Triggers

Database
Management
Systems

Prof. Pramod Bide

Introduction

- A trigger is a set of SQL statements that reside in system memory with unique names.
- It is a specialized category of stored procedure that is called automatically when a database server event occurs. Each trigger is always associated with a table.
- A trigger is called a special procedure because it cannot be called directly like a stored procedure.
- The key distinction between the trigger and procedure is that a trigger is called automatically when a data modification event occurs against a table. A stored procedure, on the other hand, must be invoked directly.

Difference between triggers and stored procedures

- The following are the main characteristics that distinguish triggers from stored procedures:
  a. We cannot manually execute/invoked triggers.
  b. Triggers have no chance of receiving parameters.
  c. A transaction cannot be committed or rolled back inside a trigger.

Types of triggers

We can define the maximum six types of actions or events in the form of triggers:

Before Insert:

It is activated before the insertion of data into the table.

After Insert:

It is activated after the insertion of data into the table.

Before Update:

It is activated before the update of data in the table.

After Update:

It is activated after the update of the data in the table.

Before Delete:

It is activated before the data is removed from the table.

After Delete:

It is activated after the deletion of data from the table.

MySQL Trigger Syntax

```
CREATE TRIGGER trigger_name
(AFTER | BEFORE) (INSERT | UPDATE | DELETE)
ON table_name FOR EACH ROW
BEGIN
--variable declarations
--trigger code
END;
```

Example- Before insert

1) Create employee table

MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM employee;
+---------+------------+--------------+---------------+
| name    | occupation | working_date | working_hours |
+---------+------------+--------------+---------------+
| Robin   | Scientist  | 2020-10-04   | 12            |
| Warner  | Engineer   | 2020-10-04   | 10            |
| Peter   | Actor      | 2020-10-04   | 13            |
| Marco   | Doctor     | 2020-10-04   | 14            |
| Brayden | Teacher    | 2020-10-04   | 12            |
| Antonio | Business   | 2020-10-04   | 11            |
+---------+------------+--------------+---------------+
6 rows in set (0.00 sec)
```

2) Next, we will create a BEFORE INSERT trigger. This trigger is invoked automatically insert the working_hours = 0 if someone tries to insert working_hours < 0

```
mysql> DELIMITER //
mysql> Create Trigger before_insert_empworkinghours
BEFORE INSERT ON employee FOR EACH ROW
BEGIN
IF NEW.working_hours < 0 THEN SET NEW.working_hours = 0;
END IF;
END //
```

Example- Before insert

3) Now, we can use the following statements to invoke this trigger:

mysql> **INSERT INTO** employee **VALUES** ('Markus', 'Former', '2020-10-08', 14);

mysql> **INSERT INTO** employee **VALUES** ('Alexander', 'Actor', '2020-10-012', -13);

Output:

MySQL 8.0 Command Line Client

```
mysql> INSERT INTO employee VALUES
    -> ('Markus', 'Former', '2020-10-08', 14);
Query OK, 1 row affected (0.18 sec)

mysql> INSERT INTO employee VALUES
    -> ('Alexander', 'Actor', '2020-10-012', -13);
Query OK, 1 row affected (0.16 sec)

mysql> SELECT * FROM employee;
+-----------+------------+--------------+---------------+
| name      | occupation | working_date | working_hours |
+-----------+------------+--------------+---------------+
| Robin     | Scientist  | 2020-10-04   | 12            |
| Warner    | Engineer   | 2020-10-04   | 10            |
| Peter     | Actor      | 2020-10-04   | 13            |
| Marco     | Doctor     | 2020-10-04   | 14            |
| Brayden   | Teacher    | 2020-10-04   | 12            |
| Antonio   | Business   | 2020-10-04   | 11            |
| Markus    | Former     | 2020-10-08   | 14            |
| Alexander | Actor      | 2020-10-12   | 0             |
+-----------+------------+--------------+---------------+
8 rows in set (0.00 sec)
```

Example- After insert

1) Create student_info table

MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM student_info;
+---------+----------+----------+---------+-------+-------------+
| stud_id | stud_code | stud_name | subject | marks | phone       |
+---------+----------+----------+---------+-------+-------------+
|       1 | 101      | Mark     | English |    68 | 34545693537 |
|       2 | 102      | Joseph   | Physics |    70 | 98765435659 |
|       3 | 103      | John     | Maths   |    70 | 97653269756 |
|       4 | 104      | Barack   | Maths   |    90 | 87698753256 |
|       5 | 105      | Rinky    | Maths   |    85 | 67531579757 |
|       6 | 106      | Adam     | Science |    92 | 79642256864 |
|       7 | 107      | Andrew   | Science |    83 | 56742437579 |
|       8 | 108      | Brayan   | Science |    85 | 75234165670 |
+---------+----------+----------+---------+-------+-------------+
8 rows in set (0.00 sec)
```

2) Again, we will create a new table named "student_detail" as follows:

```
CREATE TABLE student_detail (
  stud_id int NOT NULL,
  stud_code varchar(15) DEFAULT NULL,
  stud_name varchar(35) DEFAULT NULL,
  subject varchar(25) DEFAULT NULL,
  marks int DEFAULT NULL,
  phone varchar(15) DEFAULT NULL,
  Lasinserted Time, PRIMARY KEY (stud_id)
);
```

Example- After insert

3) Next, we will use a CREATE TRIGGER statement to create an after_insert_details trigger on the student_info table. This trigger will be fired after an insert operation is performed on the table.

```
mysql> DELIMITER //
mysql> Create Trigger after_insert_details
AFTER INSERT ON student_info FOR EACH ROW
BEGIN
INSERT INTO student_detail VALUES (new.stud_id,
new.stud_code, new.stud_name, new.subject, new.marks,
new.phone, CURTIME());
END //
```

