PIVOT = Crosstabs, SQL Style

SQL Server's PIVOT keyword lets you create crosstabs

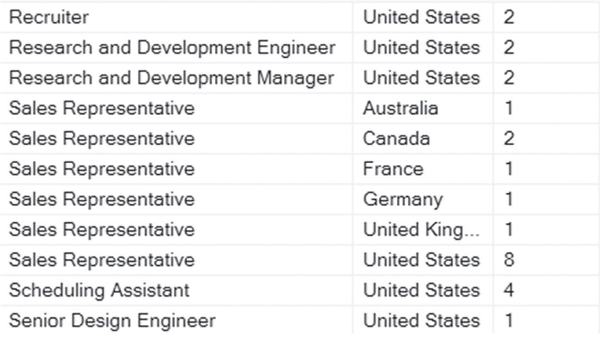
Tamar E. Granor, Ph.D.

A crosstab is a result table or cursor where the set of columns is based on data values in the source. My last article looked at creating crosstabs in VFP, where you can't create a crosstab with just a query. Since SQL Server 2005, however, you can create crosstabs without any additional code or tools.

Suppose you want to know how many employees AdventureWorks has in each country for each job title. The query in Listing 1 answers the question, but the form of the result (partially shown in Figure 1) makes it hard to grasp. The query is included in this month's downloads as JobTitleByCountry.SQL.

Listing 1. This query provides the number of employees with each job title in each country, but each record represents one job title/country combination.

```
SELECT JobTitle, CR.Name, Count(*) AS EmpCount
  FROM [HumanResources].[Employee]
    JOIN [Person].[BusinessEntityAddress] BEA
      ON Employee.BusinessEntityID =
         BEA.BusinessEntityID
    JOIN [Person].[Address]
      ON BEA.AddressID = Address.AddressID
    JOIN [Person].[StateProvince] SP
      ON Address.StateProvinceID =
         SP.StateProvinceID
    JOIN [Person].[CountryRegion] CR
      ON SP.CountryRegionCode =
         CR.CountryRegionCode
  WHERE Employee.CurrentFlag = 1
  GROUP BY JobTitle, CR.Name
  ORDER BY JobTitle, CR.Name
```



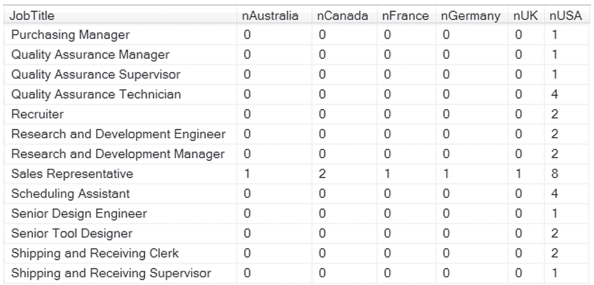
Recruiter	United States	2
Research and Development Engineer	United States	2
Research and Development Manager	United States	2
Sales Representative	Australia	1
Sales Representative	Canada	2
Sales Representative	France	1
Sales Representative	Germany	1
Sales Representative	United King...	1
Sales Representative	United States	8
Scheduling Assistant	United States	4
Senior Design Engineer	United States	1

Figure 1. Each row here shows the number of employees with the specified job title in the specified country.

A better format would be to have one column for each country and one row for each job title, with the intersection of the two containing the number of employees in that country with that job title. As shown in my last article, in VFP, one way to get this result, especially when the number of countries is small, is to use SUM(IIF()) to do the counting. You can do something analogous in T-SQL, using CASE rather than IIF. Listing 2, included in this month's downloads as JobTitleByCountryCase.SQL, shows code to do it this way. Figure 2 shows partial results, much easier to interpret than the previous version.

Listing 2. You can create a simple crosstab using CASE to break out the individual columns.

```
SELECT JobTitle,
       SUM(CASE CR.Name WHEN 'Australia'
           THEN 1 ELSE 0 END) AS nAustralia,
       SUM(CASE CR.Name WHEN 'Canada'
           THEN 1 ELSE 0 END) AS nCanada,
       SUM(CASE CR.Name WHEN 'France'
           THEN 1 ELSE 0 END) AS nFrance,
       SUM(CASE CR.Name WHEN 'Germany'
           THEN 1 ELSE 0 END) AS nGermany,
       SUM(CASE CR.Name WHEN 'United Kingdom'
           THEN 1 ELSE 0 END) AS nUK,
       SUM(CASE CR.Name WHEN 'United States'
           THEN 1 ELSE 0 END) AS nUSA
  FROM [HumanResources].[Employee]
    JOIN [Person].[BusinessEntityAddress] BEA
      ON Employee.BusinessEntityID =
         BEA.BusinessEntityID
    JOIN [Person].[Address]
      ON BEA.AddressID = Address.AddressID
    JOIN [Person].[StateProvince] SP
      ON Address.StateProvinceID =
         SP.StateProvinceID
    JOIN [Person].[CountryRegion] CR
      ON SP.CountryRegionCode =
         CR.CountryRegionCode
  WHERE Employee.CurrentFlag = 1
  GROUP BY JobTitle
  ORDER BY JobTitle
```



