Stored Procedure in SQL Server

A stored procedure is a **group of one or more pre-compiled SQL statements** into a logical unit. It is stored as an object inside the database server. It is a subroutine or a subprogram in the common computing language that has been created and stored in the database. Each procedure in SQL Server always **contains a name, parameter lists, and Transact-SQL statements**. The SQL Database Server stores the stored procedures as **named objects**. We can invoke the procedures by using triggers, other procedures, and applications such as [Java](https://www.javatpoint.com/java-tutorial), [Python](https://www.javatpoint.com/python-tutorial), [PHP](https://www.javatpoint.com/php-tutorial), etc. It can support almost all relational database systems.

[SQL Server](https://www.javatpoint.com/sql-server-tutorial) builds an execution plan when the stored procedure is called the first time and stores them in the cache memory. The plan is reused by SQL Server in subsequent executions of the stored procedure, allowing it to run quickly and efficiently.

Features of Stored Procedures in SQL Server

The following are the features of stored procedure in SQL Server:

* **Reduced Traffic:** A stored procedure reduces network traffic between the application and the database server, resulting in increased performance. It is because instead of sending several [SQL](https://www.javatpoint.com/sql-tutorial) statements, the application only needs to send the name of the stored procedure and its parameters.
* **Stronger Security:** The procedure is always secure because it manages which processes and activities we can perform. It removes the need for permissions to be granted at the database object level and simplifies the security layers.
* **Reusable:** Stored procedures are reusable. It reduces code inconsistency, prevents unnecessary rewrites of the same code, and makes the code transparent to all applications or users.
* **Easy Maintenance:** The procedures are easier to maintain without restarting or deploying the application.
* **Improved Performance:** Stored Procedure increases the application performance. Once we create the stored procedures and compile them the first time, it creates an execution plan reused for subsequent executions. The procedure is usually processed quicker because the query processor does not have to create a new plan.

Types of Stored Procedures

SQL Server categorizes the stored procedures mainly in two types:

1. User-defined Stored Procedures
2. System Stored Procedures

User-defined Stored Procedures

**Database developers or database administrators build user-defined stored procedures**. These procedures provide one or more SQL statements for selecting, updating, or removing data from database tables. A stored procedure specified by the user accepts input parameters and returns output parameters. DDL and DML commands are used together in a user-defined procedure.

**We can further divide this procedure into two types:**

* **T-SQL Stored Procedures:** Transact-SQL procedures are one of the most popular types of SQL Server procedures. It takes parameters and returns them. These procedures handle INSERT, UPDATE, and DELETE statements with or without parameters and output row data.
* **CLR Stored Procedures:** The SQL Server procedures are a group of SQL commands, and the CLR indicates the common language runtime. CLR stored procedures are made up of the CLR and a stored procedure, which is written in a CLR-based language like VB.NET or C#. CLR procedures are .Net objects that run in the SQL Server database's memory.

System Stored Procedures

The server's administrative tasks depend primarily on system stored procedures. When SQL Server is installed, it creates system procedures. The system stored procedures prevent the administrator from querying or modifying the system and database catalog tables directly. Developers often ignore system stored procedures.

SQL Server Stored Procedures Syntax

The following are the basic syntax to create stored procedures in SQL Server:

1. **CREATE** **PROCEDURE** [schema\_name].procedure\_name
2. @parameter\_name data\_type,
3. ....
4. parameter\_name data\_type
5. **AS**
6. **BEGIN**
7. -- SQL statements
8. -- SELECT, INSERT, UPDATE, or DELETE statement
9. **END**

Parameter Explanations

The stored procedure syntax has the following parameters:

**Schema\_name:** It is the name of your database or schema. By default, a procedure is associated with the current database, but we can also create it into another database by specifying the DB name.

**Procedure\_Name:** It represents the name of your stored procedure that should be meaningful names so that you can identify them quickly. It cannot be the system reserved keywords.

**Parameter\_Name:** It represents the number of parameters. It may be zero or more based upon the user requirements. We must ensure that we used the appropriate data type. **For example**, @Name VARCHAR(50).

SET NOCOUNT ON in Stored Procedure

In some cases, we use the SET NOCOUNT ON statement in the stored procedure. This statement prevents the message that displays the number of rows affected by SQL queries from being shown. NOCOUNT denotes that the count is turned off. It means that if SET NOCOUNT ON is set, no message would appear indicating the number of rows affected.

How to execute/call a stored procedure?

We can use the **EXEC command** to call/execute stored procedures in SQL Server. The following syntax illustrates the execution of a stored procedure:

1. **EXEC** procedure\_name;
2. Or,
3. **EXECUTE** procedure\_name;

If we are using the SSMS, we need to use the below steps to execute stored procedures:

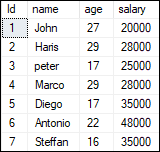
1. Navigate to the **Programmability** -> **Stored Procedures**.
2. Next, select the Stored Procedure menu and expand it. You will see the available stored procedures.
3. Right-click on the stored procedure you want to execute and choose the **Execute** **Stored** **Procedure**
4. The Execute Procedure window will appear. If the procedure has any parameters, we must assign/pass them before clicking **OK** to execute it. If no parameters are defined, click OK to run the procedure.

Stored Procedure Simple Example

We can create a stored procedure in SQL Server in two ways:

* Using T-SQL Query
* Using SQL Server Management Studio

We will take a **student table** to demonstrate the stored procedure examples. This table has the following data:



The below example uses the **CREATE PROCEDURE SQL statement** for creating a stored procedure that displays the student's list in the increasing order of a salary from the STUDENT table in the selected database:

1. **CREATE** **PROCEDURE** studentList
2. **AS**
3. **BEGIN**
4. **SELECT** **name**, age, salary
5. **FROM** STUDENT
6. **ORDER** **BY** salary;
7. **END**;

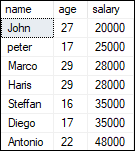
In this syntax, the **studentList** is the name of the stored procedure, and the **AS** keyword distinguishes the stored procedure's heading and body. The **BEGIN** and **END** keywords that accompany a single statement in a stored procedure are optional. However, it's a good idea to include them to make the code more understandable.

When we run this statement, and everything is correct, the following message will appear: "**Commands completed successfully**." It indicates that the stored procedure was successfully compiled and saved to the database system.

We can execute this procedure by using the below command:

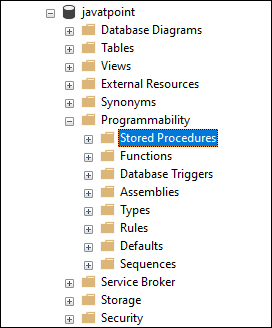
1. **EXEC** studentList;

It will return the output as follows:

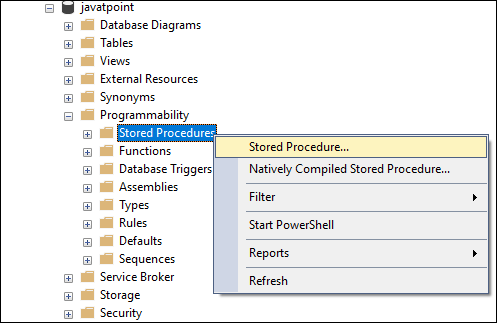


**If we are using the SSMS, use the following steps for creating the stored procedure:**

**Step 1:** Select the Database -> Programmability -> Stored Procedures.



**Step 2:** Right-click on the Stored Procedures folder to open the menu and then select the New -> Stored Procedure option as follows:



**Step 3:** When we select the New Stored Procedure option, we will get the new query window containing the default Stored Procedure Template. Here, we can add the procedure name, parameters (if any), and the SQL query we want to use.

How to rename stored procedures in SQL Server?

SQL Server does not allow us to change the name of a stored procedure. Because renaming a stored procedure does not modify the name of the corresponding object in the **sys.sql\_modules**. Therefore, if we need to change the existing stored procedure, simply DROP and recreate it with a new name.

How to modify stored procedures in SQL Server?

We need to update or modify the stored procedure over a period of time. SQL Server allows us to update or modify an existing stored procedure in two ways:

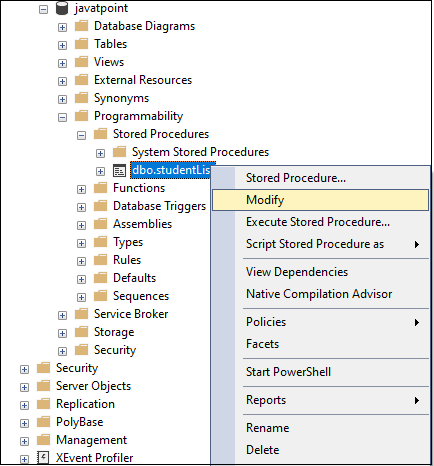
* Using T-SQL Query
* Using SQL Server Management Studio

Modify Stored Procedures using SSMS

The following steps help to learn how we can modify or make changes in stored procedures:

**Step 1:** Navigate to the Database -> Programmability -> Stored Procedures.

**Step 2:** Expand the Stored Procedures folder, right-click on the stored procedure that you want to modify, and then select the Modify option as follows:



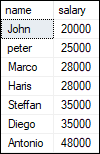
**Step 3:** Once we click the Modify option, we will get a new query window with an auto-generated **ALTER PROCEDURE** code. Here we can make changes as per our needs.

Modify Stored Procedures using T-SQL Query

SQL Server provides an ALTER PROCEDURE statement to make modifications in the existing stored procedures. If we want to modify the above created stored procedure, we can write the ALTER PROCEDURE statement as follows:

1. **ALTER** **PROCEDURE** [dbo].[studentList]
2. **AS**
3. **BEGIN**
4. **SET** NOCOUNT **ON**;
5. **SELECT** **name**, salary
6. **FROM** STUDENT
7. **ORDER** **BY** salary;
8. **END**;

Let's run the procedure to check whether we have successfully updated the **studentList** procedure or not. Using the EXECUTE statement, we will get the below output where we can see that our stored procedure is successfully modified.



How to list all stored procedures in SQL Server?

When we have several procedures, it is very important to list all procedures. Because sometimes the procedure names are the same in many databases. In that case, this query is very useful. We can list all stored procedure in the current database as follows:

1. **SELECT** \* **FROM** sys.procedures;

The best way for listing all user-defined stored procedures in a database is to use the **ROUTINES** information schema view as below:

1. **SELECT** ROUTINE\_SCHEMA, ROUTINE\_NAME
2. **FROM** INFORMATION\_SCHEMA.ROUTINES
3. **WHERE** ROUTINE\_TYPE = 'PROCEDURE';

OR,

1. **SELECT** \*
2. **FROM** db\_name.INFORMATION\_SCHEMA.ROUTINES
3. **WHERE** ROUTINE\_TYPE = 'PROCEDURE'

Another way to return a list of stored procedures is to query the **sys.objects** system catalog view.

1. **SELECT**
2. SCHEMA\_NAME(schema\_id) **AS** [**Schema**],
3. **name**
4. **FROM** sys.objects
5. **WHERE** type = 'P';

How to delete/drop stored procedures in SQL Server?

We can delete the stored procedure in SQL Server permanently. SQL Server removes the stored procedure in two ways:

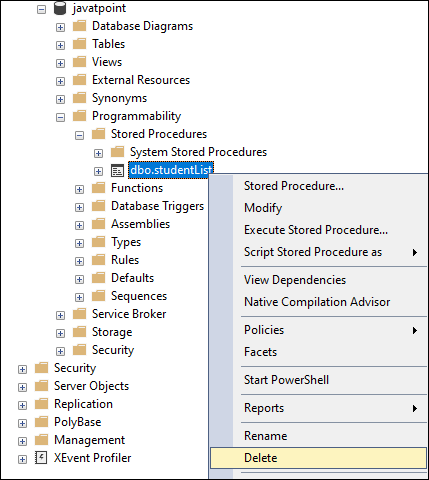
* Using T-SQL Query
* Using SQL Server Management Studio

DROP Stored Procedures using SSMS

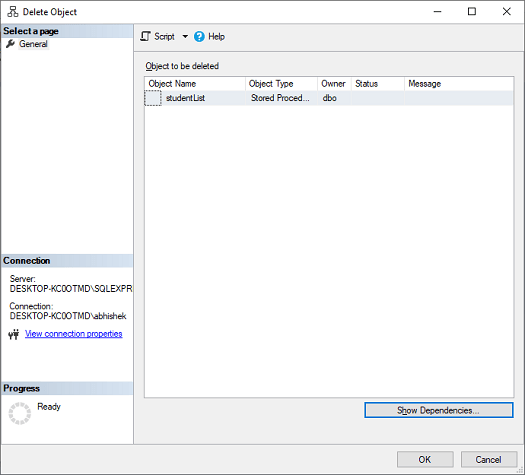
The following steps help to learn how we can delete stored procedures:

**Step 1:** Go to the Database -> Programmability -> Stored Procedures.

**Step 2:** Expand the Stored Procedures folder, right-click on the stored procedure that you want to remove, and then select the Delete option as follows:



**Step 3:** Once we click the Delete option, we will get a Delete Object window. We can check the dependencies by clicking on the Show Dependencies button and then click OK to remove the stored procedure.



