Generating Repair Rules for Database Integrity Maintenance

Feras Hanandeh and Yaser Quasmeh

Prince Al-Hussein bin Abdullah II Faculty of Information Technology
Hashemite University, Jordan
feras@hu.edu.jo, Y_quasmeh@yahoo.com

Abstract. Repair system has two essential components, which are much related to each other. When the update operation is executed, the first component is the detection of the erroneous state if any and the second component is to repair this state by finding the changes to the update operation that would repair it. Failing to have the second component, which is the repair action, will enforce the user to manually correcting and reentering an erroneous update operation. Our approach will take advantage of the integrity before the update operation, which will result on limiting the detection only to the database state after the update operation. Also the repair component will take advantage of the integrity before the update operation and integrity violation after the update operation but before the repair. The focus of this paper is to generate repairs for all first order constraints, and by using only substitution with no resolution search. Multiple constraints can be satisfied in parallel without a sequential process with no possibility of cyclic violation.

1 Introduction

The reliability of information systems is a major concern for today's society and enterprises. The correctness or maintaining database integrity of databases is one of the main reliability issues. Consequently procedures asserting correct databases are a chief focus of research. Today the prime obstacles applying these procedures are their high computational costs. Integrity maintenance is considered one of the major application fields of rule triggering systems. In the case of a given integrity constraint being violated by a database transition these systems trigger update operation (action) to maintain database integrity.

A relational database is a collection of relations; each relation corresponds to a database predicate. Each relation R is a collection of tuples. Any attempt to update the database should be controlled by integrity constraints. When any of these constraints is violated by an update operation then the system should either abort or take action to repair the erroneous update operation. Such system is called integrity maintenance subsystem. When detecting any erroneous update operation, repairing become essential since detection without repairing the erroneous state will never accommodate users need to guarantee consistency, accuracy and the integrity of their systems. The integrity maintenance subsystem separate the database state into two states, the first is before the update operation. The second state is after the update operation, so the integrity maintenance subsystem has to detect any new errors introduced by the update operation and if there is any error to be repaired. Our approach involves algebraically modifying the constraint definitions into derivative expressions that return the condition for a new violation to occur. The derived predicate is a predicate defined in terms of the database predicates. The derived predicate, which denotes a violation of database constraint, is considered as a negation of the constraint.

2 Related Work

Relational Databases usually contain massive collections of data that rapidly evolve over time; this makes perfect checking at each update too time consuming a task to be feasible. In this regard, DBMS needs to be extended with the ability to automatically verify that database updates do not introduce any violation of integrity [3; 11].

The primary tool of integrity maintenance subsystem is the database integrity rules. The aim of integrity rules is to capture the semantics of data. Integrity rules provide a much more general capability to maintain integrity than the data models since they can utilize the full power of logic based language. The high cost results from using integrity rules may become as a restriction since they often involve the execution of complex queries against a large database.

Automation of the various repairable systems was the main aim for the researchers in the last decade. Partial automation was the aim of some researchers like [1, 4, 5, 16 and 18]. They adopted the notion of entrust the final repair to be manually designed by the users provided that the guidelines which they have to follow for the repair operation is clearly generated. Other approaches [14, 17 and 19] generated sufficient conditions for repair by the user entrusting to him the final repair to be manually designed by pruned the necessary repair, a suitable decision making framework based on encompassing all the actions requested to repair the erroneous state formulated, since there is not minimal repair actions. Some approaches resort to impose severe restrictions on the quantifier structure of the constraints like no existential quantifiers followed by universal quantifiers [1, 3 and 4].

Expensive rollback is the repair action adopted by many approaches [1, 4, 5, 14, 16 and 19] since executing of the update operation first was the condition for checking any possible integrity violations.

Soumya et al [15] proposed a technique to achieve optimization of constraint checking process in distributed databases by exploiting technique of parallelism, compile time constraint checking, localized constraint checking, and history of constraint violations. The architecture mainly consists of two modules: Constraint Analyzer and Constraint Ranker for analyzing the constraints and for ranking the constraints, respectively for systems with relational databases. They achieved optimization in terms of time by executing the constraints in parallel with mobile agents.

3 Preliminaries

Our approach has been developed in the context of relational databases. A relational database is a collection of relations, each corresponding to a database predicate. Each relation P is a collection of tuples Ti satisfying the corresponding predicate P, i.e., P (Ti) is True. An intentional (derived) predicate is a predicate defined in terms of the database predicates. Let V be an intentional predicate that denotes a violation of the database integrity constraint C, (i.e., V is the negation of C), where an integrity constraint is the primary tool of integrity maintenance system specifying a 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 transaction is defined as a collection of insertions into and deletions from the database.