JobTitle	nAustralia	nCanada	nFrance	nGermany	nUK	nUSA
Purchasing Manager	0	0	0	0	0	1
Quality Assurance Manager	0	0	0	0	0	1
Quality Assurance Supervisor	0	0	0	0	0	1
Quality Assurance Technician	0	0	0	0	0	4
Recruiter	0	0	0	0	0	2
Research and Development Engineer	0	0	0	0	0	2
Research and Development Manager	0	0	0	0	0	2
Sales Representative	1	2	1	1	1	8
Scheduling Assistant	0	0	0	0	0	4
Senior Design Engineer	0	0	0	0	0	1
Senior Tool Designer	0	0	0	0	0	2
Shipping and Receiving Clerk	0	0	0	0	0	2
Shipping and Receiving Supervisor	0	0	0	0	0	1

Figure 2. Using CASE with SUM() gives one column per country and makes the results more readable.

But T-SQL offers an easier way to do this.

Introducing PIVOT

The PIVOT operator provides a way to crosstab without having to write out all the CASE expressions. PIVOT goes into the FROM clause of the query. Listing 3 shows the syntax for using PIVOT.

In my experience, this is a case where it's easiest to use "*" rather than listing specific field names. The source table can be an actual table, a derived table, or a table created as part of a CTE.

Listing 3. The PIVOT operator appears in the FROM clause of a query and specifies an aggregation function.

```
SELECT <non-pivoted column>,
       <list of pivoted columns with aliases>
FROM <source table>
PIVOT
(<aggregation function>(<column to aggregate>)
  FOR [<column name column>]
    IN (<list of values>)
) AS <alias for the pivot table>
```

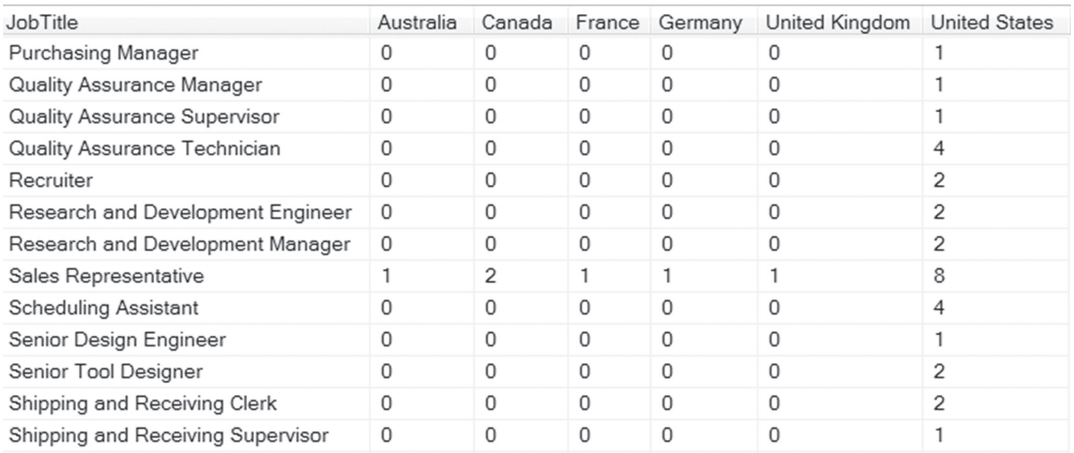
The interesting part is what goes after the PIVOT keyword. First, you need an aggregation function, such as SUM(OrderTotal). After FOR, you list the name of the source column whose values are to become columns in the result. In the job title by country example, that's the Country column.

Finally, after IN, you have to include a list of all the values of interest. Having an explicit list is both a good thing and a bad thing. It's a good thing because it allows you to include only a subset of the values from the relevant column. It's a bad thing, of course, because it requires you to know the list of values from that column.

