □

Our goal is to use the initial consistency assumption to infer information about remote data using local data. This information allows us to verify global integrity constraints when data is inserted into the local database, without accessing remote data.

Consider an integrity constraint C that references a local relation which we call L (new data is inserted into L). For simplicity, we treat the remaining

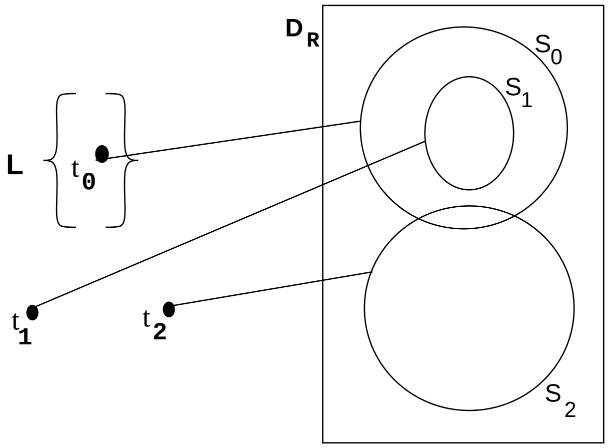

Figure 1: Intuition

relations involved in C as remote, although one or more of them could be at the same site as L. We refer to the remote relations as $\bar{R}$. Figure 1 shows the relations relevant to C. Relation L is on the left hand side of the figure. The box D_R on the right represents all the possible states of remote relations $\bar{R}$. A particular remote database state corresponds to a point in the box D_R. Points t_1 and t_2 represent tuples to be inserted into relation L.

Consider an existing tuple t_0 in the local relation L, represented by the point t_0 in Figure 1. Circle S_0 represents all possible states of $\bar{R}$ that violate constraint C with t_0. The initial consistency assumption says that the current database state does not violate constraint C. Therefore the current state of $\bar{R}$ lies outside circle S_0.

Now suppose tuple t_1 is to be inserted into local relation L, which contains tuple t_0. Circle S_1 represents all those states of $\bar{R}$ that violate constraint C with tuple t_1. Suppose that S_1 is contained in S_0. Then, all states of $\bar{R}$ that violate constraint C with tuple t_1 also violate C with tuple t_0. Given that tuple t_0 is in L, and that the current state of $\bar{R}$ does not violate C with t_0 (initial consistency assumption), we infer that the current state of $\bar{R}$ lies outside circle S_0 and therefore outside circle S_1. Hence, the current state of $\bar{R}$ does not violate constraint C with tuple t_1. In this case, we say that the tuple t_0 "covers" the inserted tuple t_1.

Now suppose tuple t_2 is to be inserted into local relation L, which contains tuple t_0. Circle S_2 represents all those states of $\bar{R}$ that violate constraint C with tuple t_2. Suppose that circle S_2 is not contained in the circle S_0. Then, there could be some state of $\bar{R}$ that violates constraint C with tuple t_2 but does not violate C with tuple t_0. Such a state would be in S_2 but not in S_0. The current state of $\bar{R}$ could be one such state. Therefore, tuple t_0 cannot be used to conclude that the current state of $\bar{R}$ does not violate C with tuple t_2.

The local verification technique presented in this paper effectively checks this "circle containment" property using a logical expression involving relation L. The remote relations do not appear in the expression, allowing it to be evaluated at the site where the tuple is inserted.

3 Constraint Language

The integrity constraint assertion language we consider is based on first order logic and is similar to that used in [CG92].[1] A logical form for integrity constraint assertions facilitates developing the local test condition and proving its correctness. In Section 5 we adapt our methods to constraints expressed in SQL and Datalog.

3.1 Syntax

An integrity constraint assertion is a first order logic sentence of the following form:

$$(C):\quad \forall \bar{X}\, \forall \bar{Y}\, \exists \bar{Z} : [(L(\bar{X}) \wedge R_1(\bar{Y}_1) \wedge \ldots \wedge R_k(\bar{Y}_k) \wedge g(\bar{X}, \bar{Y}, \bar{Z}, \bar{c})) \Rightarrow (S_1(\bar{Y}'_1, \bar{Z}'_1) \vee \ldots \vee S_n(\bar{Y}'_n, \bar{Z}'_n))]$$

where:

L represents the local relation.

$R_1, \ldots, R_k, S_1, \ldots, S_n$ represent remote relations.[2]

$\bar{X} = \{X_1, \ldots, X_t\}$ is a set of universally quantified ($\forall$) variables occurring only in L and g.

$\bar{Y} = \{Y_1, \ldots, Y_u\}$ is a set of $\forall$ variables occurring only in $R_1, \ldots, R_k, S_1, \ldots, S_n$, and g.

$\bar{Z} = \{Z_1, \ldots, Z_v\}$ is a set of $\exists$ variables occurring only in $S_1, \ldots, S_n$, and g.

$\bar{c} = \{c_1, \ldots, c_w\}$ is a set of constants occurring only in g.

