MySQL Triggers

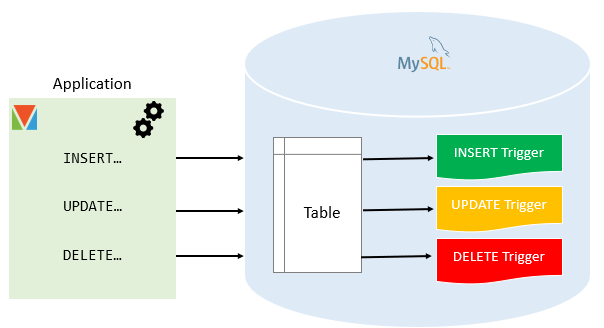
In MySQL, a trigger is a stored program invoked automatically in response to an event such as [insert](https://www.mysqltutorial.org/mysql-insert-statement.aspx), [update](https://www.mysqltutorial.org/mysql-update-data.aspx), or [delete](https://www.mysqltutorial.org/mysql-delete-statement.aspx) that 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](https://www.mysqltutorial.org/mysql-insert-statement.aspx), [UPDATE](https://www.mysqltutorial.org/mysql-update-data.aspx) or [DELETE](https://www.mysqltutorial.org/mysql-delete-statement.aspx) event.

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.



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](https://www.mysqltutorial.org/mysql-triggers/working-mysql-scheduled-event/). By using triggers, you don’t have to wait for the [scheduled events](https://www.mysqltutorial.org/mysql-triggers/working-mysql-scheduled-event/) 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](https://www.mysqltutorial.org/mysql-not-null-constraint/), [UNIQUE](https://www.mysqltutorial.org/mysql-unique-constraint/), [CHECK](https://www.mysqltutorial.org/mysql-check-constraint/) and [FOREIGN KEY](https://www.mysqltutorial.org/mysql-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.

Managing MySQL triggers

* [Create triggers](https://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx)  – describe steps of how to create a trigger in MySQL.
* [Drop triggers](https://www.mysqltutorial.org/mysql-triggers/mysql-drop-trigger/) – show you how to drop a trigger.
* [Create a BEFORE INSERT trigger](https://www.mysqltutorial.org/mysql-triggers/mysql-before-insert-trigger/) – show you how to create a BEFORE INSERT trigger to maintain a summary table from another table.
* [Create an AFTER INSERT trigger](https://www.mysqltutorial.org/mysql-triggers/mysql-after-insert-trigger/) – describe how to create an AFTER INSERT trigger to insert data into a table after inserting data into another table.
* [Create a BEFORE UPDATE trigger](https://www.mysqltutorial.org/mysql-triggers/mysql-before-update-trigger/) – learn how to create a BEFORE UPDATE trigger that validates data before it is updated to the table.
* [Create an AFTER UPDATE trigger](https://www.mysqltutorial.org/mysql-triggers/mysql-after-update-trigger/) – show you how to create an AFTER UPDATE trigger to log the changes of data in a table.
* [Create a BEFORE DELETE trigger](https://www.mysqltutorial.org/mysql-triggers/mysql-before-delete-trigger/) – show how to create a BEFORE DELETE trigger.
* [Create an AFTER DELETE trigger](https://www.mysqltutorial.org/mysql-triggers/mysql-after-delete-trigger/) – describe how to create an AFTER DELETE trigger.
* [Create multiple triggers for a table that have the same trigger event and time](https://www.mysqltutorial.org/mysql-triggers/create-multiple-triggers-for-the-same-trigger-event-and-action-time/) – MySQL 8.0 allows you to define multiple triggers for a table that have the same trigger event and time.
* [Show triggers](https://www.mysqltutorial.org/mysql-triggers/mysql-show-triggers/) – list triggers in a database, table by specific patterns.

