**Triggers:**

A type of stored program, associated with a table, that will be activated automatically once a specific event occurs this event must be related to the associated table and represented by one of the following three DML statements:

-INSERT

- UPDATE

- DELETE

A trigger is a MySQL object that can “trigger” a specific action or calculation ‘before’ or ‘after’ an INSERT, UPDATE, or DELETE statement has been executed.

use employees;

commit;

# Before Insert

delimiter $$

create trigger before\_sal\_insert

before insert on salaries

for each row

begin

if new.salary < 0 then

set new.salary = 0;

end if;

end$$

delimiter ;

select \* from salaries where emp\_no=10001;

insert into salaries values('10001',-92891,'2010-06-22','9999-01-01');

select \* from salaries where emp\_no=10001;

# Before Update

delimiter $$

create trigger before\_sal\_update

before update on salaries

for each row

begin

if new.salary <0 then

set new.salary = old.salary;

end if;

end$$

delimiter ;

update salaries set salary = 98765 where emp\_no='10001' and from\_date = '2010-06-22';

select \* from salaries where emp\_no='10001' and from\_date = '2010-06-22';

update salaries set salary = -50000 where emp\_no = '10001' and from\_date='2010-06-22';

select \* from salaries where emp\_no = '10001' and from\_date = '2010-06-22';

**system functions = built-in functions**

select sysdate();

select date\_format(sysdate(),'%y-%m-%d')as today;

select date\_format(sysdate(),'%d-%m-%y') as today\_ind;

**Example:**

**Create a trigger that checks if the hire date of an employee is higher than the current date. If true, set this date to be the current date. Format the output appropriately (YY-MM-DD).**

DELIMITER $$

CREATE TRIGGER trig\_hire\_date

BEFORE INSERT ON employees

FOR EACH ROW

BEGIN

IF NEW.hire\_date > date\_format(sysdate(), '%Y-%m-%d') THEN

SET NEW.hire\_date = date\_format(sysdate(), '%Y-%m-%d');

END IF;

END $$

DELIMITER ;

INSERT employees VALUES ('999904', '1970-01-31', 'John', 'Johnson', 'M', '2025-01-01');

SELECT

\*

FROM

employees

ORDER BY emp\_no DESC;

The Case Statement:

Graphical user interface, text, application

Description automatically generated

select emp\_no,first\_name,last\_name,

case

when gender='M' then 'Male'

else 'Female'

end as gender

from employees;

select emp\_no,first\_name,last\_name,

case gender

when 'M' then 'Male'

else 'Female'

end as gender

from

employees;

select \* from employees;

select e.emp\_no,e.first\_name,e.last\_name,

case

when dm.emp\_no is not null then 'Manager'

else 'Employee'

end as is\_manager

from employees e

left join

dept\_manager dm on dm.emp\_no = e.emp\_no

where

e.emp\_no >109990;

select emp\_no,first\_name,last\_name,

if(gender='M','Male','Female') as gender

from employees;

**Cursors:**

In MySQL, a cursor allows row-by-row processing of the result sets. A cursor is used for the result set and returned from a query. By using a cursor, you can iterate, or step through the results of a query and perform certain operations on each row.

The cursor allows you to iterate through the result set and then perform the additional processing only on the rows that require it.

A cursor contains the data in a loop. Cursors may be different from SQL commands that operate on all the rows in the returned by a query at one time.

MySQL cursor has three types of properties.

**1. Read Only**

The data in the underlying table cannot be modified via a cursor.

**2. Non\_Scrollable**

Only rows can be retrieved in the order specified by the SELECT statement. In the reverse order, users can not retrieve records. Additionally, in the result set, users cannot skip rows or jump to a particular row.

**3. Asensitive**

An asensitive cursor is used to points the actual data, whereas a temporary copy of the data is used by an insensitive cursor used. An asensitive cursor performs faster than an insensitive cursor because it does not have to make a temporary copy of data.