$g(\bar{X}, \bar{Y}, \bar{Z}, \bar{c})$ is a conjunction of equalities ($=$) and inequalities ($<, >, \leq, \geq$).

$\bar{Y}_i \subseteq \bar{Y}$ is the set of variables that occur in relation $R_i, 1 \leq i \leq k$.

$\bar{Y}'_i \subseteq \bar{Y}$ is the set of $\forall$ variables that occur in relation $S_i, 1 \leq i \leq n$.

$\bar{Z}'_i \subseteq \bar{Z}$ is the set of $\exists$ variables that occur in relation $S_i, 1 \leq i \leq n$.

We often refer to the conjunction $g(\bar{X}, \bar{Y}, \bar{Z}, \bar{c})$ as just g. The integrity constraint assertion must also satisfy some additional restrictions in order to be *evaluable* [F82, VT91]. To state the restrictions we use the usual definition of equality.

Definition 3.1 Equality: A variable is equal to itself. In addition, variable P is equal to variable Q for a given constraint of the form C if g contains the term $P = S$ and variable S is equal to Q. Equality between a variable and constant is defined similarly. □

The restrictions on integrity constraint assertions are:

1. Existentially quantified variables can occur only in relations on the RHS of the implication ($\Rightarrow$) in the integrity constraint assertion.
2. Universally quantified variables occurring in a relation on the RHS of the assertion must also either occur in some relation on the LHS, be equal to a variable that occurs in a relation on the LHS, or be equal to a constant.
3. Existentially quantified variables occurring in g must be equal to a constant or a universally quantified variable.

The restrictions on integrity constraints specified in Items 1, 2, and 3 are needed in order to ensure the practical evaluability of the constraint (*domain independence* [Ull88]).

3.2 Semantics

Consider an integrity constraint assertion C. For each variable in $\bar{X}$, $\bar{Y}$, and $\bar{Z}$, let the domain of that variable be the domain of the relation column in which that variable appears. The integrity constraint is *satisfied* if for all assignments of values to the variables in $\bar{X}$ and $\bar{Y}$ there exists an assignment of values to variables in $\bar{Z}$ such that if:

1. There is a tuple in relation L with the values assigned to $\bar{X}$.
2. For each $R_i, 1 \leq i \leq k$ in C, there is a tuple in relation R_i with the values assigned to $\bar{Y}_i$.
3. Predicate g is satisfied using constants $\bar{c}$ and the values assigned to $\bar{X}$, $\bar{Y}$, and $\bar{Z}$.

Then:

For some $S_i, 1 \leq i \leq n$ in C, there is a tuple in relation S_i with the values assigned to $\bar{Y}'_i$ and $\bar{Z}'_i$.

We express the constraints of Examples 1.1 and 1.2 as sentences of logic in the form described above.

EXAMPLE 3.1 Constraint $I1$ requires that the department number of every tuple in the `emp` relation exists in the `dept` relation.

$$\forall E, D, S, \exists D', MS : [(\texttt{emp}(E, D, S) \wedge D' = D) \Rightarrow \texttt{dept}(D', MS)]$$

The above constraint is violated if there is an employee whose department number does not occur in any tuple in the `dept` relation. □

[1] This paper considers only *static* constraints [BMM92].

[2] Recall from Section 2 that even though we treat a relation as remote, it can actually be on the same site as local relation L.

EXAMPLE 3.2 Constraint $I2$ requires that every employee earn less than every manager in the same department.

$$\forall E, D, S, D', MS : [(\texttt{emp}(E, D, S) \wedge \texttt{dept}(D', MS) \wedge D' = D) \Rightarrow MS > S]$$

In the prescribed form, this is represented as:

$$\forall E, D, S, D', MS : [(\texttt{emp}(E, D, S) \wedge \texttt{dept}(D', MS) \wedge D' = D \wedge S \geq MS) \Rightarrow \texttt{false}]$$

Constraint $I2$ is violated if there is an employee who earns at least as much as some manager in the same department. □

4 Method for Local Verification of Integrity Constraints

This section formalizes local verification and is structured as follows. Section 4.1 presents the local test condition derived from a global constraint and a tuple to be inserted into the local relation. Section 4.2 illustrates the local test condition using our examples. Section 4.3 discusses evaluability and generality of local testing.

