

# Generating crosstabs in VFP

*Several tools make it easy to change from normalized data to crosstabled data with both rows and columns determined by the data in the set.*

**Tamar E. Granor, Ph.D.**

One of the questions people often ask is how to convert their normalized VFP data into a crosstab, with each data item from a specified column turning into the header for a separate column. There are several tools for creating crosstabs from Visual FoxPro data.

To start exploring crosstabs, we'll start with something simpler. The sample Northwind company has employees only in the US and the UK. Suppose you want to know how many there are in each country. A simple query, shown in [Listing 1](#), gives you the answer; there are 4 UK employees and 5 US employees.

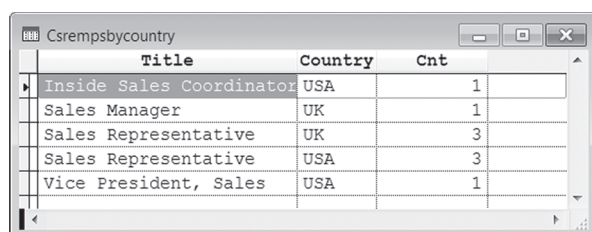
**Listing 1.** It's easy to get one record for each value of interest. Here, each record indicates how many employees are in a given country.

```
SELECT country, COUNT(*) ;
FROM Employees ;
GROUP BY Country ;
INTO CURSOR csrEmpsByCountry
```

But what if you want to know how many there are in each job in each country? Again, that's not hard. [Listing 2](#) gives us one record for each combination of job title and country; results are shown in [Figure 1](#).

**Listing 2.** Group on more fields to break the data down into smaller groups.

```
SELECT Title, Country, COUNT(*) ;
FROM Employees ;
GROUP BY Title, Country ;
INTO CURSOR csrEmpsByCountry
```



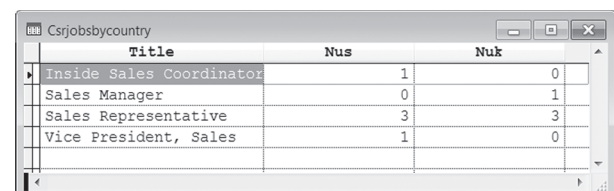
Title	Country	Cnt
Inside Sales Coordinator	USA	1
Sales Manager	UK	1
Sales Representative	UK	3
Sales Representative	USA	3
Vice President, Sales	USA	1

**Figure 1.** Grouping on Title and Country lets us see how many employees have each job in each country.

But what if what you really want is one record for each job title with a column for each country showing how many employees in that country have that job? That's a crosstab. For a simple case like this, you can get what you need with IIF(), as in [Listing 3](#) (included in this month's downloads as JobsByCountry.PRG). [Figure 2](#) shows the results, with one row for each job title and one column for each country.

**Listing 3.** Here, combining SUM() with IIF() counts the number of employees in each country with each job.

```
SELECT Title, ;
SUM(IIF(Country="US",1,0)) AS nUS,
SUM(IIF(Country="UK",1,0)) AS nUK ;
FROM Employees ;
GROUP BY Title ;
INTO CURSOR csrJobsByCountry
```



Title	Nus	Nuk
Inside Sales Coordinator	1	0
Sales Manager	0	1
Sales Representative	3	3
Vice President, Sales	1	0

**Figure 2.** A crosstab uses data to determine what columns are in the result.

This approach works fine when you know the exact number of columns you want and it's not too many. But more often, you don't know how many columns will be in the result, and you may not even know the set of values they're drawn from. That's where a crosstab tool comes in handy.

## VFPXTab

There's been a crosstab tool in the box since FoxPro 2.0, where it was called GenXTab. The version that comes with VFP is VFPXTab.PRG and it's found in the VFP home folder. The system variable `_GENXTAB` points to it; the Query Designer uses `_GENXTAB` when you create a crosstab query.

Comments at the top of VFPXTab.PRG explain its parameters and show how to call it, including the structure of the data needed for it to work. Basically, it expects to find a table or cursor with three columns. By default, the data in the first column becomes the rows in the result, the data in the second column becomes the columns in the result, and the data in the third column is aggregated (summed, by default) to form the data in the result.

For example, suppose we want the total sales for each salesperson for each year with a row for each salesperson and a column for each year. We start with a query to collect the relevant data, shown in Listing 4; Figure 3 shows partial results.

