Database Systems (ISYS1001/ISYS5008)

Lecture 8

Introduction to database programming : Triggers and Views in MySQL

Updated: 29nd September,2021

Discipline of Computing
School of Electrical Engineering, Computing and Mathematical Sciences (EECMS)

CRICOS Provide Code: 00301J

Copyright Warning

COMMONWEALTH OF AUSTRALIA
Copyright Regulation 1969

WARNING

This material has been copied and communicated to you by or on behalf of **Curtin University of Technology** pursuant to Part VB of the *Copyright Act 1968* (**the Act**)

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice

Learning outcomes

- Explain why triggers are important.
- Implement simple triggers BEFORE or AFTER executing a query
- Implement simple triggers to invoke at INSERT, UPDATE and DELETE events
- Use NOW and OLD keyword appropriately in a trigger.
- Use simple error handling methods with triggers.
- Explain the use of Views, their advantages and limitations
- Write views and retrieve data (SELECT) from the view.

Triggers

Triggers: Motivation

- When inserting, updating and deleting data in a database, it may be required to check the values involved to make sure they do not lead to issues in the values of the database..
- *Attribute* and *tuple-based* checks can be done at known times.
  - E.g.,

```
CREATE TABLE Enrolled (
    unit INT,
    student INT,
    mark REAL CHECK (mark <=100));
```

- Such checks are not powerful and not available in all variations of SQL.
- Triggers let the user decide when to check for a condition and to specify whatever action is desired.
- It is a named database object that is associated with a table, and ***that activates only when a particular event occurs for the table***.

Triggers: Motivation

- Triggers can be used in many situations: E.g.,
  - to perform checks of values to be inserted into a table
  - to perform calculations on values involved in an update.
- A trigger is defined to activate when a statement inserts, updates, or deletes rows in the associated table. These row operations are trigger events.
  - E.g., Rows can be inserted by INSERT statement, and an insert trigger activates for each inserted row.
- A trigger can be set to activate either before or after the trigger event.
  - For example, you can have a trigger activate before each row that is inserted into a table or after each row that is updated.

Event-Condition-Action Rules

- Triggers are also known as Event-Condition-Action rules or ECA rules.
  - Event : typically a type of database modification, e.g., “insert on `Units`.”
  - Condition : Any SQL Boolean-valued expression.
  - Action : Any SQL statements.

  (action can be some activity not related to the table which the event occurred)

Creating a Trigger

- Trigger can be created using CREATE TRIGGER statement:

```
CREATE 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 | PROCEDES} <other_trigger_name>
```

Trigger is associated with a table, and that activates when a particular event occurs for the table.

Trigger statement: FOR EACH ROW

- Triggers are either “row-level” or “statement-level.”
- FOR EACH ROW indicates row-level; its absence indicates statement-level.
- *Row level triggers* : execute once for each modified tuple.
- *Statement-level triggers* : execute once for an SQL statement, regardless of how many tuples are modified.
- MySQL also does not support statement-level triggers.
- You still have to list the FOR EACH ROW though.

Clauses in trigger statements : <trigger_time>

- <trigger_time> is the trigger action time.
- It can be BEFORE or AFTER to indicate that the trigger activates before or after each row to be modified.
- Basic column value checks occur prior to trigger activation, so you cannot use BEFORE triggers to convert values inappropriate for the column type to valid values.

Clauses in trigger statements : <trigger_event>

- <trigger_event> indicates the kind of operation that activates the trigger.
- following <trigger_event> values are permitted:
  - INSERT: The trigger activates whenever a new row is inserted into the table
  - UPDATE: The trigger activates whenever a row is modified
  - DELETE: The trigger activates whenever a row is deleted from the table
- E.g.,

```
CREATE TRIGGER before_emp_insert
   BEFORE INSERT ON Emp
...
```

Options in trigger statements : [trigger_order]

- It is possible to define multiple triggers for a given table that have the same trigger event and action time.
  - E.g., we can have two BEFORE UPDATE triggers for a table.
- By default, triggers that have the same trigger event and action time, activate in the order they were created.
- [trigger_order] clause can be used to change the default order.

Clauses in trigger statements : <trigger_body>

- <trigger_body> is the statement to execute when the trigger activates.
- To execute multiple statements, BEGIN ... END compound statement construct can ne used.

Trigger Example 1

- Assume we want to *calculate the total of all salaries entered when entering a salary value to emp table.*

```
Emp(emp_num INT, salary DECIMAL(8,2))
```

- Trigger definition:

```
CREATE TRIGGER before_emp_insert_total
   BEFORE INSERT ON Emp
   FOR EACH ROW
       SET @total = @total + NEW.salary;
```

*We want the trigger to be activated before inserting values to **emp** table*

- Trigger use:

```
>SET @amount= 0;
>INSERT INTO emp(1,2400);
>INSERT INTO emp(1,1000);
>SELECT @total;
```

@total
3400.00

OLD and NEW keywords in triggers

- Within the trigger body of MySQL, the **OLD** and **NEW** keywords enable us to access columns in the rows affected by a trigger.
- 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, we 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.

