Name: Katarzyna (Kasia) Swica
Date: November 30, 2022
Course: IT FDN 130 A Au 22: Foundations of Databases and SQL Programming
GitHub Link: https://github.com/kswica-uw/DBFoundations-Module07

Assignment 7- Functions

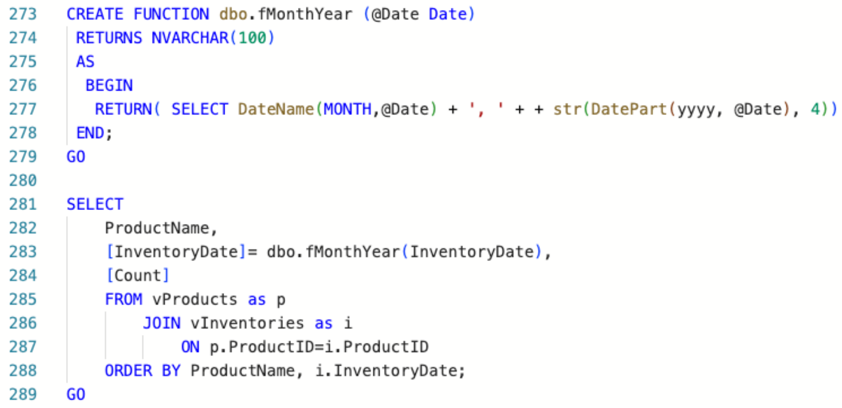
Introduction

This document will cover the different uses for SQL User Defined Functions (UDFs) and the differences between Scalar, Inline, and Multi-Statement Functions.

User Defined Functions

Functions, unlike Views, enable a user to set parameters that change the results of a query. This can be particularly helpful in reporting because the user is able to specify inputs and have those parameters drive the results of the underlying query. Check constraints are a common use case for custom scalar functions (https://www.youtube.com/watch?v=NxNJJvG7FzU, 2022).

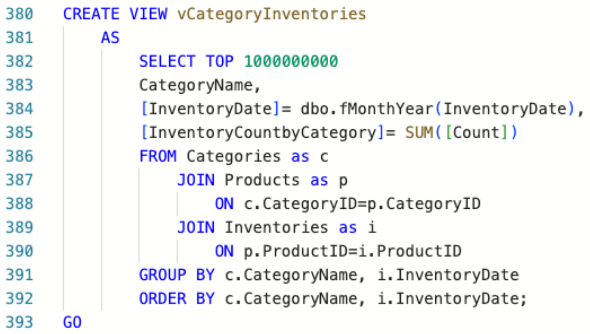
There are many system-defined SQL functions, including the aggregate functions (i.e. SUM, MAX, AVG, MIN), date functions (i.e. GetDate, IsDate), and functions that enable the user to reformat data (i.e. CAST, CONVERT, FORMAT, CASE). SQL User Defined Functions or UDFs enable a user to create a unique saved SELECT statement that returns either a single value or a table. UDFs are often created when a SELECT statement is required more than once, and the results of that SELECT statement depend on a changing input. Rather than having each user and/or use case repeat similar code, a UDF can be called. This is particularly helpful when the underlying SELECT statement is complex.



```
CREATE FUNCTION dbo.fMonthYear (@Date Date)
 RETURNS NVARCHAR(100)
 AS
  BEGIN
   RETURN( SELECT DateName(MONTH,@Date) + ', ' + + str(DatePart(yyyy, @Date), 4))
 END;
GO

SELECT
    ProductName,
    [InventoryDate]= dbo.fMonthYear(InventoryDate),
    [Count]
    FROM vProducts as p
        JOIN vInventories as i
            ON p.ProductID=i.ProductID
    ORDER BY ProductName, i.InventoryDate;
GO
```

Figure 1 Sample user defined function and its use within a SELECT statement. This UDF converts a date into a nvarchar custom format.



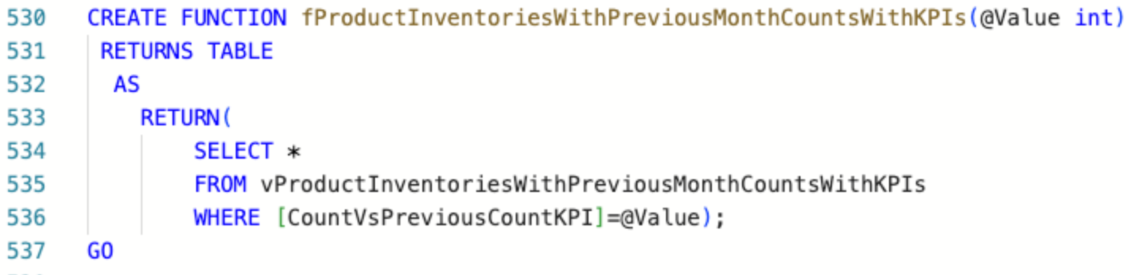
```
CREATE VIEW vCategoryInventories
    AS
        SELECT TOP 1000000000
        CategoryName,
        [InventoryDate]= dbo.fMonthYear(InventoryDate),
        [InventoryCountbyCategory]= SUM([Count])
        FROM Categories as c
            JOIN Products as p
                ON c.CategoryID=p.CategoryID
            JOIN Inventories as i
                ON p.ProductID=i.ProductID
        GROUP BY c.CategoryName, i.InventoryDate
        ORDER BY c.CategoryName, i.InventoryDate;
GO
```

Figure 2 The UDF from Figure 1 is re-used here within a View.

Different Types of Functions

Scalar functions return a single value. The syntax of a scalar function can be seen in Figure 1. The table schema name must be included in the name of the function (i.e. 'dbo'). Input variables and their data type are assigned, followed by 'Returns' and the definition of the output variable data type. The SELECT statement driving the function must be prefaced by the word 'Return'.

