

Smart Access

Solutions for Microsoft® Access® Developers

Automated Excel Pivot Reports from Access

Mark Davis



Excel pivot reports are dynamic, easy to use, and have several advantages over Access reports, including allowing end users more flexibility in their reporting capabilities. In this article, Mark Davis shows how to automate the creation of an Excel pivot report based on an Access data table, using straightforward VBA code.

CROSSTAB queries are helpful tools for displaying data in a tabular format. If I have a table of sales by month for various salespeople, I can use a crosstab query to quickly compare sales numbers across salespeople by month (see [Figure 1](#), on page 5).

This is much easier to read than the underlying Access table, where it's hard to compare sales across months and salespeople at the same time. While you can change the sort order of the table to assist in the comparison, you still must "remember" numbers from the top of the table when viewing the bottom of the table. A crosstab query allows users to assimilate data faster.

One drawback to crosstab queries is that you can only crosstab one value at a time. For example, if I want to show sales by month and sales quota by month, I can't do it with one crosstab query. I'd need to create two crosstab queries and combine them. This can get messy the more measures you have to crosstab—especially if you need to display totals by row or by column, and/or subtotals by group.

Access reports can also be created to display this information. However, the main drawback of Access reports is their inflexibility. If I want my report to crosstab different values, or make the columns display as rows and the rows as columns, I have to create a new Access report. Also, end users can't manipulate the report unless they have Access skills.

One reporting tool that I've found to be very useful for this kind of reporting is an Excel pivot report. A pivot report is a dynamic Excel report that allows the

Continues on page 5

July 2001

Volume 9, Number 7

- 1** Automated Excel Pivot Reports from Access
Mark Davis
- 4** Editorial: Connections
Peter Vogel
- 8** Access 2002 Data Projects for Developers
Mike Gunderloy
- 12** Send Your Files Anywhere
Keith Bombard
- 16** Multi-Select List Box Parser
Hugh K. Manning
- 18** Show Multiple Y-Axes with MS Graph
Doug Den Hoed
- 24** July 2001 Source Code



DOWNLOAD

www.vb123.com/smart

In code, an underscore (_) as the last character of a line indicates that the line has been wrapped for layout purposes. In Access 95 and up you can use the code as it appears, but in Access 2.0 you must recombine the wrapped lines.

Excel Pivot Reports...

Continued from page 1

user to easily change a report's layout by dragging and dropping fields onto a "report template." Users can, for instance, quickly and interactively make the rows columns and the columns rows. They can easily drill down or up to various levels of aggregation. They can also add or remove subtotals and grand totals by row or by column very quickly. Additionally, most of our end users are comfortable with Excel, and can easily make changes to number formatting and printer settings to get the report just the way they like it.

It turns out that it isn't all that difficult to automate creating an Excel pivot report from Access data using VBA code. This article will walk you through an Access program that does exactly that. The program is available in Pivot.mdb in this month's Source Code file, available at www.smartaccessnewsletter.com. The program starts an instance of Microsoft Excel, creates a new workbook, adds the appropriate worksheets, copies data from an Access table to Excel, and then builds a pivot report in Excel based on that data. Once you understand the basics of how this program works, you can easily modify the program (or write your own) to fit your specific reporting needs.

Automating the pivot report

The following sections explain the program in detail, so you can understand what it's doing. The steps in the process are:

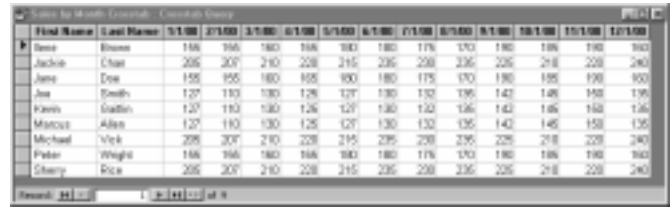
1. Create the Excel workbook and worksheet.
2. Populate an Excel worksheet with data from an Access table.
3. Create and format the pivot report.
4. Format the Excel worksheet.

At the end, I'll discuss how to code some additional Excel functionality into this process.

Creating the worksheet

The first section of the program declares the variables needed to create the pivot report. There are object variables to hold the Excel application, the Excel workbook, and the Excel worksheet. Two object variables are declared for the Access database and the Access recordset that contains the data for the report. There are variables to hold the number of records and fields in the Access table, as well as counter variables used in the looping sections of the code. Finally, three variables hold the name of the two Excel worksheets and the Excel workbook filename:

```
Function Excel_PivotTable()  
Dim objXL2 As Excel.Application  
Dim objXL2Book As Workbook  
Dim objXL2Sheet As Worksheet  
Dim db As DATABASE  
Dim rs As Recordset  
Dim intRecords As Integer  
Dim intFields As Integer  
Dim intCounter As Integer  
Dim intCounter2 As Integer
```



Product Name	Last Name	1/10	2/10	3/10	4/10	5/10	6/10	7/10	8/10	9/10	10/10	11/10	12/10
Jane	Evans	144	166	180	166	180	180	176	170	180	186	180	180
Jackie	Chen	206	207	210	228	215	226	228	236	226	218	228	240
Jane	Doe	156	155	180	155	180	180	175	170	180	155	190	180
Jill	Smith	127	110	130	126	127	130	132	136	142	146	140	136
Mary	Goldin	127	110	130	126	127	130	132	136	142	146	140	136
Michael	Allen	127	110	130	126	127	130	132	136	142	146	140	136
Michael	Wick	206	207	210	228	215	226	228	236	226	218	228	240
Peter	Wright	144	166	180	166	180	180	176	170	180	186	180	180
Tracy	Rice	206	207	210	228	215	226	228	236	226	218	228	240

Figure 1. Example of a crosstab query.

```
Dim strPage1 As String  
Dim strPage2 As String  
Dim strFileName As String
```

Once the variables are declared, the next section of code assigns some of these variables. The filename for the Excel workbook is assigned, including the file's path. This could easily be replaced with a user prompt that would ask the user to assign the filename and path they want. The names of two of the Excel worksheets are stored in the strPage variables, one to hold the Access data and one for the actual pivot report worksheet:

```
strFileName = "c:\pivot.xls"  
strPage1 = "Monthly Sales"  
strPage2 = "Pivot Data"
```

Next, I assign the variables needed to retrieve the Access table data. The first variable assigns the current database to variable db. By using the command "CurrentDb()," I'm telling Access to work with the currently open database. The rs variable is assigned to the Monthly Sales table in Access. This is the table that I'll use to supply data for the example pivot report in this article (you can use a different table by replacing the table name in this command). I could have used a query as the datasource, although the parameter "dbOpenTable" would need to be changed to "dbOpenDynaset" in that case:

```
Set db = CurrentDb()  
Set rs = db.OpenRecordset("Monthly Sales", _  
dbOpenTable)
```

Once I've assigned the recordset variable, I can access that object to find out how many records and fields are contained in the Access table. This will help me later on when I need to know how much data to export to Excel. The RecordCount property counts the number of records in a recordset object, and the Fields.Count method returns the number of fields in the recordset:

```
intRecords = rs.RecordCount  
intFields = rs.Fields.Count
```

Now that I have my Access variables set, I need to open an instance of Excel and prepare an Excel workbook to create the report. When I set the objXL2 variable equal to the Excel Application object using the CreateObject function, a new instance of Excel will be opened in memory. In the following code, I'm opening an instance of Excel 8.0 (Excel 97). For

Excel 2000, you'd need to use `Excel.Application.9`. I originally wrote this application for Excel 97, and this is the only line of code that I had to change to get this program to work in Access 2000:

```
Set objXL2 = CreateObject("Excel.Application.8")
```

Once Excel is open in memory, I need to set some properties of the application so that I can work with it. By making Excel visible, I can view the creation of the Excel pivot report while it happens. You can set the `.Visible` property to `False` if you want the workbook to be created in the background. The routine will run a little faster, but your users won't be as entertained as they will be in watching Excel create the pivot table.