Delete Stored Procedures using T-SQL Query

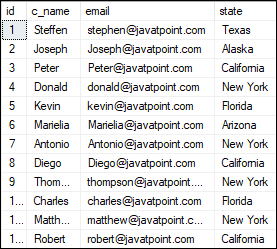
SQL Server provides a DROP PROCEDURE statement to remove the existing stored procedures. We can write the DROP PROCEDURE statement as follows:

1. IF OBJECT\_ID ('procedure\_name', 'P') **IS** NOT NULL
2. **DROP** **PROCEDURE** procedure\_name;

NOTE: It's a good idea to use IF OBJECT ID ('procedure name', 'P') IS NOT NULL to see if the stored procedure exists in the database.

Input Parameters in Stored Procedure

SQL Server allows us to create input parameters stored procedures. This type of stored procedure can enable us to pass one or more parameters to get the filtered result. Let us understand it with the help of an example. Consider the following '**customer'** table:



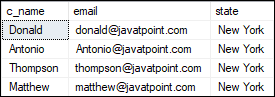
The below statement creates a stored procedure with an input parameter:

1. **CREATE** **PROCEDURE** getEmployeeDetails (@States **VARCHAR**(50))
2. **AS**
3. **BEGIN**
4. **SET** NOCOUNT **ON**;
5. **SELECT** c\_name, email, state
6. **FROM** customer
7. **WHERE** state = @States
8. **END**

If we want to execute this stored procedure, we need to pass the value for the **@States** parameter. We can pass the parameter value in any of the following ways:

1. **EXEC** [dbo].[getEmployeeDetails] 'California';
3. --OR we can write
5. **EXEC** [dbo].[getEmployeeDetails] @States = 'New York';

We will get the output as below:

Output Parameters in Stored Procedure

SQL Server enables us to provide many output parameters in a stored procedure. These output parameters can be of any valid data type, such as integer, date, or character. We can use the below syntax to create an output parameter in the stored procedure:

1. parameter\_name data\_type **OUTPUT**

Let us understand how to use Output Parameters in a stored procedure with the help of an example. The following statement will create a procedure called **countStudent**, in which we will declare an integer type variable called **@StudentCount** and use the **OUTPUT** keyword. The procedure uses the **COUNT** function to find the number of students in the STUDENT table, and then the value is assigned to the output parameter.

1. **CREATE** **PROCEDURE** countStudent (@StudentCount **INT** **OUTPUT**)
2. **AS**
3. **BEGIN**
4. **SELECT** @StudentCount = COUNT(Id)**FROM** STUDENT;
5. **END**;

Now, we will execute the stored procedure. Here, we need to pass the output parameter **@StudentaCouns** as follows:

1. -- Declare an Int Variable that corresponds to the Output parameter in SP
2. **DECLARE** @TotalStudents **INT**
4. -- Don't forget to use the keyword OUTPUT
5. **EXEC**  [dbo].[countStudent] @TotalStudents **OUTPUT**
7. -- Print the result
8. PRINT @TotalStudents

We will get the following output:

Stored Procedure in SQL Server

Temporary Stored Procedure

We can create temporary procedures in the same way as we can create temporary tables. The tempdb database is used to create these procedures. We can divide the temporary procedures into two types:

* Local Temporary Stored Procedures
* Global Temporary Procedures.

**Local Temporary Stored Procedures:** We can create this type of procedure by using the # as prefix and accessed only in the session in which they were created. When the connection is closed, this process is immediately terminated.

Here's an example of how to create a local temporary procedure:

1. **CREATE** **PROCEDURE** #**Temp**
2. **AS**
3. **BEGIN**
4. PRINT 'Local temp procedure'
5. **END**

**Global Temporary Stored Procedure:** We can create this type of procedure by using the ## as a prefix and accessed from any sessions. When the connection that was used to create the procedure is closed, this procedure is automatically terminated.

Here's an example of how to create a global temporary procedure:

1. **CREATE** **PROCEDURE** ##**TEMP**
2. **AS**
3. **BEGIN**
4. PRINT 'Global temp procedure'
5. **END**

Disadvantages of Stored Procedures

The following are the limitations of stored procedures in SQL Server:

**Debugging:** Since debugging stored procedures is never simple, it is not advised to write and execute complex business logic using them. As a result, if we will not handle it properly, it can result in a failure.

**Dependency:** As we know, professional DBAs and database developers handle vast data sets in large organizations. And the application developers must depend on them because any minor changes must be referred to a DBA, who can fix bugs in existing procedures or build new ones.

**Expensive:** Stored procedures are costly to manage in terms of DBAs because organizations would have to pay extra costs for specialist DBAs. A DBA is much more qualified to handle complex database procedures.

**Specific to a Vendor:** Stored procedures written in one platform cannot run on another. Since procedures written in Oracle are more complicated, we will need to rewrite the entire procedure for SQL Server.