Listing 4 shows a query using PIVOT that produces the same results as the query in Listing 2. A CTE collects the list of employees with their job titles and countries. The main query uses PIVOT to count the number of employees by country. The CTE has three columns: JobTitle, Country and EmpID. The main query specifies that all three are in the result (SELECT *), but the PIVOT clause indicates that Country determines the columns (the column headers are the actual values from Country), and that EmpID is aggregated, in this case, by counting. Figure 3 shows partial results. The query is included as JobTitleByCountryPivot.SQL in this month's downloads.

Listing 4. This query pivots on country to produce one record per job title with a column for each country where any employees are located.

```
WITH csrJobCountry
  (JobTitle, Country, EmpID)
AS
(SELECT JobTitle, CR.Name,
        Employee.BusinessEntityID
  FROM [HumanResources].[Employee]
    JOIN [Person].[BusinessEntityAddress] BEA
      ON Employee.BusinessEntityID =
         BEA.BusinessEntityID
    JOIN [Person].[Address]
      ON BEA.AddressID = Address.AddressID
    JOIN [Person].[StateProvince] SP
      ON Address.StateProvinceID =
         SP.StateProvinceID
    JOIN [Person].[CountryRegion] CR
      ON SP.CountryRegionCode =
         CR.CountryRegionCode
  WHERE Employee.CurrentFlag = 1
  )

SELECT *
  FROM csrJobCountry
    PIVOT(COUNT(EmpID)
    FOR Country
    IN (Australia, Canada, France, Germany,
        [United Kingdom], [United States]))
    AS EmpTotal
```



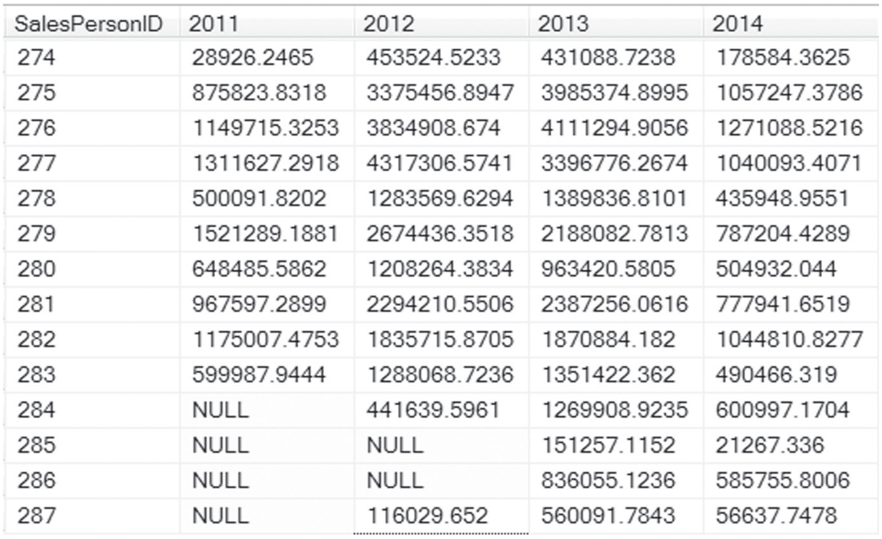
JobTitle	Australia	Canada	France	Germany	United Kingdom	United States
Purchasing Manager	0	0	0	0	0	1
Quality Assurance Manager	0	0	0	0	0	1
Quality Assurance Supervisor	0	0	0	0	0	1
Quality Assurance Technician	0	0	0	0	0	4
Recruiter	0	0	0	0	0	2
Research and Development Engineer	0	0	0	0	0	2
Research and Development Manager	0	0	0	0	0	2
Sales Representative	1	2	1	1	1	8
Scheduling Assistant	0	0	0	0	0	4
Senior Design Engineer	0	0	0	0	0	1
Senior Tool Designer	0	0	0	0	0	2
Shipping and Receiving Clerk	0	0	0	0	0	2
Shipping and Receiving Supervisor	0	0	0	0	0	1

Figure 3. The results here are the same as in Figure 2 except for the column headers, which are the actual country names from the CountryRegion table.

I suspect that the most commonly used function in the pivot is SUM, letting you see some kind of total across a set of time periods or regions or other way of dividing up data. For example, Listing 5 produces total sales for each salesperson for each year; Figure 4 shows partial results. This query is included in this month's downloads as SalesPersonAnnualSalesCTE.SQL.

