SQL Functions

.NET

SQL functions help simplify code. There may be a complex calculation that appears in many queries. A SQL function can be created that encapsulates the formula and uses it in each query.

https://www.sqlservertutorial.net/sql-server-user-defined-functions/sql-server-scalar-functions/

Function vs. Stored Procedure

https://www.c-sharpcorner.com/UploadFile/996353/difference-between-stored-procedure-and-user-defined-function/

Differences between Stored Procedure and User Defined Function in SQL Server

User Defined Function	Stored Procedure
Function must return a value.	Stored Procedure may or not return values.
Will allow only Select statements, it will not allow us to use DML statements.	Can have select statements as well as DML statements such as insert, update, delete and so on
It will allow only input parameters, doesn't support output parameters.	It can have both input and output parameters.
It will not allow us to use try-catch blocks.	For exception handling we can use try catch blocks.
Transactions are not allowed within functions.	Can use transactions within Stored Procedures.
We can use only table variables, it will not allow using temporary tables.	Can use both table variables as well as temporary table in it.
Stored Procedures can't be called from a function.	Stored Procedures can call functions.
Functions can be called from a select statement.	Procedures can't be called from Select/Where/Having and so on statements. Execute/Exec statement can be used to call/execute Stored Procedure.
A UDF can be used in join clause as a result set.	Procedures can't be used in Join clause

SQL Scalar Function

https://docs.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql?view=sql-server-ver15
https://docs.microsoft.com/en-us/sql/relational-databases/user-defined-functions/create-user-defined-functions-database-engine?
https://www.sqlservertutorial.net/sql-server-user-defined-functions/sql-server-scalar-functions/

A ‘user-defined’ function:

- accepts parameters,
- performs an action such as a complex calculation, and
- returns the result of that action as a ***scalar*** (single) value or a table.

Scalar Function – A SQL Scalar Function takes one or more parameters and returns a single value.

```
CREATE FUNCTION dbo.GetNetSale
( @quantity int,
  @unitprice decimal(10,2),
  @discount decimal(10,2)
)
RETURNS decimal(10,2)
AS
BEGIN
  return
        @quantity*@unitprice*(1-@discount);
END
-- call the function
SELECT dbo.GetNetSale(10,100.00,0.1)
AS
netSale;
```

SQL Scalar Function

https://docs.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql?view=sql-server-ver15
https://docs.microsoft.com/en-us/sql/relational-databases/user-defined-functions/create-user-defined-functions-database-engine?
https://www.sqlservertutorial.net/sql-server-user-defined-functions/sql-server-scalar-functions/

To create a Scalar Function:

1. Use the CREATE FUNCTION keywords to name the function. SQL Server may require dbo. or the schema name.
2. In parenthesis, specify a list of @<parameterName> <dataType>.
3. Use the RETURNS keyword and give the data type of the return value.
4. Use the AS keyword and BEGIN the function body.
5. RETURN the calculation.
6. End the body of the function with END.
7. To call the function,
   - SELECT <functionName(params)> AS <name>

```
CREATE FUNCTION dbo.GetNetSale
( @quantity int,
  @unitprice dec(10,2),
  @discount dec(10,2)
)
RETURNS dec(10,2)
AS
BEGIN
  return
        @quantity*@unitprice*(1-@discount);
END
-- call the function
SELECT dbo.GetNetSale(10,100.00,0.1)
AS
netSale;
```

SQL – User-Defined Function

Scalar Functions operate on a single value and then return a single value.
Scalar functions can be used wherever an expression is valid.

```
GO
CREATE FUNCTION Poke.TotalNumberOfPokemon()
RETURNS INT
AS
BEGIN
    DECLARE @result INT;

    SELECT @result = COUNT(*) FROM Poke.Pokemon;

    RETURN @result;
END
GO

SELECT Poke.TotalNumberOfPokemon();
```

SQL – User-Defined Function

SQL Functions have "read-only" access. They cannot make changes to the database.

```
GO
CREATE FUNCTION Poke.PokemonWithNameOfLength(@length INT)
RETURNS TABLE
AS
    RETURN (
        SELECT * FROM Poke.Pokemon WHERE LEN(Name) = @length
    );
GO

SELECT * FROM Poke.PokemonWithNameOfLength(8);
```

Table-Valued Parameters

https://docs.microsoft.com/en-us/sql/relational-databases/tables/use-table-valued-parameters-database-engine?view=sql-server-ver15

A ***Table-Valued Parameter*** is a Function parameter that is actually a SQL table.