Trigger Example 2

- Assume we have following two tables:

```
Emp(emp_no, name, dept_no, salary, supervisor)
Dept(dept_no, dept_name, total_sal, mgr)
```

- *Requirement : **After** an employee is **added** to the database, we want to modify the total salary column of the dept of that employee.*

Trigger Example 2


```
DELIMITER //
CREATE TRIGGER after_emp_insert_total2
   AFTER INSERT ON Emp
   FOR EACH ROW
    BEGIN
      IF NEW.salary IS NOT NULL THEN
         UPDATE Dept
           SET total_sal = total_sal + NEW.salary
           WHERE dept_no = NEW.dept_no;
      END IF;
    END//
DELIMITER ;
```

NEW.salary refer to the salary c
of a row of Emp table, after the
updated.

*NEW.dept_no refer to the dept_*
column of a row of Emp table, a
row is updated ;

Trigger Example 3

- *Requirement: When the salary of an employee is **changed**, modify the total salary of their dept.*

```
DELIMITER //
CREATE TRIGGER after_emp_update_total2
   AFTER UPDATE ON Emp
   FOR EACH ROW
   BEGIN
    IF OLD.salary IS NOT NULL
        AND NEW.salary IS NOT NULL THEN
      UPDATE Dept
        SET total_sal = total_sal +
                NEW.salary - OLD.salary
        WHERE dno = NEW.dno;
    END IF;
   END //
```

Trigger Example 4

- *Requirement: When an employee is removed, modify the total salary of the dept.*

```
DELIMITER //
CREATE TRIGGER after_emp_delete_total2
   AFTER DELETE ON Emp
   FOR EACH ROW
      IF OLD.dno IS NOT NULL
           AND OLD.salary IS NOT NULL THEN
        UPDATE Dept
          SET total_sal = total_sal - OLD.salary
          WHERE dno = OLD.dno;
      END IF;
    END //
DELIMITER ;
```

“Mutating” triggers

- If event and action specified on the same table , issues may occur (“ mutating trigger”)
- Mutating trigger error occur at run time

Example 5:

```
DELIMITER //
CREATE TRIGGER mutator
   AFTER UPDATE ON Emp
   FOR EACH ROW
      IF NEW.salary < OLD.salary THEN
        UPDATE Emp
          SET salary = OLD.salary
          WHERE eno = OLD.eno;
      END IF;
    END //
```

We cannot modify the same table the trigger is activated on.

Trigger activated after update and then again, we are trying to do another update, as the result of the trigger event.

Error handling in MySQL

- Command to show error message to know details of compilation errors:

```
SHOW ERRORS;
```

- This is not useful when error occurs due to non-compilation errors (e.g. entering a wrong value to a table)
- If a BEFORE trigger is used to avoid inserting a wrong value to a table (e.g. negative marks) , then the inserting statements will be not invoke and the query will be terminated.
- In such a case, we can provide a error message through SIGNAL SQLSTATE to indicate the error in more human readable form.

Error handling and triggers

- SIGNAL is the way to “return” an error in MySQL.
- SIGNAL provides error information to a handler and provides control over the error's characteristics (error number, SQLSTATE value, message).

```
SIGNAL condition_value
    [SET signal_information_item
    [, signal_information_item] ...]

condition_value: {
    SQLSTATE [VALUE] sqlstate_value| condition_name
  }
```

Trigger Example 6


```
DELIMITER //
CREATE TRIGGER before_book_update
BEFORE UPDATE
ON book FOR EACH ROW
BEGIN
    DECLARE err_msg_lower_price VARCHAR(100);
    SET errorMessage = CONCAT('The new price', NEW.price,
                        'cannot be lower than', OLD.price);
    IF NEW.price > OLD.price THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = err_msg_lower_price;
    END IF;
END//
DELIMITER ;
```

Getting information about triggers

- To view a list of all defined triggers in MySQL use the following :

```
SHOW TRIGGERS [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr];
```

- We can search for triggers associated with a particular table using the LIKE keyword and a string matching pattern. We can narrow down the search further by using a WHERE clause.

Drop, enable or disable triggers

- To drop a trigger in MySQL:

```
DROP TRIGGER <trigger_name> [IF EXISTS];
```

- To disable or enable a trigger in SQL:
  - `ALTER TRIGGER <trigger_name {disable|enable};`
  - My SQL doesn't allow this functionality.