Throughout this paper the same example Job Agency database is used, as given below. This example is taken from [19]:

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)

Definition 1. Given an update operation, for each database relation P, the incessant of the relation P, ΓP means that the tuples exists in relation P and not being deleted from relation P, such that:

$$\forall x\ (\Gamma P(x) \leftarrow P(x) \wedge \neg \delta P(x))$$

where δ is a deletion operator.

Definition 2. For every database relation P there are three different database states of the same relation, where P is the state of the relation before an update operation is performed, P' is the state of P after the update operation is performed and P" is the state of P after the repair operation is executed.

Definition 3. Given an update operation, for each database relation P, the new database state of the relation P' means that the tuples incessant in the database relation P, i.e. ΓP and the tuples to be inserted by the update operation, such that:

$$\forall x\ (P'(x) \leftarrow \Gamma P(x) \vee \iota P(x))$$

Assumption 1. Given an update operation, for each database relation P, it is assumed that:

- $\neg\iota P$ means that a tuple(s) were deleted from P, i.e. δP.
- $\neg\delta P$ means that a tuple(s) were inserted into P, i.e. ιP.

where ι is an insertion operator.

Given the fact that a pre-transaction state of a database is correct, a reduction of the amount of 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 a transaction. This is usually accomplished by the use of differential relations [6, 8, 9,12 and 13]. In this approach, two auxiliary relations are associated with each base relation P.

New differential relation. The new differential relation contains the set of tuples to be inserted into the relation P or the new modified tuples by the current update operation. The new differential relation associated with relation P is denoted as ιP and can be defined as:

$\forall x\ (\iota P(x) \leftarrow \neg P(x) \wedge P'(x))$, i.e. ιP means that the tuples that is not in the relation P , but in P'.

Old Differential Relation. The old differential relation contains the set of tuples to be deleted from the relation P or the old tuples that have been modified by the current update operation. The old differential relation associated with relation P is denoted as δP and can be defined as:

$\forall x\ (\delta P(x) \leftarrow P(x) \wedge \neg P'(x))$, i.e. δP means that the tuples that is in the relation P, but not in P'.

Using this auxiliary relations, constraint conditions can be reformulated.

Example. Given the domain integrity constraint C that states the placed attribute in relation Person must be either T or F

$$(\forall v, \forall w, \forall x)\ Person\ (v, w, x) \Rightarrow x = T \vee x = F$$

Given the fact that the constraint holds on the pre-transaction state of relation Person, only the new tuples to be inserted into Person have to be checked. Therefore, the constraint can be optimized as follows:

$$(\forall v, \forall w, \forall x)\ \iota Person\ (v, w, x) \Rightarrow x = T \vee x = F$$

4 Generating Integrity Tests

The main aim of this research is to contribute to the solution of constraint checking in a parallel database by deriving integrity tests from a given constraint and a given update operation. Using integrity tests to detect violations instead of integrity constraints is more fruitful because the later is costly since they often involve the execution of complex queries against a large database. These integrity tests may be necessary, sufficient or complete.

This section presents the algorithm which can be employed to derive more integrity tests than the previous approaches discussed in the related work section. The algorithm should be as general as possible, i.e. independent of any specific application domain. We take out the quantifiers Q for simplicity.

Lemma 1. Given a predicate P(X) in the form: $P(X) \leftarrow Q(Y) \wedge R(Z)$, $\delta P(X)$ will be computed as follows: $\delta P(X) \leftarrow (\delta Q(Y) \wedge R(Z)) \vee (\delta R(Z) \wedge Q(Y))$ i.e. either deleting a tuple(s) from the relation R or a tuple(s) from the relation Q will result in deleting a tuple(s) from the relation P. Deleting a tuples from both relations Q and R can be realized from the formula.