4.1 Local Test Condition

Definition 4.1 $\mathsf{LTC}(C, \bar{v})$**:** Consider an integrity constraint C:

$$(C): \quad \forall \bar{X}\, \forall \bar{Y}\, \exists \bar{Z} : [(L(\bar{X}) \wedge R_1(\bar{Y}_1) \wedge \ldots \wedge R_k(\bar{Y}_k) \wedge g(\bar{X}, \bar{Y}, \bar{Z}, \bar{c})) \Rightarrow (S_1(\bar{Y}'_1, \bar{Z}'_1) \vee \ldots \vee S_n(\bar{Y}'_n, \bar{Z}'_n))]$$

Let $\bar{v}$ be a tuple inserted into the local relation L. The local test condition is as follows:

$$(\mathsf{LTC}(C, \bar{v})): \quad \exists \bar{X}\, \forall \bar{Y}\, \forall \bar{Z} : [L(\bar{X}) \wedge (g(\bar{v}, \bar{Y}, \bar{Z}, \bar{c}) \Rightarrow g(\bar{X}, \bar{Y}, \bar{Z}, \bar{c}))]$$

□

If relation L satisfies $\mathsf{LTC}(C, \bar{v})$ then inserting tuple $\bar{v}$ into L does not violate constraint C. Notice that only relation L is involved in $\mathsf{LTC}(C, \bar{v})$. The test condition does not refer to any of the remote relations $R_1, \ldots, R_k, S_1, \ldots, S_n$.

The above test is derived at compile time by treating $\bar{v}$ as a parameter instead of using the actual value for the inserted tuple. When a tuple is actually inserted into local relation L at run time, $\mathsf{LTC}(C, \bar{v})$ is instantiated by the inserted tuple and evaluated using the local relation. In order to locally evaluate $\mathsf{LTC}(C, \bar{v})$ the universally quantified variables in $\bar{Y}$ and $\bar{Z}$ need to be eliminated from the implication $(g(\bar{v}, \bar{Y}, \bar{Z}, \bar{c}) \Rightarrow g(\bar{X}, \bar{Y}, \bar{Z}, \bar{c}))$. Eliminating the variables in $\bar{Y}$ and $\bar{Z}$ results in a set of restrictions on the variables in $\bar{X}$ that serve as the selection condition for the cover tuple in the local relation. In [GW92] we give an algorithm for eliminating universally quantified variables. The following theorem proves the correctness of Definition 4.1.

Theorem 4.1 *Consider an integrity constraint C and a tuple $\bar{v}$ inserted into relation L. If the database satisfies integrity constraint assertion C before adding tuple $\bar{v}$, and if the local test condition $\mathsf{LTC}(C, \bar{v})$ is satisfied by the local relation L, then the database satisfies integrity constraint assertion C after inserting tuple $\bar{v}$.* □

Theorem 4.1 is a special case of Theorem 6.1; a proof of Theorem 6.1 appears in [GW92].

4.2 Examples

EXAMPLE 4.1 Consider the referential integrity constraint $I1$ as stated in Example 3.1:

$$\forall E, D, S\, \exists D', MS : [(\texttt{emp}(E, D, S) \wedge D' = D) \Rightarrow \texttt{dept}(D', MS)]$$

The local relation L is the `emp` relation, g is $D = D'$, and the inserted tuple parameter is $<e^v, d^v, s^v>$. The local test according to Definition 4.1 is:

$$\exists E, D, S\, \forall D', MS : [\texttt{emp}(E, D, S) \wedge ((D' = d^v) \Rightarrow (D' = D))]$$

Suppose tuple $\texttt{emp}(e, d1, 50)$ is inserted. The parameterized variable d^v is replaced by the actual department number from the inserted tuple, namely $d1$, resulting in the test:

$$(\mathcal{T}): \quad \exists E, D, S\ \forall D', MS : [\texttt{emp}(E, D, S) \wedge ((D' = d1) \Rightarrow (D' = D))]$$

The universally quantified variables are eliminated from test condition $\mathcal{T}$ to give a condition involving only the variables of the local relation. Variable MS does not appear in $\mathcal{T}$ and can therefore be eliminated. Variable D' is eliminated by propagating the equality $D' = d1$. Hence we have:

$$\exists E, D, S : [\texttt{emp}(E, D, S) \wedge D = d1]$$

This condition is evaluated by performing a query on the `emp` relation for a tuple that has $d1$ as its department number. If the query has a non-empty answer then we conclude that inserting tuple $\texttt{emp}(e, d1, 50)$ does not violate integrity constraint $I1$. □

EXAMPLE 4.2 Consider the integrity constraint $I2$ as stated in Example 3.2.

$$\forall E, D, S, D', MS : [(\texttt{emp}(E, D, S) \wedge \texttt{dept}(D', MS) \wedge D' = D \wedge S \geq MS) \Rightarrow \texttt{false}]$$

L is again `emp`, g is $D' = D \wedge S \geq MS$, and the inserted tuple parameter is $<e^v, d^v, s^v>$. The local test according to Definition 4.1 is:

$$\exists E, D, S\ \forall D', MS : [emp(E, D, S) \wedge ((D' = d^v \wedge s^v \geq MS) \Rightarrow (D' = D \wedge S \geq MS))]$$

When tuple $\mathtt{emp}(e, d1, 50)$ is inserted, the parameterized variable d^v is replaced by the department number from the inserted tuple: $d1$, and variable s^v is replaced by the salary from the inserted tuple: 50. The resulting test is:

$$\exists E, D, S\ \forall D', MS : [emp(E, D, S) \wedge ((D' = d1 \wedge 50 \geq MS) \Rightarrow (D' = D \wedge S \geq MS))]$$

