Access Answers: An average column: I mean, what mode is your median?

Doug Steele

Doug tries to address commonly asked questions from Access developers. This month, he looks at different ways to calculate "measures of central tendency": mean, median and mode.

How can I calculate the Median for my data?

Before launching into a solution for this question, let's go over some terminology. You may have noticed the phrase "measures of central tendency" used in the introduction above. This is a term used in Statistics. One of the best ways to summarize a set of data and still retain part of the information is to represent the set with a single value. Measures of Central Tendency are ways of calculating a number that is representative of an entire list of numbers. There are 3 commonly used measures of central tendency:

- Mean: The arithmetic average of a set of numbers (the most common measure of central tendency)
- Median: The value of the middle item when the data are arranged from lowest to highest (assuming an odd number of observations) or the average value of the two middle items when there's an even number of observations)
- Mode: The observation that occurs most frequently in a data set.

Most of you are probably familiar with the concept of mean (or which is often referred to as "arithmetic average"), but the other two measures might not be as familiar to you, so an example might be in order.

Let's assume you throw 3 dice a total of 12 times, and get the following results:

Roll	Total
1	14
2	13
3	8
4	8
5	12
6	8
7	10
8	9
9	5
10	3
11	17
12	10

Table 1: DiceRolls

It's easy to calculate the mean: that's (14+13+8+8+12+8+10+9+5+3+17+10)/12 = 9.75

To find the median, arrange the 12 totals in order (it doesn't really matter whether you arrange them in ascending or descending order, although it's more common to use ascending order): 3, 5, 8, 8, 8, 9, 10, 10, 12, 13, 14, 17. Since there's an even number of samples, the median will be the average value of the sixth and seventh elements. The sixth element is 8, the seventh element is 11, therefore the median is (9+10)/2 = 9.5

Finally, the mode is the value that occurs most often. With the elements arranged in order in the paragraph above, it's fairly straight-forward to see that 8 occurs more times than any other value, so the mode is 8.

Okay, so how can we calculate these values in Access?

Access already has a function, DAvg, that will compute the mean of a set of values in a specified set of records (a domain). To calculate the mean of our DiceRolls table above, the code would be:

```
?DAvg("Total", "DiceRolls")
 9.75
```

Unfortunately, Access does not have similar functions to compute Median and Mode, so we'll have to create our own. To be consistent with DAvg (and other Domain functions), I'll call the function DMedian, and name the arguments Expr, Domain and Criteria:

```
Function DMedian( _
    Expr As String, _
    Domain As String, _
    Optional Criteria As String = "" _
) As Variant
```

Expr is an expression that identifies the field containing the numeric data for which you want the median. It can be a string expression identifying a field in a table or query, or it can be an expression that performs a calculation on data in that field. Domain is a string expression identifying the set of records that constitutes the domain (a table name or a query name). Criteria is an optional string expression used to restrict the range of data on which the DMedian function is performed. Criteria is equivalent to the WHERE clause in an SQL expression, without the word WHERE. If criteria is omitted, the DMedian function evaluates Expr against the entire domain. Note that any field that is included in criteria must also be a field in Domain:

```
Dim dbMedian As DAO.Database
Dim rsMedian As DAO.Recordset
Dim dblTemp1 As Double
Dim dblTemp2 As Double
Dim lngOffset As Long
Dim lngRecCount As Long
Dim strSQL As String
Dim varMedian As Variant
```

Create a SQL string that will return Expr sorted. If a value was supplied for Criteria, include it in the SQL statement. Note that Null values will be ignored. This is consistent with how DAvg works.

```
  strSQL = "SELECT " & Expr & " AS Data " & _
    "FROM " & Domain & " "
  strSQL = strSQL & _
    "WHERE " & Expr & " IS NOT NULL "
  If Len(Criteria) > 0 Then
    strSQL = strSQL & "AND (" & Criteria & ") "
  End If
  strSQL = strSQL & "ORDER BY " & Expr
```

Instantiate a recordset, using the SQL Statement created above, to return all of the relevant data from the domain:

```
  Set dbMedian = CurrentDb()
  Set rsMedian = dbMedian.OpenRecordset(strSQL)
```

Make sure that records were actually returned.

```
  If rsMedian.BOF = False And _
    rsMedian.EOF = False Then
```

Assuming records were returned, determine how many. To do this, move to the end of the recordset so that RecordCount will return an accurate count. If there are an odd number of records (lngRecCount Mod 2 <> 0), determine how many elements backwards to move to reach the midpoint of the recordset, and the Median will be that element.

```
    rsMedian.MoveLast
    lngRecCount = rsMedian.RecordCount
    If lngRecCount Mod 2 <> 0 Then
      lngOffset = ((lngRecCount + 1) / 2) - 2
      If lngOffset >= 0 Then
        rsMedian.Move -lngOffset - 1
      End If
      varMedian = rsMedian("DataValue")
    Else
```

