Knowledge Discovery for Trigger Conflict Resolution

Kittisak Kerdprasop and Nittaya Kerdprasop

School of Computer Engineering, Suranaree University of Technology
111 University Avenue, Muang District, Nakorn Ratchasima 30000, THAILAND
kerdpras, nittaya@ccs.sut.ac.th

ABSTRACT

Active behavior of the active database systems is obtained through the set of trigger rules, also known as the event-condition-action rules. The event, such as the modification of a database state, can activate the triggers. If the trigger's condition is satisfied, then the corresponding actions are performed. The problem may arise if several trigger rules are eligible for execution. To solve the problem of trigger rule conflict, the database management system must provide a conflict resolution policy to select a trigger rule for execution. We propose a conflict resolution scheme that incorporates derived knowledge, which is induced from the database content, as a major part of the trigger rule prioritization [1]. By means of the trigger scheduling, deterministic behavior of the trigger processing can be guaranteed.

Keywords: knowledge discovery, triggers, conflict resolution, active databases

1 INTRODUCTION

Active database systems extend the traditional database systems with the mechanisms that enable them to respond automatically to some specific events. The events may take place either inside or outside the database system. Active behavior of the database system is commonly supported through event-condition-action (ECA) rules, also known as triggers. ECA rule, or trigger, mainly consists of three parts to describe the event, condition and action, respectively. Upon the occurrence of the specified event, the rule condition is evaluated. If the condition is satisfied then some actions are performed. Active database systems can automatically react to events such as database modification, time events, and external signals (from the application). This is in contrast to traditional database systems that execute queries and transaction only when explicitly requested to do so.

The importance of integrating reactive behavior into the database systems has been recognized since the 1970s [10]. However, it was not until the late 1980s to early 1990s that the area of active databases has catched high interest among researchers [6, 8, 11, 15, 17]. At present most commercial database systems, such as Oracle, IBM DB$_2$, Informix, Sybase, support the simple forms of triggers and incorporation of triggers for commercial object-oriented databases is underway [5, 14]. TheSQL3 standard [9, 12] has extensive coverage of triggers.

Although trigger is a powerful mechanism of the active database systems, designing and writing correct trigger rules are not an easy and straightforward task. The difficulty is due to the complex and sometime unpredictable behavior of the triggers. Poorly designed triggers can activate each other indefinitely, which leads to the non-terminate execution. Several methods have been proposed [1, 4, 13, 16] to analyze trigger behavior at compile time and at runtime [3]. Another problem regarding trigger behavior is the deterministic property of the triggers. Deterministic trigger processing guarantees the same order of execution when several trigger rules are activated simultaneously. We propose a mechanism to utilize domain-knowledge in choosing among triggered rules.

The outline of this paper is as follow. Section 2 briefly describes the components of the trigger rules and their potential applications. Section 3 discusses the problem of non-deterministic processing due to the trigger conflict. Section 4 presents the algorithm to derive the knowledge for resolving the conflict problem. Section 5 concludes the paper and indicates the future work.

2 TRIGGERS AND THEIR APPLICATIONS

In SQL3 [9, 12], triggers are expressed by means of event-condition-action rules, as presented in Figure 1. Each trigger is identified by a name. It is possible to specify whether a trigger must be executed BEFORE or AFTER its triggering event. SQL3 triggers allow only the INSERT, DELETE, and UPDATE as triggering event, and limit to a single event be monitored per single trigger rule. REFERENCING clause allows trigger to access to the old and the new attribute values affected by the triggering event. Affected data item can be seen individually (OLD and NEW) or jointly as a temporary extent (OLD_TABLE and NEW_TABLE). The SQL3 triggers provide a notion of granularity to define how many times the trigger is

[1] The work reported in the paper was supported by the grant from the National Electronics and Computer Technology Center (NECTEC).

executed for the particular event. FOR EACH ROW refers to a row-level trigger, which is executed on each tuple modification of the triggering event. FOR EACH STATEMENT is a statement-level trigger, which is executed once for an event regardless of the number of tuples affected.

```
<SQL3-trigger> ::=
        CREATE TRIGGER <name>
        { BEFORE | AFTER } <event>
        ON <table-name>
        [ REFERENCING { OLD [AS] <old-value-tuple-name>
            | NEW [AS] <new-value-tuple-name>
            | OLD_TABLE [AS] <old-value-table-name>
            | NEW_TABLE [AS] <new-value-table-name> } ]
        [ FOR EACH { ROW | STATEMENT}]
        [ WHEN <condition> ]
        <action>

<event> ::= INSERT| DELETE| UPDATE [OF <column-names>]
```

