SQL Triggers

CO226 : Database Systems

Lab - 8

Kalani Udayanthi
kalaniu@eng.pdn.ac.lk

What is a Trigger ?

- A trigger is a stored program invoked automatically in response to an event such as **insert**, **update**, or **delete** that occurs in the associated table.

- For example,

  you can define a trigger that is invoked automatically before a new row is inserted into a table.

- MySQL supports triggers that are invoked in response to the **INSERT**, **UPDATE** or **DELETE** event.

Types of Triggers

The SQL standard defines two types of triggers: row-level triggers and statement-level triggers.

- A **row-level trigger** is activated for each row that is inserted, updated, or deleted.

  For example, if a table has 100 rows inserted, updated, or deleted, the trigger is automatically invoked 100 times for the 100 rows affected.

- A **statement-level trigger** is executed once for each transaction regardless of how many rows are inserted, updated, or deleted.

* * **MySQL supports only row-level triggers**. It doesn’t support statement-level triggers.

Create Trigger in MySQL

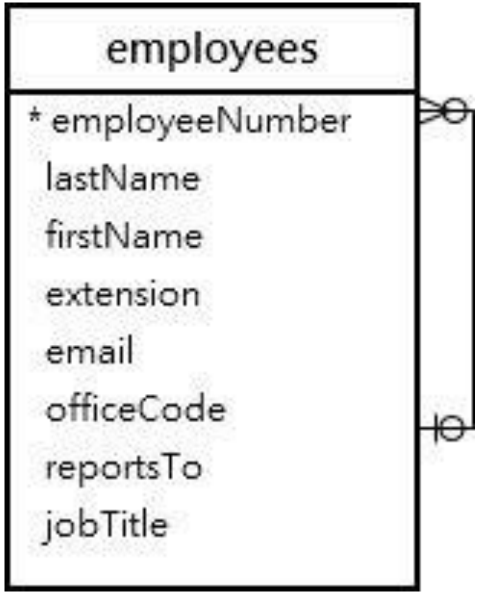
- The CREATE TRIGGER statement creates a new trigger.

  Syntax :

```
mysql > Delimiter//
mysql > CREATE TRIGGER trigger_name
        {BEFORE | AFTER} {INSERT | UPDATE|
        DELETE }
        ON table_name FOR EACH ROW
        Trigger_body;
mysql > Delimiter;
```

Example

1. Create a trigger in MySQL to log the changes of the **employees** table.



- First, create a new table named **employees_audit** to keep the changes to the **employees** table:

```
CREATE TABLE employees_audit (
    id INT AUTO_INCREMENT PRIMARY KEY,
    employeeNumber INT NOT NULL,
    lastname VARCHAR(50) NOT
    NULL,
    changedat DATETIME DEFAULT NULL,
    action VARCHAR(50) DEFAULT NULL
    );
```

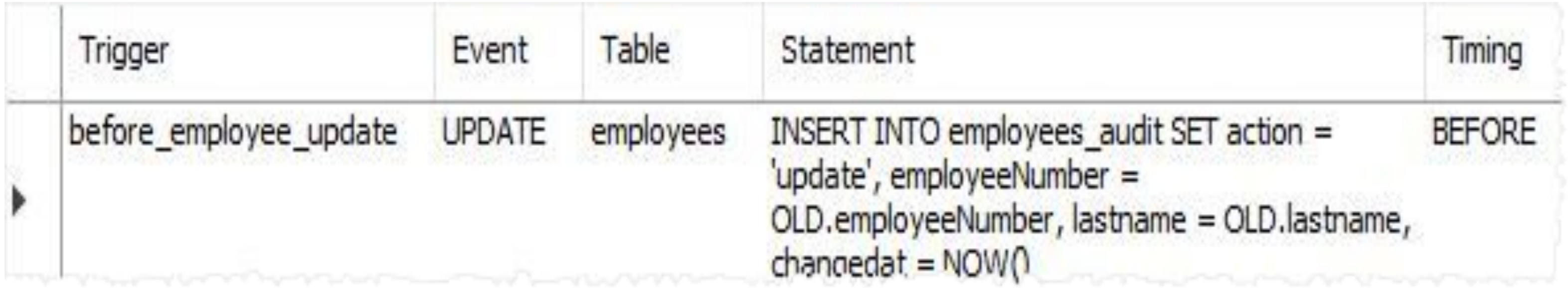
- Next, create a **BEFORE UPDATE** trigger that is invoked before a change is made to the **employees** table.

```
mysql> Delimiter //
mysql> CREATE TRIGGER before_employee_update
                 BEFORE UPDATE ON employees
                 FOR EACH ROW
             INSERT INTO employees_audit
             SET action = 'update',
                 employeeNumber = OLD.employeeNumber,
                 lastname = OLD.lastname,
                 changedat = NOW();
mysql> Delimiter;
```

- Then, show all triggers in the current database by using the SHOW TRIGGERS statement:

  Syntax :

  SHOW TRIGGERS;



Trigger	Event	Table	Statement	Timing
before_employee_update	UPDATE	employees	INSERT INTO employees_audit SET action = 'update', employeeNumber = OLD.employeeNumber, lastname = OLD.lastname, changedat = NOW()	BEFORE

