Create and Delete a Trigger

This lesson shows how a trigger is created and demonstrates how it automatically updates an event log. We also show how to drop a trigger in this lesson.

Creating a View

A trigger is associated with an event and runs automatically when the event occurs. Triggers can be associated with 6 types of events in MySQL which are; before data is inserted into the table, after data is inserted into the table, before data is updated, after data is updated, before data is deleted, and after data is deleted.

A trigger must be associated with a table. Triggers without a table cannot exist in MySQL. Every trigger associated with a table must have a unique name but the same trigger name can be used with different tables. The **CREATE TRIGGER** statement is used to create a new trigger. A trigger can be defined before or after an event and the time should be defined using keywords **BEFORE** or **AFTER**. The events that cause the trigger to execute are specified as **INSERT**, **UPDATE**, or **DELETE**.

To delete a trigger, the **DROP TRIGGER** statement is used. Since triggers are associated with tables, when tables are dropped the associated triggers automatically get dropped. In case we want to delete a trigger that belongs to another database, the name of the database is specified before the trigger name.

Syntax

```
CREATE TRIGGER trigger_name trigger_time trigger_event

ON table_name
```

FOR EACH ROW

trigger_body

DROP TRIGGER [IF EXISTS] **[database_name.]trigger_name**;

Connect to the terminal below by clicking in the widget. Once connected, the command line prompt will show up. Enter or copy-paste the command **./DataJek/Lessons/46lesson.sh** and wait for the mysql prompt to start-up.

```
-- The lesson queries are reproduced below for convenient copy/paste into the terminal.

-- Query 1
DELIMITER **
CREATE TRIGGER NetWorthCheck
BEFORE INSERT ON Actors
FOR EACH ROW
  IF  NEW.NetWorthInMillions < 0 OR NEW.NetWorthInMillions IS NULL
  THEN SET New.NetWorthInMillions = 0;
  END IF;
**
DELIMITER ;

-- Query 2
SHOW TRIGGERS;

-- Query 3
INSERT INTO Actors (FirstName, SecondName, DoB, Gender, MaritalStatus,  NetWorthInMillions) V
INSERT INTO Actors (FirstName, SecondName, DoB, Gender, MaritalStatus, NetWorthInMillions) VA
INSERT INTO Actors (FirstName, SecondName, DoB, Gender, MaritalStatus,  NetWorthInMillions) V

-- Query 4
SELECT * FROM Actors;

-- Query 5
DROP TRIGGER IF EXISTS NetWorthCheck;
```

Terminal

1. One of the basic uses of triggers is to validate the user input. Let's create such a trigger on the **Actors** table. The purpose of our trigger is to check the value for **NetWorthInMillions** column before the

record is inserted in the **Actors** table. In case the value is not specified, or if the user provides a non negative value by mistake, it will be set the value to zero. We will define the trigger **NetWorthCheck** as follows:

```
DELIMITER **
CREATE TRIGGER NetWorthCheck
BEFORE INSERT ON Actors
FOR EACH ROW
  IF  NEW.NetWorthInMillions < 0 OR NEW.NetWorthInMillions IS NUL
L
  THEN SET New.NetWorthInMillions = 0;
  END IF;
**
DELIMITER ;
```

The trigger has been created successfully. Here we have used the **DELIMITER** command in line 1. By default semicolon (;) is used to separate two statements. In the trigger above, we have multiple statements that end with semicolon character and thus the trigger won't be considered as a single statement. The **DELIMITER** keyword is used to redefine the delimiter to **** so that we can pass the whole trigger to the server as a single statement. We use our redefined delimiter in line 8 to signal the end of the trigger. Then the delimiter is set back to semicolon in line 9.

When defining triggers, we specify the trigger time (**BEFORE or AFTER**) and the trigger event (**INSERT, UPDATE or DELETE**) after the trigger name in line 3. Next, the trigger body contains the logic of the trigger. Here, we are using an **IF** statement to check the user-specified value of **NetWorthInMillions**. The keyword **NEW** in line 5 is used to access the value of the column before the trigger event occurs. We will give more explanation of this keyword in later lessons.

2. To display the triggers in a database **SHOW TRIGGERS** command is used.

```
SHOW TRIGGERS;
```

This query returns a set of columns that list the name of the trigger, the event that invokes the trigger, the table with which the trigger is associated, the SQL statement of the trigger, the time when the trigger is executed, the creation time of the trigger, the sql_mode when the trigger executes, the user account that created the trigger as well as the character set and collation information.

The **SHOW TRIGGERS** command can also be used to shortlist triggers by using the **LIKE** clause and specifying a pattern.

3. Now it is time to see how the trigger works. We will insert three rows in the Actors table where **NetWorthInMillions** is positive, negative and NULL. The INSERT operation will automatically cause the trigger to run.

```
INSERT INTO Actors (FirstName, SecondName, DoB, Gender, MaritalSt
atus,  NetWorthInMillions) VALUES ('Tom', 'Hanks', '1956-07-09',
'Male', 'Married', 350);

INSERT INTO Actors (FirstName, SecondName, DoB, Gender, MaritalSt
atus, NetWorthInMillions) VALUES ('Young', 'Actor', '2000-03-25',
 'Male', 'Single', NULL);

INSERT INTO Actors (FirstName, SecondName, DoB, Gender, MaritalSt
atus,  NetWorthInMillions) VALUES ('Old', 'Actor', '1960-10-02',
'Male', 'Married', -540);
```

The rows have been successfully inserted in the table and the associated trigger event has also occurred. Let's query the **Actors** table to check the values in the **NetWorthInMillions** column:

```
SELECT * FROM Actors;
```

As can be seen, the trigger was invoked before the rows were inserted in the table. The NULL and negative values in two INSERT queries were changed to zero.

queries were changed to zero.

4. To delete the trigger, use the following statement:

```
DROP TRIGGER IF EXISTS NetWorthCheck;
```

   The **IF EXISTS** clause is optional and is used to avoid an error message when a trigger that does not exist is deleted using the **DROP TRIGGER** statement. If this optional clause is used, a note is issued and the statement is successfully executed even when the trigger does not exist. The image shows that the trigger we created in step 1 has been successfully deleted.