What is a Stored Procedure?

DELIMITER //

CREATE PROCEDURE get\_all\_cars()

BEGIN

SELECT \* FROM cars ORDER BY make, value DESC;

END //

DELIMITER ;

CALL get\_all\_cars;

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

Stored Procedure Syntax

CREATE PROCEDURE *procedure\_name*  
AS  
*sql\_statement*  
GO;

Execute a Stored Procedure

EXEC *procedure\_name*;

Demo Database

Below is a selection from the "Customers" table in the Northwind sample database:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 |  |

Stored Procedure Example

The following SQL statement creates a stored procedure named "SelectAllCustomers" that selects all records from the "Customers" table:

Example

CREATE PROCEDURE SelectAllCustomers  
AS  
SELECT \* FROM Customers  
GO;

Execute the stored procedure above as follows:

Example

EXEC SelectAllCustomers;

Stored Procedure With One Parameter

The following SQL statement creates a stored procedure that selects Customers from a particular City from the "Customers" table:

Example

CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)  
AS  
SELECT \* FROM Customers WHERE City = @City  
GO;

Execute the stored procedure above as follows:

Example

EXEC SelectAllCustomers @City = 'London';

Stored Procedure With Multiple Parameters

Setting up multiple parameters is very easy. Just list each parameter and the data type separated by a comma as shown below.

The following SQL statement creates a stored procedure that selects Customers from a particular City with a particular PostalCode from the "Customers" table:

Example

CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode nvarchar(10)  
AS  
SELECT \* FROM Customers WHERE City = @City AND PostalCode = @PostalCode  
GO;

Execute the stored procedure above as follows:

Example

EXEC SelectAllCustomers @City = 'London', @PostalCode = 'WA1 1DP';

# **Stored Procedure Parameters: Input, Output, Optional**

Here you will learn about stored procedure parameters, optional parameters, and executing stored procedures with parameters in SQL Server.

* A stored procedure can have zero or more INPUT and OUTPUT parameters.
* A stored procedure can have a maximum of 2100 parameters specified.
* Each parameter is assigned a name, a data type, and direction like Input, Output, or Return. If a direction is not specified, then by default, it is Input.
* You can specify a default value for the parameters.
* Stored procedures can return a value to the calling program if the parameter is specified as OUTPUT.
* The parameter values must be a constant or a variable. It cannot be a function name.
* Parameter variables can be either user-defined or system variables like @spid

## Stored Procedure with Input Parameters

Consider the following stored procedure example with the input parameters.

Example: Stored Procedure with INPUT Parameters

 Copy

CREATE PROCEDURE uspUpdateEmpSalary

(

@empId int

,@salary money

)

AS

BEGIN

UPDATE dbo.Employee

SET Salary = @salary

WHERE EmployeeID = @empId

END

In the above stored procedure uspUpdateEmpSalary, the @empId and @Salary are INPUT parameters. By default, all the parameters are INPUT parameters in any stored procedure unless suffix with OUTPUT keyword. @empId is of int type and @salary is of money data type. You pass the INPUT parameters while executing a stored procedure, as shown below.

Example: Passing INPUT Parameters

 Copy

EXEC dbo.uspUpdateEmpSalary @EmpId = 4, @Salary = 25000

-- or

EXEC dbo.uspUpdateEmpSalary 4, 25000

## Parameter Names

* The stored procedure parameters names must start with a single @.
* The name must be unique in the scope of the stored procedure.
* If parameter values are passed as @Param1 = value1, @ Param2 = value2 as shown in the above example, then the parameters can be passed in any order.
* If one parameter is supplied as @param1 = value, then all parameters must be supplied in the same manner.

## OUTPUT Parameters

The OUTPUT parameter is used when you want to return some value from the stored procedure. The calling program must also use the OUTPUT keyword while executing the procedure.

The following stored procedure contains INPUT and OUTPUT parameters.

