Sending crosstabs to Excel

In many cases, users need the ability to do more with data once it's been cross-tabbed. There are a number of ways to get it there, including creating Excel Pivot Tables.

Tamar E. Granor, Ph.D.

In my last article, I looked at how to create reports with crosstab and pivot data. This time, I look at how to send the results to Excel.

Crosstab and pivot data is challenging to report on because it may have many columns and we often won't know when we write the code how many columns we're dealing with. Exporting such data to Excel can be a great solution because it can handle lots of columns, and the various export techniques don't generally require you to know how many columns you have ahead of time. In addition, sending data to Excel gives users the change to do additional crunching on it as needed.

We can export the crosstabbed data as is or we can send raw data and use Excel's pivot table capabilities to crosstab it there. We'll look at both options.

We'll work with the example in Listing 1, which collects sales by employee by month.

Listing 1. This code produces a cursor with one row for each employee for each month in which the employee had any sales.

```
SELECT EmployeeID, YEAR(OrderDate) AS Year, ;
       MONTH(OrderDate) as Month, ;
       SUM(Quantity*UnitPrice) AS OrderTotal ;
   FROM Orders ;
     JOIN OrderDetails ;
       ON Orders.OrderID = ;
          OrderDetails.OrderID ;
   GROUP BY 1, 2, 3 ;
   INTO CURSOR csrMonthlyTotals

* Add employee name
SELECT PADR(ALLTRIM(FirstName) + (' ' + ;
            LastName), 30) AS cName, ;
       csrMonthlyTotals.* ;
   FROM csrMonthlyTotals ;
     JOIN Employees ;
       ON csrMonthlyTotals.EmployeeID = ;
          Employees.EmployeeID ;
   ORDER BY LastName, FirstName ;
   INTO CURSOR csrReport
```

Sending cross-tabbed data to Excel

The simpler solution is to take the crosstab or pivot results and just send them to Excel. Depending how much control you want, there are a variety of ways to do this.

The code in Listing 2 gives us the crosstabbed results, with one row per employee and one column per month. (Make sure fastxtab.prg is in your path or add a path in the code.) The combined code from these two listings is included in this month's downloads as CrossTabSalesPersonMonthly.prg.

Listing 2. This code creates a crosstab from the results of the query in Listing 1.

```
LOCAL oXTab AS FastXTab OF "fastxtab.prg"

oXTab = NEWOBJECT("fastxtab", "fastxtab.prg")
WITH oXTab AS FastXTab OF "fastxtab.prg"
   .cOutFile = "csrXtab"
   .cRowFIELD = "cName"
   .cColField = ;
    [PADL(OrdYear,4) + "-" + PADL(OrdMonth,2)]
   .cDATAFIELD = "OrderTotal"
   .lCursorOnly = .T.
   .lCLOSETABLE = .T.
   .RunXtab()
ENDWITH
```

The most basic way to send this data to Excel is the COPY TO command. There are two output options, Excel data or CSV (comma-separated value). The only difference in the COPY TO command is the keyword you place after TYPE. To create an Excel file, use XL5; for a CSV, use CSV.

COPY TO actually supports two different Excel types, XLS and XL5. Both are somewhat outdated. XLS is the file format from Excel 2.0. Do not use it; it has two important limitations. First, dates aren't handled property. Second, it's limited to 16,384 rows. (Obviously, this is not an issue for our example, but might be in other cases.) As this discussion implies, COPY TO has no way to create the more recent XLSX file; see the next section for options for doing that.

TYPE XL5 handles dates properly, but is limited to 32,767 rows. That's one reason you may prefer TYPE CSV, which doesn't have that limit and is treated by Excel as if it were a native type.

Listing 3 shows the code to export to TYPE XL5; in this month's downloads the code is included as ExportToXL5SalesPersonMonthly.prg. Figure 1 shows partial results before doing any formatting in Excel.

Listing 3. This code exports the crosstab results to Excel directly.

```
LOCAL cXLSFile
cXLSFile =  FORCEPATH(FORCEEXT( ;
  "SalesPersonMonthly", "XLS"), SYS(2023))

COPY TO (m.cXLSFile) TYPE XL5
```

	A	B	C	D	E	F
1	cname	c_1996_7	c_1996_8	c_1996_9	c_1996_1(	c_1996_1
2	Andrew Fu	1176	1814	2950.8	5725.7	4759
3	Anne Dods	4955.3	0	0	6244.4	0
4	Janet Leve	2998.2	3557.2	1762	4317.6	3838
5	Laura Call	1726	8485.8	5248	385.2	704.8
6	Margaret F	12988.9	3670.5	3575.1	14422.1	12017.4
7	Michael Su	2587.9	2595.4	4465.6	0	2299.8
8	Nancy Dav	2018.6	6007.1	6883.7	4061.4	10261.2
9	Robert Kin	0	479.4	1330.8	4577.2	11717.4
10	Steven Bu	1741.2	0	1420	1470	4106.4

Figure 1. When you export with COPY TO, the data is simply dumped into an Excel file with no formatting.

The code to export to CSV is nearly identical; it's included in this month's downloads as ExportToCSVSalespersonMonthly.prg and shown in Listing 4. When you open the file in Excel, before doing any formatting, it looks almost identical to Figure 1. (The only difference I see is in the font used. On my Windows 7 computer, the XLS file uses 10-point Arial, while the CSV uses 11-point Calibri, my default for Excel.)

Listing 4. Exporting to a CSV file is another way to get data into Excel.

```
LOCAL cCSVFile
cCSVFile =  FORCEPATH(FORCEEXT( ;
  "SalesPersonMonthly", "CSV"), SYS(2023))

COPY TO (m.cCSVFile) TYPE CSV
```

If you need to format the spreadsheet before passing it along to users, you can use Automation to open it in Excel and do the formatting.

Creating XLSX files

Given that the XLSX format first appeared in Office 2007, your users may want that rather than the older XLS or CSV format. One easy way to create an XLSX is to generate an XLS or CSV file with COPY TO, and then use Automation to save it in a newer format. The code in Listing 5 (included in this month's downloads as ExportToXLSXSalespersonMonthly.prg) does that, with a few safeguards in case Excel or the workbook can't be opened.

Listing 5. This code exports data to Excel using COPY TO and then opens the XLS file and saves it in the XLSX format.

```
LOCAL cXLSFile, cXLSXFile
LOCAL oXL, oWorkbook, lSuccess

cXLSFile =  FORCEPATH(FORCEEXT( ;
   "SalesPersonMonthly", "XLS"), SYS(2023))
cXLSXFile = FORCEEXT(m.cXLSFile, "XLSX")

COPY TO (m.cXLSFile) TYPE XL5

TRY
   oXL = CREATEOBJECT("Excel.Application")
   lSuccess = .T.
CATCH
   MESSAGEBOX("Unable to open Excel.")
   lSuccess = .F.
ENDTRY

IF m.lSuccess
   TRY
      oWorkbook = ;
        oXL.Workbooks.Open(m.cXLSFile)
      lSuccess = .T.
   CATCH
      MESSAGEBOX("Unable to open workbook " +;
        "in Excel for conversion to XLSX.")
      lSuccess = .F.
      oXL.Quit()
   ENDTRY
ENDIF

IF m.lSuccess
   TRY
      oWorkbook.SaveAs(m.cXLSXFile, 51)
         && 51 = xlOpenXMLWorkbook
   CATCH
      MESSAGEBOX("Unable to convert " + :
        "workbook to XLSX.")
   ENDTRY
   oXL.Quit()
ENDIF
```

As before, you still need to format the workbook afterward; the result looks the same as Figure 1.

In addition, this approach requires Excel 2007 or later to be installed on the machine where the code is running. The VFP world being what it is, there are several open source projects that allow you to create XLSX files without Excel. I'll briefly cover two of them here.

XLSX Workbook

XLSX Workbook comes from Greg Green, who has created a number of utilities for VFP developers. This one is a class that lets you not only export data to Excel, but format it as well. Even better, it has extensive documentation. You can download the latest version from https://github.com/ggreen86/XLXS-Workbook-Class.