This example:

1. creates a ***table-valued*** parameter type,
2. declares a variable to reference it,
3. fills the parameter list, and
4. passes the values to a ***Stored Procedure***.

```
/* Create a table type. */
CREATE TYPE LocationTableType
   AS TABLE
      ( LocationName VARCHAR(50)
      , CostRate INT );
GO
/* Create a procedure to receive data for the table-valued parameter. */
CREATE PROCEDURE dbo. usp_InsertProductionLocation
   @TVP LocationTableType READONLY
      AS
      SET NOCOUNT ON
      INSERT INTO AdventureWorks2012.Production.Location
         (
            Name
            , CostRate
            , Availability
            , ModifiedDate
         )
      SELECT *, 0, GETDATE()
      FROM @TVP;
GO
/* Declare a variable that references the type. */
DECLARE @LocationTVP AS LocationTableType;
/* Add data to the table variable. */
INSERT INTO @LocationTVP (LocationName, CostRate)
   SELECT Name, 0.00
   FROM AdventureWorks2012.Person.StateProvince;

/* Pass the table variable data to a stored procedure. */
EXEC usp_InsertProductionLocation @LocationTVP;
```

Function access

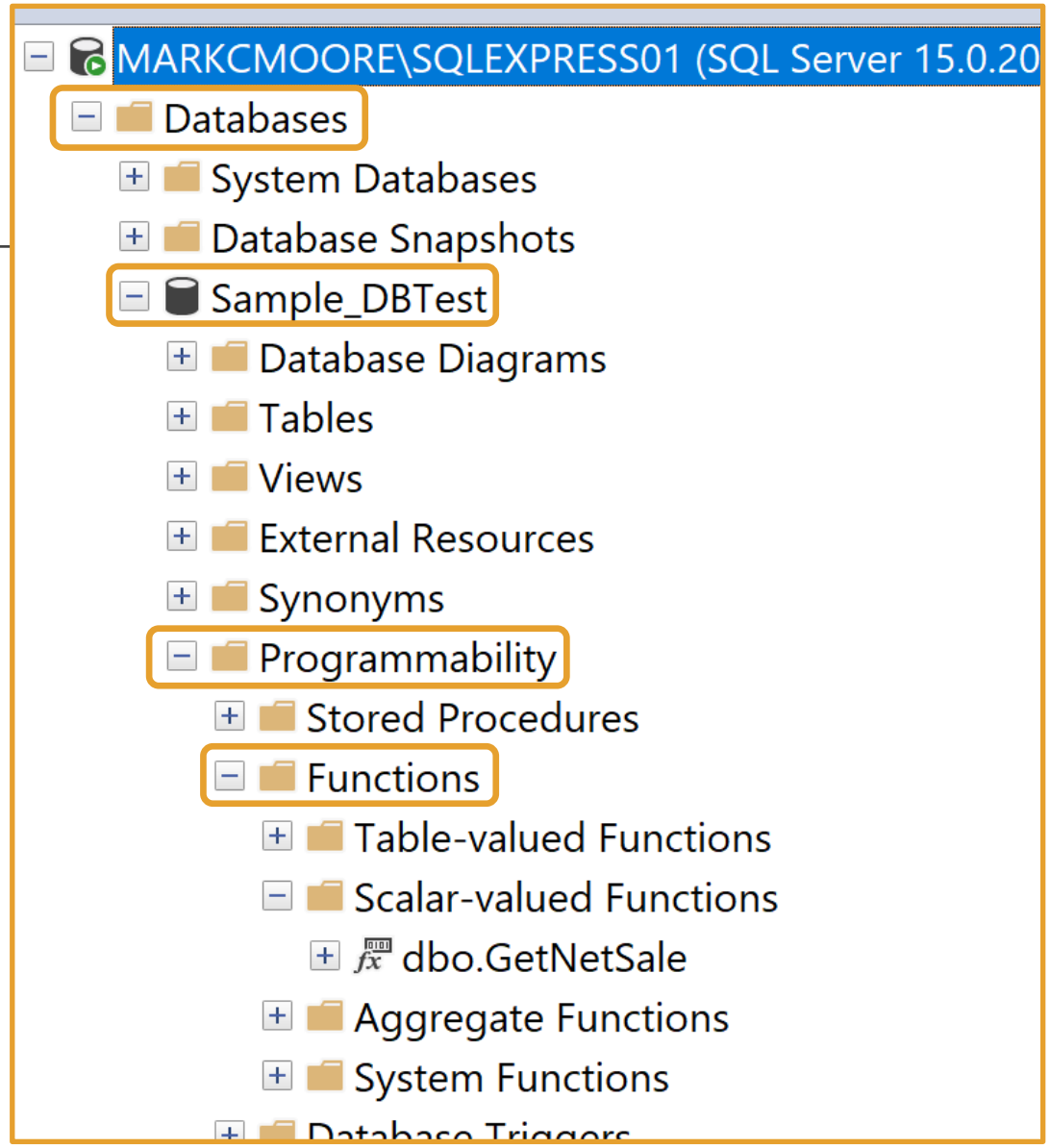
Object Explorer

>>Databases

>>[DbName]

>>Programmability

>>Functions



AGGREGATE Functions

https://docs.microsoft.com/en-us/sql/t-sql/functions/aggregate-functions-transact-sql?view=sql-server-ver15
https://docs.microsoft.com/en-us/sql/t-sql/functions/functions?view=sql-server-ver15#aggregate-functions

---

Aggregate functions:

- perform a calculation on a set of values and returns a single value.
- ignore null values (except for ***COUNT()***).
- are often used with the ***GROUP BY*** clause of the ***SELECT*** statement.

These are Aggregate functions

APPROX_COUNT_DISTINCT	MIN
AVG	STDEV
CHECKSUM_AGG	STDEVP
COUNT	STRING_AGG
COUNT_BIG	SUM
GROUPING	VAR
GROUPING_ID	VARP
MAX	

AVG( ) - Average

https://docs.microsoft.com/en-us/sql/t-sql/functions/avg-transact-sql?view=sql-server-ver15

