Cheryl Aguirre

March 11, 2025

IT FDN 130 A

GitHub Repository

Module 07: Functions

Introduction

This module explored what functions are, how users can create user defined functions (UDFs), and explained the difference between scalar, inline, and multi-statement functions. This paper explains the purpose of UDFs, how they're related to functions, and the need for scalar, inline, and multi-statement functions.

User Defined Functions (UDFs)

In SQL, functions are built-in programs that execute specific actions to manipulate and clean data. There are many functions that this course taught that come in handy every time we do homework. UDFs are functions created by users that perform specific programmed actions. They are customized to make calculations, clean data, and allow more accessible retrieval of data. UDFs can return single (scalar) values, or tables, depending on the specifics programmed by the user. Like views, UDFs help developers and administrators access data more efficiently, removing the need to rewrite code each time they access SQL. UDFs are customizable and helpful pieces of code that simplify routine data accessibility and analysis.

Scalar, Inline, and Multi-Statement Functions

There are different types of functions! UDFs and pre-programmed functions can be broken down into 3 categories: scalar, inline, and multi-statement functions. Let's walk through each kind of function and explore their unique qualities.

Scalar Functions

Scalar functions return single values, for example a single integer, a single string of characters, or a piece of data like a date! Scalar functions do not support multiple statements; however, users CAN write more than one parameter in their function to get a

single value. Parameters are placeholders for values you customize scalar functions with! It helps functions work with different values to generate new results, instead of working with fixed values. Let's look at one as an example:

```
CREATE FUNCTION Subtraction (@a INT, @b INT)
      RETURNS INT
AS
BEGIN
    RETURN @a - @b;
END;
GO
```

This function uses 2 integer parameters, a and b! I specify that I want my scalar function to return the data as int value, and I specify that I want @b to be subtracted from @a. To call this function, I would type below:

```
Select dbo.Subtraction (32, 16) AS Results;
GO
```

When I call my function, I input the 2 integer parameters, and I name the results for ease of reading. My results in SQL server look something like the below:

Results

16

In short, scalar functions are single or multi-parameter functions that users can customize to produce a single result.

Inline Table-Value Functions (TVF)

TVFs are very similar to scalar functions! They can be customized with multiple parameters and can return data as a specific type; however, TVFs return results that are formatted as tables. TVFs do not require BEGIN and END like scalar and multi-statement functions. Let's look at how these are formatted!

```
CREATE FUNCTION InStockProduct(@ProductSKU VARCHAR(75))
RETURNS TABLE
AS
```

```
RETURN (
    SELECT ProductSKU, Count, Department
    FROM Products
    WHERE ProductSKU = @ProductSKU AND InStock = 1
);
GO
```

This function calls a table with a single specified product that is currently in stock and uses a single parameter. To make a function a TVF, users must state that it returns as a table, then define the table in the return statement. In this case, I've defined my results columns as ProductSKU, Count, and Department and I'm only calling up products that are in stock (the 1 is dependent upon an imaginary column in the imaginary products table using BIT). To call this function, my command would look like this:

```
SELECT * FROM dbo.InStockProduct(33319);
```

My result set would be a small table with ProductSKU, Count, and Department and would only work if the SKU number I chose was in stock!

Multi-Statement Functions

MSFs are super similar to TVFs—they are also customizable user defined functions that return tables as their result set. They differ because they can contain multiple statements for more complex table results. Let's explore an example. I want to create an MSF that calls all products from a specific department in my store and I want to apply a small discount to products that are above $200:

```
CREATE FUNCTION ProductsByDpt (@DepartmentName NVARCHAR(50))
RETURNS @ProductTable TABLE (
    ProductID INT,
    ProductName VARCHAR(100),
    Department VARCHAR(50),
    Price DECIMAL(10, 2))
AS
BEGIN
```

```
-- Statement that Inserts products from a department into the table
    INSERT INTO @ProductTable
    SELECT ProductID, ProductName, Department, Price
    FROM Products
    WHERE Department = @DepartmentName;
    UPDATE @ProductTable
-- Apply a discount of 10% to all products that fit our criteria.
    SET Price = Price * 0.9
-- Filter products based on price
    WHERE Price > 200;
-- Return the table with products
    RETURN;
END;
GO
```

Now I have an MSF that applies a discount price to products we specify from a department that are above $200. If I wanted to use this function, I would call it like so:

```
SELECT * FROM dbo.ProductsByDpt('Jewelry');
```

Instead of having to do math (god forbid) I can call this function and apply it to a department any time I need to have a sale. I can update the function to include different discounts, too! I use multiple statements to achieve this function which is why it's an MSF and not a TVF.

Summary

Scalar functions, TVFs, and MSFs are all different formats UDFs can take that return a single value (scalar) or table values (TVFs and MSFs). I have explained each kind of function as well as given examples for practical applications of each kind of UDF. UDFs are customizable programs that users can modify to fit all sorts of business and data needs!