Lab 9– Creating Triggers

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**.

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.

Generally, **triggers are of two types** according to the SQL standard: row-level triggers and statement-level triggers.

Why we need/use triggers in MySQL?

We need/use triggers in MySQL due to the following features:

- 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.

Limitations of Using Triggers in MySQL

- 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.

Types of Triggers in MySQL?

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

1. **Before Insert:** It is activated before the insertion of data into the table.
2. **After Insert:** It is activated after the insertion of data into the table.
3. **Before Update:** It is activated before the update of data in the table.
4. **After Update:** It is activated after the update of the data in the table.
5. **Before Delete:** It is activated before the data is removed from the table.
6. **After Delete:** It is activated after the deletion of data from the table.

When we use a statement that does not use INSERT, UPDATE or DELETE query to change the data in a table, the triggers associated with the trigger will not be invoked.

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:

1. (BEFOR | **AFTER**) table_name (**INSERT** | **UPDATE** | **DELETE**)

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

How to create triggers in MySQL?

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;
```

MySQL Create Trigger

In this article, we are going to learn how to create the first trigger in MySQL. We can create a new trigger in MySQL by using the CREATE TRIGGER statement. It is to ensure that we have trigger privileges while using the CREATE TRIGGER command. The following is the basic syntax to create a trigger:

```
1. CREATE TRIGGER trigger_name  trigger_time trigger_event
2. ON table_name FOR EACH ROW
3. BEGIN
4.    --variable declarations
5.    --trigger code
6. END;
```

Parameter Explanation

The create trigger syntax contains the following parameters:
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.
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.
BEGIN END Block: Finally, we will specify the statement for execution when the trigger is activated. If we want to execute multiple statements, we will use the BEGIN END block that contains a set of queries to define the logic for the 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.
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

MySQL Show/List 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:

1. mysql> SHOW TRIGGERS;

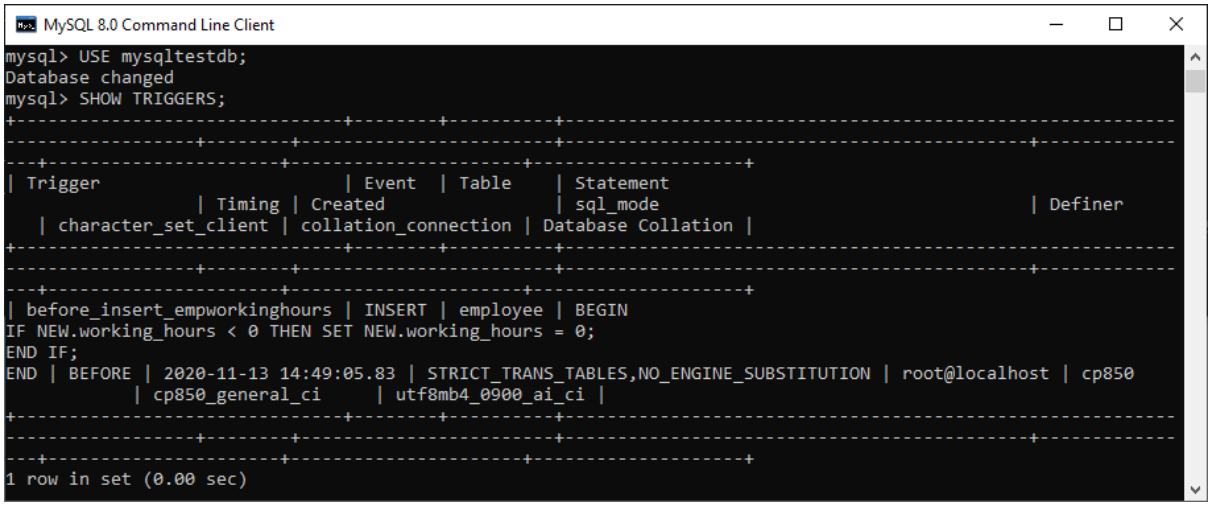
The following steps are necessary to get the list of all triggers:
Step 1: Open the MySQL Command prompt and logged into the database server using the password that you have created during MySQL's installation. After a successful connection, we can execute all the SQL statements.
Step 2: Next, choose the specific database by using the command below:

1. mysql> USE database_name;

Step 3: Finally, execute the SHOW TRIGGERS command.
Let us understand it with the example given below. Suppose we have a database name **"mysqltestdb"** that contains many tables. Then execute the below statement to list the triggers:

1. mysql> USE mysqltestdb;
2. mysql>SHOW TRIGGERS;

The following output explains it more clearly:



Show Triggers Using Pattern Matching

MySQL also provides a **LIKE** clause option that enables us to filter the trigger name using different pattern matching. The following is the syntax to use pattern matching with show trigger command:

1. mysql> SHOW TRIGGERS LIKE pattern;
2. OR,
3. mysql> SHOW TRIGGERS **FROM** database_name LIKE pattern;
1. mysql> SHOW TRIGGERS **WHERE** search_condition;
2. OR,
3. mysql> SHOW TRIGGERS **FROM** database_name **WHERE** search_condition;

Example

Suppose we want to show all triggers that belongs to the **employee** table, execute the statement as follows:

1. mysql> SHOW TRIGGERS **FROM** mysqltestdb **WHERE table** = 'employee';

We will get the output as follows:

Trigger	Event	Table	Statement	Timing	Created	sql_mode	Definer	character_set_	collation_conne	Database Collation

before_insert_empworkinghou	INSERT	employee	BEGIN
			IF NEW.working_hours < 0 THEN SET NEW.working_hours = 0;
			END IF;
			END
				BEFORE	2020-11-13 14:49:05.83	STRICT_TRANS_TABLES,NC	root@localhost	cp850	cp850_general_	utf8mb4_0900

NOTE: It is to note that we must have a SUPER privilege to execute the SHOW TRIGGERS statement.

MySQL BEFORE INSERT TRIGGER

Before Insert Trigger in MySQL is invoked automatically whenever an insert operation is executed. In this article, we are going to learn how to create a before insert trigger with its syntax and example.

Syntax

The following is the syntax to create a BEFORE INSERT trigger in MySQL:

1. **CREATE TRIGGER** trigger_name

