# **MySQL BEFORE INSERT TRIGGER**

Before Insert Trigger in MySQL is invoked automatically whenever an insert operation is executed. In this article, we are going to learn how to create a before insert trigger with its syntax and example.

### **Syntax**

The following is the syntax to create a BEFORE INSERT [trigger in MySQL](https://www.javatpoint.com/mysql-trigger):

1. **CREATE** **TRIGGER** trigger\_name
2. BEFORE **INSERT**
3. **ON** table\_name **FOR** EACH ROW
4. Trigger\_body ;

The BEFORE INSERT trigger syntax parameter can be explained as below:

* First, we will specify the **name of the trigger** that we want to create. It should be unique within the schema.
* Second, we will specify the **trigger action time**, which should be BEFORE INSERT. This trigger will be invoked before each row modifications occur on the table.
* Third, we will specify the **name of a table** to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.
* Finally, we will specify the statement for execution when the trigger is activated.

If we want to execute multiple statements, we will use the **BEGIN END** block that contains a set of queries to define the logic for the trigger. See the below syntax:

1. DELIMITER $$
2. **CREATE** **TRIGGER** trigger\_name BEFORE **INSERT**
3. **ON** table\_name **FOR** EACH ROW
4. **BEGIN**
5. variable declarations
6. **trigger** code
7. **END**$$
8. DELIMITER ;

### **Restrictions**

* We can access and change the **NEW** values only in a BEFORE INSERT trigger.
* We cannot access the **OLD** If we try to access the OLD values, we will get an error because OLD values do not exist.
* We cannot create a BEFORE INSERT trigger on a **VIEW**.

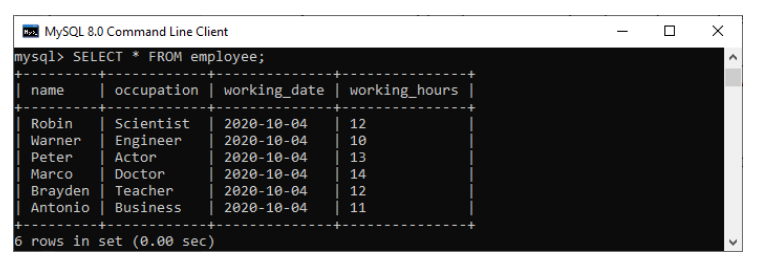
### **BEFORE INSERT Trigger Example**

Let us understand how to create a BEFORE INSERT trigger using the [CREATE TRIGGER statement](https://www.javatpoint.com/mysql-create-trigger) in [MySQL](https://www.javatpoint.com/mysql-tutorial) with an example.

Suppose we have created a table named **employee** as follows:

1. **CREATE** **TABLE** employee(
2. **name** **varchar**(45) NOT NULL,
3. occupation **varchar**(35) NOT NULL,
4. working\_date **date**,
5. working\_hours **varchar**(10)
6. );

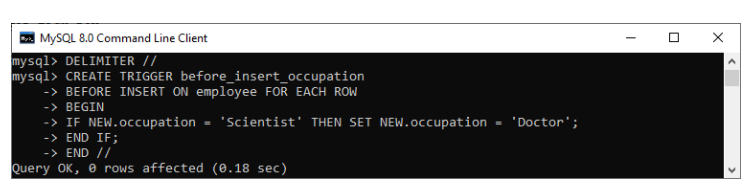
Next, we will insert some records into the employee table and then execute the [SELECT statement](https://www.javatpoint.com/mysql-select) to see the table data as follows:



Next, we will use a **CREATE TRIGGER** statement to create a BEFORE INSERT trigger. This trigger is invoked automatically that inserts the **occupation = 'Leader'** if someone tries to insert the **occupation = 'Scientist'**.

1. mysql> DELIMITER //
2. mysql> **Create** **Trigger** before\_insert\_occupation
3. BEFORE **INSERT** **ON** employee **FOR** EACH ROW
4. **BEGIN**
5. IF NEW.occupation = 'Scientist' **THEN** **SET** NEW.occupation = 'Doctor';
6. **END** IF;
7. **END** //

If the trigger is created successfully, we will get the output as follows:

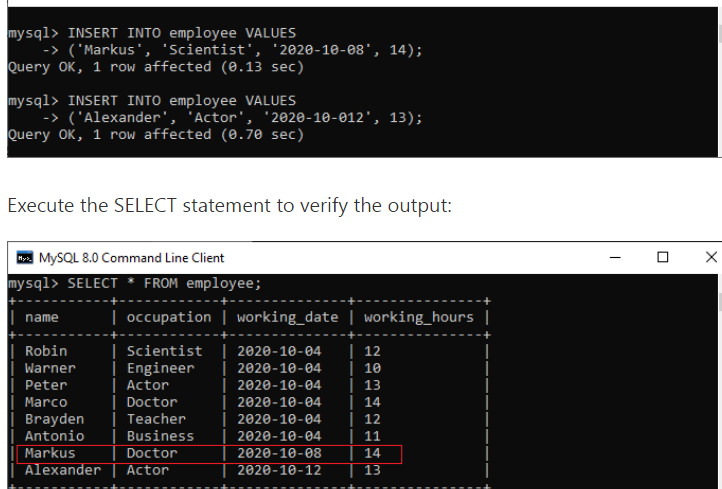


### **How to call the BEFORE INSERT trigger?**

We can use the following statements to invoke the above-created trigger:

1. mysql> **INSERT** **INTO** employee **VALUES**
2. ('Markus', 'Scientist', '2020-10-08', 14);
4. mysql> **INSERT** **INTO** employee **VALUES**
5. ('Alexander', 'Actor', '2020-10-012', 13);

After execution of the above statement, we will get the output as follows:



In this output, we can see that on inserting the occupation column values as 'Scientist', the table will automatically fill the 'Doctor' value by invoking a trigger.

# **MySQL AFTER INSERT Trigger**

After Insert Trigger in MySQL is invoked automatically whenever an insert event occurs on the table. In this article, we are going to learn how to create an after insert trigger with its syntax and example.

### **Syntax**

The following is the syntax to create an **AFTER INSERT** [trigger in MySQL](https://www.javatpoint.com/mysql-trigger):

1. **CREATE** **TRIGGER** trigger\_name
2. **AFTER** **INSERT**
3. **ON** table\_name **FOR** EACH ROW
4. trigger\_body ;

The AFTER INSERT trigger syntax parameter can be explained as below:

* First, we will specify the **name of the trigger** that we want to create. It should be unique within the schema.
* Second, we will specify the **trigger action time**, which should be AFTER INSERT clause to invoke the trigger.
* Third, we will specify the **name of a table** to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.
* Finally, we will specify the **trigger body** that contains one or more statements for execution when the trigger is activated.

If we want to execute multiple statements, we will use the **BEGIN END** block that contains a set of SQL queries to define the logic for the trigger. See the below syntax:

1. DELIMITER $$
2. **CREATE** **TRIGGER** trigger\_name **AFTER** **INSERT**
3. **ON** table\_name **FOR** EACH ROW
4. **BEGIN**
5. variable declarations
6. **trigger** code
7. **END**$$
8. DELIMITER ;

### **Restrictions**

* We can access the **NEW** values but **cannot change them** in an AFTER INSERT trigger.
* We cannot access the **OLD** If we try to access the OLD values, we will get an error because there is no OLD on the INSERT trigger.
* We cannot create the AFTER INSERT trigger on a **VIEW**.

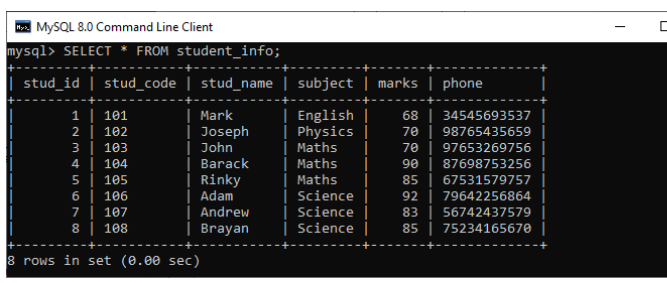
### **AFTER INSERT Trigger Example**

Let us understand how to create an AFTER INSERT trigger using the [**CREATE TRIGGER**](https://www.javatpoint.com/mysql-create-trigger) statement in [MySQL](https://www.javatpoint.com/mysql-tutorial) with an example.

Suppose we have created a table named "**student\_info**" as follows:

1. **CREATE** **TABLE** student\_info (
2. stud\_id **int** NOT NULL,
3. stud\_code **varchar**(15) **DEFAULT** NULL,
4. stud\_name **varchar**(35) **DEFAULT** NULL,
5. subject **varchar**(25) **DEFAULT** NULL,
6. marks **int** **DEFAULT** NULL,
7. phone **varchar**(15) **DEFAULT** NULL,
8. **PRIMARY** **KEY** (stud\_id)
9. )

Next, we will insert some records into this table and then execute the [SELECT statement](https://www.javatpoint.com/mysql-select) to see the table data as follows:



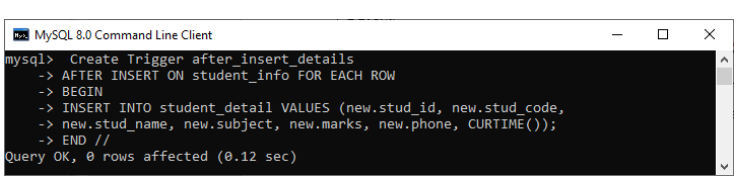
Again, we will create a new table named **"student\_detail"** as follows:

1. **CREATE** **TABLE** student\_detail (
2. stud\_id **int** NOT NULL,
3. stud\_code **varchar**(15) **DEFAULT** NULL,
4. stud\_name **varchar**(35) **DEFAULT** NULL,
5. subject **varchar**(25) **DEFAULT** NULL,
6. marks **int** **DEFAULT** NULL,
7. phone **varchar**(15) **DEFAULT** NULL,
8. Lasinserted **Time**,
9. **PRIMARY** **KEY** (stud\_id)
10. );

Next, we will use a CREATE TRIGGER statement to create an **after\_insert\_details** trigger on the **student\_info** table. This trigger will be fired after an insert operation is performed on the table.

1. mysql> DELIMITER //
2. mysql> **Create** **Trigger** after\_insert\_details
3. **AFTER** **INSERT** **ON** student\_info **FOR** EACH ROW
4. **BEGIN**
5. **INSERT** **INTO** student\_detail **VALUES** (new.stud\_id, new.stud\_code,
6. new.stud\_name, new.subject, new.marks, new.phone, CURTIME());
7. **END** //

If the trigger is created successfully, we will get the output as follows:



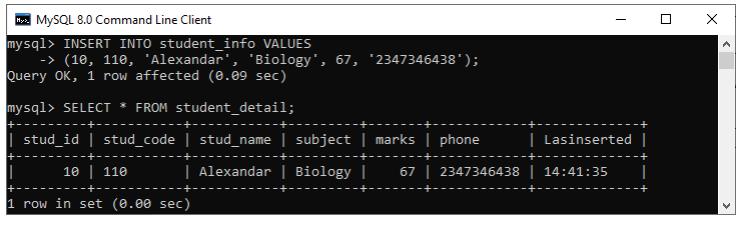
### **How to call the AFTER INSERT trigger?**

We can use the following statements to invoke the above-created trigger:

1. mysql> **INSERT** **INTO** student\_info **VALUES**
2. (10, 110, 'Alexandar', 'Biology', 67, '2347346438');

The table that has been modified after the update query executes is student\_detail. We can verify it by using the SELECT statement as follows:

1. mysql> **SELECT** \* **FROM** student\_detail;



In this output, we can see that on inserting values into the student\_info table, the student\_detail table will automatically fill the records by invoking a trigger.

MySQL BEFORE UPDATE Trigger

BEFORE UPDATE Trigger in MySQL is invoked automatically whenever an update operation is fired on the table associated with the trigger. In this article, we are going to learn how to create a before update trigger with its syntax and example.

Syntax

The following is the syntax to create a BEFORE UPDATE trigger in MySQL:

1. **CREATE** **TRIGGER** trigger\_name
2. BEFORE **UPDATE**
3. **ON** table\_name **FOR** EACH ROW
4. trigger\_body ;

The BEFORE UPDATE trigger syntax parameter are explained as below:

* First, we will specify the **trigger name** that we want to create. It should be unique within the schema.
* Second, we will specify the **trigger action time**, which should be BEFORE UPDATE. This trigger will be invoked before each row of alterations occurs on the table.
* Third, we will specify the name of a table to which the trigger is associated. It must be written after the **ON keyword**. If we did not specify the table name, a trigger would not exist.
* Finally, we will specify the **trigger body** that contains a statement for execution when the trigger is activated.

If we want to execute multiple statements, we will use the **BEGIN END** block that contains a set of queries to define the logic for the trigger

1. DELIMITER $$
2. **CREATE** **TRIGGER** trigger\_name BEFORE **UPDATE**
3. **ON** table\_name **FOR** EACH ROW
4. **BEGIN**
5. variable declarations
6. **trigger** code
7. **END**$$
8. DELIMITER ;

Restrictions

* We cannot update the OLD values in a BEFORE UPDATE trigger.
* We can change the NEW values.
* We cannot create a BEFORE UPDATE trigger on a VIEW.

BEFORE UPDATE Trigger Example

Let us understand how to create a BEFORE UPDATE trigger using the [CREATE TRIGGER statement in MySQL](https://www.javatpoint.com/mysql-create-trigger) with an example.