Proof

$\delta P(X) \leftarrow P(X) \wedge \neg P'(X)$
$\delta P(X) \leftarrow Q(Y) \wedge R(Z) \wedge \neg(Q'(Y) \wedge R'(Z))$
$\delta P(X) \leftarrow Q(Y) \wedge R(Z) \wedge (\neg Q'(Y) \vee \neg R'(Z))$
$\delta P(X) \leftarrow (\delta Q(Y) \wedge R(Z)) \vee (\delta R(Z) \wedge Q(Y))$

Lemma 2. Given a predicate P(X) in the form: $P(X) \leftarrow Q(Y) \vee R(Z)$, $\delta P(X)$ will be computed as follows: $\delta P(X) \leftarrow (\neg \Gamma Q(Y) \wedge \delta R(Z)) \vee (\neg \Gamma R(Z) \wedge \delta Q(Y))$ i.e. not existence a tuple(s) of both relations R and Q will result in deleting a tuple(s) from the relation P.

Proof

$\delta P(X) \leftarrow P(X) \wedge \neg P'(X)$
$P(X) \leftarrow (Q(Y) \vee R(Z)) \wedge \neg(Q'(Y) \vee R'(Z))$
$\leftarrow (Q(Y) \vee R(Z)) \wedge (\neg Q'(Y) \wedge \neg R'(Z))$
$\leftarrow (Q(Y) \wedge \neg Q'(Y) \wedge \neg R'(Z)) \vee (R(Z) \wedge \neg Q'(Y) \wedge \neg R'(Z))$
$\leftarrow (\delta Q(Y) \wedge \neg(R(Z) \wedge \neg\delta R(Z) \vee \iota R(Z))) \vee (\delta R(Z) \wedge \neg(Q(Y) \wedge \neg\delta Q(Y) \vee \iota Q(Y)))$
$\leftarrow (\delta Q(Y) \wedge (\neg R(Z) \vee \delta R(Z)) \wedge \neg\iota R(Z))) \vee (\delta R(Z) \wedge (\neg Q(Y) \vee \delta Q(Y) \wedge \neg\iota Q(Y)))$
$\leftarrow (\delta Q(Y) \wedge (\neg R(Z) \wedge \neg\iota R(Z) \vee \delta R(Z) \wedge \neg\iota R(Z))) \vee (\delta R(Z) \wedge (\neg Q(Y) \wedge \neg\iota Q(Y) \vee \delta Q(Y)) \wedge \neg\iota Q(Y)))$
$\leftarrow (\neg R(Z) \wedge \neg\iota R(Z) \wedge \delta Q(Y)) \vee (\delta R(Z) \wedge \neg\iota R(Z) \wedge \delta Q(Y)) \vee (\neg Q(Y) \wedge \neg\iota Q(Y) \wedge \delta R(Z)) \vee (\delta Q(Y) \wedge \neg\iota Q(Y) \wedge \delta R(Z))$
$\leftarrow (\neg R(Z) \wedge \neg\iota R(Z) \wedge \delta Q(Y)) \vee (\delta R(Z) \wedge \delta R(Z) \wedge \delta Q(Y)) \vee (\neg Q(Y) \wedge \neg\iota Q(Y) \wedge \delta R(Z)) \vee (\delta Q(Y) \wedge \delta Q(Y) \wedge \delta R(Z))$
$\leftarrow (\neg R(Z) \wedge \neg\iota R(Z) \wedge \delta Q(Y)) \vee (\delta R(Z) \wedge \delta Q(Y)) \vee (\neg Q(Y) \wedge \neg\iota Q(Y) \wedge \delta R(Z)) \vee (\delta Q(Y) \wedge \delta R(Z))$
$\leftarrow (\neg R(Z) \wedge \neg\iota R(Z) \wedge \delta Q(Y)) \vee (\neg Q(Y) \wedge \neg\iota Q(Y) \wedge \delta R(Z)) \vee \delta Q(Y) \wedge\delta R(Z)))$
$\leftarrow \neg(R(Z) \vee \neg\delta R(Z)) \wedge \delta Q(Y)) \vee (\neg Q(Y) \wedge \neg\iota Q(Y) \wedge \delta R(Z)) \vee \delta Q(Y) \wedge\delta R(Z)))$

The process of generate integrity tests starts by accepting an integrity constraint from the constraint base and a given update operation by the user. Figures 1 and 2 present the algorithms to generate integrity tests for constraints in conjunctive and disjunctive normal form respectively. Every generated integrity test is distributed to all dynamic virtual partitions [7] through generate distributed integrity test algorithm. This algorithm not presented in this paper because of space constraint. A clarified example to illustrate our technique followed the algorithms.