Exporting a table or cursor takes just a couple of lines of code, as in Listing 6. First, you instantiate the VFPxWorkbookXLSX class; be sure to either put the class library in your path, or to add the path to the NewObject() call. Then, call the SaveTableToWorkbookEx method, passing the table name or alias to export and the name of the file you want to create. The code is included as ExportXLSXWorkbookSalespersonMonthly.prg in this month's downloads. Partial results are shown in Figure 2.

Listing 6. Exporting a table or cursor to Excel with XLSX Workbook is simple.

```
LOCAL cXLSFile, oToExcel, lReturn
cXLSXFile =  FORCEPATH(FORCEEXT( ;
  "SalesPersonMonthly", "XLSX"), SYS(2023))

oToExcel = NEWOBJECT("VFPxWorkbookXLSX", ;
                     "VFPxWorkbookXLSX.vcx")
lReturn = oToExcel.SaveTableToWorkbookEx( ;
   "csrXTab", m.cXLSXFile)

IF NOT m.lReturn
   MESSAGEBOX("Unable to create workbook")
ENDIF
```

	A	B	C	D	E	F
1	CNAME	C_1996_7	C_1996_8	C_1996_9	C_1996_1(	C_1996_1:
2	Andrew Fu	1176	1814	2950.8	5725.7	4759
3	Anne Dods	4955.3	0	0	6244.4	0
4	Janet Leve	2998.2	3557.2	1762	4317.6	3838
5	Laura Calla	1726	8485.8	5248	385.2	704.8
6	Margaret F	12988.9	3670.5	3575.1	14422.1	12017.4
7	Michael Su	2587.9	2595.4	4465.6	0	2299.8
8	Nancy Dav	2018.6	6007.1	6883.7	4061.4	10261.2
9	Robert Kin	0	479.4	1330.8	4577.2	11717.4
10	Steven Buc	1741.2	0	1420	1470	4106.4

Figure 2. XSLX Workbook can export directly from a table or cursor to XSLX. By default, it does no formatting.

The class has many additional methods, including the ability to save a grid to a workbook, maintaining its formatting.

DBF2XLSX

This tool comes from Vilhelm-Ion Praisach (whose extensions to FastXtab I discussed in the May, 2016 issue). The download, which is linked in Praisach's blog at http://praisachion.blogspot.com/2017/04/export-dbf-to-excel-2007.html, includes the basic export, as well as a class to provide UI for exporting. He also provides VFP 6-compatible versions. (In addition, elsewhere in his blog, Praisach offers a tool for importing XLSX files and tools for exporting to other Office formats.)

Using DBF2XLSX is even easier than using XLSX Workbook. Listing 7 shows the necessary code. A single line of code does the export; Figure 3 shows partial results. The code is included as ExportDBF2XLSXSalespersonMonthly.prg in this month's downloads.

Listing 7. With DBF2XLSX, one line of code exports a table or cursor to a modern Excel workbook.

```
LOCAL cXLSFile, oToExcel, lReturn
cXLSXFile =  FORCEPATH(FORCEEXT( ;
  "SalesPersonMonthly", "XLSX"), SYS(2023))

DO CopyToXLSX WITH 'csrXTab', ;
   m.cXLSXFile, .T.
```

	A	B	C	D	E	F
1	CNAME	C_1996_7	C_1996_8	C_1996_9	C_1996_1(	C_1996_1:
2	Andrew Fu	$1,176.00	$1,814.00	$2,950.80	$5,725.70	$4,759.00
3	Anne Dods	$4,955.30	$0.00	$0.00	$6,244.40	$0.00
4	Janet Leve	$2,998.20	$3,557.20	$1,762.00	$4,317.60	$3,838.00
5	Laura Calla	$1,726.00	$8,485.80	$5,248.00	$385.20	$704.80
6	Margaret F	########	$3,670.50	$3,575.10	########	########
7	Michael Su	$2,587.90	$2,595.40	$4,465.60	$0.00	$2,299.80
8	Nancy Dav	$2,018.60	$6,007.10	$6,883.70	$4,061.40	########
9	Robert Kin	$0.00	$479.40	$1,330.80	$4,577.20	########
10	Steven Buc	$1,741.20	$0.00	$1,420.00	$1,470.00	$4,106.40

Figure 3. The output from CopyToXLSX needs some formatting.

