RUN TIME REPAIR RULES FOR INTEGRITY MAINTENANCE SUBSYSTEM

FERAS HANANDEH
Faculty of Computer Science and Information Technology
University Putra Malaysia, 43400 UPM, Serdang,, MALAYSIA
hfiras@hotmail.com

ABSTRACT

Run time repair system has two essential components, which are very 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.

Keywords
Active database systems, Integrity constraints, Semantic integrity maintenance

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

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 a 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, 2, 3, 6 and 8]. 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 [5, 7 and 9] 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, 2 and 3].

Expensive rollback is the repair action adopted by many approaches [1, 2, 3, 5, 6 and 9] since executing of the update operation first was the condition for checking any possible integrity violations.

3. PRELIMINARIES

For every database predicate there are three states of the same predicate, where P is the state of relation before the update operation, $P'(X)$ is the state of P after the update operation and $P''(X)$ is the state of P after the repair.

Definition: Database after an update operations is performed ($P'(X)$) can be defined as $P'(X) \leftarrow (P(X) \wedge \neg \nabla P(X)) \vee \Delta p(X)$, means that when the tuple initially exist in the relation i.e. $P(X)$ and not deleted from the relation i.e. $\neg \nabla P(X)$ or inserted into the relation i.e. $\Delta P(X)$ this state represents the database relation after the update operation i.e. $P'(X)$.

Where the following symbols used in this paper means:

$\vee$: OR
$\wedge$: And
Δ:Insert a tuple
∇: Delete a tuple
$^{\Delta}$:Insert for repair
$_{\nabla}$: Delete for repair
$\neg$: Not

Definition: Database after executing a repair action ($P''(X)$) can be defined as $P''(X) \leftarrow (P'(X) \wedge \neg {}_{\nabla}P(X)) \vee {}_{\Delta}P(X)$, means that when the tuple exist after the update operation is performed in the relation i.e. $P'(X)$ and not deleted for repair from the relation i.e. $\neg {}_{\nabla}P(X)$ or inserted for repair into the relation i.e. ${}_{\Delta}P(X)$ this state represents the database relation after the repair i.e. $P''(X)$.

Definition. For every intentional predicate $P(X)$, defined by a rule $P(X) \leftarrow Q(Y)$, $P'(X)$ and $P''(X)$ are defined by the same rule executed using different states of the database:
$P'(X) \leftarrow Q'(Y)$ and $P''(X) \leftarrow Q''(Y)$

Database transactions were defined in literature as collections of insertions and deletions such that for each database relation $P(X)$, $\Delta P(X)$ and $\nabla P(X)$ are defined respectively as insertion into or deletion from a relation $P(X)$.$\Delta P(X)$ can be defined as $\Delta P(X) \leftarrow \neg P(X) \wedge P'(X)$, which means that the tuple does not exist in the initial predicate i.e. $P(X)$ but exist after the update operation i.e. $P'(X)$. This means that the tuple is inserted into the relation, $\Delta P(X)$. $\nabla P(X)$ can be defined as $\nabla P(X) \leftarrow P(X) \wedge \neg P'(X)$, which means that the tuple was exist in the initial predicate i.e. $P(X)$ and not exist after the update operation i.e. $P'(X)$ this state means that the tuple is deleted from the relation.

Repair update operation is executed when there is a violation caused by database update operations and it is considered as a collection of insertions into for repair (i.e. ${}_{\Delta}P$) and deletions from for repair (i.e. ${}_{\nabla}P$) the database relation.

${}_{\Delta}P(X) \leftarrow \neg P'(X) \wedge P''(X)$, means that the tuple does not exist after the update operation is performed i.e. $P'(X)$ but exist after the repair update operation i.e. $P''(X)$. This means that the tuple is inserted for repairing some violation ${}_{\Delta}P(X)$.

${}_{\nabla}P(X) \leftarrow P'(X) \wedge \neg P''(X)$, means that the tuple exist after the update operation is performed i.e. $P'(X)$ but does not exist after the repair update operation is performed i.e. $P''(X)$. This means that the tuple is deleted for repairing some violation from the relation ${}_{\nabla}P(X)$.

Throughout this paper the same example Job Agency database is used, as given below. This example is taken from [9].

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)

4. MAINTENANCE PREDICATES

Maintenance predicates directed for supporting maintenance of integrity by linking new violations to necessary repairs. The objective of our research is to compute the integrity maintenance predicate $\Delta_{\nabla}P(X)$ in terms of a repair update operation, given an arbitrary integrity constraint IC and an arbitrary update operation. $\Delta_{\nabla}P(X) \leftarrow \neg \Delta P(X) \vee {}_{\nabla}P(X)$, means that non-insertion into the initial database relation i.e. $\neg \Delta P(X)$ or if inserted, then deleted by the repair update operation i.e. ${}_{\nabla}P(X)$. $\nabla_{\Delta}P(X) \leftarrow \neg \nabla P(X) \vee {}_{\Delta}P(X)$, means that not deletion from the initial database relation i.e. $\neg \nabla P(X)$ or if deleted, then inserted by the repair update operation i.e. ${}_{\Delta}P(X)$.

