Name: Silvano Ross
Date: November 25, 2021
Course: IT FDN 130A
Github: https://github.com/silvanoross/DBFoundations-Module07

Assignment 07 - Functions (Form Follows)

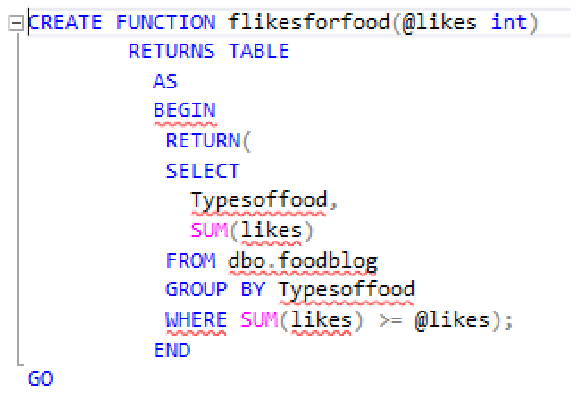
Introduction:

This writeup will be solely about functions. There are many different kinds of functions in SQL and we will make sure to distinguish them. Namely, there are built-in SQL functions and different kinds of user-defined functions (UDFs) - functions that are uniquely designed by the user. There are different flavors of UDFs that serve different purposes which we will discuss later on in this writeup. It is also important to note that UDFs, unlike views, are executed when called upon in a query rather than being stored as an abstraction layer in a database.

Purpose and Use of a UDF

UDFs can be created for a variety of helpful reasons when built-in SQL functions are limited in their ability to solve a problem. A significant reason of creating UDFs is so we, as programmers, can stick to the approach of "Don't Repeat Yourself (DRT)" - a software development theory (3). UDFs allow you to create reusable pieces of code in place of redundancies. They can return a single datatype or an entire specified table that is not fleeting (like procedures). Also, they are capable of having parameters passed through them which makes them more versatile than a view.

Let's say you want to see a table that shows the types of foods with more than 15k likes from an imaginary food blog database called "dbo.foodblog". To accomplish this with a view you would have to create a view that pulls is typesoffoods and SUM(likes) AS 'likes' then set a WHERE SUM(likes) >= 15000. This locks in this view and you are only able to pull types of foods with 15k likes or more. But let us say we want to now see types of foods with 10k likes or more. We have to create an entire new view or alter the current view to do so. Alternatively if we create a function that passes a parameter of likes through we can variably select the amount of likes we want to see and the types of foods associated with them. Here is what it would look like.



```
CREATE FUNCTION flikesforfood(@likes int)
        RETURNS TABLE
          AS
          BEGIN
           RETURN(
           SELECT
             Typesoffood,
             SUM(likes)
           FROM dbo.foodblog
           GROUP BY Typesoffood
           WHERE SUM(likes) >= @likes);
          END
GO
```

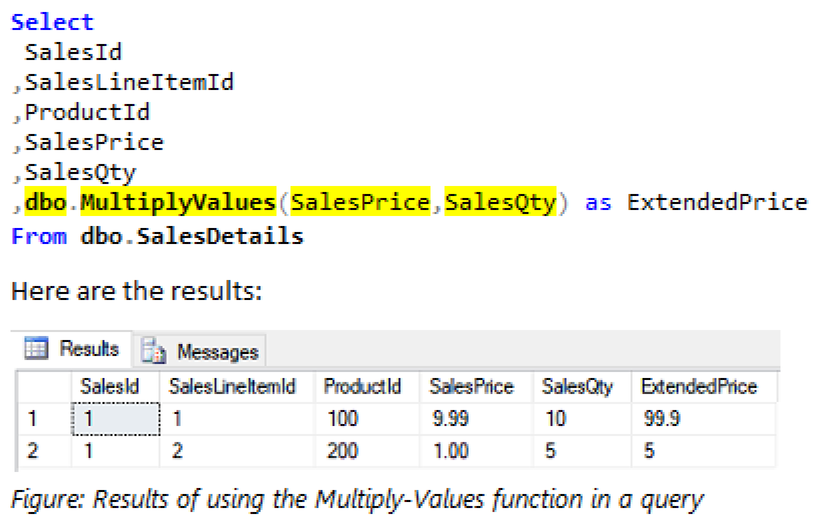
Figure 1.1 S. Ross *Random Query* 2021

We can now pass in any number of likes through our function by: SELECT * FROM flikesforfood(n) - (n = number of likes) - and we can generate a table. This example is to give a reason for using a UDF in place of a view. In this case we needed to variably be able to weed through data without having to create multiple different views and UDFs give us this opportunity.

