Simplified approach for generating integrity tests in distributed database systems

Feras Hanandeh* and Emad E. Abdallah

Department of Computer Information Systems,
Faculty of Prince Al-Hussein Bin Abdallah II for
Information Technology,
Hashemite University,
P.O. Box 150459, Zarqa 13115, Jordan
E-mail: feras@hu.edu.jo
E-mail: Emad@hu.edu.jo
*Corresponding author

Alaa E. Abdallah

Department of Computer Science and Applications,
Faculty of Prince Al-Hussein Bin Abdallah II for
Information Technology,
Hashemite University,
P.O. Box 150459, Zarqa 13115, Jordan
E-mail: aabdallah@hu.edu.jo

Essam Al-Daoud

Department of Computer Information Systems,
Faculty of Science and Information Technology,
Zarqa University,
P.O. Box 132222, Zarqa 13132, Jordan
E-mail: essamdz@zpu.edu.jo

Abstract: Distributed database systems (DDBS) usually contain massive collections of data that rapidly evolve over time. It is extremely time consuming to make a perfect checking at each database update operation. This paper introduces a new algorithm to generate simplified integrity tests which could be used to automatically verify that database updates does not introduce any violation of integrity. The main attractive features of this approach are simplicity, flexibility in generating sufficient and complete integrity tests, and considering all conjunctive and disjunctive constraints. The experiment results show that the new algorithm provides more tests in comparison with existing techniques.

Keywords: constraint simplification; integrity maintenance; innovation; distributed database; integrity tests; learning.

Reference to this paper should be made as follows: Hanandeh, F., Abdallah, E.E., Abdallah, A.E. and Al-Daoud, E. (2013) 'Simplified approach for generating integrity tests in distributed database systems', *Int. J. Innovation and Learning*, Vol. 13, No. 4, pp.375–387.

Biographical notes: Feras Hanandeh is currently an Assistant Professor at the Prince Al Hussein Bin Abdullah II, Faculty of Information Technology, Al-Hashemite University, Jordan. He obtained his PhD in Computer Science from the University Putra Malaysia, Malaysia in 2006. His current research interests include distributed databases, parallel databases focusing on issues related to integrity maintenance, transaction processing, query processing and optimisation; grid computing, artificial intelligence and geographical information systems.

Emad E. Abdallah received his PhD in Computer Science from Concordia University in 2008, where he worked on multimedia security, pattern recognition and 3D object recognition. He received his BS in Computer Science from Yarmouk University, Jordan, and his MS in Computer Science from the University of Jordan in 2000 and 2004, respectively. He is currently an Assistant Professor in the Department of Computer Information Systems at the Hashemite University (HU), Jordan. Prior to joining HU, he was a Software Developer at SAP Labs Montreal. His current research interests include computer graphics, multimedia security, pattern recognition, and computer networks.

Alaa E. Abdallah is an Assistant Professor in the Department of Computer Science of Hashemite University, having joined in 2011. He obtained his BSc in Computer Science from Yarmouk University in 2000, MSc in Computer Science from University of Jordan in 2003, and PhD in Computer Science from Concordia University, Montreal-Canada in 2008. Prior to joining Hashemite University, he was a Network Researcher at consulting private company in Montreal (2008–2011). His research interest includes the routing protocols for ad hoc networks, parallel and distributed systems, and multimedia security.

Essam Al-Daoud received his BSc from Mu'tah University, MSc from Al Al-Bayt University and his PhD in Computer Science from University Putra Malaysia in 2002. He is currently an Associate Professor in the Department of Computer Science at Zarka University. His research interests include quantum computing, cryptography, singular value decomposition and artificial intelligent.

1 Introduction

The distributed nature of the current large and complex organisations involve with a very large number of transactions from their customers/clients (Wen et al., 2011). This data will be accessed and modified from different users at different physical places. Clearly, the integrity of data is essential for any business. To preserve the database integrity, several constraints (conditions) must hold on all valid database states. In this regards all transactions (insert, update and delete) have to be verified according to a predefined constraint during the run time. Verification process against large databases (Kim et al., 2011) is extremely difficult and takes long amount of time that negatively affects the efficiency of executing these valuable business transactions.

Integrity constraint maintenance usually goes through several phases starting from declaring constraints (known as constraint specification) by which it means stating declaratively what constitutes are semantically allowed for database state and also what

should be done if constraints are invalidated (Grefen, 1992; Martinenghi, 2005). The next phase is concentrated on investigating the generated set of constraints together to guarantee the consistency.

