Stored Functions and Procedures in MYSQL

A stored routine is either a procedure or a function. Stored routines are created with **CREATE PROCEDURE** and **CREATE FUNCTION** statements. A procedure is invoked using a **CALL** statement, and can only pass back values using output variables. A function can be called from inside a statement just like any other function (that is, by invoking the function's name), and can return a scalar value. Stored routines may call other stored routines.

Procedure parameters can be defined as input-only, output-only, or both. This means that a procedure can pass values back to the caller by using output parameters. These values can be accessed in statements that follow the CALL statement. Functions have only input parameters. As a result, although both procedures and functions can have parameters, procedure parameter declaration differs from that for functions.

STORED FUNCTIONS

MySQL stored functions provide a powerful and flexible way to manipulate and process data.

Setting up a test database

To demonstrate a basic example of stored functions, let's start by creating a database that we can use for testing purposes.

```
CREATE DATABASE test;
USE test;
```

Next, create a table in the **test** database named **products**. To do this, run the following SQL statement:

```
CREATE TABLE products (prod_id INT NOT NULL AUTO_INCREMENT, prod_name
VARCHAR(20) NOT NULL, prod_cost FLOAT NOT NULL DEFAULT 0.0, prod_price
FLOAT NOT NULL DEFAULT 0.0, PRIMARY KEY(prod_id));
```

The following SQL statement adds some sample data to the **products** table:

```
INSERT INTO products (prod_name, prod_cost, prod_price) VALUES ('Basic
Widget',5.95,8.35),('Micro Widget',0.95,1.35),('Mega Widget',99.95,140.00);
```

Creating the stored function

Now that we have a database and a table to work with, we are ready to create a stored function. Let's create a function named **calcProfit**. This function takes two input parameters: the cost and the price of something. It calculates the profit by subtracting the cost from the price, and then returns the value to the calling expression.

To create this stored function, run the following MySQL statements:

```
DELIMITER $$
CREATE FUNCTION calcProfit(cost FLOAT, price FLOAT) RETURNS DECIMAL(9,2)
BEGIN
```

```
  DECLARE profit DECIMAL(9,2);
  SET profit = price-cost;
  RETURN profit;
END$$
DELIMITER ;
```

The **DELIMITER** command at the beginning of these statements prevents MySQL from processing the function definition too soon. The **DELIMITER** command at the end of these statements returns processing to normal.

Using the stored function

You can now use the stored function in a database query. The following SQL statement demonstrates how to do this:

```
SELECT *, calcProfit(prod_cost,prod_price) AS profit FROM products;
```

The **calcProfit** function automatically calculates the profit (price minus the cost) for each product in the table.

STORED PROCEDURE

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

A stored procedure is a method to encapsulate repetitive tasks. They allow for variable declarations, flow control and other useful programming techniques.

Pros

- **Share logic** with other applications. Stored procedures encapsulate functionality; this ensures that data access and manipulation are coherent between different applications.
- **Isolate** users from data tables. This gives you the ability to grant access to the stored procedures that manipulate the data but not directly to the tables.
- Provide a **security** mechanism. Considering the prior item, if you can only access the data using the stored procedures defined, no one else can execute a `DELETE` SQL statement and erase your data.
- To **improve performance** because it reduces network traffic. With a stored procedure, multiple calls can be melded into one.

Cons

- **Increased load** on the database server -- most of the work is done on the server side, and less on the client side.

Picking a Delimiter

The delimiter is the character or string of characters that you'll use to tell the mySQL client that you've finished typing in an SQL statement. 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 we can use "//" as delimiter.

How to Work with a Stored Procedure

Creating a Stored Procedure

```
DELIMITER //
```

```
CREATE PROCEDURE `p2` ()
BEGIN
   SELECT 'Hello World !';
END//
```

Calling a Stored 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);

Modify 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.

Delete a Stored Procedure :

```
DROP PROCEDURE IF EXISTS p2;
```

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.
- `CREATE PROCEDURE proc1 (INOUT varname DATA-TYPE)` : One parameter which is both input and output.

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

IN example :

```
DELIMITER //

CREATE PROCEDURE `proc_IN` (IN var1 INT)
BEGIN
   SELECT var1 + 2 AS result;
END//
```

OUT example :

```
DELIMITER //

CREATE PROCEDURE `proc_OUT` (OUT var1 VARCHAR(100))
BEGIN
   SET var1 = 'This is a test';
END //
```

INOUT example :

```
DELIMITER //

CREATE PROCEDURE `proc_INOUT` (OUT var1 INT)
BEGIN
   SET var1 = var1 * 2;
END //
```

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;
Let's declare a few variables:

```
DECLARE a, b INT DEFAULT 5;

DECLARE str VARCHAR(50);

DECLARE today TIMESTAMP DEFAULT CURRENT_DATE;

DECLARE v1, v2, v3 TINYINT;
```

Working with variables :

Once the variables have been declared, you can assign them values using the `SET` or `SELECT` command:

```
DELIMITER //

CREATE PROCEDURE `var_proc` (IN paramstr VARCHAR(20))
BEGIN
   DECLARE a, b INT DEFAULT 5;
   DECLARE str VARCHAR(50);
   DECLARE today TIMESTAMP DEFAULT CURRENT_DATE;
   DECLARE v1, v2, v3 TINYINT;

   INSERT INTO table1 VALUES (a);
   SET str = 'I am a string';
   SELECT CONCAT(str,paramstr), today FROM table2 WHERE b >=5;
END //
```

EXAMPLE :
STORED PROCEDURES

Stored functions and procedures can enhance database security, improve data integrity, and increase performance. Stored procedures are sometimes confused with stored functions, but they are different in some important ways. Stored procedures, for example, must be invoked with the **CALL** statement, whereas stored functions can be used directly in SQL expressions. The following MySQL statements demonstrate how to create a very basic stored procedure named **procedureTest**. This procedure performs a simple lookup on the **products** table that we used in the stored function example above. Although this procedure does not have much practical use, it demonstrates the correct syntax and structure for declaring a stored procedure:

```
DELIMITER $$
CREATE PROCEDURE procedureTest()
BEGIN
  SELECT prod_name FROM products;
END$$
DELIMITER ;
```

To invoke the stored procedure, use the following MySQL statement:

```
CALL procedureTest()
```