What is a CrossTab Query?

A cross tab query is a transformation of rows of data to columns. It usually involves aggregation of data e.g. totals broken down by months, products etc., where the months are represented by columns. It's very hard to visualize without an example, so we will provide one below.

Lets say you have a table of transactions that looks like this

|  |  |  |
| --- | --- | --- |
| **customer\_id** | **amount** | **purchase\_date** |
| 123 | 41.50 | 2004-10-01 |
| 12 | 30.50 | 2004-09-01 |
| 12 | 40.50 | 2004-08-15 |

One example of a cross-tab is to take the above data and show how much you earned in each month across. This allows for easy comparison of profits from quarter to quarter. You want your resulting data to look like the below.

|  |  |  |  |
| --- | --- | --- | --- |
| **m2004\_08** | **m2004\_09** | **m2004\_10** | **Total** |
| 40.50 | 30.50 | 41.50 | 112.50 |

How you get to the above representation varies from Database to Database. In the first example, we'll show you how to get there with standard ANSI-SQL tricks.

Standard ANSI-SQL Crosstab

SELECT

SUM(CASE WHEN purchase\_date BETWEEN '2004-08-01' and '2004-08-31' THEN amount ELSE 0 END) As m2004\_08,

SUM(CASE WHEN purchase\_date BETWEEN '2004-09-01' and '2004-09-30' THEN amount ELSE 0 END) As m2004\_09,

SUM(CASE WHEN purchase\_date BETWEEN '2004-10-01' and '2004-10-31' THEN amount ELSE 0 END) As m2004\_10,

SUM(amount) As Total

FROM purchases WHERE purchase\_date BETWEEN '2004-08-01' AND '2004-10-31'

Note the above assumes that your database handle dates using the ansi date format and uses ' for date delimiter. This varies from database type to database type as well as configuration settings in the database server. For example Microsoft Access uses # to delimit dates, while most databases use '

DBMS specific approaches

Microsoft Access

Microsoft Access has supported cross-tab queries natively for as far back as I can remember which would be Access 2.0. Access 1.0 may have supported it as well, but my memory doesn't go that far back.

Access supported via a predicate called TRANSFORM. If you are using the Access wizard, you can use the cross-tab wizard to automatically write the SQL statement for you.

The above query would look like

TRANSFORM Sum(purchases.amount) AS SumOfamount

SELECT Sum(purchases.amount) AS Total

FROM purchases

WHERE purchase\_date between #8/1/2004# and #10/31/2004#

PIVOT "m" & Format([purchase\_date],"yyyy\_mm") In ("m2004\_08","m2004\_09","m2004\_10")

Note: The **In** clause above is optional unless you plan to use this as a source for a report. What the **In** clause assures is that you always get the same consistent number of columns even if no data was present for say August. If you left it out and you had no data present for a specific month, then no column would be generated for that month.

Microsoft SQL Server 2005

In SQL Server 2005, the **PIVOT** and **UNPIVOT** predicates were introduced. The PIVOT predicate is a new way of doing crosstab queries that is very similar to MS Access Transform PIVOT approach. The code looks like the below

SELECT \*

FROM

(SELECT 'm' + CAST(DATEPART(year,[purchase\_date]) As varchar(4)) + '\_' + RIGHT('0' + CAST(DATEPART(month,[purchase\_date]) As varchar(2)), 2) as themonth, amount

FROM purchases

WHERE purchase\_date between '8/1/2004' and '10/31/2004' ) p

PIVOT (SUM(p.amount) FOR themonth In ([m2004\_08],[m2004\_09],[m2004\_10])) as pvt

**Problem**  
In SQL Server 2000 there was not a simple way to create cross-tab queries, but a new option in SQL Server 2005 has made this a bit easier. We took a look at how to create cross-tab queries in SQL Server 2000 in this [previous tip](http://www.mssqltips.com/tip.asp?tip=937) and in this tip we will look at this new feature in SQL Server 2005 to allow you produce cross-tab results.

**Solution**  
With SQL Server 2005 a lot of new features have been introduced. One of these new features is [**PIVOT**](http://msdn2.microsoft.com/en-us/library/ms177410.aspx).  What this allows you to do is to turn query results on their side, so instead of having results listed down like the listing below, you have results listed across.

|  |  |  |
| --- | --- | --- |
| **SalesPerson** | **Product** | **SalesAmount** |
| Bob | Pickles | $100.00 |
| Sue | Oranges | $50.00 |
| Bob | Pickles | $25.00 |
| Bob | Oranges | $300.00 |
| Sue | Oranges | $500.00 |

With a straight query the query results would be listed down, but the ideal solution would be to list the Products across the top for each SalesPerson, such as the following:

|  |  |  |
| --- | --- | --- |
| **SalesPerson** | **Oranges** | **Pickles** |
| Bob | $300.00 | $125.00 |
| Sue | $550.00 |  |

To use PIVOT you need to understand the data and how you want the data displayed.  First you have the data rows, such as SalesPerson and the columns, such as the Products and then the values to display for each cross section.  Here is a simple query that allows us to pull the cross-tab results.

|  |
| --- |
| SELECT SalesPerson, [Oranges] AS Oranges, [Pickles] AS Pickles FROM  (SELECT SalesPerson, Product, SalesAmount FROM ProductSales ) ps PIVOT ( SUM (SalesAmount) FOR Product IN ( [Oranges], [Pickles]) ) AS pvt |

***So how does this work?***

There are three pieces that need to be understood in order to construct the query.

* (1) The SELECT statement
  + **SELECT SalesPerson, [Oranges] AS Oranges, [Pickles] AS Pickles**
  + This portion of the query selects the three columns for the final result set (SalesPerson, Oranges, Pickles)

* (2) The query that pulls the raw data to be prepared
  + (**SELECT SalesPerson, Product, SalesAmount FROM ProductSales) ps**
  + This query pulls all the rows of data that we need to create the cross-tab results.  The (ps) after the query is creating a temporary table of the results that can then be used to satisfy the query for step 1.

