**MySQL Triggers - Lab. Exercise 8: 15-19, April 2024)**

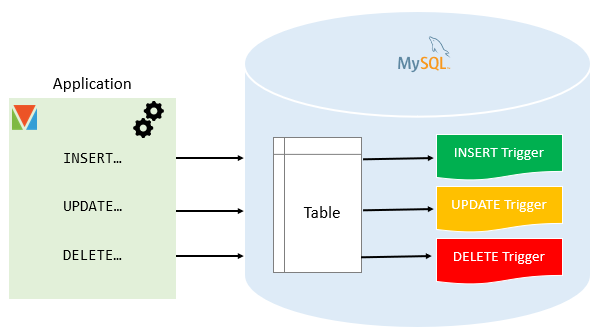
**Source:** [**https://www.geeksforgeeks.org/different-types-of-mysql-triggers-with-examples/**](https://www.geeksforgeeks.org/different-types-of-mysql-triggers-with-examples/)

In MySQL, a trigger is a **stored program** invoked automatically in response to an event such as insert, update, or delete 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.

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.



There are 6 different types of triggers scenarios in MySQL:

**1. Before Update Trigger 2. After Update Trigger**

**3. Before Insert Trigger 4. After Insert Trigger**

**5. Before Delete Trigger 6. After Delete Trigger**

**MYSQL TRIGGERS SYNTAX: Source: https://dev.mysql.com/doc/refman/8.3/en/create-trigger.html**

**15.1.22 CREATE TRIGGER Statement**

**CREATE**

**[DEFINER = *user*]**

**TRIGGER [IF NOT EXISTS] *trigger\_name***

***trigger\_time* *trigger\_event***

**ON *tbl\_name* FOR EACH ROW**

**[*trigger\_order*]**

***trigger\_body***

***trigger\_time*: { BEFORE | AFTER }**

***trigger\_event*: { INSERT | UPDATE | DELETE }**

***trigger\_order*: { FOLLOWS | PRECEDES } *other\_trigger\_name***

This statement creates a new trigger. A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. The trigger becomes associated with the table named *tbl\_name*, which must refer to a permanent table. You cannot associate a trigger with a TEMPORARY table or a view.

Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema. Triggers in different schemas can have the same name.

IF NOT EXISTS prevents an error from occurring if a trigger having the same name, on the same table, exists in the same schema.

*--------------------------------------------------------------------------*

**Source: https://dev.mysql.com/doc/refman/8.3/en/drop-trigger.html**

**15.1.34 DROP TRIGGER Statement**

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

This statement drops a trigger. The schema (database) name is optional. If the schema is omitted, the trigger is dropped from the default schema. DROP TRIGGER requires the TRIGGER privilege for the table associated with the trigger.

Use IF EXISTS to prevent an error from occurring for a trigger that does not exist. A NOTE is generated for a nonexistent trigger when using IF EXISTS. See Section 15.7.7.44, “SHOW WARNINGS Statement”.

Triggers for a table are also dropped if you drop the table. ---------------------------------------------------------------------SET SQL\_SAFE\_UPDATES = 0;

-- 1. **Before Update Trigger**

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

);

INSERT INTO customer VALUES (1000, 'Fanny', 7000);

INSERT INTO customer VALUES (1001, 'Peter', 12000);

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 ;

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**

CREATE TABLE customer2 (

acc\_no INTEGER PRIMARY KEY,

cust\_name VARCHAR(20),

avail\_balance DECIMAL

);

CREATE TABLE micro\_statement (

acc\_no INTEGER,

avail\_balance DECIMAL,

FOREIGN KEY(acc\_no) REFERENCES customer2(acc\_no) ON DELETE CASCADE

);

INSERT INTO customer2 VALUES (1002, 'Janitor', 4500);

DELIMITER //

CREATE TRIGGER update\_after

AFTER UPDATE ON customer2

FOR EACH ROW

BEGIN

INSERT INTO micro\_statement VALUES(NEW.acc\_no, NEW.avail\_balance);

END; //

DELIMITER ;

UPDATE customer2 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**

CREATE TABLE contacts1 (

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 contacts1\_pk PRIMARY KEY (contact\_id)

);