Checking the validity of the integrity tests against the database is activated at run-time once these tests are generated (computed) by logical rules at compile-time. These logical rules are specified using logical specification language. Each generation rule (conjunctive or disjunctive) has three input expressions. The output of the generation rule is sufficient, necessary or complete test(s).

```
Algorithm (Generate Texp for a constraint in a conjunctive normal form)
Inputs:   Constraint C in conjunctive normal form   /*$V \leftarrow P(X) \wedge Q(Y)$*/
            Update Operation (UO)
Output: Texp                     /*Integrity Test Expression*/
Method:
Begin
Step 1:
Exp1 $\leftarrow \Gamma P(X) \wedge \iota Q(Y)$   /*incessant of P(X) and insert a tuple into base relation Q */
where: $\Gamma P \leftarrow P \wedge \neg\delta P$
Exp2 $\leftarrow \Gamma Q(Y) \wedge \iota P(X)$   /*incessant of Q(Y) and insert a tuple into base relation P */
where: $\Gamma Q \leftarrow Q \wedge \neg\delta Q$
Exp3 $\leftarrow \iota P(X) \wedge \iota Q(Y)$   /*insert two tuples into both base relations P and Q */
Step 2:
Texp $\leftarrow$ Exp1 $\vee$ Exp2 $\vee$ Exp3
Step 3:
Negate Texp
End.
```

Fig. 1 Generate Texp algorithm for a constraint in a conjunctive normal form

```
Algorithm (Generate Texp for a constraint in a disjunctive normal form)
Inputs: Constraint IC in disjunctive normal form              /*$V \leftarrow P(X) \vee Q(Y)$*/
 Update Operation (UO)
Output: Texp                     /*Integrity Test Expression*/
Method:
Begin
Step 1:
Exp1 $\leftarrow \iota P(X) \wedge \neg Q(Y)$      /*insert a tuple into the base relation P and Q(Y) is false*/
Exp2 $\leftarrow \neg P(X) \wedge \iota Q(Y)$     /* P(X) is false and insert a tuple into the base relation Q */
Exp3 $\leftarrow \iota P(X) \wedge \iota Q(Y)$       /*insert two tuples into both base relations P and Q */
Step 2:
Texp $\leftarrow$ Exp1 $\vee$ Exp2 $\vee$ Exp3
Step 3:
Negate Texp
End.
```

Fig. 2 Generate Texp algorithm for a constraint in a disjunctive normal form

We now present detailed example to generate Texp algorithm which will clarify the steps presented above.

Example: when a company offers a Director job it must offer a Senior Secretary job first.

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

where $S(x) \leftarrow \text{Offering}(x, p, q) \wedge \text{Job}\ (p, \text{'Senior Secretary'})$

Update: $\text{Insert}(\text{Offering}(x, y, z)) = \{x/C, y/J, z/N\}$

$$C1 \leftarrow (\forall x \forall y \forall z)(\text{Offering}(x, y, z) \wedge M(x, y))$$

where $M(x, y) \leftarrow \text{Job}(y, \text{Director}) \wedge \neg S(x)$

Step1

$\text{Exp1 (Texp)} \leftarrow \Gamma\text{Offering}(x, y, z) \wedge \iota M(x, y)$
$\text{Exp2 (Texp)} \leftarrow \Gamma M(x, y) \wedge \iota\text{Offering}(x, y, z)$
$\text{Exp3 (Texp)} \leftarrow \iota\text{Offering}(x, y, z) \wedge \iota M(x, y)$

Computing $\delta S(x)$ using Lemma1

```
S(x) ← Offering(x, p, q) ∧ Job (p, 'Senior Secretary')
Exp1 ← δOffering(x, p, q) ∧ Job(p, 'Senior Secretary')
       ← False ∧ Job(p, 'Senior Secretary')
       ← False
Exp2 ← Offering(x, p, q) ∧ δJob(p, 'Senior Secretary')
       ← Offering(x, p, q) ∧ False
       ← False
δS(x) ← Exp1 ∨ Exp2
δS(x) ← False ∨ False
δS(x) ← False
```

Computing $\delta M(x, y)$ using Lemma1

```
M(x, y) ← Job(y, 'Director') ∧ ¬S(x)
Exp1 ← Job(y, 'Director') ∧ ιS(x)
Exp2 ← ¬S(x) ∧ δJob(y, 'Director')
δM(x, y) ← Exp1 ∨ Exp2
δM(x, y) ← Job(y, 'Director') ∧ ιS(x) ∨ False
δM(x, y) ← Job(y, 'Director') ∧ ιS(x)
```