**Listing 4.** This query gives us one record for each employee for each year, showing total sales.

```
SELECT EmployeeID, ;
        YEAR(OrderDate) AS OrderYear, ;
        SUM(Quantity*UnitPrice) AS OrderTotal ;
FROM Orders ;
JOIN OrderDetails ;
ON Orders.OrderID =
    OrderDetails.OrderID ;
GROUP BY 1, 2 ;
INTO CURSOR csrYearlyTotals
```

Employeeid	Orderyear	Ordertotal
1	1996	38789.0000
1	1997	97533.5800
1	1998	65821.1300
2	1996	22834.7000
2	1997	74958.6000
2	1998	79955.9600
3	1996	19231.8000
3	1997	111788.6100
3	1998	82030.8900
4	1996	53114.8000
4	1997	139477.7000
4	1998	57594.9500
5	1996	21965.2000
5	1997	32595.0500
5	1998	21007.5000
6	1996	17731.1000

**Figure 3.** The query in Listing 4 results in one record per salesperson per year.

To get the results we want, we instantiate the GenXTab class in VFPXTab.PRG and call its MakeXTab method, as in Listing 5. The class takes 10 parameters that configure its behavior. (Because I don't recommend actually using this cross-tab tool, I'm not going to go into detail about the parameters. They're documented in the code.) The two parameters passed in the example indicate that the result should be stored in a cursor (rather than a table) named csrXTab. Figure 4 shows the result.

**Listing 5.** The GenXTab class in VFPXTab accepts up to 10 parameters.

```
oXTab = NEWOBJECT("genxtab", _GENXTAB, '', ;
                  "csrXTab", .t.)
oXTab.MakeXtab()
```

Employeeid	N_1996	N_1997	N_1998
1	38789.0000	97533.5800	65821.1300
2	22834.7000	74958.6000	79955.9600
3	19231.8000	111788.6100	82030.8900
4	53114.8000	139477.7000	57594.9500
5	21965.2000	32595.0500	21007.5000
6	17731.1000	45992.0000	14475.0000
7	18104.8000	66689.1400	56502.0500
8	23161.4000	59776.5200	50363.1100
9	11365.7000	29577.5500	42020.7500

**Figure 4.** This crosstab has one row per employee and one column per year

A complete program collecting the data and running the crosstab, as well as timing the crosstab process, is included in this month's downloads as SalesPersonAnnualSales.PRG. The downloads also include SalesPersonMonthlySales.PRG, which generates a crosstab with one row per salesperson and a column for each month of 1997.

As the size of the data set supplied to VFPXTab grows, generating a crosstab gets much slower. In Listing 6, the result has one row for each date in the data set and a column for each salesperson; Figure 5 shows partial results. This example is included in this month's downloads as SalesPersonDailySales.PRG.

**Listing 6.** Here, the final result has one row per day with one column per salesperson.

```
SELECT OrderDate, ;
        EmployeeID, ;
        SUM(Quantity*UnitPrice) AS OrderTotal ;
FROM Orders ;
JOIN OrderDetails ;
ON Orders.OrderID = ;
    OrderDetails.OrderID ;
GROUP BY 1, 2 ;
INTO CURSOR csrDailyTotals
```

```
LOCAL oXTab
```

```
oXTab = NEWOBJECT("genxtab", _GENXTAB, '', ;
                  "csrXTab", .t.)
oXTab.MakeXtab()
```

Orderdate	N_1	N_2	N_3	N_4
07/04/199	0.0000	0.0000	0.0000	0.0000
07/05/96	0.0000	0.0000	0.0000	0.0000
07/08/96	0.0000	0.0000	670.8000	1813.0000
07/09/96	0.0000	0.0000	0.0000	3730.0000
07/10/96	0.0000	0.0000	1444.8000	0.0000
07/11/96	0.0000	0.0000	0.0000	0.0000
07/12/96	0.0000	0.0000	0.0000	0.0000
07/15/96	0.0000	0.0000	517.8000	0.0000
07/16/96	0.0000	0.0000	0.0000	1119.9000
07/17/96	2018.6000	0.0000	0.0000	0.0000
07/18/96	0.0000	0.0000	0.0000	100.8000
07/19/96	0.0000	0.0000	0.0000	2194.2000
07/22/96	0.0000	0.0000	0.0000	0.0000
07/23/96	0.0000	0.0000	0.0000	0.0000
07/24/96	0.0000	0.0000	0.0000	0.0000
07/25/96	0.0000	1176.0000	0.0000	0.0000

**Figure 5.** Partial results for a crosstab of daily sales by salesperson. There are more rows and more columns in the full result.

