* A function is a stored program that you can pass parameters into and return a value.
* The syntax to create a function :-

CREATE FUNCTION [schema\_name.]function\_name

( [ @parameter [ AS ] [type\_schema\_name.] datatype

[ = default ] [ READONLY ]

, @parameter [ AS ] [type\_schema\_name.] datatype

[ = default ] [ READONLY ] ]

)

RETURNS return\_datatype

[WITH {ENCRYPTION

| SCHEMABINDING

| RETURNS NULL ON NULL INPUT

| CALLED ON NULL INPUT

| EXECUTE AS Clause]

[AS]

BEGIN

[declaration\_section]

executable\_section

RETURN return\_value

END;

**schema\_name -** The name of the schema that owns the function.

**function\_name -**The name to assign to this function in SQL Server.

**@parameter -** One or more parameters passed into the function.

**type\_schema\_name -** The schema that owns the data type, if applicable.

**Datatype -** The data type for @*parameter*.

**Default-** The default value to assign to @*parameter*.

**READONLY -** It means that @parameter cannot be overwritten by the function.

**return\_datatype -** The datatype of the function's return value.

**ENCRYPTION -** Means that the source for the function will not be stored as plain text in the system views in SQL Server.

**SCHEMABINDING –** Means that the underlying objects cannot be modified so as to affect the function.

**RETURNS NULL ON NULL INPUT –** Means that the function will return NULL if any parameters are NULL without having to execute the function.

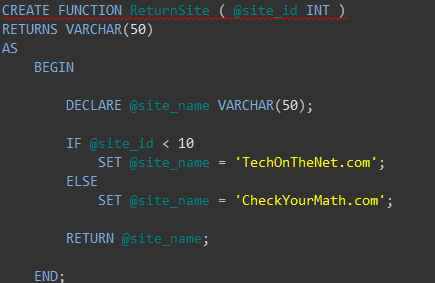
**CALL ON NULL INPUT –** Means that the function will execute the function even if any parameters are NULL.

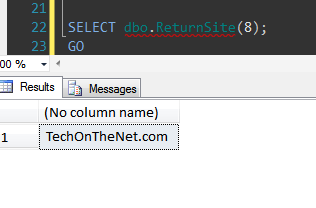
**EXECUTE AS clause -** Sets the security context to execute the function.

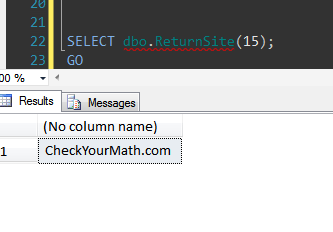
**return\_value -** The value returned by the function.

Example

**This function is called ReturnSite. It has one parameter called @site\_id which is an INT datatype. The function returns a VARCHAR(50) value, as specified by the RETURNS clause.**



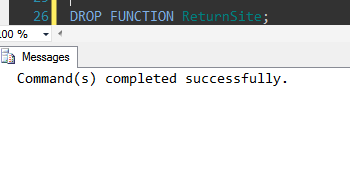




* The syntax to drop a function:-

DROP FUNCTION *function\_name*;

**function\_name -** The name of the function that you wish to drop.



**User Defined Functions**

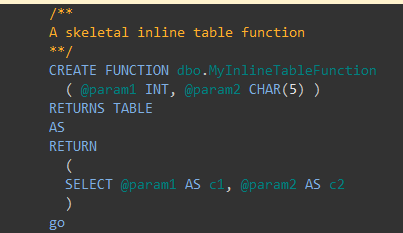
* To perform a complex logic, can accept parameters and return data.
* For example, we can call user defined function in a where clause or use a user defined function in a JOIN Where UDF returns a result set.
* SQL Server supports two types of User Defined Functions as mentioned below –   
  Scalar Functions – The function which returns a Scalar/Single value.  
  Table Valued Functions – The function which returns a row set of SQL server Table datatype.
* Inline Table
* Multi-statement Table
* Inline table-valued functions return a TABLE datatype. They each contain a single T-SQL statement. (**Table** is primarily used for temporary storage of a set of rows returned as the result set of a table-valued function.)
* Multi-statement table-valued functions return a defined table. They can contain multiple T-SQL statements.
* Inline functions do not have associated return variables, they just return a value functions.
* Multi-statement functions have a function body that is defined in a BEGIN…END block, consisting of a series of Transact-SQL statements
* In the case of a multi-statement table-valued function, these T-SQL statements build and insert rows into the TABLE variable that is then returned.
* In inline table-valued functions, the TABLE return value is defined through a single SELECT statement.

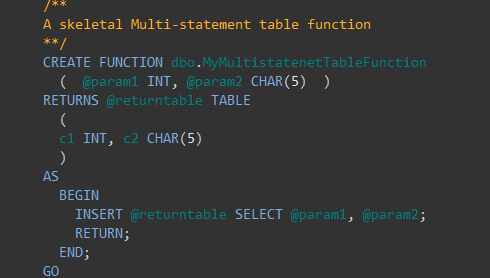
Why use them?

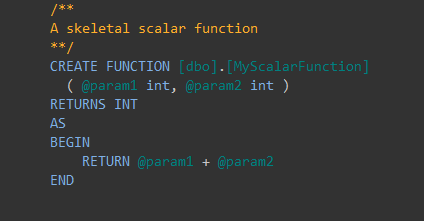
* They allow modular programming: - You can create the function once, store it in the database, and call it any number of times in your program. User-defined functions can be modified independently of the program source code.
* They allow faster execution: - Similar to stored procedures, Transact-SQL user-defined functions reduce the compilation cost of Transact-SQL code by caching the plans and reusing them for repeated executions. This means the user-defined function does not need to be reparsed and optimized with each use resulting in much faster execution times.
* They can reduce network traffic:-An operation that filters data based on some complex constraint that cannot be expressed in a single scalar expression can be expressed as a function. The function can then invoked in the WHERE clause to reduce the number or rows sent to the client.