**Working with MySQL Cursors**

There are some steps we must follow while using the MySQL Cursors, let’s see.

* Declare a Cursor
* Open a Cursor
* Fetch the Cursor
* Close the Cursor

1. Declaration of a Cursor

To declare a cursor, you must use the DECLARE statement. With the help of the variables, we need conditions and handlers to declare a cursor before we can use it.

First of all, we will give the cursor a name, this is how we will refer to it later in the procedure. We can have more than one cursor in a single procedure so it’s necessary to give it a name that will in some way tell us what it’s doing.

We then need to specify the select statement we want to associate with the cursor. The SQL statement can be any valid SQL statement and it is possible to use a dynamic where clause using variable or parameters as we have seen previously.

Syntax

DECLARE <cursor\_name> CURSOR FOR <select\_statement>;

**2. Open a Cursor**

For opening a cursor we must use the open statement. If we want to fetch rows from it, then you must have to open the cursor.

**Syntax**

OPEN <cursor\_name>;

3. Fetch the Cursor

When we have to retrieve the next row from the cursor and move the cursor to the next row then you need to fetch the cursor. If any row exists, then the below statement fetches the next row and cursor pointer moves ahead to the next row.

**Syntax**

FETCH <cursor\_name> INTO <variable\_list>;

**4. Close the Cursor**

This statement closes the open cursor, and it will deactivate the cursor and release the memory. By this statement we can close the previously opened cursor.

If it is not closed explicitly then a cursor is closed at the end of compound statement in which that was declared.

**Syntax**

CLOSE <cursor\_name>;

use employees;

DELIMITER $$

drop function if exists FindSiteID;

CREATE FUNCTION FindSiteID (name\_in VARCHAR(50)) RETURNS INT

no sql

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE DID INT DEFAULT 0;

DECLARE c1 CURSOR FOR

SELECT emp\_no

FROM employees

WHERE first\_name = name\_in;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN c1;

FETCH c1 INTO DID;

CLOSE c1;

RETURN DID;

END$$

DELIMITER ;

select FindSiteID('Mary');

Example 2:

CREATE TABLE GetVatsaCursor(

C\_ID INT PRIMARY KEY AUTO\_INCREMENT,

c\_name VARCHAR(50),

c\_address VARCHAR(200)

);

CREATE TABLE Vbackupdata(

C\_ID INT,

c\_name VARCHAR(50),

c\_address VARCHAR(200)

);

INSERT INTO GetVatsaCursor(c\_name, c\_address) VALUES('Test', '132, Vatsa Colony'),

('Admin', '133, Vatsa Colony'),

('Shareef', '134, Vatsa Colony'),

('Vedant', '135, Vatsa Colony'),

('Harish', '136, Vatsa Colony'),

('Sahoo', '137, Vatsa Colony'),

('Wasim', '138, Vatsa Colony'),

('Baharathi', '139, Vatsa Colony'),

('Srinithi', '140, Vatsa Colony');

SELECT \* FROM GetVatsaCursor;

SELECT \* FROM Vbackupdata;

delimiter $$

CREATE PROCEDURE firstCurs()

BEGIN

DECLARE d INT DEFAULT 0;

DECLARE c\_id INT;

DECLARE c\_name, c\_address VARCHAR(20);

DECLARE Get\_cur CURSOR FOR SELECT \* FROM GetVatsaCursor;

DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'

SET d = 1;

DECLARE CONTINUE HANDLER FOR SQLSTATE '23000'

SET d = 1;

OPEN Get\_cur;

lbl: LOOP

IF d = 1 THEN

LEAVE lbl;

END IF;

IF NOT d = 1 THEN

FETCH Get\_cur INTO c\_id, c\_name, c\_address;

INSERT INTO Vbackupdata VALUES(c\_id, c\_name, c\_address);

END IF;

END LOOP;

CLOSE Get\_cur;

END$$

delimiter ;

call firstCurs();

SELECT \* FROM Vbackupdata;