4) If the trigger is created successfully, we will get the output as follows:

MySQL 8.0 Command Line Client

```
mysql>  Create Trigger after_insert_details
    -> AFTER INSERT ON student_info FOR EACH ROW
    -> BEGIN
    -> INSERT INTO student_detail VALUES (new.stud_id, new.stud_code,
    -> new.stud_name, new.subject, new.marks, new.phone, CURTIME());
    -> END //
Query OK, 0 rows affected (0.12 sec)
```

Example- After insert

5) We can use the following statements to invoke the above-created trigger:

```
mysql> INSERT INTO student_info VALUES
(10, 110, 'Alexandar', 'Biology', 67, '2347346438');
```

6) The table that has been modified after the update query executes is student_detail. We can verify it by using the SELECT statement as follows:

```
mysql> SELECT * FROM student_detail;
```

MySQL 8.0 Command Line Client

```
mysql> INSERT INTO student_info VALUES
    -> (10, 110, 'Alexandar', 'Biology', 67, '2347346438');
Query OK, 1 row affected (0.09 sec)

mysql> SELECT * FROM student_detail;
+---------+----------+----------+---------+------+------------+-------------+
| stud_id | stud_code | stud_name | subject | marks | phone      | Lasinserted |
+---------+----------+----------+---------+------+------------+-------------+
|      10 | 110      | Alexandar | Biology |    67 | 2347346438 | 14:41:35    |
+---------+----------+----------+---------+------+------------+-------------+
1 row in set (0.00 sec)
```

Example- Before Update

1) Create sales_info table

MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM sales_info;
+----+--------------------+----------+------------+
| id | product            | quantity | fiscalYear |
+----+--------------------+----------+------------+
|  1 | 2003 Maruti Suzuki |      110 |       2020 |
|  2 | 2015 Avenger       |      120 |       2020 |
|  3 | 2018 Honda Shine   |      150 |       2020 |
|  4 | 2014 Apache        |      150 |       2020 |
+----+--------------------+----------+------------+
4 rows in set (0.00 sec)
```

2) Next, we will use a CREATE TRIGGER statement to create a BEFORE UPDATE trigger. This trigger is invoked automatically before an update event occurs in the table.

```
DELIMITER $$
CREATE TRIGGER before_update_salesInfo
BEFORE UPDATE
ON sales_info FOR EACH ROW
BEGIN
    DECLARE error_msg VARCHAR(255);
    SET error_msg = ('The new quantity cannot be
greater than 2 times the current quantity');
    IF new.quantity > old.quantity * 2 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = error_msg;
    END IF;
END $$
DELIMITER ;
```

Example- Before Update

3) We will execute the below statements that update the quantity of the row as 600 whose id = 2

mysql> **UPDATE** sales_info **SET** quantity = 600 **WHERE** id = 2;

4) It will give the error as follows because it violates the rule. See the below output.

Output:

MySQL 8.0 Command Line Client

```
mysql> DELIMITER ;
mysql> UPDATE sales_info SET quantity = 125 WHERE id = 2;
Query OK, 1 row affected (0.08 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE sales_info SET quantity = 600 WHERE id = 2;
ERROR 1644 (45000): The new quantity cannot be greater than 2 times the current quantity
```

Example- After Delete

1) Create salaries table

MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM salaries;
+---------+------------+----------+
| emp_num | valid_from | amount   |
+---------+------------+----------+
|     102 | 2020-01-10 | 45000.00 |
|     103 | 2020-01-10 | 65000.00 |
|     105 | 2020-01-10 | 55000.00 |
|     107 | 2020-01-10 | 70000.00 |
|     109 | 2020-01-10 | 40000.00 |
+---------+------------+----------+
5 rows in set (0.00 sec)
```

2) We will create another table named total_salary_budget that keeps the salary information from the salaries table.

```
CREATE TABLE total_salary_budget(
    total_budget DECIMAL(10,2) NOT NULL);
```

3) We will use the SUM() function that returns the total salary from the salaries table and keep this information in the total_salary_budget table:

MySQL 8.0 Command Line Client

```
mysql> INSERT INTO total_salary_budget (total_budget)
    -> SELECT SUM(amount) FROM salaries;
Query OK, 1 row affected (0.09 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM total_salary_budget;
+--------------+
| total_budget |
+--------------+
|    275000.00 |
+--------------+
1 row in set (0.00 sec)
```

Example- After Delete

14We will then create an AFTER DELETE trigger that updates the total salary into the total_salary_budget table after a row is deleted from the salaries table.

```
DELIMITER $$
CREATE TRIGGER after_delete_salaries
AFTER DELETE
ON salaries FOR EACH ROW
BEGIN
   UPDATE total_salary_budget SET total_budget =
total_budget - old.amount;
END$$
DELIMITER ;
```

5)Now, we will delete a salary from the salaries table to invoke the above-created trigger.

6)Next, we will query data from the total_salary_budget table. We can see that table has been modified after the execution of the query. See the below output:

MySQL 8.0 Command Line Client

```
mysql> DELETE FROM salaries WHERE emp_num = 105;
Query OK, 1 row affected (0.15 sec)

mysql> SELECT * FROM total_salary_budget;
+--------------+
| total_budget |
+--------------+
|    220000.00 |
+--------------+
1 row in set (0.00 sec)
```

Thank you!