Efficient checking could be achieved by deriving a simplified form of the integrity constraints to be verified against the new database state before the update operation is performed. Several techniques have been developed for producing simplified incremental checks for each update operation but none has a sufficient quality and generality for providing a true practical impact. The constraint simplification produces integrity tests to take a place in the integrity constraint maintenance. It is vitally important to use the simplified tests in order to avoid some of the overheads imposed by the naive evaluation of each constraint after each transaction.

Sufficient and complete integrity tests are the main two types of integrity test. The integrity test for a constraint C and an update operation U has a sufficiency or a completeness property, when it proves that the C will not be violated by the executed update operation. Most previous approaches derive sufficient simplified form of the initial integrity constraint (Alwan et al., 2008). The main reason is that the sufficient tests have known to be cheaper than the complete tests where less data is transferred across the network. Several violations could be reported by an update operation including domain constraint, key constraint, referential integrity, or entity integrity constraints. Integrity enforcement is used to guarantee that the database state returns to a consistent condition when the new operation violates the predefined constraint. This could be done by rejecting the operation that causes the violation or triggering additional updates so the violation is corrected (Sandhu, 2011). The main motivation for our approach is to provide an effective lower cost simplified form of the integrity constraints that could be used to automatically verify that database updates does not introduce any violation of integrity.

The remainder of this paper is organised as follows. In Section 2, we briefly review some of the related work material. In Sections 3 and 4 we introduce the proposed approach and describe in details our algorithms. In Section 5, we present the parallel execution of distributed integrity tests. Finally, we present some experimental results and comparisons with existing techniques in Section 6.

2 Literature review

None of the early algorithms in the field of data integrity cover wide range of constraints particularly in distributed database systems. Most of the approaches only considered static constraints like key, domain, referential and general semantic constraints. Several techniques used the first order logic for integrity constraint simplification, which proved to have an important influence on subsequent research. They showed that a constraint could be simplified with respect to a single tuple update by instantiating certain variables in the constraint with constants from the update. The substitution is constructed by syntactic examination of the update operation and the constraint. In general, a single tuple update is considered (Hannola et al., 2010).

Alwan et al. (2008) proposed a framework for checking integrity constraints in a distributed database by utilising the local information stored at the target site. The framework consists of two main steps:

1 simplifies the integrity constraints to produce support tests and integrate them with complete and sufficient tests

2 select the most suitable test from several alternative tests when an update operation is submitted to the system.

The early approaches assume that an update operation will be executed at one site where the relation specified in the update is located. In general this is not always true (Alwan et al., 2010). Ramification problem that is refers to the production of indirect effects after the execution of an update in order to ensure the satisfaction of integrity constraints is proposed in Papadakis et al. (2011).

In Madiraju et al. (2006), a general algorithm was presented for checking global semantic integrity constraints in an XML setting. The presented approach does not require update statement to be executed before the constraint check is carried out and hence any rollback situations are avoided. It achieves speed as the sub constraint checks are executed in parallel. Simple classes of integrity constraints were examined in Grefen (1992), where the parallel constraint checking was considered in this scheme. A limited to static integrity constraints algorithm is presented in McCarroll (1995). Furthermore, there is a need to write complex transition axioms to describe the update operations, which consumes a lot of time and makes the technique less efficient.

Other approaches based on substitution presented in Ibrahim (2002a, 2002b). These approaches are limited to only integrity constraints formed from at most two relations. Furthermore, the generated tests are limited to the sufficient ones. The presented techniques in Ibrahim (2002a, 2002b) have efficiently reduced the complexity of maintaining the database integrity against transaction during run-time.

Attribute reduction by finding the minimal attribute from a large set of attributes is proposed in Salwani et al. (2011). A limited algorithm to the semantic integrity constraints is presented in Madiraju and Sunderraman (2004). Soumya et al. (2008) proposed a technique for relational databases to achieve optimisation of constraint checking process in distributed databases by taking advantage of the parallelism environment, compile time constraint checking, localised constraint checking, and the history of constraint violations. The architecture mainly consists of two modules: Constraint analyser and Constraint ranker for analysing the generated constraints and for ranking them.

