OPTIMIZING DISTRIBUTED INTEGRITY CONSTRAINTS

Subhasish Mazumdar

Mathematics, Computer Science, and Statistics,
University of Texas at San Antonio,
San Antonio, TX 78249, U.S.A.
EMail: mazumdar@ringer.cs.utsa.edu

ABSTRACT

Database integrity constraints can be extremely expensive to maintain unless attention is paid to the problem during the design of the database. In the case of distributed databases, and particularly in the case of heterogeneous databases, the problems associated with constraint maintenance are even more acute. We lay the basis for an design-time tool that derives sufficient conditions from the original constraints as well as those derived after transaction analysis with the aim of reduction in non-locality of the constraints.

INTRODUCTION

Database integrity* is a guarantee of protection provided by a Database Management System against misuse, more specifically, against nonmalicious errors 34).

Integrity is specified through a set of integrity constraints, which are assertions that each legal or consistent database state is supposed to respect, i.e., the assertions can be expressed as predicates on database states 6). In practice, however, the database designer finds the guarantee of integrity to be limited mainly because integrity checking generally consumes excessive computing resources. Indeed, commercial relational database systems of today support few integrity constraints — at best, one can hope for the (often indirect) support of key and referential integrity 13).

Turning to the research community, we find that in exploring the process of design of relational databases, the choice of good relation forms has been examined in depth, but the problem of choosing a good set of integrity constraints 19) has been largely neglected 21). The neglect is even more dramatic in the case of distributed databases.

The simplest way of assuring integrity is of course to include the constraint as a test at the end of every transaction. But the integrity constraint is usually a non-trivial boolean query whose evaluation may necessitate the scanning of whole relations, or worse, joins of two or more relations. Such queries are costly in terms of execution time, and repeating those queries after every transaction implies a gigantic overhead for transaction processing. It is easy to see that this method can be impractically expensive.

For centralized databases, researchers have suggested that this check be optimized by exploiting the fact (first noted by Stonebraker 32)) that the constraints need only to be preserved, i.e., the query representing the integrity constraints is repeated at every state and the result of the query was true in the state prior to the update. The advantage of using precondition checks instead of backing out a transaction after failure was also suggested by Buneman and Clemons 4).

Nicolas 19) examined the syntactic structure of the constraints written in prenex normal form and showed how checking could be improved for constraint preservation after a simple update.

Bernstein and Blauenstein 9) first showed how redundant data (that is kept up-to-date) could help in the evaluation of such repeated queries. The idea was well known: the computation of the average of a growing numeric sequence, for example, can make good use of the current sum and sequence length.

Henschen et al. first suggested the use of theorem proving at compile time in order to find run-time tests that would ensure the preservation of constraints and also show that tests that are only sufficient can be found.

Sufficient tests are especially useful in cases where the necessary and sufficient tests are all expensive; cheaper sufficient tests help conserve resources.

Distributed databases exacerbate the problem of constraint maintenance 21). Owing to horizontal and vertical fragmentation of relations with the fragments stored at different locations, the integrity constraints must be translated into constraints on the fragments. Thus there are a lot more constraints to maintain when databases get distributed. In addition, replication of data imposes a constraint that the replicas have equal values at all times.

The naive method of checking constraints after each transaction is worse in the distributed case since the checking would typically require data transfer as well as computation. Further, the penalty for allowing a transaction to execute with the intention of aborting it at commit time in the event of violation of constraints is more severe since rollback and recovery must occur at all the sites in which the transaction participated.

Simon and Valduriez 28) extends the above methods of centralized databases to the case of distributed databases.

The basic question to be addressed is not merely how distributed constraints can be simplified but how they can be reformulated into constraints that are more local, and at best are entirely local?

One approach to this question is taken by Qian 21) who translates a distributed constraint into a constraint over the
fragments and then detects a situation where more than one site contains replicas of a fragment involved in a constraint, and removes that redundancy by assigning the job of checking that part of the constraint to the site containing the maximal number of relevant fragments. A general framework is suggested for finding sufficient conditions.

