**Scalar Valued Function in SQL Server with Examples**

##### **What is a function in SQL Server?**

A function in SQL Server is a subprogram that is used to perform an action such as complex calculation and returns the result of the action as a value. There are two types of functions in SQL Server, such as

1. **System Defined Function**
2. **User-Defined Function**

The functions which are already defined by the system and ready to be used by the developer are called system-defined functions whereas if the function is defined by the user or developer then such types of functions are called the user-defined function.

Some functions take parameters; do some processing and returning some results back. For example **SELECT SQUARE(3)**  
Some functions may not take any parameters, but returns some result, for example, **SELECT GETDATE()**  
So we can say that a function can have a parameter that is optional but a function should always return a value that is mandatory.

##### **Types of User-Defined Function:**

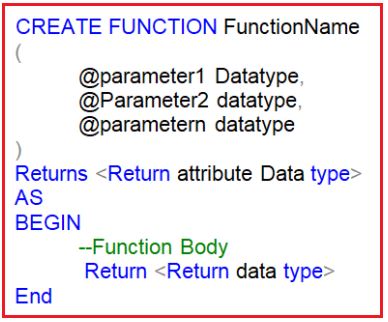
In SQL Server, we can create three types of User-Defined Functions, such as

1. **Scalar Valued Functions**
2. [**Inline Table-Valued Functions**](https://dotnettutorials.net/lesson/inline-table-valued-function-in-sql-server/)
3. [**Multi-Statement Table-Valued Functions**](https://dotnettutorials.net/lesson/multi-statement-table-valued-function-in-sql-server/)

In this article, we are going to discuss SQL Server Scalar Valued Function in detail with Examples, the rest two user-defined functions are going to be discussed in our next article.

##### **SQL Server Scalar Valued Functions**

The user-defined function which returns only a single value (scalar value) is known as the Scalar Valued Function. Scalar Value Functions in SQL Server may or may not have parameters that are optional but always return a single (scalar) value which is mandatory. The returned value which is return by the SQL Server Scalar Function can be of any data type, except text, ntext, image, cursor, and timestamp. Following is the syntax to create a User-Defined Scalar Value Function in SQL Server.



As you can see in the above image, we can use Create Function followed by the function name Statement to create a user-defined function in SQL Server. Then we need to specify the input parameters. It is optional. Then we need to specify the return value data type i.e. going to be returned from the function body by using the Returns followed by the return data type statement. This is mandatory. Then we need to write the function body in between the AS BEGIN and the AND block. From the function body, at some point, we need to write the Return statement followed by the return value. This is mandatory.

##### **Syntax for calling a Function in SQL Server:**

**SELECTdbo.<FunctionName>(Value)**  
When calling a scalar user-defined function we must specify the two-part name i.e. owner name and function name. The **dbo** stands for the database owner name. We can also invoke a scalar function in SQL Server using the complete three-part name i.e. database name. Owner name and function name.

##### **Examples: SQL Server Scalar Valued User-Defined Functions**

Let us understand the SQL Server Scalar User-Defined Function with some Examples.

###### **Example1: Create a Scalar Function in SQL Server which will return the cube of a given value.**

The function will an integer input parameter and then calculate the cube of that integer value and then returns the result. Here, the result produced will be in Integer. So, while creating the function, we need to specify the Returns data type as INT. The following function exactly does the same.

CREATE FUNCTION SVF1**(**@X INT**)**

RETURNS INT

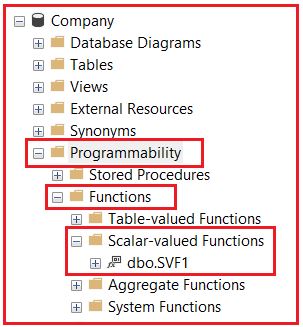
AS

BEGIN

RETURN @X \* @X \*@X

END

Once you created the function, as it is a database object, so it will be physically created inside the database with the following folder location. As it is a Scalar function, so it is created inside the Scalar-Valued Function folder as shown in the below image.



To execute the above function call it like below  
**SELECT dbo.SVF1(5)**  
**Output: 125**

###### **Example2: Write a Scalar Function to get the date difference.**

Let us now create a function that will calculate and returns the age of an employee. To compute the age we require the date of birth. So, let’s pass the date of birth as a parameter. So the user-defined CalculateAge() function returns an integer and accepts date parameter.