AVG() computes the average of a set of values by dividing the sum of those values by the count of non-null values. If the sum exceeds the maximum value for the data type of the return value, ***AVG()*** will return an error. ***AVG()*** can have 1 or 2 arguments.

- ALL – (default) Applies the aggregate function to all values.
- DISTINCT - Specifies that ***AVG()*** operates only on one unique instance of each value, regardless of how many times that value occurs.
- EX. **SELECT AVG(ALL NumbersColumn) FROM TableName;**
- The above example returns the average of all numbers. Even duplicates.

This example returns the average vacation hours each Vice President has and how many total sick leave hours all Vice Presidents have together.

```
SELECT AVG(VacationHours)AS 'Average vacation hours',
    SUM(SickLeaveHours) AS 'Total sick leave hours'
FROM HumanResources.Employee
WHERE JobTitle LIKE 'Vice President%';
```

COUNT( )

https://docs.microsoft.com/en-us/sql/t-sql/functions/count-transact-sql?view=sql-server-ver15

COUNT( ) returns the number of items found in a group. ***COUNT( )*** always returns an ***int***.

COUNT( ) has two possible arguments:

- ALL - Applies the aggregate function to all values. ALL serves as the default.
- DISTINCT - Specifies that COUNT returns the number of unique nonnull values.

This example returns the number of unique job titles there are in all.

```
SELECT COUNT(DISTINCT Title)
FROM HumanResources.Employee;
GO
```

SUM( )

https://docs.microsoft.com/en-us/sql/t-sql/functions/sum-transact-sql?view=sql-server-ver15

SUM() can be used with numeric columns only. Null values are ignored.

SUM( ) has two possible arguments:

- ALL – Default. Applies the aggregate function to all values.
- DISTINCT - Specifies that SUM returns the sum of unique values.

```
SELECT Color, SUM(ListPrice), SUM(StandardCost)
FROM Production.Product
WHERE Color IS NOT NULL
    AND ListPrice != 0.00
    AND Name LIKE 'Mountain%'
GROUP BY Color
ORDER BY Color;
GO
```

```
Color
-------------- --------------------- -----------
Black          27404.84              5214.9616
Silver         26462.84              14665.6792
White          19.00                 6.7926
```