UDFs can also be utilized as check constraints where variable data being entered into a database is evaluated through a created function to check if the data meets enumerated requirements. Check constraints in general are a little more simple, but a function used within a check constraint can often compare different variables within a database and between tables making the check constraints themselves more complex.

Differences Between Functions - (Scalar/In-line/Multi-Statement)

Scalar Functions: As the name alludes to, although it is not too incredibly clear, these functions can be used individually, or they can be "scaled" up. This is akin to how a new company aims on scaling up its business and creating operations that are implementable many times over with ease. This means, in SQL terms, a scalar function can be incorporated into a select statement and perform its code on every single defined parameter throughout a specified table or be called on multiple times.



```sql
Select
 SalesId
,SalesLineItemId
,ProductId
,SalesPrice
,SalesQty
,dbo.MultiplyValues(SalesPrice,SalesQty) as ExtendedPrice
From dbo.SalesDetails
```

Here are the results:

Results | Messages

	SalesId	SalesLineItemId	ProductId	SalesPrice	SalesQty	ExtendedPrice
1	1	1	100	9.99	10	99.9
2	1	2	200	1.00	5	5

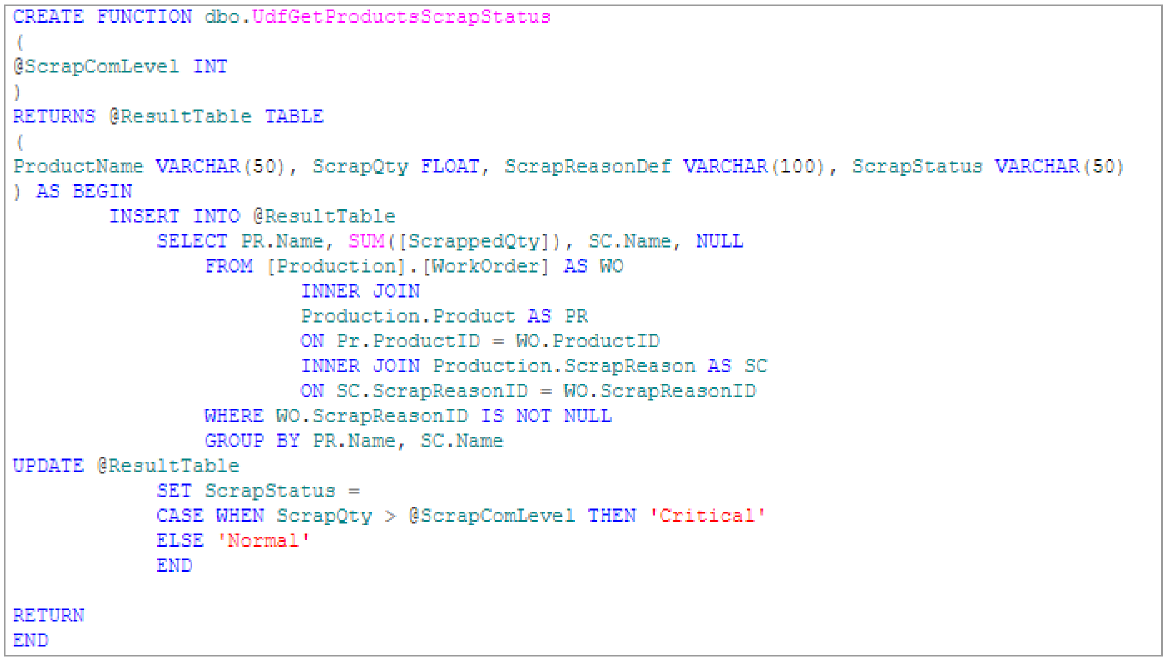
Figure: Results of using the Multiply-Values function in a query

Figure 2.1 R. Root et al, *Module 07 Notes* 2021

Here we can see the function “dbo.MultiplyValues()” operate on two different attributes within the table “SalesPrice,” and “SalesQty” and return a defined column “ExtendedPrice” representing the total price of the sale on a line item. The function was defined earlier to multiply the first and second parameter entered into the function. This can be called on to do more work on different parameters in different contexts.