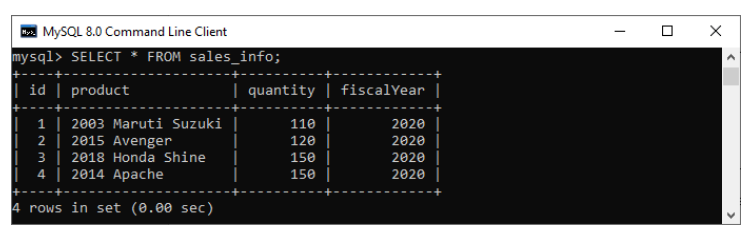
Suppose we have created a table named **sales\_info** as follows:

1. **CREATE** **TABLE** sales\_info (
2. id **INT** AUTO\_INCREMENT,
3. product **VARCHAR**(100) NOT NULL,
4. quantity **INT** NOT NULL **DEFAULT** 0,
5. **CHECK** (quantity >=0),
6. **UNIQUE**(product, fiscalYear),
7. **PRIMARY** **KEY**(id)
8. );

Next, we will insert some records into the sales\_info table as follows:

1. **INSERT** **INTO** sales\_info(product, quantity, fiscalYear)
2. **VALUES**
3. ('2003 Maruti Suzuki',110),
4. ('2015 Avenger', 120),
5. ('2018 Honda Shine', 150),
6. ('2014 Apache', 150);

Then, execute the **SELECT statement** to see the table data as follows:



Next, we will use a **CREATE TRIGGER** statement to create a BEFORE UPDATE trigger. This trigger is invoked automatically before an update event occurs in the table.

1. DELIMITER $$
3. **CREATE** **TRIGGER** before\_update\_salesInfo
4. BEFORE **UPDATE**
5. **ON** sales\_info **FOR** EACH ROW
6. **BEGIN**
7. **DECLARE** error\_msg **VARCHAR**(255);
8. **SET** error\_msg = ('The new quantity cannot be greater than 2 times the current quantity');
9. IF new.quantity > old.quantity \* 2 **THEN**
10. SIGNAL SQLSTATE '45000'
11. **SET** MESSAGE\_TEXT = error\_msg;
12. **END** IF;
13. **END** $$
15. DELIMITER ;

The trigger produces an error message and stops the updation if we update the value in the quantity column to a new value two times greater than the current value.

Let us understand the created trigger in details:

First, we have specified the trigger name as befor\_update\_salesInfo in the CREATE TRIGGER clause. Second, specify the triggering event and then the table name on which the trigger is associated. Third, we have declared a variable and set its value. Finally, we have specified the trigger body that checks if the new value is two times greater than the old value and then raises an error.

### **How to call the BEFORE UPDATE trigger?**

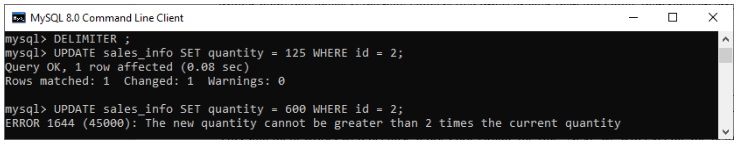
First, we can use the following statements that update the quantity of the row whose id = 2:

1. mysql> **UPDATE** sales\_info **SET** quantity = 125 **WHERE** id = 2;

This statement works well because it does not violate the rule. Next, we will execute the below statements that update the quantity of the row as 600 whose id = 2

1. mysql> **UPDATE** sales\_info **SET** quantity = 600 **WHERE** id = 2;

It will give the error as follows because it violates the rule. See the below output.



# **MySQL AFTER UPDATE TRIGGER**

The AFTER UPDATE trigger in MySQL is invoked automatically whenever an UPDATE event is fired on the table associated with the triggers. In this article, we are going to learn how to create an AFTER UPDATE trigger with its syntax and example.

### **Syntax**

The following is the syntax to create an **AFTER UPDATE** trigger in MySQL:

1. **CREATE** **TRIGGER** trigger\_name
2. **AFTER** **UPDATE**
3. **ON** table\_name **FOR** EACH ROW
4. trigger\_body ;

We can explain the parameters of AFTER UPDATE trigger syntax as below:

* First, we will specify the **trigger name** that we want to create. It should be unique within the schema.
* Second, we will specify the **trigger action time**, which should be AFTER UPDATE. This trigger will be invoked after each row of alterations occurs on the table.
* Third, we will specify the **table name** to which the trigger is associated. It must be written after the **ON** If we did not specify the table name, a trigger would not exist.
* Finally, we will specify the **trigger body** that contains a statement for execution when the trigger is activated.

