# MySQL Stored Procedures

In this section, you will learn how to develop stored programs including stored procedures and stored functions in MySQL.

### Section 1. Stored procedure basics

* [Introduction to Stored Procedures in MySQL](https://www.mysqltutorial.org/introduction-to-sql-stored-procedures.aspx)  – introduce you to stored procedures, their advantages, and disadvantages.
* [Changing the default delimiter](https://www.mysqltutorial.org/mysql-stored-procedure/mysql-delimiter/) – learn how to change the default delimiter in MySQL.
* [Creating new stored procedures](https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx) – show you how to create use the CREATE PROCEDURE statement to create a new stored procedure in the database.
* [Removing stored procedures](https://www.mysqltutorial.org/mysql-stored-procedure/mysql-drop-procedure/) – show you how to use the DROP PROCEDURE statement to drop an existing stored procedure.
* [Variables](https://www.mysqltutorial.org/variables-in-stored-procedures.aspx) – guide on you how to use variables to hold immediate results inside stored procedures.
* [Parameters](https://www.mysqltutorial.org/stored-procedures-parameters.aspx) – introduce you to various types of parameters used in stored procedures including IN, OUT, and INOUT parameter.
* [Altering stored procedure](https://www.mysqltutorial.org/mysql-stored-procedure/alter-stored-procedure/) – show you step by step how to alter a stored procedure using a sequence of DROP PROCEDURE and CREATE PROCEDURE statements in MySQL Workbench.
* [Listing stored procedures](https://www.mysqltutorial.org/listing-stored-procedures-in-mysql-database.aspx) – provide you with some useful commands to list stored procedures from databases.

### Section 2. Conditional Statements

* [IF statement](https://www.mysqltutorial.org/mysql-if-statement/) – show you how to use the IF THEN statement in stored procedures.
* [CASE statement](https://www.mysqltutorial.org/mysql-case-statement/) – introduce you to the CASE statements including simple CASE and searched CASE statements.

### Section 3. Loops

* [LOOP](https://www.mysqltutorial.org/stored-procedures-loop.aspx) – learn how to execute a list of statements repeatedly based on a condition.
* [WHILE Loop](https://www.mysqltutorial.org/mysql-stored-procedure/mysql-while-loop/) – show you how to execute a loop as long as a condition is true.
* [REPEAT Loop](https://www.mysqltutorial.org/mysql-stored-procedure/mysql-repeat-loop/) – show you how to execute a loop until a search condition is true.
* [LEAVE statement](https://www.mysqltutorial.org/mysql-stored-procedure/mysql-leave/) – guide you on how to exit a loop immediately.

### Section 4. Error Handling

* [Handing exceptions](https://www.mysqltutorial.org/mysql-error-handling-in-stored-procedures/) – show you how to handle exception and errors in stored procedures.
* [Raising errors](https://www.mysqltutorial.org/mysql-signal-resignal/)  – learn how to use SIGNAL and RESIGNAL to raise errors in stored procedures.

### Section 5. Cursors

* [Cursors](https://www.mysqltutorial.org/mysql-cursor/) – learn how to use cursors to process row by row in a result set.

### Section 6. Stored Functions

* [Creating a stored function](https://www.mysqltutorial.org/mysql-stored-function/) – show you how to use the create stored functions in the database.
* [Removing a stored function](https://www.mysqltutorial.org/mysql-stored-procedure/mysql-drop-function/) – use the DROP FUNCTION statement to remove a stored function.
* [Listing stored functions](https://www.mysqltutorial.org/mysql-stored-procedure/mysql-show-function/) – learn how to list all stored functions in the database by using the SHOW FUNCTION STATUS or querying from the data dictionary.

### Section 7. Stored Program Security

* [Stored object access control](https://www.mysqltutorial.org/mysql-stored-procedure/mysql-stored-object-access-control/) – learn how to control the security of the stored objects.

# Introduction to MySQL Stored Procedures

Summary: in this tutorial, you will learn about MySQL stored procedures, their advantages, and disadvantages.

## Getting started with stored procedures

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

SELECT

customerName,

city,

state,

postalCode,

country

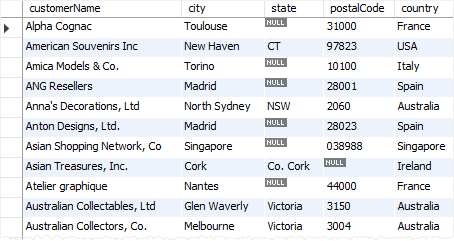
FROM

customers

ORDER BY customerName;

Code language: SQL (Structured Query Language) (sql)

This picture shows the partial output of the query:



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.

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 ;

Code language: SQL (Structured Query Language) (sql)

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();

Code language: SQL (Structured Query Language) (sql)

And the statement returns the same result as the query.

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.

A stored procedure may contain control flow statements such as [IF](https://www.mysqltutorial.org/mysql-if-statement/), [CASE](https://www.mysqltutorial.org/mysql-case-statement/), and LOOP that allow you to implement the code in the procedural way.

A stored procedure can call other stored procedures or [stored functions](https://www.mysqltutorial.org/mysql-stored-function/), which allows you to modulize your code.

Note that you will learn step by step how to [create a new stored procedure](https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx) in the next tutorial.

## MySQL stored procedures advantages

The following are the advantages of 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.

## MySQL stored procedures disadvantages

Besides those advantages, stored procedures also have disadvantages:

### Resource usages

If you use many stored procedures, the memory usage of every connection will increase substantially.

Besides, overusing a large number of logical operations in the stored procedures will increase the CPU usage because the MySQL is not well-designed for logical operations.

### Troubleshooting

It’s difficult to debug stored procedures. Unfortunately, MySQL does not provide any facilities to debug stored procedures like other enterprise database products such as Oracle and SQL Server.

### Maintenances

Developing and maintaining stored procedures often requires a specialized skill set that not all application developers possess. This may lead to problems in both application development and maintenance.

In this tutorial, you have learned about the MySQL stored procedures, their advantages, and disadvantages. Let’s move on to learn how to create a new stored procedure in MySQL.

# MySQL Delimiter

Summary: in this tutorial, you will learn how to change the default MySQL delimiter by using the DELIMITER command.

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

SELECT \* FROM products;

SELECT \* FROM customers;

Code language: SQL (Structured Query Language) (sql)

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.

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

DELIMITER delimiter\_character

Code language: SQL (Structured Query Language) (sql)

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

For example, the following statement changes the current delimiter to //:

DELIMITER //

Code language: SQL (Structured Query Language) (sql)

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

DELIMITER //

SELECT \* FROM customers //

SELECT \* FROM products //

Code language: SQL (Structured Query Language) (sql)

To change the delimiter to the default one, which is a semicolon (;), you use the following statement:

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

## Using MySQL DELIMITER for stored procedures

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

DELIMITER $$

CREATE PROCEDURE sp\_name()

BEGIN

-- statements

END $$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

Note that you will learn the syntax of [creating a stored procedure](https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx) in the next tutorial.

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 (;)

In this tutorial, you have learned how to use the MySQL DELIMITER command to change the default delimiter (;) to another.

# MySQL CREATE PROCEDURE

Summary: in this tutorial, you will learn step by step how to the MySQL CREATE PROCEDURE statement to create new stored procedures.

## MySQL CREATE PROCEDURE statement

This query returns all products in the products table from the [sample database](https://www.mysqltutorial.org/mysql-sample-database.aspx).

SELECT \* FROM products;

Code language: SQL (Structured Query Language) (sql)

The following statement creates a new stored procedure that wraps the query:

DELIMITER //

CREATE PROCEDURE GetAllProducts()

BEGIN

SELECT \* FROM products;

END //

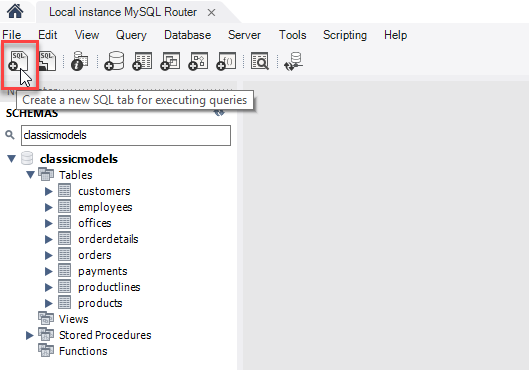
DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

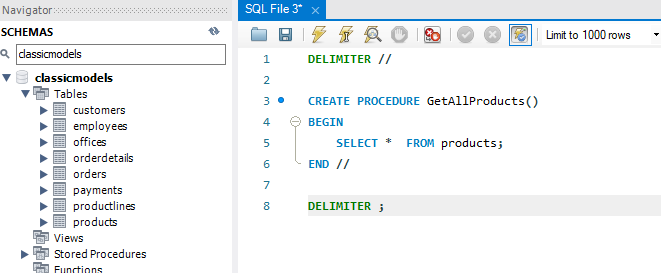
To execute these statements:

First, launch MySQL Workbench.

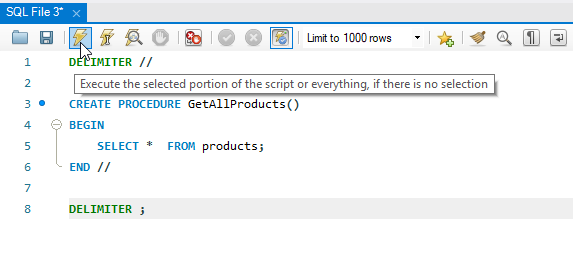
Second, create a new SQL tab for executing queries:



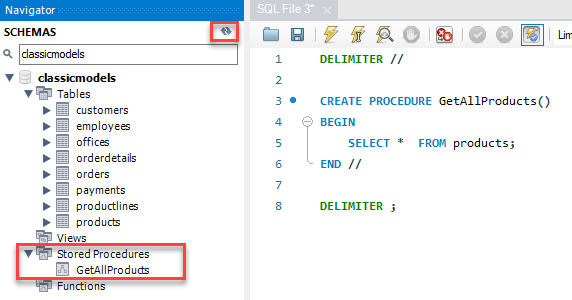
Third, enter the statements in the SQL tab:



Fouth, execute the statements. Note that you can select all statements in the SQL tab (or nothing) and click the Execute button. If everything is fine, MySQL will create the stored procedure and save it in the server.



Fifth, check the stored procedure by opening the Stored Procedures node. If you don’t see the stored procedure, you can click the Refresh button next to the SCHEMAS title:



Congratulation! you have successfully created the first stored procedure in MySQL.

Let’s examine the syntax of the stored procedure.

The first and last DELIMITER commands are not a part of the stored procedure. The first DELIMITER command changes the default delimiter to // and the last DELIMITER command changes the delimiter back to the default one which is semicolon (;).

To create a new stored procedure, you use the CREATE PROCEDURE statement.

Here is the basic syntax of the CREATE PROCEDURE statement:

CREATE PROCEDURE procedure\_name(parameter\_list)

BEGIN

statements;

END //

Code language: SQL (Structured Query Language) (sql)

In this syntax

* First, specify the name of the stored procedure that you want to create after the CREATE PROCEDURE keywords.
* Second, specify a list of comma-separated parameters for the stored procedure in parentheses after the procedure name. Note that you’ll learn how to create [stored procedures with parameters in the upcoming tutorials](https://www.mysqltutorial.org/stored-procedures-parameters.aspx).
* Third, write the code between the BEGIN END block. The above example just has a simple [SELECT](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statement. After the END keyword, you place the delimiter character to end the procedure statement.

## Executing a stored procedure

To execute a stored procedure, you use the CALL statement:

CALL stored\_procedure\_name(argument\_list);

Code language: SQL (Structured Query Language) (sql)

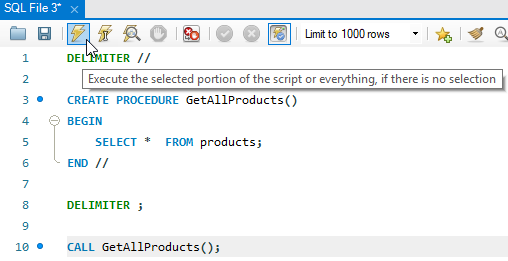
In this syntax, you specify the name of the stored procedure after the CALL keyword. If the stored procedure has parameters, you need to pass arguments inside parentheses following the stored procedure name.

