**Name:** Robert Moulton
**Date:** Nov 30, 2022
**Course:** IT FDN 130 A Au 22: Foundations Of Databases & SQL Programming
**Github:** https://github.com/rmoultonuw/DBFoundations-Module07

# Assignment 7 - Functions

## Introduction

In this assignment I learned about the various types of SQL functions, including user-defined functions (UDFs), in particular – their benefits, and when they might be used.

### Use of UDFs

On the Microsoft "Learn" site, user-defined functions are described as routines that accept parameters, perform actions, and return results as scalar values or as result sets.
Reference:
https://learn.microsoft.com/en-us/sql/relational-databases/user-defined-functions/user-defined-functions

The site lists some benefits of user-defined functions and reasons for using them:

> - 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.
> - 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 doesn't need to be reparsed and reoptimized with each use resulting in much faster execution times.
>   CLR functions offer significant performance advantage over Transact-SQL functions for computational tasks, string manipulation, and business logic. Transact-SQL functions are better suited for data-access intensive logic.
> - Reduce network traffic. An operation that filters data based on some complex constraint that can't be expressed in a single scalar expression can be expressed as a function. The function can then be invoked in the WHERE clause to reduce the number of rows sent to the client.

### UDF Types

In SQL Server there are three types of user-defined functions: Scalar, Inline, and Multi-Statement. An explanation of the differences between the three UDF types follows.

*Note: The following examples are based on the Module07 course material – specifically, the SQL code file: “DEMOS - Functions.sql”*

## Scalar

A Scalar function simply returns a single value. An example:

```
-- create a scalar function ...
Create Function dbo.AddValues(@Value1 Float,@Value2 Float)
Returns Float
As
 Begin
  Return(Select @Value1 + @Value2);
 End
go
-- ... then call it
Select dbo.AddValues(4, 5);
Go
```

## Inline

An Inline Table Valued (ITV) function contains a single SQL statement and returns a table set. Example:

```
-- create an ITV function ...
Create Function dbo.ArithmeticValues(@Value1 Float, @Value2 Float)
Returns Table
As
  Return(
   Select [Sum] = @Value1 + @Value2,
 [Difference] = @Value1 - @Value2,
    [Product] = @Value1 * @Value2,
   [Quotient] = @Value1 / @Value2
 );
go
-- ... then call it
Select * FRom ArithmeticValues(4,5)
```

## Multi-Statement

A Multi-Statement Table Valued function contains multiple statements and returns a table set. Example:

```
-- create an MSTV function ...
go
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
  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
-- ... then call it
Select * FROM dbo.fArithmeticValuesWithFormat(10, 3, 'f');
Select * FROM dbo.fArithmeticValuesWithFormat(10, 3, 'i');
Select * FROM dbo.fArithmeticValuesWithFormat(10, 3, null);
go
```

## Summary

In this module, after learning about various types of SQL functions, I practiced using built-in functions and creating user-defined functions on my own.