THE ADD-ON MULTIPLATFORM AGENT FOR MULTITRIGGER ACTIVE RELATIONAL DATABASE SYSTEM

Pattarakitti Chaiyasing
Department of Computer Engineering Khon Kaen University, Thailand
pattarakitti@gmail.com

Wiroj Taweepworadej
Department of Computer Engineering Khon Kaen University, Thailand
wirtaw@kku.ac.th

ABSTRACT
This paper proposes the new design of the add-on agent that can be implemented on any passive relational database platforms in order to transform the system acting active in multitrigger manners without changing the application source codes. The system model includes the Connection Control module to investigate and prepare all the platform based required commands for all communication usages between program and the database system. This paper also presents the multitrigger concepts which can be implemented on the MySQL database platform. The system testing shows that the proposed multitrigger model outperforms in the average execution time comparing to the traditional multitrigger model having in the PostgreSQL database system, especially when the numbers of triggers running in the system increase.

Keyword: Multiplatform Add-on Agent, Multitrigger, Active RDBS

INTRODUCTION
There have been many researches focusing on the Active Relational Database System (ARDBS) issues in the past decades. To make the RDBS active, Widom had designed the active basic rule approach which could be integrated into the application source codes (1996). However, the Widom’s model is rigid so it can be applied only to the system that allows users to alter source codes. Lijuan et al. had proposed the concept of the mediated approach by implementing the Event-Condition-Action agent as an interface between clients and servers (1999).

On the other aspects, many researchers had studied for new approaches in order to enhance the active rule capabilities. Data mining technique was applied to analyze the relation between event rules and user defined triggers, then extract only the active event rules to set up the new efficient rules set (Dai M. et al., 2007). Abbas R. et al. increased the efficiency of the event detections by proposing the framework with an algorithm to compare, evaluate and set up the better order of event rules under 5 criteria (2006): average response time, response time variance, throughput, time overhead per transaction and CPU utilization.

In 2010, Sasithorn S. et al. had proposed an add-on agent to change a passive RDBS to be an active RDBS (2010). The advantages of this model are there is no requirement for the application source code changes and the active agent can be easily portable to
work with other PRDBS. However, single platform and single trigger issues are two major limitations in this study. The add-on agent model was designed to work only with the MySQL platform in which the RDBS will operate in the single trigger active manners. As a result, this paper has proposed the new add-on multiplatform agent model which can be implemented on any RDBS and turn the systems to be active in multitrigger manners.

The next two sections describe the models of the add-on multiplatform agent and the multitrigger ARDBS. The experiment design section is added to compare the performance of the two multitrigger ARDBS models between MySQL and PostgreSQL platforms followed by the results and discussions section. The last section is the summary section including with the future works.

CONCEPTUAL DESIGN OF THE PROPOSED MODEL

The Add-on Multiplatform Agent Model

To work on the different platforms, the active agent is designed to have three main modules: Connection Control (CC), Trigger Generator (TG) and Trigger Execution as shown in figure 1. To be able to connect to each database platforms, the active agent must learn all the commands and functions, which are included into the platform based library, used to communicate between program and database system, to retrieve database schema and also to generate system triggers.

The add-on multiplatform agent structure is described as figure 2. Users must implement an agent on the RDBS by setting database configuration via agent user interface and then create triggers afterward by giving simple event-condition-action information. The CC module will detect the database system platform and choose the
perfect-match library for agent usage. The TG module is designed for creating triggers by learning the database schema and setting active rules as defined by users. All fundamental information is stored into the active agent database for the TE module usage. Finally, The TE module will perform all the active tasks by initiating predefined triggers, evaluating events and executing actions when the designated event detected.

Figure 2 The Add-on Multiplatform Agent Structure

The Multitrigger Active Relational Database System Model

One major drawback of the MySQL ARDBS is that the system, for each table, allows only one trigger detecting an event before the table update occurs and one another detecting an event after the table modification. To implement the multitrigger manners into the MySQL platform, we design the system to have two triggers called ‘system triggers’ in order to control and manipulate all user defined triggers as shown in figure 3.
One system trigger includes all user defined triggers that detect events before the table modification and the other one includes all user defined triggers that detect events after the table update.

![Figure 3 MySQL Multitrigger Active Model](image)

Trigger: System Trigger

Trigger Body

UD_trigger_1

tg1_condition
tg1_action_procedure_call

UD_trigger_2

tg2_condition
tg2_action_procedure_call

UD_trigger_n

tgn_condition
tgn_action_procedure_call

Stored Procedures:
tg1_action_procedure
tg1_action_content
tg2_action_procedure
tg2_action_content

... 
tgn_action_procedure
tgn_action_content

Trigger in MySQL RDBS composed of two major components: trigger header and trigger body. Trigger header is used to describe event detection information including with detection-required table name, detection event and also detection order preference. The system trigger header must be defined to always match table event so the user defined trigger created within will be initiated. To create trigger, users will provide trigger condition and action which will be automatically implanted into the system trigger body. Each trigger action procedure is designed to be called by the command programmed within each trigger action part so the system can provide the better responses.

Unlike MySQL platform, PostgreSQL provides a functionality to create more than one trigger working on the same table within its RDBS. Users can create n triggers working independently and simultaneously as shown in figure 4. Three major components, which are trigger header, trigger condition and trigger action, must be defined in each trigger. Trigger header is used to describe the event detection details as same as we define in the MySQL trigger header. The trigger condition is defined separately from the trigger action content which is mostly used to mention the action performed when the event is detected. To enhance the trigger capability, users can
implement a trigger action function which will be called by the command put in the trigger action part.