If we want to execute more than one statement, we will use the **BEGIN END** block that contains a set of SQL queries to define the logic for the [trigger](https://www.javatpoint.com/mysql-trigger). See the below syntax:

1. DELIMITER $$
2. **CREATE** **TRIGGER** trigger\_name **AFTER** **UPDATE**
3. **ON** table\_name **FOR** EACH ROW
4. **BEGIN**
5. variable declarations
6. **trigger** code
7. **END**$$
8. DELIMITER ;

### **Restrictions**

* We can access the OLD rows but cannot update them.
* We can access the NEW rows but cannot update them.
* We cannot create an AFTER UPDATE trigger on a **VIEW**.

### **AFTER UPDATE Trigger Example**

Let us understand how to create an AFTER UPDATE trigger using the [CREATE TRIGGER statement in MySQL](https://www.javatpoint.com/mysql-create-trigger) with an example.

Suppose we have created a table named **students** to store the student's information as follows:

1. mysql> **CREATE** **TABLE** students(
2. id **int** NOT NULL AUTO\_INCREMENT,
3. **name** **varchar**(45) NOT NULL,
4. class **int** NOT NULL,
5. email\_id **varchar**(65) NOT NULL,
6. **PRIMARY** **KEY** (id)
7. );

Next, we will insert some records into this table using the below statement:

1. **INSERT** **INTO** students (**name**, class, email\_id)
2. **VALUES** ('Stephen', 6, 'stephen@javatpoint.com'),
3. ('Bob', 7, 'bob@javatpoint.com'),
4. ('Steven', 8, 'steven@javatpoint.com'),
5. ('Alexandar', 7, 'alexandar@javatpoint.com');

Execute the [**SELECT**](https://www.javatpoint.com/mysql-select) query to see the table data.



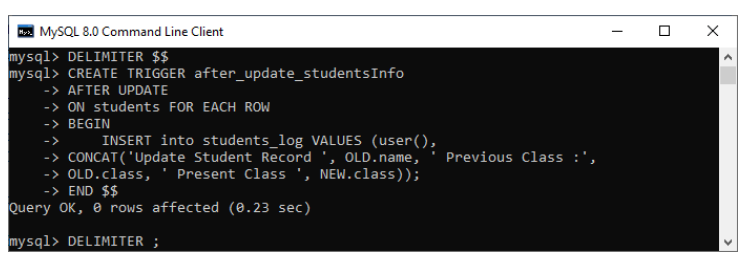
Third, we will create another table named **students\_log** that keeps the updated information in the selected user.

1. mysql> **CREATE** **TABLE** students\_log(
2. user **varchar**(45) NOT NULL,
3. descreptions **varchar**(65) NOT NULL
4. );

We will then create an AFTER UPDATE **trigger that promotes all students in the next class**, i.e., 6 will be 7, 7 will be 8, and so on. Whenever an updation is performed on a single row in the "**students**" table, a new row will be inserted in the **"students\_log**" table. This table keeps the **current user id** and a **description** regarding the current update. See the below trigger code.

ADVERTISEMENT

1. DELIMITER $$
3. **CREATE** **TRIGGER** after\_update\_studentsInfo
4. **AFTER** **UPDATE**
5. **ON** students **FOR** EACH ROW
6. **BEGIN**
7. **INSERT** **into** students\_log **VALUES** (user(),
8. CONCAT('Update Student Record ', OLD.**name**, ' Previous Class :',
9. OLD.class, ' Present Class ', NEW.class));
10. **END** $$
12. DELIMITER ;



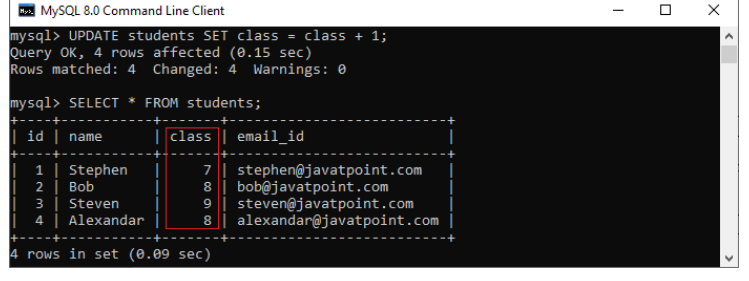
n this trigger, we have first specified the trigger name **after\_update\_studentsInfo**. Then, specify the triggering event. Third, we have specified the table name on which the trigger is associated. Finally, we have written the trigger logic inside the trigger body that performs updation in the "students" table and keeps the log information in the "students\_log" table.