Computing $\iota M(x, y)$

```
M(x, y) ← Job(y, 'Director') ∧ ¬S(x)
Exp1 ← ΓJob(y, Director) ∧ ¬ιS(x)
       ← ΓJob(y, Director) ∧ δS(x)
       ← ΓJob(y, Director) ∧ False
       ← False
Exp2 ← ¬ΓS(x) ∧ ιJob(y, 'Director')
       ← ¬(S(x) ∧ ¬δS(x)) ∧ ιJob(y, 'Director')
       ← (¬S(x) ∨ δS(x)) ∧ ιJob(y, 'Director')
       ← (¬S(x) ∨ δS(x)) ∧ False
       ← False
Exp3← ιJob(y, 'Director') ∧ δS(x)
      ← False ∧ δS(x)
      ← False
ιM(x, y) ← Exp1 ∨ Exp2 ∨ Exp3
ιM(x, y) ← False ∨ False ∨ False
ιM(x, y) ← False
```

Computing ιS(x)

```
S(x) ← Offering(x, p, q) ∧ Job(p, ‘Senior Secretary’)
Exp1 ← ΓOffering(x, p, q) ∧ ιJob(p, ‘Senior Secretary’)
Exp2 ← ΓJob(p, ‘Senior Secretary’) ∧ ιOffering(x, p, q)
Exp3 ← ιOffering(x, p, q) ∧ ιJob(p, ‘Senior Secretary’)
ιS(x) ← Exp1 ∨ Exp2 ∨ Exp3
ιS(x) ← False ∨ Job(J, ‘Senior Secretary’) ∧ x = C ∨ False
ιS(x) ← Job(J, ‘Senior Secretary’) ∧ x = C
```

Substituting the result of ιS(x) into δM(x, y):

δM(x, y) ← Job(y, ‘Directory’) ∧ Job(J, ‘Senior Secretary’) ∧ x = C

Substituting the results of ιM(x, y) into Exp1 (Texp) and Exp3 (Texp):

Exp1 (Texp) ←False
Exp3 (Texp) ←False
Exp2 (Texp) ← ΓM(x, y) ∧ ιOffering(x, y, z)
← M(x, y) ∧ ¬δM(x, y) ∧ ιOffering(x, y, z)
← Job(y, ‘Director’) ∧ ¬S(x) ∧ ¬(Job(y, ‘Director’) ∧ Job(J, ‘Senior Secretary’) ∧ x = C) ∧ (x, y, z) = (C, J, N)

Step2

Texp← Job (J, ‘Director’) ∧ ¬S(C) ∧ (¬Job(J, ‘Director’) ∨ ¬Job(J, ‘Senior Secretary’) ∨ ¬(C = C))
By substitution and negation rule
Texp ← Job(J, ‘Director’) ∧ ¬S(C) ∧ ¬ Job(J, ‘Senior Secretary’)

Step3

¬Texp ← ¬Job(J, ‘Director’) ∨ S(C) ∨ Job(J, ‘Senior Secretary’)
where
S(C) ← Offering(C, p, q) ∧ Job(p, ‘Senior Secretary’)

A number of sufficient tests can be computed by applying the substitution and resolution rules[2] to the sufficient and complete integrity tests in ¬Texp and the original constraint C1. These sufficient tests are often easier to test than the complete tests, and only one of them needs to be tested to prove that there are no violations. Given integrity constraint C, let negation of its violation be ¬Texp ← SCT, where SCT is the sufficient and complete tests for integrity checking. ¬Texp ← SCT ∨ C1 from identity rules[2], since C is not violated from the assumption of integrity before the transaction. A strengthen of our method is the more generated number of useful integrity tests than the previous methods like in [McC95]. The following sufficient tests are generated for the previous example.

¬Texp ← ¬Job(J, ‘Director’) ∨ S(C) ∨ Job(J, ‘Senior Secretary’)
∨ Offering(x, y, z) ∧ Job(y, ‘Director’) ∧ ¬S(x) /*assuming C1 is not violated before the transaction*/

$\vee$ Offering(C, y, z) $\wedge$ Job(y, 'Director') $\vee$ Offering(x, J, z) $\wedge$ $\neg$S(x) $\vee$ Offering(C, J, z)

Since all the tests distributed over OR, the satisfaction of any disjunct alone is sufficient for integrity. $\neg$Job(J, 'Director') $\vee$ S(C) $\vee$ Job(J, 'Senior Secretary') is the complete and sufficient tests, Offering(x, y, z) $\wedge$ Job(y, 'Director') $\wedge$ $\neg$S(x) is the original constraint, Offering(C, y, z) $\wedge$ Job(y, 'Director') and Offering(x, J, z) $\wedge$ $\neg$S(x) are subsumed tests. Hence, Offering(C, J, z) is the only new sufficient test.

