Database Management Systems Lab Experiment No: - 11
Aim: - Implementation of Cursor

Cursor:-

To handle a result set inside a stored procedure, you use a cursor. A cursor allows you to iterate a set of rows returned by a query and process each row individually.

MySQL cursor is read-only, non-scrollable and asensitive.

- **Read-only**: you cannot update data in the underlying table through the cursor.
- **Non-scrollable**: you can only fetch rows in the order determined by the SELECT statement. You cannot fetch rows in the reversed order. In addition, you cannot skip rows or jump to a specific row in the result set.
- **Asensitive**: there are two kinds of cursors: asensitive cursor and insensitive cursor. An asensitive cursor points to the actual data, whereas an insensitive cursor uses a temporary copy of the data. The server may or may not make a copy of its result table. An asensitive cursor performs faster than an insensitive cursor because it does not have to make a temporary copy of data. However, any change that made to the data from other connections will affect the data that is being used by an asensitive cursor, therefore, it is safer if you do not update the data that is being used by an asensitive cursor. MySQL cursor is asensitive.

You can use MySQL cursors in stored procedures, stored functions, and triggers.

Working with MySQL cursor

First, declare a cursor by using the DECLARE statement:

```
DECLARE cursor_name CURSOR FOR SELECT_statement;
```

The cursor declaration must be after any variable declaration. If you declare a cursor before the variable declarations, MySQL will issue an error. A cursor must always associate with a SELECT statement.

Next, open the cursor by using the OPEN statement. The OPEN statement initializes the result set for the cursor, therefore, you must call the OPEN statement before fetching rows from the result set.

```
OPEN cursor_name;
```

Then, use the FETCH statement to retrieve the next row pointed by the cursor and move the cursor to the next row in the result set.

```
FETCH cursor_name INTO variables list;
```

After that, check if there is any row available before fetching it.

Finally, deactivate the cursor and release the memory associated with it using the CLOSE statement:

```
CLOSE cursor_name;
```

It is a good practice to always close a cursor when it is no longer used.

When working with MySQL cursor, you must also declare a NOT FOUND handler to handle the situation when the cursor could not find any row.

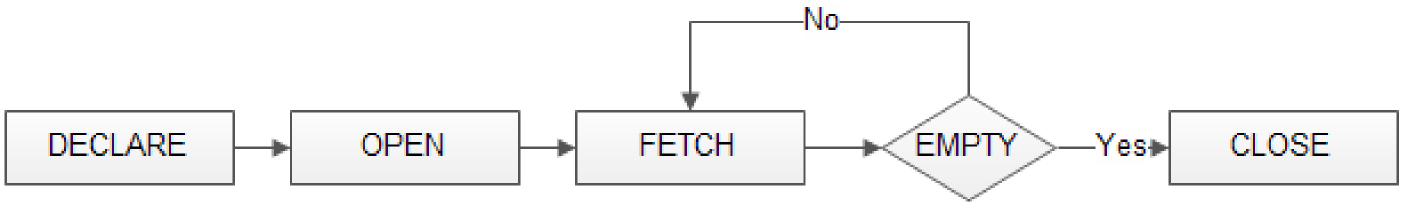
Because each time you call the FETCH statement, the cursor attempts to read the next row in the result set. When the cursor reaches the end of the result set, it will not be able to get the data, and a condition is raised. The handler is used to handle this condition.

To declare a NOT FOUND handler, you use the following syntax:

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;
```

The finished is a variable to indicate that the cursor has reached the end of the result set. Notice that the handler declaration must appear after variable and cursor declaration inside the stored procedures.

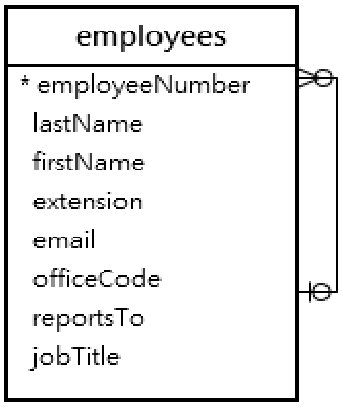
The following diagram illustrates how MySQL cursor works.



MySQL Cursor Example

Creating a stored procedure that creates an email list of all employees in the employees table in the sample database.

Aim: - Implementation of Cursor



First, declare some variables, a cursor for looping over the emails of employees, and a NOT FOUND handler:

```
        DECLARE finished INTEGER DEFAULT 0;
        DECLARE emailAddress varchar(100) DEFAULT "";
```

```
        -- declare cursor for employee email
        DEClARE curEmail
                CURSOR FOR
                        SELECT email FROM employees;
```

```
        -- declare NOT FOUND handler
        DECLARE CONTINUE HANDLER
    FOR NOT FOUND SET finished = 1;
```

Next, open the cursor by using the OPEN statement:

```
OPEN curEmail;
```

Then, iterate the email list, and concatenate all emails where each email is separated by a semicolon(;):

```
    getEmail: LOOP
                FETCH curEmail INTO emailAddress;
                IF finished = 1 THEN
                        LEAVE getEmail;
                END IF;
                -- build email list
                SET emailList = CONCAT(emailAddress,";",emailList);
        END LOOP getEmail;
```

After that, inside the loop, we used the finished variable to check if there is an email in the list to terminate the loop.

Finally, close the cursor using the CLOSE statement:

```
CLOSE email_cursor;
```

The createEmailList stored procedure is as follows:

```
DELIMITER $$
CREATE PROCEDURE createEmailList (
        INOUT emailList varchar(4000)
)
BEGIN
        DECLARE finished INTEGER DEFAULT 0;
        DECLARE emailAddress varchar(100) DEFAULT "";

        -- declare cursor for employee email
        DEClARE curEmail
                CURSOR FOR
                        SELECT email FROM employees;

        -- declare NOT FOUND handler
        DECLARE CONTINUE HANDLER
    FOR NOT FOUND SET finished = 1;

        OPEN curEmail;
        getEmail: LOOP
                FETCH curEmail INTO emailAddress;
                IF finished = 1 THEN
                        LEAVE getEmail;
                END IF;
                -- build email list
                SET emailList = CONCAT(emailAddress,";",emailList);
        END LOOP getEmail;
        CLOSE curEmail;

END$$
DELIMITER ;
```

You can test the createEmailList stored procedure using the following script:

```
SET @emailList = "";
CALL createEmailList(@emailList);
SELECT @emailList;
```

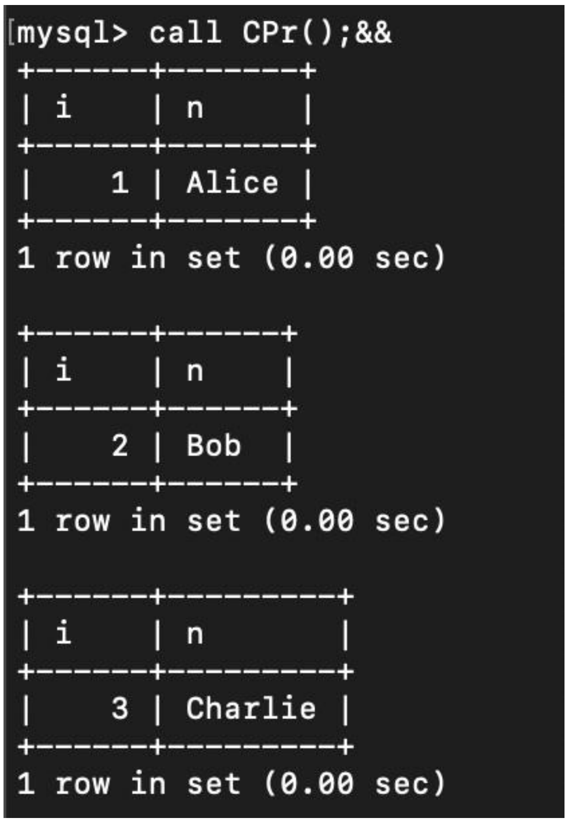
Database Management Systems Lab Experiment No: - 11

Aim: - Implementation of Cursor

Examples :

1)

