INSERT Triggers

This lesson demonstrates how to create triggers that are associated with the INSERT statement.

INSERT Triggers

The insert triggers are fired whenever an **INSERT** operation occurs. They can be executed before or after the insert query is executed. In the insert triggers, we can only use the **NEW** keyword to access the new values of a column. The **OLD** keyword cannot be used because there are no previous values for an **INSERT** statement. The **BEFORE INSERT** trigger can be used for data validation or for maintaining a summary table of another table. The **AFTER INSERT** trigger can be used for maintaining an activity log or to copy the values in a table to another table.

Syntax

> CREATE TRIGGER **trigger_name [BEFORE | AFTER]** INSERT
>
> ON **table_name**
>
> FOR EACH ROW
>
> **trigger_body**

> 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/47lesson.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
CREATE TABLE NetWorthStats (
AverageNetWorth DECIMAL(10,4)
);
INSERT INTO NetWorthStats(AverageNetWorth)
Values ((SELECT AVG(NetWorthInMillions) FROM Actors));

-- Query 2
DELIMITER **
CREATE TRIGGER BeforeActorsInsert
BEFORE INSERT ON Actors
FOR EACH ROW
BEGIN
  DECLARE TotalWorth, RowsCount INT;

  SELECT SUM(NetWorthInMillions) INTO TotalWorth
  FROM Actors;
  SELECT COUNT(*) INTO RowsCount
  FROM Actors;

  UPDATE NetWorthStats
  SET AverageNetWorth = ((Totalworth + new.NetWorthInMillions) / (RowsCount+1));
END **
DELIMITER ;

-- Query 3
INSERT INTO Actors (FirstName, SecondName, DoB, Gender, MaritalStatus, NetWorthInMillions)
VALUES ('Charlize', 'Theron', '1975-08-07', 'Female', 'Single', 130);

SELECT * FROM NetWorthStats;

-- Query 4
CREATE TABLE ActorsLog (
    LogId INT AUTO_INCREMENT PRIMARY KEY,
    ActorId INT NOT NULL,
    FirstName VARCHAR(20),
    LastName VARCHAR(20),
    DateTime DATETIME DEFAULT NULL,
    Event VARCHAR(50) DEFAULT NULL
);

-- Query 5
CREATE TRIGGER AfterActorsInsert
AFTER INSERT ON Actors
FOR EACH ROW
INSERT INTO ActorsLog
SET ActorId = NEW.Id,
    FirstName = New.FirstName,
    LastName = NEW.SecondName,
    DateTime = NOW(),
    Event = 'INSERT';

-- Query 6
INSERT INTO Actors (FirstName, SecondName, DoB, Gender, MaritalStatus, NetWorthInMillions)
VALUES ('Matt', 'Damon', '1970-10-08', 'Male', 'Married', 160);

SELECT * FROM ActorsLog;
```

Terminal

1. We will first cover the **BEFORE INSERT** triggers. Suppose we want to store the **AverageNetWorth** of all actors in our **Actors** table. This value will change every time a new row is inserted in the table. We will create a table **NetWorthStats** to store this value and populate the table as follows:

```
CREATE TABLE NetWorthStats (
AverageNetWorth DECIMAL(10,4)
);
INSERT INTO NetWorthStats(AverageNetWorth)
Values ((SELECT AVG(NetWorthInMillions) FROM Actors));
```

```
mysql> CREATE TABLE NetWorthStats (
    -> AverageNetWorth DECIMAL(10,4) NOT NULL
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql>
mysql> INSERT INTO NetWorthStats(AverageNetWorth)
    -> Values ((SELECT AVG(NetWorthInMillions) FROM Actors));
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql>
mysql> SELECT * FROM NetWorthStats;
+----------------+
| AverageNetWorth |
+----------------+
|        346.1818 |
+----------------+
1 row in set (0.00 sec)
```

The value **AverageNetWorth** in the above table will be automatically updated before a new actor is added to the **Actors** table.

2. Next, we will create a trigger **BeforeActorsInsert** for the **NetWorthStats** table as follows:

```
DELIMITER **

CREATE TRIGGER BeforeActorsInsert
BEFORE INSERT ON Actors
FOR EACH ROW
BEGIN
  DECLARE TotalWorth, RowsCount INT;
```

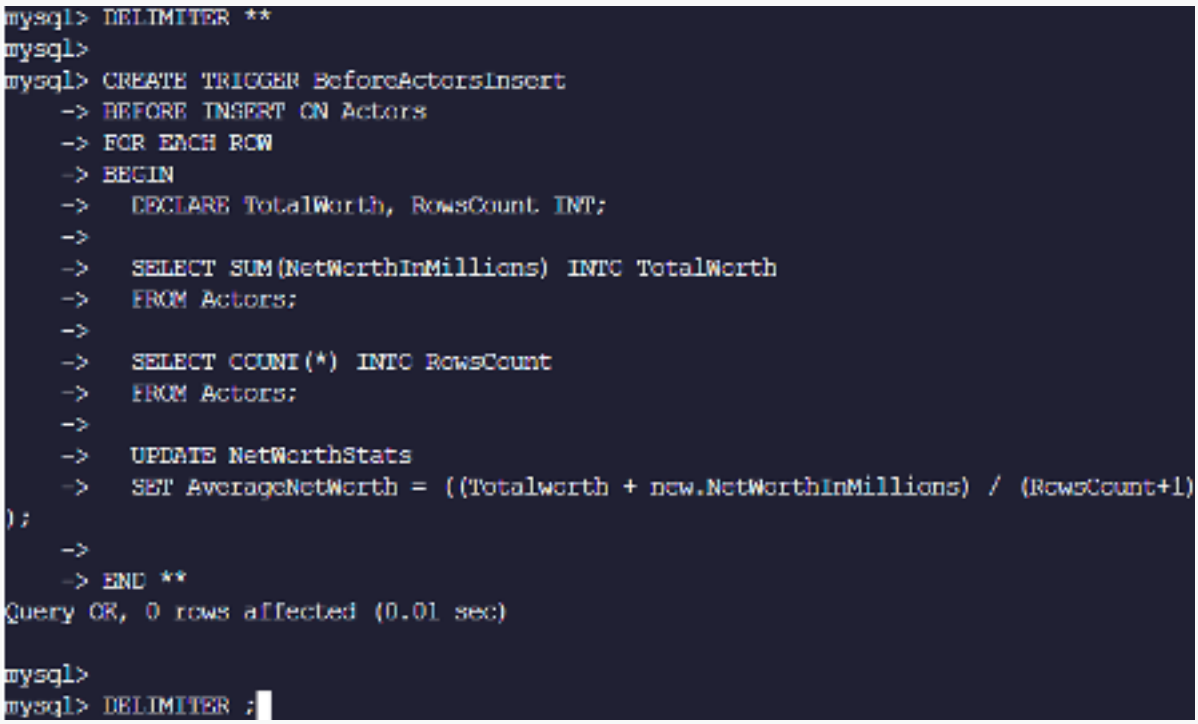
```
  SELECT SUM(NetWorthInMillions) INTO TotalWorth
  FROM Actors;

  SELECT COUNT(*) INTO RowsCount
  FROM Actors;

  UPDATE NetWorthStats
  SET AverageNetWorth = ((Totalworth + new.NetWorthInMillions) /
(RowsCount+1));

END **

DELIMITER ;
```



The image shows that trigger has been created. The **BeforeActorsInsert** trigger is associated with the **Actors** table and runs **BEFORE INSERT** operation. In the trigger body, we first fetch the sum of **NetWorthInMillions** column in a variable **TotalWorth**. Then we find the number of rows in the **Actors** table and save it in a variable **RowsCount**. Next, update the **AverageNetWorth** by adding the **NetWorthInMillions** of the new record to the total of all records and then dividing it by the total number of rows to get the result.

We have put all trigger logic between the **BEGIN** and **END** block because the trigger body consists of multiple statements.

3. To test if our trigger works, we will insert a row in the **Actors** table

and then check the **NetWorthStats** table as follows:

```
INSERT INTO Actors (FirstName, SecondName, DoB, Gender, MaritalSt
atus, NetWorthInMillions)
VALUES ('Charlize', 'Theron', '1975-08-07', 'Female', 'Single', 1
30);