CREATE FUNCTION CalculateAge

**(**

@DOB DATE

**)**

RETURNS INT

AS

BEGIN

DECLARE @AGE INT

SET @AGE = DATEDIFF**(**YEAR, @DOB, GETDATE**())**-

CASE

WHEN **(**MONTH**(**@DOB**)** **>** MONTH**(**GETDATE**()))** OR

**(**MONTH**(**@DOB**)** = MONTH**(**GETDATE**())** AND

DAY**(**@DOB**)** **>** DAY**(**GETDATE**()))**

THEN 1

ELSE 0

END

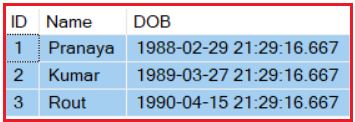
RETURN @AGE

END

**Calling the above function:**  
**SELECT dbo.CalculateAge (’02/29/1988′)**  
**SELECT dbo.CalculateAge (’02/29/1988′) AS AGE**

##### **Scalar Valued Function in Select Clause:**

The User Defined Scalar Valued Function can also be used in the select clause of an SQL Query in SQL Server. To understand the above points we are going to use the following Employee table.



Please use the following SQL Script to create and populate the Employee table with the required sample data.

-- Create Table Employee

CREATE TABLE Employee

**(**

ID INT PRIMARY KEY,

Name VARCHAR**(**50**)**,

DOB DATETIME

**)**

GO

-- Populate Employee table with some test data

INSERT INTO Employee**(**ID, Name, DOB**)** VALUES**(**1, 'Pranaya', '1988-02-29 21:29:16.667'**)**

INSERT INTO Employee**(**ID, Name, DOB**)** VALUES**(**2, 'Kumar', '1989-03-27 21:29:16.667'**)**

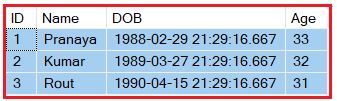
INSERT INTO Employee**(**ID, Name, DOB**)** VALUES**(**3, 'Rout', '1990-04-15 21:29:16.667'**)**

Let’s see how to use the SQL Server Scalar Function i.e. CalculateAge function in the Select clause of an SQL Query. As you can see in the below query, in the select clause we call the CalculateAge function by passing the DOB as an input parameter. The CalculateAge function takes the DOB value and returns the Age.

SELECT ID, Name, DOB, dbo.CalculateAge**(**DOB**)** AS Age

FROM Employee

Once you execute the above query, you will get the following output.



##### **SQL Server Scalar Valued Function in Where Clause:**

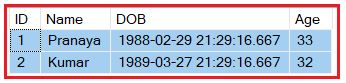
Let’s see how to use the user-defined Scalar Valued function (CalculateAge) in the where clause of an SQL Query. Let’s find out the details of those employees whose age is greater than 31. At the current date, we have two employees whose age is greater than 31.

SELECT ID, Name, DOB, dbo.calculateAge**(**DOB**)** AS Age

FROM Employee

WHERE dbo.calculateAge**(**DOB**)** **>** 31

Once you execute the above query, you will get the following output.



If you want to alter the function then you need to use **ALTER FUNCTION FuncationName** statement and to delete the function, you need to use **DROP FUNCTION FuncationName**. To view the text of the user-defined function you need to use **sp\_helptext FunctionName**

A stored procedure can also accept the DOB of an employee and return the age but we cannot use a stored procedure in a select clause or where clause. This is one of the differences between a function and a stored procedure. In our upcoming articles, we will discuss more differences between them.

###### **Example:**

CREATE PROCEDURE spCalcualateAge**(**@DOB DATE**)**

AS

BEGIN

DECLARE @AGE INT

SET @AGE =DATEDIFF**(**YEAR, @DOB,GETDATE**())**-

CASE

WHEN **(**MONTH**(**@DOB**)>**MONTH**(**GETDATE**()))**OR

**(**MONTH**(**@DOB**)**=MONTH**(**GETDATE**())**AND

DAY**(**@DOB**)>**DAY**(**GETDATE**()))**

THEN 1

ELSE 0

END

SELECT @AGE

END

--Execute

Execute spCalcualateAge '02/29/1988'

