MySQL Triggers

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

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

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

Create Trigger in MySQL

CREATE TRIGGER trigger_name

1. (AFTER | BEFORE) (INSERT | UPDATE | DELETE)
2. ON table_name FOR EACH ROW
3. BEGIN
4. --variable declarations
5. --trigger code
6. END;

MySQL BEFORE INSERT trigger example

Table 1:

```
  DROP TABLE IF EXISTS WorkCenters;

CREATE TABLE WorkCenters (  id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,     capacity INT NOT NULL );
```

Table 2:

```
DROP TABLE IF EXISTS WorkCenterStats;

CREATE TABLE WorkCenterStats( totalCapacity INT NOT NULL  );
```

Trigger :

```
DROP TRIGGER IF EXISTS before_workcenters_insert;
DELIMITER $$

CREATE TRIGGER before_workcenters_insert
BEFORE INSERT
ON WorkCenters FOR EACH ROW
BEGIN
    DECLARE rowcount INT;

    SELECT COUNT(*)     INTO rowcount     FROM WorkCenterStats;

    IF rowcount > 0 THEN
        UPDATE WorkCenterStats SET totalCapacity = totalCapacity + new.capacity;
```

```
   ELSE
      INSERT INTO WorkCenterStats(totalCapacity) VALUES(new.capacity);

   END IF;

END $$

DELIMITER ;
```

OUTPUT:

```
mysql> INSERT INTO WorkCenters(name, capacity) VALUES('Mold Machine',100);
Query OK, 1 row affected (0.12 sec)

mysql> SELECT * FROM WorkCenterStats;
+---------------+
| totalCapacity |
+---------------+
|           100 |
+---------------+
1 row in set (0.00 sec)

mysql> INSERT INTO WorkCenters(name, capacity) VALUES('Packing',200);
Query OK, 1 row affected (0.87 sec)

mysql> SELECT * FROM WorkCenterStats;
+---------------+
| totalCapacity |
+---------------+
|           300 |
+---------------+
1 row in set (0.00 sec)
```

Before Insert Trigger (With in a single table):

As the name implies, this trigger is invoked before an insert, or before an insert statement is executed.
Example:
Considering tables:

```
create table contacts (contact_id INT (11) NOT NULL AUTO_INCREMENT, last_name VARCHAR (30)
NOT NULL, first_name VARCHAR (25),birthday DATE, created_date DATE,created_by
VARCHAR(30), CONSTRAINT contacts_pk PRIMARY KEY (contact_id));
```

```
drop trigger contacts_before_insert;
delimiter //
create trigger contacts_before_insert
        before insert on contacts for each row
        begin
          DECLARE vUser varchar(50);

          select USER() into vUser;
        SET NEW.created_date = SYSDATE();

        SET NEW.created_by = vUser;
      end; //

delimiter ;


insert into contacts values (1, "Newton", "Enigma",  str_to_date ("19-08-1999", "%d-%m-%Y"),
                     str_to_date ("17-03-2018", "%d-%m-%Y"), "xyz");

INSERT INTO contacts(contact_id,last_name,first_name,birthday)  VALUES(3, 'John','Doe','1990-09-01');
```

```
mysql> INSERT INTO contacts(contact_id,last_name,first_name,birthday)  VALUES(3, 'John','Doe','1990-09-01');
Query OK, 1 row affected (0.07 sec)

mysql> select * from contacts;
+------------+-----------+------------+------------+--------------+----------------+
| contact_id | last_name | first_name | birthday   | created_date | created_by     |
+------------+-----------+------------+------------+--------------+----------------+
|          3 | John      | Doe        | 1990-09-01 | 2021-03-12   | root@localhost |
+------------+-----------+------------+------------+--------------+----------------+
1 row in set (0.00 sec)
```

Before Update Trigger:

As the name implies, it is a trigger which enacts before an update is invoked. If we write an update statement, then the actions of the trigger will be performed before the update is implemented.

Example:
Considering tables:

Table 1:

```
create table customer (acc_no integer primary key, cust_name varchar(20),
                       avail_balance decimal);
```

Table 2:

```
create table mini_statement (acc_no integer, avail_balance decimal,
      foreign key(acc_no) references customer(acc_no) on delete cascade);
```

Insertion :

```
insert into customer values (1000, "Fanny", 7000);
insert into customer values (1001, "Peter", 12000);
```

Trigger to insert (old) values into a mini_statement record (including account number and available balance as parameters) before updating any record in customer record/table:

Trigger :

```
drop trigger update_cus;
delimiter //
create trigger update_cus
     before update on customer
     for each row
     begin
     insert into mini_statement values (old.acc_no, old.avail_balance);
     end; //

delimiter ;

mysql> insert into customer values (1000, "Fanny", 7000);

Query OK, 1 row affected (0.03 sec)

mysql> insert into customer values (1001, "Peter", 12000);

Query OK, 1 row affected (0.40 sec)

mysql> select * from customer;

+--------+-----------+---------------+

| acc_no | cust_name | avail_balance |

+--------+-----------+---------------+

|   1000 | Fanny     |          7000 |

|   1001 | Peter     |         12000 |

+--------+-----------+---------------+

2 rows in set (0.00 sec)

mysql> select * from mini_statement;

Empty set (0.00 sec)

update customer set avail_balance = avail_balance + 2000 where acc_no = 1000;

update customer set avail_balance = avail_balance + 1000 where acc_no = 1001;

mysql> update customer set avail_balance = avail_balance + 2000 where acc_no = 1000;
Query OK, 1 row affected (0.21 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update customer set avail_balance = avail_balance + 1000 where acc_no = 1001;
Query OK, 1 row affected (0.73 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> select * from mini_statement;
+--------+---------------+
| acc_no | avail_balance |
+--------+---------------+
|   1000 |          7000 |
|   1001 |         12000 |
+--------+---------------+
2 rows in set (0.00 sec)
```

After Update Trigger:

As the name implies, this trigger is invoked after an updation occurs. (i.e., it gets implemented after an update statement is executed.).

Example:

We create another table:

Table 3:

```
create table micro_statement (acc_no integer, avail_balance decimal,
        foreign key(acc_no) references customer(acc_no) on delete cascade);



insert into customer values (1002, "Janitor", 4500);

drop trigger update_after;
delimiter //
create trigger update_after
      after update on customer
      for each row
      begin
      insert into micro_statement values(new.acc_no, new.avail_balance);
      end; //

delimiter ;

update customer set avail_balance = avail_balance + 1500 where acc_no = 1002;



mysql> update customer set avail_balance = avail_balance + 1500 where acc_no = 1002;

Query OK, 1 row affected (0.49 sec)

Rows matched: 1  Changed: 1  Warnings: 0



mysql> select * from customer;

+--------+-----------+---------------+
| acc_no | cust_name | avail_balance |
+--------+-----------+---------------+
|   1000 | Fanny     |          9000 |
|   1001 | Peter     |         13000 |
|   1002 | Janitor   |          6000 |
+--------+-----------+---------------+
3 rows in set (0.00 sec)

mysql> select * from mini_statement;
+--------+---------------+
| acc_no | avail_balance |
+--------+---------------+
|   1000 |          7000 |
|   1001 |         12000 |
|   1002 |          4500 |
+--------+---------------+
3 rows in set (0.00 sec)

mysql> select * from micro_statement;
+--------+---------------+
| acc_no | avail_balance |
+--------+---------------+
|   1002 |          6000 |
+--------+---------------+
1 row in set (0.00 sec)
```

Before Delete Trigger:

Table 1:

create table contacts (contact_id int (11) NOT NULL AUTO_INCREMENT, last_name VARCHAR (30) NOT NULL, first_name VARCHAR (25),birthday DATE, created_date DATE, created_by VARCHAR(30),CONSTRAINT contacts_pk PRIMARY KEY (contact_id));

Table 2:

create table contacts_audit (contact_id integer, deleted_date date, deleted_by varchar(20));

Trigger to insert contact_id and contact deletion-date/user information into contacts_audit record before a delete occurs:

Trigger :

```
delimiter //

create trigger contacts_before_delete
        before delete
        on contacts for each row
        begin

          INSERT into contacts_audit( contact_id,deleted_date,deleted_by)
          VALUES ( OLD.contact_id,SYSDATE(),USER() );
         end; //

delimiter ;

insert into contacts values (1, "Bond", "Ruskin", str_to_date ("19-08-1995", "%d-%m-%Y"),

                              str_to_date ("27-04-2018", "%d-%m-%Y"), "xyz");

delete from contacts where last_name="Bond";

mysql> select * from contacts;

+------------+-----------+------------+------------+--------------+----------------+

| contact_id | last_name | first_name | birthday   | created_date | created_by     |

+------------+-----------+------------+------------+--------------+----------------+

|          1 | Bond      | Ruskin     | 1995-08-19 | 2021-03-12   | root@localhost |

+------------+-----------+------------+------------+--------------+----------------+

1 row in set (0.00 sec)
```

```
mysql> select * from contacts_audit;

Empty set (0.00 sec)



mysql> delete from contacts where last_name="Bond";

Query OK, 1 row affected (0.05 sec)



mysql> select * from contacts_audit;

+------------+--------------+----------------+

| contact_id | deleted_date | deleted_by     |

+------------+--------------+----------------+

|          1 | 2021-03-12   | root@localhost |

+------------+--------------+----------------+

1 row in set (0.00 sec)
```

After Delete Trigger:

```
DROP TRIGGER IF EXISTS After_workcenters_insert;
DELIMITER $$

CREATE TRIGGER After_workcenters_insert
AFTER DELETE
ON WorkCenters FOR EACH ROW
BEGIN
   DECLARE cap INT;

   SET cap = old.capacity;

   UPDATE WorkCenterStats SET totalCapacity = totalCapacity-cap;

END $$

DELIMITER ;
```

