CS 207: Applied Database Practicum
Week 5

Varun Dutt

School of Computing and Electrical Engineering
School of Humanities and Social Sciences
Indian Institute of Technology Mandi, India

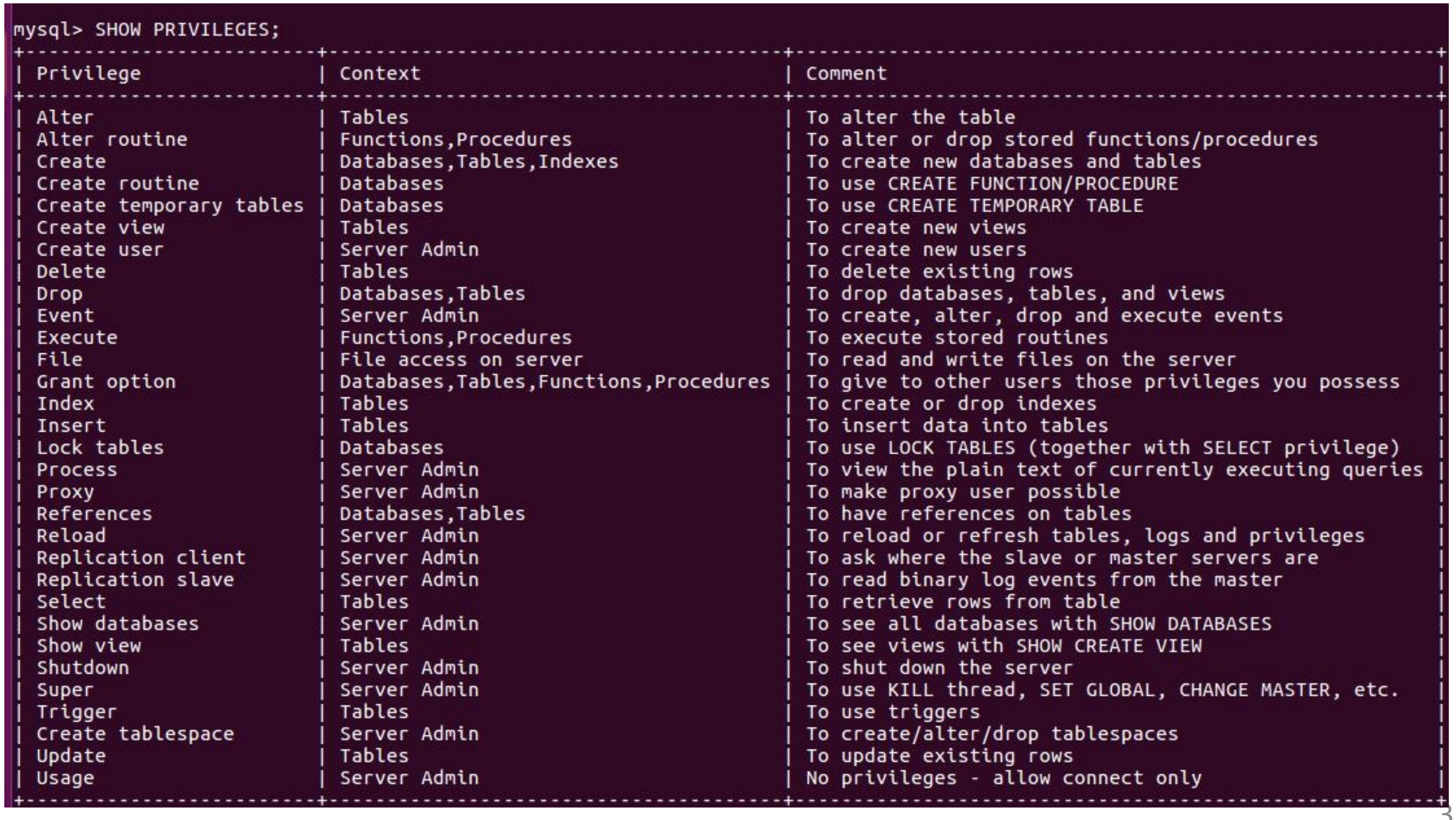
Scaling the Heights

MySQL – Stored Procedure

- A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.
- A stored procedure has three main parts:
  - **Input**: Store procedure can accept parameter values as inputs. Depending on how the parameters are defined, modified values can be passed back to the calling program
  - **Execution**: Stored procedures can execute SQL statements, utilize conditional logic such as IF THEN or CASE statements and lopping constructs to perform tasks.
  - **Outputs**: A stored procedure can return a single values such as a number or text value or a result set.