Listing 5. Here, total sales for each salesperson for each year is computed.

```
WITH SalesByYear
  (SalesPersonID, SalesYear, SubTotal)
AS
(SELECT SalesPersonID, YEAR(OrderDate),
        SubTotal
   FROM Sales.SalesOrderHeader
   WHERE SalesPersonID IS NOT NULL)

SELECT *
  FROM SalesByYear
  PIVOT(SUM(SubTotal)
    FOR SalesYear
     IN ([2011], [2012], [2013], [2014]))
    AS TotalSales
  ORDER BY SalesPersonID
```

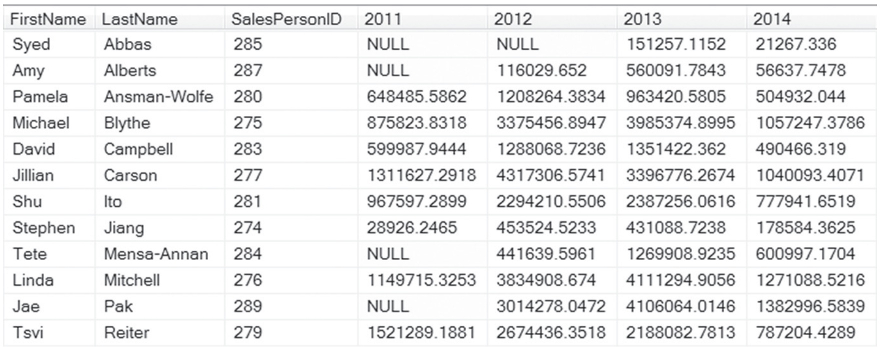


SalesPersonID	2011	2012	2013	2014
274	28926.2465	453524.5233	431088.7238	178584.3625
275	875823.8318	3375456.8947	3985374.8995	1057247.3786
276	1149715.3253	3834908.674	4111294.9056	1271088.5216
277	1311627.2918	4317306.5741	3396776.2674	1040093.4071
278	500091.8202	1283569.6294	1389836.8101	435948.9551
279	1521289.1881	2674436.3518	2188082.7813	787204.4289
280	648485.5862	1208264.3834	963420.5805	504932.044
281	967597.2899	2294210.5506	2387256.0616	777941.6519
282	1175007.4753	1835715.8705	1870884.182	1044810.8277
283	599987.9444	1288068.7236	1351422.362	490466.319
284	NULL	441639.5961	1269908.9235	600997.1704
285	NULL	NULL	151257.1152	21267.336
286	NULL	NULL	836055.1236	585755.8006
287	NULL	116029.652	560091.7843	56637.7478

Figure 4. Each row here represents one salesperson, while each column represents a year. The intersection shows the dollar total of sales for that salesperson for that year.

In this example, again, the CTE includes exactly three columns. One (SalesPersonID) determines the rows, one (SalesYear) determines the columns, and one (SubTotal) is aggregated to produce the data values.

Of course, this data would be more useful with the salespeople's names as well as their IDs. You can turn the query with the PIVOT into a CTE and add the names afterward, as in Listing 6 (which is included in this month's downloads as SalesPersonAnnualSalesWithNameCTE.SQL. Partial results are shown in Figure 5.



FirstName	LastName	SalesPersonID	2011	2012	2013	2014
Syed	Abbas	285	NULL	NULL	151257.1152	21267.336
Amy	Alberts	287	NULL	116029.652	560091.7843	56637.7478
Pamela	Ansman-Wolfe	280	648485.5862	1208264.3834	963420.5805	504932.044
Michael	Blythe	275	875823.8318	3375456.8947	3985374.8995	1057247.3786
David	Campbell	283	599987.9444	1288068.7236	1351422.362	490466.319
Jillian	Carson	277	1311627.2918	4317306.5741	3396776.2674	1040093.4071
Shu	Ito	281	967597.2899	2294210.5506	2387256.0616	777941.6519
Stephen	Jiang	274	28926.2465	453524.5233	431088.7238	178584.3625
Tete	Mensa-Annan	284	NULL	441639.5961	1269908.9235	600997.1704
Linda	Mitchell	276	1149715.3253	3834908.674	4111294.9056	1271088.5216
Jae	Pak	289	NULL	3014278.0472	4106064.0146	1382996.5839
Tsvi	Reiter	279	1521289.1881	2674436.3518	2188082.7813	787204.4289

Figure 5. Salesperson names are added to this pivoted result by putting the pivot into a CTE.