The universally quantified variables D' and MS are eliminated from the test condition to give a condition involving only the variables of the local relation:

$$\exists E, D, S : [emp(E, D, S) \wedge D = d1 \wedge S \geq 50]$$

This condition is evaluated by performing a query on the emp relation for a tuple that has $d1$ as its department number and an integer $S \geq 50$ in the salary field. If the query has a non-empty answer then we conclude that inserting tuple $\mathtt{emp}(e, d1, 50)$ does not violate integrity constraint $I2$. □

Note that even though the remote relations $R_1, \ldots, R_k$, $S_1, \ldots, S_n$ were restricted to be base relations in integrity constraint assertion C, local test condition $\mathsf{LTC}(C, \bar{v})$ is correct even if remote relations are view/derived relations. However, L is restricted to be a base relation.

4.3 Evaluability of the Local Test Condition

In this section we briefly discuss the evaluability and generality of the local test condition described in the previous section.

Notice that the local test defined in the previous section produces an implication involving g, where g is a conjunction of arithmetic predicates. If the local relation L contains a tuple that satisfies this implication then we say that the local relation satisfies the test. In order to evaluate the test condition, the satisfiability of the implication needs to be reduced to a query on relation L. We therefore need to eliminate the variables in $\bar{Y}$ and $\bar{Z}$ from the implication. The variables in $\bar{Y}$ and $\bar{Z}$ correspond to the remote relations. Examples 4.1 and 4.2 illustrate the need for this elimination. If g contains just equality predicates, then eliminating the universally quantified variables is simply a matter of propagating equalities. The cost of doing the propagation is linear in the number of equality expressions in g. With arithmetic comparisons, as in Example 1.2, the complexity of simplifying the implication is $O(n^3)$ where n is the number of inequalities in g [Dav87].

If we allowed fully general predicates, including unrestricted use of arithmetic operators like $+$ or $-$, then the implication may be unsolvable. However, we might extend the predicates in g with restricted use of $+$ and $-$. Note that even though applicability of test $\mathsf{LTC}(C, \bar{v})$ is restricted by the evaluability of the implication based on g, the correctness of our method does not depend on the structure of g.

Consider the intuition for local verification discussed in Section 2. We concluded that an inserted tuple t does not violate an integrity constraint C based on one existing tuple in the local relation (for instance, tuple t_1 was covered by tuple t_0 in Figure 1). The local verification condition defined in Section 4.1 checks for this condition: relation L satisfies the test of Definition 4.1 if one of the tuples in L is sufficient to cover the inserted tuple t with respect to the constraint C. However, in general an inserted tuple may require more than one existing tuples to cover it. In terms of Figure 1, this corresponds to a scenario in which it takes the union of the circles for several existing tuples to contain the circle for the inserted tuple. While we have a conjecture on how to adapt our local test condition for multiple covering tuples, it is likely to be more expensive to evaluate such a test. In general, the set of covering tuples could contain the entire local relation. As future work we plan to explore local testing with multiple covering tuples. For certain classes of constraints, one covering tuple can be proved to be the most general local test. One such class is those constraints that are expressible as simple conjunctive queries [GU92].

5 Getting Practical

We have developed our approach to local verification of global integrity constraints using logic as our basis. However, our results are applicable directly to environments based on SQL or Datalog. To express a constraint in such a query language, it is natural to express the constraint as a query, where the query produces a non-empty answer if and only if the current state of the database violates the constraint. The query corresponds to the negation of the integrity constraint assertion C. Recall that C, as given in Section 3, has the form:

$$(C):\quad \forall \bar{X}\, \forall \bar{Y}\, \exists \bar{Z} : [(L(\bar{X}) \wedge R_1(\bar{Y}_1) \wedge \ldots \wedge R_k(\bar{Y}_k) \wedge g(\bar{X}, \bar{Y}, \bar{Z}, \bar{c})) \Rightarrow (S_1(\bar{Y}'_1, \bar{Z}'_1) \vee \ldots \vee S_n(\bar{Y}'_n, \bar{Z}'_n))]$$

C can be equivalently written as:

$$(V):\quad \forall \bar{X}\, \forall \bar{Y}\, \exists \bar{Z} : \neg [L(\bar{X}) \wedge R_1(\bar{Y}_1) \wedge \ldots \wedge \neg S_1(\bar{Y}'_1, \bar{Z}'_1) \wedge \ldots \wedge g(\bar{X}, \bar{Y}, \bar{Z}, \bar{c})]$$

The expression within the outermost negation of V corresponds naturally to the database query.

In this section we describe the applicability of our approach in terms of mapping SQL/Datalog queries to integrity constraint assertions in our logical language and then applying local verification. However, in practice the methods are directly applicable to the SQL/Datalog queries. The local test condition is then a query in SQL/Datalog involving just the local relation.