This example illustrates how to call the GetAllProducts() stored procedure:

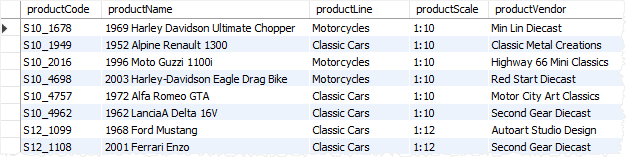
CALL GetAllProducts();

Code language: SQL (Structured Query Language) (sql)

Executing this statement is the same as executing an SQL statement:



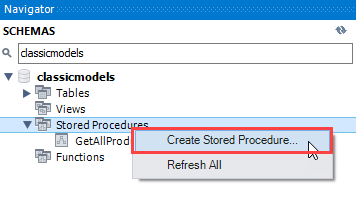
Here’s the partial output:



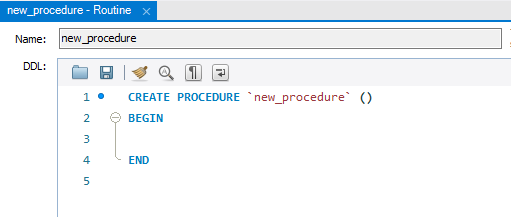
## Creating a stored procedure using the MySQL Workbench wizard

By using the MySQL Workbench wizard, you don’t have to take care of many things like delimiters or executing the command to create stored procedures.

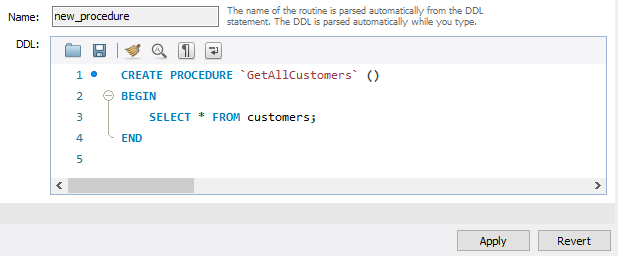
First, right-click on the Stored Procedures from the Navigator and select the Create Stored Procedure… menu item.



The following tab will open:

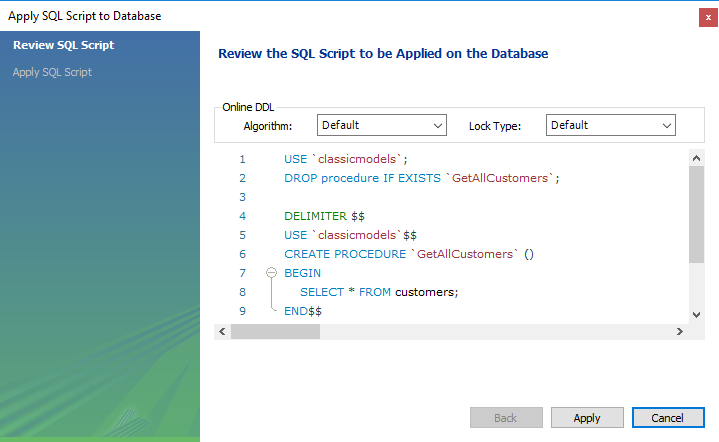


Second, change the stored procedure’s name and add the code between the BEGIN END block:

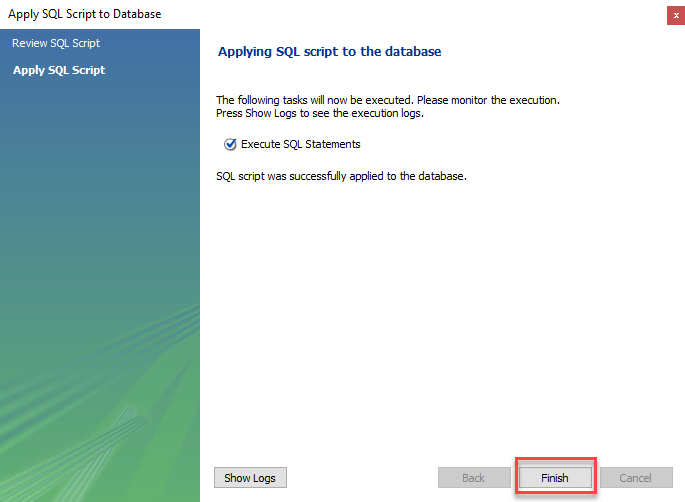


The stored procedure name is GetAllCustomers() which returns all rows in the customers table from the [sample database](https://www.mysqltutorial.org/mysql-sample-database.aspx).

Third, Click the Apply button, MySQL Workbench will open a new window for reviewing SQL script before applying it on the database:

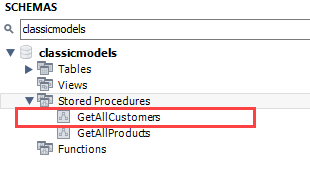


Fourth, Click the Apply button to confirm. MySQL Workbench will create the stored procedure:



Fifth, click the Finish button to close the window.

Finally, view the stored procedure in the Stored Procedures list:



## Summary

* Use the CREATE PROCEDURE statement to create a new stored procedure.
* Use the CALL statement to execute a stored procedure.
* MySQL stores the stored procedures in the server.

# MySQL DROP PROCEDURE

Summary: in this tutorial, you will learn how to use the MySQL DROP PROCEDURE statement to drop a stored procedure.

## Introduction to the MySQL DROP PROCEDURE statement

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;

Code language: SQL (Structured Query Language) (sql)

In this syntax:

* First, specify the name of the stored procedure that you want to remove after the DROP PROCEDURE keywords.
* Second, use IF EXISTS option to conditionally drop the stored procedure if it exists.

When you drop a procedure that does not exist without using the IF EXISTS option, MySQL issues an error. In this case, if you use the IF EXISTS option, MySQL issues a warning instead.

Note that you need to have the ALTER ROUTINE privilege of the stored procedure in order to remove it.

## MySQL DROP PROCEDURE examples

Let’s take some examples of using the DROP PROCEDURE statement.

### 1) Using MySQL DROP PROCEDURE example

First, [create a new stored procedure](https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx) 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 ;

Code language: SQL (Structured Query Language) (sql)

Second, use the DROP PROCEDURE to delete the GetEmployees() stored procedure:

DROP PROCEDURE GetEmployees;

Code language: SQL (Structured Query Language) (sql)

### 2) Using MySQL DROP PROCEDURE with the IF EXISTS option example

The following example drops a stored procedure that does not exist:

DROP PROCEDURE abc;

Code language: SQL (Structured Query Language) (sql)

MySQL issued the following error:

Error Code: 1305. PROCEDURE classicmodels.abc does not exist

Code language: SQL (Structured Query Language) (sql)

This statement drops the same non-existing stored procedure, but with IF EXISTS option:

DROP PROCEDURE IF EXISTS abc;

Code language: SQL (Structured Query Language) (sql)

This time MySQL issued a warning.

0 row(s) affected, 1 warning(s): 1305 PROCEDURE classicmodels.abc does not exist

Code language: SQL (Structured Query Language) (sql)

The statement SHOW WARNINGS shows the warning:

SHOW WARNINGS;

Code language: SQL (Structured Query Language) (sql)

Here is the output:



## Dropping a stored procedure using MySQL Workbench

This statement creates a new stored procedure named GetPayments() that returns the customer and payment information:

DELIMITER $$

CREATE PROCEDURE GetPayments()

BEGIN

SELECT

customerName,

checkNumber,

paymentDate,

amount

FROM payments

INNER JOIN customers

using (customerNumber);

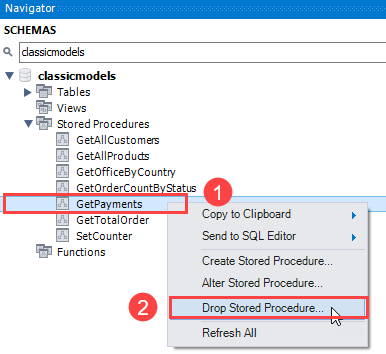
END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

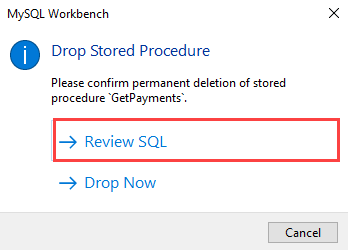
To drop the stored procedure using MySQL Workbench, you follow these steps:

First, right-click the name of the stored procedure that you want to remove and choose Drop Stored Procedure… option.

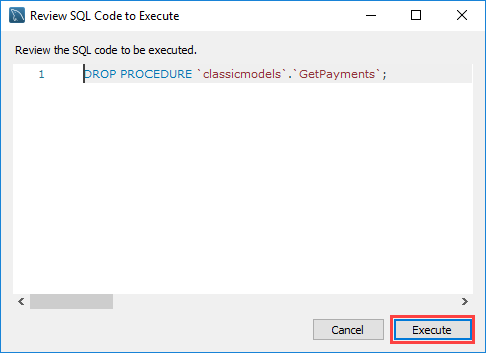


MySQL Workbench will display a confirmation window.

Second, click Review SQL to review the SQL statement that MySQL Workbench will apply to the database, or click the Drop Now if you want to immediately remove the stored procedure.



Third, review the SQL code and click the Execute button to drop the stored procedure.



## Summary

* Use DROP PROCEDURE statement to remove a stored procedure.
* Use IF EXISTS option to drop the stored procedure if it exists.

# MySQL Stored Procedure Variables

Summary: in this tutorial, you will learn about variables in the stored procedure, how to declare, and use variables. In addition, you will learn about the scopes of 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];

Code language: SQL (Structured Query Language) (sql)

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.

The following example declares a variable named totalSale with the data type DEC(10,2) and default value 0.0  as follows:

DECLARE totalSale DEC(10,2) DEFAULT 0.0;

Code language: SQL (Structured Query Language) (sql)

MySQL allows you to declare two or more variables that share the same data type using a single DECLARE statement. The following example declares two integer variables  x and  y, and set their default values to zero.

DECLARE x, y INT DEFAULT 0;

Code language: SQL (Structured Query Language) (sql)

## 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;

Code language: SQL (Structured Query Language) (sql)

For example:

DECLARE total INT DEFAULT 0;

SET total = 10;

Code language: SQL (Structured Query Language) (sql)

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;

Code language: SQL (Structured Query Language) (sql)

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.

MySQL allows you to declare two or more variables that share the same name in different scopes. Because a variable is only effective in its scope. However, declaring variables with the same name in different scopes is not good programming practice.

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 ;

Code language: SQL (Structured Query Language) (sql)

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;

Code language: SQL (Structured Query Language) (sql)

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;

Code language: SQL (Structured Query Language) (sql)

Third, select the value of the variable totalOrder.

SELECT totalOrder;

Code language: SQL (Structured Query Language) (sql)

Note that you will learn how to use variables practically in the subsequent tutorials. The example in this tutorial is just an illustration so that you understand the concept.

This statement calls the stored procedure GetTotalOrder():

CALL GetTotalOrder();

Code language: SQL (Structured Query Language) (sql)

Here is the output:



In this tutorial, you have learned how you how to declare and use variables inside stored procedures.

# MySQL Stored Procedure Parameters

Summary: in this tutorial, you will learn how to create stored procedures with parameters, including IN, OUT, and INTOUT parameters.

## Introduction to MySQL stored procedure parameters

Often, [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.

In addition, the value of an IN parameter is protected. It means that even you change the value of the IN parameter inside the stored procedure, its original value is unchanged 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)]

Code language: SQL (Structured Query Language) (sql)

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 ;

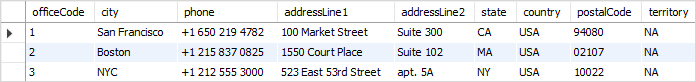
Code language: SQL (Structured Query Language) (sql)

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');

Code language: SQL (Structured Query Language) (sql)



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

CALL GetOfficeByCountry('France')

Code language: SQL (Structured Query Language) (sql)



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();

Code language: SQL (Structured Query Language) (sql)

Here’s the error:

Error Code: 1318. Incorrect number of arguments for PROCEDURE classicmodels.GetOfficeByCountry; expected 1, got 0

Code language: JavaScript (javascript)

### 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 ;

Code language: SQL (Structured Query Language) (sql)

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;

Code language: SQL (Structured Query Language) (sql)



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;

Code language: SQL (Structured Query Language) (sql)



### 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 ;

Code language: SQL (Structured Query Language) (sql)

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

Code language: SQL (Structured Query Language) (sql)

Here is the output:



In this tutorial, you have learned how create stored procedures with parameters including IN, OUT, and INOUT parameters.

# Listing Stored Procedures

Summary: in this tutorial, you will learn how to list stored procedures from databases in a MySQL Server.