Listing 6. A query that uses PIVOT can be a CTE, so you can add more data.

```
WITH SalesByYear
  (SalesPersonID, SalesYear, SubTotal)
AS
(SELECT SalesPersonID, YEAR(OrderDate) ,
        SubTotal
   FROM Sales.SalesOrderHeader
   WHERE SalesPersonID IS NOT NULL),

SalesByYearPivot
AS
(SELECT *
   FROM SalesByYear
   PIVOT(SUM(SubTotal)
     FOR SalesYear
       IN ([2011], [2012], [2013], [2014]))
     AS TotalSales)

SELECT Person.FirstName, Person.LastName,
       SalesByYearPivot.*
  FROM SalesByYearPivot
    JOIN Person.Person
    ON SalesByYearPivot.SalesPersonID =
       Person.BusinessEntityID
  ORDER BY LastName, FirstName
```

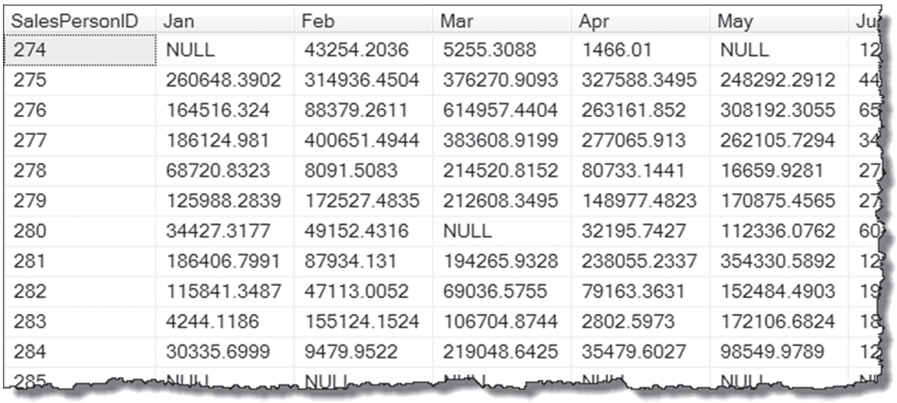
Getting meaningful column names

By default, the list you include in the IN portion of PIVOT determines the names of pivoted columns. So, in the sales example, the columns are called 2011, 2012, etc., while in the jobs example, they're the names of the countries. (This also explains why numeric values or values containing spaces need to be surrounded by square brackets; that's the standard way of referring to a column with a name that can't stand alone.)

However, you can actually specify alternative names for these columns in the field list of the query, just as you can for any field. The query in Listing 7 pulls sales data for one year and then pivots on month. The field list changes the names for those columns from the numeric month to the standard abbreviations. Figure 6 shows partial results, and the query is included as SalesPerson2013MonthlySalesWithMonthNames.SQL in this month's downloads.

Listing 7. You can rename pivoted columns in the field list of the query.

```
WITH SalesByMonth
AS
(SELECT SalesPersonID,
        MONTH(OrderDate) As SalesMonth,
        SubTotal
   FROM Sales.SalesOrderHeader
   WHERE SalesPersonID IS NOT NULL
     AND YEAR(OrderDate) = 2013)

SELECT SalesPersonID,
       [1] AS Jan, [2] AS Feb, [3] AS Mar,
       [4] AS Apr, [5] AS May, [6] AS Jun,
       [7] AS Jul, [8] AS Aug, [9] AS Sep,
       [10] AS Oct, [11] AS Nov, [12] AS Dec
  FROM SalesByMonth
  PIVOT(SUM(SubTotal)
    FOR SalesMonth
     IN ([1], [2], [3], [4], [5], [6], [7],
         [8], [9], [10], [11], [12]))
    AS TotalSales
  ORDER BY SalesPersonID;
```



SalesPersonID	Jan	Feb	Mar	Apr	May	Ju
274	NULL	43254.2036	5255.3088	1466.01	NULL	12
275	260648.3902	314936.4504	376270.9093	327588.3495	248292.2912	44
276	164516.324	88379.2611	614957.4404	263161.852	308192.3055	65
277	186124.981	400651.4944	383608.9199	277065.913	262105.7294	34
278	68720.8323	8091.5083	214520.8152	80733.1441	16659.9281	27
279	125988.2839	172527.4835	212608.3495	148977.4823	170875.4565	27
280	34427.3177	49152.4316	NULL	32195.7427	112336.0762	60
281	186406.7991	87934.131	194265.9328	238055.2337	354330.5892	12
282	115841.3487	47113.0052	69036.5755	79163.3631	152484.4903	19
283	4244.1186	155124.1524	106704.8744	2802.5973	172106.6824	18
284	30335.6999	9479.9522	219048.6425	35479.6027	98549.9789	12
285	NULL	NULL	NULL	NULL	NULL	NU

Figure 6. One year's sales were pivoted by month. Then, the field names were replaced by something more meaningful.