In my tests, the crosstab with one row per salesperson and one column per year varies from about 1/10<sup>th</sup> of a second to about 1/100<sup>th</sup> of a second, but

the daily sales example takes nearly half a second. (Note that, in each case, I'm timing only crosstab generation, not the query that assembles the data.)

## FastXTab

Fortunately, there's a faster way to get crosstabs. FastXTab was created by Alexander Golovlev specifically to address VFPXTab's speed issues. It's available for download from the Universal Thread at <https://www.universalthread.com/ShowHeaderDownloadOneItem.aspx?ID=9944>. FastXTab replaces VFPXTab's long list of parameters (which were a throwback to the program's non-OOP origins) with properties. Table 1 shows the key properties.

**Table 1.** FastXTab uses properties for configuration.

Property	Default value	Purpose
cOutFile	"xtabquery"	Alias of the output table or cursor
ICursorOnly	.F.	Indicates whether the results are placed in a table (.F.) or cursor (.T.)
ICloseTable	.T.	Indicates whether the source table should be closed after creating the crosstab
nRowField	1	Indicates which column of the source table provides the rows
nColField	2	Indicates which column of the source table provides the columns
nDataField	3	Indicates which column of the source table provides the data to be aggregated
ITotalRows	.F.	Indicates whether the result should include an extra row totaling the data in the other rows
IBrowseAfter	.F.	Indicates whether the result should be opened in a Browse window

The use of properties makes code that calls FastXTab easier to read than code that calls VFPXTab. Listing 7 shows the FastXTab equivalent of Listing 5; it's based on the same query (shown in Listing 4) and, of course, produces the same results. The complete code, including timing test is included in this month's downloads as SalesPersonAnnualSalesFast.prg.

**Listing 7.** With FastXTab, the options you choose are easy to understand because you specify them with properties.

```
oXTab = NEWOBJECT("fastxtab", "fastxtab.prg")
WITH oXTab AS FastXTab OF "fastxtab.prg"
    .cOutFile = "csrXtab"
    .ICursorOnly = .T.
    .ICloseTable = .T.
    .RunXtab()
ENDWITH
```

For this example, FastXTab isn't faster than VFPXTab. But for the daily sales example, you can see the major improvement FastXTab offers. Using the same original query as in Listing 6 and the call to FastXTab shown in Listing 7, on my computer, the crosstab is computed in about 2/100<sup>th</sup> of a second, that is, more than 2000% faster than with VFPXTab. The code, including timing test, is included in this month's downloads as SalesPersonDailySalesFast.prg.

## FastTab 1.6

The additional speed and readability are enough to make me recommend using FastXTab rather than VFPXTab. But, as the infomercials say, "wait ... there's more."

Community member Vilhelm-Ion Praisach has extended FastXTab considerably, both for usability and to provide additional capabilities. It's included in this month's downloads AS FastXTab16.ZIP. Praisach's documentation (as well as the latest version) can be found at <http://praisachion.blogspot.com/2015/02/fastxtab-version-16.html>. The documentation includes lots of examples; I've broken those examples out into individual PRGs, which are included as FastXTab16Demos.ZIP in this month's downloads.

Among the things supported in Praisach's version are specifying the relevant fields by name rather than column number; specifying multiple data columns; specifying the function to apply to a given data column, including with a custom expression; and filtering the data source.

All the properties supported by the original FastXTab remain, but there are quite a few new properties to provide new capabilities. Table 2 shows some of them.

