Stored Procedures and views

MySQL Sample Database Schema

The MySQL sample database schema consists of the following tables:

* **Customers**: stores customer’s data.
* **Products**: stores a list of different model cars.
* **ProductLines**: stores a list of product line categories.
* **Orders**: stores sales orders placed by customers.
* **OrderDetails**: stores sales order line items for each sales order.
* **Payments**: stores payments made by customers based on their accounts.
* **Employees**: stores all employee information as well as the organization structure such as who reports to whom.
* **Offices**: stores sales office data.



**STORED PROCEDURES**

* When you use MySQL Workbench or mysql shell to issue the query to MySQL Server, MySQL processes the query and returns the result set.
* 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.
* **A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.**
* So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

Example: The following [SELECT](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statement returns all rows in the table customers

SELECT

customerName,

city,

state,

postalCode,

country

FROM

customers

ORDER BY customerName;

If there is a need of running the queries on a frequent basis, create it as a procedure.

The following [CREATE PROCEDURE](https://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 ;

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

**call getcustomers();**

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.

A stored procedure can have [parameters](https://www.mysqltutorial.org/stored-procedures-parameters.aspx) so you can pass values to it and get the result back. For example, you can have a stored procedure that returns customers by country and city. In this case, the country and city are parameters of the stored procedure.

# **MySQL Delimiter**

When writing SQL statements, you use the semicolon (;) to separate two statements like the following example:

**select \* from customers;**

**select \* from products;**

A MySQL client program such as MySQL Workbench or mysql program uses the **delimiter (;) to separate statements and executes each statement separately.**

However, a stored procedure consists of multiple statements separated by a semicolon (;).

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.

# **MySQL DROP PROCEDURE**

The DROP PROCEDURE statement deletes a stored procedure created by the [CREATE PROCEDURE](https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx) statement.

The following shows the syntax of the DROP PROCEDURE statement:

DROP PROCEDURE IF EXISTS stored\_procedure\_name;

Example:

DELIMITER $$

CREATE PROCEDURE GetEmployees()

BEGIN

SELECT

firstName,

lastName,

city,

state,

country

FROM employees

INNER JOIN offices using (officeCode);

END$$

DELIMITER ;

call GetEmployees();

DROP PROCEDURE GetEmployees;

# **MySQL Stored Procedure Variables**

A variable is a named data object whose value can change during the [stored procedure](https://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.

Before using a variable, you must declare it.

## Declaring variables

To declare a variable inside a stored procedure, you use the DECLARE  statement as follows:

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

In this syntax:

* First, specify the name of the variable after the DECLARE keyword. The variable name must follow the naming rules of MySQL table column names.
* Second, specify the data type and length of the variable. A variable can have any [MySQL data types](https://www.mysqltutorial.org/mysql-data-types.aspx) such as INT, VARCHAR , and DATETIME.
* Third, assign a variable a default value using the DEFAULT option.  If you declare a variable without specifying a default value, its value is NULL.

## Assigning variables

Once a variable is declared, it is ready to use. To assign a variable a value, you use the SET statement:

SET variable\_name = value;

For example:

DECLARE total INT DEFAULT 0;

SET total = 10;

The value of the total variable is 10  after the assignment.

In addition to the SET statement, you can use the SELECT INTO statement to assign the result of a query to a variable as shown in the following example:

DECLARE productCount INT DEFAULT 0;

SELECT COUNT(\*)

INTO productCount

FROM products;

In this example:

* First, declare a variable named productCount  and initialize its value to 0.
* Then, use the SELECT INTO  statement to assign the productCount  variable the number of products selected from the products  table.

## Variable scopes

A variable has its own scope that defines its lifetime. If you declare a variable inside a stored procedure, it will be out of scope when the END statement of stored procedure reaches.

When you declare a variable inside the block BEGIN END, it will be out of scope if the END is reached.

A variable whose name begins with the @ sign is a [session variable](https://www.mysqltutorial.org/mysql-variables/). It is available and accessible until the session ends.

## Putting it all together

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 ;

call GetTotalOrder();

**How it works.**

First, declare a variable totalOrder with a default value of zero. This variable will hold the number of orders from the orders table.

DECLARE totalOrder INT DEFAULT 0;

Second, use the SELECT INTO  statement to assign the variable totalOrder the number of orders selected from the orders table:

SELECT COUNT(\*)

INTO totalOrder

FROM orders;

Third, select the value of the variable totalOrder.

SELECT totalOrder;

## MySQL stored procedure parameters

[Stored procedures](https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx) have parameters. The parameters make the stored procedure more useful and reusable. A parameter in a stored procedure 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.

### **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.

### **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 , OUT or 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.

## MySQL stored procedure parameter examples

Let’s take some examples of using stored procedure parameters.

### **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');**

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:

Because the countryName is the IN parameter, you must pass an argument. If you don’t do so, you’ll get an error:

**call GetOfficeByCountry();**

Here’s the error:

Error Code: 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:

* The orderStatus: is the IN parameter specifies the status of orders to return.
* The 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 a 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); -- 2

CALL SetCounter(@counter,1); -- 3

CALL SetCounter(@counter,5); -- 8

SELECT @counter; -- 8