## Restrictions

* Cannot create a temporary function in the same way as you would a temporary table or procedure.(with the # or ## prefix)
* Cannot use SET statements in a user-defined function to change the current session handling, because of the danger of producing a side-effect.
* Cannot be used to perform any actions that modify the database state, such as writing to a table or even using an OUTPUT INTO clause that has a table as its target.
* Cannot return a result set, only a single table data type. Stored procedure, in contrast, can be used to return one or more result sets.
* Restricted error handling. It supports neither RAISERROR nor TRY…CATCH. You can’t get at the @ERROR.
* Cannot call a stored procedure from within a UDF, but you can call an extended stored procedure.

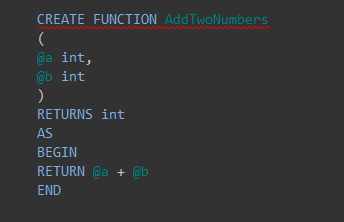


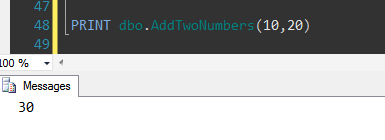
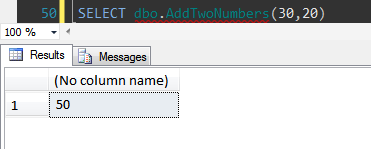


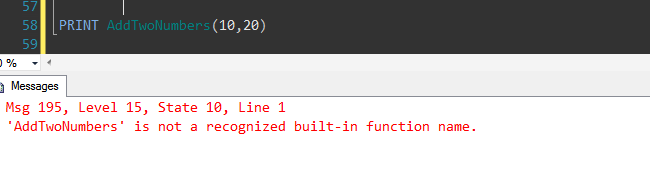


**Scalar User-Defined Function**

* Accept 0 to many input parameter and will return a single value.
* Returns one of the scalar (int, char, varchar etc) data types.
* Text, ntext, image and timestamp data types are not supported.
* For Scalar UDFS we need to use Two Part Naming Convention i.e. in the above two statements we are using **dbo.AddTwoNumbers.**

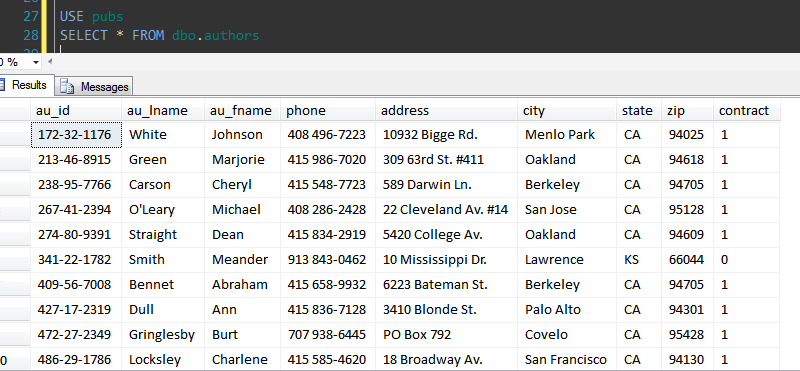


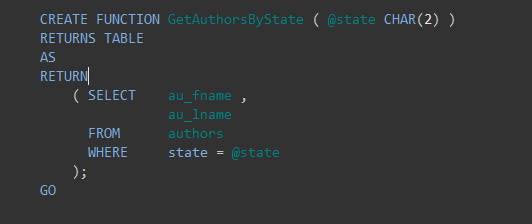


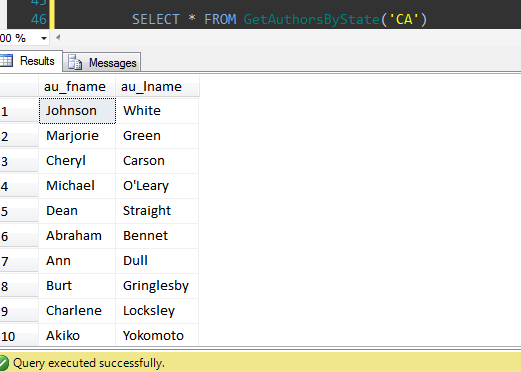


**Inline Table-Valued User-Defined Function**

* Returns a variable of data type table whose value is derived from a single SELECT statement.
* Since the return value is derived from the SELECT statement, there is no BEGIN/END block needed in the CREATE FUNCTION statement.
* No need to specify the table variable name (or column definitions for the table variable) because the structure of the returned value is generated from the columns that compose the SELECT statement.
* Because the results are a function of the columns referenced in the SELECT, no duplicate column names are allowed and all derived columns must have an associated alias.
* In this example we are creating a Inline table-valued function **GetAuthorsByState**which accepts state as the input parameter and returns first name and last name of all the authors belonging to the input state.







**Multi-statement Table-Valued User-Defined Function**

* A Multi-Statement Table-Valued user-defined function returns a table.
* Have one or more than one T-Sql statement.
* Within the create function command you must define the table structure that is being returned.
* After creating this type of user-defined function, we can use it in the FROM clause of a T-SQL command.
* In this example we are creating a Multi-Statement Table-Valued function **GetAuthorsByState**which accepts state as the input parameter and returns author id and first name of all the authors belonging to the input state.
* If for the input state there are no authors then this UDF will return a record with no columns.

