MySQL Trigger

Database

Trigger

- 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.

Trigger Types

- 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.

- When we use a statement that does not use INSERT, DELETE or UPDATE statement to change data in a table, the triggers associated with the table are not invoked.
  - For example, the TRUNCATE statement removes all data of a table but does not invoke the trigger associated with that table.
  - Syntax: mysql:>truncate table table_name;
  - If there is any FOREIGN KEY constraints from other tables which reference the table that you truncate, the TRUNCATE TABLE statement will fail.

- There are some statements that use the INSERT statement behind the scenes such as REPLACE statement or LOAD DATA statement.
- If you use these statements, the corresponding triggers associated with the table are invoked.
- You must use a unique name for each trigger associated with a table.

Replace statement

- Replace statement works as below-
- Step 1. Insert a new row into the table, if a duplicate key error occurs.
- Step 2. If the insertion fails due to a duplicate-key error occurs:
  - Delete the conflicting row that causes the duplicate key error from the table.
  - Insert the new row into the table again.

Replace to insert

- Using MySQL REPLACE to insert a new row
- Syntax
  - mysql:> REPLACE [INTO] table_name(column_list)
    VALUES(value_list);

Let's create a table

```
CREATE TABLE cities (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50),
    population INT NOT NULL
);
```

Let's insert some records

```
INSERT INTO cities(name,population)
VALUES('New York',8008278),
    ('Los Angeles',3694825),
    ('San Diego',1223405);
```

Replace to insert

use the REPLACE statement to update the population of the Los Angeles city to 3696820.

id	name	population
1	New York	8008278
2	Los Angeles	3694825
3	San Diego	1223405

```
REPLACE INTO cities(id,population)
VALUES(2,3696820);
```

id	name	population
1	New York	8008278
2	null	3696820
3	San Diego	1223405

Replace to update

- Using MySQL REPLACE statement to update a row
  - Syntax
  - mysql:> REPLACE INTO table

    SET column1 = value1,

    column2 = value2;

No where clause

Replace to update

```
REPLACE INTO cities
SET id = 4,
    name = 'Phoenix',
    population = 1768980;
```

Can be used to update the population of the Phoenix city to 1768980

MySQL Trigger Synatx

Syntax :

```
CREATE TRIGGER trigger_name
trigger_time trigger_event
ON table_name
FOR EACH ROW
BEGIN
.........
END;
```

Trigger name convention: after the CREATE TRIGGER statement, the trigger name should follow the naming convention [trigger time]_[table name]_[trigger event], for example before_employees_update.

Reference: www.mysqltutorial.org

Activation time and trigger event

- One must specify the activation time when a trigger is defined.
  - You use the BEFORE keyword if you want to process action prior to the change is made on the table and AFTER if you need to process action after the change is made.
- The trigger event can be INSERT, UPDATE or DELETE.
  - This event causes the trigger to be invoked. A trigger only can be invoked by one event. To define a trigger that is invoked by multiple events, you have to define multiple triggers, one for each event.

Trigger

- A trigger must be associated with a specific table. Without a table trigger would not exist therefore you have to specify the table name after the ON keyword.
- You place the SQL statements between BEGIN and END block. This is where you define the logic for the trigger.
- To view all triggers in the current database, you use SHOW TRIGGERS statement as follows:
  - mysql:>SHOW TRIGGERS;

Trigger example

Let's create a table

```
DROP TABLE IF EXISTS WorkCenters;

CREATE TABLE WorkCenters (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    capacity INT NOT NULL
);
```

Let's create another table

```
DROP TABLE IF EXISTS WorkCenterStats;

CREATE TABLE WorkCenterStats(
    totalCapacity INT NOT NULL
);
```

Creating before insert trigger

```
DELIMITER $$

CREATE TRIGGER before_workcenters_insert
BEFORE INSERT
ON WorkCenters FOR EACH ROW
BEGIN
   DECLARE rowcount INT;

   SELECT COUNT(*)
   INTO rowcount
   FROM WorkCenterStats;

   IF rowcount > 0 THEN
      UPDATE WorkCenterStats
      SET totalCapacity = totalCapacity + new.capacity;
   ELSE
      INSERT INTO WorkCenterStats(totalCapacity)
      VALUES(new.capacity);
   END IF;

END $$

DELIMITER ;
```