```
DELIMITER &&
CREATE PROCEDURE CPr()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE i INT;
    DECLARE n VARCHAR(100);
    DECLARE cursi CURSOR FOR SELECT roll_no, fname FROM Student;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    OPEN cursi;
    read_loop: LOOP
        FETCH cursi INTO i, n;
        IF done = 1 THEN
            LEAVE read_loop;
        END IF;
        SELECT i, n;
    END LOOP read_loop;
    CLOSE cursi;
END &&
DELIMITER ;
```



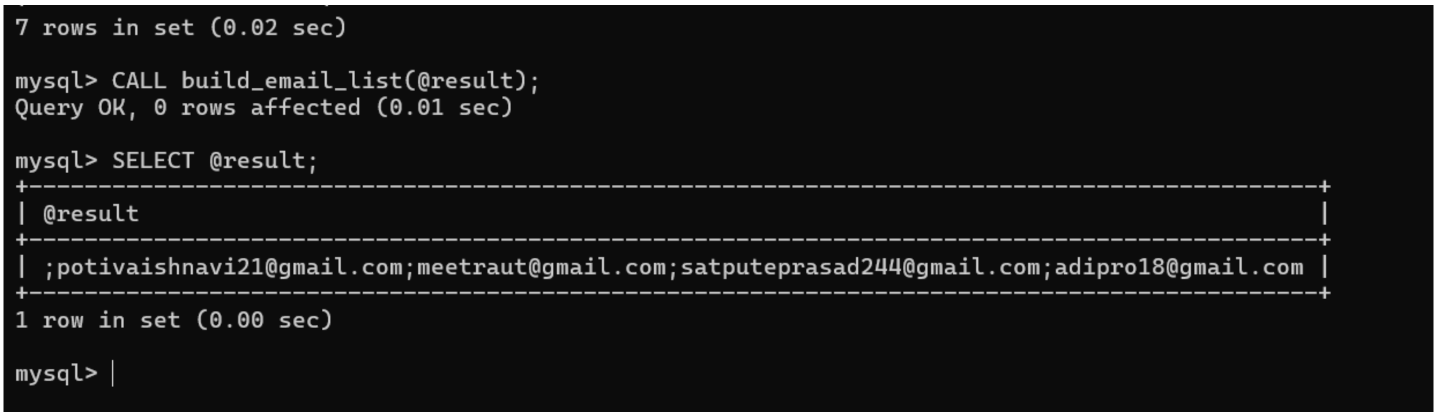
```
mysql> call CPr();&&
+------+-------+
| i    | n     |
+------+-------+
|    1 | Alice |
+------+-------+
1 row in set (0.00 sec)

+------+------+
| i    | n    |
+------+------+
|    2 | Bob  |
+------+------+
1 row in set (0.00 sec)

+------+---------+
| i    | n       |
+------+---------+
|    3 | Charlie |
+------+---------+
1 row in set (0.00 sec)
```

2)

```
DELIMITER $$
CREATE PROCEDURE build_email_list (INOUT email_list VARCHAR(4000))
BEGIN
  DECLARE finished INTEGER DEFAULT 0;
  DECLARE v_email VARCHAR(4000) DEFAULT "";
  DECLARE emailAddress VARCHAR(100);
  DECLARE email_cursor CURSOR FOR
    SELECT email FROM person;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;
  OPEN email_cursor;
  getEmail: LOOP
    FETCH email_cursor INTO emailAddress;
    IF finished = 1 THEN
      LEAVE getEmail;
    END IF;
    SET v_email = CONCAT(v_email, ";", emailAddress);
  END LOOP getEmail;
  CLOSE email_cursor;
  SET email_list = v_email;
END$$
DELIMITER ;
```



```
7 rows in set (0.02 sec)

mysql> CALL build_email_list(@result);
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT @result;
+----------------------------------------------------------------------------------------------+
| @result                                                                                      |
+----------------------------------------------------------------------------------------------+
| ;potivaishnavi21@gmail.com;meetraut@gmail.com;satputeprasad244@gmail.com;adipro18@gmail.com |
+----------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> |
```

Conclusion:- LO mapped LO3, LO4.