In Alwan et al. (2009), a new methodology is proposed to rank and to select the suitable test to be evaluated from given several alternative tests. The technique uses the amount of data transferred across the network, the number of sites involved, and the amount of data accessed as the parameters in deciding the suitable test. Christiansen and Martinenghi (2006) and Christiansen and Rekouts (2007) presented a concrete techniques based on transformation operators that apply to integrity constraints written in a rich DATALOG-like language with negation. The resulting procedure produces at design-time simplified constraints for parametric transaction patterns, which can then be instantiated and checked for consistency at run-time. The problem with the above approaches is the high cost of generating the simplified tests. Motivated by the need for a new approach that perform the test in an effective and lower cost we propose a new methodology to generate a simplified form of integrity constraints in distributed database systems.

3 Preliminaries

Our approach has been developed in the context of the relational databases. Let V is an intentional predicate that denotes a violation of the database integrity constraint C in relation P, (i.e., V is the negation of C), where an integrity constraint is the condition that should be satisfied by the database. Efficient computation of V is critical in detecting semantic violations caused by erroneous database update operations. A database update operation is defined as a collection of INSERT a new tuple in a relation, DELETE an existing tuple from a relation, and MODIFY an attribute of an existing tuple. For every database relation P there are two different states of the same relation:

1 P is the state of the relation before an update operation is performed (old state)

2 P' is the state of P of the relation after the update operation is performed (new state). Let the following relations be defined for any database relation P:

$\neg insert\ P$ means that a tuple(s) were deleted from P, i.e., $delete\ P$ (R1)

$\neg delete\ P$ means that a tuple(s) were inserted into P, i.e., $insert\ P$ (R2)

$$\forall X\ (exist\ P(X) \leftarrow P(X) \wedge \neg delete\ P(X)) \tag{R3}$$

$$\forall X\ (not\ exist\ P(X) \leftarrow \neg P(X) \wedge \neg insert\ P(X)) \tag{R4}$$

$$\forall X\ (P'(X) \leftarrow exist\ P(X) \vee insert\ P(X)) \tag{R5}$$

In a database relation P, the existence of $P(X)$, means that the tuple(s) exists in P and are not being deleted from P, i.e., R1. Similarly not persistent of $P(X)$, means that the tuple(s) are does not exist in P and are not being inserted into P, i.e., R2. Given the fact that the pre-transaction state of the database is correct, a reduction of the data to be checked in constraint enforcement can be obtained by inspecting only those parts of relations that have been changed in a relevant way by the new update operation. This is usually accomplished by the use of the differential relations. See Grefen (1992), McCarroll (1995) and Ibrahim (2002a, 2002b). Motivated by the need for a new auxiliary relations for INSERT and DELETE transactions we propose two new differential relations and provide all the necessarily constraint.

- *Insert differential relation:* A new tuple(s) need to be inserted into P or a saved tuple(s) need to be updated. The new differential relation associated with P is denoted as *insert P* and can be defined as:

  $$\forall X(insert\ P(X) \leftarrow \neg P(X) \wedge P'(X)) \tag{R6}$$

  R6: The tuple(s) does not exist in the old state of relation P and exist in the new state P'.

- *Delete differential relation*: Saved tuple(s) need to be deleted from P. The old differential relation associated with P is denoted as *delete P* and can be defined as:

  $$\forall X(delete\ P(X) \leftarrow P(X) \wedge \neg P'(X)) \tag{R7}$$

  R7: The tuple(s) initially exist in the old state of relation P, but does not exist in the new state P'.

4 Sufficient and complete test generation

This section introduces our approach for constraint checking in distributed database environment. This could be done by deriving integrity tests (sufficient or complete) for a given constraint and an update operation(s). Integrity tests are used to detect violations as a replacement for of integrity constraints. Clearly it is more rewarding because integrity constraints is very time consuming as it often involve an execution for a complex queries against large databases.