1.	*Select_Exp*	::=	`SELECT * FROM` $T_1, T_2, \ldots, T_n$ [`WHERE` *Predicate*]
2.	*Predicate*	::=	*Item Compare Item*
		|	`NOT EXISTS` (*Simple_Select*)
		|	*Item* `NOT IN` (*Simple_Select*)
		|	*Predicate* `AND` *Predicate*
3.	*Item*	::=	*Col_Name* | constant
4.	*Col_Name*	::=	$[T.]\,C$
5.	*Compare*	::=	$= \mid \leq \mid \geq \mid > \mid <$
6.	*Simple_Select*	::=	`SELECT * FROM` T [`WHERE` *Simple_Predicate*]
7.	*Simple_Predicate*	::=	*Item* = *Item*
		|	*Simple_Predicate* `AND` *Simple_Predicate*

Figure 2: Grammar for Integrity Constraint Definitions in SQL

5.1 SQL

Figure 2 gives a grammar for SQL queries that can be mapped into sentences of the form C. Note that all nested subqueries are either `NOT EXISTS` or `NOT IN` queries; "positive" subqueries such as `EXISTS` and `IN` can always be translated to joins [CG85]. An SQL query Q is mapped to a logic sentence C as follows:

1. All attributes of the relations that occur in the `FROM` list of Q occur as universally quantified variables in C.
2. The relations that occur in the `FROM` list of Q occur on the LHS in C: $L, R_1, \ldots, R_k$.
3. All attributes of the relations that occur in the nested subqueries of Q occur as existentially quantified variables in C.
4. The relations that occur in the nested subqueries of Q occur on the RHS in C: $S_1, \ldots, S_n$.
5. The conjunction of the `WHERE` clauses of the nested subqueries of Q and the `WHERE` clause of query Q occur as g in C.

The integrity constraint assertion C obtained from query Q is guaranteed to satisfy all the restrictions discussed in Section 3. Local verification handles insertions made to one of the relations in the `FROM` list of the SQL query Q: that relation is mapped to L. The local test derived by Theorem 4.1 is expressed as the query:

```
SELECT *  FROM L
WHERE Test_Cond
```

where *Test_Cond* is the selection condition on the local relation obtained by using the inserted tuple and C, and simplifying the local test condition, as discussed in Section 4.1.

By *safe* we mean something that satisfies the restrictions of before. Unsafe is the opposite. Restrictions on the `NOT EXISTS` query were imposed as follows:

- We cannot permit any inequalities in the `WHERE` clause because inequalities inside render the logical sentence unsafe. Those inequalities that can be expressed safely, can be pulled out of the nested query.
- Only equalities are therefore permissible. Even here, any inequality that involves both variables from the relation *in* the nested query, then that inequality should be pushed into the schema of the relation.
- When the logic sentence for the entire query is made, all variables that belong to the relation *in* the nested query, are *existentially* quantified. Of these variables, those that take part in equalities can actually be either universally or existentially quantified, but those that occur only in the schema and not in g, need to be existentially quantified by the safety requirements.

Some properties of the variables in g that arise because of the safety requirements imposed on the queries.

- Equality expressions that involve only existentially quantified variables are useless. This is because an existentially quantified variable is forced to ALSO be equal to some constant or some $\forall$ variable. In which case, both the existentially quantified variables might as well be equated to the same constant/$\forall$ variable.

5.2 Examples

EXAMPLE 5.1 Consider the referential integrity constraint $I1$ involving relations `emp` and `dept` from Example 1.1. The violation condition for constraint $I1$ is expressed as the following SQL query:

```
SELECT *  FROM emp
WHERE  NOT EXISTS (SELECT *  FROM dept
                   WHERE  emp.D = dept.D)
```

If a tuple $\texttt{emp}(e, d1, 50)$ is inserted into relation `emp`, the local test condition is checked by the following SQL query:

```
SELECT *  FROM emp
WHERE  emp.D = d1
```

If the above query has a non-empty answer, then we conclude that the inserted tuple does not violate integrity constraint $I1$. □

EXAMPLE 5.2 Consider constraint $I2$ from Example 1.2. The violation condition for constraint $I2$ is expressed by the following SQL query:

```
SELECT *  FROM emp, dept
WHERE  emp.D = dept.D
       AND emp.S ≥ dept.MS
```

If a tuple $\texttt{emp}(e, d1, 50)$ is inserted into relation `emp`, the local test is evaluated by the following SQL query:

```
SELECT *  FROM emp
WHERE  emp.D = d1
       AND emp.S ≥ 50
```

□

5.3 Datalog

Conjunctive queries in Datalog [C88] extended with arithmetic inequalities and safe negation map to our language for integrity constraint assertions directly. An integrity constraint is expressed as a safe Datalog query that defines a 0-ary derived predicate `panic`. `Panic` becomes true when the violation condition for the constraint is satisfied. Similarly, the local test condition is expressed as a safe Datalog query that uses only the local relation in its body and defines a 0-ary predicate `no-panic`. `No-panic` becomes true if the local test condition is satisfied.