SELECT * FROM NetWorthStats;
```

```
mysql> INSERT INTO Actors (FirstName, SecondName, DoB, Gender, MaritalStatus, NetW
orthInMillions)
    -> VALUES ('Charlize', 'Theron', '1975-08-07', 'Female', 'Single', 130);
Query OK, 1 row affected (0.01 sec)

mysql>
mysql> SELECT * FROM NetWorthStats;
+-----------------+
| AverageNetWorth |
+-----------------+
|        328.1667 |
+-----------------+
1 row in set (0.00 sec)
```

As seen, the value of **AverageNetWorth** has changed from 346.1818 to 328.1667 when a new row is added in the **Actors** table.

The value in the **NetWorthStats** table can also change in the event of an update or delete operation on the Actors table. Hence, we need more triggers to handle those events and update the value of **AverageNetWorth** accordingly.

4. Now we will discuss the **AFTER INSERT** triggers. Triggers can be used to maintain logs. Let us create such a trigger on the **Actors** table. We will begin by creating a table **ActorsLog** which will be used to keep track of changes made to the **Actors** table. We want to store the ActorId, first and last name, the time and date of the action as well as the type of event (insert, update or delete) that caused a change in the table. Execute the following query:

```
CREATE TABLE ActorsLog (
    LogId INT AUTO_INCREMENT PRIMARY KEY,
    ActorId INT NOT NULL,
    FirstName VARCHAR(20),
    LastName VARCHAR(20),
    DateTime DATETIME DEFAULT NULL,
    Event VARCHAR(50) DEFAULT NULL
);
```

```
);
```

```
mysql> CREATE TABLE ActorsLog (
    ->     LogId INT AUTO_INCREMENT PRIMARY KEY,
    ->     ActorId INT NOT NULL,
    ->     FirstName VARCHAR(20),
    ->     LastName VARCHAR(20),
    ->     DateTime DATETIME DEFAULT NULL,
    ->     Event VARCHAR(50) DEFAULT NULL
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql> SELECT * FROM ActorsLog;
Empty set (0.01 sec)
```

5. Next, we will define a trigger **AfterActorsInsert**, that will insert a row in the **ActorsLog** table whenever an **INSERT** operation occurs on the **Actors** table.

```
CREATE TRIGGER AfterActorsInsert
AFTER INSERT ON Actors
FOR EACH ROW
INSERT INTO ActorsLog
SET ActorId = NEW.Id,
    FirstName = New.FirstName,
    LastName = NEW.SecondName,
    DateTime = NOW(),
    Event = 'INSERT';
```

```
mysql> CREATE TRIGGER AfterActorsInsert
    -> AFTER INSERT ON Actors
    -> FOR EACH ROW
    -> INSERT INTO ActorsLog
    -> SET ActorId = NEW.Id,
    ->     FirstName = New.FirstName,
    ->     LastName = NEW.SecondName,
    ->     DateTime = NOW(),
    ->     Event = 'INSERT';
Query OK, 0 rows affected (0.01 sec)
```

6. Now it is time to see how the trigger works. We will insert a row in the **Actors** table and the **INSERT** operation will automatically cause the trigger to run.

```
INSERT INTO Actors (FirstName, SecondName, DoB, Gender, MaritalSt
atus, NetWorthInMillions)
VALUES ('Matt', 'Damon', '1970-10-08', 'Male', 'Married', 160);
```

```
SELECT * FROM ActorsLog;
```

The row is successfully inserted in the **Actors** table and the associated trigger event has also occurred as can be seen from the **ActorsLog** table:

```
mysql> INSERT INTO Actors (FirstName, SecondName, DoB, Gender, MaritalStatus, NetW
orthInMillions)
    -> VALUES ('Matt', 'Damon', '1970-10-08', 'Male', 'Married', 160);
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> SELECT * FROM ActorsLog;
+-------+---------+-----------+----------+---------------------+--------+
| LogId | ActorId | FirstName | LastName | DateTime            | Event  |
+-------+---------+-----------+----------+---------------------+--------+
|     1 |      13 | Matt      | Damon    | 2020-03-27 16:39:02 | INSERT |
+-------+---------+-----------+----------+---------------------+--------+
1 row in set (0.00 sec)

mysql>
```