Lemma 1: Given a predicate $P(X)$ in the form: $P(X) \leftarrow Q(Y) \wedge R(Z)$, delete $P(X)$ derived as: deleting a tuple(s) from the relations R or Q leads to delete these tuple(s) from P.

$$delete\ P(X) \leftarrow (delete\ Q(Y) \wedge R(Z)) \vee (delete\ R(Z) \wedge Q(Y)) \tag{R8}$$

Lemma 2: Given a predicate $P(X)$ in the form: $P(X) \leftarrow Q(Y) \vee R(Z)$, delete $P(X)$ derived as: deleting a tuple(s) from both relations R, Q and not insert into any of these two relations lead delete a tuple(s) from P.

$$delete\ P(X) \leftarrow (not\ exist\ Q(Y) \wedge delete\ R(Z)) \vee (not\ exist\ R(Z) \wedge delete\ Q(Y)) \tag{R9}$$

Lemma 3: Given a predicate $P(X)$ in the form: $P(X) \leftarrow Q(Y) \wedge R(Z)$, insert $P(X)$ derived as: existence (inserted or persistent) of a tuple(s) in both relations R, Q leads to insert these tuple(s) into P.

$$\begin{aligned} insert\ P(X) \leftarrow\ & (exist\ Q(Y) \wedge insert\ R(Z)) \vee (exist\ R(Z) \wedge insert\ Q(Y)) \\ & \vee (insert\ P(X) \wedge insert\ Q(Y)) \end{aligned} \tag{R10}$$

Lemma 4: Given a predicate $P(X)$ in the form: $P(X) \leftarrow Q(Y) \vee R(Z)$, insert $P(X)$ derived as: inserting a tuple(s) into any of the relations R or Q or both of them leads to insert these tuple(s) into P.

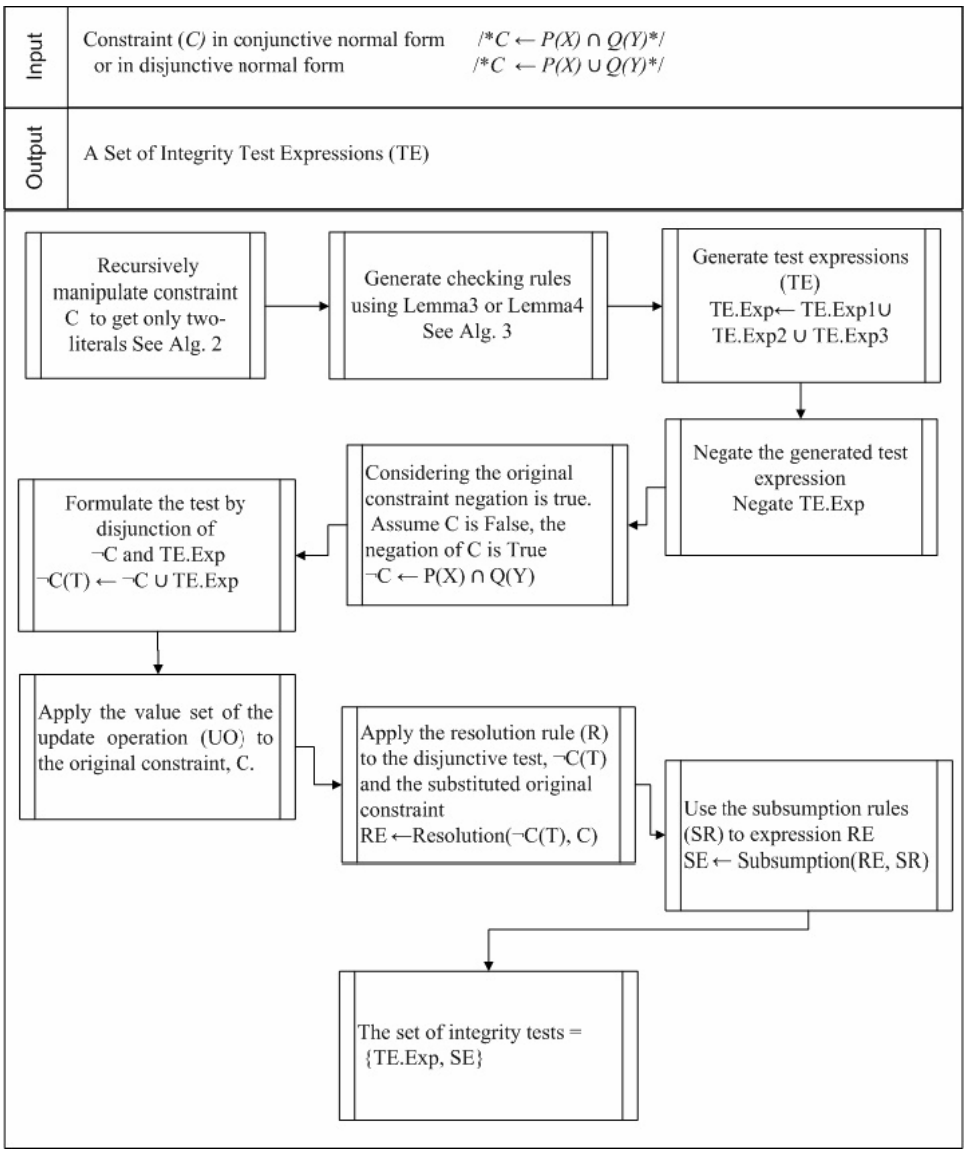
$$\begin{aligned} insert\ P(X) \leftarrow\ & (\neg Q(Y) \wedge insert\ R(Z)) \vee (\neg R(Z) \wedge insert\ Q(Y)) \\ & \vee (insert\ Q(Y) \wedge insert\ R(Z)) \end{aligned} \tag{R11}$$

