DELETE Triggers

In this lesson we will learn how to create triggers associated with the DELETE statement.

We'll cover the following

DELETE Triggers

Delete triggers for a table are fired when a **DELETE** statement is executed. Just like the insert and update triggers, delete triggers can be executed before or after a record is deleted from the table. Since the **DELETE** statement is meant to delete a record, the columns do not have a **NEW** value. Only the **OLD** value of a column is accessible and that too cannot be updated.

Delete triggers can be used to archive deleted records. In some cases, **BEFORE DELETE** triggers are used to stop an invalid delete operation for example, if there are two tables for Course and Registration information, then it does not make sense to delete a course when there are students registered for it. Delete triggers can also be used to update a summary table or maintain a change log after records are removed from the table. Delete triggers are not available for views.

Syntax

```
CREATE TRIGGER trigger_name [BEFORE | AFTER] DELETE

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/49lesson.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 ActorsArchive (
       RowId INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
       DeletedAt TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP)
AS (SELECT * FROM Actors WHERE 1=2);

-- Query 2
DELIMITER **
CREATE TRIGGER BeforeActorsDelete
BEFORE DELETE
ON Actors
FOR EACH ROW
BEGIN
  INSERT INTO ActorsArchive
         (Id, Firstname, SecondName, DoB, Gender, MaritalStatus, NetWorthInMillions)
  VALUES (OLD.Id, OLD.Firstname, OLD.SecondName, OLD.DoB, OLD.Gender, OLD.MaritalStatus, OLD.NetWo
END **
DELIMITER ;

-- Query 3
DELETE FROM Actors
WHERE NetWorthInMillions < 150;

-- Query 4
DELIMITER **
CREATE TRIGGER AfterActorsDelete
AFTER DELETE ON Actors
FOR EACH ROW
BEGIN
   DECLARE TotalWorth, RowsCount INT;

   INSERT INTO ActorsLog
   SET ActorId = OLD.Id, FirstName = OLD.FirstName, LastName =  OLD.SecondName, DateTime = NOW(),
   SELECT SUM(NetWorthInMillions) INTO TotalWorth
   FROM Actors;
   SELECT COUNT(*) INTO RowsCount
   FROM Actors;

   UPDATE NetWorthStats
   SET AverageNetWorth = ((Totalworth) / (RowsCount));
END **
DELIMITER ;

-- Query 5
DELETE FROM Actors
WHERE Id = 13;
```

```
SELECT * FROM NetWorthStats;
SELECT * FROM ActorsLog;
```

Terminal

1. We will create a table **ActorsArchive** to store the deleted rows for later reference. This table will be a copy of the **Actors** table because we want to save all information about an actor in the **Actors** table before the record gets deleted. Here is a simple way to create a copy of an existing table:

   ```
   CREATE TABLE ActorsArchive (
          RowId INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
          DeletedAt TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP)
   AS (SELECT * FROM Actors WHERE 1=2);
   ```

   This query will copy all the columns from the **Actors** table and add two new columns **RowId** and **DeletedAt**. The condition in the **WHERE** clause is used to restrict copying rows from **Actors** table to **ActorsArchive** table. In the absence of this **WHERE** clause, the new table will get populated with all the rows from the old table.

2. Now we will define a trigger **BeforeActorsDelete** on the **Actors** table which will copy the record in the **DELETE** query to the **ActorsArchive** table.

   ```
   DELIMITER **

   CREATE TRIGGER BeforeActorsDelete
   BEFORE DELETE
   ON Actors
   FOR EACH ROW
   BEGIN
     INSERT INTO ActorsArchive
            (Id, Firstname, SecondName, DoB, Gender, MaritalStatus, NetW
   orthInMillions)
     VALUES (OLD.Id, OLD.Firstname, OLD.SecondName, OLD.DoB, OLD.Gender,
    OLD.MaritalStatus, OLD.NetWorthInMillions);
   END **
   ```

```
DELIMITER ;
```

The trigger has been created. It will insert a new row in the **ActorArchive** table copying all the details of the record mentioned in the **DELETE** query. We do not need to specify the values for **Id** and **DeletedAt** columns of the **ActorsArchive** table as their default values have been mentioned at the time of creation of the table.

3. To test this trigger, execute the following **DELETE** query:

```
DELETE FROM Actors
WHERE NetWorthInMillions < 150;
```

Four rows match this criterion and are deleted from the **Actors** table. The actor details and time of deletion is saved in the **ActorsArchive** table as seen below:

4. Now let's consider an example of **AFTER DELETE** triggers using the **ActorsLog** and **NetWorthStats** tables created in the previous lessons. Whenever an actor is deleted from the table, we will keep a log of this activity in the **ActorsLog** table. The summary table will also be updated to reflect the change in the **NetWorthInMillions** column. The trigger **AfterActorsDelete** is defined as follows:

```
DELIMITER **

CREATE TRIGGER AfterActorsDelete
AFTER DELETE ON Actors
FOR EACH ROW
BEGIN
   DECLARE TotalWorth, RowsCount INT;

   INSERT INTO ActorsLog
   SET ActorId = OLD.Id, FirstName = OLD.FirstName, LastName =  OLD.S
econdName, DateTime = NOW(), Event = 'DELETE';

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

```
    SELECT COUNT(*) INTO RowsCount

    FROM Actors;

    UPDATE NetWorthStats
    SET AverageNetWorth = ((Totalworth) / (RowsCount));
END **

DELIMITER ;
```

This trigger will perform an **INSERT** in the **ActorsLog** table and **UPDATE** the **NetWorthStats** table. We used a similar **INSERT** query in the **AFTER UPDATE** trigger in the previous lesson. The difference here is that **DELETE** triggers only have access to **OLD** values while **UPDATE** triggers can access both the **NEW** and **OLD** values.

5. To test this trigger, we will delete a row from the **Actors** table.

```
DELETE FROM Actors
WHERE Id = 13;

SELECT * FROM NetWorthStats;
SELECT * FROM ActorsLog;
```

AfterDeleteTrigger was fired after the **DELETE** operation was successful and changed the **AverageNetWorth** as well as inserting a row in the **ActorsLog** table:

BeforeDeleteTrigger was also fired before the record was deleted and a new row has been inserted in the **ActorsArchive** table as seen below: