**Lab 5: Stored Procedures**

## MySQL Stored Procedure

## Getting started with stored procedures

The following [SELECT](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statement returns all rows in the table customers from the [classicmodels database](http://www.mysqltutorial.org/mysql-sample-database.aspx) that we have created during Lab-2:

SELECT

    customerName,

    city,

    state,

    postalCode,

    country

FROM

    customers

ORDER BY customerName;

If you want to save this query on the database server for execution later, one way to do it is to use a stored procedure as we studied in the class.

The following [CREATE PROCEDURE](http://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx) statement creates a new stored procedure that wraps the query above:

DELIMITER $$

CREATE PROCEDURE GetCustomers()

BEGIN

    SELECT

        customerName,

        city,

        state,

        postalCode,

        country

    FROM

        customers

    ORDER BY customerName;

END$$

DELIMITER ;

By definition, a stored procedure is a segment of declarative SQL statements stored inside the MySQL Server. In this example, we have just created a stored procedure with the name GetCustomers().

Once you save the stored procedure, you can invoke it by using the CALL statement:

CALL GetCustomers();

And the statement returns the same result as the query that we have written above.

To view the procedures, you can write following query;

Show procedure status;

This will list all of the stored procedures in the database.

In XAMPP, the procedures are found under the Routines Section.

## Using MySQL DELIMITER for stored procedures

A stored procedure typically contains multiple statements separated by semicolon (;).  To use compile the whole stored procedure as a single compound statement, you need to temporarily change the delimiter from the semicolon (;) to anther delimiters such as $$ or //:

DELIMITER $$

CREATE PROCEDURE sp\_name()

BEGIN

  -- statements

END $$

DELIMITER ;

In this code:

* First, change the default delimiter to $$
* Second, use (;) in the body of the stored procedure and $$ after the END keyword to end the stored procedure.
* Third, change the default delimiter back to a semicolon (;)

The first time you invoke a stored procedure, MySQL looks up for the name in the database catalog, compiles the stored procedure’s code, place it in a memory area known as a cache, and execute the stored procedure.

If you invoke the same stored procedure in the same session again, MySQL just executes the stored procedure from the cache without having to recompile it.

Procedure for getting all products, following procedure will result all the products from products table;

DELIMITER $$

CREATE PROCEDURE Getallproducts()

BEGIN

SELECT \*

FROM products;

END$$

DELIMITER ;

CALL Getallproducts();

Following statement creates a new stored procedure named GetPayments() and results payments of customers:

DELIMITER $$

CREATE PROCEDURE GetPayments()

BEGIN

    SELECT

        customerName,

        checkNumber,

        paymentDate,

        amount

    FROM payments

    INNER JOIN customers

        using (customerNumber);

END$$

DELIMITER ;

CALL GetPayments();

## MySQL Drop Procedure Statement

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;

First, create a new stored procedure that returns employee and office information:

DELIMITER $$

CREATE PROCEDURE GetEmployees()

BEGIN

    SELECT

        firstName,

        lastName,

        city,

        state,

        country

    FROM employees

    INNER JOIN offices using (officeCode);

END$$

DELIMITER ;

Call the stored procedure as;

CALL GetEmployees();

You can delete the so created stored procedure as below;

DROP PROCEDURE GetEmployees;

**Note: In phpMyAdmin you can see all of the created stored procedures are under the routines tab. You can easily edit, execute and drop the procedure.**

## MySQL Stored Procedure Parameters

Almost stored procedures that you develop require parameters. The parameters make the stored procedure more flexible and useful.

In MySQL, a parameter has one of three modes: IN, OUT, or INOUT.

### IN parameters

IN is the default mode. When you define an IN parameter in a stored procedure, the calling program has to pass an argument to the stored procedure. In addition, the value of an IN parameter is protected. It means that even the value of the IN parameter is changed inside the stored procedure, its original value is retained after the stored procedure ends. In other words, the stored procedure only works on the copy of the IN parameter.

### OUT parameters

The value of an OUT parameter can be changed inside the stored procedure and its new value is passed back to the calling program. Notice that the stored procedure cannot access the initial value of the OUT parameter when it starts.

### INOUT parameters

An INOUT  parameter is a combination of IN  and OUT  parameters. It means that the calling program may pass the argument, and the stored procedure can modify the INOUT parameter, and pass the new value back to the calling program.