I like to maximize the window so I don't have to worry about sizing it. Also, setting the `DisplayAlerts` property to `False` will prevent Excel from displaying warning messages such as "Are you sure you want to close Excel without saving?" These messages will stop a VBA program dead in its tracks. Setting the `Interactive` property to `True` allows the user to manipulate the version of Excel opened through code. I prefer this setting so that I can leave Excel open when the program completes, and the user can begin manipulating the report right away.

```
With objXL2
    .Visible = True
    .WindowState = xlMaximized
    .DisplayAlerts = False
    .Interactive = True
End With
```

Once I've set all of the Excel application properties the way I want them, the next step is to create a new workbook. The `Workbooks.Add` method will accomplish this objective. Once I have a new workbook added, I can assign it to an object variable to make it easier to access (my code will also execute faster if I go through the object variable). At this point, I also save the workbook to the filename specified in my code:

```
.Workbooks.Add
Set objXL2Book = .ActiveWorkbook
.ActiveWorkbook.SaveAs strFileName
```

In the same way that I added a workbook, I also can add worksheets to the workbook. This is accomplished in the following section of code. By using the `Worksheets.Count` property, I can be sure to add my worksheets at the end of the workbook, regardless of the number of worksheets already contained in the workbook. After adding a sheet and assigning it to an object variable for ease of manipulation, I name the worksheet based on the values set earlier in the code. I also `Activate` the page so I can begin manipulating the worksheet. Once I've added the new worksheets, I delete the default worksheets that are created whenever a new workbook is opened. By using a `For...Next` loop, I can delete all of the default worksheets, regardless of the number created (Excel 97 creates two default worksheets, while Excel

2000 creates three default worksheets):

```
.Worksheets.Add After:= _
    objXL2.Worksheets(objXL2.Worksheets.Count)
Set objXL2Sheet = .ActiveWorkbook.ActiveSheet
objXL2Sheet.Name = strPage1
objXL2Sheet.Activate

.Worksheets.Add After:= _
    objXL2.Worksheets(objXL2.Worksheets.Count)
Set objXL2Sheet = .ActiveWorkbook.ActiveSheet
objXL2Sheet.Name = strPage2
objXL2Sheet.Activate

For intCounter = 1 To objXL2.Worksheets.Count - 2
    .ActiveWorkbook. _
        Sheets("Sheet" & intCounter).Delete
Next intCounter
```

Populating an Excel worksheet

At this stage of the program, I have an Excel workbook created, along with the necessary worksheets. The next step is to add the Access table data to the Excel workbook so that I can create the pivot report. I begin by adding the field headings from the Access table to the Excel worksheet last activated, which (by no accident) happens to be the "Pivot Data" worksheet. Using a `For...Next` loop, I loop through every field in the recordset (in this case, the Access table "Monthly Sales"), adding the field name to the Excel worksheet using the `Name` property of the recordset's `.Fields()` collection. The neat thing about this code is that it works regardless of the size of the Access recordset, and regardless of the names of the Access fields.

```
For intCounter = 0 To rs.Fields.Count - 1
    .ActiveSheet.Cells(1, 1 + intCounter).Value = _
        rs(intCounter).Name
Next intCounter
```

Using a similar technique, I can add all of the data values from the Access table to the worksheet. This time I use two nested `For...Next` loops, one to loop by record and one to loop by field within the record. Again, this code works regardless of the size and structure of the Access recordset: These two loops are all the code you need to write any size table to an Excel worksheet! Of course, Excel does have a limit to the number of records it can hold, so you need to keep that in mind:

```
For intCounter = 1 To intRecords
    For intCounter2 = 0 To rs.Fields.Count - 1
        .ActiveSheet.Cells(1 + intCounter, _
            intCounter2 + 1).Value = rs(intCounter2)
    Next intCounter2
    rs.MoveNext
Next intCounter
```

Creating and formatting

Now that I have all of my Access data in Excel, I can begin to create the pivot report. As you'll see, there isn't a lot of code needed to create the report. The `PivotTableWizard` command that I use in my code does most of the work. The Excel range that the report is based on is calculated from the size of the Access recordset. The report will be placed on the page created for the pivot report specified

earlier (in the current workbook).