**Table 2.** FastXTab 1.6 includes many new properties to support new capabilities.

Property	Default value	Purpose
cRowField	""	Expression from the source table that provides the rows
cColField	""	Expression from the source table that provides the columns
cDataField	""	Name of the column in the source table that provides the data to be aggregated
cPageField	""	Expression from the source table that provides "pages"
nFunctionType	1	Indicates how to aggregate the data; see Table 3.
cFunctionExp	""	Expression to use for custom aggregation
cCondition	""	Filter to apply to source data before aggregating
cHaving	""	Filter to apply to source data after aggregating
nMultiDataField	1	Indicates how many data fields are specified
anDataField		Array of data fields specified by column position
acDataField		Array of data fields specified by name
anFunctionType		Array of aggregation function choices
acFunctionExp		Array of custom aggregation expressions

Calling FastXTab 1.6 is the same as calling the original FastXTab (which I'll refer to as FastXTab 1.0), except that you need to point to the FastXTab 1.6 version of FastXTab.PRG. This month's downloads include SalesPersonAnnualSalesFast16.PRG, which is identical to SalesPersonAnnualSalesFast.PRG, except for the path to the class library. In my tests, it runs just as fast as the FastXTab 1.0 version.

But FastXTab 1.6 lets you do much more. Suppose you want to know how many sales each salesperson had in each year. Specifying 2 for the nFunctionType property indicates Count; Table 3 shows the options for this property.

**Table 3.** FastXTab 1.6 offers six ways of aggregating the data.

nFunctionType	Meaning
1	Sum
2	Count
3	Average
4	Min
5	Max
6	Custom expression specified in cFunctionExp (or acFunctionExp for the relevant column)

The code in Listing 8 (in this month's downloads SalesPersonAnnualSalesCountFast16.prg) shows another cool feature of FastXTab 1.6, the ability to use an expression to specify the rows or columns rather than just a field name. To get the correct results here, we need to see every order, but we want them counted by year. So the query that gathers the data keeps the original OrderDate column, but the .cColField property specifies "YEAR(OrderDate)," so that the result has one column per year. (In fact, the data collection query is unnecessary here. FastXTab could be run directly against the Orders table.) Figure 6 shows the results.

**Listing 8.** FastXTab 1.6 lets you specify the aggregation function to apply, as well as specify expressions for rows and columns.

```

SELECT EmployeeID, OrderDate, OrderID ;
FROM Orders ;
INTO CURSOR csrOrders

LOCAL oXTab AS FastXTab OF ;
    "fastxtab16\fastxtab.prg"

oXTab = NEWOBJECT("fastxtab", ;
    "fastxtab16\fastxtab.prg")
WITH oXTab AS FastXTab OF "fastxtab.prg"
    .nFunctionType = 2 && Count
    .cRowField = "EmployeeID"
    .cColField = "YEAR(OrderDate)"
    .cDataField = "OrderID"
    .cOutFile = "csrXtab"
    .lCursorOnly = .T.
    .lCloseTable = .T.
    .RunXtab()
ENDWITH

```

What if you want to know not just total sales or the number of sales for each salesperson for each year,



Employeeid	N_1996	N_1997	N_1998
1	26	55	42
2	16	41	39
3	18	71	38
4	31	81	44
5	11	18	13
6	15	33	19
7	11	36	25
8	19	54	31
9	5	19	19

**Figure 6.** Here, the nFunctionType property was set to 2 to count the number of sales for each employee each year.

but the total sales, the average sale and the number of sales? With FastXTab 1.6, you can specify multiple data columns and apply a different aggregation function to each. To specify multiple data columns, set nMultiDataField to the number of data columns, and then populate either anDataField or acDataField with the list of data columns. If you want different aggregation for different columns, fill anFunctionType as well. In Listing 9 (included in this month's downloads as SalesPersonAnnualSumAvgCnt.prg), nMultiDataField is set to 3. The first two columns use the same field from the source, OrderTotal, but each applies a different function. Figure 7 shows the results. There are three columns for each year, one showing the total, one the average, and one the count.