Stored Procedure

- For creating and executing a stored procedure the user should have “**create routine”** privilege



```
mysql> SHOW PRIVILEGES;
+-------------------------+---------------------------------------+-------------------------------------------------------+
| Privilege               | Context                               | Comment                                               |
+-------------------------+---------------------------------------+-------------------------------------------------------+
| Alter                   | Tables                                | To alter the table                                    |
| Alter routine           | Functions,Procedures                  | To alter or drop stored functions/procedures          |
| Create                  | Databases,Tables,Indexes              | To create new databases and tables                    |
| Create routine          | Databases                             | To use CREATE FUNCTION/PROCEDURE                      |
| Create temporary tables | Databases                             | To use CREATE TEMPORARY TABLE                         |
| Create view             | Tables                                | To create new views                                   |
| Create user             | Server Admin                          | To create new users                                   |
| Delete                  | Tables                                | To delete existing rows                               |
| Drop                    | Databases,Tables                      | To drop databases, tables, and views                  |
| Event                   | Server Admin                          | To create, alter, drop and execute events             |
| Execute                 | Functions,Procedures                  | To execute stored routines                            |
| File                    | File access on server                 | To read and write files on the server                 |
| Grant option            | Databases,Tables,Functions,Procedures | To give to other users those privileges you possess   |
| Index                   | Tables                                | To create or drop indexes                             |
| Insert                  | Tables                                | To insert data into tables                            |
| Lock tables             | Databases                             | To use LOCK TABLES (together with SELECT privilege)   |
| Process                 | Server Admin                          | To view the plain text of currently executing queries |
| Proxy                   | Server Admin                          | To make proxy user possible                           |
| References              | Databases,Tables                      | To have references on tables                          |
| Reload                  | Server Admin                          | To reload or refresh tables, logs and privileges      |
| Replication client      | Server Admin                          | To ask where the slave or master servers are          |
| Replication slave       | Server Admin                          | To read binary log events from the master             |
| Select                  | Tables                                | To retrieve rows from table                           |
| Show databases          | Server Admin                          | To see all databases with SHOW DATABASES              |
| Show view               | Tables                                | To see views with SHOW CREATE VIEW                    |
| Shutdown                | Server Admin                          | To shut down the server                               |
| Super                   | Server Admin                          | To use KILL thread, SET GLOBAL, CHANGE MASTER, etc.   |
| Trigger                 | Tables                                | To use triggers                                       |
| Create tablespace       | Server Admin                          | To create/alter/drop tablespaces                      |
| Update                  | Tables                                | To update existing rows                               |
| Usage                   | Server Admin                          | No privileges - allow connect only                    |
+-------------------------+---------------------------------------+-------------------------------------------------------+
```

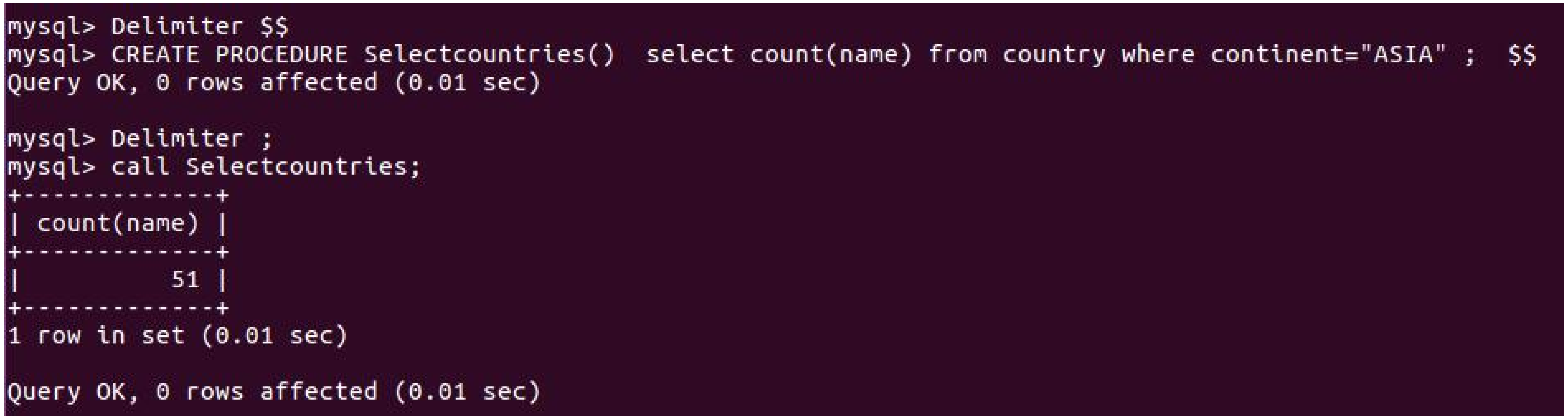
Stored Procedure Syntax

PROCEDURE *procedure_name* ([proc_parameter[,...]])
proc_parameter: [*IN | OUT | INOUT*] param_name type
type:
Any valid MySQL data type
routine_body:
Valid SQL routine statement