## Listing stored procedures using SHOW PROCEDURE STATUS statement

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

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

Code language: SQL (Structured Query Language) (sql)

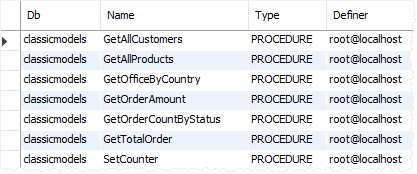
The SHOW PROCEDURE STATUS statement shows all characteristic of stored procedures including stored procedure names. It returns stored procedures that you have a privilege to access.

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

SHOW PROCEDURE STATUS;

Code language: SQL (Structured Query Language) (sql)

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;

Code language: SQL (Structured Query Language) (sql)

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';

Code language: SQL (Structured Query Language) (sql)

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

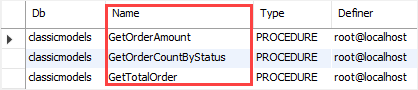
SHOW PROCEDURE STATUS LIKE '%pattern%'

Code language: SQL (Structured Query Language) (sql)

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

SHOW PROCEDURE STATUS LIKE '%Order%'

Code language: SQL (Structured Query Language) (sql)



## Listing stored procedures using the data dictionary

The routines table in the information\_schema database contains all information on the stored procedures and stored functions of all databases in the current MySQL server.

To show all stored procedures of a particular database, you use the following query:

SELECT

routine\_name

FROM

information\_schema.routines

WHERE

routine\_type = 'PROCEDURE'

AND routine\_schema = '<database\_name>';

Code language: SQL (Structured Query Language) (sql)

For example, this statement lists all stored procedures in the classicmodels database:

SELECT

routine\_name

FROM

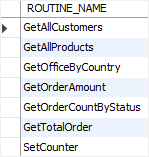
information\_schema.routines

WHERE

routine\_type = 'PROCEDURE'

AND routine\_schema = 'classicmodels';

Code language: SQL (Structured Query Language) (sql)

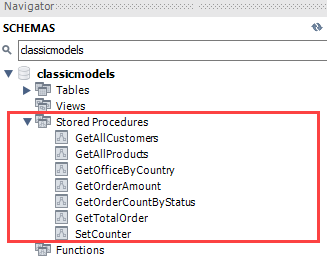


## Showing stored procedures using MySQL Workbench

In MySQL Workbench, you can view all stored procedures from a database.

Step 1. Access the database that you want to view the stored procedures.

Step 2. Open the Stored Procedures menu. You will see a list of stored procedures that belong to the current database.



In this tutorial, you have learned how to list the stored procedures in a database by querying them from the data dictionary.

# MySQL IF Statement

Summary: in this tutorial, you will learn how to use MySQL IF statement to execute a block of SQL code based on a specified condition.

Note that MySQL has an [IF()](https://www.mysqltutorial.org/mysql-if-function.aspx) function that is different from the IF statement described in this tutorial.

The IF statement has three forms: simple IF-THEN statement, IF-THEN-ELSE statement, and IF-THEN-ELSEIF- ELSE statement.

## MySQL simple IF-THEN statement

The IF-THEN statement allows you to execute a set of SQL statements based on a specified condition. The following illustrates the syntax of the IF-THEN statement:

IF condition THEN

statements;

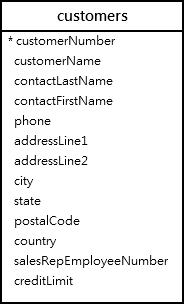
END IF;

Code language: SQL (Structured Query Language) (sql)

In this syntax:

* First, specify a condition to execute the code between the IF-THEN and END IF . If the condition evaluates to TRUE, the statements between IF-THEN and END IF will execute. Otherwise, the control is passed to the next statement following the END IF.
* Second, specify the code that will execute if the condition evaluates to TRUE.

We’ll use the customers table from the sample database for the demonstration:



See the following GetCustomerLevel() stored procedure.

DELIMITER $$

CREATE PROCEDURE GetCustomerLevel(

IN pCustomerNumber INT,

OUT pCustomerLevel VARCHAR(20))

BEGIN

DECLARE credit DECIMAL(10,2) DEFAULT 0;

SELECT creditLimit

INTO credit

FROM customers

WHERE customerNumber = pCustomerNumber;

IF credit > 50000 THEN

SET pCustomerLevel = 'PLATINUM';

END IF;

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

The stored procedure GetCustomerLevel() accepts two parameters: pCustomerNumber and pCustomerLevel.

* First, select creditLimit of the customer specified by the pCustomerNumber from the customers table and store it in the local variable credit.
* Then, set value for the OUT parameter pCustomerLevel to PLATINUM if the credit limit of the customer is greater than 50,000.

This statement finds all customers that have a credit limit greater than 50,000:

SELECT

customerNumber,

creditLimit

FROM

customers

WHERE

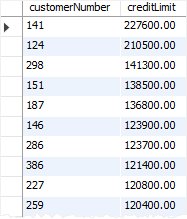
creditLimit > 50000

ORDER BY

creditLimit DESC;

Code language: SQL (Structured Query Language) (sql)

Here is the partial output:



These statements call the GetCustomerLevel() stored procedure for customer 141 and show the value of the OUT parameter pCustomerLevel:

CALL GetCustomerLevel(141, @level);

SELECT @level;

Code language: SQL (Structured Query Language) (sql)



Because the customer 141 has a credit limit greater than 50,000, its level is set to PLATINUM as expected.

## MySQL IF-THEN-ELSE statement

In case you want to execute other statements when the condition in the IF branch does not evaluate to TRUE, you can use the IF-THEN-ELSE statement as follows:

IF condition THEN

statements;

ELSE

else-statements;

END IF;

Code language: SQL (Structured Query Language) (sql)

In this syntax, if the condition evaluates to TRUE, the statements between IF-THEN and ELSE execute. Otherwise, the else-statements between the ELSE and END IF execute.

Let’s modify the GetCustomerLevel() stored procedure.

First, drop the GetCustomerLevel() stored procedure:

DROP PROCEDURE GetCustomerLevel;

Code language: SQL (Structured Query Language) (sql)

Then, create the GetCustomerLevel() stored procedure with the new code:

DELIMITER $$

CREATE PROCEDURE GetCustomerLevel(

IN pCustomerNumber INT,

OUT pCustomerLevel VARCHAR(20))

BEGIN

DECLARE credit DECIMAL DEFAULT 0;

SELECT creditLimit

INTO credit

FROM customers

WHERE customerNumber = pCustomerNumber;

IF credit > 50000 THEN

SET pCustomerLevel = 'PLATINUM';

ELSE

SET pCustomerLevel = 'NOT PLATINUM';

END IF;

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

In this new stored procedure, we include the ELSE branch. If the credit is not greater than 50,000, we set the customer level to NOT PLATINUM in the block between ELSE and END IF.

This query finds customers that have credit limit less than or equal 50,000:

SELECT

customerNumber,

creditLimit

FROM

customers

WHERE

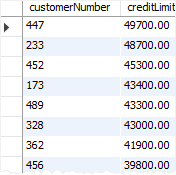
creditLimit <= 50000

ORDER BY

creditLimit DESC;

Code language: SQL (Structured Query Language) (sql)

This picture shows the partial output:



The following statements call the stored procedure for customer number 447  and show the value of the OUT parameter pCustomerLevel:

CALL GetCustomerLevel(447, @level);

SELECT @level;

Code language: SQL (Structured Query Language) (sql)



The credit limit of the customer 447 is less than 50,000, therefore, the statement in the ELSE branch executes and sets the value of the OUT parameter pCustomerLevel to NOT PLATINUM.

## MySQL IF-THEN-ELSEIF-ELSE statement

If you want to execute statements conditionally based on multiple conditions, you use the following IF-THEN-ELSEIF-ELSE statement:

IF condition THEN

statements;

ELSEIF elseif-condition THEN

elseif-statements;

...

ELSE

else-statements;

END IF;

Code language: SQL (Structured Query Language) (sql)

In this syntax, if the condition evaluates to TRUE , the statements in the IF-THEN branch executes; otherwise, the next elseif-condition is evaluated.

If the elseif-condition evaluates to TRUE, the elseif-statement executes; otherwise, the next elseif-condition is evaluated.

The IF-THEN-ELSEIF-ELSE statement can have multiple ELSEIF branches.

If no condition in the IF and ELSE IF evaluates to TRUE, the else-statements in the ELSE branch will execute.

We will modify the GetCustomerLevel() stored procedure to use the IF-THEN-ELSEIF-ELSE statement.

First, drop the GetCustomerLevel() stored procedure:

DROP PROCEDURE GetCustomerLevel;

Code language: SQL (Structured Query Language) (sql)

Then, create the new GetCustomerLevel() stored procedure that uses the the IF-THEN-ELSEIF-ELSE statement.

DELIMITER $$

CREATE PROCEDURE GetCustomerLevel(

IN pCustomerNumber INT,

OUT pCustomerLevel VARCHAR(20))

BEGIN

DECLARE credit DECIMAL DEFAULT 0;

SELECT creditLimit

INTO credit

FROM customers

WHERE customerNumber = pCustomerNumber;

IF credit > 50000 THEN

SET pCustomerLevel = 'PLATINUM';

ELSEIF credit <= 50000 AND credit > 10000 THEN

SET pCustomerLevel = 'GOLD';

ELSE

SET pCustomerLevel = 'SILVER';

END IF;

END $$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

In this stored procedure:

* If the credit is greater than 50,000, the level of the customer is PLATINUM.
* If the credit is less than or equal 50,000 and greater than 10,000, then the level of customer is GOLD.
* Otherwise, the level of the customer is SILVER.

These statements call the stored procedure GetCustomerLevel() and show the level of the customer 447:

CALL GetCustomerLevel(447, @level);

SELECT @level;

Code language: SQL (Structured Query Language) (sql)



If you test the stored procedure with the customer that has a credit limit of 10000 or less, you will get the output as SILVER.

In this tutorial, you have learned how to use MySQL IF statement to conditionally execute a block of code based on specified conditions.

# MySQL CASE Statement

Summary: in this tutorial, you will learn how to use MySQL CASE statements to construct complex conditional statements inside stored procedures.

Besides the [IF](https://www.mysqltutorial.org/mysql-if-statement/) statement, MySQL provides an alternative conditional statement called the CASE statement for constructing conditional statements in stored procedures. The CASE statements make the code more readable and efficient.

The CASE statement has two forms: simpleCASE and searched CASE statements.

Note that if you want to add the if-else logic to an SQL statement, you use the [CASE expression](https://www.mysqltutorial.org/mysql-case-function/) which is different from the CASE statement described in this tutorial.

## Simple CASE statement

The following is the basic syntax of the simple CASE statement:

CASE case\_value

WHEN when\_value1 THEN statements

WHEN when\_value2 THEN statements

...

[ELSE else-statements]

END CASE;

Code language: SQL (Structured Query Language) (sql)

In this syntax, the simple CASE statement sequentially compares the case\_value is with the when\_value1, when\_value2, … until it finds one is equal. When the CASE finds a case\_value equal to a when\_value, it executes statements in the corresponding THEN clause.

If CASE cannot find any when\_value equal to the case\_value, it executes the else-statements in the ELSE clause if the ELSE clause is available.

When the ELSE clause does not exist and the CASE cannot find any when\_value equal to the case\_value, it issues an error:

Case not found for CASE statement

Code language: PHP (php)

Note that the case\_value can be a literal value or an expression. The statements can be one or more SQL statements, and cannot have zero statement.

To avoid the error when the  case\_value does not equal any when\_value, you can use an empty BEGIN END block in the ELSE clause as follows:

CASE case\_value

WHEN when\_value1 THEN ...

WHEN when\_value2 THEN ...

ELSE

BEGIN

END;

END CASE;

Code language: SQL (Structured Query Language) (sql)

The simple CASE statement tests for equality ( =), you cannot use it to test equality with [NULL](https://www.mysqltutorial.org/mysql-null/); because NULL = NULL returns FALSE.

### Simple CASE statement example

The following stored procedure illustrates how to use the simple CASE statement:

DELIMITER $$

CREATE PROCEDURE GetCustomerShipping(

IN pCustomerNUmber INT,

OUT pShipping VARCHAR(50)

)

BEGIN

DECLARE customerCountry VARCHAR(100);

SELECT

country

INTO customerCountry FROM

customers

WHERE

customerNumber = pCustomerNUmber;

CASE customerCountry

WHEN 'USA' THEN

SET pShipping = '2-day Shipping';

WHEN 'Canada' THEN

SET pShipping = '3-day Shipping';

ELSE

SET pShipping = '5-day Shipping';

END CASE;

END$$

DELIMITER ;

Code language: PHP (php)

How it works.