An old common-sense idea is relevant to the question at hand: when a number of people co-operate to sell raffle tickets, they do not need to make a conference call every time any individual seller encounters a prospective buyer. The solution is to divide the tickets among the sellers so that they may proceed independently for the most part, that is, until some sellers have exhausted their supply at which time a redistribution of unsold tickets takes place. Garcia-Molina 10) has clearly been thinking along these lines. He has suggested that numeric constraints 1 on heterogeneous distributed databases be broken up so that they become entirely local constraints. He gives the example of an airline which wants to sell tickets for one of its flights, but also wants other airlines to sell for that flight. This implies a global constraint that the sum of the number of tickets sold by the two airlines should be less than some maximum number. In practice, airlines give each other blocks of seats to sell and thus convert a distributed constraint into a local one. He suggests the desirability of tools to break up distributed constraints into local ones.

This paper addresses the last suggestion. With the aim of reducing the non-locality of a constraint, i.e., the number of sites involved, this paper generalizes the above common sense idea beyond numeric constraints. It describes the basis for a tool that uses the standard method of backchaining for finding sufficient conditions, and thus can easily be integrated with a design-time theorem proving approach. Like Qian, this approach aims at the reformulation of constraints, but it uses a standard approach to finding sufficient conditions. Further, the optimizations do not require replicated data. Like Bernstein and Blaustein we suggest that in some cases redundant data be stored, but our use of that data and method of its computation are different. The approach is especially fruitful because it addresses the needs of multi-databases which offer the advantages of distributed database in spite of accepting full site autonomy.

Another advantage of our approach is that we can find sufficient conditions not only for the distributed integrity constraints given but also those arising as tests for particular transactions. Often these tests are simpler.

Finally, our approach conforms with the admonition of Nicolas to concentrate on the problem of finding good constraints.

This paper is structured as follows. The next section reviews related work. The subsequent section introduces a metric on constraints called scatter. Next we introduce our theorem proving approach. In the following section, we discuss the constraint reformulation approach and in the penultimate section deal with the application of that approach on binary relation constraints. We use the last section for making concluding remarks.

RELATED WORK

Stonebraker 32) first showed that a query modification technique exploiting the fact that the database is initially consistent (prior to the update) leads to simplification of constraint checking. For constraints expressed in the relational language QUEL, Stonebraker gives efficient and straightforward algorithms for obtaining tests that must be added as conjunctions to the precondition of the update. However, INGRES 33) offers the support of key and domain constraints only. Simon and Valduriez 27) have improved on the technique for enforcement of domain constraints.

Sarin 24) concentrated on transactions that allow at most one update on any relation. He did a perturbation analysis of the effect of the transaction on the constraint, converting perturbations that falsify the constraint into runtime tests. Although he offered compelling examples, he did not present an algorithm.

Nicolas 13) also studied the same class of transactions; his constraints could be range-restricted well-formed formulas of first-order predicate calculus. He showed how to instantiate variable expressions in constraints so as to obtain simplified constraints that could be evaluated on the database after the transaction (i.e., post-tests; the method could be used to obtain pre-tests as well). However, the substitutions would not work for certain patterns of quantifiers.

Bernstein and Blaustein 21) worked on transactions that could be single tuple insert and delete statements. Constraints could be relational calculus formulas expressed in prenex normal form with the restriction that the variables in the prefix must range over distinct relations. A well formed formula is in prenex normal form if it consists of a prefix of quantifiers (universal or existential or both) followed by a matrix of quantifier-free formula which is either in conjunctive normal form or is an implication with both antecedent and consequent in conjunctive normal form. Examining only the pattern of the quantifiers in the prefix (and thereby neglecting the semantics of the predicates in the matrix) and the updates, they determined how the constraint could be simplified into a simpler but equivalent one. Kobayashi 15) extended this work of Bernstein and Blaustein by including prefixes in which the same relation could appear more than once, i.e., more than one variable could be quantified over the same relation.

Hsu and Imielinski 14) further generalized (and corrected) the algorithms of Bernstein and Blaustein allowing multiple quantifiers and multiple tuple updates for the same relation. They too examine only the quantifier prefix during the analysis.