**Listing 9.** FastXTab 1.6 lets you specify multiple data columns.

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

LOCAL oXTab AS FastXTab OF ;
"fastxtab16\fastxtab.prg"
```

Employeeid	N_1996	N_1996_2	N_1996_3	N_1997	N_1997_2	N_1997_3	N_1998	N_1998_2	N_1998_3
1	38789.0000	1491.885	26	97533.5800	1773.338	55	65821.1300	1567.170	42
2	22834.7000	1427.169	16	74958.6000	1828.259	41	79955.9600	2050.153	39
3	19231.8000	1068.433	18	111788.6100	1574.488	71	82030.8900	2158.708	38
4	53114.8000	1713.381	31	139477.7000	1721.947	81	57594.9500	1308.976	44
5	21965.2000	1996.836	11	32595.0500	1810.836	18	21007.5000	1615.962	13
6	17731.1000	1182.073	15	45992.0000	1393.697	33	14475.0000	761.842	19
7	18104.8000	1645.891	11	66689.1400	1852.476	36	56502.0500	2260.082	25
8	23161.4000	1219.021	19	59776.5200	1106.973	54	50363.1100	1624.617	31
9	11365.7000	2273.140	5	29577.5500	1556.713	19	42020.7500	2211.618	19

**Figure 7.** Using the nMultiDataField property, you can get multiple data columns for each value of the specified column field. Here, there are three columns for each year.

```
oXTab = NEWOBJECT("fastxtab", ;
"fastxtab16\fastxtab.prg")
WITH oXTab AS FastXTab OF ;
"fastxtab16\fastxtab.prg"
.cRowField = 'EmployeeID'
.cColField = 'YEAR(OrderDate)'
.nMultiDataField = 3
.acDataField[1] = 'OrderTotal'
.anFunctionType[1] = 1
.acDataField[2] = 'OrderTotal'
.anFunctionType[2] = 3
.anDataField[3] = 'OrderID'
.anFunctionType[3] = 2
.cOutFile = "csrXtab"
.lCursorOnly = .T.
.lCloseTable = .F.
.RunXtab()
ENDWITH
```

The previous examples showed that you can use an expression to specify the column field. In fact, you can use an expression for the row field, too and that expression can be fairly complex (in either case). Listing 10 inverts the problem we've been looking at, putting employees in columns and time period in rows. In this case, each row specifies a quarter, using an expression that gives a result like 1998\_Q1 (for the first quarter of 1998). Figure 8 shows partial results. This query is included in this month's downloads as SalesPersonColsQuarterly.PRG.

**Listing 10.** The expressions used to specify row and columns can be complex.

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

LOCAL oXTab AS FastXTab OF ;
"fastxtab16\fastxtab.prg"

oXTab = NEWOBJECT("fastxtab", ;
"fastxtab16\fastxtab.prg")
WITH oXTab AS FastXTab OF ;
"fastxtab16\fastxtab.prg"
.cRowField = 'PADL(YEAR(OrderDate),4) + ;
```

```

    "_Q" + PADL(QUARTER(OrderDate),1)'
.cColField = 'EmployeeID'
.nMultiDataField = 3
.acDataField[1] = 'OrderTotal'
.anFunctionType[1] = 1
.acDataField[2] = 'OrderTotal'
.anFunctionType[2] = 3
.anDataField[3] = 'OrderID'
.anFunctionType[3] = 2
.cOutFile = "csrXtab"
.lCursorOnly = .T.
.lCloseTable = .F.
.RunXtab()
ENDWITH

```

```

oXTab = NEWOBJECT("fastxtab", ;
                  "fastxtab16\fastxtab.prg")
WITH oXTab AS FastXTab ;
    OF "fastxtab16\fastxtab.prg"
    .cPageField = 'ShipCountry'
    .cRowField = 'CategoryName'
    .cColField = 'YEAR(OrderDate)'
    .cDataField = 'NumSold'
    .cOutFile = "csrXtab"
    .lCursorOnly = .T.
    .lCloseTable = .F.
    .RunXtab()
ENDWITH

```

C_1996_q3	N_1	N_1_2	N_1_3	N_2	N_2_2
1996_Q3	14909.4000	1355.400	11	5940.8000	742.600
1996_Q4	23879.6000	1836.892	13	16893.9000	2111.738
1997_Q1	15330.1000	1703.344	9	7639.3000	848.811
1997_Q2	15520.9000	1552.090	10	25667.5000	2566.750
1997_Q3	33578.4800	1865.471	18	19999.7500	1818.159
1997_Q4	33104.1000	2069.006	16	21652.0500	1968.368
1998_Q1	45146.4800	1881.103	24	45101.4100	2653.024
1998_Q2	20674.6500	2067.465	10	34854.5500	2178.409

**Figure 8.** Each row here represents a quarter and each set of three columns represents an employee.

Both versions of FastXTab support “pages,” the ability to group rows. (Imagine each “page” as being a tab in a workbook.) With FastXTab 1.0, you specify the column that determines “pages” with the nPageField property. FastXTab 1.6’s cPageField property lets you specify a field name or an expression, just as you do for rows and columns. **Listing 11** (included in this month’s downloads as ProductsSold.PRG) shows how many units of each category were sold and shipped to each country each year; **Figure 9** shows partial results. The most obvious use for data in this form is making grouping in a report simpler.