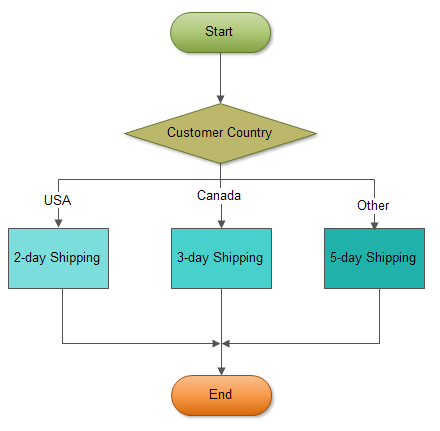
The GetCustomerShipping() stored procedure accepts two parameters: pCustomerNumber as an IN parameter and pShipping as an OUT parameter.

In the stored procedure:

First, select the country of the customer from the customers table by the input customer number.

Second, use the simple CASE statement to determine the shipping time based on the country of the customer. If the customer locates in USA , the shipping time is 2-day shipping . If the customer locates in Canada , the shipping time is 3-day shipping . The customers from other countries have 5-day shipping .

The following flowchart demonstrates the logic of the  CASE statement for determining the shipping time:

[](https://www.mysqltutorial.org/wp-content/uploads/2013/01/mysql-case-statement.png)

This statement calls the stored procedure and passes the customer number 112:

CALL GetCustomerShipping(112,@shipping);

Code language: SQL (Structured Query Language) (sql)

The following statement returns the shipping time of the customer 112:

SELECT @shipping;

Code language: SQL (Structured Query Language) (sql)

Here is the output:

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

| @shipping |

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

| 2-day Shipping |

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

1 row in set (0.00 sec)

Code language: SQL (Structured Query Language) (sql)

## Searched CASE statement

The simple CASE statement only allows you to compare a value with a set of distinct values.

To perform more complex matches such as ranges, you use the searched CASE statement. The searched CASE statement is equivalent to the IF  statement, however, it’s much more readable than the IF statement.

Here is the basic syntax of the searched CASE statement:

CASE

WHEN search\_condition1 THEN statements

WHEN search\_condition1 THEN statements

...

[ELSE else-statements]

END CASE;

Code language: SQL (Structured Query Language) (sql)

In this syntax, searched CASE evaluates each search\_condition in the WHEN clause until it finds a condition that evaluates to TRUE , then it executes the corresponding THEN clause statements.

If no search\_condition evaluates to TRUE, the CASE will execute else-statements in the ELSE clause if an ELSE clause is available.

Similar to the simple CASE statement, if you don’t specify an ELSE clause and no condition is TRUE, MySQL raises the same error:

Case not found for CASE statement

Code language: PHP (php)

MySQL also does not allow you to have empty statements in the THEN or ELSE clause. If you don’t want to handle the logic in the ELSE clause while preventing MySQL from raising an error in case no search\_condition is true, you can use an empty BEGIN END  block in the ELSE clause.

### Searched CASE statement example

The following example demonstrates how to use a searched CASE statement to find customer level SILVER , GOLD or PLATINUM based on customer’s credit limit.

DELIMITER $$

CREATE PROCEDURE GetDeliveryStatus(

IN pOrderNumber INT,

OUT pDeliveryStatus VARCHAR(100)

)

BEGIN

DECLARE waitingDay INT DEFAULT 0;

SELECT

DATEDIFF(requiredDate, shippedDate)

INTO waitingDay

FROM orders

WHERE orderNumber = pOrderNumber;

CASE

WHEN waitingDay = 0 THEN

SET pDeliveryStatus = 'On Time';

WHEN waitingDay >= 1 AND waitingDay < 5 THEN

SET pDeliveryStatus = 'Late';

WHEN waitingDay >= 5 THEN

SET pDeliveryStatus = 'Very Late';

ELSE

SET pDeliveryStatus = 'No Information';

END CASE;

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

How it works.

The stored procedure GetDeliveryStatus() accepts an order number as an IN parameter and returns the delivery status as an OUT parameter.

First, calculate the number of days between the required date and shipped date.

Second, determine the delivery status based on the number of waiting days using the searched CASE statement:

* If the number of waiting days is zero, then the delivery is on time.
* When the number of waiting days is between 1 and 5, the delivery is late.
* When the number of waiting days is more than 5 days, then the delivery is very late.
* If the number of waiting days is NULL or else, the delivery has the status of no information specified in the ELSE clause.

This statement uses the stored procedure GetDeliveryStatus() to get the delivery status of the order 10100 :

CALL GetDeliveryStatus(10100,@delivery);

Code language: SQL (Structured Query Language) (sql)

Here is the result:



## MySQL CASE vs. IF

Both [IF](https://www.mysqltutorial.org/mysql-if-statement/) and CASE statements allow you to execute a block of code based on a specific condition. Choosing between IF or CASE sometimes is just a matter of personal preference. Here are some guidelines:

* A simple CASE statement is more readable and efficient than an IF statement when you compare a single expression against a range of unique values.
* When you check complex expressions based on multiple values, the IF statement is easier to understand.
* If you use the CASE statement, you have to make sure that at least one of the CASE condition is matched. Otherwise, you need to define an [error handler](https://www.mysqltutorial.org/mysql-error-handling-in-stored-procedures/) to catch the error. Note that you do not have to do this with the IF statement.
* In some situations, you can use both IF and CASE to make the code more readable and efficient.

In this tutorial, you have learned how to use two forms of the MySQL CASE statements including simple CASE statement and searched CASE statement.

# MySQL LOOP

Summary: in this tutorial, you will learn how to use MySQL LOOP statement to run a block of code repeatedly based on a condition.

## Introduction to MySQL LOOP statement

The LOOP statement allows you to execute one or more statements repeatedly.

Here is the basic syntax of the LOOP statement:

[begin\_label:] LOOP

statement\_list

END LOOP [end\_label]

Code language: SQL (Structured Query Language) (sql)

The LOOP can have optional labels at the beginning and end of the block.

The LOOP executes the statement\_list repeatedly. The statement\_list may have one or more statements, each terminated by a semicolon (;) statement delimiter.

Typically, you terminate the loop when a condition is satisfied by using the [LEAVE](https://www.mysqltutorial.org/mysql-stored-procedure/mysql-leave/) statement.

This is the typical syntax of the LOOP statement used with LEAVE statement:

[label]: LOOP

...

-- terminate the loop

IF condition THEN

LEAVE [label];

END IF;

...

END LOOP;

Code language: SQL (Structured Query Language) (sql)

The LEAVE statement immediately exits the loop. It works like the break statement in other programming languages like PHP, C/C++, and Java.

In addition to the LEAVE statement, you can use the ITERATE statement to skip the current loop iteration and start a new iteration. The ITERATE is similar to the continue statement in PHP, C/C++, and Java.

## MySQL LOOP statement example

The following statement [creates a stored procedure](https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx) that uses a LOOP loop statement:

DROP PROCEDURE LoopDemo;

DELIMITER $$

CREATE PROCEDURE LoopDemo()

BEGIN

DECLARE x INT;

DECLARE str VARCHAR(255);

SET x = 1;

SET str = '';

loop\_label: LOOP

IF x > 10 THEN

LEAVE loop\_label;

END IF;

SET x = x + 1;

IF (x mod 2) THEN

ITERATE loop\_label;

ELSE

SET str = CONCAT(str,x,',');

END IF;

END LOOP;

SELECT str;

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

In this example:

* The stored procedure constructs a string from the even numbers e.g., 2, 4, and 6.
* The loop\_label  before the LOOPstatement for using with the ITERATE and LEAVE statements.
* If the value of  x is greater than 10, the loop is terminated because of the LEAVEstatement.
* If the value of the x is an odd number, the ITERATE ignores everything below it and starts a new loop iteration.
* If the value of the x is an even number, the block in the ELSEstatement will build the result string from even numbers.

The following statement calls the stored procedure:

CALL LoopDemo();

Code language: SQL (Structured Query Language) (sql)

Here is the output:

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

| str |

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

| 2,4,6,8,10, |

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

1 row in set (0.01 sec)

Query OK, 0 rows affected (0.02 sec)

Code language: JavaScript (javascript)

In this tutorial, you have learned how to use the MySQL LOOP statement to execute a block of code repeatedly based on a condition.

# MySQL WHILE Loop

Summary: in this tutorial, you will learn how to use the MySQL WHILE loop statement to execute one or more statements repeatedly as long as a condition is true.

## Introduction to MySQL WHILE loop statement

The WHILE loop is a loop statement that executes a block of code repeatedly as long as a condition is true.

Here is the basic syntax of the WHILE statement:

[begin\_label:] WHILE search\_condition DO

statement\_list

END WHILE [end\_label]

Code language: SQL (Structured Query Language) (sql)

In this syntax:

First, specify a search condition after the WHILE keyword.

The WHILE checks the search\_condition at the beginning of each iteration.

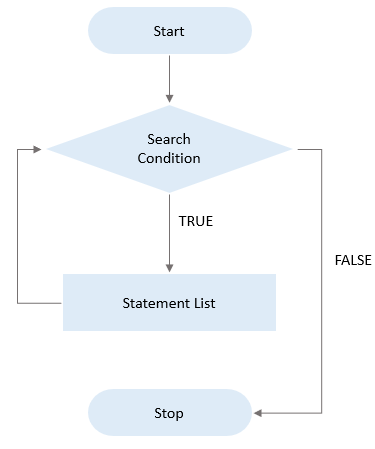
If the search\_condition evaluates to TRUE, the WHILE executes the statement\_list as long as the search\_condition is TRUE.

The WHILE loop is called a pretest loop because it checks the search\_condition before the statement\_list executes.

Second, specify one or more statements that will execute between the DO and END WHILE keywords.

Third, specify optional labels for the WHILE statement at the beginning and end of the loop construct.

The following flowchart illustrates the MySQL WHILE loop statement:



## MySQL WHILE loop statement example

First, [create a table](https://www.mysqltutorial.org/mysql-create-table/) namedcalendars which stores dates and derived date information such as day, month, quarter, and year:

CREATE TABLE calendars(

id INT AUTO\_INCREMENT,

fulldate DATE UNIQUE,

day TINYINT NOT NULL,

month TINYINT NOT NULL,

quarter TINYINT NOT NULL,

year INT NOT NULL,

PRIMARY KEY(id)

);

Code language: SQL (Structured Query Language) (sql)

Second, [create a new stored procedure](https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx) to [insert](https://www.mysqltutorial.org/mysql-insert-statement.aspx) a date into the calendars table:

DELIMITER $$

CREATE PROCEDURE InsertCalendar(dt DATE)

BEGIN

INSERT INTO calendars(

fulldate,

day,

month,

quarter,

year

)

VALUES(

dt,

EXTRACT(DAY FROM dt),

EXTRACT(MONTH FROM dt),

EXTRACT(QUARTER FROM dt),

EXTRACT(YEAR FROM dt)

);

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

Third, create a new stored procedure LoadCalendars() that loads a number of days starting from a start date into the calendars table.

DELIMITER $$

CREATE PROCEDURE LoadCalendars(

startDate DATE,

day INT

)

BEGIN

DECLARE counter INT DEFAULT 1;

DECLARE dt DATE DEFAULT startDate;

WHILE counter <= day DO

CALL InsertCalendar(dt);

SET counter = counter + 1;

SET dt = DATE\_ADD(dt,INTERVAL 1 day);

END WHILE;

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

The stored procedure LoadCalendars() accepts two arguments:

* startDate is the start date inserted into the calendars table.
* day is the number of days that will be loaded starting from the startDate.

In the LoadCalendars() stored procedure:

First, declare a counter and dt variables for keeping immediate values. The default values of counter and dt are 1 and startDate respectively.

Then, check if the counter is less than or equal day, if yes:

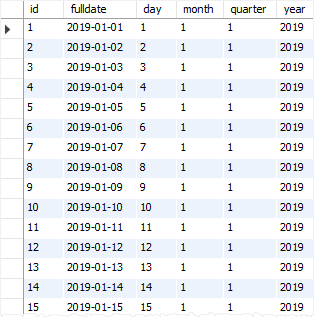
* Call the stored procedure InsertCalendar() to insert a row into the calendars table.
* Increase the counter by one. Also, increase the dt by one day using the DATE\_ADD() function.

The WHILE loop repeatedly inserts dates into the calendars table until the counter is equal to day.

The following statement calls the stored procedure LoadCalendars() to load 31 days into the calendars table starting from January 1st 2019.

CALL LoadCalendars('2019-01-01',31);

Code language: SQL (Structured Query Language) (sql)



In this tutorial, you have learned how to use the MySQL WHILE loop to execute one or more statements repeatedly as long as a condition is true.