## ****Inline Table-Valued Function in SQL Server with Examples****

##### **What is a Table-Valued Function in SQL Server?**

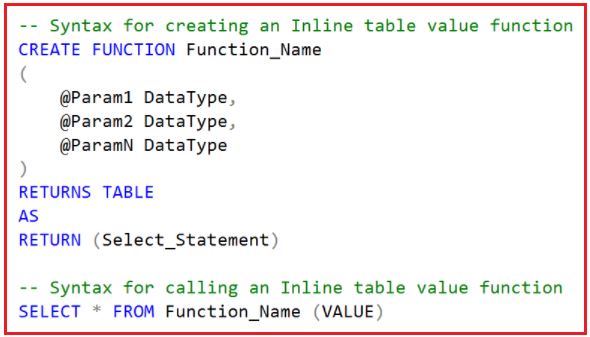
In the case of a Table-Valued Function, we can return a table as an output from the function. These are again of two types.

1. [**Inline Table-valued Function**](https://dotnettutorials.net/lesson/inline-table-valued-function-in-sql-server/)
2. [**Multi-statement table value function**](https://dotnettutorials.net/lesson/multi-statement-table-valued-function-in-sql-server/)

**Note:**In this article, we are going to discuss Inline Table-Valued Function and in our next article, we will discuss Multi-Statement Table-valued Functions with Examples.

##### **What are Inline Table-Valued functions in SQL Server?**

In the case of an Inline Table-Valued Function, the body of the function will have only a **Single Select Statement** prepared with the “**RETURN**” statement. And here, we need to specify the Return Type as TABLE by using the **RETURNS TABLE** statement. The following image shows the syntax of the Inline Table-Valued Function and how to call an Inline Table-Valued Function in SQL Server.

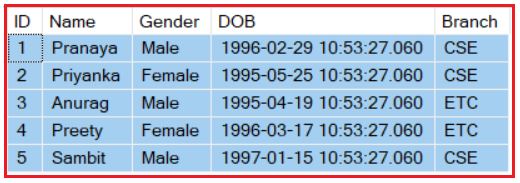


##### **Points to Remember:**

1. We specify **TABLE** as the Return Type instead of any scalar data type.
2. The function body is not closed between **BEGIN** and **END** blocks. This is because the function is going to return a single select statement.
3. The structure of the Table that is going to be returned is determined by the select statement used in the function.

###### **Example: Inline Table-Valued Function in SQL Server**

Let us understand the Inline Table-Valued Function in SQL Server with some examples. We are going to use the following Student table to understand this concept.



Please use the following SQL Script to create and populate the Student table with the required sample data which we are going to use.

-- Create Student Table

###### **Example:**

**Create a function that accepts student id as input and returns that student details from the table.**

CREATE FUNCTION FN\_GetStudentDetailsByID

**(**

@ID INT

**)**

RETURNS TABLE

AS

RETURN **(**SELECT \* FROM Student WHERE ID = @ID**)**

Once you create the above function, then call it like below.  
**SELECT \* FROM FN\_GetStudentDetailsByID(2)**  
Once you execute the above SQL Statement, it will give you the following output.

Inline Table Valued Function in SQL Server

##### **Example:**

Create a function to accept branch name as input and returns the list of students who belongs to that branch.

CREATE FUNCTION FN\_GetStudentDetailsByBranch

**(**

@Branch VARCHAR**(**50**)**

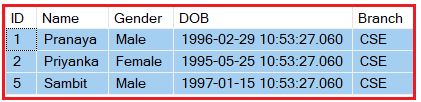
**)**

RETURNS TABLE

AS

RETURN **(**SELECT \* FROM Student WHERE Branch = @Branch**)**

Once you create the above function, then call it by executing the below statement.  
**SELECT \* FROM FN\_GetStudentDetailsByBranch(‘CSE’)**  
Once you execute the above SQL Statement, it will give you the following output.



**Note:** As the inline table-valued user-defined function, is returning a table, issue the select statement against the function, as if we are selecting the data from a TABLE.

##### **Example:**

Create a function that returns student Name, DOB, and Branch by GENDER.

CREATE FUNCTION FN\_GetStudentDetailsByGender

**(**

@Gender VARCHAR**(**50**)**

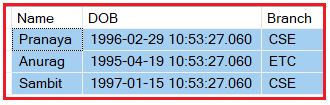
**)**

RETURNS TABLE

AS

RETURN **(**SELECT Name, DOB, Branch FROM Student WHERE Gender = @Gender**)**

Once you create the above function, then call it by executing the below statement.  
**SELECT \* FROM FN\_GetStudentDetailsByGender(‘Male’)**  
Once you execute the above SQL Statement, it will give you the following output.

