MySQL AFTER UPDATE Trigger

MySQL AFTER UPDATE [triggers](https://www.mysqltutorial.org/mysql-triggers.aspx) are invoked automatically after an [update](https://www.mysqltutorial.org/mysql-update-data.aspx) event occurs on the table associated with the triggers.

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

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

**AFTER** **UPDATE**

**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 AFTER UPDATE clause to specify the time to invoke the trigger.

Third, specify the name of the table to which the trigger belongs after the ON keyword.

Finally, specify the trigger body which consists of one or more statements.

If the trigger body has more than one statement, you need to use the BEGIN END block. And, you also need to [change the default delimiter](https://www.mysqltutorial.org/mysql-stored-procedure/mysql-delimiter/) as shown in the following code:

DELIMITER $$

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

**AFTER** **UPDATE**

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

**BEGIN**

*-- statements*

**END**$$

DELIMITER ;

In a AFTER UPDATE trigger, you can access OLD and NEW rows but cannot update them.

MySQL AFTER UPDATE trigger example

Let’s look at an example of creating a AFTER UPDATE trigger.

Setting up a sample table

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

**DROP** **TABLE** **IF** **EXISTS** Sales;

**CREATE** **TABLE** Sales (

**id** INT AUTO\_INCREMENT,

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

quantity INT **NOT** NULL **DEFAULT** 0,

fiscalYear SMALLINT **NOT** NULL,

fiscalMonth TINYINT **NOT** NULL,

**CHECK**(fiscalMonth >= 1 **AND** fiscalMonth <= 12),

**CHECK**(fiscalYear **BETWEEN** 2000 **and** 2050),

**CHECK** (quantity >=0),

**UNIQUE**(product, fiscalYear, fiscalMonth),

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

);

Second, [insert sample data](https://www.mysqltutorial.org/mysql-insert-multiple-rows/) into the Sales table:

**INSERT** **INTO** Sales(product, quantity, fiscalYear, fiscalMonth)

**VALUES**

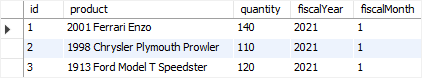
('2001 Ferrari Enzo',140, 2021,1),

('1998 Chrysler Plymouth Prowler', 110,2021,1),

('1913 Ford Model T Speedster', 120,2021,1);

Third, [query data](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) from the Sales table to display its contents:

**SELECT** \* **FROM** Sales;



Finally, [create a table](https://www.mysqltutorial.org/mysql-create-table/) that stores the changes in the quantity column from the sales table:

**DROP** **TABLE** **IF** **EXISTS** SalesChanges;

**CREATE** **TABLE** SalesChanges (

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

salesId INT,

beforeQuantity INT,

afterQuantity INT,

changedAt TIMESTAMP **NOT** NULL **DEFAULT** **CURRENT\_TIMESTAMP**

);

Creating AFTER UPDATE trigger example

The following statement creates an AFTER UPDATE trigger on the sales table:

DELIMITER $$

**CREATE** **TRIGGER** after\_sales\_update

**AFTER** **UPDATE**

**ON** sales **FOR** **EACH** **ROW**

**BEGIN**

**IF** OLD.quantity <> new.quantity **THEN**

**INSERT** **INTO** SalesChanges(salesId,beforeQuantity, afterQuantity)

**VALUES**(old.id, old.quantity, new.quantity);

**END** **IF**;

**END**$$

DELIMITER ;

This after\_sales\_update trigger is automatically fired before an update event occurs for each row in the sales table.

If you update the value in the quantity column to a new value the trigger insert a new row to log the changes in the SalesChanges table.

Let’s examine the trigger in detail:

First, the name of the trigger is after\_sales\_update specified in the [CREATE TRIGGER](https://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx) clause:

**CREATE** **TRIGGER** after\_sales\_update

Second, the triggering event is:

AFTER **UPDATE**

Code language: SQL (Structured Query Language) (sql)

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

ON Sales FOR EACH ROW

Code language: SQL (Structured Query Language) (sql)

Finally, use the [IF-THEN](https://www.mysqltutorial.org/mysql-if-statement/) statement inside the trigger body to check if the new value is not the same as the old one, then insert the changes into the SalesChanges table:

IF OLD.quantity <> new.quantity THEN

**INSERT** **INTO** SalesChanges(salesId,beforeQuantity, afterQuantity)

**VALUES**(old.id, old.quantity, new.quantity);

**END** **IF**;

Testing the MySQL AFTER UPDATE trigger

First, [update](https://www.mysqltutorial.org/mysql-update-data.aspx) the quantity of the row with id 1 to 350:

**UPDATE** Sales

**SET** quantity = 350

**WHERE** **id** = 1;

The after\_sales\_update was invoked automatically.

Second, query data from the SalesChanges table:

**SELECT** \* **FROM** SalesChanges;

MySQL AFTER UPDATE trigger - example

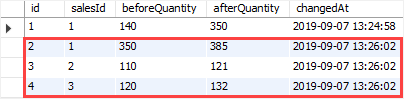
Third, increase the sales quantity of all rows to 10%:

**UPDATE** Sales

**SET** quantity = **CAST**(quantity \* 1.1 **AS** **UNSIGNED**);

Fourth, query data from the SalesChanges table:

**SELECT** \* **FROM** SalesChanges;



The trigger fired three times because of the updates of the three rows.