It is well known that redundant data improves the performance of certain queries. Integrity constraints are special cases of (boolean) queries. The fact that the database is consistent prior to the update indicates that the result of the last boolean integrity constraint query is known to be true.
Thus it is clear that the combination of this fact and the use of redundant data, provided it is kept up to date cheaply, can be exploited for better test generation. Bernstein et al. 3) introduced this scheme for a small class of integrity constraints. Hsu and Imielinski 14) also take advantage of this idea through their use of system-maintained sets called effective ranges. Koenig and Paige 19) generalized the idea with their technique of finite differencing. Their method can deal with aggregate constraints directly. Paige 30) also introduced rudimentary complexity measures for the predicates in the constraints and associated the goal of simplification with the reduction in the complexity of the evaluation of the constraints.

The above approaches can be thought of as syntactic ones because they exploit the pattern of the quantifiers ignoring the matrix.

Qian suggests 22) the Floyd-Hoare-Dijkatra approach 7, 12) for test generation. However, the weakest precondition so obtained may not be in simplified form.

All the above approaches are deficient in that they cannot generate tests that are sufficient (i.e., tests that would guarantee the preservation of the constraints, but are not always necessary for their preservation). As we have mentioned before, sufficient tests are often cheaper. This deficiency was rectified by Henschen, McCune, and Naqvi 11) who first suggested the use of a theorem prover for test generation. They prove a theorem at compile time stating that the integrity constraints are preserved by transactions that are single updates. Converting constraints into clausal form (i.e., prefix of universal quantifiers only and a matrix in conjunctive normal form), they used a resolution theorem prover and showed how sufficient tests could be generated from the failure to prove that the update maintained the constraint. Recently their work has been extended by McCune and Henschen 18) to produce necessary and sufficient tests as well.

Sheard and Stemple 26) also use a theorem-proving approach but they focus on transactions written in a rich high level language called ADABTPL. Transactions are programs that change the database state atomically 1, and so the integrity assertions need to be invariants of all transactions whereas updates within transactions may legally violate the constraints. They use an extension of a Boyer-Moore theorem-prover in order to show at the time of compilation that transactions will preserve constraints with the knowledge of the types of their inputs but not their actual values. The feasibility of this approach has been demonstrated 25).

The preservation of constraints may well depend on data values input to the transaction and thus the designer must code correct but inexpensive run-time tests within the transactions. Since it is not realistic to expect database designers to write correct and optimal database specifications all at once, automated help is necessary in identifying relatively inexpensive run-time tests. Mazumdar 17) extends the ADABTPL approach and shows how run-time tests (both necessary and sufficient and only sufficient) can be generated and other useful mechanical feedback offered to the designer 31).

The approach of Henschen et al. 11) is similar to the ADABTPL approach in that both make use of theorem proving, but the latter uses a very powerful theorem prover and a richer transaction language with sophisticated update functions. The syntactic approaches, on the other hand, use a very simple update language.

In the case of distributed databases, there are much fewer results.

Simon and Valuduriez 28) argue that the methods of test generation suggested for centralized databases outlined earlier can be extended to the fragments of distributed relations. They make use of finite differencing and the storage of redundant data in the manner of Bernstein and Blaustein.

Qian 21) shows that distributed constraints can be translated into constraints on the fragments given the definition of the fragmentation, and offers a framework for constraint reformulation 29). Initially a site $s$ is allowed to participate in the computation of a constraint $C$ if $s$ contains a fragment involved in $C$. The next improvement detects a redundancy in that assignment: for a constraint $C$, if sites $s_1$ and $s_2$ both contain replicas of some of the fragments involved in $C$, but all of the $s_2$ fragments of $C$ are also contained in $s_1$, then $s_2$ is relieved from the computation of $C$. Sufficient conditions are also recommended by an algorithm that performs an unbounded search over all possible constraints attempting to minimize a generalized cost function; the effectiveness of the suggested heuristics in helping that search is not reported.

SCATTERING METRIC

Our aim in this section is to define a metric on constraints that captures the amount of their non-locality.