Figure 1: Definition of SQL3 triggers

The WHEN clause specifies an additional condition to be checked once the trigger rule is fired and before the action is executed. Conditions are predicates over the database state. If the WHEN clause is missing, the condition is supposed to be true and the trigger action is executed as soon as the trigger event occurs. The action is executed when the rule is triggered and its condition is true. SQL3 allows multiple action statements in triggers, each of which is executed according to the order they are written. Actions are stored procedures and may include SQL statements, control constructs, and calls to user-defined functions.

The following example shows a trigger rule to impose a constraint on the database that the salary of any employee may never decrease.

Example 1: Trigger rule to guarantee no decrease on employees' salaries.

```
CREATE TRIGGER emp-salary-no-decrease
BEFORE UPDATE OF Employee
FOR EACH ROW
  WHEN (new.Salary < old.Salary)
     begin
        log the event;
        signal error condition;
     end
```

□

The potential applications of triggers are significant [7]: signal integrity constraint violation and force rollbacks of the violating transactions, maintain consistency across system catalogs or other metadata, notify users in the form of messages (alerters), implement business rules or workflow management, and many more.

3 TRIGGER CONFLICT ISSUE

The behavior of triggers is defined as the "execution model." It specifies how trigger rules are evaluated and treated at runtime. Figure 2 illustrates the steps in processing triggers [15].

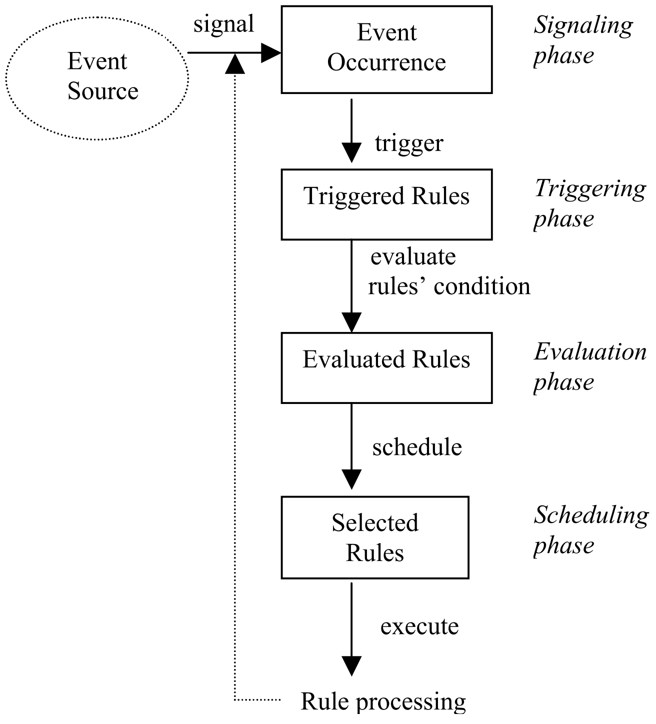


Figure 2: Steps in trigger rule execution

The signaling phase detects and signals the occurrence of an event. The event activates the corresponding trigger rules in the triggering phase, and the condition parts of the triggered rules are evaluated in the evaluation phase. The **trigger conflict problem** occurs when the conditions of more than one trigger rules are evaluated to be true. The scheduling phase indicates the order to process conflict triggers. The execution phase processes the scheduled trigger rules. On processing rule's action, the change in a database state may trigger another or even the same set of rules.

After the evaluation phase, more than one rule may be eligible for execution. This problem is known as *trigger rule conflict* [3,15]. To solve the problem, the database management system must provide a *conflict resolution policy* to select a trigger rule for execution. The common conflict resolution policy adopted by most systems is assigning rule priority [14, 17]. The rule prioritization is based on either the recency of update or the complexity of the rule's condition. The former approach assigns a high priority to the rule that is most recently fired. The latter approach considers the priority on the specificity of the rule. The two approaches are dynamic, which means they are less practical in the system with large and complex trigger rule set. When deterministic behavior is highly desirable, the scheme to associate rules with priority statically is more appropriate.