The process of generating integrity tests starts once an actual update operation is submitted. Figure 1 presented in details the steps to derive integrity tests in conjunctive and disjunctive normal forms. The proposed novel algorithm has written in a general form, where it is independent of any specific application domain. Checking the validity (Belt et al., 2010) of the integrity tests against the database is performed at run time. The proposed algorithm accepts integrity constraints in conjunctive or disjunctive normal form. If the constraint consists of more than two-literal, the algorithm shown in Figure 2 need to be used to recursively manipulate any pairs of literals and assign them to one literal variable using Lemmas 1, 2, 3 and 4.

As an update operation is executed, a complete, sufficient and necessary integrity tests need to be checked. Let the test expressions (*TE*) are generated from the three checking rules (see Figure 3). A sufficient test(s) can be computed by applying the substitution and resolution rules to the results of $\neg TE$ and the original integrity constraint C. These sufficient tests are often easier to test than the complete

tests. Only one of the sufficient tests needs to be checked to prove that there are no violations in the database relation. For example, let an integrity constraint C and the negation of its violation TE is the generated sub-tests, then $TE \leftarrow TE \vee C$ from the identity rule, i.e., $A \vee F \leftrightarrow A$, where A is an arbitrary expression and F is False Boolean value. The integrity constraint C is not violated from the assumption of integrity before the transaction is performed. Our method can generate more useful integrity tests than the previous methods in McCarroll (1995) and Ibrahim (2002a, 2002b).

Figure 1 The main algorithm for generating integrity tests of initial constraint in conjunctive or disjunctive normal forms



5 Parallel execution of distributed integrity tests

At run-time, when an actual update is submitted by an end user, the requesting node selects the integrity rules that may violate the update operation. For example, a *Test-C* consists of a disjunction set *y* sub-tests, *SubT1* ∨ *SubT2* ∨ … ∨ *SubTy*. *Test-C* is True if and only if at least one of the *SubTi* is True, where $1 <= i <= y$.

In parallel execution of the distributed integrity sub-tests, there are two types of nodes, the main node (MN) and the other nodes (ONs). Let the database and the application is distributed among all nodes, as the integrity test is generated during the compile-time, the MN propagates the integrity sub-tests to the ONs. All generated integrity sub-tests are independent and separated by the *OR* conditional operation. Every node of the ONs does the check independently and in parallel for its fragment of tuple(s) to verify whether the integrity sub-test is violated or not. MN receives responses from all ONs carrying the results of the distributed sub-test. If one of the responses received by the MN is true, this would be a clear indication that no violation(s) to the integrity tests has occurred. As a result the MN will perform the update operation. If none of the received responses is true then another sub-test will be executed. If all complete-tests are carried out and without a success case then the test is False and the update operation will not be performed. If only sufficient-tests are used, then no conclusion could be drawn regarding the validity of the test. Therefore, another alternative test (if any) either sufficient or complete will be selected for evaluation.

Figure 2 The algorithm for generating a two-literal constraint by manipulating a given constraint recursively

Input	A constraint C consist of more than two-literals
Output	A constraint C' consist of only two-literals

```
While there is pair of literals assigned to literal value
Each parenthesized two literals is assigned to P(X) and computed as follows:
        If P(X) ← Q(Y) ∩ R(Z)
                If UO is Insert operation,
                        Compute P(X) using Lemma3
                Else
                        Compute P(X) using Lemma1
                End If

        Else If P(X) ← Q(Y) ∪ R(Z)
                If UO is Insert operation,
                        Compute P(X) using Lemma4
                Else
                        Compute P(X) using Lemma2
                End If
        End If
End Loop.
```

6 Experiment results

Our goals in this section are to evaluate the proposed method; we perform two case studies (company and banking) to prove the creditability of our algorithms and the effectiveness of our proposed technique. Comparisons between our proposed scheme and other techniques were also conducted. In general, banking information systems have:

1 large number of records physically distributed among different sites

2 massive number of transactions that need to be verified at run time according to a predefined constraint

3 database is distributed among different sites that could be accessed remotely.