If there are an even number of records, move backwards to the element after the midpoint of the recordset and retrieve that value. Move backwards once more to the element before the midpoint and retrieve that

value. Compute the mean of the two values retrieved, and that will be the Median (if there are only 2 elements to begin with, you'll simply compute the mean of those two elements):

```
      lngOffset = (lngRecCount / 2) - 2
      If lngOffset >= 0 Then
        rsMedian.Move -lngOffset - 1
      End If
      dblTemp1 = rsMedian("DataValue")
      rsMedian.MovePrevious
      dblTemp2 = rsMedian("DataValue")
      varMedian = (dblTemp1 + dblTemp2) / 2
    End If
  Else
```

If no records were returned, the Median will be Null:

```
    varMedian = Null
  End If
```

Clean up after yourself, and the function's complete:

```
  rsMedian.Close
  Set rsMedian = Nothing
  Set dbMedian = Nothing

  DMedian = varMedian

End Function
```

You'd use this function in the same method as DAvg:

```
?DMedian("Total", "DiceRolls")
 9.5
```

Okay, can I determine the Mode values of my data as well?

Determining the Mode is complicated by the fact that it's possible for more than one value to be the mode. Therefore, the DMode function needs to be able to return an array. (The definitions for Expr, Domain and Criteria are the same as above):

```
Function DMode( _
  Expr As String, _
  Domain As String, _
  Optional Criteria As String = "" _
) As Variant

Dim dbMode As DAO.Database
Dim rsMode As DAO.Recordset
Dim lngLoop As Long
Dim lngMaxFreq As Long
Dim strSQL As String
Dim varMode As Variant
```

As before, a SQL statement must be created. This time, though, the SQL statement doesn't simply return all of the qualifying values in the domain. Instead, it's going to be an Aggregate query that returns each unique value in the domain, plus how many times that value occurs. Further, it's going to return the values in descending order of occurrence. In other words, it'll return the value that occurs the most times, followed by the value that occurs the second most times and so on until the value that occurs the fewest times.

```
  strSQL = "SELECT [" & FieldName & "], " & _
    "Count(*) AS Frequency " & _
    "FROM [" & TableName & "] "
  If Len(WhereClause) > 0 Then
    strSQL = strSQL & _
      "WHERE " & WhereClause & " "
  End If
  strSQL = strSQL & _
    "GROUP BY [" & FieldName & "] "
  strSQL = strSQL & "ORDER BY 2 DESC, 1 ASC"
```

Instantiate a recordset, using the SQL Statement created above, to return all of the relevant data from the domain:

```
  Set dbMode = CurrentDb()
  Set rsMode = dbMode.OpenRecordset(strSQL)
```

Make sure that records were actually returned.

```
  If rsMode.BOF = False And _
    rsMode.EOF = False Then
```

Assuming records were returned, determine how many occurrences there were for the value that occurred the most number of times and save that value in a variable lngMaxFreq. Loop through the recordset until a value with fewer occurrences is encountered. For each value that occurs the same number of times as what's stored in lngMaxFreq, add the value to an array. The contents of that array will represent the Mode value(s) for the domain:

```
    varMode = Array()
    lngLoop = 0
    lngMaxFreq = rsMode("Frequency")
    Do While rsMode("Frequency") = lngMaxFreq
      ReDim Preserve varMode(0 To lngLoop)
      varMode(lngLoop) = rsMode(FieldName)
      lngLoop = lngLoop + 1
      rsMode.MoveNext
    Loop
  Else
    varMode = Null
  End If
```

Clean up after yourself, and the function's done:

```
  rsMode.Close
  Set rsMode = Nothing
  Set dbMode = Nothing
  DMode = varMode

End Function
```

Unfortunately, it's not quite as easy to use this DMode function as it is to use the other Domain functions.

If you were to type

```
?DMode("Total", "DiceRolls")
```

you'd get a Run Time error 13 (Type Mismatch).

Instead, use code like the following:

```
Sub DetermineMode( _
  Expr As String, _
  Domain As String, _
  Optional Criteria As String = "" _
)

On Error GoTo Err_DetermineMode

Dim lngCount As Long
Dim lngLoop As Long
Dim strField As String
Dim strTable As String
Dim varMode As Variant

  varMode = DMode(Expr, Domain, Criteria)
  If IsNull(varMode) Then
    Debug.Print "No Mode found"
  Else
    lngCount = UBound(varMode) - _
      LBound(varMode) + 1
```

```
    If lngCount = 1 Then
      Debug.Print "One Mode value found:"
    Else
      Debug.Print lngCount & _
        " Mode values found:"
    End If
    For lngLoop = LBound(varMode) To _
      UBound(varMode)
      Debug.Print _
        (lngLoop - LBound(varMode) + 1) & _
        ": " & varMode(lngLoop)
    Next lngLoop
  End If

End Sub
```

Using this routine, you'll see something like:

```
Call DetermineMode("Total", "DiceRolls")
One Mode value found:
1: 8
```

What happens, though, if our sample data is slightly different?

