# MySQL Triggers

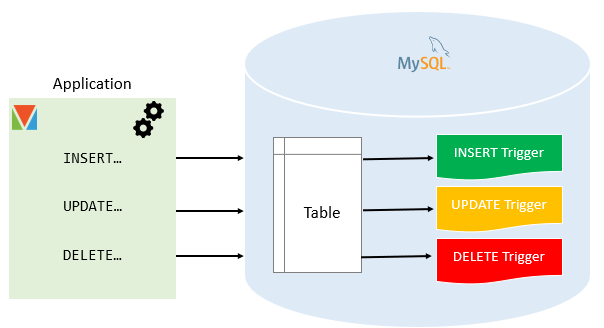
In MySQL, a trigger is a stored program invoked **automatically** in response to an event such as [insert](https://www.mysqltutorial.org/mysql-insert-statement.aspx), [update](https://www.mysqltutorial.org/mysql-update-data.aspx), or [delete](https://www.mysqltutorial.org/mysql-delete-statement.aspx)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

**Different types of MySQL Triggers (with examples)**

A MySQL **trigger** is a stored program (with queries) which is executed automatically to respond to a specific event such as insertion, updation or deletion occurring in a table.

There are 6 different types of triggers in MySQL:

**1. 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:

create table customer (acc\_no integer primary key,

cust\_name varchar(20),

avail\_balance decimal);

create table mini\_statement (acc\_no integer,

avail\_balance decimal,

foreign key(acc\_no) references customer(acc\_no) on delete cascade);

Inserting values in them:

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:

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; //

Making updates to invoke trigger:

delimiter;

update customer set avail\_balance = avail\_balance + 3000 where acc\_no = 1001;

update customer set avail\_balance = avail\_balance + 3000 where acc\_no = 1000;

**Output:**

select \*from mini\_statement;

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

| acc\_no | avail\_balance |

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

| 1001 | 12000 |

| 1000 | 7000 |

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

2 rows in set (0.0007 sec)

**2. 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:

create table micro\_statement (acc\_no integer,

avail\_balance decimal,

foreign key(acc\_no) references customer(acc\_no) on delete cascade);

Insert another value into customer:

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

Query OK, 1 row affected (0.0786 sec)

Trigger to insert (new) values of account number and available balance into micro\_statement record after an update has occurred:

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; //

Making an update to invoke trigger:

delimiter ;

update customer set avail\_balance = avail\_balance + 1500 where acc\_no = 1002;

**Output:**

select \*from micro\_statement;

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

| acc\_no | avail\_balance |

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

| 1002 | 6000 |

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

1 row in set (0.0007 sec)

**3. Before Insert Trigger:**  
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));

Trigger to insert contact information such as name, birthday and creation-date/user into a table contact before an insert occurs:

delimiter //

create trigger contacts\_before\_insert

-> before insert

-> on contacts for each row

-> begin

-> DECLARE vUser varchar(50);

->

-> -- Find username of person performing INSERT into table

-> select USER() into vUser;

->

-> -- Update create\_date field to current system date

-> SET NEW.created\_date = SYSDATE();

->

-> -- Update created\_by field to the username of the person performing the INSERT

-> SET NEW.created\_by = vUser;

-> end; //

Making an insert to invoke the trigger:

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");

**Output:**

select \*from contacts;

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

| contact\_id | last\_name | first\_name | birthday | created\_date | created\_by |

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

| 1 | Newton | Enigma | 1999-08-19 | 2019-05-11 | root@localhost |

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

**4. After Insert Trigger:**  
As the name implies, this trigger gets invoked after an insert is implemented.

**Example:**  
Consider tables:

create table contacts (contact\_id int (11) NOT NULL AUTO\_INCREMENT,

last\_name VARCHAR(30) NOT NULL,

first\_name VARCHAR(25), birthday DATE,

->CONSTRAINT contacts\_pk PRIMARY KEY (contact\_id));

create table contacts\_audit (contact\_id integer,

created\_date date,

created\_by varchar (30));

Trigger to insert contact\_id and contact creation-date/user information into contacts\_audit record after an insert occurs:

delimiter //

create trigger contacts\_after\_insert

-> after insert

-> on contacts for each row

-> begin

-> DECLARE vUser varchar(50);

->

-> -- Find username of person performing the INSERT into table

-> SELECT USER() into vUser;

->

-> -- Insert record into audit table

-> INSERT into contacts\_audit

-> ( contact\_id,

-> created\_date,

-> created\_by)

-> VALUES

-> ( NEW.contact\_id,

-> SYSDATE(),

-> vUser );

-> END; //

Making an insert to invoke the trigger:

insert into contacts values (1, "Kumar", "Rupesh",

str\_to\_date("20-06-1999", "%d-%m-%Y"));

**Output:**

select \*from contacts\_audit;

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

| contact\_id | created\_date | created\_by |

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

| 1 | 2019-05-11 | root@localhost |

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

1 row in set (0.0006 sec)

**5. Before Delete Trigger:**  
As the name implies, this trigger is invoked before a delete occurs, or before deletion statement is implemented.

**Example:**  
Consider 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));

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:

delimiter //

create trigger contacts\_before\_delete

-> before delete

-> on contacts for each row

-> begin

->

-> DECLARE vUser varchar(50);

->

-> -- Find username of person performing the DELETE into table

-> SELECT USER() into vUser;

->

-> -- Insert record into audit table

-> INSERT into contacts\_audit

-> ( contact\_id,

-> deleted\_date,

-> deleted\_by)

-> VALUES

-> ( OLD.contact\_id,

-> SYSDATE(),

-> vUser );

-> end; //

Making an insert and then deleting the same to invoke the trigger:

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";

**Output:**

select \*from contacts\_audit;

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

| contact\_id | deleted\_date | deleted\_by |

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

| 1 | 2019-05-11 | root@localhost |

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

1 row in set (0.0007 sec)

**6. After Delete Trigger:**  
As the name implies, this trigger is invoked after a delete occurs, or after a delete operation is implemented.

**Example:**  
Consider the 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));

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 after a delete occurs:

create trigger contacts\_after\_delete

-> after delete

-> on contacts for each row

-> begin

->

-> DECLARE vUser varchar(50);

->

-> -- Find username of person performing the DELETE into table

-> SELECT USER() into vUser;

->

-> -- Insert record into audit table

-> INSERT into contacts\_audit

-> ( contact\_id,

-> deleted\_date,

-> deleted\_by)

-> VALUES

-> ( OLD.contact\_id,

-> SYSDATE(),

-> vUser );

-> end; //

Making an insert and deleting the same to invoke the trigger:

delimiter;

insert into contacts values (1, "Newton", "Isaac",

str\_to\_date ("19-08-1985", "%d-%m-%Y"),

str\_to\_date ("23-07-2018", "%d-%m-%Y"), "xyz");

delete from contacts where first\_name="Isaac";

**Output:**

select \*from contacts\_audit;

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

| contact\_id | deleted\_date | deleted\_by |

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

| 1 | 2019-05-11 | root@localhost |

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

1 row in set (0.0009 sec)