* (3) The PIVOT expression
  + **PIVOT (SUM (SalesAmount) FOR Product IN ( [Oranges], [Pickles]) ) AS pvt**
  + This query does the actual summarization and puts the results into a temporary table called pvt

Another key thing to notice in here is the use of the square brackets [ ] around the column names in both the SELECT in part (1) and the IN in part (3).  These are key, because the pivot operation is treating the values in these columns as column names and this is how the breaking and grouping is done to display the data.

* **Introduction:**
* ***"Sometimes it is necessary to rotate results so that [the data in] columns are presented horizontally* *and [the data in]* *rows are* *presented vertically. This is known as creating a PivotTable®, creating a* *cross-tab report, or rotating* *data."***
* In other words, you can use a Cross Tab or Pivot to convert or transpose information from rows to columns either for reporting or to convert some special long skinny tables known as EAV's or NVP's into a more typical form data.
* The purpose of this article is to provide an introduction to Cross Tabs and Pivots and how they can be used to "rotate" data...
* **Before you say anything...**
* The reason I'm writing a series of articles on the simple concept of Cross Tabs and Pivots is because of the recent number of requests for this type of information on the SQL Server Central forums... there was a while when not a day went by when two or three such requests were posted each day.
* Also, yes, I aware that a lot of this type of "formatting" should be done in the GUI, reporting tool, or maybe even a Spreadsheet. I'm also aware that using EAV/NVP tables isn't considered to be a "best practice". But, like I said about the number of recent number of posts, folks get forced into a corner by their bosses and, if they have to do such a thing, I thought they could use a little help.
* **Notes of Interest:**
* I wrote all of the example code and data using Temp Tables just to be safe. Sure, I could have used Table Variables, but they don't really allow for people to do partial runs and they don't all people to look and see what's in the Table Variable after each section. Also, some of the data we'll end up using is a wee bit bigger than what I would normally use a table variable for.
* Last but not least, I currently only have SQL Server 2000 and 2005 installed. I indicate which rev each section of code will run on in parenthesis. I'm pretty sure that most of this will work on 2008 and that a good portion of the code for Cross Tabs will also work on 7... but I don't have access to either which means I haven't tested it.
* Also, for your convenience, all of the code has been attached in the "Resources" section near the end of the article.
* Ok... let's get started...
* **A simple introduction to Cross Tabs:**
* The Cross Tab Report example from Books Online is very simple and easy to understand. I've shamelessly borrowed from it to explain this first shot at a Cross Tab.
* **The Test Data**
* Basically, the table and data looks like this...
* --===== Sample data #1 (#SomeTable1)  
  --===== Create a test table and some data   
   CREATE TABLE #SomeTable1  
   (  
   Year SMALLINT,  
   Quarter TINYINT,   
   Amount DECIMAL(2,1)  
   )  
  GO  
   INSERT INTO #SomeTable1   
   (Year, Quarter, Amount)  
   SELECT 2006, 1, 1.1 UNION ALL  
   SELECT 2006, 2, 1.2 UNION ALL  
   SELECT 2006, 3, 1.3 UNION ALL  
   SELECT 2006, 4, 1.4 UNION ALL  
   SELECT 2007, 1, 2.1 UNION ALL  
   SELECT 2007, 2, 2.2 UNION ALL  
   SELECT 2007, 3, 2.3 UNION ALL  
   SELECT 2007, 4, 2.4 UNION ALL  
   SELECT 2008, 1, 1.5 UNION ALL  
   SELECT 2008, 3, 2.3 UNION ALL  
   SELECT 2008, 4, 1.9  
  GO

Every row in the code above is unique in that each row contains ALL the information for a given quarter of a given year. Unique data is NOT a requirement for doing Cross Tabs... it just happens to be the condition that the data is in. Also, notice that the 2nd quarter for 2008 is missing.

The goal is to make the data look more like what you would find in a spreadsheet... 1 row for each year with the amounts laid out in columns for each quarter with a grand total for the year. Kind of like this...

... and, notice, we've plugged in a "0" for the missing 2nd quarter of 2008.

Year 1st Qtr 2nd Qtr 3rd Qtr 4th Qtr Total   
------ ------- ------- ------- ------- -----   
2006 1.1 1.2 1.3 1.4 5.0  
2007 2.1 2.2 2.3 2.4 9.0  
2008 1.5 0.0 2.3 1.9 5.7

**The KEY to Cross Tabs!**

Let's start out with the most obvious... we want a Total for each year. This isn't required for Cross Tabs, but it will help demonstrate what the key to making a Cross Tab is.

To make the Total, we need to use the SUM aggregate and a GROUP BY... like this...

--===== Simple sum/total for each year  
 SELECT Year,  
 SUM(Amount) AS Total  
 FROM #SomeTable1  
 GROUP BY Year  
 ORDER BY Year

And, that returns the following...

Year Total   
------ ----------------------------------------   
2006 5.0  
2007 9.0  
2008 5.7

Not so difficult and really nothing new there. So, how do we "pivot" the data for the Quarter?

Let's do this by the numbers...

1. How many quarters are there per year? Correct, 4.
2. How many columns do we need to show the 4 quarters per year? Correct, 4.
3. How many times do we need the Quarter column to appear in the SELECT list to make it show up 4 times per year? Correct, 4.
4. Now, look at the total column... it gives the GRAND total for each year. What would we have to do to get it to give us, say, the total just for the first quarter for each year? Correct... we need a CASE statement inside the SUM.