For every intentional predicate P, defined by a rule: $P(X) \leftarrow Q(Y)$, follows the rules to compute repair actions for this intentional predicate is given in 6 intentional rules are:

1) $\Delta P(X) \leftarrow \Delta Q(Y) \wedge \neg P(X)$
Means that X is inserted into P i.e. $\Delta P(X)$ if Y is inserted into Q i.e. $\Delta Q(Y)$ and X is not already in P i.e. $\neg P(X)$.

2) ${}_{\Delta}P(X) \leftarrow {}_{\Delta}Q(Y) \wedge \neg P'(X)$
Means that X is inserted by the repair update operation into P i.e. ${}_{\Delta}P(X)$ if Y is inserted for repair into Q i.e. ${}_{\Delta}Q(Y)$ and X is not already in P' i.e. $\neg P'(X)$.

3) $\nabla P(X) \leftarrow \nabla Q(Y) \wedge \neg P'(X)$
Means that X is deleted from P i.e. $\nabla P(X)$ if Y is deleted from Q i.e. $\nabla Q(Y)$ and X is not already in P' i.e. $\neg P'(X)$.

4) ${}_{\nabla}P(X) \leftarrow {}_{\nabla}Q(Y) \wedge \neg P''(X)$
Means that X is deleted by the repair update operation from P i.e. ${}_{\nabla}P(X)$ if Y is deleted for repair from Q i.e. ${}_{\nabla}Q(Y)$ and X is not already in P'' i.e. $\neg P''(X)$.

5) $\Delta_{\nabla}P(X) \leftarrow \Delta_{\nabla}Q(Y) \vee P(X)$

Proof:
$\Delta_{\nabla}P(X) \leftarrow \neg\Delta P(X) \vee {}_{\nabla}P(X)$
$\neg(\Delta_{\nabla}P(X)) \leftarrow \neg(\neg\Delta P(X) \vee {}_{\nabla}P(X))$

By negation to both sides

$\neg(\Delta_{\nabla}P(X)) \leftarrow \Delta P(X) \wedge \neg{}_{\nabla}P(X)$
$\neg(\Delta_{\nabla}P(X)) \leftarrow \neg P(X) \wedge P'(X) \wedge P''(X)$
$\neg(\Delta_{\nabla}P(X)) \leftarrow \neg P(X) \wedge \neg Q(Y) \wedge Q'(Y) \wedge Q''(Y)$
$\neg(\Delta_{\nabla}P(X)) \leftarrow \neg P(X) \wedge \Delta Q(Y) \wedge \neg{}_{\nabla}Q(Y))$
$\neg(\Delta_{\nabla}P(X)) \leftarrow \neg\Delta_{\nabla}Q(Y) \wedge \neg P(X)$
$\Delta_{\nabla}P(X) \leftarrow \neg(\neg\Delta_{\nabla}Q(Y) \wedge \neg P(X))$
$\Delta_{\nabla}P(X) \leftarrow \Delta_{\nabla}Q(Y) \vee P(X)$

6) $\nabla_{\Delta}P(X) \leftarrow (\neg\nabla P(X) \vee {}_{\Delta}Q(Y)) \wedge \neg P'(X)$

Proof:

$\nabla_{\Delta}P(X) \leftarrow \neg\nabla P(X) \vee {}_{\Delta}P(X)$
$\nabla_{\Delta}P(X) \leftarrow \neg\nabla P(X) \vee {}_{\Delta}Q(Y) \wedge \neg P'(X)$

From the rule ${}_{\Delta}P(X) \leftarrow {}_{\Delta}Q(Y) \wedge \neg P'(X)$

$\nabla_{\Delta}P(X) \leftarrow \neg\nabla P(X) \vee {}_{\Delta}Q(Y) \wedge \neg P'(X)$

Example:

Given
P (name) ← Person (pid, pname, placed)
where
P contains the names of all persons:

ΔP (pname) ← ΔPerson (pid, pname, placed)$\wedge\neg$P (pname)

∇P (pname) ← ∇Person (pid, pname, placed)$\wedge\neg$P' (pname)

where
P' (pname) ← Person' (pid, pname, placed)
From the rule $P'(X) \leftarrow Q'(Y)$
where
Person' (pid, pname, placed) ← Person (pid, pname, placed) $\wedge\neg\nabla$ Person (pid, pname, placed) $\vee\Delta$ Person (pid, pname, placed)
From the rule $P'(X) \leftarrow (P(X) \wedge \neg\nabla P(X)) \vee \Delta P(X)$
∇P (pname) ← ∇Person (pid, pname, placed)$\wedge\neg$(Person (pid, pname, placed) $\wedge\neg\nabla$ Person (pid, pname, placed) $\vee\Delta$ Person (pid, pname, placed))

Incremental Integrity Maintenance

The critical predicate for incremental integrity maintenance is Δ_{∇}IC, which is empty maintenance for violations indicating no new violations introduced by the update operation or deletion by repair update operation of all new violations of integrity introduced by the update operation.