To capture the amount of non-local access necessary to evaluate a constraint, we define a metric on constraints. We will call it the the scatter of a constraint and denote it by $\sigma$. Let $\mathcal{K}$ be the set of all relation fragments. We assume that each relation fragment is located in exactly one of a set of sites $\mathcal{S}$; hence we define a function $\text{site}:\mathcal{K} \rightarrow \mathcal{S}$, i.e., $\text{site}(r)$ denotes the location of a relation fragment $r$.

We assume that integrity constraints are conjunctions of simpler constraints, the latter being safe 34) first-order formulas. A simple constraint $C$ is given by:

$$(Q_1 x_1 \in r_1) (Q_2 x_2 \in r_2) \ldots (Q_n x_n \in r_n) \phi(x_1, x_2, \ldots, x_n),$$

where $Q_i \in \{\exists, \forall\}, 1 \leq i \leq n$, and $\phi$ is a computable predicate 4). We define the site-set of the constraint $C$ as the set of sites of the relations that its bound variables range over.

A transaction is a sequence of deferred updates in Date’s terminology 5).

More correctly, a relation fragment is stored in one primary site: we do not rule out replication of fragments.

By computable, we mean that taking a database snapshot as the model 8), the constraint can be evaluated to true or false. This must be so even if some of the relations $r_i$ are not stored in the database but are derived.
for this intuition is that we are primarily interested in data joint with site-set(C); these are intersected with each other for in the distributed case.

For a compound constraint \( C = \bigwedge_{i=1}^{m} C_i \), where \( C_i \) (1 ≤ i ≤ m) are simple constraints, we define its scatter as:

\[
\sigma(C) = \max(\sigma(C_1), \sigma(C_2), \ldots, \sigma(C_m)).
\]

The intuition is that the scatter of a compound constraint is determined by the worst scatter of the conjuncts; the basis for this intuition is that we are interested primarily in data transmission and the conjuncts can be evaluated in parallel. For a centralized database, all constraints have \( \sigma = 1 \). Hence a constraint with a value of 1 for \( \sigma \) is the best we can hope for in the distributed case.

By abuse of notation, we will refer to the site of a bound variable and mean the site of the relation over which that bound variable ranges. For the above formula, the site of variable \( z_2 \) would mean site(r2).

Constraints that assure the preservation of integrity after the execution of a transaction would typically involve the transaction inputs. For such constraints (and others we will encounter shortly), we need to extend the definition of \( \sigma \) to take care of free variables.

Given a formula with free variables, we must first associate with each free variable a set over which its values may range. For example, in the case of transaction inputs, this would be a singleton set. Next, we find the sites at which that set can be located, and call it possible-sites for that free variable. This set is obtained from the meaning of the free variable. For transaction inputs, for example, possible-sites would include the site of every relation that is updated by the transaction through this input.

Let \( C \) be a simple constraint:

\[
(Q_1 x_1 \in r_1)(Q_2 x_2 \in r_2) \ldots (Q_n x_n \in r_n) \ p(a, x_1, x_2, \ldots, x_n),
\]

\( C \) has a free variable \( a \) and bound variables \( x_1, \ldots, x_n \); its scatter \( \sigma \) can be found by the following rule:

\[
\text{if } (\text{possible-sites}(a) \cap \text{site-set}(C)) = \emptyset \\
\text{then } 1 + \text{cardinality(site-set}(C)) \\
\text{else } \text{cardinality(site-set}(C))
\]

The case of more than one free variable is an extension of the above. For each free variable, we intersect the possible-site set with site-set(C). If the intersection is nonempty, we ignore that variable. The other sets are all disjoint with site-set(C); these are intersected with each other and the minimum number of sites necessary to cover them is added to the cardinality of site-set(C). Although this sounds complex, it is rarely important because we do not often encounter more than two variables with overlapping possible-site sets.

### A THEOREM PROVING APPROACH

Mechanical theorem proving is a method of symbolic manipulation employing rules and techniques that respect logical inference; an element of heuristic-based search is usually involved in the selection of these rules and techniques, and thus the time required to come to a conclusion cannot be predicted in advance. A test generation approach based on theorem proving is feasible since the designer deals with transactions at the time of their compilation (as contrasted with their execution), when considerable computing resources can be invested at ease, with enormous potential savings during execution.

