

1. Stored Procedure

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;
```

Stored Procedure With One Parameter

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

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

Execute the stored procedure above as follows:

```
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:

```
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:

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

2. User-defined Functions

A user defined function is a database object that is used to save a lengthy or complicated query so that it can be ran repeatedly using simple commands. A function can return a single value back to the caller, or it can return a result set. Through the use of parameters, functions can also be made dynamic.

Scalar Valued Function

Scalar Valued Function is a function that returns a single value back to the caller. A Scalar Valued Function can be used anywhere a single value is expected. You can pass in one or more parameters to a Scalar Valued Function and do work based on those parameters.

Here is the general layout of a Scalar Valued Function:

```
CREATE FUNCTION functionName(<optional parameter list>)  
RETURNS <data type of return value>  
AS  
BEGIN  
<function body>  
RETURN <value to return>  
END
```

And here is that example again of a Scalar Valued Function definition:

```
CREATE FUNCTION getProdSum(@prodID as INT)  
RETURNS DECIMAL(5,2)  
AS  
BEGIN  
    declare @sumProd DECIMAL(5,2)  
    SELECT @sumProd = SUM(P.Price)  
    FROM Products P  
    INNER JOIN SalesOrderInfo S  
    ON P.ProdID = S.ProdID  
    WHERE P.ProdID = @prodID  
  
    RETURN @sumProd  
END
```

Here is how you would call that function:

SELECT dbo.getProdSum(1) as 'Prod 1 Sum'	
Results Messages	
Prod 1 Sum	
594.00	

Inline Table Valued Function

An Inline Table Valued Function returns a result set to the caller by way of an inner SELECT statement that is part of the function definition. You can query an Inline Table Valued function similar to how you would query a table. They accept input parameters, which allow them to return a different result set depending on the parameters passed in.

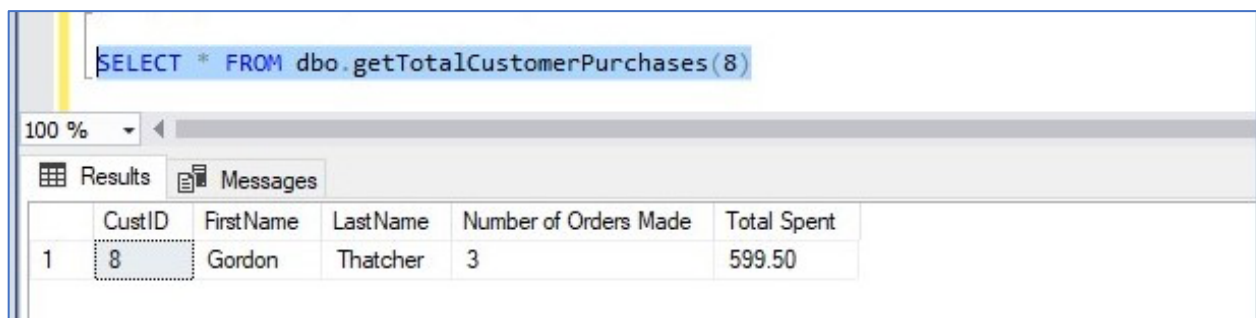
The general layout of an Inline Table Valued Function is like this:

```
CREATE FUNCTION <schema>.function_name(<optional_parameters>)  
RETURNS TABLE  
AS  
RETURN  
<SELECT statement>
```

Here is an example of an Inline Table Valued Function Definition:

```
CREATE FUNCTION getTotalCustomerPurchases(@customerID INT)  
RETURNS TABLE  
AS  
RETURN  
select C.CustID, C.FirstName, C.LastName, COUNT(*) AS 'Number of Orders Made',  
SUM(P.Price) as 'Total Spent'  
FROM Orders as O  
INNER JOIN Customers as C  
on O.CustID = C.CustID  
INNER JOIN Products as P  
on O.ProdID = P.ProdID  
Where C.CustID = @customerID  
group by C.CustID, C.FirstName, C.LastName
```

Here is how you would call that function:



The screenshot shows a SQL query window with the following query:

```
SELECT * FROM dbo.getTotalCustomerPurchases(8)
```

Below the query window, the 'Results' tab is active, displaying a table with the following data:

	CustID	FirstName	LastName	Number of Orders Made	Total Spent
1	8	Gordon	Thatcher	3	599.50

3. Basic Differences between Stored Procedure and Function in SQL Server

- Stored procedures are compiled once and stored in executable form, so procedure calls are quick and efficient. A function is compiled and executed every time whenever it is called.
- The function must return a value but in Stored Procedure it is optional. A procedure can return zero or n values with 'output variable'.
- Functions can have only input parameters for it whereas Procedures can have input or output parameters.
- Functions can be called from Procedure whereas Procedures cannot be called from a Function.
- The procedure allows SELECT as well as DML(INSERT/UPDATE/DELETE) statement in it whereas Function allows only SELECT statement in it.
- Procedures cannot be utilized in a SELECT statement whereas Function can be embedded in a SELECT statement.
- Stored Procedures cannot be used in the SQL statements anywhere in the WHERE/HAVING/SELECT section whereas Function can be.
- Functions that return tables can be treated as another rowset. This can be used in JOINS with other tables.
- An exception can be handled by try-catch block in a Procedure whereas try-catch block cannot be used in a Function.