MySql Procedures

Stored Procedure in MySql:

“A stored routine is a set of SQL statements that can be stored in the server.”

A stored procedure (or, simply 'procedure') is a method to encapsulate repetitive tasks. They allow variable declarations, flow control and other useful programming techniques.

Here, we are introducing the procedure by using very simple examples and table manipulation for understanding. There is a separate course/material on “Stored Procedures”. Documentation is available here:

https://dev.mysql.com/doc/connector-net/en/connector-net-tutorials-stored-procedures.html

It is somewhat similar to PL/SQL in Oracle.

Basic Syntax:

```
DELIMITER //

CREATE PROCEDURE `procedure_name` ([argument list])
...
COMMENT 'A procedure'
BEGIN
     What to do – goes here
END//
```

Picking a Delimiter

The delimiter is the character or string of characters that you will use to tell the mySQL client that you've finished typing in an SQL statement. For ages, the delimiter has always been a semicolon. That, however, causes problems, because, in a stored procedure, one can have many statements, and each must end with a semicolon (;). In this sheet we have used “//”.

Once you set a delimiter, you must end every query with the delimiter. For example, you may not need to use a procedure for sometime, rather want to manipulate a table without-procedure syntaxes (i.e. by using standard mysql queries), then still you must end your queries with //.

```
mysql> SELECT * FROM Employee;// [Enter]
```

How to Work with a Stored Procedure

Instruction: Create a database for today's practice.

```
mysql> DROP DATABASE IF EXISTS PROC;

mysql> CREATE DATABASE PROC;

Query OK, 1 row affected (0.00 sec)
```

```
 mysql> USE PROC;
```

```
Database changed
```

Do not work with BANK database. (Anyway, it does not matter; you can drop the full database and create a new one and then again load the bank.sql).

Let see how a procedure looks like.

```
DELIMITER //

CREATE PROCEDURE `p1` ()
LANGUAGE SQL
DETERMINISTIC
SQL SECURITY DEFINER
COMMENT 'A procedure'
BEGIN
     SELECT 'Hello World !';
END//
```

The first part of the statement creates the procedure. The next clauses defines the optional characteristics of the procedure. Then you have the name and finally the body or routine code.

Stored procedure names are case insensitive, and you cannot create procedures with the same name. Inside a procedure body, you can't put database-manipulation statements.

The four characteristics of a procedure are:

- **Language** : For portability purposes; the default value is SQL.
- **Deterministic** : If the procedure always returns the same results, given the same input. This is for replication and logging purposes. The default value is `NOT DETERMINISTIC`.
- **SQL Security** : At call time, check privileges of the user. `INVOKER` is the user who calls the procedure. `DEFINER` is the creator of the procedure. The default value is `DEFINER`.
- **Comment** : For documentation purposes; the default value is `""`.

Calling a procedure:

To call a procedure, you only need to enter the word `CALL`, followed by the name of the procedure, and then the parentheses, including all the parameters between them (variables or values). Parentheses are compulsory.

```
CALL stored_procedure_name (param1, param2, ...)
```

```
CALL procedure1(10 , 'string parameter' , @parameter_var);
```

What is this @ symbol and when to use it?

The @variable syntax in MySQL denotes a user-defined session variable. You can set these user variables outside a stored procedure, but you can also set them inside a stored procedure, and the effect is that the variable retains the value after your procedure call returns.