The ADABTPL environment offers a design-time tool for database designers. Although the tool uses a theorem prover, the database designer need not be concerned with its internal working. The prover has at its disposal a set of rewrite rules (a lemma base or rule base), each rule having been mechanically validated. The set of rewrite rules may also be viewed as an algebraic theory capturing the interaction among the data structures in the schema, the predicates in the integrity constraints, and the update functions that occur in the transactions. By describing the effect of a transaction on a database predicate through an algebraic expression, the rewrite rules of the theory can be used to transform the expression or parts thereof: thus one can reason about a transaction.

Given a transaction and a set of constraints, the prover attempts to establish that the transaction will preserve all the constraints. If it fails, it computes tests which need be added to the transaction as pre-conditions. These tests are necessary and sufficient for the preservation of the constraints.

We have also demonstrated that tests that are only sufficient can also be found by adapting the operational semantics of PROLOG (which uses backtracking with unification on Horn clauses). We find sufficient conditions by back-chaining from a necessary and sufficient test. Back-chaining is an automated procedure of rewriting an expression of the form \( p(x, y) \) that unifies with the consequent of a rule, which is an implication \( p(x, y) \leftarrow p_1(u) \land p_2(v) \) (written in PROLOG style; it reads \( p \) if \( p_1 \) and \( p_2 \)) by another expression of the form \( p_1(u) \land p_2(v) \). Note that variables in the left hand side of rules, i.e., the consequent of the implication, are universally quantified whereas variables occurring only in the right hand side are existentially quantified.

We are extending the above tool in order to deal with distributed constraints. The extension involves an additional module which, while backchaining, attempts to reduce scatter.

**Example:** Consider, for example, a referential integrity expressed by the following constraint:

\[ \text{we use the notation } t.fld \text{ to mean the fld component of tuple } t. \]
where site(r1) = s1, site(r2) = s2, and s1 \neq s2.

If a transaction inserts a tuple a into the relation ~1, the necessary and sufficient condition for the preservation of the referential integrity constraint is the simpler constraint†

\[ C_2: \text{a.frnkey} \in \Pi_{\text{frnkey} \in r_1} \text{a.frnkey} = x_2.\text{key}. \]

Since possible-site(a) = \{s1\}, we have \( \sigma(C_1) = \sigma(C_2) \), i.e., \( C_2 \) is no worse than \( C_1 \) in terms of scatter, but not any better either.

However, if we consider the sufficient condition \( C_3 \) for the preservation of \( C_1 \) for the above transaction:

\[ C_3: \text{a.frnkey} \in r_1, \]

i.e., \( (\exists x_2 \in r_2) \text{a.frnkey} = x_2.\text{key} \),

we see that \( \sigma(C_3) < \sigma(C_1) \), i.e., there is a definite reduction in scatter.

\( C_3 \) is obtained by the prover using backchaining.

CONSTRAINT REFORMULATION

In this section, we look at the basis for reformulation of constraints through sufficient conditions. Note that reformulation is not restricted to logically equivalent formulas.

In general, we look at the given integrity constraints and attempt to reformulate them without regard to the transactions. However, the transaction-specific tests (such as the one in the example at the end of the last section) are also amenable to this method; often such tests are simpler and lead to simple sufficient tests. This is an advantage of our method.

Basis for Reformulation

Here we examine the basis for our belief that the standard backchaining approach to finding sufficient conditions, can lead to new constraints that are less distributed.

Typically, the rule that computes a sufficient condition either introduces new variables or drops some. The following two lemmas explore these two cases.

The first lemma says that if a backchaining rule drops some variables, then it can be used without worsening scatter and in certain cases, scatter can improve as well.