# Create Trigger in MySQL

**Summary**: in this tutorial, you will learn how to use the MySQL CREATE TRIGGER statement to create a trigger in the database**.**

## Introduction to MySQL CREATE TRIGGER statement

The CREATE TRIGGER statement creates a new trigger. Here is the basic syntax of the CREATE TRIGGER statement:

**CREATE** **TRIGGER** trigger\_name

{**BEFORE** | **AFTER**} {**INSERT** | **UPDATE**| **DELETE** }

**ON** table\_name **FOR** **EACH** **ROW**

trigger\_body;

In this syntax:

* First, specify the name of the trigger that you want to create after the CREATE TRIGGER keywords. Note that the trigger name must be unique within a database.
* Next, specify the trigger action time which can be either BEFORE or AFTER which indicates that the trigger is invoked before or after each row is modified.
* Then, specify the operation that activates the trigger, which can be [INSERT](https://www.mysqltutorial.org/mysql-insert-statement.aspx), [UPDATE](https://www.mysqltutorial.org/mysql-update-data.aspx), or [DELETE](https://www.mysqltutorial.org/mysql-delete-statement.aspx).
* After that, specify the name of the table to which the trigger belongs after the ON keyword.
* Finally, specify the statement to execute when the trigger activates. If you want to execute multiple statements, you use the BEGIN END compound statement.

The trigger body can access the values of the column being affected by the DML statement.

To distinguish between the value of the columns BEFORE and AFTER the DML has fired, you use the NEW and OLD modifiers.

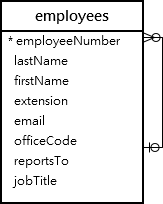
For example, if you update the column description, in the trigger body, you can access the value of the description before the update OLD.description and the new value NEW.description.

The following table illustrates the availability of the OLD and NEW modifiers:

|  |  |  |
| --- | --- | --- |
| Trigger Event | OLD | NEW |
| INSERT | No | Yes |
| UPDATE | Yes | Yes |
| DELETE | Yes | No |

## MySQL trigger examples

Let’s start creating a trigger in MySQL to log the changes of the employees table.



First, [create a new table](https://www.mysqltutorial.org/mysql-create-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.

**CREATE** **TRIGGER** before\_employee\_update

**BEFORE** **UPDATE** **ON** employees

**FOR** **EACH** **ROW**

**INSERT** **INTO** employees\_audit

**SET** **action** = 'update',

employeeNumber = OLD.employeeNumber,

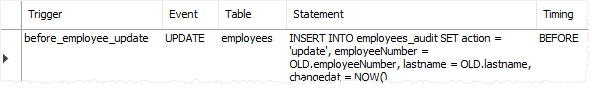
lastname = OLD.lastname,

changedat = **NOW**();

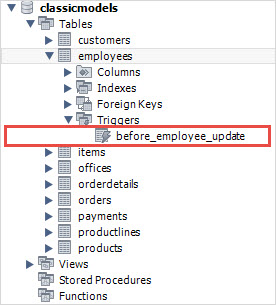
Inside the body of the trigger, we used the OLD keyword to access values of the columns employeeNumber and lastname of the row affected by the trigger.

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

**SHOW** **TRIGGERS**;



In addition, if you look at the schema using MySQL Workbench under the **employees > triggers**, you will see the before\_employee\_update trigger as shown in the screenshot below:



After that, update a row in the employees table:

**UPDATE** employees

**SET**

lastName = 'Phan'

**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:

MySQL CREATE TRIGGER example

As you see clearly from the output, the trigger was automatically invoked and inserted a new row into the employees\_audit table.

# MySQL DROP TRIGGER

**Summary**: in this tutorial, you will learn how to use the MySQL DROP TRIGGER statement to drop a trigger from the database.

## Introduction to MySQL DROP TRIGGER statement

The DROP TRIGGER statement deletes a trigger from the database.

Here is the basic syntax of the DROP TRIGGER statement:

**DROP** **TRIGGER** [**IF** **EXISTS**] [schema\_name.]trigger\_name;

In this syntax:

* First, specify the name of the trigger that you want to drop after the DROP TRIGGER keywords.
* Second, specify the name of the schema to which the trigger belongs. If you skip the schema name, the statement will drop the trigger in the current database.
* Third, use IF EXISTS option to conditionally drops the trigger if the trigger exists. The IF EXISTS clause is optional.