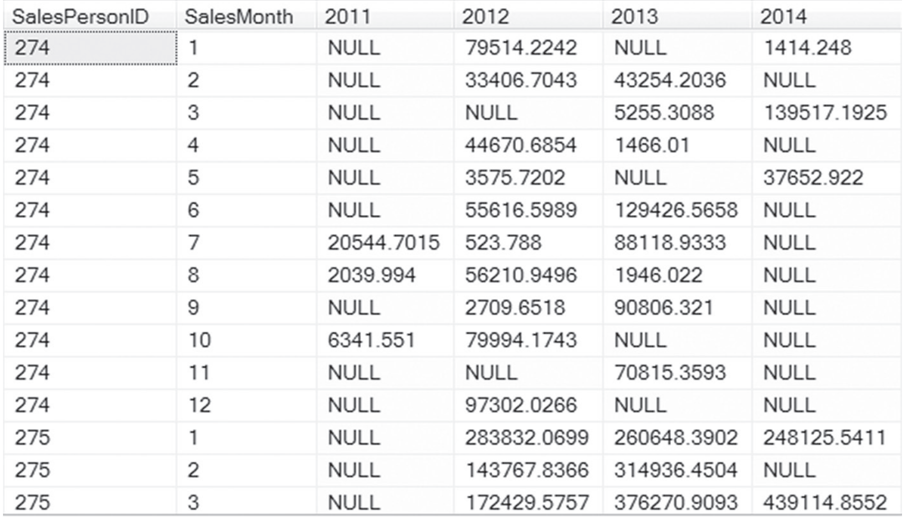
Determining rows by multiple columns

In the examples above, the set of rows was determined by a single field, JobTitle in the first case and SalesPersonID in the others. But it's possible to use multiple fields to specify the rows. All you have to do is have multiple columns in the query that aren't listed in the PIVOT clause.

For example, the query in Listing 8 has one row for each salesperson for each month. The CTE result has four fields: salesperson ID, month, year and invoice amount. The main query totals the invoice amount and specifies that year determines the columns. That leaves both salesperson ID and month to specify the rows. Partial results are shown in Figure 7. The query is included as SalesPersonMonthlySales.SQL in this month's downloads.

Listing 8.This query uses two fields (SalesPersonID and SalesMonth) to specify the rows in the pivoted result.

```
WITH csrSalesByYear
AS
(SELECT SalesPersonID,
        MONTH(OrderDate) As SalesMonth,
        YEAR(orderDate) AS SalesYear,
        SubTotal
   FROM Sales.SalesOrderHeader
   WHERE SalesPersonID IS NOT NULL)

SELECT *
  FROM csrSalesByYear
  PIVOT(SUM(SubTotal)
    FOR SalesYear
      IN ([2011], [2012], [2013], [2014]))
    AS TotalSales
  ORDER BY SalesPersonID, SalesMonth;
```



SalesPersonID	SalesMonth	2011	2012	2013	2014
274	1	NULL	79514.2242	NULL	1414.248
274	2	NULL	33406.7043	43254.2036	NULL
274	3	NULL	NULL	5255.3088	139517.1925
274	4	NULL	44670.6854	1466.01	NULL
274	5	NULL	3575.7202	NULL	37652.922
274	6	NULL	55616.5989	129426.5658	NULL
274	7	20544.7015	523.788	88118.9333	NULL
274	8	2039.994	56210.9496	1946.022	NULL
274	9	NULL	2709.6518	90806.321	NULL
274	10	6341.551	79994.1743	NULL	NULL
274	11	NULL	NULL	70815.3593	NULL
274	12	NULL	97302.0266	NULL	NULL
275	1	NULL	283832.0699	260648.3902	248125.5411
275	2	NULL	143767.8366	314936.4504	NULL
275	3	NULL	172429.5757	376270.9093	439114.8552

Figure 7. Here, the pivot result uses two columns to distinguish the rows.

This query also shows why it's generally easier to use SELECT * in a PIVOT. Otherwise, you need to list each pivoted column by name.

Aggregating on more than one column

A more complicated problem is computing more than one aggregate result. For example, suppose you want to get both total sales and the number of sales by year for each salesperson. You might think that you could simply list multiple aggregate functions after PIVOT, but that doesn't work.

