SeQueL 4 – Queries – and their Hidden Functions!

by Clark Anderson

A friend recently exclaimed "Can you really use this function in SQL! In this article of my series I will explore and demonstrate many of the standard VBA built-in functions that can be used in SQL. I did use the Len() function in the previous article, **"SeQueL 3 – Queries - JOIN from Afar!"**

For starters, you can discover a lot of possibilities by creating a new form in Access in the design view adding a data entry control, like a TextBox. Open its properties. Under the Data tab, locate Control Source. Click in its entry field, then click on the "More" button with the ellipses (…), to open the Expression Builder. In the lower left window, double click on the Functions Folder plus, then click on the Build-In-Functions. The lower middle window will list Function Categories and the lower right window lists the Function names in the highlighted category. Double clicking on the function name or selecting and clicking the Paste button will paste the selected Function with arguments in the top Expression Builder window. If you select a function, then click the Help button, an explanation with some examples will be available. For some reason, my system cannot find the help file, VEENLR3.HLP. It asks me if I want to find it. Mine is located in the \Program Files\Common Files\Microsoft Shared\VBA directory. To save you some of this effort, I have listed almost all of these functions below.

TestTable2 contains the field DateOfYear with a series of dates. The first query uses some of the date functions. (DateDiff(), Date(), DatePart(), DateValue() & Month()) as well as some ProgramFlow functions (Choose() & IIF()).

DateDiff() determines the difference between two dates. Its first argument is the interval, "d" for days. The second is Date() to provide the current date. The third argument is the field, DateOfYear. The DaysFromNow values in the table sampling indicates the January & February dates were almost 300 days before the query was run.

The DatePart() function with its first argument, "w" for weekday, and second argument, DateOfYear, provides the DayOfWeek. It starts with 1 for Sunday through 7 for Saturday.

The DatePart() function is used again within the Immediate IF, IIF(), function to indicate which dates are on a Weekend and which are on a Weekday. The IIF() first argument is a Boolean expression: (Is the day of week a Sunday, 1, or a Saturday ,7.) If the Boolean expression is true, the second argument, "Weekend", becomes the result of the IIF() function, otherwise the third argument, "Weekday", is the result.

For the final column of this query, the DateValue() function converts the DateOfYear to a numeric representation that is used by the Month() function to extract the numeric month value, 1 to 12. This numeric month is used as the first argument, Index, for the CHOOSE() function to "choose" which of its list of arguments to provide to the data record.

```
SELECT
DateOfYear,
DateDiff ("d", Date(),DateOfYear ) AS DaysFromNow,
DatePart("w",DateOfYear) AS DayOfWeek,
IIF(DatePart("w",DateOfYear)=1 OR DatePart("w",DateOfYear)=7,"Weekend","Weekday") AS DayType,
CHOOSE(Month (DateValue(DateOfYear)),"Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec") AS
Month
FROM TestTable2;
```

DateOfYear	DaysFromNow	DayOfWeek	DayType	Month
01/28/2000	-292	6	Weekday	Jan
01/29/2000	-291	7	Weekend	Jan
01/30/2000	-290	1	Weekend	Jan
01/31/2000	-289	2	Weekday	Jan
02/01/2000	-288	3	Weekday	Feb
02/02/2000	-287	4	Weekday	Feb
02/03/2000	-286	5	Weekday	Feb
02/04/2000	-285	6	Weekday	Feb

The forth column could also have been achieved with the SQL phrase:

```
SWITCH(DatePart("w",DateOfYear) IN(1,7),"Weekend",DatePart("w",DateOfYear) IN(2,3,4,5,6),"Weekday") AS DayType,
```

This uses the SWITCH() function that is made up of a series of argument pairs. Each pair has a Boolean expression followed by a resultant expression. The Boolean expressions use DatePart() with the familiar IN() operator.

This next query uses the very powerful FORMAT$() function and the DateAdd() function. FORMAT$() creates a string based on the DateOfYear. The second argument is the format. The “dddd” represents the full name of the day. The “mmmm” is the full name of the month. The “dd” is the 2 digit day of the month. The “yyyy” is the 4 digit year and the spaces and comma are literal characters that will appear in the resultant string.