Views

Views

- A view is a table which is derived from other stored tables which are called base tables or defining tables.
- View is simply a stored query in the database.
- A view does not *necessarily* exist in a physical form.
- In general there are two kinds of views in SQL:
  - Virtual = not stored in the database; just a query for constructing the relation.
  - Materialized = actually constructed and stored.
- Default is virtual in SQL.
- MySQL doesn't support materialized views.
- Therefore, in MySQL , a view is a virtual table which is derived from other tables.
- Views are extremely useful and very common construct in Database systems.

Views

- CREATE VIEW statement is used to define views in MySQL as follows:

```
CREATE [OR REPLACE] VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION];
```

- Dropping a view can be done same way as dropping a table:

```
DROP VIEW [IF EXISTS]
   view_name;
```

Views

- SELECT statement in a view can refer to base tables or other views. It can use joins, UNION and subqueries. It can refer to in-build functions (without referring a table) also.
- However, there are restrictions relate to local variables, user-defined variables etc.
- Many other optional parameters are available .
- A trigger cannot be associated with a view.

View definition : Example 1

- Consider a join query we have written in the lecture 5.

```
SELECT studentID, firstname, Students.courseID
FROM Students INNER JOIN Courses ON
    country = 'Australia' AND Students.courseID=Courses.courseID;
```

- We can create a View *LocalStudents* to "contain" the required information from *Students* and *Courses* tables:

```
CREATE VIEW LocalStudents AS
    SELECT studentID, firstname, Students.courseID AS course
    FROM Students INNER JOIN Courses ON
    country = 'Australia' AND Students.courseID=Courses.courseID;
```

View definition: Example 2

- If different names for the columns are required, they can be specified in the optional *column_list.*

```
CREATE VIEW LocalStudents(sid,sname,scourse) AS
    SELECT studentID, firstname, Students.courseID AS course
    FROM Students INNER JOIN Courses ON
    country = 'Australia' AND Students.courseID=Courses.courseID;
```

Using a view :

- We can query a view as it were a base table (referring to Example 1)

```
SELECT studentID, firstname, course
    FROM LocalStudents;
```

```
SELECT studentID, firstname, course
    FROM LocalStudents WHERE course= 'C20';
```

Example 3

- Consider some tables in book store database

```
Author(a_name)
Write(author, book_id, date)
Book(isbn, title)
```

- We can create a view Stock(title, author, isbn) to "contain" information from Authors and Books to record books that are currently in stock:

```
CREATE VIEW Stock AS
        SELECT title, a_name AS author, isbn
        FROM (Books JOIN Writes ON isbn = book_id)
          JOIN Authors ON Author.a_name = Writes.author;
```

- Using the view:

```
SELECT title
        FROM Stock
        WHERE isbn = 9780312094119;
```

Using a view : INSERT, UPDATE

- There is limited ability to insert, update delete base table values via view.
- There must be a one-to-one relationship between the rows in the view and the rows in the underlying table to update a table.
- View should contain all columns in the base table that do not have a default value.
- Joins and subqueries can be problematic.
- Views with aggregate functions , ORDER BY, GROUP BY are also not possible to be updated.

Use of views

- Hide the complexity of queries (**Query Simplicity):**

  If complex queries are used frequently, a view can be created based on the query an then use it (SELECT statement from the view) instead of writing the query again. A view can draw data from several different tables and present it as a single table, turning multi-table queries into single-table queries against the view.

- Limit the access to sensitive tables (**security**) :

  As each user can be given permission to access the database only through a small set of views that contain the specific data the user is authorized to see, thus restricting the user's access to stored data.

- Make consistent business rules in the database:

If a formula is used to calculate some value in many places, it can be created as a view once and use it in many queries.

Limitations of views

- **Performance :**
  - Views create the appearance of a table, but the DBMS must still translate queries against the view into queries against the underlying source tables.
  - If the view is defined by a complex, multi-table query then simple queries on the views may take considerable time
- **Update restrictions**

Summary

- Triggers respond to changes in the database, but otherwise work much like procedures.
- Triggers are activated only when a certain event occur.
- Triggers can occur BEFORE or AFTER some action on a table ; actions can be INSERT, UPDATE or DELETE operation.
- We have to avoid mutating trigger errors when designing triggers
- View is a table which is derived from other stored tables which are called base tables or defining tables.
- Views are extremely common object in databases, which helps to simplify complex queries, and to improve access control.

References

- Triggers MySQL reference:

https://dev.mysql.com/doc/refman/8.0/en/triggers.html

- Views MySQL reference:

https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html

Happy Database systems

Next week : Indexes, Transactions

Practical worksheet 9

There will be no assessment in the next week