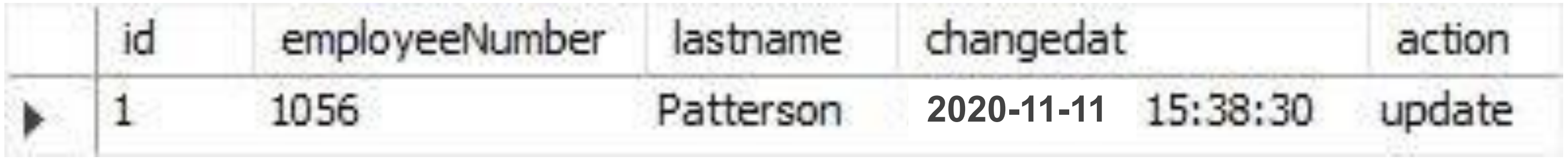
- After that, update a row in the **employees** table:

```
UPDATE employees
SET
    lastName = 'Peterson'
WHERE
    employeeNumber = 1056;
```

- Finally, query the **employees_audit** table to check if the trigger was fired by the **UPDATE** statement:

```
SELECT * FROM employees_audit;
```

- The following shows the output of the query:



	id	employeeNumber	lastname	changedat	action
▶	1	1056	Patterson	2020-11-11 15:38:30	update

Displaying a Trigger

- To display all the Triggers in the current database we use **SHOW TRIGGERS** syntax.

Syntax :

```
SHOW TRIGGERS;
```

Drop a Trigger

- To drop a Trigger in MySQL we use **DROP TRIGGER** syntax.

  Syntax :

  DROP TRIGGER [trigger_name];

- Ex :

  To drop the before_employee_update trigger in previous example:

  DROP TRIGGER before_employee_update;

Advantages of Triggers

- Triggers provide another way to check the integrity of data.
- Triggers handle errors from the database layer.
- Triggers give an alternative way to **run scheduled tasks**. By using triggers, you don’t have to wait for the **scheduled events** to run because the triggers are invoked automatically *before* or *after* a change is made to the data in a table.
- Triggers can be useful for auditing the data changes in tables.

Disadvantages of Triggers

- Triggers can only provide extended validations, not all validations. For simple validations, you can use the **NOT NULL**, **UNIQUE**, **CHECK** and **FOREIGN KEY** constraints.
- Triggers can be difficult to troubleshoot because they execute automatically in the database, which may not invisible to the client applications.
- Triggers may increase the overhead of the MySQL Server.

MySQL - PHP Connectivity

MySQL using MySQL Binary

- You can establish the MySQL database using the **mysql** binary at the command prompt.

Example

Here is a simple example to connect to the MySQL server from the command prompt −

```
[root@host]# mysql -u root -p
Enter password:******
```

Or

```
[root@host]# mysql -u root -ppassword
```

- That will give you the mysql> command prompt where you will be able to execute any SQL command. Following is the result of above command.

  The following code block shows the result of above code −

```
Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 2854760 to server version: 5.0.9



Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

Disconnect MySQL

You can disconnect from the MySQL database any time using the **exit** command at **mysql>** prompt.

```
mysql> exit

Bye
```

MySQL Connection Using PHP Script

- PHP provides **mysql_connect()** function to open a database connection.
- This function takes five parameters and returns a MySQL link identifier on success or FALSE on failure.

Syntax:

```
$conn = mysql_connect(server,user,password,new_link,client_flag);
```

MySQL Connection Using MySQL Improved

- MySQL Improved extension uses the **mysqli** class.
- It uses mysqli_connect() to open a database connection.

Syntax:

```
$conn =  mysqli_connect(server,user,password);
```

MySQL Connection Using PDO

- PDO provide supports for many databases.
- It uses PDO() to open a database connection.

Syntax:

```
$conn =  PDO(‘mysql:host=localhost;dbname=myDB’,user,password);
```

Disconnect MySQL Connection Using PHP Script

- **MySQL Improved**

  You can disconnect from the MySQL database anytime using another PHP function **mysqli_close()**.

  Syntax:

  ```
  mysqli_close ($conn );
  ```

- **PDO**

  Syntax:

  ```
  $conn = null;
  ```

Example :

Try the following example to connect to a MySQL server using MySQLi −

```
<html>
  <head>
    <title>Connecting MySQL Server</title>
  </head>
  <body>
    <?php
      $dbhost = 'localhost:3306';
      $dbuser = 'guest';
      $dbpass = 'guest123';
      $conn = mysqli_connect($dbhost, $dbuser, $dbpass);
      if(! $conn ) {
        die('Could not connect: ' . mysqli_connect_error());
      }
      echo 'Connected successfully';
      mysqli_close($conn);
    ?>
  </body>
</html>
```

// Create connection

// Check connection

// Close connection

How to Insert a record through a PHP Script

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';
  $dbname = ‘test_db’;
  $conn = mysqli_connect($dbhost, $dbuser, $dbpass, $dbname);

  if(! $conn ) {
    die('Could not connect: ' . mysqli_connect_error());
  }

  $sql = 'INSERT INTO employee '.
    '(emp_name,emp_address, emp_salary, join_date)
    '.  'VALUES ( "guest", "XYZ", 20000, NOW() )';

  if(mysqli_query($conn, $sql){
    echo ‘Record inserted successfully’;
  }
  else{
    echo ‘Could not insert record : ’, mysqli_error($conn);
  }

  mysqli_close($conn);
?>
```

Summary

- MySQL using MySQL Binary
- Disconnect MySQL
- MySQL Connection Using PHP Script
    - MySQL
    - MySQLi
    - PDO
- Disconnect MySQL Connection using PHP Script