**Trigger**

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

* 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.

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:

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

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

output:

select \*from mini\_statement;

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

| acc\_no | avail\_balance |

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

| 1000 | 7000 |

| 1001| 12000 |

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

* After Update Trigger:  
  As the name implies, this trigger is invoked after an updating 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);

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:

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

Output:

select \*from micro\_statement;

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

| acc\_no | avail\_balance |

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

| 1002 | 6000 |

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

* 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:

insert into contacts values (1, "Newton", "Enigma", "1999-08-19", "2018-03-17", "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 |

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

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

Example:  
Consider tables:

create table contacts1(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 contacts1\_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 contacts1 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 contacts1\_audit

( contact\_id,

created\_date,

created\_by)

VALUES

( NEW.contact\_id,

SYSDATE(),

vUser );

END; //

Making an insert to invoke the trigger:

insert into contacts1 values (1, "Kumar", "Rupesh", "1999-06-20");

**Output:**

select \*from contacts\_audit;

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

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

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

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

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

* **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:

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)

* **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:

delimiter //

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:

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)

* Drop trigger:

Drop trigger contacts\_after\_insert

**View**

## CREATE VIEW Statement:

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

CREATE VIEW Bangladeshi\_Customers AS

SELECT CustomerName, ContactName

FROM CustomerInfo

WHERE Country = 'Bangladesh';

select \*from Bangladeshi\_Customers

CREATE VIEW Above\_Average\_Price AS

SELECT ProductName, Price

FROM ProductDetails

WHERE Price > (SELECT AVG(Price) FROM ProductDetails);

select \*from Above\_Average\_Price

## Updating a View:

A view can be updated with the CREATE OR REPLACE VIEW statement.

The following SQL adds the "City" column to the "Bangladeshi\_Customers" view:

CREATE OR REPLACE VIEW Bangladeshi\_Customers AS

SELECT CustomerName, ContactName, City

FROM CustomerInfo

WHERE Country = 'Bangladesh';

select \*from Bangladeshi\_Customers

## Dropping a View:

Drop view Bangladeshi\_Customers

**Stored Procedures**

# Procedures in MySQL:

A procedure is a subroutine (like a subprogram) in a regular scripting language, stored in a database. In the case of MySQL, procedures are written in MySQL and stored in the MySQL database/server. A MySQL procedure has a name, a parameter list, and SQL statement(s).

There are four different types of MySQL procedures:

1. Procedure with no parameters:  
A procedure without parameters does not take any input or casts an output indirectly. It is simply called with its procedure name followed by () (without any parameters). It is used for simple queries.

Example:  
Consider two tables author and book:

create table author (author\_id integer primary key, authorName varchar(30), email varchar (25), gender varchar (6));

create table book (BookId integer not null unique, ISBN integer primary key, book\_name varchar (30) not null, author integer, ed\_num integer, price integer, pages integer, foreign key (author) references author (author\_id) on delete cascade);

Inserting values into them:

insert into author values

(1, "Kraig Muller", "Wordnewton@gmail.com", "Male");

insert into author values

(2, "Karrie Nicolette", "karrie23@gmail.com", "Female");

insert into book values

(1, 001, "Glimpses of the past", 1, 1, 650, 396);

insert into book values

(2, 002, "Beyond The Horizons of Venus", 1, 1, 650, 396);

insert into book values

(3, 003, "Ultrasonic Aquaculture", 2, 1, 799, 500);

insert into book values

(4, 004, "Cyrogenic Engines", 2, 1, 499, 330);

Procedure (with no parameters) to display all the books:

delimiter //

create procedure display\_book()

begin

select \*from book;

end //

call display\_book()

Output:

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

| BookId | ISBN | book\_name | author | ed\_num | price | pages |

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

| 1 | 1 | Glimpses of the past | 1 | 1 | 650 | 396 |

| 2 | 2 | Beyond The Horizons of Venus | 1 | 1 | 650 | 396 |

| 3 | 3 | Ultrasonic Aquaculture | 2 | 1 | 799 | 500 |

| 4 | 4 | Cyrogenic Engines | 2 | 1 | 499 | 330 |

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

2. Procedure with IN parameter:  
An IN parameter is used to take a parameter as input.

Example:  
Procedure to update price of a book taking ISBN of the book and its new price as input: (considering the tables above)

delimiter //

create procedure update\_price (IN temp\_ISBN varchar(10), IN new\_price integer)

begin

update book set price=new\_price where ISBN=temp\_ISBN;

end //

call update\_price(001, 600)

We changed the price of book with ISBN ‘001’(Glimpses of the past) to 600 (from its default price 650).

Output:

select \*from book;

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

| BookId | ISBN | book\_name | author | ed\_num | price | pages |

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

| 1 | 1 | Glimpses of the past | 1 | 1 | 600 | 396 |

| 2 | 2 | Beyond The Horizons of Venus | 1 | 1 | 650 | 396 |

| 3 | 3 | Ultrasonic Aquaculture | 2 | 1 | 799 | 500 |

| 4 | 4 | Cyrogenic Engines | 2 | 1 | 499 | 330 |

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

3. Procedure with OUT parameter:  
An OUT parameter is used to pass a parameter as output or display like the select operator.

Example:  
Procedure to display the highest price among all the books with an output parameter:

delimiter //

create procedure disp\_max(OUT highestprice integer)

begin

select max(price) into highestprice from book;

end //

call disp\_max(@Maximumprice) //

select @Maximumprice

**Output:**  
The highest price from our book database is of the book with ISBN 003 (Ultrasonic Aquaculture) with a price of 799, which is displayed.

+-----+

| @M |

+-----+

| 799 |

+-----+

4. Procedure with IN-OUT parameter:  
An INOUT parameter is a combination of IN and OUT parameters.

Example:  
Procedure to take gender type input (‘Male’/’Female’ here) with an in-out parameter which reflects the number of authors falling in that gender category/type:

delimiter //

create procedure disp\_gender(INOUT total\_gender integer, IN take\_gender varchar(6))

begin

select COUNT(gender) INTO total\_gender FROM author where gender = take\_gender;

end; //

delimiter ;

call disp\_gender(@TM, "Male");

select @TM;

call disp\_gender(@TF, "Female");

select @TF;

**Output:**  
We have two authors, one being male and one being female as per insertions in the table author. Hence, output is 1 for one male author and 1 for one female author respectively.

+----+

| @TM |

+----+

| 1 |

+----+

+----+

| @TF |

+----+

| 1 |

+----+

Drop procedure:

Drop procedure disp\_gender

Bottom of Form