Another example

Let's create members table

```
DROP TABLE IF EXISTS members;

CREATE TABLE members (
   id INT AUTO_INCREMENT,
   name VARCHAR(100) NOT NULL,
   email VARCHAR(255),
   birthDate DATE,
   PRIMARY KEY (id)
);
```

create another table called reminders that stores reminder messages to members

```
DROP TABLE IF EXISTS reminders;

CREATE TABLE reminders (
   id INT AUTO_INCREMENT,
   memberId INT,
   message VARCHAR(255) NOT NULL,
   PRIMARY KEY (id , memberId)
);
```

After insert trigger

```
DELIMITER $$

CREATE TRIGGER after_members_insert
AFTER INSERT
ON members FOR EACH ROW
BEGIN
   IF NEW.birthDate IS NULL THEN
      INSERT INTO reminders(memberId, message)
      VALUES(NEW.id, CONCAT('Hi ', NEW.name, ', please upd
ate your date of birth.'));
   END IF;
END$$

DELIMITER ;
```

New: To access the new value provided in the insert statement

Next Example

```
DROP TABLE IF EXISTS sales;

CREATE TABLE sales (
    id INT AUTO_INCREMENT,
    product VARCHAR(100) NOT NULL,
    quantity INT NOT NULL DEFAULT 0,
    fiscalYear SMALLINT NOT NULL,
    fiscalMonth TINYINT NOT NULL,
    CHECK(fiscalMonth >= 1 AND fiscalMonth <= 12),
    CHECK(fiscalYear BETWEEN 2000 and 2050),
    CHECK (quantity >=0),
    UNIQUE(product, fiscalYear, fiscalMonth),
    PRIMARY KEY(id)
);
```

Before update trigger

```
DELIMITER $$

CREATE TRIGGER before_sales_update
BEFORE UPDATE
ON sales FOR EACH ROW
BEGIN
   DECLARE errorMessage VARCHAR(255);
   SET errorMessage = CONCAT('The new quantity ',
                  NEW.quantity,
                  ' cannot be 3 times greater than the current quantity ',
                  OLD.quantity);

   IF new.quantity > old.quantity * 3 THEN
      SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT = errorMessage;
   END IF;
END $$

DELIMITER ;
```

Old and new values can be accessed using OLD and NEW modifiers

Signal statement

- The SIGNAL statement can be used to return an error or warning condition to the caller from a stored program e.g., stored procedure, stored function, trigger or event.
- The SIGNAL statement provides you with control over which information for returning such as value and message SQLSTATE.
  - Example: SIGNAL SQLSTATE 45000
  - It is used to raise an error along with an error message
  - Notice that 45000 is a generic SQLSTATE value that illustrates an unhandled user-defined exception

Types of trigger

- Row level
  - A row-level trigger is activated for each row that is inserted, updated, or deleted. For example, if a table has 100 rows inserted, updated, or deleted, the trigger is automatically invoked 100 times for the 100 rows affected.
- Statement level
  - A statement-level trigger is executed once for each transaction regardless of how many rows are inserted, updated, or deleted.

MySQL supports row level trigger only

Advantages of Trigger

- Triggers provide another way to check the integrity of data.
- Triggers handle errors from the database layer.
- Triggers give an alternative way to run scheduled tasks. By using triggers, you don't have to wait for the scheduled events to run because the triggers are invoked automatically *before* or *after* a change is made to the data in a table.
- Triggers can be useful for auditing the data changes in tables.

Disadvantages

- Triggers can only provide extended validations, not all validations.
- For simple validations, you can use the NOT NULL, UNIQUE, CHECK and FOREIGN KEY constraints.
- Triggers can be difficult to troubleshoot because they execute automatically in the database, which may not invisible to the client applications.
- Triggers may increase the overhead of the MySQL Server.