### **How to call the AFTER UPDATE trigger?**

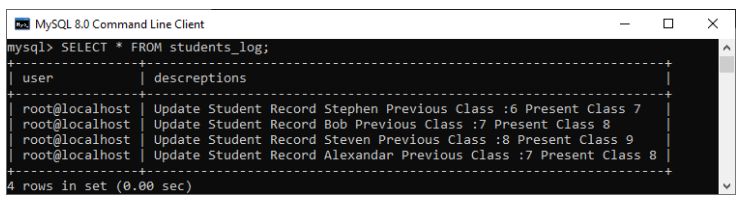
First, we will update the "students" table using the following statements that invoke the above-created trigger:

1. mysql> **UPDATE** students **SET** class = class + 1;

Next, we will query data from the **students** and **students\_log table**. We can see that table has been updated after the execution of the query. See the below output:



Again, we will query data from the students\_log table that keeps the current user id and a description regarding the current update. See the below output:



# **MySQL BEFORE DELETE Trigger**

BEFORE DELETE Trigger in MySQL is invoked automatically whenever a delete operation is fired on the table. In this article, we are going to learn how to create a before delete trigger with its syntax and example.

### **Syntax**

The following is the syntax to create a BEFORE DELETE trigger in MySQL:

1. **CREATE** **TRIGGER** trigger\_name
2. BEFORE **DELETE**
3. **ON** table\_name **FOR** EACH ROW
4. Trigger\_body ;

The BEFORE DELETE trigger syntax parameter can be explained as below:

* First, we will specify the name of the trigger that we want to create. It should be unique within the schema.
* Second, we will specify the trigger action time, which should be BEFORE DELETE. This trigger will be invoked before each row of alterations occurs on the table.
* Third, we will specify the name of a table to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.
* Finally, we will specify the statement for execution when the trigger is activated.

If we want to execute multiple statements, we will use the BEGIN END block that contains a set of queries to define the logic for the trigger. See the below syntax:

1. DELIMITER $$
2. **CREATE** **TRIGGER** trigger\_name BEFORE **DELETE**
3. **ON** table\_name **FOR** EACH ROW
4. **BEGIN**
5. variable declarations
6. **trigger** code
7. **END**$$
8. DELIMITER ;

### **Restrictions**

* We can access the OLD rows but cannot update them in a BEFORE DELETE trigger.
* We cannot access the NEW rows. It is because there are no new row exists.
* We cannot create a BEFORE DELETE trigger on a VIEW.

### **BEFORE DELETE Trigger Example**

Let us understand how to create a BEFORE DELETE trigger using the [CREATE TRIGGER statement in MySQL](https://www.javatpoint.com/mysql-create-trigger) with an example.

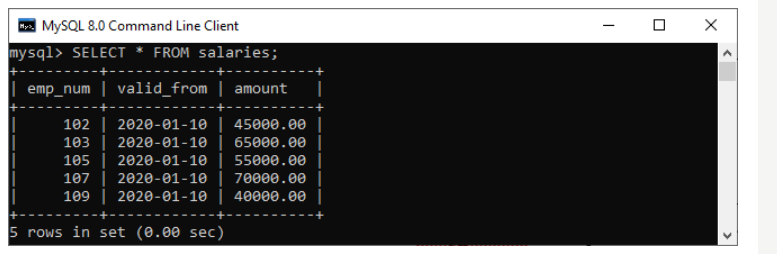
Suppose we have created a table named salaries to store the salary information of an employee as follows:

1. **CREATE** **TABLE** salaries (
2. emp\_num **INT** **PRIMARY** **KEY**,
3. valid\_from **DATE** NOT NULL,
4. amount **DEC**(8 , 2 ) NOT NULL **DEFAULT** 0
5. );

Next, we will insert some records into this table using the below statement:

1. **INSERT** **INTO** salaries (emp\_num, valid\_from, amount)
2. **VALUES**
3. (102, '2020-01-10', 45000),
4. (103, '2020-01-10', 65000),
5. (105, '2020-01-10', 55000),
6. (107, '2020-01-10', 70000),
7. (109, '2020-01-10', 40000);

Execute the SELECT query to see the table data.



Third, we will create another table named salary\_archives that keeps the information of deleted salary.

1. **CREATE** **TABLE** salary\_archives (
2. id **INT** **PRIMARY** **KEY** AUTO\_INCREMENT,
3. emp\_num **INT**,
4. valid\_from **DATE** NOT NULL,
5. amount **DEC**(18 , 2 ) NOT NULL **DEFAULT** 0,
6. deleted\_time **TIMESTAMP** **DEFAULT** NOW()
7. );

We will then create a BEFORE DELETE trigger that inserts a new record into the salary\_archives table before a row is deleted from the salaries table.

1. DELIMITER $$
3. **CREATE** **TRIGGER** before\_delete\_salaries
4. BEFORE **DELETE**
5. **ON** salaries **FOR** EACH ROW
6. **BEGIN**
7. **INSERT** **INTO** salary\_archives (emp\_num, valid\_from, amount)
8. **VALUES**(OLD. emp\_num, OLD.valid\_from, OLD.amount);
9. **END**$$
11. DELIMITER ;

In this trigger, we have first specified the trigger name before\_delete\_salaries. Then, specify the triggering event. Third, we have specified the table name on which the trigger is associated. Finally, we have written the trigger logic inside the trigger body that insert the deleted row into the salary\_archives table.