Number 4 above is the KEY to doing this Cross Tab... It should be a SUM and it MUST have a CASE to identify the quarter even though each quarter only has 1 value. Yes, if each quarter had more than 1 value, this would still work! If any given quarter is missing, a zero will be substituted.

To emphasize, **each column for each quarter is just like the Total column, but it has a CASE statement to trap info only for the correct data for each quarter's column.** Here's the code...

--===== Each quarter is just like the total except it has a CASE  
 -- statement to isolate the amount for each quarter.  
 SELECT Year,  
 SUM(CASE WHEN Quarter = 1 THEN Amount ELSE 0 END) AS [1st Qtr],  
 SUM(CASE WHEN Quarter = 2 THEN Amount ELSE 0 END) AS [2nd Qtr],  
 SUM(CASE WHEN Quarter = 3 THEN Amount ELSE 0 END) AS [3rd Qtr],  
 SUM(CASE WHEN Quarter = 4 THEN Amount ELSE 0 END) AS [4th Qtr],  
 SUM(Amount) AS Total  
 FROM #SomeTable1  
 GROUP BY Year

... and that gives us the following result in the text mode (modified so it will fit here)...

Year 1st Qtr 2nd Qtr 3rd Qtr 4th Qtr Total  
------ ------- ------- ------- ------- -----  
2006 1.1 1.2 1.3 1.4 5.0  
2007 2.1 2.2 2.3 2.4 9.0  
2008 1.5 .0 2.3 1.9 5.7

Also notice... because there is only one value for each quarter, we could have gotten away with using MAX instead of SUM. Go ahead... try it. We'll use a similar method for normalizing an EAV table in the future.

For most applications, that's good enough. If it's supposed to represent the final output, we might want to make it a little prettier. The STR function inherently right justifies, so we can use that to make the output a little prettier. Please, no hate mail here! I'll be one of the first that formatting of this nature is supposed to be done in the GUI!

--===== We can use the STR function to right justify data and make it prettier.  
 -- Note that this should really be done by the GUI or Reporting Tool and  
 -- not in T-SQL  
 SELECT Year,  
 STR(SUM(CASE WHEN Quarter = 1 THEN Amount ELSE 0 END),5,1) AS [1st Qtr],  
 STR(SUM(CASE WHEN Quarter = 2 THEN Amount ELSE 0 END),5,1) AS [2nd Qtr],  
 STR(SUM(CASE WHEN Quarter = 3 THEN Amount ELSE 0 END),5,1) AS [3rd Qtr],  
 STR(SUM(CASE WHEN Quarter = 4 THEN Amount ELSE 0 END),5,1) AS [4th Qtr],  
 STR(SUM(Amount),5,1) AS Total  
 FROM #SomeTable1  
 GROUP BY Year

The code above gives us the final result we were looking for...

Year 1st Qtr 2nd Qtr 3rd Qtr 4th Qtr Total   
------ ------- ------- ------- ------- -----   
2006 1.1 1.2 1.3 1.4 5.0  
2007 2.1 2.2 2.3 2.4 9.0  
2008 1.5 0.0 2.3 1.9 5.7

**Just to emphasize what the very simple KEY to making a Cross Tab is... it's just like making a Total using SUM and Group By, but we've added a CASE statement to isolate the data for each Quarter.**

**A simple introduction to Pivots:**

Microsoft introduced the PIVOT function in SQL Server 2005. It works about the same (has some limitations) as a Cross Tab. Using the same test table we used in the Cross Tab examples above, let's see how to use PIVOT to do the same thing...

--===== Use a Pivot to do the same thing we did with the Cross Tab  
 SELECT Year, --(4)  
 [1] AS [1st Qtr], --(3)  
 [2] AS [2nd Qtr],  
 [3] AS [3rd Qtr],  
 [4] AS [4th Qtr],  
 [1]+[2]+[3]+[4] AS Total --(5)  
 FROM (SELECT Year, Quarter,Amount FROM #SomeTable1) AS src --(1)  
 PIVOT (SUM(Amount) FOR Quarter IN ([1],[2],[3],[4])) AS pvt --(2)  
 ORDER BY Year

Ok... let's break that code down and figure out what each part does... the items below have numbers in the code above so you can more easily see what's going on...

1. The FROM clause is actually a derived table. It very simply contains the columns that we want to use in the cross tab from the source table we want to use the pivot on. It will sometimes work as a normal FROM clause with just the table listed instead of a derived table, but most of the time it will not and is unpredictable when it does work.
2. This is the "Pivot" line. It identifies the aggregate to be used, the column to pivot in the FOR clause, and the list of values that we want to pivot in the IN clause... in this case, the quarter number. Also notice that you must treat those as if they were column names. They must either be put in brackets or double quotes (if the quoted identifier setting is ON).
3. This is the pivoted SELECT list. Notice that you have to bring everything in the IN clause from (2) up to the SELECT list. Aliasing the column names is optional but usually a good thing to do just to make the output obvious.
4. You must also bring Year up as the row identifier in the pivot. Think of this as your "anchor" for the rows.
5. Last but not least, if you want a total for each row in the pivot, you can no longer use just an aggregate. Instead, you must add all the columns together.

When you run the code, you get this...

Year 1st Qtr 2nd Qtr 3rd Qtr 4th Qtr Total  
------ ------- ------- ------- ------- -----  
2006 1.1 1.2 1.3 1.4 5.0  
2007 2.1 2.2 2.3 2.4 9.0  
2008 1.5 NULL 2.3 1.9 NULL

Notice the NULL's where there are no values or where a NULL has been added into a total. Remember that anything plus a NULL is still a NULL. All of this occurs because the Pivot doesn't do any substitutions like the Case statements we used in the Cross Tab. To fix this little problem, we have to use COALESCE (or ISNULL) on the columns... every bloody column! So, you end up with code that looks like this...