Static priority mechanism determines order of trigger rules either by the system (e.g., based on rule creation time) or by the user (e.g., explicitly associate each rule with a numeric value). We propose a conflict resolution mechanism to incorporate derived knowledge (i.e., the knowledge obtained from the database content) into the rule prioritization scheme.

4 ALGORITHM TO HANDLE TRIGGER CONFLICT

In this section, we define an algorithm to handle trigger conflict by reorganizing the trigger rules into different layers, or strata. Then, associate each stratum with a numeric priority. The major mechanism leading to priority assigning is the induced knowledge regarding the database state modification. Prior to the algorithm description, some definitions of terms are explained.

Definition 4.1 A *stratum* is an ordered set of trigger rules that locally converge. A stratum locally converges if after any transaction invoking trigger rules, rule processing terminates in a final state in which the set of triggered rules is empty.

□

Definition 4.2 *Stratum set* is an unordered set of strata in which each stratum is independent from other strata so that the trigger rule execution in one stratum does not affect rules in other strata.

□

The concept of stratum [2] is applied to guarantee that trigger rule execution eventually terminates. The example of non-terminate trigger execution due to the cycle in action-triggering events is shown in Figure 3. Breaking the cycle into different layers, as shown in Figure 4, and the local convergence within each stratum are two sufficient conditions for termination on trigger execution.

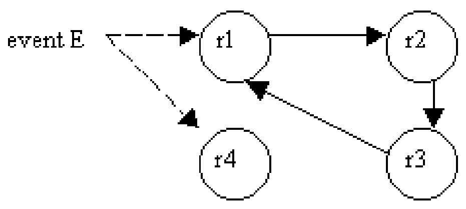


Figure 3: The cycle among trigger rules

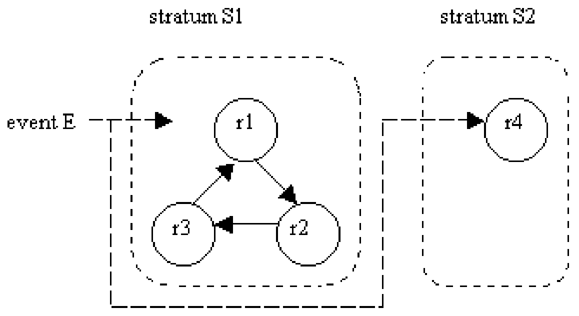


Figure 4: Organizing cyclic rules into strata

The next step is to set priorities among strata when several strata are activated by the occurrence of an event. We propose the trigger conflict resolution algorithm to solve the problem of multiple stratum activation on an event E. The key concept is to apply knowledge induced from the database modification and the metadata (i.e., the set of integrity constraints) to guide the priority assigning scheme. The function to compute priority for each stratum is defined as in the following definition.

Definition 4.3 A *Degree_of_Constraint* Function for the induced knowledge K comparing to the given set of integrity constraints IC is defined as:

$$Degree\_of\_Constraint(\mathrm{K}, \mathrm{IC}) = \frac{\#matched\_rules}{\#total\_rules} + \frac{A}{10}$$

where # *matched_rule* is the number of induced knowledge, represented in the format of rule, completely matching with the integrity constraint rule (i.e., NOT (k) AND c yields the contradiction when k ∈ K and c ∈ IC),

total_rules is the total number of induced knowledge when represented as rules, and
A is the average accuracy of the induced knowledge rules.

□

Note that the number "10" is proposed as a divider of the average accuracy (i.e., the part (A/10) in the computation of *Degree_of_Constraint*) in order to prevent the domination of the accuracy over the proportion of *matched_rules* and *total_rules*.

Algorithm 4.4 Trigger conflict resolution algorithm.

Given an unordered stratum set S, a relational active database R and its metadata, the algorithm returns an ordered stratum set O in which the priority of each stratum has been assigned.

