**MySQL Stored Procedures**

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. Stored procedures must be invoked using the CALL statement.

**Why Stored Procedures?**

### **Reduce network traffic**

Stored procedures help reduce the network traffic between applications and MySQL Server. Because instead of sending multiple lengthy SQL statements, applications have to send only the name and parameters of stored procedures.

### **Centralize business logic in the database**

You can use the stored procedures to implement business logic that is reusable by multiple applications. The stored procedures help reduce the efforts of duplicating the same logic in many applications and make your database more consistent.

### **Make database more secure**

The database administrator can grant appropriate privileges to applications that only access specific stored procedures without giving any privileges on the underlying tables.

**Syntax of procedure**

**DELIMITER //**

**CREATE PROCEDURE procedure\_name (**

**[IN | OUT | INOUT] parameter\_name datatype[(length)]**

**)**

**BEGIN**

**declaration\_section**

**executable\_section**

**END;**

**DELIMITER ;**

**MySQL Delimiter**

When you write SQL statements, you use the semicolon (;) to separate two statements

If you use a MySQL client program to define a stored procedure that contains semicolon characters, the MySQL client program will not treat the whole stored procedure as a single statement, but many statements.

Therefore, you must redefine the delimiter temporarily so that you can pass the whole stored procedure to the server as a single statement.

To redefine the default delimiter, you use the **DELIMITER** command:

**DELIMITER delimiter\_character**

The delimiter\_character may consist of a single character or multiple characters e.g., // or $$. However, you should avoid using the backslash (\) because this is the escape character in MySQL.

Once change the delimiter, you can use the new delimiter to end a statement as follows:

**DELIMITER //**

**SELECT \* FROM customers //**

**SELECT \* FROM products //**

**Parameters of procedure**

The most important part is parameters. Parameters are used to pass values to the Procedure. There are 3 different types of parameters, they are as follows:

1. **IN:**  
   This is the Default Parameter for the procedure. It always receives the values from calling program(procedure). The value of the parameter cannot be overwritten by the procedure.
2. **OUT:**  
   This parameter always sends the values to the calling program(procedure), but the value of the parameter can be overwritten by the procedure.
3. **IN OUT:**  
   This parameter performs both the operations. It Receives value from as well as sends the values to the calling program.

**MySQL stored procedure parameter examples**

### **The IN-parameter example**

The following example creates a stored procedure that finds all offices that locate in a country specified by the input parameter countryName:

DELIMITER //

CREATE PROCEDURE GetOfficeByCountry(

IN countryName VARCHAR(255)

)

BEGIN

SELECT \*

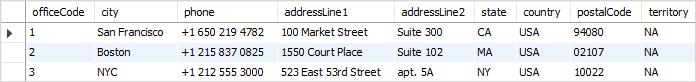
FROM offices

WHERE country = countryName;

END //

DELIMITER ;

CALL GetOfficeByCountry('USA');



### **The OUT parameter example**

The following stored procedure returns the number of orders by order status.

DELIMITER $$

CREATE PROCEDURE GetOrderCountByStatus (

IN orderStatus VARCHAR(25),

OUT total INT

)

BEGIN

SELECT COUNT(orderNumber)

INTO total

FROM orders

WHERE status = orderStatus;

END$$

DELIMITER ;

The stored procedure **GetOrderCountByStatus()** has **two parameters**:

* **orderStatus** : is the **IN** parameter specifies the status of orders to return.
* **total** : is the **OUT** parameter that stores the number of orders in a specific status.

To find the number of orders that **already shipped**, you call **GetOrderCountByStatus**  and pass the order status as of Shipped, and also pass a **session variable ( @total )** to receive the return value.

CALL GetOrderCountByStatus('Shipped',@total);

SELECT @total;



**MySQL DROP PROCEDURE**

The **DROP PROCEDURE** deletes a stored procedure from the database.

The following shows the syntax of the DROP PROCEDURE statement:

**DROP PROCEDURE [IF EXISTS] stored\_procedure\_name;**

**Listing Stored Procedures**

Here is the basic syntax of the **SHOW PROCEDURE STATUS** statement:

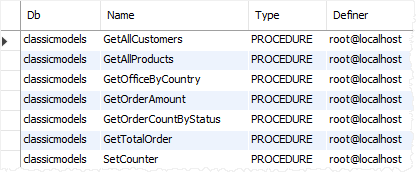
**SHOW PROCEDURE STATUS [LIKE 'pattern' | WHERE search\_condition]**

The SHOW PROCEDURE STATUS statement shows all characteristic of stored procedures including stored procedure names. It returns stored procedures that you have a privilege to access.

The following statement shows all stored procedure in the current MySQL server:

**SHOW PROCEDURE STATUS;**

Here is the partial output:



If you just want to show stored procedures in a particular database, you can use a WHERE clause in the  SHOW PROCEDURE STATUS as shown in the following statement:

**SHOW PROCEDURE STATUS WHERE search\_condition;**

For example, this statement lists all stored procedures in the [sample database](https://www.mysqltutorial.org/mysql-sample-database.aspx) classicmodels:

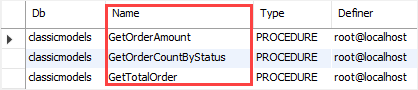
**SHOW PROCEDURE STATUS WHERE db = 'classicmodels';**

In case you want to find stored procedures, whose names contain a specific word, you can use the LIKE clause as follows:

**SHOW PROCEDURE STATUS LIKE '%pattern%'**

The following statement shows all stored procedure whose names contain the wordOrder:

**SHOW PROCEDURE STATUS LIKE '%Order%'**



**MySQL Stored Function**

A stored function is a special kind stored program that returns a single value. Typically, you use stored functions to encapsulate common formulas or business rules that are reusable among SQL statements or stored programs.

Different from a stored procedure, you can use a stored function in SQL statements wherever an expression is used.

## **MySQL CREATE FUNCTION syntax**

DELIMITER $$

CREATE FUNCTION function\_name(

param1,

param2,…

)

RETURNS datatype

[NOT] DETERMINISTIC

BEGIN

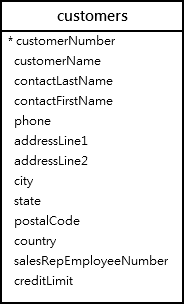
-- statements

END $$

DELIMITER ;

A deterministic function always returns the same result for the same input parameters whereas a non-deterministic function returns different results for the same input parameters.

Let’s take the example of creating a stored function. We will use the **customers** table in the sample database for the demonstration.



**The following CREATE FUNCTION statement creates a function that returns the customer level based on credit:**

DELIMITER $$

CREATE FUNCTION CustomerLevel(

credit DECIMAL(10,2)

)

RETURNS VARCHAR(20)

DETERMINISTIC

BEGIN

DECLARE customerLevel VARCHAR(20);

IF credit > 50000 THEN

SET customerLevel = 'PLATINUM';

ELSEIF (credit <= 50000 AND

credit >= 10000) THEN

SET customerLevel = 'GOLD';

ELSEIF credit < 10000 THEN

SET customerLevel = 'SILVER';

END IF;

-- return the customer level

RETURN (customerLevel);

END$$

DELIMITER ;

### **Calling a stored function in an SQL statement**

The following statement uses the **CustomerLevel** stored function:

SELECT

customerName,

CustomerLevel(creditLimit)

FROM

customers

ORDER BY

customerName;