```
2. BEFORE INSERT
3. ON table_name FOR EACH ROW
4. Trigger_body ;
```

The BEFORE INSERT trigger syntax parameter can be explained as below:

- First, we will specify the **name of the trigger** that we want to create. It should be unique within the schema.
- Second, we will specify the **trigger action time**, which should be BEFORE INSERT. This trigger will be invoked before each row modifications occur on the table.
- Third, we will specify the **name of a 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.
- Finally, we will specify the statement for execution when the trigger is activated.

If we want to execute multiple statements, we will use the **BEGIN END** block that contains a set of queries to define the logic for the trigger.

```
1. DELIMITER $$
2. CREATE TRIGGER trigger_name BEFORE INSERT
3. ON table_name FOR EACH ROW
4. BEGIN
5.    variable declarations
6.    trigger code
7. END$$
8. DELIMITER ;
```

Restrictions

- We can access and change the **NEW** values only in a BEFORE INSERT trigger.
- We cannot access the **OLD** If we try to access the OLD values, we will get an error because OLD values do not exist.
- We cannot create a BEFORE INSERT trigger on a **VIEW**.

MySQL AFTER INSERT Trigger

After Insert Trigger in MySQL is invoked automatically whenever an insert event occurs on the table. In this article, we are going to learn how to create an after insert trigger with its syntax and example.

Syntax

The following is the syntax to create an **AFTER INSERT** trigger in MySQL:

```
1. CREATE TRIGGER trigger_name
2. AFTER INSERT
3. ON table_name FOR EACH ROW
4. trigger_body ;
```

The AFTER INSERT trigger syntax parameter can be explained as below:

- First, we will specify the **name of the trigger** that we want to create. It should be unique within the schema.
- Second, we will specify the **trigger action time**, which should be AFTER INSERT clause to invoke the trigger.

- Third, we will specify the **name of a 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.
- Finally, we will specify the **trigger body** that contains one or more statements for execution when the trigger is activated.

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.