**Lemma 1:** If there is a rule \( p(x_1, \ldots, x_n) \leftarrow p_1(x_1, \ldots, x_{n-1}, x_n) \) and a constraint \( C \) given by

\[ (Q_1 x_1 \in r_1) \ldots (Q_n x_n \in r_n) p(x_1, x_2, \ldots, x_{n-1}, x_n), \]

then \( C \) has a sufficient condition \( C_s \):

\[ (Q_1 x_1 \in r_1) \ldots (Q_{n-1} x_{n-1} \in r_{n-1}) p_1(x_1, x_2, \ldots, x_{n-1}), \]

**Corollary 1:** \( \sigma(C_s) \leq \sigma(C) \).

**Corollary 2:** If \( \text{site}(r_n) \notin \text{site-set}(C_s) \), then \( \sigma(C_s) < \sigma(C) \).

The first corollary says that the application of this lemma during backchaining does not worsen the scatter. Thus the process of can continue if \( p_1 \) unifies with any other rule.

The second corollary identifies a condition (in fact the condition) when the scatter will definitely reduce.

The second lemma says that if there is a backchaining rule that adds variables, then also, it can be used to reduce scatter in some conditions, and at worst, does not reduce scatter.

**Lemma 2:** If there is a rule \( p(x_1, \ldots, x_n) \leftarrow p_1(x_1, \ldots, x_j, z) \land p_2(z, x_{j+1}, \ldots, x_n) \), where \( n > 1 \) and \( j < n \), and a constraint \( C \) given by

\[ (Q_1 x_1 \in r_1) \ldots (Q_n x_n \in r_n) p(x_1, \ldots, x_n), \]

then \( C \) has a sufficient condition \( C_s = C_1 \land C_2 \), where \( C_1 \) is given by

\[ (Q_1 x_1 \in r_1) \ldots (Q_j x_j \in r_j) (\exists z \in r') p_1(x_1, \ldots, x_j, z), \]

and \( C_2 \) is given by:

\[ (\forall z \in r')(Q_{j+1} x_{j+1} \in r_{j+1}) \ldots (Q_n x_n \in r_n) p_2(x, x_{j+1}, \ldots, x_n) \]

**Proof:** Let \( M \) be a model for \( C_s \). We will show that \( M \models C_s \). Since \( M \models C_s \), it follows that \( M \models C_1 \) and \( M \models C_2 \).

Choose \( x_1 \) from \( r_1, \ldots, x_j \) from \( r_j \) as many times as dictated by the quantifiers \( Q_1, \ldots, Q_j \). For each such choice, since \( C_1 \) holds in \( M \), an assignment can be made for the variable \( z \) such that \( p_1 \) holds with \( x_1, \ldots, x_j, z \) as input.

Similarly choose \( x_{j+1} \) through \( x_n \) as per the quantifiers of \( C_2 \) and with any of the \( z \) values found from \( C_1, p_2 \) will hold.

In that case, a choice of \( x_1, \ldots, x_j, z \) and a choice of \( x_{j+1} \) through \( x_n \) satisfies the right hand side of our rule. Hence for the given choice of \( x_1, \ldots, x_n \), by the given rule, \( p \) holds. But the quantifiers of \( C \) are exactly those of \( C_1 \) and \( C_2 \). Hence \( M \models C \).

**Corollary 3:** \( \sigma(C_s) \leq \sigma(C) \).

**Corollary 4:** Let \( \text{site-set}(C_1) = S_1, \text{site-set}(C_2) = S_1, \) and \( \text{site-set}(C_3) = S_2 \). Then if \( S-S_1 \neq \emptyset \) and \( S-S_2 \neq \emptyset \), then \( \sigma(C_s) < \sigma(C) \).

The last corollary gives us a condition about the distribution of the relations that yields a reduction in scatter.

For example, consider a rule:

\[ p(x_1, x_2) \leftarrow p_1(x_1, z) \land p_2(z, x_2), \]

and a constraint:

\[ (\forall z \in r_1)(\exists x_2 \in r_2) p(x_1, x_2) \]

we get the sufficient condition:

\[ [(\forall z \in r_1)(\exists z \in r') p_1(x_1, z)) \land (\forall z \in r')(\exists x_2 \in r_2) p_2(z, y)] \]
The sufficient condition has introduced a new bound
variable \( z \), but the set \( r' \) that it ranges over must be com-
puted for the constraint to be useful. The above proofs
indicate that the set is to be computed by evaluating the
sufficient condition in a model, i.e., a database instance.
Furthermore, it can always be done since the rules always
involved computable predicates.

