September, 2003

Advisor Answers

Computing statistics from a table

VFP 8/7/6

Q: I need to calculate statistics like average and median based on numeric data in a table. Computing the average of a field seems easy enough, but how do you get the median?

–Name withheld by request

A: I usually think of average and median as part of a group of three items; the third is mode. Let's start by looking at what each represents and then consider how to compute them in VFP.

Average, also known as mean, is the most familiar of the three. It's the value you get by adding up a list of numbers and dividing by the count. While the mean can give you a quick feel for the values in the list, it's easy for one or two values that are much larger or much smaller than the rest to cause misleading results. (Think back to school where a 0 on one assignment has a disproportionate effect on an overall grade.)

Median offers a more accurate picture. It's the middle value in the list. That is, to find the median, you put all the values in numerical order and grab the one in the middle. If the list has an even number of values, the median is the average of the two in the middle.

Mode is a fancy way of saying "most common." That is, the mode of a set of numbers is the value that occurs most frequently.

I created a class that allows you to specify a table and field and has methods to compute the mean, median, and mode of the specified field. The results are stored in properties of the class.

VFP offers three built-in ways of computing the average of a field or expression: the AVERAGE command, the CALCULATE command and the SELECT-SQL statement. AVERAGE accepts a list of fields or expressions and computes the average of each. CALCULATE accepts a list of expressions that involve any of several functions, including AVG(), and computes the specified results. Both AVERAGE and CALCULATE can use FOR and WHILE clauses to limit the records they consider, and both can save results to variables using the TO clause.

Both ignore null values. In fact, when computing averages, the only real distinction is that CALCULATE accepts an IN clause to indicate what table to work on, while AVERAGE always works on the table open in the current work area.

The third way to compute averages uses the SQL SELECT command with the aggregate function, AVG(). It differs from the others in that it opens the specified table again and thus, doesn't affect the record pointer if the table's already open. In addition, SQL SELECT looks at the actual data on disk, and doesn't see any buffered changes.

Because I didn't want to worry about work areas, I chose CALCULATE for my ComputeMean method, but any of the three choices can do the job. Here's the code:

```
PROCEDURE ComputeMean
* Compute the average value for the specified field
CALCULATE AVG(EVALUATE(This.cField)) ;
   IN (This.cAlias) ;
   TO This.nMean
RETURN This.nMean
ENDPROC
```

Computing the median is more complicated. There are no commands to do it in one step. However, it only takes a few lines of code. The basic idea is to put the records in order and then either grab the middle value or average the two middle values. Here's my code for ComputeMedian:

```
PROCEDURE ComputeMedian
* Compute the median for the specified field
LOCAL cFieldName, nCount, nLowValue, nHighValue
cFieldName = This.cField
SELECT &cFieldName as nDataField ;
   FROM (This.cTable) ;
   ORDER BY 1 ;
   INTO CURSOR InOrder NOFILTER

nCount = _TALLY
DO CASE
CASE _TALLY == 0
   This.nMedian  = 0
CASE MOD(nCount, 2) = 1
   * Odd number of records--go right to middle
   GO CEILING(nCount/2)
   This.nMedian = InOrder.nDataField
OTHERWISE
   * Even number--average two middle records
   GO nCount/2
   nLowValue = InOrder.nDataField
   SKIP 1
```

```
   nHighValue = InOrder.nDataField
   This.nMedian = (nLowValue + nHighValue)/2
ENDCASE
USE IN InOrder
RETURN This.nMedian
```

Be aware that because this method uses SQL SELECT, it doesn't see any buffered changes. The only other interesting thing in this code is the CEILING() function, not one I use every day. CEILING() returns the next integer greater than or equal to the number you pass it.

Like median, there's no built-in way to find the most common value for a field, but it's not terribly hard to do. Here's my code for ComputeMode:

```
PROCEDURE ComputeMode
* Find the most common value for the specified field
LOCAL cFieldName, nMaxCount
cFieldName = This.cField
SELECT TOP 1 &cFieldName as nDataField, ;
             CNT(&cFieldName) as nCount ;
   FROM (This.cTable) ;
   GROUP BY 1 ;
   ORDER BY 2 DESC ;
   INTO CURSOR GetMode
nMaxCount = GetMode.nCount

* Copy results into array. Deal with possibility of a tie
This.nModeCount = 0
DIMENSION This.aMode[1]
SCAN
   This.nModeCount = This.nModeCount + 1
   DIMENSION This.aMode[ This.nModeCount ]
   This.aMode[ This.nModeCount ] = GetMode.nDataField
ENDSCAN
USE IN GetMode
RETURN nMaxCount
ENDPROC
```

Like ComputeMedian, this method uses a SQL SELECT. In this case, a different approach would take a lot more code.

The TOP 1 clause of the query says to return only the first record. Since the query sorts the results into descending frequency of values, that's exactly what we want. If several values of the specified field share the top frequency, the cursor contains several records. My object has an array property to hold all the values found most frequently and a numeric property, nModeCount, to indicate how many elements are in the array. Note that while ComputeMean and

ComputeMedian return the computed value, ComputeMode returns the frequency of the most common value.

You'll find the complete definition for the TableStatistics object on this month's Professional Resource CD. You might want to extend the class to handle a few other common statistical functions. CALCULATE supports standard deviations and variances directly, so it would be easy to incorporate those.

–Tamar