### Defining a parameter

Here is the basic syntax of defining a parameter in stored procedures:

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

In this syntax,

* First, specify the parameter mode, which can be IN , OUTor INOUT , depending on the purpose of the parameter in the stored procedure.
* Second, specify the name of the parameter. The parameter name must follow the naming rules of the column name in MySQL.
* Third, specify the data type and maximum length of the parameter.

**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 ;

In this example, the countryName is the IN parameter of the stored procedure. Suppose that you want to find offices locating in the USA, you need to pass an argument (USA) to the stored procedure as shown in the following query:

CALL GetOfficeByCountry('USA');

To find offices in France, you pass the literal string France to the GetOfficeByCountry stored procedure as follows:

CALL GetOfficeByCountry('France')

Because the countryName is the IN parameter, you must pass an argument. Fail to do so will result in an error:

CALL GetOfficeByCountry();

#1318 - Incorrect number of arguments for PROCEDURE classicmodels.GetOfficeByCountry; expected 1, got 0

**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;

To get the number of orders that are in-process, you call the stored procedure GetOrderCountByStatus as follows:

CALL GetOrderCountByStatus('in process',@total);

SELECT @total AS  total\_in\_process;

**The INOUT parameter example**

The following example demonstrates how to use an INOUT parameter in the stored procedure.

DELIMITER $$

CREATE PROCEDURE SetCounter(

    INOUT counter INT,

    IN inc INT

)

BEGIN

    SET counter = counter + inc;

END$$

DELIMITER ;

In this example, the stored procedure SetCounter()  accepts one INOUT  parameter ( counter ) and one IN parameter ( inc ). It increases the counter ( counter ) by the value of specified by the inc parameter.

These statements illustrate how to call the SetSounter  stored procedure:

SET @counter = 1;

CALL SetCounter(@counter,1);

CALL SetCounter(@counter,1);

CALL SetCounter(@counter,5);

SELECT @counter;

## Stored Procedures that Return Multiple Values

Let’s take a look at the orders table in the classicmodels database that we have been using since Lab-2;

The following stored procedure accepts customer number and returns the total number of orders that were shipped, canceled, resolved, and disputed.

DELIMITER $$

CREATE PROCEDURE get\_order\_by\_cust(

    IN cust\_no INT,

    OUT shipped INT,

    OUT canceled INT,

    OUT resolved INT,

    OUT disputed INT)

BEGIN

        -- shipped

        SELECT

            count(\*) INTO shipped

        FROM

            orders

        WHERE

            customerNumber = cust\_no

                AND status = 'Shipped';

        -- canceled

        SELECT

            count(\*) INTO canceled

        FROM

            orders

        WHERE

            customerNumber = cust\_no

                AND status = 'Canceled';

        -- resolved

        SELECT

            count(\*) INTO resolved

        FROM

            orders

        WHERE

            customerNumber = cust\_no

                AND status = 'Resolved';

        -- disputed

        SELECT

            count(\*) INTO disputed

        FROM

            orders

        WHERE

            customerNumber = cust\_no

                AND status = 'Disputed';

END

In addition to the IN parameter, the stored procedure takes four additional OUT parameters: shipped, canceled, resolved, and disputed. Inside the stored procedure, you use a [SELECT](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statement with the [COUNT](http://www.mysqltutorial.org/mysql-count/) function to get the corresponding total of orders based on the order’s status and assign it to the respective parameter.

To use the get\_order\_by\_cust stored procedure, you pass customer number and four user-defined variables to get the out values.

After executing the stored procedure, you use the SELECT statement to output the variable values.

CALL get\_order\_by\_cust(141,@shipped,@canceled,@resolved,@disputed);

SELECT @shipped,@canceled,@resolved,@disputed;

You will see following result;

|  |  |  |  |
| --- | --- | --- | --- |
| **Shipped** | **Canceled** | **resolved** | **disputed** |
| 22 | 0 | 1 | 1 |

**Using Variables in Stored Procedure**

A variable is a named data object whose value can change during the [stored procedure](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx) execution. You typically use variables in stored procedures to hold immediate results. These variables are local to the stored procedure. We can define variable using DECLARE as;

DECLARE variable\_name datatype(size) [DEFAULT default\_value];

The following example illustrates how to declare and use a variable in a stored procedure:

DELIMITER $$

CREATE PROCEDURE GetTotalOrder()

BEGIN

DECLARE totalOrder INT DEFAULT 0;

    SELECT COUNT(\*)

    INTO totalOrder

    FROM orders;

    SELECT totalOrder;

END$$

DELIMITER ;

This statement calls the stored procedure GetTotalOrder():

CALL GetTotalOrder();