### **How to call the BEFORE DELETE trigger?**

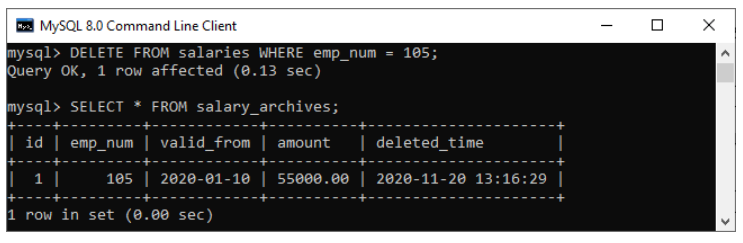
Let us test the above created BEFORE DELETE trigger and how we can call them. So first, we will remove a row from the salaries table:

1. mysql> **DELETE** **FROM** salaries **WHERE** emp\_num = 105;

Second, we will query data from the salary\_archives table to verify the above-created trigger is invoked or not by using the select statement:

1. mysql> **SELECT** \* **FROM** salary\_archives;

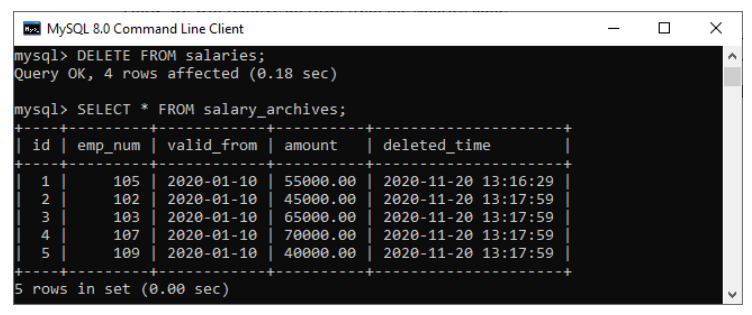
After executing a statement, we can see that the trigger was invoked successfully and inserted a new record into the salary\_archives table.



Third, we will remove all rows from the salaries table:

1. mysql> **DELETE** **FROM** salaries;

Finally, we will query data from the salary\_archives table again. The trigger was called four times because the DELETE statement removed four records from the salaries table. See the below output:



# **MySQL AFTER DELETE Trigger**

The AFTER DELETE Trigger in MySQL is invoked automatically whenever a delete event is fired on the table. In this article, we are going to learn how to create an AFTER DELETE trigger with its syntax and example.

### **Syntax**

The following is the syntax to create an **AFTER DELETE** trigger in MySQL:

1. **CREATE** **TRIGGER** trigger\_name
2. **AFTER** **DELETE**
3. **ON** table\_name **FOR** EACH ROW
4. Trigger\_body ;

The AFTER DELETE trigger syntax parameter can be explained as below:

* First, we will specify the **name of the trigger** that we want to create. It should be unique within the schema.
* Second, we will specify the **trigger action time**, which should be AFTER DELETE. This trigger will be invoked after each row of alterations occurs on the table.
* Third, we will specify the **name of a table** to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.
* Finally, we will specify the **trigger body** that contains a statement for execution when the trigger is activated.

If we want to execute multiple statements, we will use the **BEGIN END** block that contains a set of SQL queries to define the logic for the trigger. See the below syntax:

1. DELIMITER $$
2. **CREATE** **TRIGGER** trigger\_name **AFTER** **DELETE**
3. **ON** table\_name **FOR** EACH ROW
4. **BEGIN**
5. variable declarations
6. **trigger** code
7. **END**$$
8. DELIMITER ;

### **Restrictions**

* We can access the OLD rows but cannot update them in the AFTER DELETE trigger.
* We cannot access the NEW rows. It is because there are no NEW row exists.
* We cannot create an AFTER DELETE trigger on a VIEW.

### **AFTER DELETE Trigger Example**

Let us understand how to create an AFTER DELETE trigger using the [CREATE TRIGGER statement in MySQL](https://www.javatpoint.com/mysql-create-trigger) with an example.

Suppose we have created a table named **salaries** to store the salary information of an employee as follows:

1. **CREATE** **TABLE** salaries (
2. emp\_num **INT** **PRIMARY** **KEY**,
3. valid\_from **DATE** NOT NULL,
4. amount **DEC**(8 , 2 ) NOT NULL **DEFAULT** 0
5. );
6. The AFTER DELETE trigger syntax parameter can be explained as below:

* First, we will specify the **name of the trigger** that we want to create. It should be unique within the schema.
* Second, we will specify the **trigger action time**, which should be AFTER DELETE. This trigger will be invoked after each row of alterations occurs on the table.
* Third, we will specify the **name of a table** to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.
* Finally, we will specify the **trigger body** that contains a statement for execution when the trigger is activated.