# MySQL REPEAT Loop

Summary: in this tutorial, you will learn how to use the MySQL REPEAT statement to execute one or more statements until a search condition is true.

The REPEAT statement executes one or more statements until a search condition is true.

Here is the basic syntax of the REPEAT loop statement:

[begin\_label:] REPEAT

statement

UNTIL search\_condition

END REPEAT [end\_label]

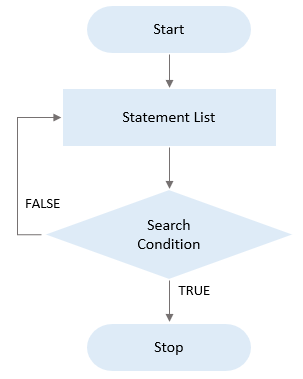
Code language: SQL (Structured Query Language) (sql)

The REPEAT executes the statement until the search\_condition evaluates to true.

The REPEAT checks the search\_condition after the execution of statement, therefore, the statement always executes at least once. This is why the REPEAT is also known as a post-test loop.

The REPEAT statement can have labels at the beginning and at the end. These labels are optional.

The following flowchart illustrates the REPEAT loop:



## MySQL REPEAT loop example

This statement [creates a stored procedure](https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx) called RepeatDemo  that uses the REPEAT statement to concatenate numbers from 1 to 9:

DELIMITER $$

CREATE PROCEDURE RepeatDemo()

BEGIN

DECLARE counter INT DEFAULT 1;

DECLARE result VARCHAR(100) DEFAULT '';

REPEAT

SET result = CONCAT(result,counter,',');

SET counter = counter + 1;

UNTIL counter >= 10

END REPEAT;

-- display result

SELECT result;

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

In this stored procedure:

First, declare two [variables](https://www.mysqltutorial.org/mysql-variables/) counter and result and set their initial values to 1 and blank.

The counter variable is used for counting from 1 to 9 in the loop. And the result variable is used for storing the concatenated string after each loop iteration.

Second, append counter value to the result variable using the [CONCAT()](https://www.mysqltutorial.org/sql-concat-in-mysql.aspx) function until the counter is greater than or equal to 10.

The following statement calls the RepeatDemo() stored procedure:

CALL RepeatDemo();

Code language: SQL (Structured Query Language) (sql)

Here is the output:

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

| result |

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

| 1,2,3,4,5,6,7,8,9, |

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

1 row in set (0.02 sec)

Query OK, 0 rows affected (0.02 sec)

Code language: SQL (Structured Query Language) (sql)

In this tutorial, you will learn how to use the MySQL REPEAT statement to execute one or more statement until a search condition is true.

# MySQL LEAVE

Summary: in this tutorial, you will learn how to the MySQL LEAVE statement to exit a stored program or terminate a loop.

## Introduction to MySQL LEAVE statement

The LEAVE statement exits the flow control that has a given label.

The following shows the basic syntax of the LEAVE statement:

LEAVE label;

Code language: SQL (Structured Query Language) (sql)

In this syntax, you specify the label of the block that you want to exit after the LEAVE keyword.

## Using the LEAVE statement to exit a stored procedure

If the label is the outermost of the [stored procedure](https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx)  or [function](https://www.mysqltutorial.org/mysql-stored-function/) block, LEAVE terminates the stored procedure or function.

The following statement shows how to use the LEAVE statement to exit a stored procedure:

CREATE PROCEDURE sp\_name()

sp: BEGIN

IF condition THEN

LEAVE sp;

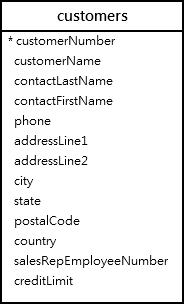
END IF;

-- other statement

END$$

Code language: SQL (Structured Query Language) (sql)

For example, this statement creates a new stored procedure that checks the credit of a given customer in the customers table from the [sample database](https://www.mysqltutorial.org/mysql-sample-database.aspx):



DELIMITER $$

CREATE PROCEDURE CheckCredit(

inCustomerNumber int

)

sp: BEGIN

DECLARE customerCount INT;

-- check if the customer exists

SELECT

COUNT(\*)

INTO customerCount

FROM

customers

WHERE

customerNumber = inCustomerNumber;

-- if the customer does not exist, terminate

-- the stored procedure

IF customerCount = 0 THEN

LEAVE sp;

END IF;

-- other logic

-- ...

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

## Using LEAVE statement in loops

The LEAVE statement allows you to terminate a loop. The general syntax for the LEAVE statement when using in the [LOOP](https://www.mysqltutorial.org/stored-procedures-loop.aspx), REPEAT and [WHILE](https://www.mysqltutorial.org/mysql-stored-procedure/mysql-while-loop/) statements.

Using LEAVE with the LOOP statement:

[label]: LOOP

IF condition THEN

LEAVE [label];

END IF;

-- statements

END LOOP [label];

Code language: SQL (Structured Query Language) (sql)

Using LEAVE with the REPEAT statement:

[label:] REPEAT

IF condition THEN

LEAVE [label];

END IF;

-- statements

UNTIL search\_condition

END REPEAT [label];

Code language: SQL (Structured Query Language) (sql)

Using LEAVE with the WHILE statement:

[label:] WHILE search\_condition DO

IF condition THEN

LEAVE [label];

END IF;

-- statements

END WHILE [label];

Code language: SQL (Structured Query Language) (sql)

The LEAVE causes the current loop specified by the label to be terminated. If a loop is enclosed within another loop, you can break out of both loops with a single LEAVE statement.

### Using LEAVE statement in a loop example

The following stored procedure generates a string of integer with the number from 1 to a random number between 4 and 10:

DELIMITER $$

CREATE PROCEDURE LeaveDemo(OUT result VARCHAR(100))

BEGIN

DECLARE counter INT DEFAULT 1;

DECLARE times INT;

-- generate a random integer between 4 and 10

SET times = FLOOR(RAND()\*(10-4+1)+4);

SET result = '';

disp: LOOP

-- concatenate counters into the result

SET result = concat(result,counter,',');

-- exit the loop if counter equals times

IF counter = times THEN

LEAVE disp;

END IF;

SET counter = counter + 1;

END LOOP;

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

This statement calls the LeaveDemo procedure:

CALL LeaveDemo(@result);

SELECT @result;

Code language: SQL (Structured Query Language) (sql)

Here is one of the outputs:

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

| @result |

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

| 1,2,3,4,5,6,7,8, |

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

1 row in set (0.00 sec)

Code language: SQL (Structured Query Language) (sql)

In this tutorial, you have learned how to the MySQL LEAVE statement to exit a stored program or terminate a loop.

# MySQL Error Handling in Stored Procedures

Summary: in this tutorial, you will learn how to use MySQL handler to handle errors encountered in stored procedures.

When an error occurs inside a stored procedure, it is important to handle it appropriately, such as continuing or exiting the current code block’s execution, and issuing a meaningful error message.

MySQL provides an easy way to define handlers that handle from general conditions such as warnings or exceptions to specific conditions e.g., specific error codes.

## Declaring a handler

To declare a handler, you use the  DECLARE HANDLER statement as follows:

DECLARE action HANDLER FOR condition\_value statement;

Code language: SQL (Structured Query Language) (sql)

If a condition whose value matches the  condition\_value , MySQL will execute the statement and continue or exit the current code block based on the action .

The action accepts one of the following values:

* CONTINUE :  the execution of the enclosing code block ( BEGIN … END ) continues.
* EXIT : the execution of the enclosing code block, where the handler is declared, terminates.

The  condition\_value specifies a particular condition or a class of conditions that activate the handler. The  condition\_value accepts one of the following values:

* A MySQL error code.
* A standard SQLSTATE value. Or it can be an SQLWARNING , NOTFOUND or SQLEXCEPTION condition, which is shorthand for the class of SQLSTATE values. The NOTFOUND condition is used for a [cursor](https://www.mysqltutorial.org/mysql-cursor/) or  SELECT INTO variable\_list statement.
* A named condition associated with either a MySQL error code or SQLSTATE value.

The statement could be a simple statement or a compound statement enclosing by the BEGIN and END keywords.

## MySQL error handling examples

Let’s take some examples of declaring handlers.

The following handler set the value of the  hasError variable to 1 and continue the execution if an SQLEXCEPTION occurs

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION

SET hasError = 1;

Code language: SQL (Structured Query Language) (sql)

The following handler rolls back the previous operations, issues an error message, and exit the current code block in case an error occurs. If you declare it inside the BEGIN END block of a stored procedure, it will terminate the stored procedure immediately.

DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

ROLLBACK;

SELECT 'An error has occurred, operation rollbacked and the stored procedure was terminated';

END;

Code language: SQL (Structured Query Language) (sql)

The following handler sets the value of the  RowNotFound variable to 1 and continues execution if there is no more row to fetch in case of a [cursor](https://www.mysqltutorial.org/mysql-cursor/) or SELECT INTO statement:

DECLARE CONTINUE HANDLER FOR NOT FOUND

SET RowNotFound = 1;

Code language: SQL (Structured Query Language) (sql)

If a duplicate key error occurs, the following handler issues an error message and continues execution.

DECLARE CONTINUE HANDLER FOR 1062

SELECT 'Error, duplicate key occurred';

Code language: SQL (Structured Query Language) (sql)

## MySQL handler example in stored procedures

First, [create a new table](https://www.mysqltutorial.org/mysql-create-table/) named SupplierProductsfor the demonstration:

CREATE TABLE SupplierProducts (

supplierId INT,

productId INT,

PRIMARY KEY (supplierId , productId)

);

Code language: SQL (Structured Query Language) (sql)

The table SupplierProducts stores the relationships between the table suppliers and products. Each supplier may provide many products and each product can be provided by many suppliers. For the sake of simplicity, we don’t create Products and Suppliers tables, as well as the [foreign keys](https://www.mysqltutorial.org/mysql-foreign-key/) in the  SupplierProducts table.

Second, [create a stored procedure](https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx) that inserts product id and supplier id into the SupplierProducts table:

CREATE PROCEDURE InsertSupplierProduct(

IN inSupplierId INT,

IN inProductId INT

)

BEGIN

-- exit if the duplicate key occurs

DECLARE EXIT HANDLER FOR 1062

BEGIN

SELECT CONCAT('Duplicate key (',inSupplierId,',',inProductId,') occurred') AS message;

END;

-- insert a new row into the SupplierProducts

INSERT INTO SupplierProducts(supplierId,productId)

VALUES(inSupplierId,inProductId);

-- return the products supplied by the supplier id

SELECT COUNT(\*)

FROM SupplierProducts

WHERE supplierId = inSupplierId;

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

How it works.

The following exit handler terminates the stored procedure whenever a duplicate key occurs (with code 1062). In addition, it returns an error message.

DECLARE EXIT HANDLER FOR 1062

BEGIN

SELECT CONCAT('Duplicate key (',supplierId,',',productId,') occurred') AS message;

END;

Code language: SQL (Structured Query Language) (sql)

This statement [inserts a row](https://www.mysqltutorial.org/mysql-insert-statement.aspx) into the SupplierProducts table. If a duplicate key occurs, the code in the handler section will execute.

INSERT INTO SupplierProducts(supplierId,productId)

VALUES(supplierId,productId);

Code language: SQL (Structured Query Language) (sql)

Third, call the InsertSupplierProduct() to insert some rows into the SupplierProducts table:

CALL InsertSupplierProduct(1,1);

CALL InsertSupplierProduct(1,2);

CALL InsertSupplierProduct(1,3);

Code language: SQL (Structured Query Language) (sql)

Fourth, attempt to insert a row whose values already exist in the SupplierProducts table:

CALL InsertSupplierProduct(1,3);

Code language: SQL (Structured Query Language) (sql)

Here is the error message:

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

| message |

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

| Duplicate key (1,3) occurred |

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

1 row in set (0.01 sec)

Code language: JavaScript (javascript)

Because the handler is an EXIT handler, the last statement does not execute:

SELECT COUNT(\*)

FROM SupplierProducts

WHERE supplierId = inSupplierId;

Code language: SQL (Structured Query Language) (sql)

If  you change the EXIT in the handler declaration to CONTINUE , you will also get the number of products provided by the supplier:

DROP PROCEDURE IF EXISTS InsertSupplierProduct;

DELIMITER $$

CREATE PROCEDURE InsertSupplierProduct(

IN inSupplierId INT,

IN inProductId INT

)

BEGIN

-- exit if the duplicate key occurs

DECLARE CONTINUE HANDLER FOR 1062

BEGIN

SELECT CONCAT('Duplicate key (',inSupplierId,',',inProductId,') occurred') AS message;

END;

-- insert a new row into the SupplierProducts

INSERT INTO SupplierProducts(supplierId,productId)

VALUES(inSupplierId,inProductId);

-- return the products supplied by the supplier id

SELECT COUNT(\*)

FROM SupplierProducts

WHERE supplierId = inSupplierId;

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

Finally, call the stored procedure again to see the effect of the CONTINUE handler:

CALL InsertSupplierProduct(1,3);

Code language: SQL (Structured Query Language) (sql)

Here is the output:

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

| COUNT(\*) |

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

| 3 |

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

1 row in set (0.01 sec)

Query OK, 0 rows affected (0.02 sec)

Code language: SQL (Structured Query Language) (sql)

## MySQL handler precedence

In case you have multiple handlers that handle the same error, MySQL will call the most specific handler to handle the error first based on the following rules:

* An error always maps to a MySQL error code because in MySQL it is the most specific.
* An SQLSTATE may map to many MySQL error codes, therefore, it is less specific.
* An SQLEXCPETION or an SQLWARNING is the shorthand for a class of SQLSTATES values so it is the most generic.

