School of Information and Computer Technology
Sirindhorn International Institute of Technology
Thammasat University
ITS351 Database Programming Laboratory

Laboratory #13: Trigger

Objective:
- To learn build in trigger in MySQL
- To learn how to write trigger to MySQL
- To learn how call trigger in MySQL to PHP

MySQL Resource: http://www.mysql.com/

1 Create Trigger in MYSQL

A SQL trigger is a set of SQL statements stored in the database catalog. A SQL trigger is executed or fired whenever an event associated with a table occurs e.g., insert, update or delete.

A SQL trigger is a special type of stored procedure. It is special because it is not called directly like a stored procedure. The main difference between a trigger and a stored procedure is that a trigger is called automatically when a data modification event is made against a table whereas a stored procedure must be called explicitly.

It is important to understand SQL trigger's advantages and disadvantages so that you can use it appropriately. In the following sections, we will discuss about the advantages and disadvantages of using SQL triggers.

Advantages of using SQL triggers

- SQL triggers provide an alternative way to check the integrity of data.
- SQL triggers can catch errors in business logic in the database layer.
- SQL triggers provide an alternative way to run scheduled tasks. By using SQL triggers, you don't have to wait to run the scheduled tasks because the triggers are invoked automatically *before* or *after* a change is made to the data in the tables.
- SQL triggers are very useful to audit the changes of data in tables.

Disadvantages of using SQL triggers

- SQL triggers only can provide an extended validation and they cannot replace all the validations. Some simple validations have to be done in the application layer. For example, you can validate user's inputs in the client side by using JavaScript or in the server side using server side scripting languages such as JSP, PHP, ASP.NET, Perl, etc.
- SQL triggers are invoked and executed invisibly from client-applications therefore it is difficult to figure out what happen in the database layer.
- SQL triggers may increase the overhead of the database server.

In MySQL, a trigger is a set of SQL statements that is invoked automatically when a change is made to the data on the associated table. A trigger can be defined to be invoked either before or after the data is changed by INSERT, UPDATE or DELETE statement.

- BEFORE INSERT – activated before data is inserted into the table.
- AFTER INSERT – activated after data is inserted into the table.
- BEFORE UPDATE – activated before data in the table is updated.
- AFTER UPDATE – activated after data in the table is updated.
- BEFORE DELETE – activated before data is removed from the table.
- AFTER DELETE – activated after data is removed from the table.

MySQL trigger limitations

MySQL triggers cover all features defined in the standard SQL. However, there are some limitations that you should know before using them in your applications.

MySQL triggers cannot:

- Use SHOW , LOAD DATA , LOAD TABLE , BACKUP DATABASE, RESTORE , FLUSH and RETURN statements.
- Use statements that commit or rollback implicitly or explicitly such as COMMIT , ROLLBACK , START TRANSACTION , LOCK/UNLOCK TABLES , ALTER , CREATE , DROP , RENAME , etc.
- Use prepared statements such as PREPARE , EXECUTE , etc.
- Use dynamic SQL statements.

CREATE TRIGGER Syntax

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    [trigger_order]
    trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

This statement creates a new trigger. A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. The trigger becomes associated with the table named *tbl_name*, which must refer to a permanent table. You cannot associate a trigger with a TEMPORARY table or a view.

MySQL takes the DEFINER user into account when checking trigger privileges as follows:

- At CREATE TRIGGER time, the user who issues the statement must have the TRIGGER privilege.
- At trigger activation time, privileges are checked against the DEFINER user. This user must have these privileges:
    - The TRIGGER privilege for the subject table.
    - The SELECT privilege for the subject table if references to table columns occur using OLD.*col_name* or NEW.*col_name* in the trigger body.
    - The UPDATE privilege for the subject table if table columns are targets of SET NEW.*col_name* = *value* assignments in the trigger body.
    - Whatever other privileges normally are required for the statements executed by the trigger.

Here is a simple example that associates a trigger with a table, to activate for INSERT operations. The trigger acts as an accumulator, summing the values inserted into one of the columns of the table.

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
    -> FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.06 sec)
```

The CREATE TRIGGER statement creates a trigger named ins_sum that is associated with the account table. It also includes clauses that specify the trigger action time, the triggering event, and what to do when the trigger activates:

- The keyword BEFORE indicates the trigger action time. In this case, the trigger activates before each row inserted into the table. The other permitted keyword here is AFTER.
- The keyword INSERT indicates the trigger event; that is, the type of operation that activates the trigger. In the example, INSERT operations cause trigger activation. You can also create triggers for DELETE and UPDATE operations.
- The statement following FOR EACH ROW defines the trigger body; that is, the statement to execute each time the trigger activates, which occurs once for each row affected by the triggering event. In the example, the trigger body is a simple SET that accumulates into a user variable the values inserted into the amount column. The statement refers to the column as NEW.amount which means "the value of the amount column to be inserted into the new row."

To use the trigger, set the accumulator variable to zero, execute an INSERT statement, and then see what value the variable has afterward:

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+-----------------------+
| Total amount inserted |
+-----------------------+
| 1852.48               |
+-----------------------+
```

In this case, the value of @sum after the INSERT statement has executed is 14.98 + 1937.50 - 100, or 1852.48.