If we want to execute multiple statements, we will use the **BEGIN END** block that contains a set of SQL queries to define the logic for the trigger. See the below syntax:

1. DELIMITER $$
2. **CREATE** **TRIGGER** trigger\_name **AFTER** **DELETE**
3. **ON** table\_name **FOR** EACH ROW
4. **BEGIN**
5. variable declarations
6. **trigger** code
7. **END**$$
8. DELIMITER ;

### **Restrictions**

* We can access the OLD rows but cannot update them in the AFTER DELETE trigger.
* We cannot access the NEW rows. It is because there are no NEW row exists.
* We cannot create an AFTER DELETE trigger on a VIEW.

### **AFTER DELETE Trigger Example**

Let us understand how to create an AFTER DELETE trigger using the [CREATE TRIGGER statement in MySQL](https://www.javatpoint.com/mysql-create-trigger) with an example.

Suppose we have created a table named **salaries** to store the salary information of an employee as follows:

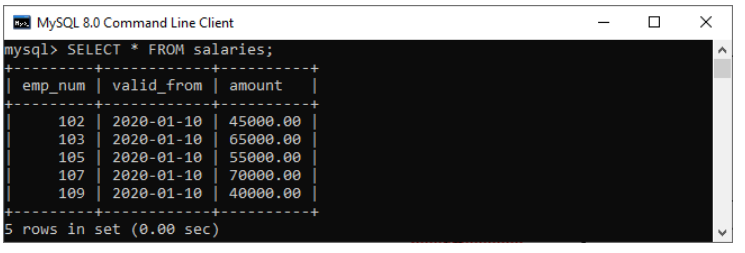
1. **CREATE** **TABLE** salaries (
2. emp\_num **INT** **PRIMARY** **KEY**,
3. valid\_from **DATE** NOT NULL,
4. amount **DEC**(8 , 2 ) NOT NULL **DEFAULT** 0
5. );

Next, we will insert some records into this table using the below statement:

Next, we will insert some records into this table using the below statement:

1. **INSERT** **INTO** salaries (emp\_num, valid\_from, amount)
2. **VALUES**
3. (102, '2020-01-10', 45000),
4. (103, '2020-01-10', 65000),
5. (105, '2020-01-10', 55000),
6. (107, '2020-01-10', 70000),
7. (109, '2020-01-10', 40000);

Execute the SELECT query to see the table data.



Third, we will create another table named **total\_salary\_budget** that keeps the salary information from the salaries table.

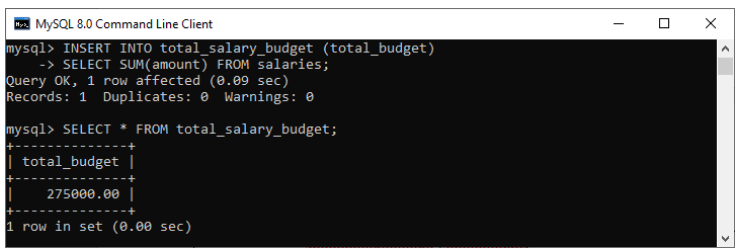
1. **CREATE** **TABLE** total\_salary\_budget(
2. total\_budget **DECIMAL**(10,2) NOT NULL
3. );

Fourth, we will use the **SUM()** function that returns the total salary from the salaries table and keep this information in the total\_salary\_budget table:

ADVERTISEMENT

1. mysql> **INSERT** **INTO** total\_salary\_budget (total\_budget)
2. **SELECT** SUM(amount) **FROM** salaries;

Execute the SELECT statement to verify the table:



We will then create an AFTER DELETE trigger that updates the total salary into the total\_salary\_budget table after a row is deleted from the salaries table.

1. DELIMITER $$
3. **CREATE** **TRIGGER** after\_delete\_salaries
4. **AFTER** **DELETE**
5. **ON** salaries **FOR** EACH ROW
6. **BEGIN**
7. **UPDATE** total\_salary\_budget **SET** total\_budget = total\_budget - old.amount;
8. **END**$$
10. DELIMITER ;

In this trigger, we have first specified the trigger name after\_delete\_salaries. Then, specify the triggering event. Third, we have specified the table name on which the trigger is associated. Finally, we have written the trigger logic inside the trigger body that updates the total salary into the total\_salary\_budget table after a row is deleted from the salaries table.