**Listing 11.** Use cPageField to specify an expression to “page” by in the crosstab.

```

SELECT ProductName, CategoryName, ;
       OrderDate, ShipCountry, ;
       SUM(Quantity) AS NumSold ;
FROM Orders ;
JOIN OrderDetails OD ;
    ON Orders.OrderID = OD.OrderID ;
JOIN Products ;
    ON OD.ProductID = Products.ProductID ;
JOIN Categories ;
    ON Products.CategoryID = ;
       Categories.CategoryID ;
GROUP BY 1, 2, 3, 4 ;
INTO CURSOR csrProductsSold

LOCAL oXTab AS FastXTab ;
    OF "fastxtab16\fastxtab.prg"

```

Shipcountry	Categoryname	N_1996	N_1997	N_1998
Argentina	Beverages	0	3	79
Argentina	Condiments	0	10	35
Argentina	Confections	0	29	28
Argentina	Dairy Products	0	3	51
Argentina	Grains/Cereals	0	0	20
Argentina	Produce	0	19	14
Argentina	Seafood	0	30	18
Austria	Beverages	188	335	459
Austria	Condiments	184	410	126
Austria	Confections	65	393	117
Austria	Dairy Products	212	430	385
Austria	Grains/Cereals	60	220	300
Austria	Meat/Poultry	14	283	65
Austria	Produce	99	201	88
Austria	Seafood	127	75	331
Belgium	Beverages	12	92	168
Belgium	Condiments	0	60	87
Belgium	Confections	40	123	107
Belgium	Dairy Products	65	110	120
Belgium	Grains/Cereals	0	67	78
Belgium	Meat/Poultry	40	14	35
Belgium	Produce	28	50	20

**Figure 9.** Here, each row represents one category for one country and each column represents a year. The data is the number of units of that category shipped to that country in the specified year.

When you set nFunctionType to 6 (or set anFunctionType for a particular column to 6), you can specify a custom aggregation calculation. In **Listing 12**, we calculate the ratio of shipping cost (Freight) to the order total for each month for each customer. **Figure 10** shows partial results; the code is included in this month’s downloads as FreightRatio.PRG

**Listing 12.** You can set nFunctionType to 6 and cFunctionExp to a custom aggregation calculation. Here, it's the ratio of shipping cost to order total.

```
SELECT CustomerID, Orders.OrderID, ;
       OrderDate, ;
       SUM(Quantity*UnitPrice) AS OrderTotal,;
       Freight ;
FROM Orders ;
JOIN OrderDetails OD ;
ON Orders.OrderID = OD.OrderID ;
GROUP BY 1, 2, 3, 5 ;
INTO CURSOR csrOrderTotals

LOCAL nStart, nEnd
LOCAL oXTab AS FastXTab ;
OF "fastxtab16\fastxtab.prg"

oXTab = NEWOBJECT("fastxtab", ;
                  "fastxtab16\fastxtab.prg")
WITH oXTab AS FastXTab ;
OF "fastxtab16\fastxtab.prg"
.cRowField = 'CustomerID'
.cColField = 'PADL(YEAR(OrderDate),4)' + ;
             ' + "_" + PADL(MONTH(OrderDate),2,"0")'
.nFunctionType = 6
.cFunctionExp = ;
             'SUM(Freight)/SUM(OrderTotal)'
.cOutFile = "csrXtab"
.lCursorOnly = .T.
.lCloseTable = .T.
.RunXtab()
ENDWITH
```

FastXTab 1.6 lets you filter the data both before and after it's aggregated. Use cCondition to filter before aggregating, and cHaving to filter afterward.

In **Listing 13**, we retrieve data on all orders, but use only those from one year in the crosstab, which shows totals for each salesperson for each month. You might use this approach in a loop to generate a crosstab for each year. **Figure 11** shows the results. The code is included in this month's downloads as SalesPersonMonthlyFilter.PRG.

**Listing 13.** The cCondition property filters data out of the cursor or table supplied to FastXTab 1.6. Here, we keep only one year's data.