--===== Converting NULLs to zero's in the Pivot using COALESCE  
 SELECT Year,   
 COALESCE([1],0) AS [1st Qtr],  
 COALESCE([2],0) AS [2nd Qtr],  
 COALESCE([3],0) AS [3rd Qtr],  
 COALESCE([4],0) AS [4th Qtr],  
 COALESCE([1],0) + COALESCE([2] ,0) + COALESCE([3],0) + COALESCE([4],0) AS Total  
 FROM (SELECT Year, Quarter,Amount FROM #SomeTable1) AS src   
 PIVOT (SUM(Amount) FOR Quarter IN ([1],[2],[3],[4])) AS pvt   
 ORDER BY Year

That ***finally*** gives us the same result as a Cross Tab sans any right hand justification... again, you'd need to add the STR function to the code to do that.

Year 1st Qtr 2nd Qtr 3rd Qtr 4th Qtr Total  
------ ------- ------- ------- ------- -----  
2006 1.1 1.2 1.3 1.4 5.0  
2007 2.1 2.2 2.3 2.4 9.0  
2008 1.5 .0 2.3 1.9 5.7

**Readability Comparison**

Just for grins, here are both the Cross Tab and the Pivot code real close together so that you can do a comparison...

--===== The Cross Tab example   
 SELECT Year,  
 SUM(CASE WHEN Quarter = 1 THEN Amount ELSE 0 END) AS [1st Qtr],  
 SUM(CASE WHEN Quarter = 2 THEN Amount ELSE 0 END) AS [2nd Qtr],  
 SUM(CASE WHEN Quarter = 3 THEN Amount ELSE 0 END) AS [3rd Qtr],  
 SUM(CASE WHEN Quarter = 4 THEN Amount ELSE 0 END) AS [4th Qtr],  
 SUM(Amount) AS Total  
 FROM #SomeTable1  
 GROUP BY Year

--===== The Pivot Example  
 SELECT Year,   
 COALESCE([1],0) AS [1st Qtr],  
 COALESCE([2],0) AS [2nd Qtr],  
 COALESCE([3],0) AS [3rd Qtr],  
 COALESCE([4],0) AS [4th Qtr],  
 COALESCE([1],0) + COALESCE([2] ,0) + COALESCE([3],0) + COALESCE([4],0) AS Total  
 FROM (SELECT Year, Quarter,Amount FROM #SomeTable1) AS src   
 PIVOT (SUM(Amount) FOR Quarter IN ([1],[2],[3],[4])) AS pvt   
 ORDER BY Year

I'm sure that you'll have a preference, but I like the Cross Tab code better for two reasons... the Cross Tab code is simpler, in my eyes... all I have to remember how to do are those very simple Case statements, I only have to list the values of the pivot columns once, and I don't have to use COALESCE anywhere. The second reason is how simple it is to do a row total.

There's actually several other reasons and one of them is performance. We'll get to performance later, but first let's talk about...

**Multiple Aggregations In a Cross Tab (or, "The Problem with Pivots")**

We're going to do this section backwards from what we've been doing... we're going to cover how to Pivot multiple aggregations before we cover the equivalent Cross Tab.

A "multiple aggregation Pivot" is just that... we want to show two different aggregates in the Pivot something like this (notice both the Qty and Amt columns have been aggregated)...

Company Year Q1Amt Q1Qty Q2Amt Q2Qty Q3Amt Q3Qty Q4Amt Q4Qty TotalAmt TotalQty  
------- ------ ----- ----- ----- ----- ----- ----- ----- ----- -------- --------  
ABC 2006 1.1 2.2 1.2 2.4 1.3 1.3 1.4 4.2 5.0 10.1  
ABC 2007 2.1 2.3 2.2 3.1 2.3 2.1 2.4 1.5 9.0 9.0  
ABC 2008 1.5 5.1 0.0 0.0 2.3 3.3 1.9 4.2 5.7 12.6  
XYZ 2006 2.1 3.6 2.2 1.8 3.3 2.6 2.4 3.7 10.0 11.7  
XYZ 2007 3.1 1.9 1.2 1.2 3.3 4.2 1.4 4.0 9.0 11.3  
XYZ 2008 2.5 3.9 3.5 2.1 1.3 3.9 3.9 3.4 11.2 13.3

This type of Pivot is a common request so that both aggregates can be viewed for the same time period at the same time. Otherwise, you'd have two completely separate Pivots and you'd have to visually scan back and forth to make simple comparisons. As you'll see the "Problem with Pivots" is that each Pivot can only aggregate one column. To do something like this using Pivots, you have two use two Pivots.

**The Test Data**

Before we begin, we need some data to test with...

