MySQL: Triggers

Tushar B. Kute,
http://tusharkute.com

Trigger

- A trigger in MySQL is a set of SQL statements that reside in a system catalog.
- It is a special type of stored procedure that is invoked automatically in response to an event.
- Each trigger is associated with a table, which is activated on any DML statement such as INSERT, UPDATE, or DELETE.

Trigger

- A trigger is called a special procedure because it cannot be called directly like a stored procedure.
- The main difference between the trigger and procedure is that a trigger is called automatically when a data modification event is made against a table.
- In contrast, a stored procedure must be called explicitly.

Trigger : Types

- Generally, triggers are of two types according to the SQL standard: row-level triggers and statement-level triggers.
- Row-Level Trigger:
  - It is a trigger, which is activated for each row by a triggering statement such as insert, update, or delete. For example, if a table has inserted, updated, or deleted multiple rows, the row trigger is fired automatically for each row affected by the insert, update, or delete statement.
- Statement-Level Trigger:
  - It is a trigger, which is fired once for each event that occurs on a table regardless of how many rows are inserted, updated, or deleted.

Trigger : Why?

- Triggers help us to enforce business rules.
- Triggers help us to validate data even before they are inserted or updated.
- Triggers help us to keep a log of records like maintaining audit trails in tables.
- SQL triggers provide an alternative way to check the integrity of data.
- Triggers provide an alternative way to run the scheduled task.
- Triggers increases the performance of SQL queries because it does not need to compile each time the query is executed.
- Triggers reduce the client-side code that saves time and effort.
- Triggers help us to scale our application across different platforms.
- Triggers are easy to maintain.

Trigger : Limitations

- MySQL triggers do not allow to use of all validations; they only provide extended validations.
  - For example, we can use the NOT NULL, UNIQUE, CHECK and FOREIGN KEY constraints for simple validations.
- Triggers are invoked and executed invisibly from the client application. Therefore, it isn't easy to troubleshoot what happens in the database layer.
- Triggers may increase the overhead of the database server.

Trigger : Where?

- 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.

Naming Conventions

- Naming conventions are the set of rules that we follow to give appropriate unique names. It saves our time to keep the work organize and understandable.
- Therefore, we must use a unique name for each trigger associated with a table. However, it is a good practice to have the same trigger name defined for different tables.
- The following naming convention should be used to name the trigger in MySQL:
  - (BEFORE | AFTER) table_name (INSERT | UPDATE | DELETE)

    Thus,
  - Trigger Activation Time: BEFORE | AFTER
  - Trigger Event: INSERT | UPDATE | DELETE

How to create trigger?

- We can use the CREATE TRIGGER statement for creating a new trigger in MySQL. Below is the syntax of creating a trigger in MySQL:

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

How to create trigger?

- trigger_name:
  - It is the name of the trigger that we want to create. It must be written after the CREATE TRIGGER statement. It is to make sure that the trigger name should be unique within the schema.
- trigger_time:
  - It is the trigger action time, which should be either BEFORE or AFTER. It is the required parameter while defining a trigger.
  - It indicates that the trigger will be invoked before or after each row modification occurs on the table.

How to create trigger?

- trigger_event:
  - It is the type of operation name that activates the trigger. It can be either INSERT, UPDATE, or DELETE operation.
  - The trigger can invoke only one event at one time. If we want to define a trigger which is invoked by multiple events, it is required to define multiple triggers, and one for each event.
- table_name:
  - It is the name of the table to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.

How to create trigger?

- The trigger body can access the column's values, which are affected by the DML statement. The NEW and OLD modifiers are used to distinguish the column values BEFORE and AFTER the execution of the DML statement.
- We can use the column name with NEW and OLD modifiers as OLD.col_name and NEW.col_name.
- The OLD.column_name indicates the column of an existing row before the updation or deletion occurs.
- NEW.col_name indicates the column of a new row that will be inserted or an existing row after it is updated.

How to create trigger?

- For example, suppose we want to update the column name message_info using the trigger.
- In the trigger body, we can access the column value before the update as OLD.message_info and the new value NEW.message_info.
- We can understand the availability of OLD and NEW modifiers with the below table:

Trigger Event	OLD	NEW
INSERT	No	Yes
UPDATE	Yes	Yes
DELETE	Yes	No

Example:

- Let us start creating a trigger in MySQL that makes modifications in the employee table.
- First, we will create a new table named employee by executing the below statement:

```
CREATE TABLE employee(
   name varchar(45) NOT NULL,
   occupation varchar(35) NOT NULL,
   working_date date,
   working_hours varchar(10)
);
```

Example:

- Next, execute the below statement to fill the records into the employee table:
- ```
  INSERT INTO employee VALUES
  ('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);
  ```
  ```

Example:

- 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:

- 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);
```

After Insert

- After Insert Trigger in MySQL is invoked automatically whenever an insert event occurs on the table.
- Syntax:

```
CREATE TRIGGER trigger_name
AFTER INSERT
ON table_name FOR EACH ROW
trigger_body ;
```

After Insert

- If we want to execute multiple statements, we will use the BEGIN END block that contains a set of SQL queries to define the logic for the trigger. See the below syntax:

```
DELIMITER $$
CREATE TRIGGER trigger_name AFTER INSERT
ON table_name FOR EACH ROW
BEGIN
  variable declarations
  trigger code
END$$
DELIMITER ;
```

After Insert

- We can access the NEW values but cannot change them in an AFTER INSERT trigger.
- We cannot access the OLD If we try to access the OLD values, we will get an error because there is no OLD on the INSERT trigger.
- We cannot create the AFTER INSERT trigger on a VIEW.

After Insert: Example

- Suppose we have created a table named "student_info" as follows:

```
CREATE TABLE student_info (
  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,
  PRIMARY KEY (stud_id)
)
```

After Insert: Example

- 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)
);
```

After Insert: Example

- 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 //
```

After Insert: Example

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

- ```
  mysql> INSERT INTO student_info VALUES
  (10, 110, 'Alexandar', 'Biology', 67,
  '2347346438');
  ```
- The table that has been modified after the update query executes is student_detail.
  ```

Show triggers

- The show or list trigger is much needed when we have many databases that contain various tables.
- Sometimes we have the same trigger names in many databases; this query plays an important role in that case.
- We can get the trigger information in the database server using the below statement.
- This statement returns all triggers in all databases:

```
mysql> SHOW TRIGGERS;
```

Drop triggers

- We can drop an existing trigger from the database by using the DROP TRIGGER statement with the below syntax:

- ```
  DROP TRIGGER [IF EXISTS]
  [schema_name.]trigger_name;
  ```

- ```
  mysql> DROP TRIGGER
  employeedb.before_update_salaries;
  ```

Thank you

@mitu_skillologies

@mITuSkillologies

@mitu_group

@mitu-skillologies

@MITUSkillologies

kaggle

@mituskillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

@mituskillologies

contact@mitu.co.in

tushar@tusharkute.com