Based on the handler precedence rules,  MySQL error code handler, SQLSTATE handler and SQLEXCEPTION takes the first, second and third precedence.

Suppose that we have three handlers in the handlers in the stored procedure insert\_article\_tags\_3 :

DROP PROCEDURE IF EXISTS InsertSupplierProduct;

DELIMITER $$

CREATE PROCEDURE InsertSupplierProduct(

IN inSupplierId INT,

IN inProductId INT

)

BEGIN

-- exit if the duplicate key occurs

DECLARE EXIT HANDLER FOR 1062 SELECT 'Duplicate keys error encountered' Message;

DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'SQLException encountered' Message;

DECLARE EXIT HANDLER FOR SQLSTATE '23000' SELECT 'SQLSTATE 23000' ErrorCode;

-- insert a new row into the SupplierProducts

INSERT INTO SupplierProducts(supplierId,productId)

VALUES(inSupplierId,inProductId);

-- return the products supplied by the supplier id

SELECT COUNT(\*)

FROM SupplierProducts

WHERE supplierId = inSupplierId;

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

Call the stored procedure to insert a duplicate key:

CALL InsertSupplierProduct(1,3);

Code language: SQL (Structured Query Language) (sql)

Here is the output:

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

| Message |

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

| Duplicate keys error encountered |

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

1 row in set (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

Code language: JavaScript (javascript)

As you see the MySQL error code handler is called.

## Using a named error condition

Let’s start with an error handler declaration.

DELIMITER $$

CREATE PROCEDURE TestProc()

BEGIN

DECLARE EXIT HANDLER FOR 1146

SELECT 'Please create table abc first' Message;

SELECT \* FROM abc;

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

What does the number 1146 really mean? Imagine you have stored procedures polluted with these numbers all over places; it will be difficult to understand and maintain the code.

Fortunately, MySQL provides you with the DECLARE CONDITION statement that declares a named error condition, which associates with a condition.

Here is the syntax of the DECLARE CONDITION statement:

DECLARE condition\_name CONDITION FOR condition\_value;

Code language: SQL (Structured Query Language) (sql)

The condition\_value  can be a MySQL error code such as 1146 or a SQLSTATE value. The condition\_value is represented by the condition\_name .

After the declaration, you can refer to condition\_name instead of condition\_value .

So you can rewrite the code above as follows:

DROP PROCEDURE IF EXISTS TestProc;

DELIMITER $$

CREATE PROCEDURE TestProc()

BEGIN

DECLARE TableNotFound CONDITION for 1146 ;

DECLARE EXIT HANDLER FOR TableNotFound

SELECT 'Please create table abc first' Message;

SELECT \* FROM abc;

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

As you can see, the code is more obviously and readable than the previous one. Notice that the condition declaration must appear before handler or cursor declarations.

In this tutorial, you have learned how to use MySQL handlers to handle exception or errors occurred in stored procedures.

# Raising Error Conditions with MySQL SIGNAL / RESIGNAL Statements

Summary: in this tutorial, you will learn how to use SIGNAL  and RESIGNAL statements to raise error conditions inside stored procedures.

## MySQL SIGNAL statement

You use the SIGNAL statement to return an error or warning condition to the caller from a stored program e.g., stored procedure, [stored function](https://www.mysqltutorial.org/mysql-stored-function/), [trigger](https://www.mysqltutorial.org/mysql-triggers.aspx) or [event](https://www.mysqltutorial.org/mysql-triggers/modifying-mysql-events/). The SIGNAL  statement provides you with control over which information for returning such as value and messageSQLSTATE.

The following illustrates syntax of the SIGNAL statement:

SIGNAL SQLSTATE | condition\_name;

SET condition\_information\_item\_name\_1 = value\_1,

condition\_information\_item\_name\_1 = value\_2, etc;

Code language: SQL (Structured Query Language) (sql)

Following the SIGNAL keyword is a SQLSTATE value or a condition name declared by the  DECLARE CONDITION statement. Notice that the SIGNAL statement must always specify a SQLSTATE value or a named condition that defined with an  SQLSTATE value.

To provide the caller with information, you use the SET clause. If you want to return multiple condition information item names with values, you need to separate each name/value pair by a comma.

The  condition\_information\_item\_name can be MESSAGE\_TEXT, MYSQL\_ERRORNO, CURSOR\_NAME , etc.

The following stored procedure adds an order line item into an existing sales order. It issues an error message if the order number does not exist.

DELIMITER $$

CREATE PROCEDURE AddOrderItem(

in orderNo int,

in productCode varchar(45),

in qty int,

in price double,

in lineNo int )

BEGIN

DECLARE C INT;

SELECT COUNT(orderNumber) INTO C

FROM orders

WHERE orderNumber = orderNo;

-- check if orderNumber exists

IF(C != 1) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Order No not found in orders table';

END IF;

-- more code below

-- ...

END

Code language: SQL (Structured Query Language) (sql)

First, it counts the orders with the input order number that we pass to the stored procedure.

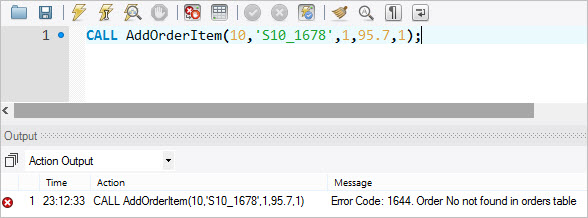
Second, if the number of order is not 1, it raises an error with  SQLSTATE 45000 along with an error message saying that order number does not exist in the orders table.

Notice that 45000 is a generic SQLSTATE value that illustrates an unhandled user-defined exception.

If we call the stored procedure  AddOrderItem() and pass a nonexistent order number, we will get an error message.

CALL AddOrderItem(10,'S10\_1678',1,95.7,1);

Code language: JavaScript (javascript)



## MySQL RESIGNAL statement

Besides the SIGNAL  statement, MySQL also provides the RESIGNAL  statement used to raise a warning or error condition.

The RESIGNAL  statement is similar to SIGNAL  statement in term of functionality and syntax, except that:

* You must use the RESIGNAL  statement within an error or warning handler, otherwise, you will get an error message saying that “RESIGNAL when the handler is not active”. Notice that you can use SIGNAL  statement anywhere inside a stored procedure.
* You can omit all attributes of the RESIGNAL statement, even the SQLSTATE value.

If you use the RESIGNAL statement alone, all attributes are the same as the ones passed to the condition handler.

The following stored procedure changes the error message before issuing it to the caller.

DELIMITER $$

CREATE PROCEDURE Divide(IN numerator INT, IN denominator INT, OUT result double)

BEGIN

DECLARE division\_by\_zero CONDITION FOR SQLSTATE '22012';

DECLARE CONTINUE HANDLER FOR division\_by\_zero

RESIGNAL SET MESSAGE\_TEXT = 'Division by zero / Denominator cannot be zero';

--

IF denominator = 0 THEN

SIGNAL division\_by\_zero;

ELSE

SET result := numerator / denominator;

END IF;

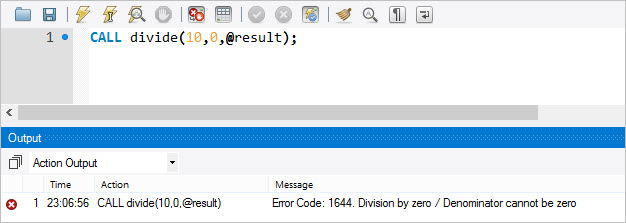
END

Code language: SQL (Structured Query Language) (sql)

Let’s call the  Divide() stored procedure.

CALL Divide(10,0,@result);

Code language: CSS (css)



In this tutorial, we have shown you how to raise error conditions inside stored programs using SIGNAL  and  RESIGNAL statements.

# MySQL Cursor

Summary: in this tutorial, you will learn how to use MySQL cursor in stored procedures to iterate through a result set returned by a SELECT statement.

## Introduction to MySQL cursor

To handle a result set inside a [stored procedure](https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx), you use a cursor. A cursor allows you to [iterate](https://www.mysqltutorial.org/stored-procedures-loop.aspx) a set of rows returned by a query and process each row individually.

MySQL cursor is read-only, non-scrollable and asensitive.

* Read-only: you cannot update data in the underlying table through the cursor.
* Non-scrollable: you can only fetch rows in the order determined by the [SELECT](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statement. You cannot fetch rows in the reversed order. In addition, you cannot skip rows or jump to a specific row in the result set.
* Asensitive: there are two kinds of cursors: asensitive cursor and insensitive cursor. An asensitive cursor points to the actual data, whereas an insensitive cursor uses a temporary copy of the data. An asensitive cursor performs faster than an insensitive cursor because it does not have to make a temporary copy of data. However, any change that made to the data from other connections will affect the data that is being used by an asensitive cursor, therefore, it is safer if you do not update the data that is being used by an asensitive cursor. MySQL cursor is asensitive.

You can use MySQL cursors in [stored procedures](https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx), [stored functions](https://www.mysqltutorial.org/mysql-stored-function/), and [triggers](https://www.mysqltutorial.org/mysql-triggers.aspx).

## Working with MySQL cursor

First, declare a cursor by using the DECLARE statement:

DECLARE cursor\_name CURSOR FOR SELECT\_statement;

Code language: SQL (Structured Query Language) (sql)

The cursor declaration must be after any [variable](https://www.mysqltutorial.org/variables-in-stored-procedures.aspx) declaration. If you declare a cursor before the variable declarations, MySQL will issue an error. A cursor must always associate with a SELECT statement.

Next, open the cursor by using the OPEN statement. The OPEN statement initializes the result set for the cursor, therefore, you must call the OPEN statement before fetching rows from the result set.

OPEN cursor\_name;

Code language: SQL (Structured Query Language) (sql)

Then, use the FETCH statement to retrieve the next row pointed by the cursor and move the cursor to the next row in the result set.

FETCH cursor\_name INTO variables list;

Code language: SQL (Structured Query Language) (sql)

After that, check if there is any row available before fetching it.

Finally, deactivate the cursor and release the memory associated with it  using the CLOSE statement:

CLOSE cursor\_name;

Code language: SQL (Structured Query Language) (sql)

It is a good practice to always close a cursor when it is no longer used.

When working with MySQL cursor, you must also declare a NOT FOUND handler to handle the situation when the cursor could not find any row.

Because each time you call the FETCH statement, the cursor attempts to read the next row in the result set. When the cursor reaches the end of the result set, it will not be able to get the data, and a condition is raised. The handler is used to handle this condition.

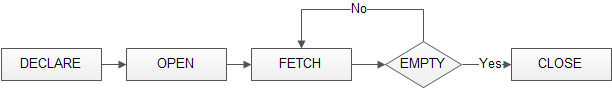
To declare a NOT FOUND handler, you use the following syntax:

DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;

Code language: SQL (Structured Query Language) (sql)

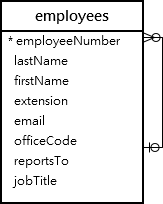
The finished is a variable to indicate that the cursor has reached the end of the result set. Notice that the handler declaration must appear after variable and cursor declaration inside the stored procedures.

The following diagram illustrates how MySQL cursor works.



## MySQL Cursor Example

We’ll develop a stored procedure that creates an email list of all employees in the employees table in the [sample database](https://www.mysqltutorial.org/mysql-sample-database.aspx).