--===== Sample data #2 (#SomeTable2)  
--===== Create a test table and some data   
 CREATE TABLE #SomeTable2  
 (  
 Company VARCHAR(3),  
 Year SMALLINT,  
 Quarter TINYINT,   
 Amount DECIMAL(2,1),  
 Quantity DECIMAL(2,1)  
 )  
GO  
 INSERT INTO #SomeTable2  
 (Company,Year, Quarter, Amount, Quantity)  
 SELECT 'ABC', 2006, 1, 1.1, 2.2 UNION ALL  
 SELECT 'ABC', 2006, 2, 1.2, 2.4 UNION ALL  
 SELECT 'ABC', 2006, 3, 1.3, 1.3 UNION ALL  
 SELECT 'ABC', 2006, 4, 1.4, 4.2 UNION ALL  
 SELECT 'ABC', 2007, 1, 2.1, 2.3 UNION ALL  
 SELECT 'ABC', 2007, 2, 2.2, 3.1 UNION ALL  
 SELECT 'ABC', 2007, 3, 2.3, 2.1 UNION ALL  
 SELECT 'ABC', 2007, 4, 2.4, 1.5 UNION ALL  
 SELECT 'ABC', 2008, 1, 1.5, 5.1 UNION ALL  
 SELECT 'ABC', 2008, 3, 2.3, 3.3 UNION ALL  
 SELECT 'ABC', 2008, 4, 1.9, 4.2 UNION ALL  
 SELECT 'XYZ', 2006, 1, 2.1, 3.6 UNION ALL  
 SELECT 'XYZ', 2006, 2, 2.2, 1.8 UNION ALL  
 SELECT 'XYZ', 2006, 3, 3.3, 2.6 UNION ALL  
 SELECT 'XYZ', 2006, 4, 2.4, 3.7 UNION ALL  
 SELECT 'XYZ', 2007, 1, 3.1, 1.9 UNION ALL  
 SELECT 'XYZ', 2007, 2, 1.2, 1.2 UNION ALL  
 SELECT 'XYZ', 2007, 3, 3.3, 4.2 UNION ALL  
 SELECT 'XYZ', 2007, 4, 1.4, 4.0 UNION ALL  
 SELECT 'XYZ', 2008, 1, 2.5, 3.9 UNION ALL  
 SELECT 'XYZ', 2008, 2, 3.5, 2.1 UNION ALL   
 SELECT 'XYZ', 2008, 3, 1.3, 3.9 UNION ALL  
 SELECT 'XYZ', 2008, 4, 3.9, 3.4  
GO

**The Multi-Aggregate Pivot**

Like I said... we'll do the Pivot first this time... then we'll show you how easy it is to do using a Cross Tab.

In order to do a single Pivot, you have to have a derived table and a Pivot clause. The "Problem with Pivots" is that you can only Pivot one aggregate per Pivot clause. If you want to Pivot two aggregates as shown at the beginning of this section, you have to make two Pivots and join them as well as adding the necessary columns to the Select list. You already know how to use a single Pivot... Here's how we would do a double Pivot using the data above...

--===== The "Problem with Pivots" is you need to do one Pivot for each aggregate.  
 -- This code Pivots the Amt and Qty values by quarter.  
 SELECT amt.Company,  
 amt.Year,  
 COALESCE(amt.[1],0) AS Q1Amt,  
 COALESCE(qty.[1],0) AS Q1Qty,  
 COALESCE(amt.[2],0) AS Q2Amt,  
 COALESCE(qty.[2],0) AS Q2Qty,  
 COALESCE(amt.[3],0) AS Q3Amt,  
 COALESCE(qty.[3],0) AS Q3Qty,  
 COALESCE(amt.[4],0) AS Q4Amt,  
 COALESCE(qty.[4],0) AS Q4Qty,  
 COALESCE(amt.[1],0)+COALESCE(amt.[2],0)+COALESCE(amt.[3],0)+COALESCE(amt.[4],0) AS TotalAmt,  
 COALESCE(qty.[1],0)+COALESCE(qty.[2],0)+COALESCE(qty.[3],0)+COALESCE(qty.[4],0) AS TotalQty  
 FROM (SELECT Company, Year, Quarter, Amount FROM #SomeTable2) t1  
 PIVOT (SUM(Amount) FOR Quarter IN ([1], [2], [3], [4])) AS amt  
 INNER JOIN   
 (SELECT Company, Year, Quarter, Quantity FROM #SomeTable2) t2  
 PIVOT (SUM(Quantity) FOR Quarter IN ([1], [2], [3], [4])) AS qty  
 ON qty.Company = amt.Company   
 AND qty.Year = amt.Year   
 ORDER BY amt.Company, amt.Year

I don't know about you, but my personal feeling is that's starting to look a bit complicated and it's starting to be more difficult to read. Certainly, if we tried to convert this to dynamic SQL, you'd have your work cut out for you.

Notice that the FROM clause has two nearly identical derived tables and the ***only*** difference in the Pivot clauses are the columns being SUMmed. And, take a look at the row totals in the Select list... thank goodness this example only has 4 columns each for Quantity and Amount.

**The Multi-Aggregate Cross Tab**

We saw how complicated Multi-Aggregate Pivots can get. And the example above was just for two 4 column aggregates... just image what it might look like for three 12 column aggregates!

Let's see how complicated it might be in a Cross Tab...ready?

--===== Doing multiple aggregations in Cross Tabs is as simple as CPR  
 -- (CPR = Cut, Paste, Replace). AND, the table is "dipped" only  
 -- once instead of twice so there are NO JOINS to worry about!  
 SELECT Company,  
 Year,  
 SUM(CASE WHEN Quarter = 1 THEN Amount ELSE 0 END) AS Q1Amt,  
 SUM(CASE WHEN Quarter = 1 THEN Quantity ELSE 0 END) AS Q1Qty,  
 SUM(CASE WHEN Quarter = 2 THEN Amount ELSE 0 END) AS Q2Amt,  
 SUM(CASE WHEN Quarter = 2 THEN Quantity ELSE 0 END) AS Q2Qty,  
 SUM(CASE WHEN Quarter = 3 THEN Amount ELSE 0 END) AS Q3Amt,  
 SUM(CASE WHEN Quarter = 3 THEN Quantity ELSE 0 END) AS Q3Qty,  
 SUM(CASE WHEN Quarter = 4 THEN Amount ELSE 0 END) AS Q4Amt,  
 SUM(CASE WHEN Quarter = 4 THEN Quantity ELSE 0 END) AS Q4Qty,  
 SUM(Amount) AS TotalAmt,  
 SUM(Quantity) AS TotalQty  
 FROM #SomeTable2  
 GROUP BY Company, Year  
 ORDER BY Company, Year

How easy is that!? There're no derived tables... no fancy Pivot clauses... no huge lines of code to make simple row totals... and no joins!. It's a breeze to make using a little CPR (Copy, Paste, Replace).