Only the first two parameters are required, providing the table or cursor to be copied and the name of the result file. The third parameter indicates whether the first row of the result should contain the column names. Additional parameters let you indicate which columns to include, as well as what to do with memo fields.

The downloads don't include documentation other than some test code, but a number of posts on Praisach's blog discuss how to use this routine and the others. Several posts indicate that the class ExportXLSX has the ability to export a grid, not just a table or cursor.

Other options

There are several other projects floating around the VFP world that have the capability of creating XLSX files.

FoxyXLS was created by Cesar Chalom, the creator of FoxCharts and FoxyPreviewer. Unlike the tools above, it doesn't provide a one-line way to convert, but it doesn't take too much code. Rick Schummer wrote about it in detail in the March, 2014 issue.

Çetin Basoz's VFP2Excel is a simple procedure that copies VFP data to Excel using ADO. Unlike the other tools described here, it expects Excel to be present and running. One of its parameters is a reference to the range in Excel where the data should be placed.

Creating pivot tables

All of the previous solutions simply take the generated crosstab and dump it into a spreadsheet. But Excel has some very nice capabilities relating to crosstabs and pivots. You can give users the ability to slice and dice the data in different ways, including filtering based on the values provided and collapsing whole sections.

Figure 4 shows (part of) the result we'll be working toward in this section. The whole spreadsheet is included in this month's downloads as PivotedSalesPersonMonthly.xlsx.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1															
2															
3	**Orders ($)**	**Months/Years**													
4		**⊟1996**						**1996 Total**	**⊟1997**						
5	**Salesperson**	**7**	**8**	**9**	**10**	**11**	**12**		**1**	**2**	**3**	**4**	**5**	**6**	**7**
6	Andrew Fuller	$1,176	$1,814	$2,951	$5,726	$4,759	$6,409	$22,835	$3,150	$1,584	$2,905	$14,019	$4,590	$7,059	$10,320
7	Anne Dodsworth	$4,955			$6,244		$166	$11,366	$1,209		$1,771	$611	$140	$3,762	$28
8	Janet Leverling	$2,998	$3,557	$1,762	$4,318	$3,838	$2,759	$19,232	$7,478	$10,581	$11,599	$10,297	$18,640	$5,872	$1,109
9	Laura Callahan	$1,726	$8,486	$5,248	$385	$705	$6,612	$23,161	$6,701	$7,764	$4,806	$801	$4,793	$2,616	$3,984
10	Margaret Peacock	$12,989	$3,671	$3,575	$14,422	$12,017	$6,441	$53,115	$25,620	$13,530	$5,645	$14,333	$7,573	$4,232	$6,105
11	Michael Suyama	$2,588	$2,595	$4,466		$2,300	$5,782	$17,731	$1,500	$1,352	$1,258	$9,691	$752	$4,228	$1,301
12	Nancy Davolio	$2,019	$6,007	$6,884	$4,061	$10,261	$9,557	$38,789	$7,332	$2,505	$5,494	$240	$9,168	$6,113	$19,998
13	Robert King		$479	$1,331	$4,577	$11,717		$18,105	$13,703	$3,891	$3,867	$5,707	$6,041	$2,082	$6,145
14	Steven Buchanan	$1,741		$1,420	$1,470	$4,106	$13,228	$21,965			$2,634		$5,128	$3,125	$6,475
15	**Grand Total**	**$30,192**	**$26,609**	**$27,636**	**$41,204**	**$49,704**	**$50,953**	**$226,299**	**$66,693**	**$41,207**	**$39,980**	**$55,699**	**$56,824**	**$39,088**	**$55,465**

Figure 4. Excel's pivot tables present crosstab data in a way that lets users explore it.