A Datalog query P that expresses an integrity constraint violation can be mapped to a sentence of the form C if P obeys the following restrictions (we omit giving a rigorous grammar):

- P is a conjunction of base, derived, and built in predicates. The only permitted built in predicates are $=, <, >, \leq, \geq$. Joins between predicates are expressed using $=$.
- Predicates that occur positively in P can only be base predicates.
- Derived predicates that occur negatively in P should be defined by a single rule. The defining rule can use the base predicates in P and at most one other new base predicate. The only built in predicate permitted in the rule is equality, i.e. joins.

The subset of Datalog specified above corresponds to the subset of SQL described by the grammar in Figure 2. The examples below illustrate the local test expressed as a Datalog query that defines predicate `no-panic`.

EXAMPLE 5.3 Consider the referential integrity constraint $I1$ from Example 1.1.

$$\texttt{panic} :\!-\ \texttt{emp}(E, D, S)\ \ \&\ \ \neg\texttt{all\_dept}(D).$$
$$\texttt{all\_dept}(D) :\!-\ \texttt{dept}(D, MS).$$

The local test resulting from inserted tuple $\texttt{emp}(e, d1, 50)$ is:

$$\texttt{no\_panic} :\!-\ \texttt{emp}(E, D, S)\ \ \&\ \ D = d1.$$

□

EXAMPLE 5.4 Consider constraint $I2$ from Example 1.2.

$$\texttt{panic} :\!-\ \texttt{emp}(E, D, S)\ \&\ \texttt{dept}(D, MS)\ \&\ S \geq MS$$

The local test resulting from inserted tuple $\texttt{emp}(e, d1, 50)$ is:

$$\texttt{no\_panic} :\!-\ \texttt{emp}(E, D, S)\ \&\ D = d1\ \&\ \ S \geq 50$$

□

6 More General Constraints

We now consider two extensions to the integrity constraint assertion language described in Section 3. The first extension removes the restriction on the quantification of $\bar{Y}$ and $\bar{Z}$ in sentence C of Section 3. The second extension permits arithmetic inequalities to occur on the RHS of the integrity constraint assertion. Local tests for both of these languages are presented.

6.1 Unrestricted Quantifiers

We consider a sentence C of the form described in Section 3 and remove the restrictions on the quantifiers of $\bar{Y}$ and $\bar{Z}$. We therefore consider a first order logic sentence B with the variables $\forall\bar{Y}\exists\bar{Z}$ replaced by a sequence of arbitrarily quantified variables denoted $\Theta\bar{Y}$.

$$(B):\quad \forall\bar{X}\,\Theta\bar{Y} : [(L(\bar{X})\ \wedge\ R_1(\bar{Y}_1)\ \wedge \ldots \wedge\ R_k(\bar{Y}_k)\ \wedge\ g(\bar{X}, \bar{Y}, \bar{c}))\ \Rightarrow\ (S_1(\bar{Y}'_1)\ \vee \ldots \vee\ S_n(\bar{Y}'_n))]$$

where:
$\Theta\bar{Y} = \alpha_1 Y_1 \alpha_2 Y_2 \ldots$, each α_i is a $\forall$ or $\exists$ quantifier, and $Y_1, Y_2, \ldots$ are variables in $\bar{Y}$.

The other terms occurring in the assertion are the same as described before. The restrictions for evaluability given in Section 3 also apply here.

Definition 6.1 $\mathsf{LTC}_g(B, \bar{v})$**:** Consider an integrity constraint B of the form specified above. Let $\bar{v}$ represent a tuple inserted into the local relation L. The local test condition is as follows:

$$(\mathsf{LTC}_g(B, \bar{v})):\quad \exists\bar{X}\,\forall\bar{Y} : [g(\bar{v}, \bar{Y}, \bar{c})\ \ \Rightarrow\ \ (L(\bar{X})\ \wedge\ g(\bar{X}, \bar{Y}, \bar{c}))]$$

□

Theorem 6.1 *Consider an integrity constraint B and a tuple $\bar{v}$ inserted into relation L. If the database satisfies integrity constraint assertion B before adding tuple $\bar{v}$, and if the local test condition $\mathsf{LTC}_g(B, \bar{v})$ is satisfied by the local relation L, then the database satisfies integrity constraint assertion B after inserting tuple $\bar{v}$.* □

Proof: Omitted due to space constraints. □

Recall Definition 4.1. If the test $\mathsf{LTC}_g(B, \bar{v})$ of Definition 6.1 is applied to the more restrictive integrity constraint assertion stated in Section 3, we do not get test $\mathsf{LTC}(C, \bar{v})$ of Definition 4.1. Instead we get:

$$(\mathsf{LTC}'(C, \bar{v})): \quad \exists \bar{X}\, \forall \bar{Y}\, \forall \bar{Z} : [g(\bar{v}, \bar{Y}, \bar{Z}, \bar{c}) \;\Rightarrow\; (L(\bar{X}) \wedge g(\bar{X}, \bar{Y}, \bar{Z}, \bar{c}))]$$

The local relation L in test condition $\mathsf{LTC}'(C, \bar{v})$ occurs in a different position than in the test condition $\mathsf{LTC}(C, \bar{v})$ stated in Definition 4.1. The difference between the two tests, $\mathsf{LTC}'(C, \bar{v})$ and $\mathsf{LTC}(C, \bar{v})$, is that if $g(\bar{v}, \bar{Y}, \bar{Z}, \bar{c})$ is false, then $\mathsf{LTC}'(C, \bar{v})$ does not look for a tuple in the local relation L whereas test $\mathsf{LTC}(C, \bar{v})$ does. We illustrate this with an example.