Roll	Total
1	14
2	14
3	8
4	8
5	11
6	8
7	10
8	9
9	6
10	5
11	14
12	10

Table 2: AlternateDiceRolls

Now, it turns out that the Mean and Median for this data is exactly the same as before:

```
?DAvg("Total", "AlternateDiceRolls")
 9.75

DMedian("Total", "AlternateDiceRolls")
 9.5
```

However, now we have two different values that are both Modes for our data:

```
?DetermineMode("Total", "AlternateDiceRolls")
2 Mode values found:
1: 8
2: 14
```

What if my numbers aren't in a table and I want to compute the median?

VBA allows you to use the keyword ParamArray as the last argument in the list of arguments for a function or subroutine to indicate that the final argument is an Optional array of Variant elements. This means that you can pass an arbitrary number of values to the routine, and you can treat that list of values as a single array.

This means that it's possible to declare a function as

```
Function Median( _
  ParamArray DataPoints() As Variant _
) As Variant
```

then call the function as Median(3, 6, 1, 2, 4) and have it return a value.

Within the function, you have an array DataPoints that you need to sort in ascending order, then find the middle position to determine the median.

There are a couple of "gotchas". You cannot pass the ParamArray array to another routine, so it's not just a simple matter of calling a sort routine to arrange the numbers into ascending order.

A second issue, though, means that passing the array to a sort routine probably wouldn't be a good idea anyhow. Since you can pass anything to the array, it's necessary to validate all of the values before you try to compute the median. How would you determine Median("red", "green", "blue")?

One way to solve both issues is to create a new array within the routine, only transferring valid (i.e. numeric) arguments into the new array. Assuming that at least one numeric value ends up in this new array, sort the new array and compute the median.

While I don't intend to discuss the routine I used for doing the sort (there are plenty of comments in the code), something like the following will let you compute the median for an arbitrary number of values:

```
Function Median( _
  ParamArray DataPoints() As Variant _
) As Variant

Dim lngArraySize As Long
Dim lngCurrPos As Long
Dim lngLoop As Long
Dim lngPos1 As Long
Dim lngPos2 As Long
Dim varValues() As Variant

  If IsMissing(DataPoints) = True Then
    Median = Null
  Else
    ReDim varValues(LBound(DataPoints) To _
      UBound(DataPoints))
    lngCurrPos = LBound(DataPoints) - 1
    For lngLoop = LBound(DataPoints) To _
      UBound(DataPoints)
      If IsNull(DataPoints(lngLoop)) = False Then
        If IsNumeric(DataPoints(lngLoop)) Then
          lngCurrPos = lngCurrPos + 1
          varValues(lngCurrPos) = _
            DataPoints(lngLoop)
        End If
      End If
    Next lngLoop
    If lngCurrPos >= 0 Then
      ReDim Preserve varValues( _
        LBound(DataPoints) To lngCurrPos)

      Call QuickSortVariants(varValues, _
        LBound(varValues), UBound(varValues))

      lngArraySize = UBound(varValues) - _
        LBound(varValues) + 1
      If lngArraySize Mod 2 = 0 Then
        lngPos1 = lngArraySize / 2 - 1
        lngPos2 = lngPos1 + 1
        Median = (varValues(lngPos1) + _
          varValues(lngPos2)) / 2
      Else
        lngPos1 = (lngArraySize - 1) / 2
        Median = varValues(lngPos1)
      End If
    Else
      Median = Null
    End If
  End If

End Function
```

Plugging our original values into this function, we get:

```
?Median(14, 13, 8, 8, 12, 8, 10, 9, 5, 3, 17, 10)
 9.5
```

I'm not sure there's really any reason for such a function: as far as I'm concerned, requiring the ability to calculate the median like this is likely an indication that your tables haven't been properly normalized. However, now you have a function if you need it.

Doug Steele has worked with databases, both mainframe and PC, for many years. Microsoft has recognized him as an Access MVP for his contributions to the Microsoft-sponsored newsgroups. Check http://I.Am/DougSteele for some Access information, as well as Access-related links. You can reach him at AccessHelp@rogers.com, but note that personal replies are not guaranteed. However, ***please*** *don't hesitate to send ideas for future columns or, even better, complete columns!*