The next query data field uses the same format specification, but, its input argument depends on the function DateAdd(). The first argument of DateAdd() is an interval specification, “yyyy”, year. The second argument is –300 years, and the third argument is the DateOfYear. The resulting third column is the shorter formatted date 3 centuries prior to the DateOfYear. The 3 character “ddd” and “mmm” format specifications provide the abbreviated day and month names.

```
SELECT DateOfYear,
FORMAT$( DateOfYear ,"dddd mmmm dd, yyyy") AS FullDate,
FORMAT$(DateAdd ("yyyy",-300 , DateOfYear) ,"ddd mmm dd, yyyy") AS OldDate
FROM TestTable2;
```

```
|DateOfYear|         FullDate          |    OldDate     |
|01/26/2000|Wednesday January 26, 2000 |Tue Jan 26, 1700|
|01/27/2000|Thursday January 27, 2000  |Wed Jan 27, 1700|
|01/28/2000|Friday January 28, 2000    |Thu Jan 28, 1700|
|01/29/2000|Saturday January 29, 2000  |Fri Jan 29, 1700|
|01/30/2000|Sunday January 30, 2000    |Sat Jan 30, 1700|
|01/31/2000|Monday January 31, 2000    |Sun Jan 31, 1700|
|02/01/2000|Tuesday February 01, 2000  |Mon Feb 01, 1700|
|02/02/2000|Wednesday February 02, 2000|Tue Feb 02, 1700|
```

This next query combines the Switch() function with the IsNumeric() data inspection function and several of the Text manipulation functions (Len(), Left$(), Right$() and Ucase$()) to format some postal codes the way the USA and Canada want them. The first Switch() Boolean expression checks to see if the postal code is an expected length. USA expects 5 digits, a hyphen and 4 digits for a total of 10 characters. Canada expects 3 alphanumeric characters a space and 3 more alphanumeric characters, for a total of 7. The first result expression takes the input string as is but applies Ucase$() to assure that any letters in a Canadian postal code are uppercase. The second Boolean expression verifies that all characters are numeric and that the length is 9. Its resultant expression inserts a hyphen between the left and right portions if the zip code. The third Boolean expression identifies postal codes consisting of 5 numeric digits so that its resultant expression can append a hyphen and 4 zeros. The final Boolean expression inspects the input postal code to determine that it is not a number. We will assume that it has the correct alternating number and letter pattern for the Canadian postal code. Its length is also checked to be 6 characters. When the criteria in this Boolean expression are satisfied, a space is inserted between the left hand three characters and the right hand three characters. The Ucase$() function is used again to assure that all letters in the resultant are upper case. The query result table shows that the last input postal code met none of the Switch function criteria and was not formatted.

```
SELECT PostalCode,
SWITCH(LEN(PostalCode) IN(10,7),
UCASE$(PostalCode),
IsNumeric(PostalCode) AND LEN(PostalCode)=9,
LEFT$(PostalCode,5) & "-" & RIGHT$(PostalCode,4),
IsNumeric(PostalCode) AND LEN(PostalCode)=5,
PostalCode & "-0000",
NOT IsNumeric(PostalCode) AND LEN(PostalCode)=6,
UCASE$(LEFT$(PostalCode,3) & " " & RIGHT$(PostalCode,3)))
AS fmtPostalCode FROM TestZip1;
```

```
|   PostalCode    |    fmtPostalCode    |
| 80538-1200      | 80538-1200          |
| 80502           | 80502-0000          |
| 805032110       | 80503-2110          |
| k1a0b1          | K1A 0B1             |
| A1C 5X4         | A1C 5X4             |
| 123             |                     |
```

There are many more functions available for use in SQL. I will list all of the functions I can think of below. I will admit, I personally have had no use for the Math and Financial functions. I know they are there if I ever do need them. Writing this article has stretched my knowledge of Program Flow functions. I have used the IIF() function many times and now can see good uses for the Choose() function.

Many of the SQL Aggregate functions have been very useful to me. They are readily available in the query design window along with the Group By and Where summary operators and the First() and Last() functions. They are used to achieve a combined result from groups of records.

I hope you enjoy more newly discovered functionality in SQL!