To destroy the trigger, use a DROP TRIGGER statement. You must specify the schema name if the trigger is not in the default schema:

```
mysql> DROP TRIGGER test.ins_sum;
```

If you drop a table, any triggers for the table are also dropped.

Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema. Triggers in different schemas can have the same name.

In addition to the requirement that trigger names be unique for a schema, there are other limitations on the types of triggers you can create. In particular, there cannot be multiple triggers for a given table that have the same trigger event and action time. For example, you cannot have two BEFORE UPDATE triggers for a table. To work around this, you can define a trigger that executes multiple statements by using the BEGIN ... END compound statement construct after FOR EACH ROW. (An example appears later in this section.)

Within the trigger body, the OLD and NEW keywords enable you to access columns in the rows affected by a trigger. OLD and NEW are MySQL extensions to triggers; they are not case sensitive.

In an INSERT trigger, only NEW.*col_name* can be used; there is no old row. In a DELETE trigger, only OLD.*col_name* can be used; there is no new row. In an UPDATE trigger, you can use OLD.*col_name* to refer to the columns of a row before it is updated and NEW.*col_name* to refer to the columns of the row after it is updated.

A column named with OLD is read only. You can refer to it (if you have the SELECT privilege), but not modify it. You can refer to a column named with NEW if you have the SELECT privilege for it. In a BEFORE trigger, you can also change its value with SET NEW.*col_name* = *value* if you have the UPDATE privilege for it. This means you can use a trigger to modify the values to be inserted into a new row or used to update a row. (Such a SET statement has no effect in an AFTER trigger because the row change will have already occurred.)

In a BEFORE trigger, the NEW value for an AUTO_INCREMENT column is 0, not the sequence number that is generated automatically when the new row actually is inserted.

By using the BEGIN ... END construct, you can define a trigger that executes multiple statements. Within the BEGIN block, you also can use other syntax that is permitted within stored routines such as conditionals and loops. However, just as for stored routines, if you use the **mysql** program to define a trigger that executes multiple statements, it is necessary to redefine the **mysql** statement delimiter so that you can use the ; statement delimiter within the trigger definition. The following example illustrates these points. It defines an UPDATE trigger that checks the new value to be used for updating each row, and modifies the value to be within the range from 0 to 100. This must be a BEFORE trigger because the value must be checked before it is used to update the row:

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.amount < 0 THEN
    ->         SET NEW.amount = 0;
    ->     ELSEIF NEW.amount > 100 THEN
    ->         SET NEW.amount = 100;
    ->     END IF;
    -> END;//
mysql> delimiter ;
```

It can be easier to define a stored procedure separately and then invoke it from the trigger using a simple CALL statement. This is also advantageous if you want to execute the same code from within several triggers.

There are limitations on what can appear in statements that a trigger executes when activated:

- The trigger cannot use the CALL statement to invoke stored procedures that return data to the client or that use dynamic SQL. (Stored procedures are permitted to return data to the trigger through OUT or INOUT parameters.)
- The trigger cannot use statements that explicitly or implicitly begin or end a transaction, such as START TRANSACTION, COMMIT, or ROLLBACK. (ROLLBACK to SAVEPOINT is permitted because it does not end a transaction.).
- Prior to MySQL 5.0.10, triggers cannot contain direct references to tables by name.

Worksheet

1. Import database named "BANK" from given resource file in database folder. The following figure shows the structure of BANK database. Note that all fields must set to allow NULL value except primary key.

account Table

Field	Type	Length Values	Extra	Primary Key
id	INT		Auto_increment	Yes
no	VARCHAR	20		
name	VARCHAR	200		
creditLimit	DOUBLE			
bal	Float			

transaction Table

Field	Type	Length Values	Extra	Primary Key
id	INT		Auto_increment	Yes
type	char	1		
amount	float			
date	datetime			
accid	INT			

2. Write Trigger to prevent transaction edit.
3. Write Trigger to check balance of money before insert. If balance of money is not enough to Withdraw, please alert message "Not enough money", however, if balance of money is enough, insert the record and update new balance to "bal" column.

Exercise

Extend worksheet and write trigger code to store delete transaction to log table. The following table shows structure of "**transaction_history**"

transaction_history Table

Field	Type	Length Values	Extra	Primary Key
id	INT		Auto_increment	Yes
type	char	1		
amount	float			
date	datetime			
accid	INT			
deldate	datetime			

* Deldate column will store current date/time when user deletes transaction.