Introducing pivot tables

Excel has included pivot tables for a long time; the functionality was first introduced in Excel 5.0, back in 1994. The Pivot Table wizard, introduced in Excel 97, made it easy to create pivot tables.

As Figure 4 shows, pivot tables can let you use multiple criteria for pivoting the data. Here, the rows are based on salesperson, while the columns are based on both month and year. When you use multiple criteria, pivot tables let you collapse and expand; Figure 5 shows the same worksheet with the 1996 data collapsed to a single column.

	A	B	C	D	E	F
1						
2						
3	**Orders ($)**	**Months/Years**				
4		**⊞1996**	**⊟1997**			
5	**Salesperson**		**1**	**2**	**3**	**4**
6	Andrew Fuller	$22,835	$3,150	$1,584	$2,905	$14,019
7	Anne Dodsworth	$11,366	$1,209		$1,771	$611
8	Janet Leverling	$19,232	$7,478	$10,581	$11,599	$10,297
9	Laura Callahan	$23,161	$6,701	$7,764	$4,806	$801
10	Margaret Peacock	$53,115	$25,620	$13,530	$5,645	$14,333
11	Michael Suyama	$17,731	$1,500	$1,352	$1,258	$9,691
12	Nancy Davolio	$38,789	$7,332	$2,505	$5,494	$240
13	Robert King	$18,105	$13,703	$3,891	$3,867	$5,707
14	Steven Buchanan	$21,965			$2,634	
15	**Grand Total**	**$226,299**	**$66,693**	**$41,207**	**$39,980**	**$55,699**

Figure 5. When you pivot on multiple criteria, Excel's pivot tables let you collapse and expand on the higher-level values. Here, the 1996 data has been collapsed to a single column.

Excel's pivot tables also let you filter and sort, based on either row or column values. Figure 6 shows the dropdown that appears when you click the arrow on the Salesperson header.

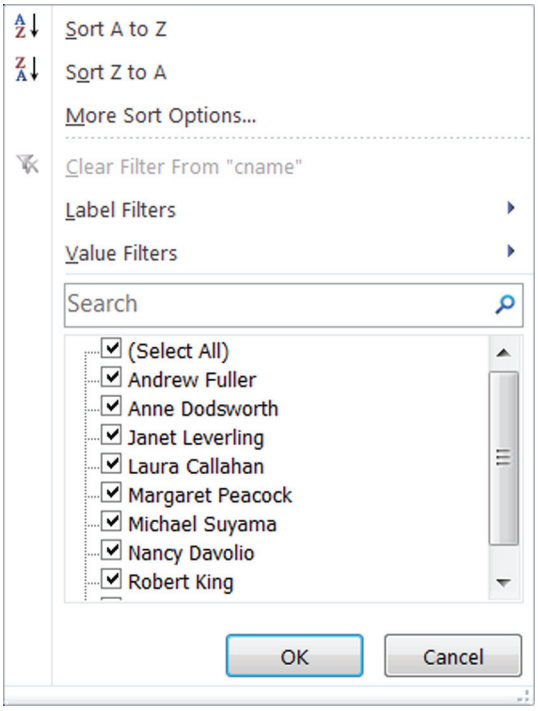

Figure 6. Excel's pivot tables let you filter and sort, based on row or column values. This window opens when you click the dropdown arrow on the Salesperson header.

Note that you can filter on either the label or the value. That is, you could select only those salespeople in a certain part of the alphabet (which seems silly), or you could select only those with total sales (in the Grand Total column at the very right edge of the pivot table) in a certain range. Figure 7 shows the pivot table with all three years collapsed and with a filter of total sales greater than or equal to $100,000. Note the icon on the Salesperson header that lets you know you're seeing filtered data.

	A	B	C	D	E
1					
2					
3	**Orders ($)**	**Months/Years**			
4		**⊞1996**	**⊞1997**	**⊞1998**	**Grand Total**
5	**Salesperson**				
6	Andrew Fuller	$22,835	$74,959	$79,956	$177,749
7	Janet Leverling	$19,232	$111,789	$82,031	$213,051
8	Laura Callahan	$23,161	$59,777	$50,363	$133,301
9	Margaret Peacock	$53,115	$139,478	$57,595	$250,187
10	Nancy Davolio	$38,789	$97,534	$65,821	$202,144
11	Robert King	$18,105	$66,689	$56,502	$141,296
12	**Grand Total**	**$175,237**	**$550,224**	**$392,268**	**$1,117,729**

Figure 7. You can filter a pivot table based on the computed data.