Go back and compare the incredible simplicity of this Cross Tab with the relatively complex Pivot code to do the same thing. I don't know about you, but I won't be using Pivot to do such a simple thing.

**"Pre-Aggregation"**

I found something very handy in the past... I call it "Pre-Aggregation" and it can be used on either a Cross Tab or a Pivot.

The general purpose of pre-aggregation is to make it very easy to summarize the data and then format the data for display. Sometimes you'll have some complex aggregations that are a bit difficult or impossible to do when mixed with the rotation in the Select list, so the best thing to do is to do the aggregations as a derived table and *then* rotate the results. For example, if you want to aggregate dates by month, you'll find it's much easier to pre-aggregate the data using a formula to convert all dates to the first of the month. We'll cover more on that subject in the next article on Cross Tabs.

**Pre-aggregation is nothing more than doing the aggregation as part of a derived table and then doing a Cross Tab or Pivot on that result.** That's all it is.

You'll find pre-aggregation code for both Cross Tabs and Pivots in the next section of code where you'll also find another really good reason for doing pre-aggregation even if you don't need it to solve complexity...

**Performance**

Ah yes... what about performance? Just because the code looks simple or complex doesn't necessarily mean faster or slower nor fewer or more resources. Here's the full test code I used... I intentionally did NOT calculate Quarters from the date in the Cross Tabs or the Pivots because I wanted to show you just how much of a performance difference a simple tweak here and there can make... the biggest tweaks I made was the use of pre-aggregation and the use of CTE's...

--===== Create and populate a 1,000,000 row test table.  
 -- Column "RowNum" has a range of 1 to 1,000,000 unique numbers  
 -- Column "Company" has a range of "AAA" to "BBB" non-unique 3 character strings  
 -- Column "Amount has a range of 0.0000 to 9999.9900 non-unique numbers  
 -- Column "Quantity" has a range of 1 to 50,000 non-unique numbers  
 -- Column "Date" has a range of >=01/01/2000 and <01/01/2010 non-unique date/times  
 -- Columns Year and Quarter are the similarly named components of Date  
 -- Jeff Moden

SELECT TOP 1000000 --<<Look! Change this number for testing different size tables  
 RowNum = IDENTITY(INT,1,1),  
 Company = CHAR(ABS(CHECKSUM(NEWID()))%2+65)  
 + CHAR(ABS(CHECKSUM(NEWID()))%2+65)  
 + CHAR(ABS(CHECKSUM(NEWID()))%2+65),  
 Amount = CAST(ABS(CHECKSUM(NEWID()))%1000000/100.0 AS MONEY),  
 Quantity = ABS(CHECKSUM(NEWID()))%50000+1,  
 Date = CAST(RAND(CHECKSUM(NEWID()))\*3653.0+36524.0 AS DATETIME),  
 Year = CAST(NULL AS SMALLINT),  
 Quarter = CAST(NULL AS TINYINT)  
 INTO #SomeTable3  
 FROM Master.sys.SysColumns t1  
 CROSS JOIN  
 Master.sys.SysColumns t2

--===== Fill in the Year and Quarter columns from the Date column  
 UPDATE #SomeTable3  
 SET Year = DATEPART(yy,Date),  
 Quarter = DATEPART(qq,Date)

--===== A table is not properly formed unless a Primary Key has been assigned  
 -- Takes about 1 second to execute.  
 ALTER TABLE #SomeTable3  
 ADD PRIMARY KEY CLUSTERED (RowNum)

CREATE NONCLUSTERED INDEX IX\_#SomeTable3\_Cover1   
 ON dbo.#SomeTable3 (Company, Year)  
 INCLUDE (Amount, Quantity, Quarter)   
GO  
 SET STATISTICS TIME OFF  
 SET STATISTICS IO OFF  
GO  
---------------------------------------------------------------------------------------------------  
--===== "Normal" Cross Tab  
 PRINT REPLICATE('=',100)  
 PRINT '=============== "Normal" Cross Tab ==============='  
 SET STATISTICS IO ON  
 SET STATISTICS TIME ON

SELECT Company,  
 Year,  
 SUM(CASE WHEN Quarter = 1 THEN Amount ELSE 0 END) AS Q1Amt,  
 SUM(CASE WHEN Quarter = 1 THEN Quantity ELSE 0 END) AS Q1Qty,  
 SUM(CASE WHEN Quarter = 2 THEN Amount ELSE 0 END) AS Q2Amt,  
 SUM(CASE WHEN Quarter = 2 THEN Quantity ELSE 0 END) AS Q2Qty,  
 SUM(CASE WHEN Quarter = 3 THEN Amount ELSE 0 END) AS Q3Amt,  
 SUM(CASE WHEN Quarter = 3 THEN Quantity ELSE 0 END) AS Q3Qty,  
 SUM(CASE WHEN Quarter = 4 THEN Amount ELSE 0 END) AS Q4Amt,  
 SUM(CASE WHEN Quarter = 4 THEN Quantity ELSE 0 END) AS Q4Qty,  
 SUM(Amount) AS TotalAmt,  
 SUM(Quantity) AS TotalQty  
 FROM #SomeTable3  
 GROUP BY Company, Year  
 ORDER BY Company, Year

SET STATISTICS TIME OFF  
 SET STATISTICS IO OFF  
---------------------------------------------------------------------------------------------------  
--===== "Normal" Pivot  
 PRINT REPLICATE('=',100)  
 PRINT '=============== "Normal" Pivot ==============='  
 SET STATISTICS IO ON  
 SET STATISTICS TIME ON