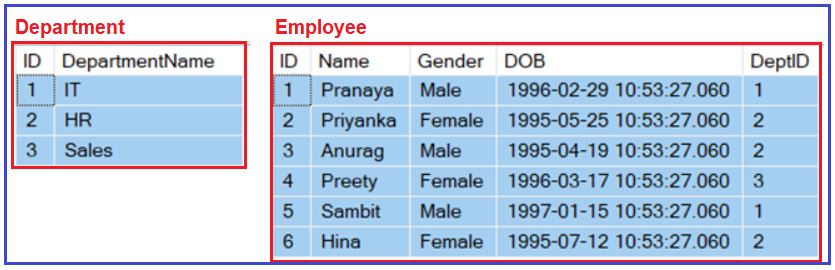


##### **Where can we use Inline Table-valued Functions in SQL Server?**

The Inline Table-Valued function in SQL Server can be used to achieve the functionalities of parameterized views, and the table returned by the inline table-valued function in SQL Server can also be used in joins with other tables.

##### **Inline Table-Valued Function with JOINs in SQL Server**

Let us understand how to use Inline Table-Valued Function with Joins with an example. We are going to use the following Department and Employee tables to understand this concept.



Please use the following SQL Script to create and populate the Department and Employee tables with sample data.

-- Create Department Table

CREATE TABLE Department

**(**

ID INT PRIMARY KEY,

DepartmentName VARCHAR**(**50**)**

**)**

GO

-- Populate the Department Table with test data

INSERT INTO Department VALUES**(**1, 'IT'**)**

INSERT INTO Department VALUES**(**2, 'HR'**)**

INSERT INTO Department VALUES**(**3, 'Sales'**)**

GO

-- Create Employee Table

CREATE TABLE Employee

**(**

ID INT PRIMARY KEY,

Name VARCHAR**(**50**)**,

Gender VARCHAR**(**50**)**,

DOB DATETIME,

DeptID INT FOREIGN KEY REFERENCES Department**(**ID**)**

**)**

GO

-- Populate the Employee Table with test data

INSERT INTO Employee VALUES**(**1, 'Pranaya', 'Male','1996-02-29 10:53:27.060', 1**)**

INSERT INTO Employee VALUES**(**2, 'Priyanka', 'Female','1995-05-25 10:53:27.060', 2**)**

INSERT INTO Employee VALUES**(**3, 'Anurag', 'Male','1995-04-19 10:53:27.060', 2**)**

INSERT INTO Employee VALUES**(**4, 'Preety', 'Female','1996-03-17 10:53:27.060', 3**)**

INSERT INTO Employee VALUES**(**5, 'Sambit', 'Male','1997-01-15 10:53:27.060', 1**)**

INSERT INTO Employee VALUES**(**6, 'Hina', 'Female','1995-07-12 10:53:27.060', 2**)**

GO

**Let’s first create an Inline Table-Valued Function that returns the Employees by Gender from the Employees table.**

CREATE FUNCTION FN\_GetEmployeessByGender

**(**

@Gender VARCHAR**(**50**)**

**)**

RETURNS TABLE

AS

RETURN **(**SELECT ID, Name, Gender, DOB, DeptID FROM Employee WHERE Gender = @Gender**)**

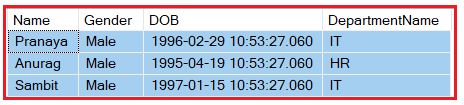
**Now, let’s join the Employees returned by the inline table-valued function with the Departments table as shown below**

SELECT Name, Gender, DOB, DepartmentName

FROM FN\_GetEmployeessByGender**(**'Male'**)** Emp

JOIN Department Dept **on** Dept.ID = Emp.DeptID

**When we execute the above query, it will give us the following output.**



##### **Example: Table-valued Function Returning data From two Tables using Join in SQL Server**

In the below example we are joining the Employee and Department table and returning data from the function body as a single SQL Statement. This is very similar to Views but here we are taking an input parameter and based on the parameter we are returning the result. This is not possible (input parameter) in the case of SQL Server Views.