Next, we examine specific classes of constraints.

**BINARY RELATION CONSTRAINTS**

In this section, we focus on what we believe is the most
important form of integrity constraints — binary relations
and examine cases of total ordering, partial orderings, and
equivalence relations.

**Total Orderings**

The first lemma gives us the important result that it
is enough to store a singleton set as redundant data and get
a reduction in scatter for certain constraints.

**Lemma 3:** Let \( C \) be given by
\[
(\forall x_1 \in r_1)(\forall x_2 \in r_2) \ldots (\forall x_n \in r_n) f(x_1, \ldots, x_j) > g(x_{j+1}, \ldots, n),
\]
where \( > \) is a total ordering. Then a sufficient condition
\( C_s \) can be found for \( C \), where \( C_s = C_1 \land C_2 \), and \( C_1 \) is given by:
\[
[(\forall x_1 \in r_1) \ldots (\forall x_j \in r_j) (\exists z \in r') f(x_1, \ldots, x_j) > z],
\]
and \( C_2 \) is given by:
\[
[(\forall z \in r')(\forall z_{j+1} \in r_{j+1}) \ldots (\forall z_n \in r_n) z > g(x_{j+1}, \ldots, x_n)],
\]
and \( r' \) is a singleton set.

**Proof:** We use the transitivity rule for \( > \) to get \( C_s \)
through the lemma 2.

Now no matter what the choice of \( x_1, \ldots, x_j \), we
can take the minimum of the set \( r' \) (well defined since
since \( > \) is a total order and the set is finite) and let that
element constitute a singleton set \( r' \).

Each previous choice of \( x_1, \ldots, x_j \) remains a choice
that satisfies \( C_1 \) and each choice of \( x_{j+1}, \ldots, x_n \) still sat-
sify \( C_2 \).

**Corollary 5:** \( \sigma(C_s) \leq \sigma(C) \).

Now it is clear that the constraint with the \( > \) predi-
cate has a \( f \)-term and \( g \)-germ on its two sides. For certain
functions \( f, g \), it is possible to switch them around the \( > \)
predicates. For such functions, it will be possible to rewrite
the constraint so that the resulting rewritten constraint has
less scatter. Let us illustrate this observation with an ex-
ample.

**Example:** Consider the following constraint:

\[
(\forall x_1 \in r_1)(\exists x_2 \in r_2)(\forall x_3 \in r_3) x_2.fld_2 + x_3.fld_3 > x_1.fld_1
\]

Let \( site(r_1) = s_1, site(r_2) = s_2, \) and \( site(r_3) = s_3 \), with
\( s_1 \neq s_2 \neq s_3 \). The constraint has a scatter value of 3.

Here \( f \) is the + function and \( g \) is the identity function.

Applying the above lemma, we get the following conjunct
as the sufficient condition:
\[
(\exists x_2 \in r_2)(\forall x_3 \in r_3) x_2.fld_2 + x_3.fld_3 > k_1,
\]

where \( k_1 \) is a constant. Clearly the conjunct has a scatter
value of 2.

It should be possible to go further as the transitive
rule for \( > \) yields formulas with \( > \) predicates. Clearly it
is possible to move the additive terms on the other side
of the \( > \) predicate (the switching-around property we were
alluding to). Hence exploiting that characteristic of \( f \) and
\( g \), with respect to \( > \), this process can finally result in unit
scatter constraints.

The second conjunct already has unit scatter, so we
can focus on the first and using the fact that additive terms
can be switched around the \( > \) predicate, we get the following
condition equivalent to the first conjunct:
\[
(\exists x_2 \in r_2)(\forall x_3 \in r_3) x_2.fld_2 - k_1 > -x_3.fld_3
\]

Now we apply Lemma 2 to get its sufficient condition
as the conjunct:
\[
(\exists x_2 \in r_2) x_2.fld_2 - k_1 > k_2
\]

where \( k_2 \) is a constant. Clearly, now we have unit scatter
for the first conjunct as well as the second conjunct.