Executing a Stored Procedure:
CALL *procedure_name;*

Deleting a Stored Procedure:
DROP PROCEDURE *procedure_name;*

Stored Procedure:Example

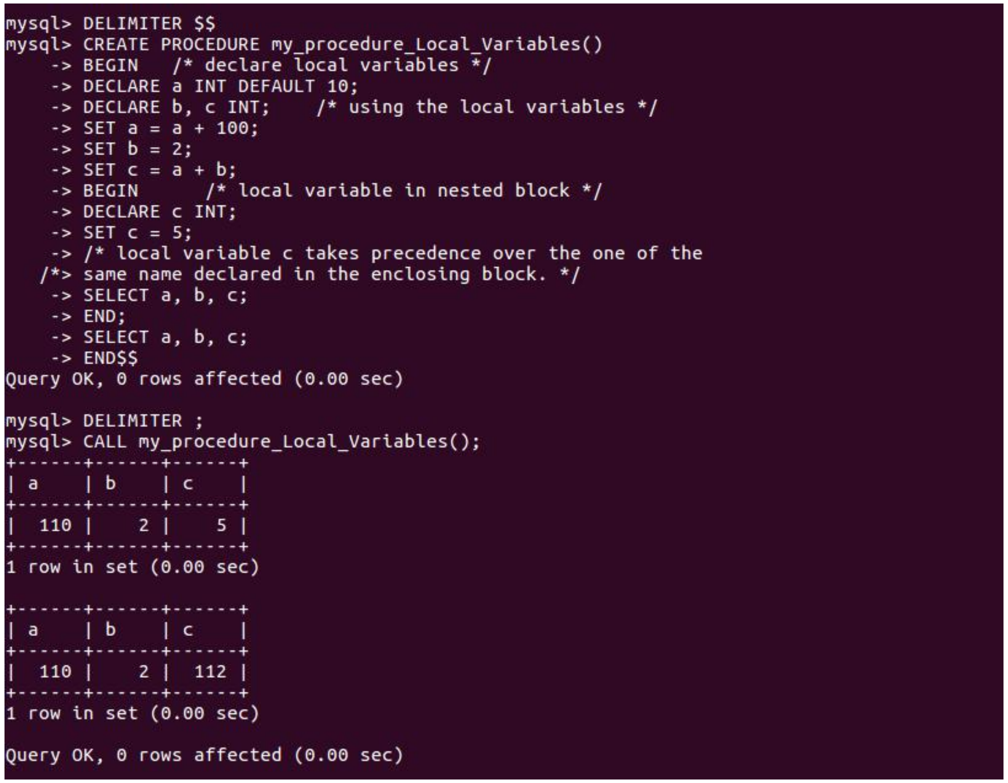


```
mysql> Delimiter $$
mysql> CREATE PROCEDURE Selectcountries()  select count(name) from country where continent="ASIA" ;  $$
Query OK, 0 rows affected (0.01 sec)

mysql> Delimiter ;
mysql> call Selectcountries;
+-------------+
| count(name) |
+-------------+
|          51 |
+-------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

Delimiter is used to tell that statement ending with ‘delimiter’ denotes the end of one statement. Here, the use of delimiter is necessary to so that the MySQL treats the statements inside (BEGIN...END) as part of one statement.

Stored Procedure: Compound Statement

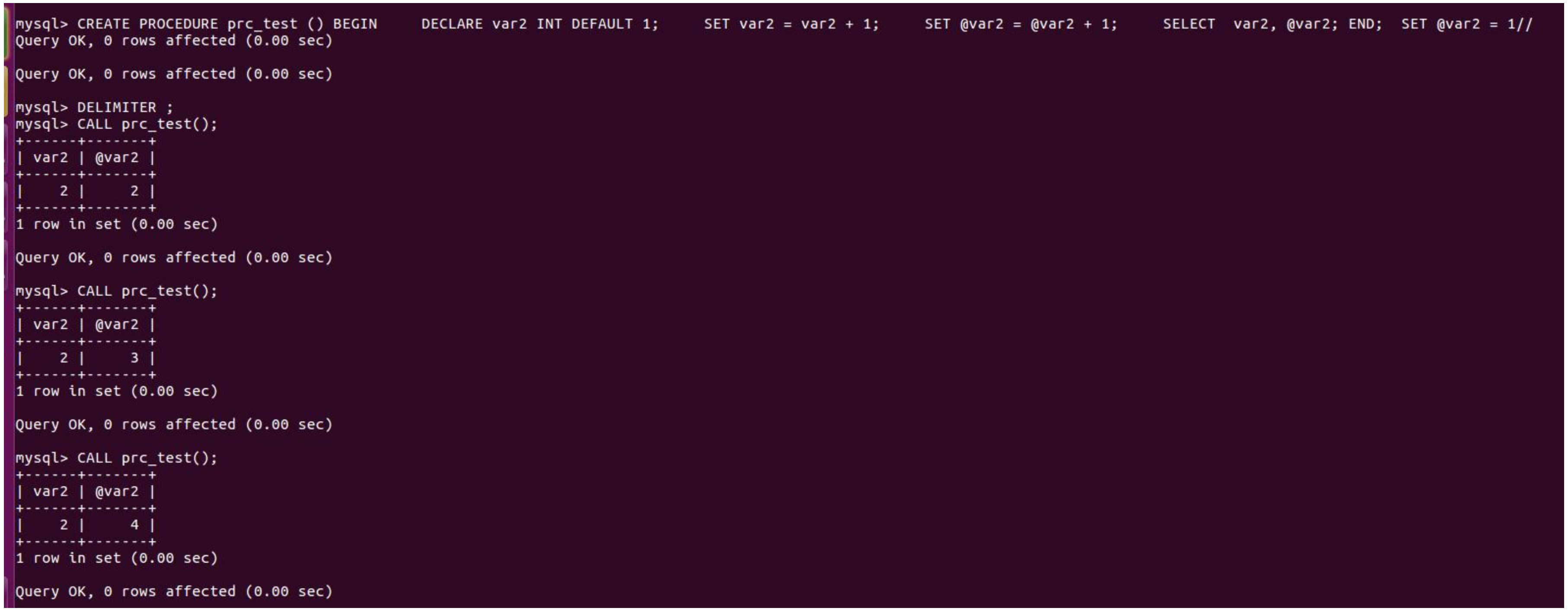
- BEGIN ... END block is used to write compound statements, i.e. when you need more than one statement within stored programs (e.g. stored procedures, functions, triggers, and events)



```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE my_procedure_Local_Variables()
    -> BEGIN   /* declare local variables */
    -> DECLARE a INT DEFAULT 10;
    -> DECLARE b, c INT;    /* using the local variables */
    -> SET a = a + 100;
    -> SET b = 2;
    -> SET c = a + b;
    -> BEGIN      /* local variable in nested block */
    -> DECLARE c INT;
    -> SET c = 5;
    -> /* local variable c takes precedence over the one of the
   /*> same name declared in the enclosing block. */
    -> SELECT a, b, c;
    -> END;
    -> SELECT a, b, c;
    -> END$$
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> CALL my_procedure_Local_Variables();
+------+------+------+
| a    | b    | c    |
+------+------+------+
|  110 |    2 |    5 |
+------+------+------+
1 row in set (0.00 sec)

+------+------+------+
| a    | b    | c    |
+------+------+------+
|  110 |    2 |  112 |
+------+------+------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