**CREATE** **FUNCTION** FN\_EmployeessByGender

(

@Gender **VARCHAR**(50)

)

**RETURNS** **TABLE**

**AS**

**RETURN** (

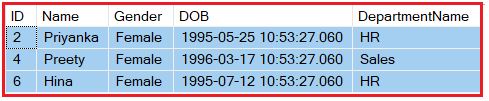
**SELECT** Emp.**ID**, Name, Gender, **DOB**, DepartmentName

**FROM** Employee Emp

**JOIN** Department Dept **on** Emp.DeptId = Dept.Id

**WHERE** Gender = @Gender)

Once you create the above function, then call it by executing the below statement.  
**SELECT \* FROM dbo.FN\_EmployeessByGender(‘Female’);**  
Once you execute the above SQL Statement, it will give you the following output.



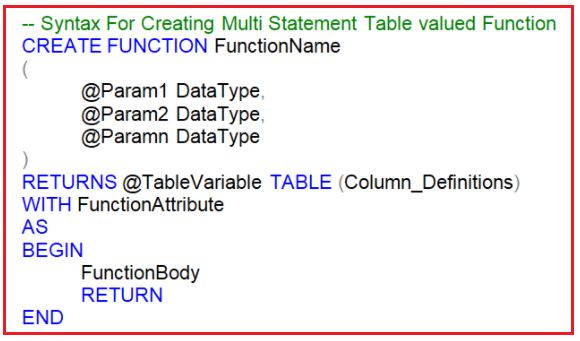
## ****Multi-Statement Table Valued Function in SQL Server****

##### **Multi-Statement Table-Valued Function in SQL Server**

The Multi-Statement Table Valued Function in SQL Server is the same as the Inline Table-Valued Function means it is also going to returns a table as an output but with the following differences.

1. The Multi-Statement Table-Valued Function body can contain more than one statement. In Inline Table-Valued Function, it contains only a single Select statement prepared by the return statement.
2. In Multi-Statement Table-Valued Function, the structure of the table returned from the function is defined by us. But, in Inline Table-Valued Function, the structure of the table is defined by the Select statement that is going to return from the function body.

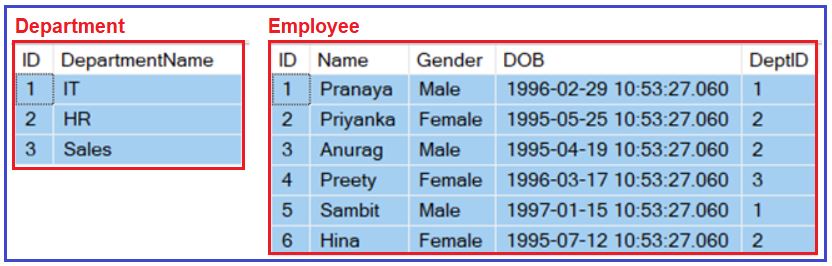
The following image shows the syntax of the Multi-Statement Table-Valued Function in SQL Server.



**Note:** In the case of Multi-Statement Table Valued Function in SQL Server, we need to define the structure of the table being return.

##### **Example: Multi-Statement Table-Valued Function in SQL Server**

Let us understand Multi-Statement Table-Valued Function comparing with the Inline Table-Valued Function in SQL Server with an example. We are going to use the following Department and Employee tables.



**Please use the below SQL Script to create and populate the Department and Employee tables with sample data.**

-- Create Department Table

CREATE TABLE Department

**(**

ID INT PRIMARY KEY,

DepartmentName VARCHAR**(**50**)**

**)**

GO

-- Populate the Department Table with test data

INSERT INTO Department VALUES**(**1, 'IT'**)**

INSERT INTO Department VALUES**(**2, 'HR'**)**

INSERT INTO Department VALUES**(**3, 'Sales'**)**

GO

-- Create Employee Table

CREATE TABLE Employee

**(**

ID INT PRIMARY KEY,

Name VARCHAR**(**50**)**,

Gender VARCHAR**(**50**)**,

DOB DATETIME,

DeptID INT FOREIGN KEY REFERENCES Department**(**ID**)**

**)**

GO

-- Populate the Employee Table with test data

INSERT INTO Employee VALUES**(**1, 'Pranaya', 'Male','1996-02-29 10:53:27.060', 1**)**

INSERT INTO Employee VALUES**(**2, 'Priyanka', 'Female','1995-05-25 10:53:27.060', 2**)**

INSERT INTO Employee VALUES**(**3, 'Anurag', 'Male','1995-04-19 10:53:27.060', 2**)**

INSERT INTO Employee VALUES**(**4, 'Preety', 'Female','1996-03-17 10:53:27.060', 3**)**