If you drop a trigger that does not exist without using the IF EXISTS clause, MySQL issues an error. However, if you use the IF EXISTS clause, MySQL issues a NOTE instead.

The DROP TRIGGER requires the TRIGGER privilege for the table associated with the trigger.

Note that if you [drop a table](https://www.mysqltutorial.org/mysql-drop-table), MySQL will automatically drop all triggers associated with the table.

## MySQL DROP TRIGGER example

First, [create a table](https://www.mysqltutorial.org/mysql-create-table/) called billings for demonstration:

**CREATE** **TABLE** billings (

billingNo INT AUTO\_INCREMENT,

customerNo INT,

billingDate DATE,

amount DEC(10 , 2 ),

PRIMARY **KEY** (billingNo)

);

Second, [create a new trigger](https://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx) called BEFORE UPDATE that is associated with the billings table:

DELIMITER $$

**CREATE** **TRIGGER** before\_billing\_update

**BEFORE** **UPDATE**

**ON** billings **FOR** **EACH** **ROW**

**BEGIN**

**IF** new.amount > old.amount \* 10 **THEN**

SIGNAL **SQLSTATE** '45000'

**SET** MESSAGE\_TEXT = 'New amount cannot be 10 times greater than the current amount.';

**END** **IF**;

**END**$$

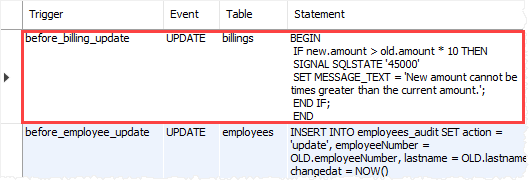
DELIMITER ;

If you are not familiar with the DELIMITER statement, check it out here in the [stored procedure statement](https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx).

The trigger activates before any update. If the new amount is 10 times greater than the current amount, the trigger raises an error.

Third, show the triggers:

**SHOW** **TRIGGERS**;

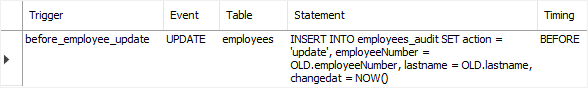


Fourth, drop the before\_billing\_update trigger:

**DROP** **TRIGGER** before\_billing\_update;

Finally, show the triggers again to verify the removal:

**SHOW** **TRIGGERS**;



# MySQL BEFORE INSERT Trigger

**Summary**: in this tutorial, you will learn how to create a MySQL BEFORE INSERT trigger to maintain a summary table of another table.

## Introduction to MySQL BEFORE INSERT triggers

MySQL BEFORE INSERT [triggers](https://www.mysqltutorial.org/mysql-triggers.aspx) are automatically fired before an [insert](https://www.mysqltutorial.org/mysql-insert-statement.aspx) event occurs on the table.

The following illustrates the basic syntax of creating a MySQL BEFORE INSERT trigger:

**CREATE** **TRIGGER** trigger\_name

**BEFORE** **INSERT**

**ON** table\_name **FOR** **EACH** **ROW**

trigger\_body;

In this syntax:

First, specify the name of the trigger that you want to create in the [CREATE TRIGGER](https://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx) clause.

Second, use BEFORE INSERT clause to specify the time to invoke the trigger.

Third, specify the name of the table that the trigger is associated with after the ON keyword.

Finally, specify the trigger body which contains one or more SQL statements that execute when the trigger is invoked.