Steps:

```
1. Active_Tuple_Set_i ← Activate( S_i , E)
       /* Activate every stratum S_i , S_i ∈ S, such that
       the occurrence of an event E can invoke its
       trigger rule(s), and record all affected tuples in
       the corresponding Active_Tuple_Set. */

2. K_i ← Induce(Active_Tuple_Set_i )
       /* The knowledge induction method (such as
       association-rule learning, decision-tree
       induction) is applied to induce knowledge from
       the content of each Active_Tuple_Set. The
       induced knowledge is stored in K_i. */

3. q_i = Degree_of_Constraint (K_i , metadata)
       /* Calculate the value q, or the
       Degree_of_Constraint, of each set of induced
       knowledge K comparing to the integrity
       constraints given as a metadata. */

4. sort(i, q)
       /* Apply any sorting algorithm on the q-value
       associated with each stratum S_i. */

5. return an ordered stratum set O = { S_i | S has been
   sorted by its index i }
```

□

Running Example

To explain the algorithm4.4, we assume the following database instances, the integrity constraints, and trigger rules are given.

Employee Database *R*:

Name	Position	Working _hours	Salary	Working _years	Project _manager
Jack	Senior_ programmer	18	72,000	10	Yes
Joe	Programmer	10	20,000	5	No
Jim	Programmer	12	40,000	8	No
Gupta	Programmer	15	40,000	7	No
Lee	Programmer	16	50,000	6	No
Carl	Computer_ engineer	18	68,000	12	No

Integrity Constraints *IC*:

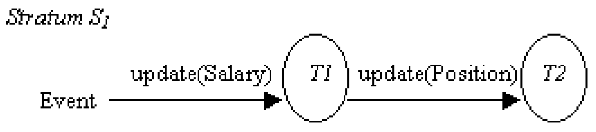
IC1: IF Position = "Computer_engineer"
THEN Salary >= 50,000
IC2: IF Position = "Senior_programmer"
THEN Project_manager = "Yes"
IC3: IF Position = "Supervisor"
THEN Project_manager = "Yes"

Trigger rules *T*:

```
T1: CREATE TRIGGER promote_position
    AFTER UPDATE OF Salary ON Employee
    FOR EACH ROW
    WHEN (Employee.Salary > 65000)
        Set Employee.Position = 'Supervisor'

T2: CREATE TRIGGER promote_manager
    AFTER UPDATE OF Position ON Employee
    FOR EACH ROW
    WHEN (Employee.Working_hours > 12)
          AND (Employee.Salary > 65000)
        Set Employee.Project_manager = 'Yes'
```

Triggers *T1* and *T2* work on the same set of database instances but they do not activate each other on a cyclic manner. Therefore, they can be grouped into the same stratum.



Step 1: $Active\_Tuple\_Set_i \leftarrow Activate(S_i , E)$
On occurrence of the event *E*, the *Active_Tuple_Set* contains the following two instances:

Name	Position	Working _hours	Salary	Working _years	Project_ manager
Jack	Supervisor	18	72,000	10	Yes
Carl	Supervisor	18	68,000	12	No

Step 2: $K_i \leftarrow$ *Induce*(*Active_Tuple_Set$_i$*)

Supply the above instances to the induction algorithm (e.g., the associative-rule learning). The induced knowledge rules are as follows:

1. Position=supervisor
==> Working_hours=18
acc:(1)
2. Working_hours=18
==> Position=supervisor
acc:(1)
3. Position=supervisor
==> Project_manager=yes
acc:(1)
4. Position=supervisor Working_hours=18
Project_manager=yes
==> Working_years=10
acc:(1)
5. Position=supervisor Working_years=10
Project_manager=yes
==> Working_hours=18
acc:(1)
6. Working_hours=18 Salary=72000 Working_years=10
Project_manager=yes
==> Position=supervisor
acc:(1)
7. Position=supervisor Salary=68000
Project_manager=no
==> Working_hours=18 Working_years=12
acc:(1)
8. Position=supervisor Working_years=12
Project_manager=no
==> Working_hours=18 Salary=68000
acc:(1)
9. Working_hours=18 Salary=68000 Working_years=12
==> Position=supervisor Project_manager=no
acc:(1)
10. Working_hours=18 Salary=68000
Project_manager=no
==>Position=supervisor Working_years=12
acc:(1)

Step 3: q_i = *Degree_of_Constraint (K_i , metadata)*

From the total of ten induced rules, only rule number 3 matches the *IC3*.