EXAMPLE 6.1 Consider an integrity constraint $I3$ that requires all employees with salaries greater than 1000 to be in the finance department. In this integrity constraint, g contains the predicate $S > 1000$. Say a tuple $\mathtt{emp}(e, d1, 50)$ is inserted into relation `emp`. Intuitively, we do not need to check for violation of $I3$ because $50 \not> 1000$ and therefore employee e need not be in the finance department. Test $\mathsf{LTC}'(C, \bar{v})$ will not require reading the local relation in this case. However, test $\mathsf{LTC}(C, \bar{v})$ does require reading relation `emp`. Furthermore, in the case where `emp` contains no tuples, the test will fail. □

We introduced test $\mathsf{LTC}(C, \bar{v})$ first because it captures the intuition of local verification. Henceforth "local test condition" refers to the more general test $\mathsf{LTC}'(C, \bar{v})$.[3]

6.2 Arithmetic Inequalities on the RHS

Consider a first order logic sentence of the following form:

$$(A): \quad \forall \bar{X}\, \forall \bar{Y}\, \exists \bar{Z} : [(L(\bar{X}) \wedge R_1(\bar{Y}_1) \wedge \ldots \wedge R_k(\bar{Y}_k) \wedge g_1(\bar{X}, \bar{Y}, \bar{Z}, \bar{c})) \Rightarrow (S_1(\bar{Y}'_1, \bar{Z}'_1) \wedge \ldots \wedge S_n(\bar{Y}'_n, \bar{Z}'_n) \wedge g_2(\bar{X}, \bar{Y}, \bar{Z}, \bar{c}))]$$

[3] Note that the difference between the two tests $\mathsf{LTC}(C, \bar{v})$ and $\mathsf{LTC}'(C, \bar{v})$ is an instance of the conditions for irrelevant updates derived in [BCL89, Elk90].

Restrictions for evaluability, similar to those given in Section 3, apply to this language. The above integrity constraint assertion corresponds to an SQL query Q that has one `NOT EXISTS` or `NOT IN` subquery S_Q. S_Q can have an arbitrary number of relations in its `FROM` list (that map to the relations $S_1, \ldots, S_n$ in A). The `WHERE` clause of S_Q can use arithmetic inequalities $\leq, <, \geq, >$ (that map to $g_2(\bar{X}, \bar{Y}, \bar{Z}, \bar{c})$ in A).

Definition 6.2 $\mathsf{LTC}_a(A, \bar{v})$**:** Consider an integrity constraint A of the form specified above. Let $\bar{v}$ represent a tuple inserted into the local relation L. The local test condition is as follows:

$$(\mathsf{LTC}_a(A, \bar{v})): \quad \exists \bar{X}\, \forall \bar{Y}\, \forall \bar{Z} : [L(\bar{X}) \wedge (g_1(\bar{v}, \bar{Y}, \bar{Z}, \bar{c}) \Rightarrow g_1(\bar{X}, \bar{Y}, \bar{Z}, \bar{c})) \wedge (g_2(\bar{X}, \bar{Y}, \bar{Z}, \bar{c}) \Rightarrow g_2(\bar{v}, \bar{Y}, \bar{Z}, \bar{c}))]$$

□

Theorem 6.2 *Consider an integrity constraint A and a tuple $\bar{v}$ inserted into relation L. If the database satisfies integrity constraint assertion A before adding tuple $\bar{v}$, and if the local test condition $\mathsf{LTC}_a(A, \bar{v})$ is satisfied by the local relation L, then the database satisfies integrity constraint assertion A after inserting tuple $\bar{v}$.* □

Proof: Omitted due to space constraints. □

7 Conclusions and Future Work

This paper introduces a technique for optimizing integrity constraint verification in distributed databases. Our verification technique checks integrity constraints that involve data from multiple sites using only the data local to the site where modifications are made. Our method uses the initial consistency assumption, an integrity constraint assertion, and the relation into which a tuple is inserted to produce a parameterized local test that checks whether the insertion violates the constraint. The test is evaluated at run time using the inserted tuple and the local database. If the local relation satisfies the test then the inserted tuple does not violate the constraint. If the local relation does not satisfy the test, then an alternative way of checking the integrity constraint is required. Our methods are developed for integrity constraint assertions that are expressed in a subset of first order logic. We also give subsets of SQL and Datalog that can be mapped to this logic. Local verification is directly usable on constraints expressed in these query languages.