DELIMITER //

CREATE TRIGGER contacts1\_before\_insert

BEFORE INSERT ON contacts1

FOR EACH ROW

BEGIN

DECLARE vUser VARCHAR(50);

-- Find username of person performing INSERT into table

SELECT USER() INTO vUser;

-- Update created\_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; //

DELIMITER ;

INSERT INTO contacts1 (last\_name, first\_name, birthday)

VALUES ('Newton', 'Enigma', STR\_TO\_DATE('19-08-1999', '%d-%m-%Y'));

**Output:**

**select \*from contacts1;**

**+------------+-----------+------------+------------+--------------+----------------+**

**| 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**

CREATE TABLE contacts2 (

contact\_id INT(11) NOT NULL AUTO\_INCREMENT,

last\_name VARCHAR(30) NOT NULL,

first\_name VARCHAR(25),

birthday DATE,

CONSTRAINT contacts2\_pk PRIMARY KEY (contact\_id)

);

CREATE TABLE contacts2\_audit (

contact\_id INTEGER,

created\_date DATE,

created\_by VARCHAR(30)

);

DELIMITER //

CREATE TRIGGER contacts2\_after\_insert

AFTER INSERT ON contacts2

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 contacts2\_audit

(contact\_id,

created\_date,

created\_by)

VALUES

(NEW.contact\_id,

SYSDATE(),

vUser);

END; //

DELIMITER ;

INSERT INTO contacts2 (last\_name, first\_name, birthday)

VALUES ('Kumar', 'Rupesh', STR\_TO\_DATE('20-06-1999', '%d-%m-%Y'));

**Output:**

**select \*from contacts2\_audit;**

**+------------+--------------+----------------+**

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

**+------------+--------------+----------------+**

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

**+------------+--------------+----------------+**

**1 row in set (0.0006 sec)**

-- **5. Before Delete Trigger**

CREATE TABLE contacts3 (

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 contacts3\_pk PRIMARY KEY (contact\_id)

);

CREATE TABLE contacts3\_audit (

contact\_id INTEGER,

deleted\_date DATE,

deleted\_by VARCHAR(20)

);

DELIMITER //

CREATE TRIGGER contacts3\_before\_delete

BEFORE DELETE ON contacts3

FOR EACH ROW

BEGIN

DECLARE vUser VARCHAR(50);

-- Find username of person performing the DELETE from table

SELECT USER() INTO vUser;

-- Insert record into audit table

INSERT INTO contacts3\_audit

(contact\_id,

deleted\_date,

deleted\_by)

VALUES

(OLD.contact\_id,

SYSDATE(),

vUser);

END; //

DELIMITER ;

INSERT INTO contacts3 (last\_name, first\_name, birthday, created\_date, created\_by)

VALUES ('Bond', 'Ruskin', STR\_TO\_DATE('19-08-1995', '%d-%m-%Y'), STR\_TO\_DATE('27-04-2018', '%d-%m-%Y'), 'xyz');

DELETE FROM contacts3 WHERE last\_name = 'Bond';

**Output:**

**select \*from contacts3\_audit;**

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

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

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

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

-- **6. After Delete Trigger**

CREATE TABLE contacts4 (

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 contacts4\_pk PRIMARY KEY (contact\_id)

);

CREATE TABLE contacts4\_audit (

contact\_id INTEGER,

deleted\_date DATE,

deleted\_by VARCHAR(20)

);

DELIMITER //

CREATE TRIGGER contacts4\_after\_delete

AFTER DELETE ON contacts4

FOR EACH ROW

BEGIN

DECLARE vUser VARCHAR(50);

-- Find username of person performing the DELETE from table

SELECT USER() INTO vUser;

-- Insert record into audit table

INSERT INTO contacts4\_audit

(contact\_id,

deleted\_date,

deleted\_by)

VALUES

(OLD.contact\_id,

SYSDATE(),

vUser);

END; //

DELIMITER ;

INSERT INTO contacts4 (last\_name, first\_name, birthday, created\_date, created\_by)

VALUES ('Newton', 'Isaac', STR\_TO\_DATE('19-08-1985', '%d-%m-%Y'), STR\_TO\_DATE('23-07-2018', '%d-%m-%Y'), 'xyz');

DELETE FROM contacts4 WHERE first\_name = 'Isaac';

**Output:**

**select \*from contacts4\_audit;**

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

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

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

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