Stored Procedure: User Variables

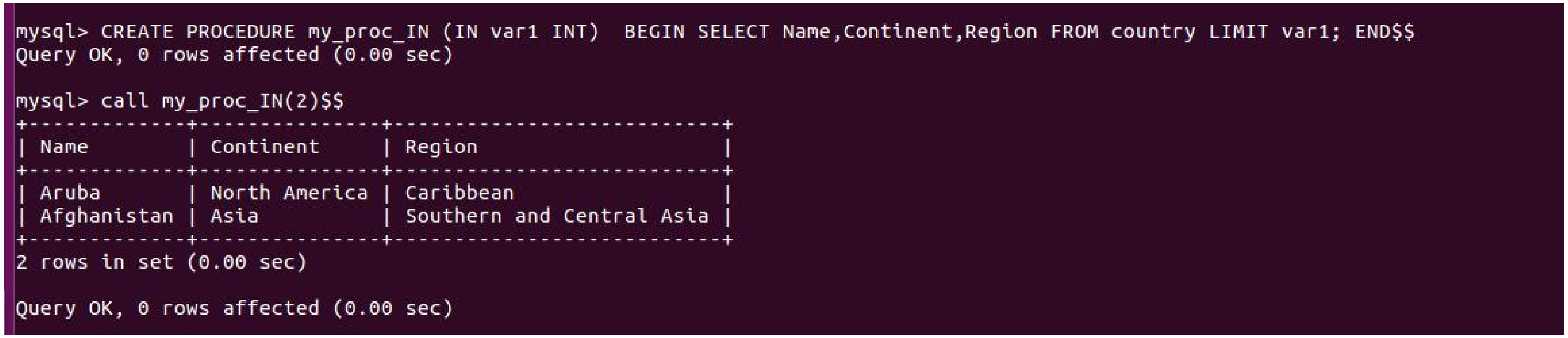
In MySQL stored procedures, user variables are referenced with an ampersand (@) prefixed to the user variable name (for example, @x and @y).


```
mysql> CREATE PROCEDURE prc_test () BEGIN     DECLARE var2 INT DEFAULT 1;     SET var2 = var2 + 1;     SET @var2 = @var2 + 1;     SELECT  var2, @var2; END;  SET @var2 = 1//
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> CALL prc_test();
+------+-------+
| var2 | @var2 |
+------+-------+
|    2 |     2 |
+------+-------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> CALL prc_test();
+------+-------+
| var2 | @var2 |
+------+-------+
|    2 |     3 |
+------+-------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> CALL prc_test();
+------+-------+
| var2 | @var2 |
+------+-------+
|    2 |     4 |
+------+-------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

The difference between a procedure variable and a session-specific user-defined variable is that procedure variable is reinitialized to NULL each time the procedure is called, while the session-specific variable is not

Parameters:IN

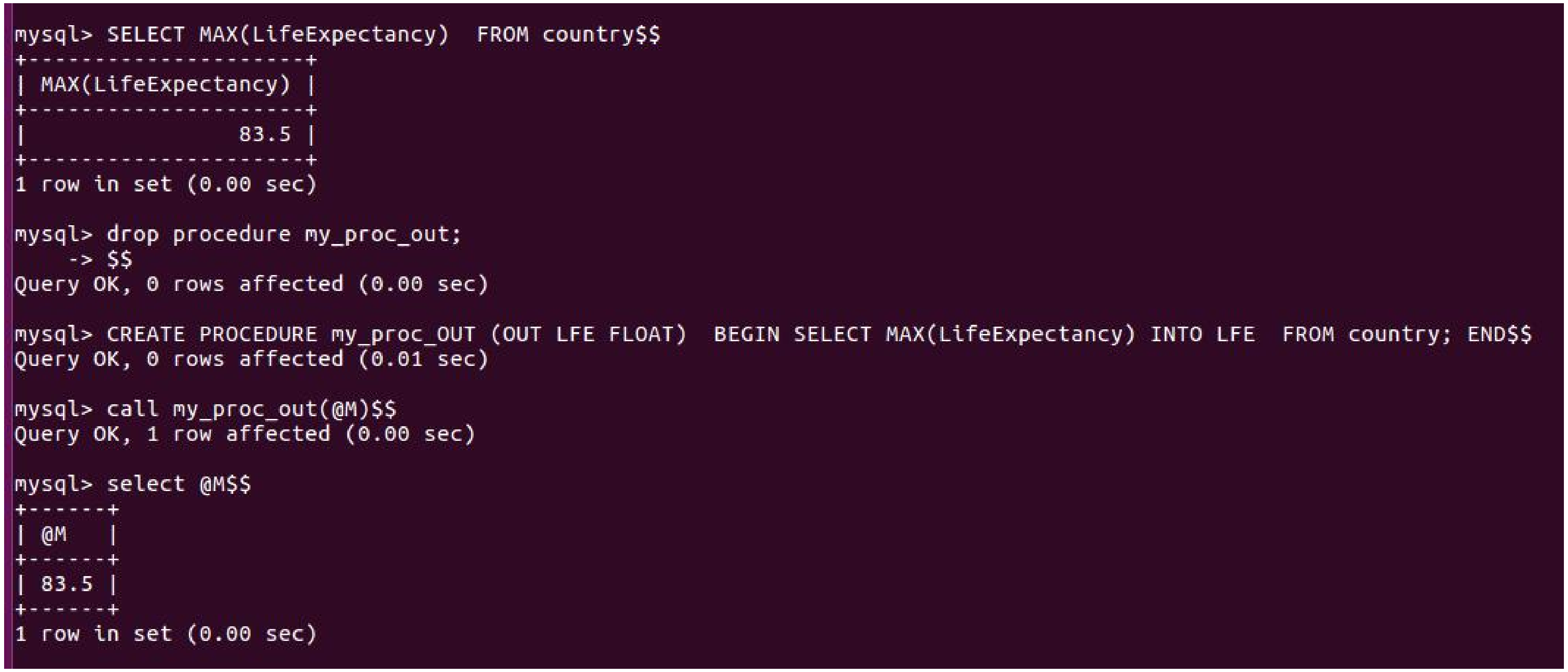
In the following procedure, we have used a IN parameter 'var1' (type integer) which accept a number from the user. Within the body of the procedure, there is a SELECT statement which fetches rows from 'country' table and the number of rows will be supplied by the user.