Example:

```
mysql> CREATE DATABASE PROC;

Query OK, 1 row affected (0.00 sec)

mysql> USE PROC;

Database changed

mysql> CREATE TABLE Employee(ID INT, Name VARCHAR(40));

Query OK, 0 rows affected (0.08 sec)


mysql> DESC Employee;

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

| Field | Type        | Null | Key | Default | Extra |

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

| ID    | int(11)     | YES  |     | NULL    |       |

| Name  | varchar(40) | YES  |     | NULL    |       |

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

2 rows in set (0.00 sec)
```

Try the following:

```
mysql> DELIMITER //

mysql> CREATE PROCEDURE emp_count_2() BEGIN SELECT COUNT(*) INTO
@empCount FROM Employee; END;//

Query OK, 0 rows affected (0.00 sec)

mysql> CALL emp_count_2();

    -> //

Query OK, 1 row affected (0.00 sec)


mysql> SELECT @empcount;//

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

| @empcount |

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

|         0 |

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

1 row in set (0.00 sec)
```

```
mysql> INSERT INTO Employee VALUES((1, 'aaaa'),(2,'bbbb'), (3,
'cccc'), (4, 'dddd'), (5, 'eeee'));
    -> //
ERROR 1241 (21000): Operand should contain 1 column(s)
mysql> INSERT INTO Employee VALUES (1, 'aaaa'),(2,'bbbb'), (3,
'cccc'), (4, 'dddd'), (5, 'eeee');//
Query OK, 5 rows affected (0.03 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Now try:

```
mysql> CALL emp_count_2();
    -> //
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @empcount;//
+-----------+
| @empcount |
+-----------+
|         5 |
+-----------+
1 row in set (0.00 sec)
```

It's okay for multiple sessions to set the user variable in this way concurrently, because user variables are scoped to a single session, and concurrent sessions may have variables of the same name, but with different values.

The variable syntax with no @ prefix is for variables local to the procedure, either procedure parameters, or else local variables declared with DECLARE within the procedure body.

This usage you have, passing a user variable as a parameter and assigning it in the body of the procedure, is useful if you want to call a procedure several times and store the result in separate user variables. Otherwise each call to the procedure would overwrite the previous value in the @empCount user variable for the current session.

Modifying a stored procedure:

MySQL provides an `ALTER PROCEDURE` statement to modify a routine, but only allows for the ability to change certain characteristics. If you need to alter the body or the parameters, you must drop and recreate the procedure.

Drop a stored procedure:

```
mysql> DROP PROCEDURE IF EXISTS emp_count_2;//
Query OK, 0 rows affected (0.00 sec)
```

This is a simple command. The `IF EXISTS` clause prevents an error in case the procedure does not exist.

Parameters:

Let's examine how you can define parameters within a stored procedure.

- `CREATE PROCEDURE proc1 ()` : Parameter list is empty
- `CREATE PROCEDURE proc1 (IN varname DATA-TYPE)` : One input parameter. The word `IN` is optional because parameters are `IN` (input) by default.
- `CREATE PROCEDURE proc1 (OUT varname DATA-TYPE)` : One output parameter. (No examples given here)
- `CREATE PROCEDURE proc1 (INOUT varname DATA-TYPE)` : One parameter which is both input and output. (No examples given here)

Of course, you can define multiple parameters defined with different types.

IN

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE `proc_IN` (IN var1 INT)
    -> BEGIN
    ->     SELECT var1 + 2 AS result;
    -> END//
Query OK, 0 rows affected (0.00 sec)

mysql> call proc_IN(5);
    -> //
+--------+
| result |
+--------+
|      7 |
+--------+
1 row in set (0.00 sec)
```

Variables:

The following step will teach you how to define variables, and store values inside a procedure. You must declare them explicitly at the start of the `BEGIN/END` block, along with their data types. Once you've declared a variable, you can use it anywhere that you could use a session variable, or literal, or column name.