SELECT amt.Company,  
 amt.Year,  
 COALESCE(amt.[1],0) AS Q1Amt,  
 COALESCE(qty.[1],0) AS Q1Qty,  
 COALESCE(amt.[2],0) AS Q2Amt,  
 COALESCE(qty.[2],0) AS Q2Qty,  
 COALESCE(amt.[3],0) AS Q3Amt,  
 COALESCE(qty.[3],0) AS Q3Qty,  
 COALESCE(amt.[4],0) AS Q4Amt,  
 COALESCE(qty.[4],0) AS Q5Qty,  
 COALESCE(amt.[1],0)+COALESCE(amt.[2],0)+COALESCE(amt.[3],0)+COALESCE(amt.[4],0) AS TotalAmt,  
 COALESCE(qty.[1],0)+COALESCE(qty.[2],0)+COALESCE(qty.[3],0)+COALESCE(qty.[4],0) AS TotalQty  
 FROM (SELECT Company, Year, Quarter, Amount FROM #SomeTable3) t1  
 PIVOT (SUM(Amount) FOR Quarter IN ([1], [2], [3], [4])) AS amt  
 INNER JOIN   
 (SELECT Company, Year, Quarter, Quantity FROM #SomeTable3) t2  
 PIVOT (SUM(Quantity) FOR Quarter IN ([1], [2], [3], [4])) AS qty  
 ON qty.Company = amt.Company   
 AND qty.Year = amt.Year   
 ORDER BY amt.Company, amt.Year

SET STATISTICS TIME OFF  
 SET STATISTICS IO OFF  
---------------------------------------------------------------------------------------------------  
--===== "Pre-aggregated" Cross Tab  
 PRINT REPLICATE('=',100)  
 PRINT '=============== "Pre-aggregated" Cross Tab ==============='  
 SET STATISTICS IO ON  
 SET STATISTICS TIME ON

SELECT Company,  
 Year,  
 SUM(CASE WHEN Quarter = 1 THEN Amount ELSE 0 END) AS Q1Amt,  
 SUM(CASE WHEN Quarter = 1 THEN Quantity ELSE 0 END) AS Q1Qty,  
 SUM(CASE WHEN Quarter = 2 THEN Amount ELSE 0 END) AS Q2Amt,  
 SUM(CASE WHEN Quarter = 2 THEN Quantity ELSE 0 END) AS Q2Qty,  
 SUM(CASE WHEN Quarter = 3 THEN Amount ELSE 0 END) AS Q3Amt,  
 SUM(CASE WHEN Quarter = 3 THEN Quantity ELSE 0 END) AS Q3Qty,  
 SUM(CASE WHEN Quarter = 4 THEN Amount ELSE 0 END) AS Q4Amt,  
 SUM(CASE WHEN Quarter = 4 THEN Quantity ELSE 0 END) AS Q4Qty,  
 SUM(Amount) AS TotalAmt,  
 SUM(Quantity) AS TotalQty  
 FROM (SELECT Company,Year,Quarter,SUM(Amount) AS Amount,SUM(Quantity) AS Quantity  
 FROM #SomeTable3 GROUP BY Company,Year,Quarter) d  
 GROUP BY Company, Year  
 ORDER BY Company, Year

SET STATISTICS TIME OFF  
 SET STATISTICS IO OFF  
---------------------------------------------------------------------------------------------------  
--===== "Pre-aggregated" Pivot  
 PRINT REPLICATE('=',100)  
 PRINT '=============== "Pre-aggregated" Pivot ==============='  
 SET STATISTICS IO ON  
 SET STATISTICS TIME ON

SELECT amt.Company,  
 amt.Year,  
 COALESCE(amt.[1],0) AS Q1Amt,  
 COALESCE(qty.[1],0) AS Q1Qty,  
 COALESCE(amt.[2],0) AS Q2Amt,  
 COALESCE(qty.[2],0) AS Q2Qty,  
 COALESCE(amt.[3],0) AS Q3Amt,  
 COALESCE(qty.[3],0) AS Q3Qty,  
 COALESCE(amt.[4],0) AS Q4Amt,  
 COALESCE(qty.[4],0) AS Q5Qty,  
 COALESCE(amt.[1],0)+COALESCE(amt.[2],0)+COALESCE(amt.[3],0)+COALESCE(amt.[4],0) AS TotalAmt,  
 COALESCE(qty.[1],0)+COALESCE(qty.[2],0)+COALESCE(qty.[3],0)+COALESCE(qty.[4],0) AS TotalQty  
 FROM (SELECT Company, Year, Quarter, SUM(Amount) AS Amount FROM #SomeTable3 GROUP BY Company, Year, Quarter) t1  
 PIVOT (SUM(Amount) FOR Quarter IN ([1], [2], [3], [4])) AS amt  
 INNER JOIN   
 (SELECT Company, Year, Quarter, SUM(Quantity) AS Quantity FROM #SomeTable3 GROUP BY Company, Year, Quarter) t2  
 PIVOT (SUM(Quantity) FOR Quarter IN ([1], [2], [3], [4])) AS qty  
 ON qty.Company = amt.Company   
 AND qty.Year = amt.Year   
 ORDER BY amt.Company, amt.Year

SET STATISTICS TIME OFF  
 SET STATISTICS IO OFF  
---------------------------------------------------------------------------------------------------  
--===== "Pre-aggregated" Cross Tab with CTE  
 PRINT REPLICATE('=',100)  
 PRINT '=============== "Pre-aggregated" Cross Tab with CTE ==============='  
 SET STATISTICS IO ON  
 SET STATISTICS TIME ON