```
mysql> CREATE PROCEDURE my_proc_IN (IN var1 INT)  BEGIN SELECT Name,Continent,Region FROM country LIMIT var1; END$$
Query OK, 0 rows affected (0.00 sec)

mysql> call my_proc_IN(2)$$
+-------------+---------------+---------------------------+
| Name        | Continent     | Region                    |
+-------------+---------------+---------------------------+
| Aruba       | North America | Caribbean                 |
| Afghanistan | Asia          | Southern and Central Asia |
+-------------+---------------+---------------------------+
2 rows in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

Parameters:OUT

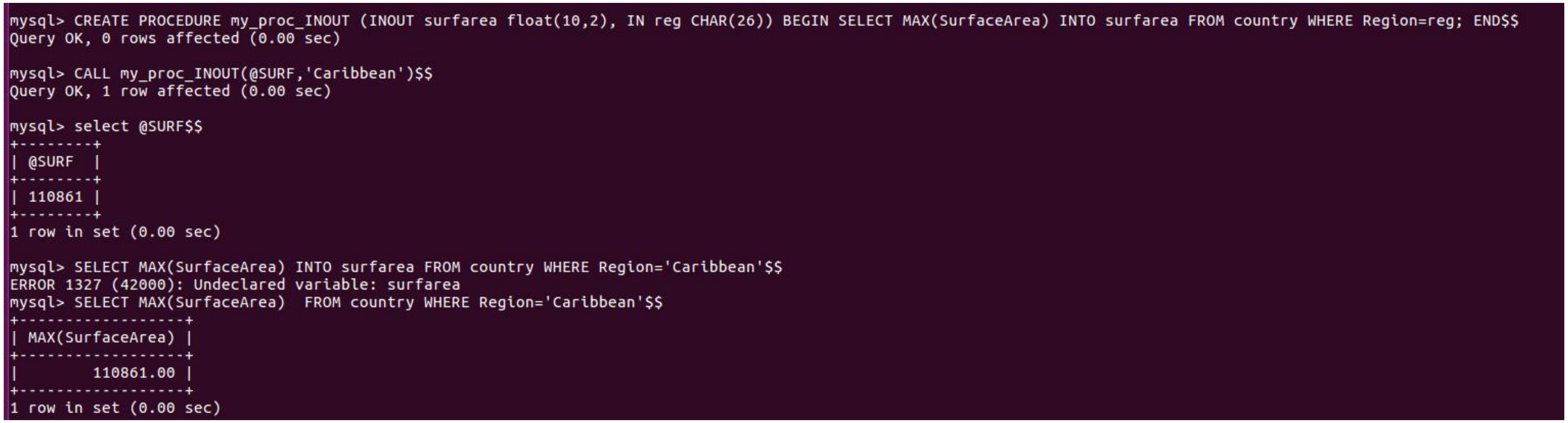
In the body of this procedure, the parameter will get the highest Life Expectancy from **LifeExpectancy** column. After calling the procedure the word OUT tells the DBMS that the value goes out from the procedure. Here LFE is the name of the output parameter and we have passed its value to a session variable named @M, in the CALL statement.



```
mysql> SELECT MAX(LifeExpectancy)  FROM country$$
+---------------------+
| MAX(LifeExpectancy) |
+---------------------+
|                83.5 |
+---------------------+
1 row in set (0.00 sec)

mysql> drop procedure my_proc_out;
    -> $$
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE PROCEDURE my_proc_OUT (OUT LFE FLOAT)  BEGIN SELECT MAX(LifeExpectancy) INTO LFE  FROM country; END$$
Query OK, 0 rows affected (0.01 sec)

mysql> call my_proc_out(@M)$$
Query OK, 1 row affected (0.00 sec)

mysql> select @M$$
+------+
| @M   |
+------+
| 83.5 |
+------+
1 row in set (0.00 sec)
```

Parameters:INOUT

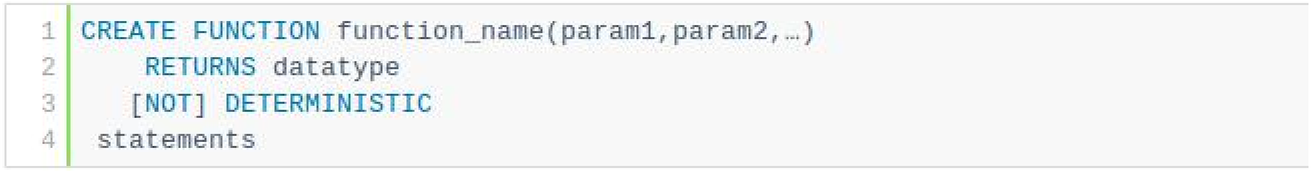
In this stored procedure that uses an INOUT parameter and an IN parameter. The user will supply a Region name through IN parameter (reg) to find the highest Surface Area in the region given by the user . The INOUT parameter (surfarea) will return the result to a user.



```
mysql> CREATE PROCEDURE my_proc_INOUT (INOUT surfarea float(10,2), IN reg CHAR(26)) BEGIN SELECT MAX(SurfaceArea) INTO surfarea FROM country WHERE Region=reg; END$$
Query OK, 0 rows affected (0.00 sec)

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

mysql> select @SURF$$
+--------+
| @SURF  |
+--------+
| 110861 |
+--------+
1 row in set (0.00 sec)

mysql> SELECT MAX(SurfaceArea) INTO surfarea FROM country WHERE Region='Caribbean'$$
ERROR 1327 (42000): Undeclared variable: surfarea
mysql> SELECT MAX(SurfaceArea)  FROM country WHERE Region='Caribbean'$$
+------------------+
| MAX(SurfaceArea) |
+------------------+
|        110861.00 |
+------------------+
1 row in set (0.00 sec)
```