In fact, to include multiple pivoted aggregations, you have to perform the pivots separately and then join the results. You also have to make sure that whatever you're selecting from contains only the columns relevant to that particular aggregation.

The easiest way to do this is with a series of CTEs, as in Listing 9. The first two CTEs, SalesByYear and SalesTotal, are the same as previous examples, producing one row per salesperson with one column per year. The final CTE, SalesCount, produces one row per salesperson with one column per year containing the number of orders for that salesperson in that year. Finally, the main query joins SalesTotal and SalesCount on SalesPersonID, including all the pivoted columns from each of them. Figure 8 shows partial results. This query is included as SalesPersonAnnualSalesMulti.SQL in this month's downloads.

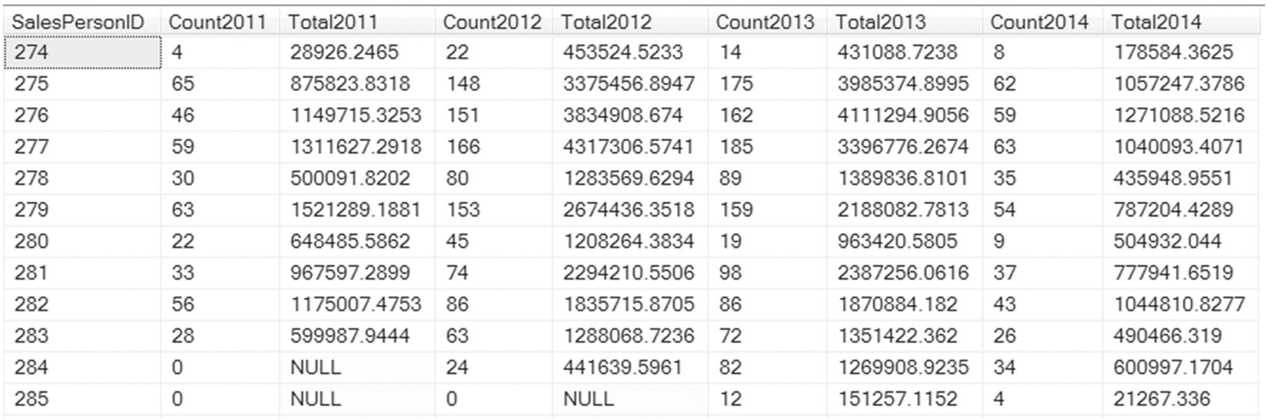
Listing 9. To pivot and aggregate on multiple columns, you have to do each pivot separately, and then join the results.

```
WITH SalesByYear
  (SalesPersonID, SalesYear, SubTotal)
AS
(SELECT SalesPersonID,
        YEAR(OrderDate), SubTotal
   FROM Sales.SalesOrderHeader
   WHERE SalesPersonID IS NOT NULL),

SalesTotal
AS
(SELECT SalesPersonID,
        [2011] AS Total2011,
        [2012] AS Total2012,
        [2013] AS Total2013,
        [2014] AS Total2014
  FROM SalesByYear
  PIVOT(SUM(SubTotal)
    FOR SalesYear
      IN ([2011], [2012], [2013], [2014]))
    AS TotalSales),

SalesCount
AS
(SELECT SalesPersonID,
        [2011] AS Count2011,
        [2012] AS Count2012,
        [2013] AS Count2013,
        [2014] AS Count2014
  FROM SalesByYear
  PIVOT(COUNT(SubTotal)
    FOR SalesYear
      IN ([2011], [2012], [2013], [2014]))
    AS Sales)

SELECT ST.SalesPersonID,
       SC.Count2011, ST.Total2011,
       SC.Count2012, ST.Total2012,
       SC.Count2013, ST.Total2013,
       SC.Count2014, ST.Total2014
  FROM SalesTotal ST
    JOIN SalesCount SC
      ON ST.SalesPersonID = SC.SalesPersonID
  ORDER BY ST.SalesPersonID
```

In my initial attempts at doing this (because, for some reason, I mistakenly thought that doing COUNT(Subtotal) would count only distinct values), I tried using a single CTE containing both Subtotal and SalesOrderID as the source for both pivots. However, even though the unneeded field was omitted from the field list of the queries performing the pivots, the field was still used in determining the rows of the result. Every field in the source table for a pivot is used either in determining rows, determining columns, or aggregation. The query in Listing 10 demonstrates the issue. The CTE includes SalesOrderID, though it's not mentioned in the main query. Nonetheless, the results (partially shown in Figure 9) have one row per sales order rather than one row per salesperson. This faulty query is included in this month's downloads as SalesPersonAnnualExtraField.SQL

Listing 10. Every field in the table specified for a pivot is used somehow. If it's not otherwise specified, it helps determine the list of rows.