When there are multiple criteria, as with the columns in this example, you can choose which criterion you're sorting or filtering on. Figure 8 shows the dropdown that lets you pick the field to which the rest of the dialog applies.

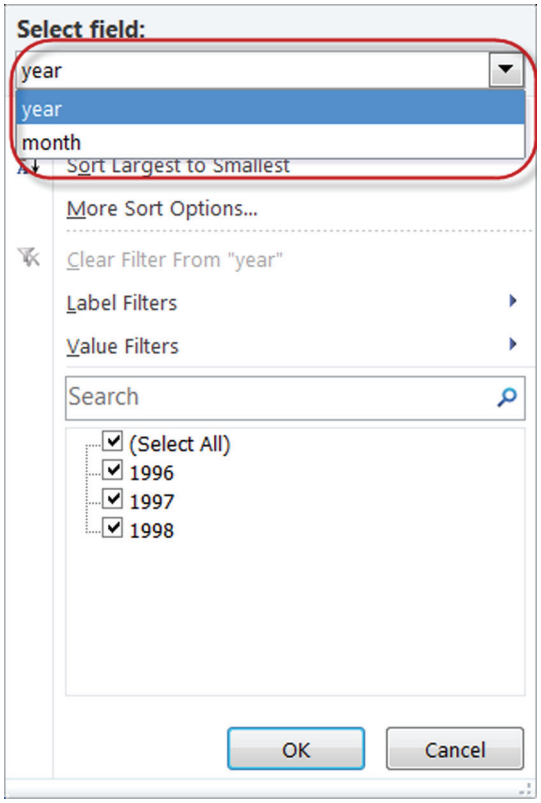

Figure 8. When you use multiple criteria to specify rows or columns, you can choose which to sort or filter on.

Creating pivot tables programmatically

If you can do it interactively in Excel, you can almost always do it programmatically. While I've written (for example, http://tinyurl.com/ybhfsdfc) that recording a macro can result in bad code, when you're first trying to automate a task and have no idea what objects are involved, recording a macro can get you started; that's what I did to figure out how to create a pivot table. In fact, I'd hardly looked at Excel's pivot tables before working on this article, so I actually started out using Excel's Pivot Table Wizard to figure out how to create them at all. (It wasn't obvious to me where to find that Wizard; it's on the Insert tab.) Once I felt comfortable, I recorded a macro using the Wizard, and then adapted that code in VFP.

The first step in creating a pivot table is to send the raw data to Excel. The first part of this article showed multiple ways to do that. Since creating a pivot table implies that Excel is available, I chose to simply create a CSV file with COPY TO, as in Listing 8.

Listing 8. The first step in creating a pivot table is sending the raw data to Excel.

```
OPEN DATABASE HOME(2) + "Northwind\Northwind"

SELECT EmployeeID, ;
       YEAR(OrderDate) AS Year, ;
       MONTH(OrderDate) as Month, ;
       SUM(Quantity*UnitPrice) AS OrderTotal ;
   FROM Orders ;
     JOIN OrderDetails ;
       ON Orders.OrderID = ;
          OrderDetails.OrderID ;
   GROUP BY 1, 2, 3 ;
   INTO CURSOR csrMonthlyTotals

* Add employee name
SELECT PADR(ALLTRIM(FirstName) + ;
         (' ' + LastName), 30) AS cName, ;
       csrMonthlyTotals.* ;
   FROM csrMonthlyTotals ;
     JOIN Employees ;
       ON csrMonthlyTotals.EmployeeID = ;
          Employees.EmployeeID ;
   ORDER BY LastName, FirstName ;
   INTO CURSOR csrReport

LOCAL cCSVFile
cCSVFile = FORCEPATH(FORCEEXT( ;
  "SalesPersonMonthly", "CSV"), SYS(2023))

COPY TO (m.cCSVFile) TYPE csv
```