MySQL Stored Functions

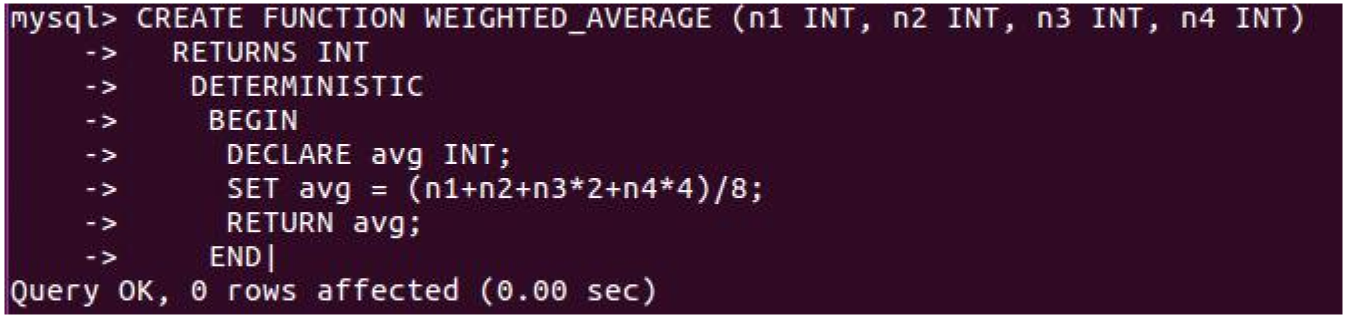
- A stored function is a special kind stored program that returns a single value.
- It is different from a stored procedure in the sense that we can use a stored function in SQL statements wherever an expression is used.
- Syntax



```
CREATE FUNCTION function_name(param1,param2,…)
    RETURNS datatype
  [NOT] DETERMINISTIC
 statements
```

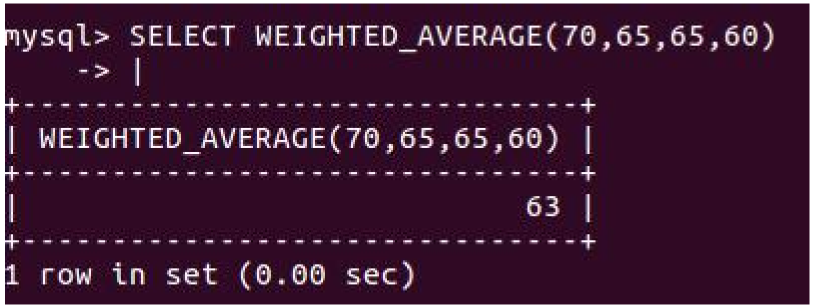
- Parameters of the stored function are listed inside the parentheses. By default, all parameters are the IN parameters. One cannot specify IN , OUT or INOUT modifiers to the parameters.

Stored Function example



```
mysql> CREATE FUNCTION WEIGHTED_AVERAGE (n1 INT, n2 INT, n3 INT, n4 INT)
    ->   RETURNS INT
    ->    DETERMINISTIC
    ->     BEGIN
    ->      DECLARE avg INT;
    ->      SET avg = (n1+n2+n3*2+n4*4)/8;
    ->      RETURN avg;
    ->     END|
Query OK, 0 rows affected (0.00 sec)
```

- Result:



```
mysql> SELECT WEIGHTED_AVERAGE(70,65,65,60)
    -> |
+-------------------------------+
| WEIGHTED_AVERAGE(70,65,65,60) |
+-------------------------------+
|                            63 |
+-------------------------------+
1 row in set (0.00 sec)
```

MySQL Cursors

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

- MySQL cursor has three properties:
  - **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.

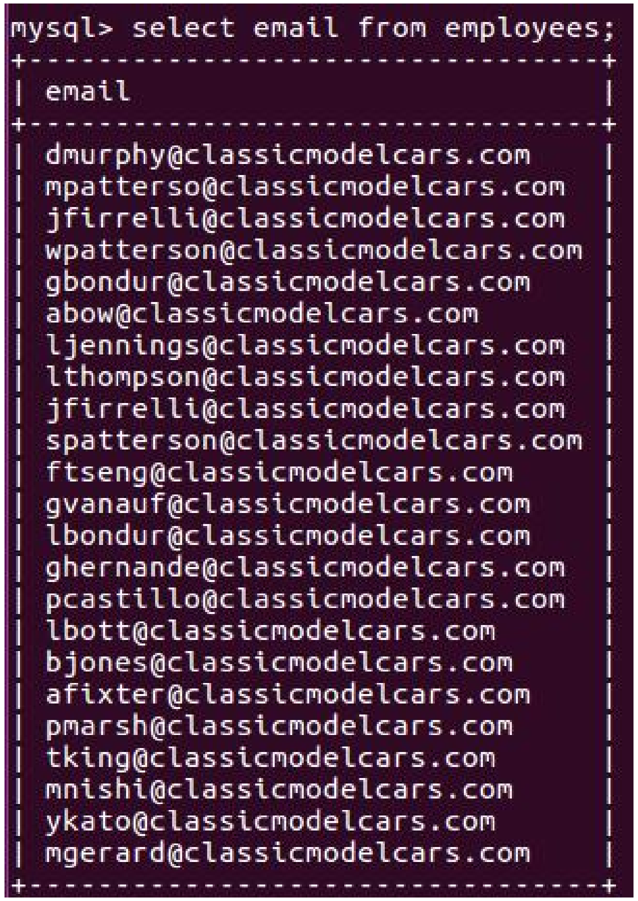
MySQL Cursors

- **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. An asensitive cursor performs faster than an insensitive cursor because it does not have to make a temporary copy of data. However, any changes 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.