Example: Stored Procedure with OUTPUT Parameter

 Copy

CREATE PROCEDURE uspGetManagerID

@empId int,

@managerId int OUTPUT

AS

BEGIN

SELECT @managerId = ManagerID

FROM dbo.Employee

WHERE EmployeeID = @empId

END

In the above uspGetManagerID stored procedure, @manageId is an OUTPUT parameter. The value will be assigned in the stored procedure and returned to the calling statement. The following pass the OUTPUT parameter while executing the stored procedure.

Example: Passing OUTPUT Parameter

 Copy

DECLARE @managerID int

EXECUTE uspGetManagerID @empId = 2, @managerId OUTPUT

PRINT @managerId

Above, the uspGetManagerID is called by passing INPUT parameter @employeeID = 2 and @managerID OUTPUT as the output parameter. Notice that we have not assigned any value to an OUTPUT variable @managerID and also specified the OUTPUT keyword.

There are a total of three methods of returning data from a stored procedure: OUTPUT parameter, result sets, and return codes.

Result sets: If the body of the stored procedure has a SELECT statement, then the rows returned by the select statement are directly returned to the client.

Return code: A stored procedure can return an integer value called the Return code which will indicate the execution status of the procedure. You specify the return code using the RETURN keyword in the procedure.

## Optional Parameters

SQL Server allows you to specify the default values for parameters. It allows you to skip the parameters that have default values when calling a stored procedure.

The default value is used when no value is passed to the parameter or when the DEFAULT keyword is specified as the value in the procedure call.

Specify the default value when you declare parameters, as shown below.

CREATE PROCEDURE uspUpdateEmpSalary

(

@empId int

,@salary money = 1000

)

AS

BEGIN

UPDATE dbo.Employee

SET Salary = @salary

WHERE EmployeeID = @empId

END

Above, @empsalary money = 0 declares @salary parameter and assigns the default value. Now, you can call the above procedure without passing @salary parameter, as shown below.

Example: Calling Stored Procedure

 Copy

EXEC uspUpdateEmpSalary 4

The above statement will update the Salary column with the default value 1000 for the EmployeeID 4. Thus, making @salary parameter as optional.

The CallableStatement of JDBC API is used to call a stored procedure. A Callable statement can have output parameters, input parameters, or both. The prepareCall() method of connection interface will be used to create CallableStatement object.

Following are the steps to use Callable Statement in Java to call Stored Procedure:

**1) Load MySQL driver and Create a database connection.**

*import java.sql.\*;*

*public class JavaApplication1 {*

*public static void main(String[] args) throws Exception*

*{*

*Class.forName(“com.mysql.jdbc.Driver”);*

*Connection con=DriverManager.getConnection(“jdbc:mysql://localhost/root”,”geek”,”geek”);*

*}*

*}*

**2) Create a SQL String**

We need to store the SQL query in a String.

*String sql\_string=”insert into student values(?,?,?)”;*

**3) Create CallableStatement Object**

The prepareCall() method of connection interface will be used to create CallableStatement object. The sql\_string will be passed as an argument to the prepareCall() method.

*CallableStatement cs = con.prepareCall(sql\_string);*

**4) Set The Input Parameters**

Depending upon the data type of query parameters we can set the input parameter by calling setInt() or setString() methods.

*cs.setString(1,”geek1″);*

*cs.setString(2,”python”);*

*cs.setString(3,”beginner”);*

**5) Call Stored Procedure**

Execute stored procedure by calling execute() method of CallableStatement class.

Example of using Callable Statement in Java to call Stored Procedure

* Java

|  |
| --- |
| // Java program  to use Callable Statement  // in Java to call Stored Procedure    package javaapplication1;    import java.sql.\*;    public class JavaApplication1 {        public static void main(String[] args) throws Exception      {          Class.forName("com.mysql.jdbc.Driver");            // Getting the connection          Connection con = DriverManager.getConnection("jdbc:<mysql://localhost/root>", "acm", "acm");            String sql\_string = "insert into students values(?,?,?)";            // Preparing a CallableStateement          CallableStatement cs = con.prepareCall(sql\_string);            cs.setString(1, "geek1");          cs.setString(2, "python");          cs.setString(3, "beginner");          cs.execute();          System.out.print("uploaded successfully\n");      } |