**Partial Orderings**

In the case of partial orders, we need to store at most \( m \)
elements, where \( m \) is the maximum number of incomparable
elements in the partial order. But note that the actual size
of the storage need be the number of incomparable elements
in any snapshot of the database.

**Lemma 4:** Let \( C \) be given by
\[
(\forall x_1 \in r_1)(\forall x_2 \in r_2) \ldots (\forall x_n \in r_n) f(x_1, \ldots, x_j) > g(x_{j+1}, \ldots, n),
\]
where \( > \) is a partial ordering with at most \( m \incompara-
tble elements. Then a sufficient condition \( C_s \) can be
found for \( C \), where \( C_s = C_1 \land C_2 \), and \( C_1 \) is given by:
\[
[(\forall x_1 \in r_1) \ldots (\forall x_j \in r_j) (\exists z \in r') f(x_1, \ldots, x_j) > z],
\]
and \( C_2 \) is given by:
\[
[(\forall z \in r')(\forall z_{j+1} \in r_{j+1}) \ldots (\forall z_n \in r_n) z > g(x_{j+1}, \ldots, x_n)],
\]
and \( r' \) has at most \( m \) elements.

**Proof:** As in the previous lemma, we choose \( x_1, \ldots, x_j \)
and find values of \( z \) that satisfy \( p \). We make all the
choices dictated by the quantifiers $Q_1, \ldots, Q_j$ and from the set of $z$ values so generated, we find the minimum of all comparable elements. The resulting set cannot contain more than $m$ elements.

**Equivalence Relations**

In the case of equivalence relation, it is enough to store the number of partitions in the equivalence relation.

**Lemma 5:** Let $C$ be given by

$$(Q_1 x_1 \in r_1) \cdots (Q_n x_n \in r_n) f(x_1, \ldots, x_j) \equiv g(x_{j+1}, \ldots, n),$$

where $\equiv$ is an equivalence relation with $m$ partitions. Then a sufficient condition $C_s$ can be found for $C$, where $C_s = C_1 \land C_2$, and $C_1$ is given by:

$$(Q_1 x_1 \in r_1) \cdots (Q_j x_j \in r_j)(3 z \in r') f(x_1, \ldots, x_j) \equiv z,$$

and $C_2$ is given by

$$(\forall z \in r')(Q_{j+1} x_{j+1} \in r_{j+1}) \cdots (Q_n x_n \in r_n) z \equiv g(x_{j+1}, \ldots, x_n),$$

and $r'$ contains at most $m$ elements.

For example, consider the constraint:

$$(\forall x)(\exists y) x = y \pmod{5}$$

It generates the following conjunct as sufficient condition:

$$(\forall x)(\exists e) e = e \pmod{5},$$

$$(\forall y)(\exists z) y = z \pmod{5}$$

**CONCLUSION**

In this paper, we have described the basis for a design-time tool that analyzes integrity constraints as well as transactions, and derives sufficient conditions with reduced scatter, a metric we have introduced for the amount of non-locality of a constraint. We plan to implement these ideas as an extension of our earlier work with centralized databases, and we have argued that they can be integrated.

This approach is especially relevant in the context of heterogeneous distributed databases where not only is the reduction of scatter useful, but reduction to unit scatter is extremely useful owing to site autonomy.

We are working on a precise description of a class of constraints for which it is possible to get unit scatter in reformulated constraints.

Future work involves the inclusion of aggregate constraints and an evaluation of the advantages (if any) of using a first-order formula representation of constraints as against a functional format. In the latter, referential integrity, for example, would be encapsulated as a single function taking both relations as arguments instead of a formula with two quantifiers. Also we are working on protocols for heterogeneous databases to follow when a violation of a sufficient condition occurs. Relaxation of the notion of constraint violation — making it permissible for a constraint to be violated for a period of time or number of updates — is another idea that we would like to explore.

**ACKNOWLEDGEMENTS**

The author is grateful to the anonymous referees for their helpful comments and suggestions.

**REFERENCES**