The simplest way to create a pivot table is to first create a pivotcache object. A pivotcache is an object containing the data you want to put in the pivot table. (Besides being really easy, creating a pivotcache first lets you use the same data for multiple pivot tables.) The pivotcache object has a CreatePivotTable method that does most of the heavy lifting. In Listing 9, the CSV file is opened in Excel, and the pivotcache and pivottable objects are created. Note the use of Excel's UsedRange object to refer to all rows and columns in the exported data

Listing 9. Creating the pivot table object is straightforward, using a pivotcache.

```
LOCAL oExcel AS Excel.Application, ;
      oWorkbook as Excel.Workbook, ;
      oSheet AS Excel.Worksheet, ;
      oPC AS Excel.PivotCache, ;
      oPT AS Excel.PivotTable, ;
      oRange AS Excel.Range

oExcel = CREATEOBJECT("Excel.Application")
oWorkbook = oExcel.Workbooks.Open(m.cCSVFile)
oExcel.Visible = .T.

* Adapted from PivotTable macro
oRange = oExcel.ActiveSheet.UsedRange()
oSheet = oExcel.Sheets.Add()
oSheet.Name = "SalesPivot"
oPC = oExcel.ActiveWorkbook.PivotCaches.;
      Create(1, ; && xlDatabase
             m.oRange, 4)
         && 5=xlPivotTable15; 4=xlPivotTable14

oPT = oPC.CreatePivotTable( ;
  "SalesPivot!R3C1", "PivotTable2", .T., 4)
```

The pivot table object created is empty. If you look at the workbook at this point, the worksheet exists, but there's no data there. Instead you see a prompt from Excel, as in Figure 9. If you click in that area, the Pivot Table Wizard pane appears so you can specify the layout of the pivot table.

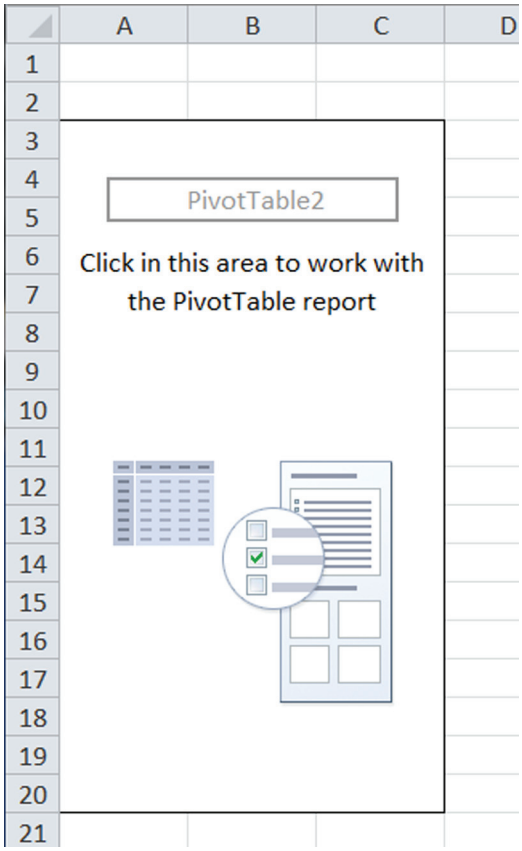

Figure 9. After creating a pivot table, Excel prompts you to populate it.

Of course, we don't want to work with the wizard; we want to do this programmatically. To do that, add one or more data fields using the pivot table's AddDataField method. The method accepts three parameters: the field to add, a caption for it, and a function to apply to the field. Only the first parameter is required.

The secret to providing the pivot field is the pivot table's PivotFields collection, which contains all the fields put into the pivotcache; they're named by the column headers in the original data. In our example, there are five: cname, employeeid, year, month, and ordertotal. The only field there that we want to treat as data is ordertotal, so that's the one we specify in the AddDataField method.

The caption parameter specifies the string that will appear in the upper-left corner of the pivot table. In our example, we want "Orders ($)." Omitting this parameter uses the field name as the caption.