![Figure 4 PostgreSQL Multitrigger Active Model](image)

**SYSTEM TESTING**

The system experiment was designed with the purpose to investigate the performance of the designed MySQL multitrigger active model by comparing to the PostgreSQL active model. In this experiment, we created the same RDBS in both MySQL and PostgreSQL platforms. The RDBS had 15 tables including TABLE_A which was designed and set up with 500 datasets for testing all the triggers. TABLE_A data structure is shown in table 1.

Triggers were created to detect the event occurs to p_id and to test the condition depend on the amount’s value. We ran 5 experiments for each platform by setting 5, 10, 15, 20 and 25 triggers at a time in order to test the multitrigger performance. All triggers were differently defined under 3 criteria: detecting event (every record of p_id, the designated value of p_id), detecting time (before/after the table modification) and amount testing condition (greater than, less than, in the range of). To test the designed triggers, we had developed the command set composed of 200 condition-matched transactions for each trigger and 200 condition-mismatched transactions. Every command set was run 100 times in each experiment. The multitrigger performance was evaluated by the average execution time per transaction.
### TABLE 1 TABLE_A DATA STRUCTURE

<table>
<thead>
<tr>
<th>field</th>
<th>Data type</th>
<th>key</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_id</td>
<td>varchar (50)</td>
<td>Primary key</td>
<td>not null</td>
</tr>
<tr>
<td>name_th</td>
<td>varchar (100)</td>
<td></td>
<td>not null</td>
</tr>
<tr>
<td>name_en</td>
<td>varchar (50)</td>
<td></td>
<td>not null</td>
</tr>
<tr>
<td>detail</td>
<td>varchar (50)</td>
<td></td>
<td>not null</td>
</tr>
<tr>
<td>formular</td>
<td>varchar (50)</td>
<td></td>
<td>not null</td>
</tr>
<tr>
<td>exp</td>
<td>date</td>
<td></td>
<td>not null</td>
</tr>
<tr>
<td>price</td>
<td>varchar (50)</td>
<td></td>
<td>not null</td>
</tr>
<tr>
<td>amount</td>
<td>int (10)</td>
<td></td>
<td>not null</td>
</tr>
<tr>
<td>unit</td>
<td>varchar (20)</td>
<td></td>
<td>not null</td>
</tr>
<tr>
<td>modified_date</td>
<td>date</td>
<td></td>
<td>not null</td>
</tr>
</tbody>
</table>

### RESULTS AND DISCUSSIONS

The result comparison graphs of the multitigger system performance between MySQL and PostgreSQL platforms are shown in figure 5, in which figure 5 (a) to (d) presents the outcomes of simultaneously running 10, 15, 20 and 25 triggers respectively. It is obvious that the proposed MySQL multitigger model shows the much better performance concerning the average execution time comparing to PostgreSQL’s traditional model. This is because the proposed model contains only two system triggers running during the operation time. The multitigger manners are manipulated and controlled by the algorithm programmed within the system triggers. When the designated event is detected, the system will automatically call the predefined active procedure and then finish all the detection tasks.

On the other hand, The PostgreSQL multitigger ARDBS must evaluate all the triggers running in the system. Even one trigger is detected and the action is performed, all the other triggers left are still evaluated. Table 2 shows the average execution time in both platforms, in which the numbers of triggers running in the system seems to affect the execution time in the increasing manner. In addition, figure 6 also shows that the average execution time of the PostgreSQL mutitigger ARDBS, but MySQL, dramatically increases by the numbers of triggers.
Figure 5 Graph Comparing Multitrigger System Performances between MySQL and PostgreSQL (a) 10 Triggers (b) 15 Triggers (c) 20 Triggers (d) 25 Triggers

TABLE 2 THE MULTITRIGGER ACTIVE MODEL PERFORMANCE

<table>
<thead>
<tr>
<th>Triggers</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL Platform</td>
<td>0.249773628</td>
<td>0.278416790</td>
<td>0.317448932</td>
<td>0.330446678</td>
<td>0.364962651</td>
</tr>
<tr>
<td>PostgreSQL Platform</td>
<td>0.904373929</td>
<td>1.039946864</td>
<td>1.138586064</td>
<td>1.251077038</td>
<td>1.400388789</td>
</tr>
</tbody>
</table>

Figure 6 Overall Performance Comparisons between MySQL and PostgreSQL Multitrigger Models

SUMMARY

The multiplatform add-on agent and the multitrigger model are presented in this paper with the purpose to use with any passive relational database system in order to transform the system to be active without any source code changes. The proposed models were designed, implemented and system tested with two database platforms which are MySQL and PostgreSQL. The experimental results have shown that the designed system can properly and correctly work on both systems in multitriggers manners. It is obvious...
to see that MySQL multitrigger model has shown the better performance in average execution time especially when many triggers are running in the system. The proposed multiplatform add-on agent should be modified to work with other RDBS and the designed multitrigger model can be applied to use in any RDBS which has no multitrigger functionality.

REFERENCES


Norman W. Paton and Oscar Diaz. Active Database System. *ACM Computing Surveys, 31*(1); March 1999