Our approach computes the integrity maintenance predicates Δ_{∇}IC in terms of a repair update operation; given an arbitrary constraint IC and an arbitrary update operation. The computation can be done before the execution of the update operation, and a repair action is attached to the original update operation, creating a correct and complete update operation.

Example

Assume that IC← Application (P, J) $\wedge\neg$Offering (c1, J, N) $\wedge\neg$Offering (c2, J, N) is the given constraint, which states that, either company c1 or company c2 must offer the job J for applied by person P.

ΔApplication (p1, j1) is the update operation.
Applying the rules we introduced, by substituting for the update operation an the database predicates,

Δ_{∇}IC← ${}_{\nabla}$Application (p1, j1) $\vee_{\Delta}$Offering (c1, j1, N) $\vee_{\Delta}$Offering (c2, j1, N)
Is the repair, which either aborts the update operation, or forces either company c1 or company c2 to offer j1.

Example

Assuming the current state of the database state before the update operation is:

Offering (c1, j5, no_of_places)
Job (j5, programmer)

C1$\leftarrow$Offering (c1, J, no_of_places) $\wedge$Job (J, technician)
Preventing company c1 from offering technician jobs

C2$\leftarrow\neg$Offering (c1, J, no_of_places) $\wedge$Job (J, programmer)

Requiring company c1 to offer all programmer jobs.
Consider this update operation:
T=Δjob (j5, technician)
Update operation will violate C1 and can be repaired by:

$_\nabla$Offering (c1, j5, no_of_places), but this repair action will violate C2, since Job (j5, programmer) is true in the database, so the complete repair would be:
$_\nabla$Offering (c1, j5, no_of_places) $\wedge_\nabla$Job (j5, programmer)

Finally either we delete Job (j5, technician) for repair i.e. $_\nabla$Job (j5, technician) or we delete for repair both Offering (c1, j5, no_of_places) and Job (j5, programmer) i.e. $_\nabla$Offering (c1, j5, no_of_places) $\wedge_\nabla$Job (j5, programmer)

Δ_∇IC$\leftarrow$ $_\nabla$Job (j5, technician) $\vee_\nabla$Offering (c1, j5, no_of_places) $\wedge_\nabla$Job (j5, programmer)

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. and Widom, J. 1990. Deriving Production Rules for Constraint Maintenance. *In* Very Large Data Bases Conference, vol.16, pp.566-577.
[2] Ceri, S., Fraternali, P., Paraborchi, S. and Tanca, L. 1994. Automatic Generation of Production Rules for Integrity Maintenance. *ACM Transaction Database Systems,* vol.19, no.3, pp.366-421.
[3] Gertz M. and Lipeck U.W. 1993. Deriving Integrity Maintenance Triggers From Transaction Graphs. *In* Ninth IEEE Conference Data Eng. pp. 22-30.
[4] Moerkotte, G. and Lockemann, P.C. 1991. Reactive Consistency Control in Deductive Databases. *ACM Trans. Database Systems*, vol. 16, no. 4, pp. 670-702.
[5] Schewe, K.D., Thalheim, B., Schmidt, J.W. and Wetzel, I. 1993. Integrity Enforcement in Object Oriented Database. *In* Modeling Database Dynamics, pp. 174-195.
[6] Urban, S.D. and Delcambre, L.M. 1990. Constraint Analysis: A Design Process for Specifying Operations on Objects. *IEEE Trans. Knowledge and Database Eng.* Vol.2, no.4, pp.391-400.
[7] Urban, S.D. and Lim, B.B.L. 1993. An Intelligent Framework for Active Support of Database Semantics. *Int'l J. Expert Systems*, vol.6, no.1, pp.1-37.
[8] Wuethrich, B. 1993. On Updates and Inconsistency Repairing in Knowledge Bases. *In* IEEE Conference of Data Eng.
[9] Wang, X.Y. 1992. *The Development of a Knowledge-Based Transaction Design Assistant.* PhD Thesis, Department of Computing Mathematics, University of Wales College of Cardiff, Cardiff (UK).