In-line Functions: These functions are much simpler in how they are used, but their base coding can still be complex. They often only hold one statement and are called upon “in-line” or by simply using:
SELECT * FROM fFunction(parameter);
GO
They will mostly return a table that has been defined by the inserted parameter and do not usually require a BEGIN or END circumscription.

Multi-Statement Table Valued Functions (MSTVFs): These functions are exactly as their name implies, multi-statement and centered around a table. Unlike an in-line function which holds only one statement, these functions can hold a multitude of statements if necessary. They are also more nuanced in their coding returning a self defined and created table. Let’s take a look at an example from (3).

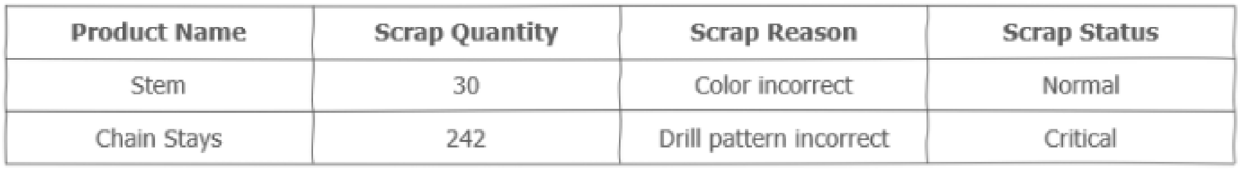


```
CREATE FUNCTION dbo.UdfGetProductsScrapStatus
(
@ScrapComLevel INT
)
RETURNS @ResultTable TABLE
(
ProductName VARCHAR(50), ScrapQty FLOAT, ScrapReasonDef VARCHAR(100), ScrapStatus VARCHAR(50)
) AS BEGIN
        INSERT INTO @ResultTable
            SELECT PR.Name, SUM([ScrappedQty]), SC.Name, NULL
                FROM [Production].[WorkOrder] AS WO
                        INNER JOIN
                        Production.Product AS PR
                        ON Pr.ProductID = WO.ProductID
                        INNER JOIN Production.ScrapReason AS SC
                        ON SC.ScrapReasonID = WO.ScrapReasonID
                WHERE WO.ScrapReasonID IS NOT NULL
                GROUP BY PR.Name, SC.Name
UPDATE @ResultTable
            SET ScrapStatus =
            CASE WHEN ScrapQty > @ScrapComLevel THEN 'Critical'
            ELSE 'Normal'
            END

RETURN
END
```

Figure 2.2 *Image from* https://www.sqlshack.com/sql-server-multi-statement-table-valued-functions/

For this example in particular a company wants to know the "Scrap Status" of certain products to determine which products have to be tossed due to improper manufacturing and whether or not this is harming their budget due to the amount of scrapped items exceeding the predetermined, budgeted amount. The return table below is what will show.



Product Name	Scrap Quantity	Scrap Reason	Scrap Status
Stem	30	Color incorrect	Normal
Chain Stays	242	Drill pattern incorrect	Critical

Figure 2.3 *Image from* https://www.sqlshack.com/sql-server-multi-statement-table-valued-functions/

If we pick this MSTVF apart we can see a plethora of SQL techniques. A table is being created - @ResultTable. It is having data inserted into it from 3 different tables within the specified database with an aggregate function in the first statement including a renaming of these values. Then, in the second statement, we can see that the one value initially coded as "NULL" will be updated to hold a case value to check if the parameter that is tested (@ScrapCOMlevel) when utilizing the MSTVF is a "Critical" amount of scrapped product or a "Normal" amount of scrapped product. This is an incredibly intricate piece of code that falls into a singular function.

Summary

We have seen when a UDF is important and it aids in avoiding redundancy in coding. UDFs are also important in creating complex check constraints as well as variable select statements that would otherwise require multiple views. They help streamline processes and can be used in a variety of ways. Scalar functions can be used, reused and recycled performing their code in a scalable way. In-line functions allow for single statement functions to be called upon with variable defined parameters. Multi-statement table valued functions, as we’ve seen, can be quite complex, holding more than a single statement and returning a declared table that allows us to answer more nuanced questions within a database. All of these functions fall under the function category, but they all have their unique uses and niches allowing for a versatile SQL coding repertoire. .

Bibliography

1. S. Ross *Random Query* 2021
2. R. Root et al, *Module 07 Notes* 2021
3. https://www.sqlshack.com/sql-server-multi-statement-table-valued-functions/
4. https://www.wiseowl.co.uk/blog/s347/in-line.htm