First, declare some variables, a cursor for looping over the emails of employees, and a NOT FOUND handler:

DECLARE finished INTEGER DEFAULT 0;

DECLARE emailAddress varchar(100) DEFAULT "";

-- declare cursor for employee email

DEClARE curEmail

CURSOR FOR

SELECT email FROM employees;

-- declare NOT FOUND handler

DECLARE CONTINUE HANDLER

FOR NOT FOUND SET finished = 1;

Code language: SQL (Structured Query Language) (sql)

Next, open the cursor by using the OPEN statement:

OPEN curEmail;

Code language: SQL (Structured Query Language) (sql)

Then, iterate the email list, and [concatenate](https://www.mysqltutorial.org/sql-concat-in-mysql.aspx) all emails where each email is separated by a semicolon(;):

getEmail: LOOP

FETCH curEmail INTO emailAddress;

IF finished = 1 THEN

LEAVE getEmail;

END IF;

-- build email list

SET emailList = CONCAT(emailAddress,";",emailList);

END LOOP getEmail;

Code language: SQL (Structured Query Language) (sql)

After that, inside the loop, we used the finished variable to check if there is an email in the list to terminate the loop.

Finally, close the cursor using the CLOSE statement:

CLOSE email\_cursor;

Code language: SQL (Structured Query Language) (sql)

The createEmailList stored procedure is as follows:

DELIMITER $$

CREATE PROCEDURE createEmailList (

INOUT emailList varchar(4000)

)

BEGIN

DECLARE finished INTEGER DEFAULT 0;

DECLARE emailAddress varchar(100) DEFAULT "";

-- declare cursor for employee email

DEClARE curEmail

CURSOR FOR

SELECT email FROM employees;

-- declare NOT FOUND handler

DECLARE CONTINUE HANDLER

FOR NOT FOUND SET finished = 1;

OPEN curEmail;

getEmail: LOOP

FETCH curEmail INTO emailAddress;

IF finished = 1 THEN

LEAVE getEmail;

END IF;

-- build email list

SET emailList = CONCAT(emailAddress,";",emailList);

END LOOP getEmail;

CLOSE curEmail;

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

You can test the createEmailList stored procedure using the following script:

SET @emailList = "";

CALL createEmailList(@emailList);

SELECT @emailList;

Code language: SQL (Structured Query Language) (sql)

In this tutorial, we have shown you how to use MySQL cursor to iterate a result set and process each row accordingly.

# MySQL Stored Function

Summary: in this tutorial, you will learn how to create stored functions using the CREATE FUNCTION statement.

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](https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx), you can use a stored function in SQL statements wherever an expression is used. This helps improve the readability and maintainability of the procedural code.

To create a stored function, you use the CREATE FUNCTION statement.

## MySQL CREATE FUNCTION syntax

The following illustrates the basic syntax for creating a new stored function:

DELIMITER $$

CREATE FUNCTION function\_name(

param1,

param2,…

)

RETURNS datatype

[NOT] DETERMINISTIC

BEGIN

-- statements

END $$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

In this syntax:

First, specify the name of the stored function that you want to create after CREATE FUNCTION  keywords.

Second, list all [parameters](https://www.mysqltutorial.org/stored-procedures-parameters.aspx) of the stored function inside the parentheses followed by the function name. By default, all parameters are the IN parameters. You cannot specify IN , OUT or INOUT modifiers to parameters

Third, specify the data type of the return value in the RETURNS statement, which can be any valid [MySQL data types](https://www.mysqltutorial.org/mysql-data-types.aspx).

Fourth, specify if a function is deterministic or not using the DETERMINISTIC keyword.

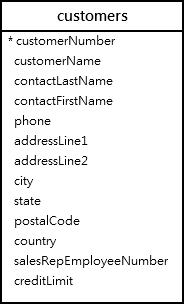
A deterministic function always returns the same result for the same input parameters whereas a non-deterministic function returns different results for the same input parameters.

If you don’t use DETERMINISTIC or NOT DETERMINISTIC, MySQL uses the NOT DETERMINISTIC option by default.

Fifth, write the code in the body of the stored function in the BEGIN END block. Inside the body section, you need to specify at least one RETURN statement. The RETURN statement returns a value to the calling programs. Whenever the RETURN statement is reached, the execution of the stored function is terminated immediately.

## MySQL CREATE FUNCTION example

Let’s take the example of creating a stored function. We will use the customers table in the [sample database](https://www.mysqltutorial.org/mysql-sample-database.aspx) 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

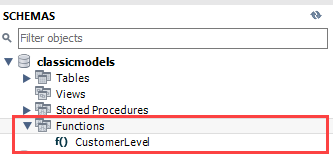
RETURN (customerLevel);

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

Once the function is created, you can view it in MySQL Workbench under the Functions section:



Or you can view all stored functions in the current classicmodels database by using the SHOW FUNCTION STATUS as follows:

SHOW FUNCTION STATUS

WHERE db = 'classicmodels';

Code language: SQL (Structured Query Language) (sql)



### 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;

Code language: SQL (Structured Query Language) (sql)



### Calling a stored function in a stored procedure

The following statement [creates a new stored procedure](https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx) that calls the CustomerLevel() stored function:

DELIMITER $$

CREATE PROCEDURE GetCustomerLevel(

IN customerNo INT,

OUT customerLevel VARCHAR(20)

)

BEGIN

DECLARE credit DEC(10,2) DEFAULT 0;

-- get credit limit of a customer

SELECT

creditLimit

INTO credit

FROM customers

WHERE

customerNumber = customerNo;

-- call the function

SET customerLevel = CustomerLevel(credit);

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

The following illustrates how to call the GetCustomerLevel() stored procedure:

CALL GetCustomerLevel(-131,@customerLevel);

SELECT @customerLevel;

Code language: SQL (Structured Query Language) (sql)

It’s important to notice that if a stored function contains SQL statements that query data from tables, then you should not use it in other SQL statements; otherwise, the stored function will slow down the speed of the query.

In this tutorial, you have learned how to create a stored function to encapsulate the common formula or business rules.

# MySQL DROP FUNCTION

Summary: in this tutorial, you will learn how to use the MySQL DROP FUNCTION statement to drop a stored function.

## Introduction to MySQL DROP FUNCTION statement

The DROP FUNCTION statement drops a [stored function](https://www.mysqltutorial.org/mysql-stored-function/). Here is the syntax of the DROP FUNCTION statement:

DROP FUNCTION [IF EXISTS] function\_name;

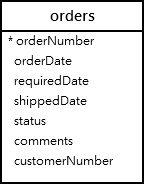
Code language: SQL (Structured Query Language) (sql)

In this syntax, you specify the name of the stored function that you want to drop after the DROP FUNCTION keywords.

The IF EXISTS option allows you to conditionally drop a stored function if it exists. It prevents an error from arising if the function does not exist.

## MySQL DROP FUNCTION example

We’ll use the orders table in the [sample database](https://www.mysqltutorial.org/mysql-sample-database.aspx) for the demonstration.



First, [create a new function](https://www.mysqltutorial.org/mysql-stored-function/) called OrderLeadTime that calculates the number of days between ordered date and required date:

DELIMITER $$

CREATE FUNCTION OrderLeadTime (

orderDate DATE,

requiredDate DATE

)

RETURNS INT

DETERMINISTIC

BEGIN

RETURN requiredDate - orderDate;

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

Second, use the DROP FUNCTION statement to drop the function OrderLeadTime:

DROP FUNCTION OrderLeadTime;

Code language: SQL (Structured Query Language) (sql)

Third, use the DROP FUNCTION to drop a non-existing function:

DROP FUNCTION IF EXISTS NonExistingFunction;

Code language: SQL (Structured Query Language) (sql)

MySQL issued a warning:

0 row(s) affected, 1 warning(s): 1305 FUNCTION classicmodels.NonExistingFunction does not exist

Code language: CSS (css)

If you want to view the warning in detail, use the SHOW WARNINGS statement:

SHOW WARNINGS;

Code language: SQL (Structured Query Language) (sql)



In this tutorial, you will learn how to use the DROP FUNCTION statement to drop a stored function.

# Listing Stored Functions

Summary: in this tutorial, you will learn how to show stored functions from databases by using the SHOW FUNCTION STATUS or querying the data dictionary.

## Listing stored functions using SHOW FUNCTION STATUS statement

The SHOW FUNCTION STATUS is like the [SHOW PROCEDURE STATUS](https://www.mysqltutorial.org/listing-stored-procedures-in-mysql-database.aspx) but for the stored functions.

Here is the basic syntax of the SHOW FUNCTION STATUS statement:

SHOW FUNCTION STATUS

[LIKE 'pattern' | WHERE search\_condition];

Code language: SQL (Structured Query Language) (sql)

The SHOW FUNCTION STATUS statement returns all characteristics of stored functions. The following statement shows all stored functions in the current MySQL server:

SHOW FUNCTION STATUS;

Code language: SQL (Structured Query Language) (sql)

Note that the SHOW FUNCTION STATUS only shows the function that you have a privilege to access.

If you just want to show stored functions in a particular database, you can use a [WHERE](https://www.mysqltutorial.org/mysql-where/) clause in the  SHOW FUNCTION STATUS as shown in the following statement:

SHOW FUNCTION STATUS

WHERE search\_condition;

Code language: SQL (Structured Query Language) (sql)

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

SHOW FUNCTION STATUS

WHERE db = 'classicmodels';

Code language: SQL (Structured Query Language) (sql)

If you want to find the stored functions whose names contain a specific word, you can use the [LIKE](https://www.mysqltutorial.org/mysql-like/) clause:

SHOW FUNCTION STATUS

LIKE '%pattern%';

Code language: SQL (Structured Query Language) (sql)

The following statement shows all stored functions whose names contain the word Customer:

SHOW FUNCTION STATUS LIKE '%Customer%';

Code language: SQL (Structured Query Language) (sql)

## Listing stored functions using the data dictionary

MySQL data dictionary has a routines table that stores information about the stored functions of all databases in the current MySQL server.

This query finds all stored functions in a particular database:

SELECT

routine\_name

FROM

information\_schema.routines

WHERE

routine\_type = 'FUNCTION'

AND routine\_schema = '<database\_name>';

Code language: SQL (Structured Query Language) (sql)

For example, the following statement returns all stored functions in the classicmodels database:

SELECT

routine\_name

FROM

information\_schema.routines

WHERE

routine\_type = 'FUNCTION'

AND routine\_schema = 'classicmodels';

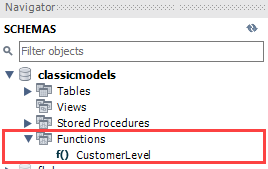
Code language: SQL (Structured Query Language) (sql)

## Showing stored functions using MySQL Workbench

If you use MySQL Workbench, you can view all stored functions from a database.

Step 1. Connect to the database that you want to show the stored functions.

Step 2. Open the **Functions** menu, you will see a list of functions which belong to the database.



In this tutorial, you have learned how to show stored functions in a database by using the SHOW FUNCTION STATUS statement and querying from the data dictionary.

# MySQL Stored Object Access Control

Summary: in this tutorial, you will learn about the stored object access control in MySQL.

In MySQL, stored routines (stored procedures and [functions](https://www.mysqltutorial.org/mysql-stored-function/)), [triggers](https://www.mysqltutorial.org/mysql-triggers.aspx), [events](https://www.mysqltutorial.org/mysql-triggers/working-mysql-scheduled-event/), and [views](https://www.mysqltutorial.org/mysql-views-tutorial.aspx) execute within a security context which determines their privileges.

MySQL uses DEFINER and SQL SECURITY characteristics to control these privileges.

## The DEFINER attribute

When you define a stored routine such as a stored procedure or function, you can optionally specify the DEFINER attribute, which is the name of a MySQL account:

CREATE [DEFINER=user] PROCEDURE spName(parameter\_list)

...

CREATE [DEFINER=user] FUNCTION sfName()

...

Code language: SQL (Structured Query Language) (sql)

If you skip the DEFINER attribute, the default is the current user account.

You can specify any account in the DEFINER attribute if you have the SUPER or SET\_USER\_ID privilege. If you specify the user account that does not exist, MySQL will issue a warning.

As of MySQL 8.0.16, you must have the SYSTEM\_USER privilege in order to set the DEFINER attribute for a stored object to a user account that has the SYSTEM\_USER privilege.

## The SQL SECURITY characteristic

Stored routines (stored procedures and functions) and views can include an SQL SECURITY clause with a value of DEFINER or INVOKER:

CREATE [DEFINER=user] PROCEDURE spName(parameter\_list)

SQL SECURITY [DEFINER | INVOKER]

...

CREATE [DEFINER=user] FUNCTION sfName(parameter\_list)

SQL SECURITY [DEFINER | INVOKER]

...

