How to create and use a MySQL stored procedure

In this guide we are going to see how to create and configure **stored procedures in MySQL** . Stored procedures are often confused with **stored functions** , but they are two different concepts. For example, procedures must be invoked with the **CALL statement** , while it is possible to use any function stored directly in an SQL statement.

Procedures can be used for an infinite number of tasks, allowing you to better organize your code and preserve optimal **data integrity** . The use of procedures also usually results in **improved performance** in relatively complex tasks. As with stored functionals, we must not forget that the use of stored procedures will also improve the readability of the code.

Contents

What is a stored procedure

MySQL stored procedure is nothing more than a piece of code that you can save and reuse. It is useful when you repeat the same task repeatedly, being a good method to encapsulate the code. As with functions, it can also accept data as parameters, so it acts based on them.

In order to create a stored procedure you need to have **INSERT** and **DELETE permissions** on the database.

Syntax

The **syntax** of a stored procedure is as follows:

```
CREATE PROCEDURE procedure_name
ACE
sql_statements
GO;
```

To **execute** a stored procedure we invoke it like this:

```
EXEC procedure_name (param1, param 2, .... );
```

Oracle version of SQL we can also invoke the procedure like this:

```
CALL procedure_name (param1, param 2, .... );
```

Parameters

As you have seen, the parameters are defined separated by a comma. The parameters of MySQL stored procedures can be of three types:

- **IN** : It is the type of parameter that is used by default. The application or code that invokes the procedure will have to pass an argument for this parameter. The procedure will work with a copy of its value, with the parameter having its original value at the end of the execution of the procedure.
- **OUT** : The value of this parameter can be changed in the procedure, and its modified value will also be sent back to the code or program that invokes the procedure.
- **INOUT** : It is a mixture of the two previous concepts. The application or code that invokes the procedure can pass a value to it, returning the modified value upon completion of execution. If you find it confusing, take a look at the example you will see below.

Advantages and disadvantages of its use

Like any technology, the use of stored procedures has its pros and cons.

Advantages

By reducing the load on the upper layers of the application , **network traffic** is reduced and, if the use of stored procedures is correct, can **improve performance** .

By **encapsulating operations** in the SQL code itself, we ensure that data access is consistent between all applications that use it.

In terms of security, it is possible **to limit user access permissions** to stored procedures and not to tables directly. In this way we avoid problems derived from a poorly programmed application that misuses the tables.

Disadvantages

As with all technology, we have to train to learn how to create procedures, so there is a certain learning curve.

Another possible problem can occur with migrations. Not all database management systems use the procedures in the same way, which reduces code **portability .**

Database configuration

As an example, we are going to configure a database on which we will then create a stored procedure. We'll use the command line, but you can also create a **MySQL database** using other methods.

Connect to MySQL with this command, replacing *user* with your username.

```
mysql -u user -p
```

Let's create a database named ***example_base*** . To do this, follow these steps:

1. Use the create statement to create the database:

   ```
   CREATE DATABASE example_base ;
   ```

2. Now use the following command to select the database:

   ```
   USE example_base ;
   ```

   If you use phpMyAdmin , you simply click on the name of the database, which in our case is *example_base* .

3. Now create a table in the database that we will call ***products*** . To do this, run the following command:

   ```
   CREATE TABLE products (
   id INT NOT NULL AUTO_INCREMENT ,
   name VARCHAR ( 20 ) NOT NULL ,
   state VARCHAR ( 20 ) NOT NULL DEFAULT 'available' ,
   FLOAT NOT NULL price DEFAULT 0.0 ,
      PRIMARY KEY (id)
   ```

```
);
```

4. Now let's insert some test data into the ***products table*** :

```
INSERT INTO products (name, status, price) VALUES ( 'Product A' , 'available ' , 8 ),
( 'Product B' , 'available' , 1.5 ), ( 'Product C' , 'out of stock' , 80 );
```

We already have the database created. Now let's create a stored procedure.

How to create a stored procedure

Now that we have set up a database and a table, let's create three **stored procedures** to show the usage and differences of each parameter type.

When defining procedures, we will need to use delimiters to tell MySQL that it is a separate block. In the following examples, **DELIMITER $ $** stops MySQL execution , which will be resumed in the **DELIMITER statement** at the end.

Stored procedure with IN parameters

We will obtain the products that have a certain status. To create the procedure, run these SQL statements:

```
DELIMITER $$
CREATE PROCEDURE getProductsByState ( IN state_name VARCHAR ( 255 ))
BEGIN
  SELECT *
  FROM products
  WHERE state = state_name ;
END $$
DELIMITER
```

The state name is contained in the ***state_name parameter*** that we have defined as **IN** . Assuming you want to obtain the products with *available status* , you would have to invoke the procedure like this:

```
CALL getProductsByStatus ( 'available' )
```

This will be the result:

```
+--------+-------------+-----------+--------+
| id | name | status | price |
+--------+-------------+-----------+--------+
| 1 | Product A | available | 8.00 |
```

```
| 2 | Product B | available | 1.50 |
+---------+-------------+-----------+---------+
```

Stored procedure with OUT parameters

We will obtain the number of products according to their status. To create the procedure, run these SQL statements:

```
DELIMITER $$
CREATE PROCEDURE countProductsByState (
IN state_name VARCHAR ( 25 ),
   OUT number INT )
BEGIN
   SELECT count (id)
   INTO number
   FROM products
   WHERE state = state_name ;
END $$
DELIMITER
```

As before, we pass the state as ***state_name*** , defined as **IN** . We also define ***number*** as the **OUT parameter** . Assuming you want to get the number of products with *available status* , you should call the procedure like this:

```
CALL countProductsByStatus ( 'available' , @number );
SELECT @number AS available;
```

```
+------------+
```

To obtain the number of products out of stock, we must invoke the procedure like this:

```
CALL countProductsByStatus ( 'sold out' , @number );
SELECT @number AS out of stock;---------- +
```

Stored procedure with INOUT parameters

We are going to create a procedure that increases a variable called profit when a product is sold. To create the procedure, run these SQL statements:

```
DELIMITER $$
CREATE PROCEDURE sellProduct (
```

```
  INOUT benefit INT ( 255 ),
IN product_id INT )
BEGIN
  SELECT @ price_increment = price
  FROM products
  WHERE id = product_id ;
  SET profit = profit + @price_increment ;
END $$
DELIMITER
```

In the procedure we pass the ***benefit parameter*** as **INOUT** so that it maintains its value after the execution of the procedure. We also define the ***product_id*** parameter as **IN** to tell the procedure the product we want to sell. To simplify things we assume that the product is always available. Let's sell some products to see how the variable changes:

```
SET @profit = 0 ;
CALL sellProduct ( @profit, 1 );
CALL sellProduct ( @profit, 2 );
CALL sellProduct ( @profit, 2 );
SELECT @profit;
```

This will be the result:

How to delete a stored procedure

You can delete a stored procedure using the **DROP PROCEDURE statement** . For example, if you want to eliminate the ***sellProduct procedure*** from the previous example, you will have to execute this statement:

```
DROP PROCEDURE sellProduct ;
```

And that's all. If you want, you can consult more details about stored procedures in the official **MySQL documentation** .