DBMS
Experiment 9

Name: Ayush Jain **Sapid**: 60004200132

Div./Batch: B/B1 **Branch**: Computer Engineering

Aim: Write Functions, cursor and procedure.

Theory: A procedure (often called a stored procedure) is a subroutine like a subprogram in a regular computing language, stored in database. A procedure has a name, a parameter list, and SQL statement(s). All most all relational database system supports stored procedure, MySQL 5 introduce stored procedure.MySQL 5.6 supports "routines" and there are two kinds of routines : stored procedures which you call, or functions whose return values you use in other SQL statements the same way that you use pre-installed MySQL functions like pi(). The major difference is that UDFs can be used like any other expression within SQL statements, whereas stored procedures must be invoked using the CALL statement

Create a procedure:

```
CREATE [DEFINER = { user | CURRENT_USER }]
PROCEDURE sp_name ([proc_parameter[,...]])
[characteristic ...] routine_body
proc_parameter: [ IN | OUT | INOUT ] param_name type
type:
Any valid MySQL data type
characteristic:
COMMENT 'string'
| LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA
| MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
routine_body:
```

Pick a Delimiter : The delimiter is the character or string of characters which is used to complete an SQL statement. By default we use semicolon (;) as a delimiter. But this causes problem in stored procedure because a procedure can have many statements, and everyone must end with a semicolon. So for your delimiter, pick a string which is rarely occur within statement or within procedure. Here we have used double dollar sign i.e. $$.You can use whatever you want. To resume using ";" as a delimiter later, say "DELIMITER ; $$". See here how to change the delimiter :

```
mysql> DELIMITER $$ ;
```

Now the default DELIMITER is "$$". Let execute a simple SQL command :

mysql> SELECT * FROM user $$

Example : MySQL Procedure

mysql> DELIMITER $$;mysql> CREATE PROCEDURE job_data()
> SELECT * FROM JOBS; $$

Valid SQL routine statement

Here we have created a simple procedure called job_data, when we will execute the procedure it will display all the data from "jobs" tables.

mysql> DELIMITER $$;mysql> CREATE PROCEDURE job_data()
> SELECT * FROM JOBS; $$

Call a procedure

The CALL statement is used to invoke a procedure that is stored in a DATABASE. Here is the syntax :

CALL job_data()

MySQL Procedure : Parameter IN example

In the following procedure, we have used a IN parameter 'var1' (type integer) which accept a number from the user. Within the body of the procedure, there is a SELECT statement which fetches rows from 'jobs' table and the number of rows will be supplied by the user. Here is the procedure :

mysql> CREATE PROCEDURE my_proc_IN (IN var1 INT)
-> BEGIN
-> SELECT * FROM jobs LIMIT var1;
-> END$$
Query OK, 0 rows affected (0.00 sec)
To execute the first 2 rows from the 'jobs' table execute the following command :

mysql> CALL my_proc_in(2)$$
+----------+-----------------------------------+------------+------------+
| JOB_ID | JOB_TITLE | MIN_SALARY | MAX_SALARY |
+----------+-----------------------------------+------------+------------+
| AD_PRES | President | 20000 | 40000 |
| AD_VP | Administration Vice President | 15000 | 30000 |
+----------+-----------------------------------+------------+------------+
2 rows in set (0.00 sec)Query OK, 0 rows affected (0.03 sec)

Now execute the first 5 rows from the 'jobs' table :

```
mysql>
CALL my_proc_in(5)$$
+---------+-----------------------------------+------------+------------+
| JOB_ID     | JOB_TITLE                    | MIN_SALARY | MAX_SALARY |
+---------+-----------------------------------+------------+------------+
| AD_PRES    | President                    |      20000 |      40000 |
| AD_VP      | Administration Vice President |      15000 |      30000 |
| AD_ASST    | Administration Assistant      |       3000 |       6000 |
| FI_MGR     | Finance Manager              |       8200 |      16000 |
| FI_ACCOUNT | Accountant                   |       4200 |       9000 |
+---------+-----------------------------------+------------+------------+
5 rows in set (0.00 sec)Query OK, 0 rows affected (0.05 sec)
```