Compute the *Degree_of_Constraint*
= (*# matched_rules / # total_rules*) + (A / 10)
= (1 / 10) + (1 / 10)
= 0.2

Step 4: *sort(i, q)*

To keep the running trace understandable, we provide only two trigger rules that form themselves as a single stratum. Therefore, there is no need for sorting on the *Degree_of_Constraint*. In a more practical situation that triggers are grouped into several strata, each stratum will be sorted according to the *Degree_of_Constraint* values.

Step 5: *return an ordered stratum set O = { S_i | S has been sorted by its index i }*

The stratum *S1* has been returned with the priority 0.2 associated with it.

5 CONCLUSION AND FUTURE WORK

The trigger rule conflict occurs when an event activates several trigger rules simultaneously. To maintain the deterministic property of the active database processing, the database management system has to provide a conflict resolution policy. The common policy adopted by most systems is to assign rule priority. The rule prioritization is based on either the recency of update or the complexity of the rule's condition. We propose a different scheme of prioritization by taking into account the knowledge regarding the database state. Moreover, we consider priority at the level of stratum, which may contain several related triggers. The concept of stratification preserves the termination property of trigger rule processing.

The design of the trigger-conflict-resolution algorithm is the preliminary work toward the design and implementation of a tool to help database designer on designing and analyzing a complex set of triggers. A further investigation on a more practical active database with a larger trigger set is necessary.

REFERENCES

[1] A.Aiken, J.M.Hellerstein, and J.Widom. Static analysis techniques for predicting the behavior of active database rules. *ACM Transactions on Database Systems*, vol.20, no.1, pp.3-41, March 1995.

[2] E.Baralis, S.Ceri, and S,Paraboschi. Modularization techniques for active rule design. *ACM Transactions on Database Systems*, vol.21, no.1, pp.1-29, March 1996.

[3] E.Baralis, S.Ceri, and S,Paraboschi. Compile-time and runtime analysis of active behaviors. *IEEE Transactions on Knowledge and Data Engineering*, vol.10, no.3, pp.353-370, May/June 1998.

[4] E.Baralis and J.Widom. An algebraic approach to rule analysis in expert database system. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pp.475-486, Santiago, Chile, September 1994.

[5] E.Bertino, G.Guerrini, and I.Merlo. Triggers in Java-based databases. *L'object*, vol.6, no.3, 2000.

[6] A.Buchmann. Current trends in active databases: Are we solving the right problems. In *Information Systems Design and Multimedia, Proceedings of the Basque International Workshop on IT*, pp.121-133, 1994.

[7] S.Ceri, R.J.Cochrane, and J.Widom. Practical applications of triggers and constraints: Successes and lingering issues. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pp.254-262, Cairo, Egypt, 2000.

[8] S.Chakravarthy. Rule management and evaluation: An active DBMS perspective. *ACM SIGMOD Records*, vol.18, no. 3, pp.20-28, 1989.

[9] A.Eisenberg and J.Melton. SQL:1999, formerly known as SQL3. *ACM SIGMOD Records*, vol.28, no.1, pp.131-138, 2000.

[10] K.P.Eswaran. Specifications, implementations and interactions of a trigger subsystem in an integrated database system. IBM Research Report RJ1820, IBM San Jose Research Laboratory, San Jose, California, August 1976.

[11] E.N.Hanson and J.widom. An overview of production rules in database systems. *Knowledge Engineering Review*, vol.8, no.2, pp.121-143, 1993.

[12] ISO/IEC 9075-2: 1999 Information technology – Database language – SQL – Part 2: Foundation (SQL/Foundation), 1999.

[13] A.P.Karadimce and S.D.Urban. Conditional term rewriting as a formal basis for analysis of active database rules. In *Proceedings of the Fourth International Workshop on Research issues in Data Engineering, RIDE-Ads '94*, pp.156-162, Houston, February 1994.

[14] N.Paton. *Active Rules in Database Systems*. Springer-Verlag, 1999.

[15] N.W.Paton and O.Diaz. Active database systems. *ACM Computing Surveys*, vol.31, no.1, pp.63-103, March 1999.

[16] L.van der Voort and A.Siebes. Termination and confluence of rule execution. In *Proceedings of the Second International Conference on Information and Knowledge Management*, Washington, D.C., November 1993.

[17] J.Widom and S.Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, San Francisco, California, 1996.