Addendum List of SQL Built In functions with arguments:

Text
Asc («stringexpr»)
Chr («charcode»)
Chr$ («charcode»)
Format («expr»[, «fmt»[, «firstweekday»[, «firstweek»]]])
Format$ («expr»[, «fmt»[, «firstweekday»[, «firstweek»]]])
Instr ([«start»,] «stringexpr1», «stringexpr2»[, «compare»])
LCase («stringexpr»)
LCase$ («stringexpr»)
Left («stringexpr», «n»)
Left$ («stringexpr», «n»)
Len («stringexpr»)
LTrim («stringexpr»)
LTrim$ («stringexpr»)
Mid («stringexpr», «start»[, «length»])
Mid$ («stringexpr», «start»[, «length»])
Right («stringexpr», «n»)
Right$ («stringexpr», «n»)
RTrim («stringexpr»)
RTrim$ («stringexpr»)
Space («number»)
Space$ («number»)
StrComp («stringexpr1», «stringexpr2»[, «compare»])
String («number», «charcode»[, «stringexpr»])
String$ («number», «charcode»)
Trim («stringexpr»)
Trim$ («stringexpr»)
UCase («stringexpr»)
UCase$ («stringexpr»)

Math
Abs («number»)
Atn («number»)
Cos («angle») in Radians = Degrees*(Pi/180)
Exp («number»)
Fix («number»)
Int («number»)
Log («number») Base e
Rnd («number»)
Sgn («number»)
Sin («angle») in Radians = Degrees*(Pi/180)
Sqr («number»)
Tan («angle») in Radians = Degrees*(Pi/180)

Financial
DDB («cost», «salvage», «life», «period»)
FV («rate», «nper», «pmt»[, «pv»[, «due»]])
IPmt («rate», «period», «nper», «pv»[, «fv»[, «due»]])
IRR («ValueArray»[, «guess»])
MIRR («ValueArray», «FinanceRate», «ReinvestRate»)
NPer («rate», «pmt», «pv»[, «fv»[, «due»]])

NPV («rate», «valueArray»)
Pmt («rate», «nper», «pv»[, «fv»[, «due»]])
PPmt («rate», «period», «nper», «pv»[, «fv»[, «due»]])
PV («rate», «nper», «pmt»[, «fv»[, «due»]])
Rate («nper», «pmt», «pv»[, «fv»[, «due»[, «guess»]]])
SLN («cost», «salvage», «life»)
SYD («cost», «salvage», «life», «period»)

Inspection
IsDate («varexpr»)
IsEmpty («varexpr»)
IsNull («varexpr»)
IsNumeric («varexpr»)
VarType («varexpr»)

SQLAggregate
Avg («expr»)
Count («expr»)
Max («expr»)
Min («expr»)
StDev («expr»)
StDevP («expr»)
Sum («expr»)
Var («expr»)
VarP («expr»)

ProgramFlow
Choose («indexnum», «varexpr»)
IIf («expr», «truepart», «falsepart»)
Switch («varexpr1», «varexpr1», «varexpr2», «varexpr2»)

Conversion
Asc («stringexpr»)
CBool («expr»)
CCur («expr»)
CDate («date»)
CDbl («expr»)
Chr («charcode»)
Chr$ («charcode»)
CInt («expr»)
CLng («expr»)
CSng («expr»)
CStr («expr»)
CVar («expr»)
CVDate («expr»)
DateSerial («year», «month», «day»)
DateValue («stringexpr»)
Day («number»)
Hex («number»)
Hex$ («number»)
Hour («number»)
Minute («number»)
Month («number»)
Oct («number»)
Oct$ («number»)
Second («number»)
Str («number»)
Str$ («number»)
TimeSerial («hour», «minute», «second»)

TimeValue («stringexpr»)
Val («stringexpr»)
Weekday («number»)
Year («number»)

Date/Time
CDate («date»)
CVDate («expr»)
Date ()
Date$ ()
DateAdd («interval», «number», «date»)
DateDiff («interval», «date1», «date2»[, «firstweekday»[, «firstweek»]])
DatePart («interval», «date»[, «firstweekday»[, «firstweek»]])
DateSerial («year», «month», «day»)
DateValue («stringexpr»)
Day («number»)
Hour («number»)
IsDate («varexpr»)
Minute («number»)
Month («number»)
Now ()
Second («number»)
Time ()
Time$ ()
Timer ()
TimeSerial («hour», «minute», «second»)
TimeValue («stringexpr»)
Weekday («number»)
Year («number»)

The DateAdd(), DateDiff() & DatePart() interval argument has these settings:

Setting	Description
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week
h	Hour
n	Minute
s	Second