MySQL Procedure : Parameter OUT example

The following example shows a simple stored procedure that uses an OUT parameter. Within the procedure MySQL MAX() function retrieves maximum salary from MAX_SALARY of jobs table.

```
mysql> CREATE PROCEDURE my_proc_OUT (OUT highest_salary INT)
-> BEGIN
-> SELECT MAX(MAX_SALARY) INTO highest_salary FROM JOBS;
-> END$$
Query OK, 0 rows affected (0.00 sec)
```

In the body of the procedure, the parameter will get the highest salary from MAX_SALARY column. After calling the procedure the word OUT tells the DBMS that the value goes out from the procedure. Here highest_salary is the name of the output parameter and we have passedits value to a session variable named @M, in the CALL statement.

```
mysql> CALL my_proc_OUT(@M)$$
Query OK, 1 row affected (0.03 sec)

mysql< SELECT @M$$+ ------+
| @M    |
+---------+
| 40000 |
+---------+
1 row in set (0.00 sec)
```

MySQL Procedure : Parameter INOUT example

The following example shows a simple stored procedure that uses an INOUT parameter and an IN parameter. The user will supply 'M' or 'F' through IN parameter (emp_gender) to count

a number of male or female from user_details table. The INOUT parameter (mfgender) will return the result to a user. Here is the code and output of the procedure :

```
mysql> CALL my_proc_OUT(@M)$$Query OK, 1 row affected (0.03 sec)mysql> CREATE
PROCEDURE my_proc_INOUT (INOUT mfgender INT, IN emp_gender CHAR(1))
-> BEGIN
-> SELECT COUNT(gender) INTO mfgender FROM user_details WHERE gender =
emp_gender;
-> END$$
Query OK, 0 rows affected (0.00 sec)
```

Now check the number of **male** and **female** users of the said tables :

```
mysql> CALL my_proc_INOUT(@C,'M')$$
Query OK, 1 row affected (0.02 sec)

mysql> SELECT @C$$
+.....+
| @C   |
+------+
|   3 |
+.....+
1 row in set (0.00 sec)

mysql> CALL my_proc_INOUT(@C,'F')$$
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @C$$
+.....+
| @C   |
+------+
|   1 |
+.....+
1 row in set (0.00 sec)
```

MYSQL: Creating stored function:

The CREATE FUNCTION statement is used for creating a stored function and user-defined functions. A stored function is a set of SQL statements that perform some operation and return a single value.

Just like Mysql in-built function, it can be called from within a Mysql statement.

By default, the stored function is associated with the default database.

The CREATE FUNCTION statement require CREATE ROUTINE database privilege.

Syntax:

The syntax for CREATE FUNCTION statement in Mysql is:

```
CREATE FUNCTION function_name(func_parameter1, func_parameter2, ..)
        RETURN datatype [characteristics]
```

func_body

Example:

```
DELIMITER //

CREATE FUNCTION no_of_years(date1 date) RETURNS int DETERMINISTIC
BEGIN
 DECLARE date2 DATE;
  Select current_date()into date2;
RETURN year(date2)-year(date1);
END
//
DELIMITER ;
```

Calling of above function:

```
Select emp_id, fname, lname, no_of_years(start_date) as 'years' from employee;
```

MySQL: Cursors

A database cursor is a control structure that enables traversal over the records in a database. Cursors are used by database programmers to process individual rows returned by database system queries. Cursors enable manipulation of whole result sets at once. In this scenario, a cursor enables the rows in a result set to be processed sequentially. In SQL procedures, a cursor makes it possible to define a result set (a set of data rows) and perform complex logic on a row by row basis. By using the same mechanics, an SQL procedure can also define a result set and return it directly to the caller of the SQL procedure or to a client application.