Declare a variable using the following syntax:

```
DECLARE varname DATA-TYPE DEFAULT defaultvalue;
```

IF statement:

```
mysql> CREATE PROCEDURE `proc_IF` (IN param1 INT)
    -> BEGIN
    ->     DECLARE variable1 INT;
    ->     SET variable1 = param1 + 1;
    ->
    ->     IF variable1 = 0 THEN
    ->         SELECT variable1;
    ->     END IF;
    ->
    ->     IF param1 = 0 THEN
    ->         SELECT 'Parameter value = 0';
    ->     ELSE
    ->         SELECT 'Parameter value <> 0';
    ->     END IF;
    -> END //
mysql> call proc_IF(50) ;//
+----------------------+
| Parameter value <> 0 |
+----------------------+
| Parameter value <> 0 |
+----------------------+
1 row in set (0.00 sec)
```

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

mysql> call proc_IF(0) ;//
+---------------------+
| Parameter value = 0 |
+---------------------+
| Parameter value = 0 |
+---------------------+
1 row in set (0.00 sec)

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

CASE statement

The CASE statement is another way to check conditions and take the appropriate path. It's an excellent way to replace multiple IF statements. The statement can be written in two different ways, providing great flexibility to handle multiple conditions.

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE `proc_CASE` (IN param1 INT)
    -> BEGIN
    ->     DECLARE variable1 INT;
    ->     SET variable1 = param1 + 1;
    ->
    ->     CASE variable1
    ->         WHEN 0 THEN
    ->             INSERT INTO table1 VALUES (param1);
    ->         WHEN 1 THEN
    ->             INSERT INTO table1 VALUES (variable1);
    ->         ELSE
    ->             INSERT INTO table1 VALUES (99);
    ->     END CASE;
    ->
    -> END //
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CREATE TABLE table1 ( id INT);
    -> //
Query OK, 0 rows affected (0.09 sec)

mysql> call proc_CASE(0);
    -> //
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM table1;
    -> //
+------+
| id   |
+------+
|    1 |
+------+
1 row in set (0.00 sec)
```

Or,

```
mysql> DROP PROCEDURE IF EXISTS proc_CASE;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE `proc_CASE` (IN param1 INT)
    -> BEGIN
    ->     DECLARE variable1 INT;
    ->     SET variable1 = param1 + 1;
    ->
    ->     CASE
    ->         WHEN variable1 = 0 THEN
    ->             INSERT INTO table1 VALUES (param1);
    ->         WHEN variable1 = 1 THEN
    ->             INSERT INTO table1 VALUES (variable1);
    ->         ELSE
    ->             INSERT INTO table1 VALUES (99);
    ->     END CASE;
    ->
    -> END //
Query OK, 0 rows affected (0.00 sec)

mysql> call proc_CASE(0);
    -> //
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM table1;//
+------+
| id   |
+------+
|    1 |
|    1 |
+------+
2 rows in set (0.00 sec)
```

`WHILE` statement

There are technically three standard loops: `WHILE` loops, `LOOP` loops, and `REPEAT` loops. You also have the option of creating a loop using the “Darth Vader” of programming techniques: the `GOTO` statement.

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE `proc_WHILE` (IN param1 INT)
    -> BEGIN
    ->     DECLARE variable1, variable2 INT;
    ->     SET variable1 = 0;
    ->
    ->     WHILE variable1 < param1 DO
    ->         INSERT INTO table1 VALUES (param1);
    ->         SELECT COUNT(*) INTO variable2 FROM table1;
    ->         SET variable1 = variable1 + 1;
    ->     END WHILE;
    -> END //
Query OK, 0 rows affected (0.02 sec)

mysql> proc_WHILE(5);//
ERROR 1064 (42000): You have an error in your SQL syntax; check
the manual that corresponds to your MySQL server version for the
right syntax to use near 'proc_WHILE(5)' at line 1
mysql> call proc_WHILE(5);//
Query OK, 1 row affected (0.18 sec)

mysql> SELECT * FROM table1;
    -> //
+------+
| id   |
+------+
|    1 |
|    1 |
|    5 |
|    5 |
|    5 |
|    5 |
|    5 |
+------+
7 rows in set (0.00 sec)
```

Task:

(1) Write a procedure to check whether an input argument is odd or even.

(2) Write a mysql procedure to find out the factorial of a number that should be passed with CALL. Use limitations on the value of argument such as argument should be below 20.