If you have multiple statements in the trigger\_body, you have to use the BEGIN END block and change the default [delimiter](https://www.mysqltutorial.org/mysql-stored-procedure/mysql-delimiter/):

DELIMITER $$

**CREATE** **TRIGGER** trigger\_name

**BEFORE** **INSERT**

**ON** table\_name **FOR** **EACH** **ROW**

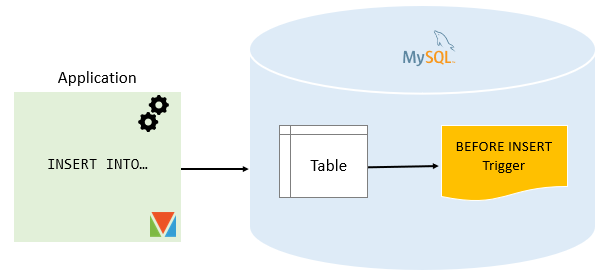
**BEGIN**

*-- statements*

**END**$$

DELIMITER ;

Note that in a BEFORE INSERT trigger, you can access and change the NEW values. However, you cannot access the OLD values because OLD values obviously do not exist.



## MySQL BEFORE INSERT trigger example

We will create a BEFORE INSERT trigger to maintain a summary table from another table.

### Setting up a sample table

First, [create a new table](https://www.mysqltutorial.org/mysql-create-table/) called WorkCenters:

**DROP** **TABLE** **IF** **EXISTS** WorkCenters;

**CREATE** **TABLE** WorkCenters (

**id** INT AUTO\_INCREMENT PRIMARY **KEY**,

**name** VARCHAR(100) **NOT** NULL,

**capacity** INT **NOT** NULL

);

Second, create another table called WorkCenterStats that stores the summary of the capacity of the work centers:

**DROP** **TABLE** **IF** **EXISTS** WorkCenterStats;

**CREATE** **TABLE** WorkCenterStats(

totalCapacity INT **NOT** NULL

);

### Creating BEFORE INSERT trigger example

The following trigger updates the total capacity in the WorkCenterStats table before a new work center is inserted into the WorkCenter table:

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 ;

In this trigger:

First, the name of the trigger is before\_workcenters\_insert specified in the CREATE TRIGGER clause:

**CREATE** **TRIGGER** before\_workcenters\_insert

Second, the triggering event is:

BEFORE **INSERT**

Third, the table that the trigger associated with is WorkCenters table:

ON WorkCenters FOR EACH ROW

Finally, inside the trigger body, we check if there is any row in the WorkCenterStats table.

If the table WorkCenterStats has a row, the trigger adds the capacity to the totalCapacity column. Otherwise, it inserts a new row into the WorkCenterStats table.

### Testing the MySQL BEFORE INSERT trigger

First, [insert a new row](https://www.mysqltutorial.org/mysql-insert-statement.aspx) into the WorkCenter table:

**INSERT** **INTO** WorkCenters(**name**, **capacity**)

**VALUES**('Mold Machine',100);

Second, [query data](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) from the WorkCenterStats table:

**SELECT** \* **FROM** WorkCenterStats;

https://sp.mysqltutorial.org/wp-content/uploads/2019/09/MySQL-BEFORE-INSERT-Trigger-Example.png

The trigger has been invoked and inserted a new row into the WorkCenterStats table.

Third, insert a new work center:

**INSERT** **INTO** WorkCenters(**name**, **capacity**)

**VALUES**('Packing',200);

Finally, query data from the WorkCenterStats:

**SELECT** \* **FROM** WorkCenterStats;

MySQL BEFORE INSERT Trigger Example 2

The trigger has updated the total capacity from 100 to 200 as expected.

Note that to properly maintain the summary table WorkCenterStats, you should also create triggers to handle update and delete events on the WorkCenters table.

# MySQL AFTER INSERT Trigger

**Summary**: in this tutorial, you will learn how to create a MySQL AFTER INSERT trigger to insert data into a table after inserting data into another table.

## Introduction to MySQL AFTER INSERT triggers

MySQL AFTER INSERT triggers are automatically invoked after an insert event occurs on the table.

The following shows the basic syntax of creating a MySQL AFTER INSERT trigger:

**CREATE** **TRIGGER** trigger\_name

**AFTER** **INSERT**

**ON** table\_name **FOR** **EACH** **ROW**

trigger\_body

In this syntax:

First, specify the name of the trigger that you want to create after the [CREATE TRIGGER](https://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx) keywords.