You can also tell Excel to query an external database for the pivot report data with the PivotTableWizard method, but I prefer to have the data stored in the Excel worksheet. You can read more about the PivotTableWizard command by searching for the PivotTableWizard method in Excel VBA Help. You can also specify whether or not to display row and column grand totals, or to save the pivot data with the pivot report. Is this cool or what?

```
.ActiveSheet.PivotTableWizard _
  SourceType:=xlDatabase, _
  SourceData:=.ActiveSheet.Range(.Cells(1, 1), _
    Cells(1 + intRecords, _
      1 + rs.Fields.Count - 1)), _
  TableDestination:= _
    "[" & objXL2Book.Name & "]" & strPage1 & _
    "!R1C1:R3C1", _
  TableName:="PivotTable1", RowGrand:=True, _
  ColumnGrand:=True, SaveData:=True
```

Once the pivot report is created, the next section tells Excel which fields to use as the page, row, and column fields. If desired, this can easily be modified to force a different pivot report layout:

```
.ActiveSheet._
  PivotTables("PivotTable1").AddFields _
  RowFields:=Array( _
    "First Name", "Last Name", "Data"), _
  ColumnFields:="Month", PageFields:= _
    "Office"
```

The following code turns off subtotals for each row in the Excel pivot report. In this data, since there are 12 elements (months) at the column level, Excel requires the entire array to be set to False. If you want subtotals for each row element, you can eliminate these commands since subtotals are displayed by default when you create a pivot report in Excel.

```
.ActiveSheet.PivotTables("PivotTable1"). _
  PivotFields("First Name").Subtotals = _
  Array(False, False, False, False, False, _
    False, False, False, False, False, False)
.ActiveSheet.PivotTables("PivotTable1"). _
  PivotFields("Last Name").Subtotals = _
  Array(False, False, False, False, False, _
    False, False, False, False, False, False)
```

Now I can format the data values in the pivot report by setting the position, number format, and function of each measure. Notice that the third measure, "% of Monthly Quota," is an average calculation, while the other two measures are sums:

```
With .ActiveSheet.PivotTables("PivotTable1"). _
  PivotFields("Quota In Thousands")
  .Orientation = xlDataField
  .Name = "Quota"
  .Position = 1
  .NumberFormat = "#,##0"
  .Function = xlSum
End With

With .ActiveSheet.PivotTables("PivotTable1"). _
  PivotFields("Sales In Thousands")
  .Orientation = xlDataField
  .Name = "Sales"
```

```
.Position = 2
.NumberFormat = "#,##0"
.Function = xlSum
End With

With .ActiveSheet.PivotTables("PivotTable1"). _
  PivotFields("% of Monthly Quota")
  .Orientation = xlDataField
  .Name = "% of Quota"
  .Position = 3
  .NumberFormat = "#,###.##"
  .Function = xlAverage
End With
```

I finish up this section by formatting the column headings to display the months properly:

```
.ActiveSheet.PivotTables("PivotTable1"). _
  PivotFields("Month").NumberFormat = "mmm-yy"
```

Formatting the Excel worksheet

Now that the pivot report is created, the only thing left to do is to format the page setup and Excel worksheet the way I want it. I start by setting page setup properties including margins, centering, orientation, print to fit, and footers. These settings are based on my current printer, an HP Laserjet 5:

```
.ActiveSheet.PageSetup.LeftMargin = _
  .InchesToPoints(0.25)
.ActiveSheet.PageSetup.RightMargin = _
  .InchesToPoints(0.25)
.ActiveSheet.PageSetup.TopMargin = _
  .InchesToPoints(1.5)
.ActiveSheet.PageSetup.BottomMargin = _
```

Continues on page 20

Excel Pivot Reports...