Code language: SQL (Structured Query Language) (sql)

### SQL SECURITY DEFINER

When you use the SQL SECURITY DEFINER for a stored object, it will execute in definer security context with the privilege of the user specified in by the DEFINER attribute.

Note that the user that invokes the stored object (or invoker) may not have the same privilege as the definer.

In case the invoker has the least privilege and the definer has most privileges, the invoker can perform operations above its privilege within the stored object.

### SQL SECURITY INVOKER

If you use the SQL SECURITY INVOKER for a stored routine or view, it will operate within the privileges of the invoker.

The DEFINER attribute has no effect during object execution.

## Stored object access control example

First, [create a new database](https://www.mysqltutorial.org/mysql-create-database/) called testdb:

CREATE DATABASE testdb;

Code language: SQL (Structured Query Language) (sql)

Second, [select the database](https://www.mysqltutorial.org/mysql-select-database/) testdb to work with:

USE testdb;

Code language: SQL (Structured Query Language) (sql)

Third, [create a new table](https://www.mysqltutorial.org/mysql-create-table/) called messages:

CREATE TABLE messages (

id INT AUTO\_INCREMENT,

message VARCHAR(100) NOT NULL,

PRIMARY KEY (id)

);

Code language: SQL (Structured Query Language) (sql)

Fourth, [create a stored procedure](https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx) that [inserts a new row](https://www.mysqltutorial.org/mysql-insert-statement.aspx) into the messages table:

DELIMITER $$

CREATE DEFINER = root@localhost PROCEDURE InsertMessage(

msg VARCHAR(100)

)

SQL SECURITY DEFINER

BEGIN

INSERT INTO messages(message)

VALUES(msg);

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

In this stored procedure, the definer is root@localhost that is the superuser which has all privileges.

The SQL Security is set to the definer. It means that any user account which calls this stored procedure will execute with all privileges of the definer i.e., root@localhost user account.

Fifth, create a new user named dev@localhost:

CREATE USER dev@localhost

IDENTIFIED BY 'Abcd1234';

Code language: SQL (Structured Query Language) (sql)

Sixth, grant the EXECUTE privilege to dev@localhost so that it can execute any stored procedure in the testdb database:

GRANT EXECUTE ON testdb.\*

TO dev@localhost;

Code language: SQL (Structured Query Language) (sql)

Seventh, use the dev@localhost to log in to the MySQL Server:

mysql -u dev@localhost -p

Code language: SQL (Structured Query Language) (sql)

Eight, use the SHOW DATABASES to display the database that dev@localhost can access:

mysql> show databases;

Code language: SQL (Structured Query Language) (sql)

Here is the list:

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

| Database |

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

| information\_schema |

| testdb |

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

2 rows in set (0.00 sec)

Code language: SQL (Structured Query Language) (sql)

Ninth, select the testdb database:

mysql> use testdb;

Code language: SQL (Structured Query Language) (sql)

Tenth, call the InsertMessage procedure to insert a row into the messages table:

mysql> call InsertMessage('Hello World');

Code language: SQL (Structured Query Language) (sql)

This is the output:

Query OK, 1 row affected (0.01 sec)

Code language: SQL (Structured Query Language) (sql)

Even though dev@localhost does not have any privilege on the messages table, it can insert a new row into that table successfully via the stored procedure because the stored procedure executes in the security context of the root@localhost user account.

Eleventh, go to the root’s session and create a stored procedure that updates the messages table:

DELIMITER $$

CREATE DEFINER=root@localhost

PROCEDURE UpdateMessage(

msgId INT,

msg VARCHAR(100)

)

SQL SECURITY INVOKER

BEGIN

UPDATE messages

SET message = msg

WHERE id = msgId;

END$$

DELIMITER ;

Code language: SQL (Structured Query Language) (sql)

The UpdateMessage has the security context of INVOKER who will call this stored procedure.

Twelfth, go to the dev@localhost‘s session and call the UpdateMessage() stored procedure:

mysql> call UpdateMessage(1,'Good Bye');

Code language: SQL (Structured Query Language) (sql)

This time the UpdateMessage() stored procedure executes with the privileges of the caller which is dev@localhost.

Because dev@localhost does not have any privileges on the messages table, MySQL issues an error and rejects the update:

ERROR 1142 (42000): UPDATE command denied to user 'dev'@'localhost' for table 'messages'

Code language: SQL (Structured Query Language) (sql)

In this tutorial, you have learned about the MySQL stored object access control.

https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx

# MySQL - Stored Procedures and Triggers

* [MySQL](https://www.careerride.com/MySQL-frequently-asked-questions.aspx) >> [MySQL - Part 2](https://www.careerride.com/MySQL-interview-questions-part-2.aspx)
* [Next Page »](https://www.careerride.com/MySQL-primary-keys-and-auto-increment-fields.aspx)

|  |
| --- |
| MySQL - Stored Procedures and Triggers When multiple applications need to perform common database operations, stored procedures can be used. It is a set of sql statements stored on the server. It has a name, parameters and some sql statements.  **Example:**  CREATE OR REPLACE PROCEDURE sample (x VARCHAR) IS "BEGIN " "INSERT INTO sample\_table VALUES(x); " "END;";  A trigger is a set of code which is executed in response to some event.  E.g Update employee\_perfomance table when a new task is inserted in task table. Here, the trigger is “update” and the event is “inserting a new row”.  **Example:**  CREATE TRIGGER reminder ON titles FOR INSERT, UPDATE AS some\_function (50009, 16, 10) MySQL - Stored Procedures and Triggers **Stored Procedures:**  A set of SQL statements is called a stored procedure, which can be compiled and stored in the server. The purpose of stored procedures is that, reissuing the entire queries is avoided. The query is parsed only once, thus provides better performance and information passed between server and the client is very less.  **Triggers:**  A trigger is fired when a particular event occurs. It is also a stored procedure. A stored procedure can be installed which triggers automatically every time a record is deleted or updated or inserted. |

# What is stored procedure and how can we create MySQL stored procedures?

Stored procedure, in the context of regular computing language, may be defined as a subroutine like a subprogram that is stored in a database. In the context of MySQL, it is a segment of declarative SQL statements stored inside the database catalog. Before writing stored procedures in MySQL, we must have to check the version because MySQL 5 introduces stored procedure. Following is the syntax for creating a stored procedure −

## Syntax

CREATE [DEFINER = { user | CURRENT\_USER }]

PROCEDURE sp\_name ([proc\_parameter[,...]])

[characteristic ...] routine\_body

proc\_parameter: [ IN | OUT | INOUT ] param\_name type

type:

Any valid MySQL data type

characteristic:

COMMENT 'string'

| LANGUAGE SQL

| [NOT] DETERMINISTIC

| { CONTAINS SQL | NO SQL | READS SQL DATA

| MODIFIES SQL DATA }

| SQL SECURITY { DEFINER | INVOKER }

routine\_body:

Valid SQL routine statement

## Example

Following is an example in which we created a simple procedure to get all the records from the table ‘student\_info’ which have the following data −

mysql> select \* from student\_info;

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

| id  | Name    | Address    | Subject    |

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

| 100 | Aarav   | Delhi      | Computers  |

| 101 | YashPal | Amritsar   | History    |

| 105 | Gaurav  | Jaipur     | Literature |

| 110 | Rahul   | Chandigarh | History    |

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

4 rows in set (0.00 sec)

Now, with the help of following queries we created the stored procedure named allrecords()

mysql> Delimiter //

mysql> Create Procedure allrecords()

    -> BEGIN

    -> Select \* from Student\_info;

    -> END//

Query OK, 0 rows affected (0.02 sec)

mysql> DELIMITER ;

# What is stored procedure and how can we create MySQL stored procedures?

Stored procedure, in the context of regular computing language, may be defined as a subroutine like a subprogram that is stored in a database. In the context of MySQL, it is a segment of declarative SQL statements stored inside the database catalog. Before writing stored procedures in MySQL, we must have to check the version because MySQL 5 introduces stored procedure. Following is the syntax for creating a stored procedure −

## Syntax

CREATE [DEFINER = { user | CURRENT\_USER }]

PROCEDURE sp\_name ([proc\_parameter[,...]])

[characteristic ...] routine\_body

proc\_parameter: [ IN | OUT | INOUT ] param\_name type

type:

Any valid MySQL data type

characteristic:

COMMENT 'string'

| LANGUAGE SQL

| [NOT] DETERMINISTIC

| { CONTAINS SQL | NO SQL | READS SQL DATA

| MODIFIES SQL DATA }

| SQL SECURITY { DEFINER | INVOKER }

routine\_body:

Valid SQL routine statement

## Example

Following is an example in which we created a simple procedure to get all the records from the table ‘student\_info’ which have the following data −

mysql> select \* from student\_info;

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

| id  | Name    | Address    | Subject    |

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

| 100 | Aarav   | Delhi      | Computers  |

| 101 | YashPal | Amritsar   | History    |

| 105 | Gaurav  | Jaipur     | Literature |

| 110 | Rahul   | Chandigarh | History    |

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

4 rows in set (0.00 sec)

Now, with the help of following queries we created the stored procedure named allrecords()

mysql> Delimiter //

mysql> Create Procedure allrecords()

    -> BEGIN

    -> Select \* from Student\_info;

    -> END//

Query OK, 0 rows affected (0.02 sec)

mysql> DELIMITER ;

# How can we alter a MySQL stored procedure?

If we have ALTER ROUTINE privileges for the procedure then with the help of ALTER PROCEDURE statement we can alter a MySQL stored procedure. To demonstrate it we are taking an example of a stored procedure named ‘delete\_studentinfo’ which have the following create a statement −

mysql> SHOW CREATE PROCEDURE Delete\_studentinfo\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

           Procedure: Delete\_studentinfo

            sql\_mode: ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_

ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION

    Create Procedure: CREATE DEFINER=`root`@`localhost` PROCEDURE `Delete\_studentinfo`( IN p\_id INT)

BEGIN

DELETE FROM student\_info

WHERE ID=p\_id;

END

character\_set\_client: cp850

collation\_connection: cp850\_general\_ci

  Database Collation: latin1\_swedish\_ci

1 row in set (0.01 sec)

In the above result set, it is clear that there is no comment in the stored procedure ‘delete\_studentinfo’. Now, with the help of ALTER PROCEDURE statement, we can add a comment as follows −

mysql> ALTER PROCEDURE Delete\_studentinfo

   -> COMMENT 'deleting the record'//

Query OK, 0 rows affected (0.03 sec)

Now, it can be confirmed, that comments have been added in the stored procedure, with the help of result set of the following query −

mysql> SHOW CREATE PROCEDURE Delete\_studentinfo\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

           Procedure: Delete\_studentinfo

            sql\_mode: ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_

ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION

    Create Procedure: CREATE DEFINER=`root`@`localhost` PROCEDURE `Delete\_studen

tinfo`( IN p\_id INT)

   COMMENT 'deleting the record'

BEGIN

DELETE FROM student\_info

WHERE ID=p\_id;

END

character\_set\_client: cp850

collation\_connection: cp850\_general\_ci

  Database Collation: latin1\_swedish\_ci

1 row in set (0.00 sec)

# How can we write MySQL handler in a stored procedure?

Whenever an exception occurred in MySQL stored procedure, it is very important to handle it by throwing a proper error message. Suppose, if we do not handle the exception, there would be a chance to fail application with that certain exception in a stored procedure. MySQL provides a handler to handle the exception in the stored procedure. Followings are the four kinds of MySQL handlers which can be used in a stored procedure −

#### **DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SELECT 'got an error';**

The above handler will throw an error message and continues the execution.

#### **DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET got\_error=1;**

The above handler will set the variable got\_error to 1 and continues the execution.

#### **DECLARE EXIT HANDLER FOR SQLEXCEPTION SET got\_error=1;**

The above handler will set the variable got\_error to 1 and terminates the execution.

#### **DECLARE EXIT HANDLER FOR SQLSTATE '23000' SET got\_error=1;**

The above handler will throw a default MySQL error message and terminates the execution by setting the variable got\_error to 1.

# How can we invoke MySQL stored procedure?

With the help of the CALL statement, we can invoke the MySQL stored procedure. To make it understand we are invoking the allrecords() stored procedure −

mysql> CALL allrecords();

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

| id   | Name    | Address    | Subject    |

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

| 100  | Aarav   | Delhi      | Computers  |

| 101  | YashPal | Amritsar   | History    |

| 105  | Gaurav  | Jaipur     | Literature |

| 110  | Rahul   | Chandigarh | History    |

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

4 rows in set (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

https://www.tutorialspoint.com/How-can-we-invoke-MySQL-stored-procedure