MySQL supports cursors inside stored programs. The syntax is as in embedded SQL. Cursors have these properties :

- Asensitive: The server may or may not make a copy of its result table
- Read only: Not updatable
- Nonscrollable: Can be traversed only in one direction and cannot skip rows

To use cursors in MySQL procedures, you need to do the following :

- Declare a cursor.
- Open a cursor.
- Fetch the data into variables.
- Close the cursor when done.

Declare a cursor:

The following statement declares a cursor and associates it with a SELECT statement that retrieves the rows to be traversed by the cursor.

```
DECLARE cursor_name
CURSOR FOR select_statement
```

Open a cursor:

The following statement opens a previously declared cursor.

```
OPEN cursor_name
```

Fetch the data into variables :

This statement fetches the next row for the SELECT statement associated with the specified cursor (which must be open) and advances the cursor pointer. If a row exists, the fetched columns are stored in the named variables. The number of columns retrieved by the SELECT statement must match the number of output variables specified in the FETCH statement.

```
FETCH [[NEXT] FROM] cursor_name
INTO var_name [, var_name] ...
```

Close the cursor when done :

This statement closes a previously opened cursor. An error occurs if the cursor is not open.

```
CLOSE cursor_name
```

Example:

The procedure starts with three variable declarations. Incidentally, the order is important. First, declare variables. Then declare conditions. Then declare cursors. Then, declare handlers. If you put them in the wrong order, you will get an error message.

```
DELIMITER $$
CREATE PROCEDURE my_procedure_cursors(INOUT return_val INT)
BEGIN
DECLARE a,b INT;
DECLARE cur_1 CURSOR FOR
SELECT max_salary FROM jobs;
DECLARE CONTINUE HANDLER FOR NOT FOUNDSET b = 1;
OPEN cur_1;REPEATFETCH cur_1 INTO a;
UNTIL b = 1END REPEAT;
CLOSE cur_1;
SET return_val = a;
END;
$$
```

Now execute the procedure:

```
mysql>
CALL my_procedure_cursors(@R);
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT @R;
+-------+
| @R    |
+-------+
| 10500 |
+-------+
1 row in set (0.00 sec)
```

Performed Queries and screenshots: -

**********Procedures**********

1) Empty Parameter

delimiter $$
create procedure get_all_artist()
select * from
artist $$

```
mysql> delimiter $$
mysql> create procedure get_all_artist()
    -> select * from
    -> artist $$
Query OK, 0 rows affected (0.50 sec)
```

call get_all_artist() $$

```
mysql> call get_all_artist() $$
+-----------+-----------------+--------+
| artist_id | artist_name     | rating |
+-----------+-----------------+--------+
|         3 | Justin Bieber   |      5 |
|         4 | Ed Sheeran      |      9 |
|         5 | Doja Cat        |      7 |
|         7 | Juice WRLD      |      8 |
|         9 | Adele           |     10 |
|        10 | Lil Nas X       |      6 |
|        11 | Mythpat         |      9 |
|        12 | Triggered Insaan |      8 |
|        15 | Martin Garix    |      2 |
+-----------+-----------------+--------+
9 rows in set (0.05 sec)

Query OK, 0 rows affected (0.09 sec)
```

2) IN Parameter

create procedure get_some_artist(in var1 int)
select * from artist limit var1$$

```
mysql> create procedure get_some_artist(in var1 int)
    -> select * from artist limit var1$$
Query OK, 0 rows affected (0.19 sec)
```

call get_some_artist(4) $$

```
mysql> call get_some_artist(4) $$
+-----------+--------------+--------+
| artist_id | artist_name  | rating |
+-----------+--------------+--------+
|         3 | Justin Bieber |      5 |
|         4 | Ed Sheeran    |      9 |
|         5 | Doja Cat      |      7 |
|         7 | Juice WRLD    |      8 |
+-----------+--------------+--------+
4 rows in set (0.00 sec)

Query OK, 0 rows affected (0.04 sec)
```

call get_some_artist(7) $$

```
mysql> call get_some_artist(7) $$
+-----------+---------------+--------+
| artist_id | artist_name   | rating |
+-----------+---------------+--------+
|         3 | Justin Bieber |      5 |
|         4 | Ed Sheeran    |      9 |
|         5 | Doja Cat      |      7 |
|         7 | Juice WRLD    |      8 |
|         9 | Adele         |     10 |
|        10 | Lil Nas X     |      6 |
|        11 | Mythpat       |      9 |
+-----------+---------------+--------+
7 rows in set (0.00 sec)

Query OK, 0 rows affected (0.05 sec)
```