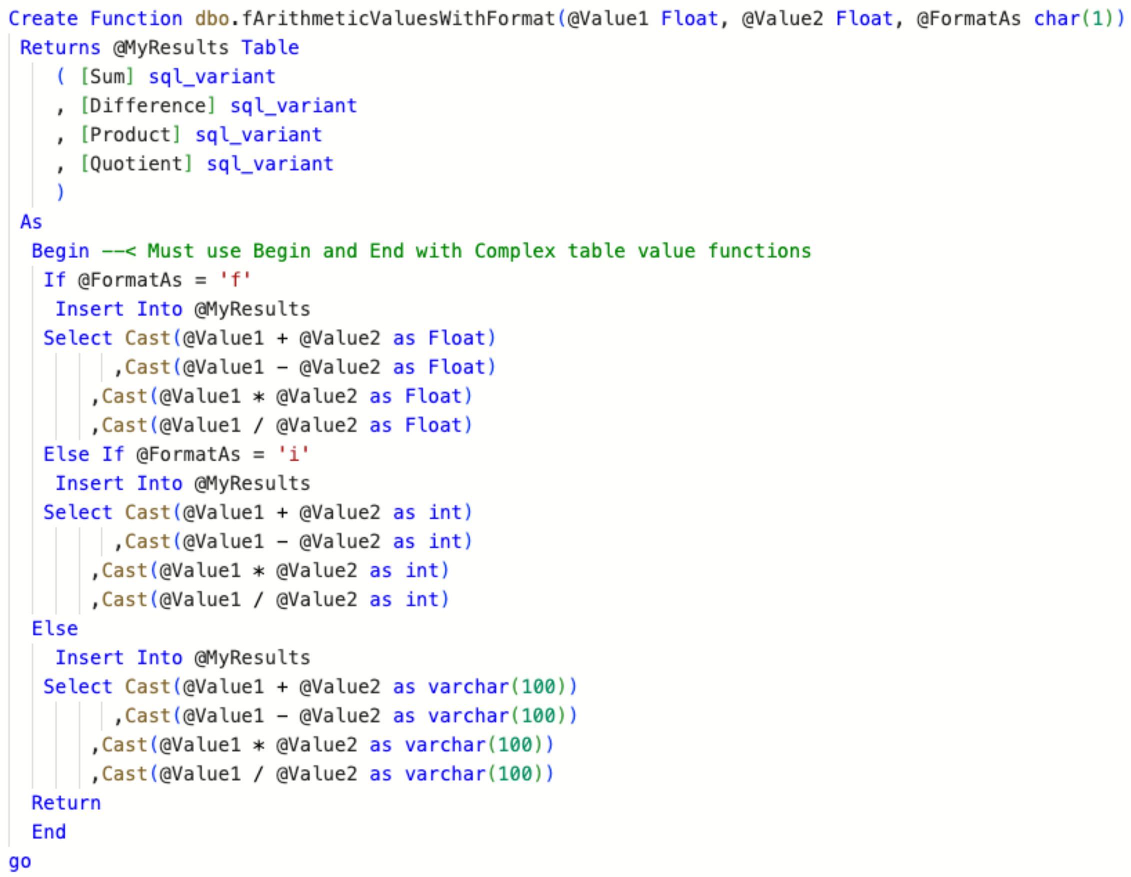
Inline functions, also known as simple table-valued functions, return a table of values ('Returns Table'). Inline functions can have multiple input parameters, but only return one SELECT statement. Once created, this function can be called as if it were a table.



```
CREATE FUNCTION fProductInventoriesWithPreviousMonthCountsWithKPIs(@Value int)
 RETURNS TABLE
  AS
    RETURN(
        SELECT *
        FROM vProductInventoriesWithPreviousMonthCountsWithKPIs
        WHERE [CountVsPreviousCountKPI]=@Value);
GO
```

Figure 3 A sample inline table valued function.

Multi-statement functions (MSFs), like inline functions, return a table of values. However, the construction of multi-statement functions is more complex. MSFs involve the explicit definition of a table and its columns, followed by several select statements. The results of these more complex functions can be called as though they were a table.



```
Create Function dbo.fArithmeticValuesWithFormat(@Value1 Float, @Value2 Float, @FormatAs char(1))
 Returns @MyResults Table
    ( [Sum] sql_variant
    , [Difference] sql_variant
    , [Product] sql_variant
    , [Quotient] sql_variant
    )
 As
  Begin --< Must use Begin and End with Complex table value functions
   If @FormatAs = 'f'
    Insert Into @MyResults
   Select Cast(@Value1 + @Value2 as Float)
         ,Cast(@Value1 - @Value2 as Float)
       ,Cast(@Value1 * @Value2 as Float)
       ,Cast(@Value1 / @Value2 as Float)
   Else If @FormatAs = 'i'
    Insert Into @MyResults
   Select Cast(@Value1 + @Value2 as int)
         ,Cast(@Value1 - @Value2 as int)
       ,Cast(@Value1 * @Value2 as int)
       ,Cast(@Value1 / @Value2 as int)
  Else
    Insert Into @MyResults
   Select Cast(@Value1 + @Value2 as varchar(100))
         ,Cast(@Value1 - @Value2 as varchar(100))
       ,Cast(@Value1 * @Value2 as varchar(100))
       ,Cast(@Value1 / @Value2 as varchar(100))
  Return
  End
go
```

Figure 4 Sample multi-statement function taken from Professor Randal Root's Module 7 Notes.

Summary

SQL User Defined Functions or UDFs enable a user to create a unique saved SELECT statement with defined parameters that returns either a single value or a table. Scalar functions return a single value while table defined functions, such as in-line or multi-statement functions, return a table of values.