Second, use AFTER INSERT clause to specify the time to invoke the trigger.

Third, specify the name of the table on which you want to create the trigger after the ON keyword.

Finally, specify the trigger body which consists of one or more statements that execute when the trigger is invoked.

In case the trigger body has multiple statements, you need to use the BEGIN END block and change the default [delimiter](https://www.mysqltutorial.org/mysql-stored-procedure/mysql-delimiter/):

DELIMITER $$

**CREATE** **TRIGGER** trigger\_name

**AFTER** **INSERT**

**ON** table\_name **FOR** **EACH** **ROW**

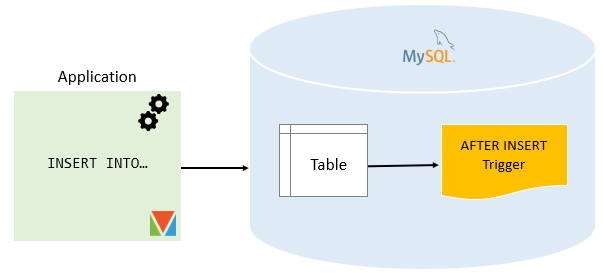
**BEGIN**

*-- statements*

**END**$$

DELIMITER ;

In an AFTER INSERT trigger, you can access the NEW values but you cannot change them. Also, you cannot access the OLD values because there is no OLD on INSERT triggers.



## MySQL AFTER INSERT trigger example

Consider the following AFTER INSERT trigger example.

### Setting up a sample table

First, [create a new table](https://www.mysqltutorial.org/mysql-create-table/) called members:

**DROP** **TABLE** **IF** **EXISTS** members;

**CREATE** **TABLE** members (

**id** INT AUTO\_INCREMENT,

**name** VARCHAR(100) **NOT** NULL,

email VARCHAR(255),

birthDate DATE,

PRIMARY **KEY** (**id**)

);

Second, create another table called reminders that stores reminder messages to members.

**DROP** **TABLE** **IF** **EXISTS** reminders;

**CREATE** **TABLE** reminders (

**id** INT AUTO\_INCREMENT,

memberId INT,

message VARCHAR(255) **NOT** NULL,

PRIMARY **KEY** (**id** , memberId)

);

### Creating AFTER INSERT trigger example

The following statement creates an AFTER INSERT trigger that inserts a reminder into the reminders table if the birth date of the member is NULL.

DELIMITER $$

**CREATE** **TRIGGER** after\_members\_insert

**AFTER** **INSERT**

**ON** members **FOR** **EACH** **ROW**

**BEGIN**

**IF** NEW.birthDate **IS** NULL **THEN**

**INSERT** **INTO** reminders(memberId, message)

**VALUES**(new.id,**CONCAT**('Hi ', NEW.name, ', please update your date of birth.'));

**END** **IF**;

**END**$$

DELIMITER ;

In this trigger:

First, the name of the trigger is after\_members\_insert specified in the CREATE TRIGGER clause:

**CREATE** **TRIGGER** after\_members\_insert

Second, the triggering event is:

AFTER **INSERT**

Third, the table that the trigger associated with is members table:

ON members FOR EACH ROW

Finally, inside the trigger body, insert a new row into the reminder table if the birth date of the member is NULL.

### Testing the MySQL AFTER INSERT trigger

First, [insert two rows](https://www.mysqltutorial.org/mysql-insert-multiple-rows/) into the members table:

**INSERT** **INTO** members(**name**, email, birthDate)

**VALUES**

('John Doe', 'john.doe@example.com', NULL),

('Jane Doe', 'jane.doe@example.com','2000-01-01');

Second, [query data](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) from the members table:

**SELECT** \* **FROM** members;

MySQL AFTER INSERT Trigger example

Third, query data from reminders table:

**SELECT** \* **FROM** reminders;

MySQL AFTER INSERT Trigger Output

We inserted two rows into the members table. However, only the first row that has a birth date value NULL, therefore, the trigger inserted only one row into the reminders table.