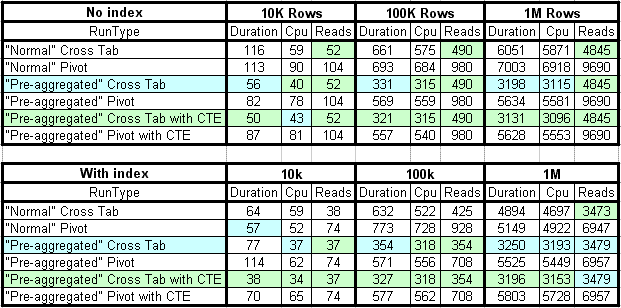
;WITH  
ctePreAgg AS  
(SELECT Company,Year,Quarter,SUM(Amount) AS Amount,SUM(Quantity) AS Quantity  
 FROM #SomeTable3   
 GROUP BY Company,Year,Quarter  
)  
 SELECT Company,  
 Year,  
 SUM(CASE WHEN Quarter = 1 THEN Amount ELSE 0 END) AS Q1Amt,  
 SUM(CASE WHEN Quarter = 1 THEN Quantity ELSE 0 END) AS Q1Qty,  
 SUM(CASE WHEN Quarter = 2 THEN Amount ELSE 0 END) AS Q2Amt,  
 SUM(CASE WHEN Quarter = 2 THEN Quantity ELSE 0 END) AS Q2Qty,  
 SUM(CASE WHEN Quarter = 3 THEN Amount ELSE 0 END) AS Q3Amt,  
 SUM(CASE WHEN Quarter = 3 THEN Quantity ELSE 0 END) AS Q3Qty,  
 SUM(CASE WHEN Quarter = 4 THEN Amount ELSE 0 END) AS Q4Amt,  
 SUM(CASE WHEN Quarter = 4 THEN Quantity ELSE 0 END) AS Q4Qty,  
 SUM(Amount) AS TotalAmt,  
 SUM(Quantity) AS TotalQty  
 FROM ctePreAgg  
 GROUP BY Company, Year  
 ORDER BY Company, Year

SET STATISTICS TIME OFF  
 SET STATISTICS IO OFF  
---------------------------------------------------------------------------------------------------  
--===== "Pre-aggregated" Pivot with CTE  
 PRINT REPLICATE('=',100)  
 PRINT '=============== "Pre-aggregated" Pivot with CTE ==============='  
 SET STATISTICS IO ON  
 SET STATISTICS TIME ON

;WITH  
ctePreAgg AS  
(SELECT Company,Year,Quarter,SUM(Amount) AS Amount,SUM(Quantity) AS Quantity  
 FROM #SomeTable3   
 GROUP BY Company,Year,Quarter  
)  
 SELECT amt.Company,  
 amt.Year,  
 COALESCE(amt.[1],0) AS Q1Amt,  
 COALESCE(qty.[1],0) AS Q1Qty,  
 COALESCE(amt.[2],0) AS Q2Amt,  
 COALESCE(qty.[2],0) AS Q2Qty,  
 COALESCE(amt.[3],0) AS Q3Amt,  
 COALESCE(qty.[3],0) AS Q3Qty,  
 COALESCE(amt.[4],0) AS Q4Amt,  
 COALESCE(qty.[4],0) AS Q5Qty,  
 COALESCE(amt.[1],0)+COALESCE(amt.[2],0)+COALESCE(amt.[3],0)+COALESCE(amt.[4],0) AS TotalAmt,  
 COALESCE(qty.[1],0)+COALESCE(qty.[2],0)+COALESCE(qty.[3],0)+COALESCE(qty.[4],0) AS TotalQty  
 FROM (SELECT Company, Year, Quarter, Amount FROM ctePreAgg) AS t1  
 PIVOT (SUM(Amount) FOR Quarter IN ([1], [2], [3], [4])) AS amt  
 INNER JOIN   
 (SELECT Company, Year, Quarter, Quantity FROM ctePreAgg) AS t2  
 PIVOT (SUM(Quantity) FOR Quarter IN ([1], [2], [3], [4])) AS qty  
 ON qty.Company = amt.Company   
 AND qty.Year = amt.Year   
 ORDER BY amt.Company, amt.Year

SET STATISTICS TIME OFF  
 SET STATISTICS IO OFF

The test code was executed 10 times each for 10k, 100k, and 1 million rows both with and without the index created at the beginning of the code. The averaged results, calculated from a profiler table (not included in the code), are fascinating. The light green cells indicate the fastest run times or the least number of reads. The light blue indicate the second place for the same thing...



http://www.sqlservercentral.com/Resources/Images/zoom.gif[Zoom in](javascript:;)  |  [Open in new window](javascript:;)

Notice that even for "normal" Cross Tabs and Pivots that **the only place a Pivot wins in any category is in the paltry 10k row test**. The Cross Tab wins everywhere else. That's good news for SQL Server 2000 users because you won't want to change your code if and when you upgrade to SQL Server 2005. Using CTE's helps a bit but not as much as pre-aggregation on the larger row counts does. Again, that's good news for SQL Server 2000 users.

**Review**

In this article, we learned the basis of how to change rows to columns using both Cross Tabs and Pivot. We've discovered that Cross Tabs are nothing more than simple aggregations that have a built in selection condition in the form of a simple Case statement. We've seen how to use a Pivot to do the same thing as a Cross Tab and, in the process, discovered that they're a bit more complicated to create, read, and understand especially when compared to the simplicity of the Cross Tab code. We've been introduced to the concept of "pre-aggregation and the fact that pre-aggregation can make more complex aggregations both easier to read and to contrive. Through testing, we've found that the Cross Tab beats Pivot code in all but the smallest of tables. Through that same testing, we've also found that pre-aggregation adds a substantial performance gain in all but the smallest of tables.

Last but certainly not least, we've discovered that there's no reason to rewrite properly written Cross Tabs to become Pivots when shifting from SQL Server 2000 to SQL Server 2005. To do so would actually cause a negative impact to performance most of the time.