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

LOCAL oXTab AS FastXTab ;
OF "fastxtab16\fastxtab.prg"

oXTab = NEWOBJECT("fastxtab", ;
                  "fastxtab16\fastxtab.prg")
WITH oXTab AS FastXTab ;
OF "fastxtab16\fastxtab.prg"
.cRowField = 'EmployeeID'
.cColField = 'MONTH(OrderDate)'
.cDataField = 'OrderTotal'
.cCondition = 'YEAR(OrderDate) = 1998'
.cOutFile = "csrXtab"
.lCursorOnly = .T.
.lCloseTable = .T.
.RunXtab()
ENDWITH
```

Customerid	C_1996_07	C_1996_08	C_1996_09	C_1996_10	C_1996_11
ALFKI	0.0000	0.0000	0.0000	0.0000	0.0000
ANATR	0.0000	0.0000	0.0181	0.0000	0.0000
ANTON	0.0000	0.0000	0.0000	0.0000	0.0546
AROUT	0.0000	0.0000	0.0000	0.0000	0.0874
BERGS	0.0000	0.0484	0.0000	0.0000	0.0000
BLAUS	0.0000	0.0000	0.0000	0.0000	0.0000
BLONP	0.0470	0.0000	0.0040	0.0000	0.0178
BOLID	0.0000	0.0000	0.0000	0.0793	0.0000
BONAP	0.0000	0.0000	0.0000	0.0665	0.0620
BOTTM	0.0000	0.0000	0.0000	0.0000	0.0000
BSBEV	0.0000	0.0475	0.0000	0.0000	0.0000
CACTU	0.0000	0.0000	0.0000	0.0000	0.0000
CENTC	0.0322	0.0000	0.0000	0.0000	0.0000
CHOPS	0.0368	0.0000	0.0000	0.0000	0.0000
COMMI	0.0000	0.0367	0.0000	0.0000	0.0000
CONSH	0.0000	0.0000	0.0000	0.0000	0.0000

**Figure 10.** The data here is the ratio of freight cost to order total for a customer for a month.



Employeeid	N_1	N_2	N_3	N_4	N_5
1	8712.1300	11586.9000	24847.4500	13620.9000	7053.7500
2	4693.9500	26056.9000	14350.5600	32681.0500	2173.5000
3	27833.3500	21540.2400	18358.3500	14298.9500	0.0000
4	21029.3000	11325.0500	8835.3900	10130.2100	6275.0000
5	12280.5000	5872.4000	2644.6000	210.0000	0.0000
6	1975.6000	1953.4000	5206.0000	5340.0000	0.0000
7	6610.0000	6795.5500	7130.8000	34792.6500	1173.0500
8	12092.7500	106.0000	20885.7000	14055.3000	3223.3600
9	5627.1400	19325.5100	7566.6000	9501.5000	0.0000

**Figure 11.** This crosstab shows sales for each employee by month for the year specified in the cCondition property.

To demonstrate cHaving, we'll extend the freight ratio example. Perhaps you're interested in seeing only cases where customers seem to be spending too much on freight charges. Add the line in [Listing 14](#) inside the WITH clause in Listing 12 to see only cases where a customer's total monthly freight charges were more than 5% of the total orders. This version of the example is included in this month's downloads as FreightRatioFiltered.PRG.

**Listing 14.** The cHaving property filters after aggregation. Here, it keeps only data where the ratio of freight to order total (for the month) is more than 5%.

```
.cHaving = ;
'SUM(Freight)/SUM(OrderTotal) >= 0.05'
```

## Summing up

It should be obvious that I recommend FastXTab 1.6 over VFPTab or FastXTab 1.0. Believe it or not, FastXTab 1.6 has some additional capabilities not discussed in this article. If you use crosstabs (or if you now see how you can use them), I strongly recommend spending some time not only with the examples from this article, but with the ones that Vilhelm-Ion Praisach provides.

In my next article, I'll take a look at PIVOT, SQL Server's analogue to crosstabs.

## 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](http://www.foxrockx.com). Her other books are available from Hentzenwerke Publishing ([www.hentzenwerke.com](http://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](mailto:tamar@thegranors.com) or through [www.tomorrowssolutionsllc.com](http://www.tomorrowssolutionsllc.com).*