5 Conclusion

Increasing the semantic content of the database model and a separate integrity maintenance subsystem are two approaches to maintaining integrity in database systems. The former leads to additional complexity for the users. The later creates additional overheads for the system. Separating integrity maintenance subsystem is more useful in minimizing the complexity faced by the users, since the overhead on the system can be managed and carefully optimized. It detects errors caused by database update operations and computes the repairs for these errors. The computed repairs are attached to the original erroneous update operation to create a correct and complete update operation. Our approach generates all minimal repairs to be presented to the user or the system administrator to select one of them to correct the update operation.

References

[1] Ceri, S., Widom, J.: Deriving Production Rules for Constraint Maintenance. In: Very Large Data Bases Conference, vol. 16, pp. 566–577 (1990)

[2] Chang, C., Lee, R.C.: Symbolic Logic and Mechanical Theorem Proving. Academic Press (1973)

[3] Christiansen, H., Martinenghi, D.: On simplification of database integrity constraints. Fundamental Informaticae 71(2), 371–417 (2006)

[4] Ceri, S., Fraternali, P., Paraborchi, S., Tanca, L.: Automatic Generation of Production Rules for Integrity Maintenance. ACM Transaction Database Systems 19(3), 366–421 (1994)

[5] Gertz, M., Lipeck, U.W.: Deriving Integrity Maintenance Triggers From Transaction Graphs. In: Ninth IEEE Conference Data Eng., pp. 22–30 (1993)

[6] Grefen, P.W.P.J.: Integrity Control in Parallel Database Systems. PhD Thesis, University of Twente (Netherlands) (October 1992)

[7] Hanandeh, F., Ibrahim, H., Mamat, A., Johari, R.: Virtual rule partitioning method for maintaining database integrity. Int. Arab J. of Information Technology 1(1), 103–108 (2004)

[8] Ibrahim, H.: Semantic Integrity Constraints Enforcement for Distributed Database. PhD Thesis, University of Wales College of Cardiff, Cardiff (UK) (June 1998)

[9] Ibrahim, H.: A Strategy for Semantic Integrity Checking in Distributed Databases. In: Proceedings of the 9th International Conference on Parallel and Distributed Systems (ICPADS 2002), Taiwan, December 17-20, pp. 139–144 (2002)

[10] Ibrahim, H.: Extending Transactions with Integrity Rules for Maintaining Database Integrity. In: Proceedings of the International Conference on Information and Knowledge Engineering, Las Vegas, USA, June 24-27, pp. 341–347 (2002)

[11] Martinenghi, D.: Advanced techniques for efficient data integrity checking. PhD dissertation, Roskilde University, Denmark (2005)

[12] McCarroll, N.F.: Semantic Integrity Enforcement in Parallel Database Machines. PhD Thesis, Department of Computer Science, University of Sheffield, Sheffield (UK) (May 1995)

[13] Moerkotte, G., Lockemann, P.C.: Reactive Consistency Control in Deductive Databases. ACM Trans. Database Systems 16(4), 670–702 (1991)

[14] Schewe, K.D., Thalheim, B., Schmidt, J.W., Wetzel, I.: Integrity Enforcement in Object Oriented Database. In: Modeling Database Dynamics, pp. 174–195 (1993)

[15] Soumya, B., Madiraju, P., Ibrahim, H.: Constraint optimization for a system of relational databases. In: Proc. of the IEEE Int. Conf. on Computer and Info. Technology, Sydney, pp. 155–160 (2008)

[16] Urban, S.D., Delcambre, L.M.: Constraint Analysis: A Design Process for Specifying Operations on Objects. IEEE Trans. Knowledge and Database Eng. 2(4), 391–400 (1990)

[17] Urban, S.D., Lim, B.B.L.: An Intelligent Framework for Active Support of Database Semantics. Int'l J. Expert Systems 6(1), 1–37 (1993)

[18] Wuethrich, B.: On Updates and Inconsistency Repairing in Knowledge Bases. In: IEEE Conference of Data Eng. (1993)

[19] Wang, X.Y.: The Development of a Knowledge-Based Transaction Design Assistant. PhD Thesis, Department of Computing Mathematics, University of Wales College of Cardiff, Cardiff, UK (1992)