We have also developed local tests for determining when deletions and updates do not violate constraints. We can adapt our local test condition to be based on multiple local relations. However, the test may become very expensive. As future work we plan to explore efficient multi-relation tests. Recall from Section 4.3 that we also plan to explore tests for multiple covering tuples.

We have also begun initial exploration of extensions and adaptations of our results for:

- Handling an even more general class of integrity constraints.
- Exploiting relationships between constraints to optimize integrity constraint verification.
- Using (partially) inconsistent databases.
- Maintaining materialized views.
- Efficient caching of remote data.

7.1 Acknowledgements

We are grateful to Jeff Ullman for helping bring most of the ideas in this paper to the fore. We would also like to thank Stefano Ceri, Hector Garcia-Molina, and Sanjai Tiwari for valuable comments and discussion.

References

[BCL89] J. A. Blakeley, N. Coburn, and P. Larson. Updating Derived Relations: Detecting Irrelevant and Autonomously Computable Updates. *ACM Transactions on Database Systems*, 14(3):369–400, 1989.

[BBC80] P. A. Bernstein, B. T. Blaustein, and E. M. Clarke. Fast Maintenance of Semantic Integrity Assertions Using Redundant Aggregate Data. In *Sixth VLDB*, pages 126–136, 1980.

[BB82] P. A. Bernstein and B. T. Blaustein. Fast Methods for Testing Quantified Relational Calculus Assertions. In *Proceedings of ACM SIGMOD 1982*, pages 39–50, 1982.

[BGM92] D. Barbara and H. Garcia-Molina. The Demarcation Protocol: A Technique for Maintaining Arithmetic Constraints in Distributed Database Systems. In *Extending Database Technology Conference, LNCS 580*, pages 373–397, Vienna, March, 1992.

[Bla81] B. T. Blaustein. *Enforcing Database Assertions: Techniques and Applications.* PhD thesis, Harvard University, Cambridge, Massachusetts, Division of Applied Sciences, 1981.

[BMM92] F. Bry, R. Manthey, and B. Martens. Integrity Verification in Knowledge Bases. In *Logic Programming, LNAI 592 (subseries of LNCS)*, pages 114–139, 1992.

[CG92] S. Ceri and F. Garzotto. Specification and Management of Database Integrity Constraints through Logic Programming. Technical Report 88-025, Dipartimento Di Elettronica - Politecnico Di Milano, 1992.

[CG85] S. Ceri and G. Gottlob. Translating SQL into Relational Algebra: Optimization, Semantics and Equivalence of SQL Queries. *IEEE Transaction of Software Engineering*, 11(4):324–345, April 1985.

[CP84] S. Ceri and G. Pelagatti. *Distributed Databases: Principles and Systems.* McGraw-Hill Book Company, New York, N.Y., 1984.

[C88] A. K. Chandra. Theory of Database Queries. In *Proceedings of ACM SIGMOD 1988*, pages 1–9. ACM, 1988.

[Dav87] E. Davis. Constraint Propagation with Interval Labels. *Artificial Intelligence*, (32):281–331, 1987.

[Elk90] C. Elkan. Independence of Logic Database Queries and Updates. In *Proceedings of the Ninth PODS*, pages 154–160, Nashville, TN, 1990. ACM SIGACT-SIGMOD-SIGART.

[F82] R. Fagin. Horn Clauses and Database Dependencies.

[GU92] A. Gupta and J. D. Ullman. Generalizing Conjunctive Query Containment for View Maintenance and Integrity Constraint Checking. In *Workshop on Deductive Databases, JICLSP*, 1992.

[GW92] A. Gupta and J. Widom. Local Verification of Global Integrity Constraints in Distributed Databases. Technical Report, Stanford University, 1993.

[KSS87] R. Kowalski, F. Sadri, and P. Soper. Integrity Checking in Deductive Databases. In *Proceedings of the Thirteenth International Conference on Very Large Databases (VLDB)*, pages 61–69, 1987.

[Kuc91] V. Kuchenhoff. On the Efficient Computation of the Difference Between Consecutive Database States. In *Second International Conference, DOOD, LNCS 566*, pages 478–502, 1991.

[LST87] J.W. Lloyd, E. A. Sonenberg, and R. W. Topor. Integrity Constraint Checking in Stratified Databases. *Journal of Logic Programming*, 4(4):331–343, 1987.

[Nic82] J. M. Nicolas. Logic for Improving Integrity Checking in Relational Data Bases. *Acta Informatica*, 18(3):227–253, 1982.

[OV91] T. M. Oszu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[Ull88] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, Volumes 1 and 2. Computer Science Press, New York, 1989.

[VT91] A. V. Gelder and R W. Topor. Safety and Translation of Relational Calculus Queries. *ACM Transactions on Database Systems*, 16(3):235–278, June 1991.