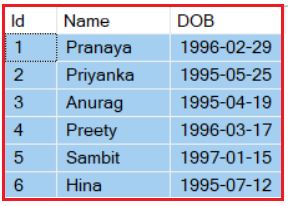
INSERT INTO Employee VALUES**(**5, 'Sambit', 'Male','1997-01-15 10:53:27.060', 1**)**

INSERT INTO Employee VALUES**(**6, 'Hina', 'Female','1995-07-12 10:53:27.060', 2**)**

GO

##### **Example:**

**Let’s write both Inline and Multi-Statement Table-Valued functions in SQL Server that return the following output.**



###### **Using Inline Table-Valued function**

-- Inline Table Valued function:

**CREATE** **FUNCTION** ILTVF\_GetEmployees()

**RETURNS** **TABLE**

**AS**

**RETURN** (**SELECT** **ID**, Name, Cast(**DOB** **AS** Date) **AS** **DOB**

**FROM** Employee)

Calling the Inline Table-Valued Function: **SELECT \* FROM ILTVF\_GetEmployees()**

###### **Using Multi-Statement Table-Valued function**

-- Multi-statement Table Valued function:

**CREATE** **FUNCTION** MSTVF\_GetEmployees()

**RETURNS** @**Table** **Table** (**ID** int, Name nvarchar(20), **DOB** Date)

**AS**

**BEGIN**

**INSERT** **INTO** @**Table**

**SELECT** **ID**, Name, Cast(**DOB** **AS** Date)

**FROM** Employee

Return

End

Calling the Multi-statement Table Valued Function: **SELECT \* FROM MSTVF\_GetEmployees()**

##### **What are the differences between Inline and Multi-Statement Table-Valued Functions in SQL Server?**

1. In an**Inline Table-Valued Function**, the returns clause cannot define the structure of the table that the function is going to return whereas in the **Multi-Statement Table-Valued Function** the returns clause defines the structure of the table that the function is going to return.
2. The **Inline Table-Valued Function** cannot have **BEGIN and END** blocks whereas the **Multi-Statement Table-Valued Function** has the **Begin and End** blocks.
3. It is possible to update the underlying database table using the inline table-valued function but it is not possible to update the underlying database table using the multi-statement table-valued function.
4. Inline Table-Valued functions are better for performance than the Multi-Statement Table-Valued function. So, if the given task can be achieved using an Inline Table-Valued Function, then it is always preferred to use Inline Table-valued Function over the Multi-Statement Table-Valued function.

**Reason For Better Performance:** Internally SQL Server treats an Inline Table-Valued function much like a view and treats a Multi-Statement Table-Valued function as a stored procedure.

##### **Example: Update underlying database table using the inline table-valued function in SQL Server**

**SELECT \* FROM dbo.ILTVF\_GetEmployees()**  
For the above function, Employee is the underlying database table.  
**UPDATE ILTVF\_GetEmployees() SET Name=’Pranaya1′ WHERE ID= 1**

The above update query will change the name **Pranaya** to **Pranaya1**, in the underlying table **Employee**. When we try to do the same thing with the multi-statement table-valued function, we will get an error stating **‘Object ‘MSTVF\_GetEmployees’ cannot be modified.**‘ The reason is that the multi-statement table-valued function did not get the data directly from the underlying database table instead it gets the data from the table variable.

Note: In Inline Table-Valued functions, we get the data directly from the underlying base table(s), and in the case of the Multi-Statement Table-Valued function, it gets the data from the table variable.

##### **What is the Difference Between Functions and Procedures in SQL Server?**

1. A function must return a value, it is mandatory whereas a procedure returning a value is optional.
2. The procedure can have parameters of both input and output whereas a function can have only input parameters.
3. In a procedure, we can perform Select. Update, Insert and Delete operations whereas function can only be used to perform select operations. It cannot be used to perform Insert, Update, and Delete operations that can change the state of the database.
4. A procedure provides the options to perform Transaction Management, Error Handling, etc whereas these operations are not possible in a function.
5. We call a procedure using **EXECUTE/ EXEC** command whereas a function is called by using **SELECT** command only.
6. From a procedure, we can call another procedure or a function whereas from a function we can call another function but not a procedure.
7. User-Defined Functions can be used in the SQL statements anywhere in the **WHERE/HAVING/SELECT** section where as Stored procedures cannot be.