6.1 Case study 1: banking systems

The distributed nature of banking systems involves a large amount of data. This data will be accessed and modified from different users and clients at different physical places.

6.1.1 Schema

Customer (customer_id, name, address, e-mail)

Account (account_id, customer_id, account_type, branch_name, balance)

Transaction (tx_id, account_id, amount, description).

6.1.2 Domain constraint

'The ATM transaction must be greater than 10 and less than 1,000'

$$(\forall w \forall x \forall y \forall z)(Transaction(w, x, y, z) \rightarrow (10 < y < 1{,}000))$$

update operation *insert Transaction* (a, b, c, d)

complete test $(10 < c < 1{,}000)$.

6.1.3 Key constraints

'customer_id is the primary key of any customer'

$$(\forall w \forall x1 \forall x2 \forall y1 \forall y2 \forall z1 \forall z2)(Customer(w, x1, y1, z1) \wedge customer(w, x2, y2, z2) \rightarrow (x1 = x2) \wedge (y1 = y2) \wedge (z1 = z2))$$

update operation insert customer (a, b, c, d)

complete test $(\forall x2\ \forall y2\ \forall z2)(\neg Ccustomer\ (a, x2, y2, z2) \vee [(b = x2) \wedge (c = y2) \wedge (d = z2)])$.

6.1.4 Referential integrity constraints

'The customer_id of every tuple in the account relation exists in the customer relation'

$$(\forall q \forall t \forall v \forall u \forall w \exists x \exists y \exists z)(Account(q, t, v, u, w) \rightarrow Customer(t, x, y, z))$$

update operation *insert Account*(a, b, c, d, e)

complete test $(\exists x \exists y \exists z)(Customer\ (b, x, y, z))$

sufficient test $(\exists q \exists v \exists u \exists w)\ Account(q, b, v, u, w)$.

6.1.5 Semantic integrity constraints

1 'Every saving account balance should be greater than 500 JD'

$$(\forall v \forall w \forall x \forall y \forall z)(Account(v, w, x, y, z) \wedge (x = \text{'}saving\text{'}) \rightarrow (z > 500))$$

update operation *insert Account* (a, b, c, d, e)

complete test $(c <> \text{'}saving\text{'}) \vee (e > 500)$.

2 'The customers are prevented from performing transactions on their saving accounts before performing transactions on their check accounts'

$$(\forall w \forall x \forall y \forall z \forall u \forall n \forall m \exists p \exists g \exists r \exists t \exists s \exists o)(Transaction(w, x, y, z)$$
$$\wedge Account(x, u, \text{'}saving\text{'}, n, m) \rightarrow Transaction(p, g, r, t)$$
$$\wedge Account(g, u, \text{'}check\text{'}, s, o)$$

update operation *insert*(*Transaction*(a, b, c, d)

complete test $(\forall u \forall n \forall m)(\neg Account(b, u, \text{'}saving\text{'}, n, m)$
$\vee (\exists w \exists y \exists z \exists u \exists n \exists m))Transaction(w, b, y, z)$
$\wedge Account(b, u, \text{'}check\text{'}, n, m)$.

sufficient test $(\exists w \exists y \exists z)\ (Transaction(w, b, y, z)$.

6.2 Case study 2: job agency

This example is used in most of the literature in the area of constraint checking (Alwan et al., 2008; Ibrahim, 2002a; Hanandeh, 2006), as given below:

6.2.1 Schema

Person (pid, pname, placed)

Company (cid, cname, totsal)

Job (jid, jdescr)

Placement (pid, cid, jid, sal)

Application (pid, jid)

Offering (cid, jid, no_of_places)

When a company offers a Vice Director job, it must have a Director or offer a Director Job first,

$$C1 \leftarrow (\forall x \forall y \forall z)(Offering(x, y, z) \wedge Job(y,\ \text{'}Vice\ Director\text{'}) \wedge S(x)).$$