```
1. DELIMITER $$
2. CREATE TRIGGER trigger_name AFTER INSERT
3. ON table_name FOR EACH ROW
4. BEGIN
5.    variable declarations
6.    trigger code
7. END$$
8. DELIMITER ;
```

Restrictions

- 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**.

MySQL BEFORE UPDATE Trigger

BEFORE UPDATE Trigger in MySQL is invoked automatically whenever an update operation is fired on the table associated with the trigger. In this article, we are going to learn how to create a before update trigger with its syntax and example.

Syntax

The following is the syntax to create a BEFORE UPDATE trigger in MySQL:

```
1. CREATE TRIGGER trigger_name
2. BEFORE UPDATE
3. ON table_name FOR EACH ROW
4. trigger_body ;
```

The BEFORE UPDATE trigger syntax parameter are explained as below:

- First, we will specify the **trigger name** that we want to create. It should be unique within the schema.
- Second, we will specify the **trigger action time**, which should be BEFORE UPDATE. This trigger will be invoked before each row of alterations occurs on the table.
- Third, we will specify the name of a 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.

- Finally, we will specify the **trigger body** that contains a statement for execution when the trigger is activated.

If we want to execute multiple statements, we will use the **BEGIN END** block that contains a set of queries to define the logic for the trigger

```
1. DELIMITER $$
2. CREATE TRIGGER trigger_name BEFORE UPDATE
3. ON table_name FOR EACH ROW
4. BEGIN
5.    variable declarations
6.    trigger code
7. END$$
8. DELIMITER ;
```

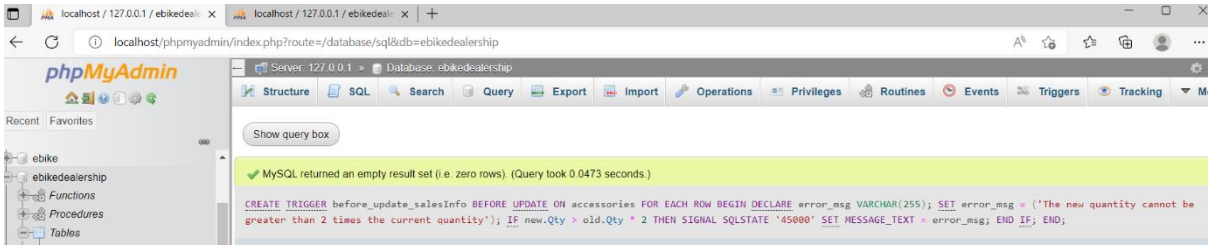
Restrictions

- We cannot update the OLD values in a BEFORE UPDATE trigger.
- We can change the NEW values.
- We cannot create a BEFORE UPDATE trigger on a VIEW.

Demo Example for Ebike:

Create a trigger to show error message and stops the updation of acessories table if we update the value in the quantity column to a new value two times greater than the current value.

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



Execution:

Check the current value of accessories table:

Extra options

	P_ID	Description	Qty	Price	Service_ID
Edit Copy Delete	700	Air Filter	0	217.99	400
Edit Copy Delete	701	Chain Set	3	1987.99	401
Edit Copy Delete	702	Clutch Plate	2	517.99	404
Edit Copy Delete	703	Handle Bar	5	2519.99	405
Edit Copy Delete	704	Horn	10	1675.78	402
Edit Copy Delete	705	Leg Guard	101	6217.99	403

Execute update statement:

UPDATE accessories
set Qty=25
where P_ID=704;

Error

SQL query: Copy

```
UPDATE accessories
set Qty=25
where P_ID=704;
```

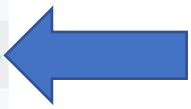
MySQL said:

```
#1644 - The new quantity cannot be greater than 2 times the current quantity
```

Now we will update P_ID=703 and it gets updated from Qty =5 to Qty=8 as its new value is not 2 times larger than the current quantity.

```
UPDATE accessories
set Qty=8
where P_ID=703;
```

P_ID	Description	Qty	Price	Service_ID
700	Air Filter	0	217.99	400
701	Chain Set	1	1987.99	401
702	Clutch Plate	2	517.99	404
703	Handle Bar	8	2519.99	405
704	Horn	10	1675.78	402
705	Leg Guard	101	6217.99	403



Assignment1:

Write a trigger on insert to compartment table when a new compartment gets added to a train and make sure that total number of compartments available in the train does not exceed more than 4.

Assignment2:

create a trigger to add payment information to the backup table when we try to delete some information from ticket table.

Submission Instructions:
Please create a single pdf file with Trigger code and execution screenshots for normal case and error case.