3) OUT Parameter

create procedure get_highest_artist_rating(out highest_rating int)
select max(rating) into highest_rating from artist$$

```
mysql> create procedure get_highest_artist_rating(out highest_rating int)
    -> select max(rating) into highest_rating from artist$$
Query OK, 0 rows affected (0.18 sec)
```

call get_highest_artist_rating(@Top_artist_rating) $$

select @Top_artist_rating $$

```
mysql> call get_highest_artist_rating(@Top_artist_rating) $$
Query OK, 1 row affected (0.00 sec)

mysql> select @Top_artist_rating $$
+--------------------+
| @Top_artist_rating |
+--------------------+
|                 10 |
+--------------------+
1 row in set (0.00 sec)
```

create procedure get_lowest_artist_rating(out lowest_rating int)
select min(rating) into lowest_rating from artist$$

```
mysql> create procedure get_lowest_artist_rating(out lowest_rating int)
    -> select min(rating) into lowest_rating from artist$$
Query OK, 0 rows affected (0.14 sec)
```

call get_lowest_artist_rating(@Lowest_artist_rating) $$

select @Lowest_artist_rating $$

```
mysql> call get_lowest_artist_rating(@Lowest_artist_rating) $$
Query OK, 1 row affected (0.04 sec)

mysql> select @Lowest_artist_rating $$
+----------------------+
| @Lowest_artist_rating |
+----------------------+
|                    2 |
+----------------------+
1 row in set (0.00 sec)
```

4) INOUT Parameter

create procedure count_rating(inout count int,in artist_rating int)
select count(rating) into count from artist where rating=artist_rating $$

```
mysql> create procedure count_rating(inout count int,in artist_rating int)
    -> select count(rating) into count from artist where rating=artist_rating $$
Query OK, 0 rows affected (0.15 sec)
```

call count_rating(@C,8)$$

select @C $$

```
mysql> call count_rating(@C,8)$$
Query OK, 1 row affected (0.04 sec)

mysql> select @C $$
+------+
| @C   |
+------+
|    2 |
+------+
1 row in set (0.00 sec)
```

**********Functions**********
create function rating_required_for_top_position(curr_rating int)returns int deterministic
return 10-curr_rating$$

```
mysql> create function rating_required_for_top_position(curr_rating int)returns int deterministic
    -> return 10-curr_rating$$
Query OK, 0 rows affected (0.18 sec)
```

Select artist_id,artist_name,rating as current_rating,rating_required_for_top_position(rating) from artist$$

```
mysql> Select artist_id,artist_name,rating as current_rating,rating_required_for_top_position(rating) from artist$$
+-----------+-----------------+---------------+-----------------------------------------+
| artist_id | artist_name     | current_rating | rating_required_for_top_position(rating) |
+-----------+-----------------+---------------+-----------------------------------------+
|         3 | Justin Bieber   |             5 |                                       5 |
|         4 | Ed Sheeran      |             9 |                                       1 |
|         5 | Doja Cat        |             7 |                                       3 |
|         7 | Juice WRLD      |             8 |                                       2 |
|         9 | Adele           |            10 |                                       0 |
|        10 | Lil Nas X       |             6 |                                       4 |
|        11 | Mythpat         |             9 |                                       1 |
|        12 | Triggered Insaan |             8 |                                       2 |
|        15 | Martin Garix    |             2 |                                       8 |
+-----------+-----------------+---------------+-----------------------------------------+
9 rows in set (0.05 sec)
```

Conclusion: Procedure and functions are implemented with cursor.