Continued from page 7

```
.InchesToPoints(0.5)
.ActiveSheet.PageSetup.HeaderMargin = _
.InchesToPoints(0.5)
.ActiveSheet.PageSetup.FooterMargin = _
.InchesToPoints(0#)
.ActiveSheet.PageSetup.CenterHorizontally = True
.ActiveSheet.PageSetup.Orientation = xlLandscape
.ActiveSheet.PageSetup.Zoom = False
.ActiveSheet.PageSetup.FitToPagesWide = 1
.ActiveSheet.PageSetup.FitToPagesTall = 1
.ActiveSheet.PageSetup.CenterFooter = _
"Page &P of &N"
.ActiveSheet.PageSetup.RightFooter = "&D"
```

Next, I define the title of the pivot table report in the header setting. I also set the Zoom property so that I can see more of the report at one time:

```
.ActiveSheet.PageSetup.CenterHeader = _
"&"Arial,Bold"&16Monthly Sales Report" & _
Chr(10) & "Report Date: &D"
.ActiveWindow.Zoom = 75
```

The next command tells the pivot report to display each of its page fields on a different Excel worksheet. This is a nice feature that can come in handy when you want to create multiple page reports.

```
.ActiveSheet.PivotTables("PivotTable1")._
ShowPages PageField:="Office"
```

I now set the cursor to a specific page. This is the page the user will see first upon opening the Excel workbook. I then save the workbook. Since I already saved the workbook to a specific filename, I don't need to specify the filename again:

```
.Sheets("Monthly Sales").Select
.ActiveWorkbook.Save
```

At this point, you can quit Excel if you wish (I have the exit line commented out since I like the workbook to remain open when I'm done so that the user can interact with it). The last few lines clean up all of my object variables in memory by setting them to nothing:

```
'objXL2.Quit
Set rs = Nothing
Set db = Nothing
Set objXL2Sheet = Nothing
Set objXL2Book = Nothing
Set objXL2 = Nothing
```

Voilà! You now have an Excel pivot report, generated entirely from Access VBA code. An example of the final report is shown in [Figure 2](#).

More power

If you want to add additional Excel formatting or commands that aren't described in this article, you can figure them out by using Excel's macro functionality. In Excel, simply choose Tools | Macro | Record New Macro and give the macro any name. Then perform the formatting or functionality you want within Excel. Stop the macro recording, and go to



			Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Total
4	Item Name	Quantity													
5	Sales	Sales	100	100	100	100	100	100	100	100	100	100	100	100	1200
6	Profit	Profit	20	20	20	20	20	20	20	20	20	20	20	20	240
7	Total	Total	120	120	120	120	120	120	120	120	120	120	120	120	1440
8															
9															
10															
11															
12															
13															
14															
15															
16															
17															
18															
19															
20															
21															
22															
23															
24															
25															
26															
27															
28															
29															
30															
31															
32															
33															
34															
35															
36															
37															
38															
39															
40															
41															
42															
43															
44															
45															
46															
47															
48															
49															
50															
51															
52															
53															
54															
55															
56															
57															
58															
59															
60															
61															
62															
63															
64															
65															
66															
67															
68															
69															
70															
71															
72															
73															
74															
75															
76															
77															
78															
79															
80															
81															
82															
83															
84															
85															
86															
87															
88															
89															
90															
91															
92															
93															
94															
95															
96															
97															
98															
99															
100															

Figure 2. The final pivot report.



- [EXCELPVT.ZIP](#)—Mark Davis has included a sample application that demonstrates how you can generate Excel Pivot tables. The sample application exports data to Excel, so you must have Excel installed on your computer to use this sample. (Access 97)

Tools | Macro | Visual Basic Editor. You can find the VBA code that Excel wrote to perform your task under the Modules tab. With a little trial and error, you should be able to cut and paste this code into your Access program to automate that function.

I hope you found this article a helpful primer on Excel automation. Even if you don't necessarily need to automate a pivot report, you can use these basic concepts to automate any kind of Excel workbook. With a little bit of effort, your end users will be calling you their hero. ▲