The function parameter is based on the xlConsolidationFunction enumeration; the most common values are shown in Table 1. Other available functions include variance and standard deviation, as well as some variants on count. If you omit this parameter, you get a sum.

Table 1. You can specify what aggregation function to use on a data field.

Function	Excel constant	Value
Average	xlAverage	-4106
Count	xlCount	-4112
Maximum	xlMax	-4136
Minimum	xlMin	-4139
Sum	xlSum	-4157

Listing 10 shows the code to add ordertotal as a data field to the pivot table.

Listing 10. It takes only one line of code to set up a data field in the pivot table.

```
oPT.AddDataField( ;
  oPT.PivotFields("ordertotal"), ;
  "Orders ($)", -4157) && xlSum
```

The other thing we need to do is specify which fields are columns and which are rows. The easiest way to do this is set the relevant properties for each field individually. The two key properties are Orientation and Position. Orientation determines whether the field is used for rows (1 = xlRowField) or for columns (2 = xlColumnField). Position determines the priority of the row or column when

multiple rows or columns are specified. Listing 11 shows the settings for our example, making cname a row field, and year and month column fields, with year coming first.

Listing 11. Set the orientation and position properties of the items in the PivotFields collection to specify the rows and columns you want to show in the pivot table.

```
With oPT.PivotFields("cname")
   .Orientation = 1   && xlRowField
   .Position = 1
EndWith
With oPT.PivotFields("year")
   .Orientation = 2 && xlColumnField
   .Position = 1
ENDWITH
With oPT.PivotFields("month")
   .Orientation = 2 && xlColumnField
   .Position = 2
EndWith
```

At this point, you have a pivot table showing the relevant data. The final steps are cosmetic: setting the row and column descriptions and formatting the data properly.

The CompactLayoutRowHeader and CompactLayoutColumnHeader properties of the pivot table specify the headers for the rows and columns, respectively. These are also the items that contain the dropdowns for sorting and filtering.

Formatting all the data cells in a pivot table is surprisingly simple; just set the NumberFormat property of the relevant member of the PivotFields collection. The code in Listing 12 sets the headers and formats the data cells.

Listing 12. Formatting a pivot table takes just a few lines of code.

```
oPT.CompactLayoutRowHeader = "Salesperson"
oPT.CompactLayoutColumnHeader = "Months/Years"
oPT.PivotFields("Orders ($)").NumberFormat = ;
  "$#,#0"
```

The complete code to create the pivot table shown in Figure 4 is included in this month's downloads as PivotExcelSalespersonMonthly.prg.

You can do lots more with pivot tables, including specifying multiple pivot fields, and including a third dimension (pages). You'll find documentation for the PivotTable object at http://tinyurl.com/y8dzgcft. Note that once we've set a caption for a field, we need to use that caption to refer to it in the collection rather than the field name. So, the code refers to oPT.PivotFields("Orders ($)") rather than oPT.PivotFields("ordertotal").

Next up: Graphing

For many situations, sending crosstab data to Excel will give users exactly what they need. But others want a more visual approach. In my next article, I'll explore ways of getting crosstab data into graphs.

Author Profile

Tamar E. Granor, Ph.D. is the owner of Tomorrow's Solutions, LLC. She has developed and enhanced numerous Visual FoxPro applications for businesses and other organizations. Tamar is author or co-author of a dozen books including the award winning Hacker's Guide to Visual FoxPro, Microsoft Office Automation with Visual FoxPro *and* Taming Visual FoxPro's SQL. *Her latest collaboration is* VFPX: Open Source Treasure for the VFP Developer, *available at www.foxrockx.com. Her other books are available from Hentzenwerke Publishing (www.hentzenwerke.com). Tamar was a Microsoft Support Most Valuable Professional from the program's inception in 1993 until 2011. She is one of the organizers of the annual Southwest Fox conference. In 2007, Tamar received the Visual FoxPro Community Lifetime Achievement Award. You can reach her at tamar@thegranors.com or through www.tomorrowssolutionsllc.com.*