Where from Figure 2

$$S(x) \leftarrow (\exists p \exists q)(Offering(x, p, q) \wedge Job(p, \text{'}Director\text{'}))$$

update *Insert* (*Offering*(*C*, *J*, *N*)), where the substitution set, $s = \{x/C, y/J, z/N\}$

We apply the algorithm shown in Figure 1 to generate integrity tests:

complete tests $TE \leftarrow \neg Job(J, \text{'}Vice\ Director\text{'}) \vee S(C) \vee Job(J, \text{'}Director\text{'})$, where

$$S(C) \leftarrow Offering(C, p, q) \wedge Job(p, \text{'}Director\text{'})$$

As a direct result one of the following tests needs to be checked to prove that there are no violations in the database relation:

1 $\neg Job(J, \text{'}Vice\ Director\text{'})$

2 $Offering(C, p, q) \wedge Job(p, \text{'}Director\text{'})$

3 $Job(J, \text{'}Director\text{'})$.

Sufficient test:

1 $(\exists z)\ (Offering(C, J, z))$.

Figure 3 The algorithm for generating a set of checking rules in conjunctive or disjunctive normal forms

Input	A Constraint *C* in conjunctive normal form
Output	A Set of test expressions (TE)

If constraint *C* in conjunctive normal form /**C* ← *P(X)* ∩ *Q(Y)**/
Generate checking rules using Lemma3.
/*Persistent of P(X) and insert a tuple(s) into relation *Q* */
TE.Exp1←*exist P(X)* ∩ *insert Q(Y)* where: *exist P(X)* ← *P(X)* ∩ ¬ *delete P(X)*
/*Persistent of Q(Y) and insert a tuple(s) into relation *P* */
TE.Exp2 ←*exist Q(Y)* ∩ *insert P(X)* where: *exist Q(Y)* ← *Q(Y)* ∩ ¬*delete Q(Y)*
/*Insert a tuple(s) into relation *P* and a tuple(s) into relation *Q**/
TE.Exp3 ← *insert P(X)* ∩ *insert Q(Y)*

Else If Constraint *C* in disjunctive normal form /**C* ← *P(X)* ∪ *Q(Y)**/
Generate checking rules using Lemma4.
/*Insert a tuple(s) into the relation *P* and Q(Y) is false*/
TE.Exp1 ← *insert P(X)* ∩ ¬*Q(Y)*
/* P(X) is false and insert a tuple(s) into the relation *Q* */
TE.Exp2 ← ¬*P(X)* ∩ *insert Q(Y)*
/*Insert a tuple(s) into relation *P* and a tuple(s) into the relation *Q* */
TE.Exp3 ← *insert P(X)* ∩ *insert Q(Y)*
End If

Several experiments have been conducted to compare the integrity tests of the proposed method with the related existing techniques (Madiraju and Sunderraman, 2004; Hanandeh et al., 2004; Hanandeh, 2006) that derive integrity tests in conjunctive or disjunctive normal forms. In terms of the types of integrity tests, some schemes provides sufficient test only. On the other hand, Madiraju and Sunderraman (2004) provide complete tests only. Our proposed algorithm and (Hanandeh, 2006) are the only two

techniques that generate sufficient and complete tests but it is limited to conjunctive normal forms. However, the new proposed algorithm generates more complete tests and considers all conjunctive and disjunctive normal forms. Hence, the new algorithm outperforms all the existing technique in this field.

7 Conclusions

In this paper, we introduced a simple and computationally inexpensive constraint checking algorithm in distributed database systems. The proposed technique first generates a simplified form of integrity constraints (integrity tests) from initial constraint and an update operation. The generated complete and sufficient tests are used to automatically verify that database updates does not introduce any violation of integrity. We considered all types of integrity constraints including domain, key, referential, simple/complex general semantic, and simple/complex transition. The performance of the proposed algorithm was evaluated through extensive experiments that clearly showed significant additional performance gains in comparison with existing techniques. For future work, we plan to validate the appropriateness of the produced integrity tests. A complete solution is needed for ranking and selecting the suitable test(s) from several alternatives.

References

Alwan, A., Ibrahim, H. and Udzir, N. (2008) 'A framework for checking integrity constraints in a distributed database', *Proc. of the Int. Conf. on Convergence and Hybrid Information Technology*, pp.644–650, Korea.