MySQL Cursor Example

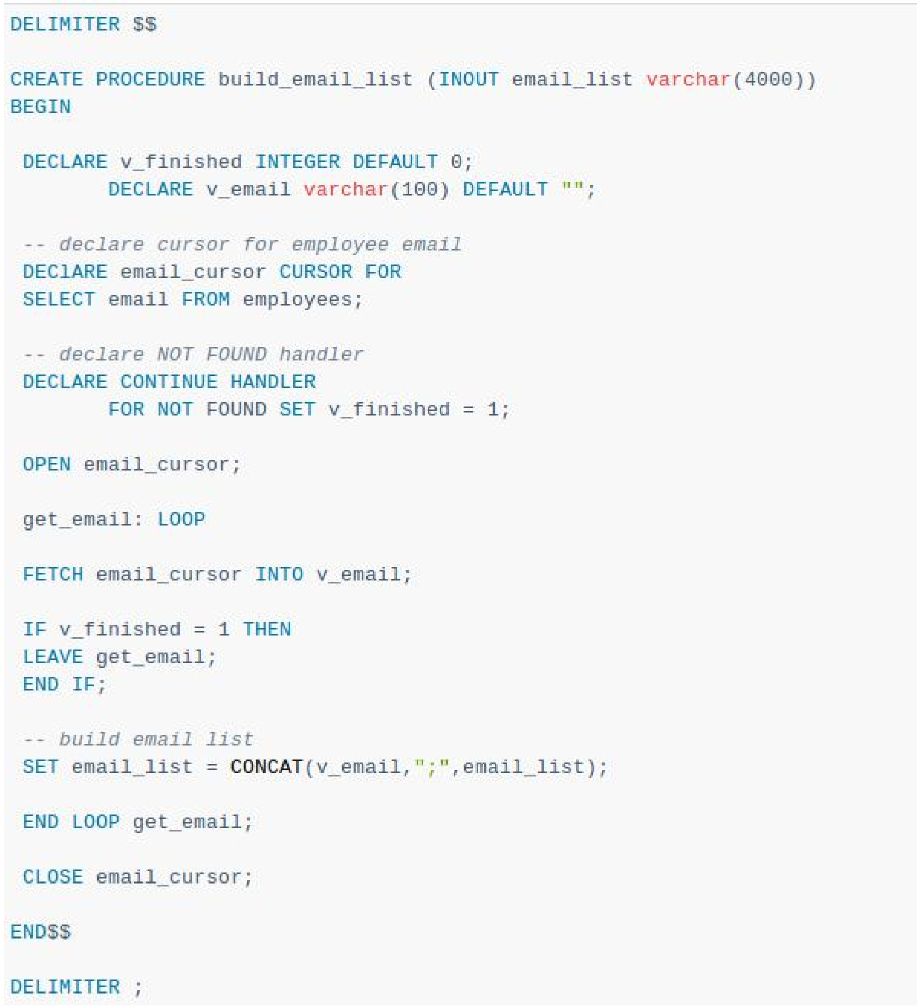
We are going to develop a stored procedure with a cursor that concatenates the email of all customers in the employees table:



```
mysql> select email from employees;
+---------------------------------+
| email                           |
+---------------------------------+
| dmurphy@classicmodelcars.com    |
| mpatterso@classicmodelcars.com  |
| jfirrelli@classicmodelcars.com  |
| wpatterson@classicmodelcars.com |
| gbondur@classicmodelcars.com    |
| abow@classicmodelcars.com       |
| ljennings@classicmodelcars.com  |
| lthompson@classicmodelcars.com  |
| jfirrelli@classicmodelcars.com  |
| spatterson@classicmodelcars.com |
| ftseng@classicmodelcars.com     |
| gvanauf@classicmodelcars.com    |
| lbondur@classicmodelcars.com    |
| ghernande@classicmodelcars.com  |
| pcastillo@classicmodelcars.com  |
| lbott@classicmodelcars.com      |
| bjones@classicmodelcars.com     |
| afixter@classicmodelcars.com    |
| pmarsh@classicmodelcars.com     |
| tking@classicmodelcars.com      |
| mnishi@classicmodelcars.com     |
| ykato@classicmodelcars.com      |
| mgerard@classicmodelcars.com    |
+---------------------------------+
```

MySQL Cursor Example

- The build_email_list stored procedure is as follows:



```
DELIMITER $$

CREATE PROCEDURE build_email_list (INOUT email_list varchar(4000))
BEGIN

 DECLARE v_finished INTEGER DEFAULT 0;
        DECLARE v_email varchar(100) DEFAULT "";

 -- declare cursor for employee email
 DECLARE email_cursor CURSOR FOR
 SELECT email FROM employees;

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

 OPEN email_cursor;

 get_email: LOOP

 FETCH email_cursor INTO v_email;

 IF v_finished = 1 THEN
 LEAVE get_email;
 END IF;

 -- build email list
 SET email_list = CONCAT(v_email,";",email_list);

 END LOOP get_email;

 CLOSE email_cursor;

END$$

DELIMITER ;
```

MySQL Cursor Example

- Declare a HANDLER variable and an email variable (for storing the email Ids)

```
DECLARE v_finished INTEGER DEFAULT 0;
        DECLARE v_email varchar(100) DEFAULT "";
```

- A cursor for looping over the email_address of customers, and a NOT FOUND handler:

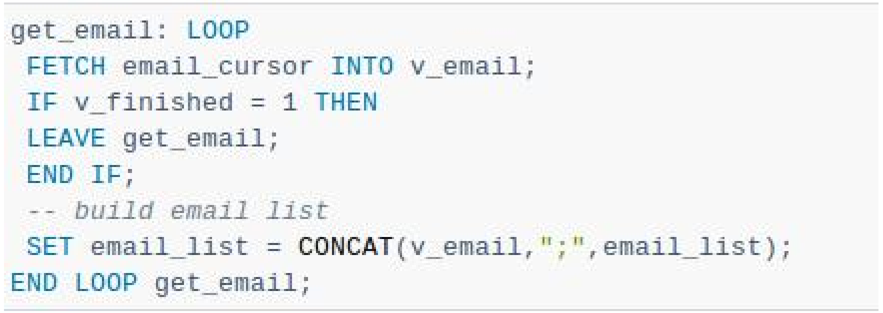
```
DECLARE finished INTEGER DEFAULT 0;
DECLARE email varchar(255) DEFAULT "";

-- declare cursor for employee email
DECLARE email_cursor CURSOR FOR
 SELECT email FROM employees;

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

Next, open the email_cursor by using the OPEN statement:

```
OPEN email_cursor;
```

MySQL Cursor Example

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



```
get_email: LOOP
 FETCH email_cursor INTO v_email;
 IF v_finished = 1 THEN
 LEAVE get_email;
 END IF;
 -- build email list
 SET email_list = CONCAT(v_email,";",email_list);
END LOOP get_email;
```

- After that, inside the loop we used the v_finished variable to check if there is an address in the list to terminate the loop.

MySQL Cursor Example

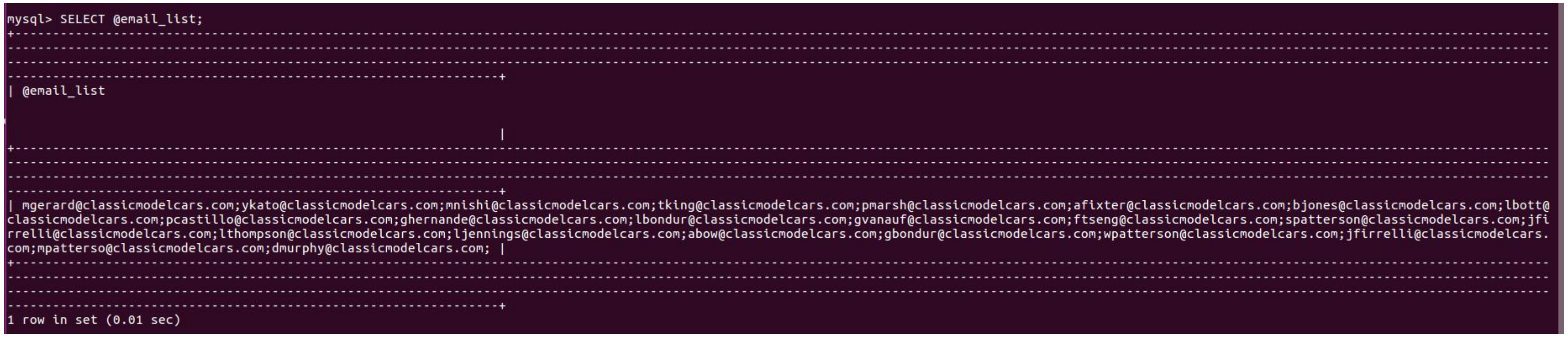
- Finally, close the cursor using the CLOSE statement:

```
CLOSE email_cursor;
```

- You can test the build_email_list stored procedure using the following script:

```
SET @email_list = "";
CALL build_email_list(@email_list);
SELECT @email_list;
```

MySQL Cursor Result



```
mysql> SELECT @email_list;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+
| @email_list

                                                                 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+
| mgerard@classicmodelcars.com;ykato@classicmodelcars.com;mnishi@classicmodelcars.com;tking@classicmodelcars.com;pmarsh@classicmodelcars.com;afixter@classicmodelcars.com;bjones@classicmodelcars.com;lbott@
classicmodelcars.com;pcastillo@classicmodelcars.com;ghernande@classicmodelcars.com;lbondur@classicmodelcars.com;gvanauf@classicmodelcars.com;ftseng@classicmodelcars.com;spatterson@classicmodelcars.com;jfi
rrelli@classicmodelcars.com;lthompson@classicmodelcars.com;ljennings@classicmodelcars.com;abow@classicmodelcars.com;gbondur@classicmodelcars.com;wpatterson@classicmodelcars.com;jfirrelli@classicmodelcars.
com;mpatterso@classicmodelcars.com;dmurphy@classicmodelcars.com; |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+
1 row in set (0.01 sec)
```

Using MySQL Cursor

- First, you have to 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 variables declaration, MySQL will issue an error. A cursor must always be associated with a SELECT statement.
- Next, you 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;

Using MySQL Cursor

- Then, you 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, you can check to see if there is any row available before fetching it.
- Finally, you call the CLOSE statement to deactivate the cursor and release the memory associated with it as follows:

CLOSE cursor_name;

- When the cursor is no longer used, you should close it.

NOT FOUND Handler

- 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.

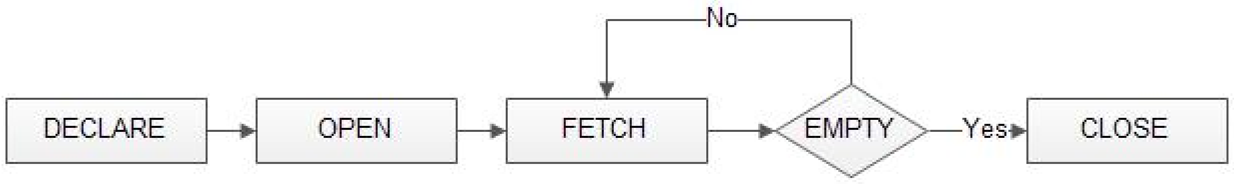
- To declare a NOT FOUND handler, you use the following syntax:

```
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.

Flow Diagram for MySQL Cursor

- The following diagram illustrates how MySQL cursor works.



References

https://dev.mysql.com/doc/refman/8.0/en/cursors.html