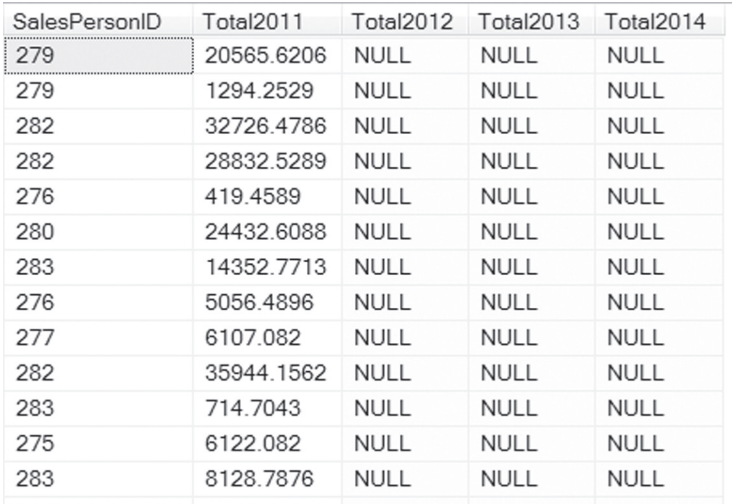
```
WITH SalesByYear
  (SalesPersonID, SalesYear,
   SubTotal, OrderID)
AS
(SELECT SalesPersonID, YEAR(OrderDate),
        SubTotal, SalesOrderID
   FROM Sales.SalesOrderHeader
   WHERE SalesPersonID IS NOT NULL)

SELECT SalesPersonID,
       [2011] AS Total2011,
       [2012] AS Total2012,
```



SalesPersonID	Count2011	Total2011	Count2012	Total2012	Count2013	Total2013	Count2014	Total2014
274	4	28926.2465	22	453524.5233	14	431088.7238	8	178584.3625
275	65	875823.8318	148	3375456.8947	175	3985374.8995	62	1057247.3786
276	46	1149715.3253	151	3834908.674	162	4111294.9056	59	1271088.5216
277	59	1311627.2918	166	4317306.5741	185	3396776.2674	63	1040093.4071
278	30	500091.8202	80	1283569.6294	89	1389836.8101	35	435948.9551
279	63	1521289.1881	153	2674436.3518	159	2188082.7813	54	787204.4289
280	22	648485.5862	45	1208264.3834	19	963420.5805	9	504932.044
281	33	967597.2899	74	2294210.5506	98	2387256.0616	37	777941.6519
282	56	1175007.4753	86	1835715.8705	86	1870884.182	43	1044810.8277
283	28	599987.9444	63	1288068.7236	72	1351422.362	26	490466.319
284	0	NULL	24	441639.5961	82	1269908.9235	34	600997.1704
285	0	NULL	0	NULL	12	151257.1152	4	21267.336

Figure 8. By joining the results of two separate pivots, we can do two different aggregations.

```
     [2013] AS Total2013,
     [2014] AS Total2014
FROM SalesByYear
PIVOT(SUM(SubTotal)
  FOR SalesYear
    IN ([2011], [2012], [2013], [2014]))
  AS TotalSales
```



SalesPersonID	Total2011	Total2012	Total2013	Total2014
279	20565.6206	NULL	NULL	NULL
279	1294.2529	NULL	NULL	NULL
282	32726.4786	NULL	NULL	NULL
282	28832.5289	NULL	NULL	NULL
276	419.4589	NULL	NULL	NULL
280	24432.6088	NULL	NULL	NULL
283	14352.7713	NULL	NULL	NULL
276	5056.4896	NULL	NULL	NULL
277	6107.082	NULL	NULL	NULL
282	35944.1562	NULL	NULL	NULL
283	714.7043	NULL	NULL	NULL
275	6122.082	NULL	NULL	NULL
283	8128.7876	NULL	NULL	NULL

Figure 9. Because the table used for this pivot includes SalesOrderID, the result has one row per sales order, rather than just one per salesperson.

But wait, there's more

In my next article, I'll look at how you can pivot when you don't know the list of values in the pivot column, as well as at the UNPIVOT command that gives you an easy way to normalize non-normalized data.

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.*