Alwan, A., Ibrahim, H. and Udzir, N. (2009) 'Improved integrity constraints checking in distributed database by exploiting local checking', *J. of Computer Science and Technology*, Vol. 24, No. 4, pp.665–674.

Alwan, A., Ibrahim, H. and Udzir, N. (2010) 'A model for ranking and selecting integrity tests in a distributed database', *Int. J. of Inf. Tech. and Web Engineering*, Vol. 3, No. 3, pp.65–84, University of Milan, Italy.

Belt, P., Harkonen, J., Mottonen, M., Kropsu-Vehkapera, H. and Haapasalo, H. (2010) 'Technological uncertainty and verification and validation activities', *Int. J. of Innovation and Learning*, Vol. 7, No. 2, pp.223–243.

Christiansen, H. and Martinenghi, D. (2006) 'On simplification of database integrity constraints', *Fundamenta Informaticae*, Vol. 71, No. 2, pp.371–417.

Christiansen, H. and Rekouts, M. (2007) 'Integrity checking and maintenance with active rules in XML databases', *Proc. of BNCOD Web Workshop on Web Information Management*, pp.59–67, Glasgow.

Grefen, P. (1992) 'Integrity control in parallel database systems', PhD dissertation, University of Twente, Netherlands.

Hanandeh, F. (2006) 'Integrity constraints maintenance for parallel databases', PhD dissertation, University of Putra, Malaysia.

Hanandeh, F., Ibrahim, H., Mamat, A. and Johari, R. (2004) 'Virtual rule partitioning method for maintaining database integrity', *Int. Arab J. of Information Technology*, Vol. 1, No. 1, pp.103–108.

Hannola, L., Elfvengren, K. and Tuominen, M. (2010) 'A group support system process for the definition of software requirements', *Int. J. of Innovation and Learning*, Vol. 7, No. 2, pp.171–186.

Ibrahim, H. (2002a) 'A strategy for semantic integrity checking in distributed databases', *Proc. Int. Conf. on Parallel and Distributed Systems*, pp.139–144, Taiwan.

Ibrahim, H. (2002b) 'Extending transactions with integrity rules for maintaining database integrity', *Proc. Int. Conf. on Info. and Knowledge Engineering*, pp.341–347, Las Vegas.

Kim, J., Shim, J. and Ahn, K. (2011) 'Social network service: motivation, pleasure, and behavioral intention to use', *J. of Computer Information Systems*, Vol. 51, No, 4, pp.92–101.

Madiraju, P. and Sunderraman, R. (2004) 'A mobile agent approach for global database constraint checking', *Proc. of the ACM Symposium on Computing*, pp.679–683, Nicosia.

Madiraju, P., Sunderraman, R. and Haibin, W. (2006) 'Framework for global constraint checking involving aggregates in multi databases using granular computing', *Proc. of IEEE Int. Conf. on Granular Computing*, pp.506–509, Atlanta.

Madiraju, P., Sunderraman, R. and Wang, H. (2006) 'Semantic integrity constraint checking for multiple XML databases', *J. of Database Management*, Vol. 17, No. 1, pp.1–19.

Martinenghi, D. (2005) 'Advanced techniques for efficient data integrity checking', PhD dissertation, Roskilde University, Denmark.

McCarroll, N. (1995) 'Semantic integrity enforcement in parallel database machines', PhD dissertation, University of Sheffield, UK.

Papadakis, N., Christodolou, Y., Sartzetakis, P. and Papadakis, K. (2011) 'Constraint satisfaction in XML temporal databases', *Int. J. of Reasoning-Based Intelligent Systems*, Vol. 3, No. 1, pp.44–58.

Salwani, A., Laleh, G. and Mohd, Z. (2011) 'Re-heat simulated annealing algorithm for rough set attribute reduction', *Int. J. of the Physical Sciences*, Vol. 6, No. 8, pp.2083–2089.

Sandhu, K. (2011) 'The use of qualitative evidence in e-services systems implementation', *Int. J. of Innovation and Learning*, Vol. 9, No. 1, pp.63–81.

Soumya, B., Madiraju, P. and Ibrahim, H. (2008) 'Constraint optimization for a system of relation databases', *Proc. of the IEEE Int. Conf. on Computer and Info. Technology*, pp.155–160, Sydney.

Wen, C., Prybutok, V. and Xu, C. (2011) 'An integrated model for customer online repurchase intention', *J. of Computer Information Systems*, Vol. 52, No. 1, pp.14–23.