Contents of tables Before Activation of Trigger

```
mysql> select * from WorkCenters;

+----+--------------+----------+

| id | name         | capacity |

+----+--------------+----------+

|  4 | Mold Machine |      100 |

|  5 | Packing      |      200 |

+----+--------------+----------+

mysql> select * from WorkCenterStats;

+---------------+

| totalCapacity |

+---------------+

|           300 |

+---------------+

mysql> delete from WorkCenters where id=4;

Query OK, 1 row affected (0.05 sec)


mysql> select * from WorkCenters;

+----+---------+----------+

| id | name    | capacity |

+----+---------+----------+

|  5 | Packing |      200 |

+----+---------+----------+

1 row in set (0.00 sec)
```

```
mysql> select * from WorkCenterStats;
+---------------+
| totalCapacity |
+---------------+
|           200 |
+---------------+
mysql> delete from WorkCenters where id=5;
Query OK, 1 row affected (0.45 sec)

mysql> select * from WorkCenterStats;
+---------------+
| totalCapacity |
+---------------+
|             0 |
+---------------+
1 row in set (0.00 sec)

mysql> select * from WorkCenters;
Empty set (0.00 sec)
```

MySQL cursor

A cursor allows you to iterate a set of rows returned by a query and process each row individually.

To handle a result set inside a sored procedurest

1) declare a cursor by using the DECLARE statement:

```
DECLARE cursor_name CURSOR FOR SELECT_statement;
```

The cursor declaration must be after any variable declaration. If you declare a cursor before the variable declarations, MySQL will issue an error.

A cursor must always associate with a SELECT statement.

2) Open the cursor by using the OPEN statement.

The OPEN statement initializes the result set for the cursor, therefore, you must call the OPEN statement before fetching rows from the result set.

OPEN cursor_name;

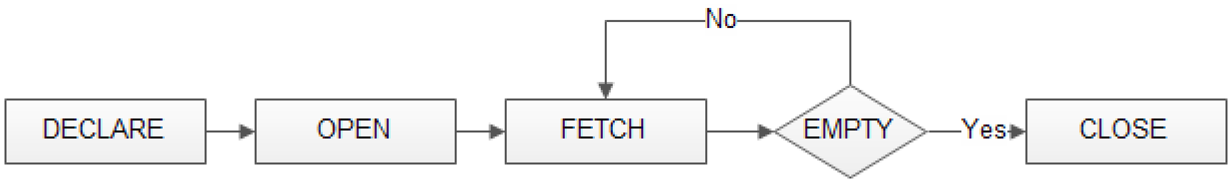
3) Use the FETCH statement to retrieve the next row pointed by the cursor and move the cursor to the next row in the result set.

FETCH cursor_name INTO variables list;

4) deactivate the cursor and release the memory associated with it using the CLOSE statement:

CLOSE cursor_name;

The following diagram illustrates how MySQL cursor works.



When the cursor reaches the end of the result set, it will not be able to get the data, and a condition is raised. The handler is used to handle this condition.

DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;

Example:

mysql> create table emp(id int,name varchar(50),dept varchar(10),phone int,emailvarchar(20));

mysql> insert into emp values(101,"Arun","IT",12345,"Arun@nitk.edu.in");

mysql> insert into emp values(102,"Anu","CSE",23456,"Anu@nitt.edu.in");

mysql> insert into emp values(103,"Bala","IT",34567,"Bala@nitt.edu.in");

mysql> insert into emp values(104,"Hari","IT",45678,"Hari@nitk.edu.in");

mysql> insert into emp values(105,"Suresh","CSE",56789,"Sureh@nitk.edu.in");

insert into emp values(101,"Arun","IT",12345,"Arun@nitk.edu.in");

CURSOR :

```
DROP PROCEDURE createEmailList;
DELIMITER $$
CREATE PROCEDURE createEmailList (INOUT emailList varchar(4000))
BEGIN
	DECLARE finished INTEGER DEFAULT 0;
	DECLARE emailAddress varchar(100) DEFAULT "";


	DEClARE curEmail CURSOR FOR SELECT email FROM emp;

	DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;

	OPEN curEmail;

	getEmail: LOOP
		FETCH curEmail INTO emailAddress;
		IF finished = 1 THEN
			LEAVE getEmail;
		END IF;
		SET emailList = CONCAT(emailAddress,";",emailList);
	END LOOP getEmail;


	    CLOSE curEmail;

END$$
DELIMITER ;

mysql> set @emaillist="";

Query OK, 0 rows affected (0.00 sec)


mysql> call createEmailList(@emaillist);

Query OK, 0 rows affected (0.01 sec)


mysql> select @emaillist;

+------------------------------------------------------------------------------------+

| @emaillist                                                                         |

+------------------------------------------------------------------------------------+

| Sureh@nitk.edu.in;Hari@nitk.edu.in;Bala@nitt.edu.in;Anu@nitt.edu.in;Arun@nitk.edu.in; |

+------------------------------------------------------------------------------------+

1 row in set (0.00 sec)
```