Featherman’s TSQL Intermediate Functions #2 © (SELECT CASE, PIVOT, Temp Tables)

This document provides the next step for the student that has mastered selecting, filtering and aggregating data using GROUP BY functionality. While this document is referred to as Intermediate #2, it is not dependent on completing the content in Intermediate #1, While the GROUP BY() functionality is great to provide a base table to access from Excel (via a view) or SSRS (via a stored procedure), and in fact Excel’s pivot tables or SSRS’ matrix report will usually finish the job; it is also helpful to know how to aggregate scads of data into a compact tabular format. Two methodologies are shown to turn large amounts of data (many rows) into a compact table, SELECT CASE(), and PIVOT(). PIVOT() is similar to creating a pivot table in Excel.  
  
While the analyst may rely on their trusted pivot table when performing analysis, the DBA can perform this initial aggregation as a part of the extract, transform and load process. It is also worth noting that PIVOT() can work on very big data sets, which are too big for Excel (at time of this writing are limited to 2gb.

*Transforming Data from large tables with many rows to compact aggregated tables*

The heavy lifting of analysis occurs in the cleaning and preparation the data which largely occurs in the extract, transform and load process. The data is extracted from some large transaction system, transformed and often compiled and condensed as an intermediate step for subsequent processing and data visualization. While the traditional ETL usually copies transformed data into the data warehouse at the lowest level of granularity – at the detail record level – it is also common to build summary tables of compiled data. Historically crosstab queries have been a main enabler of this data compaction process. This document explores the cross-tabulation process with the older implementation CASE() statements and the newer PIVOT() statements. These cross-tabulation processes are amongst the most powerful and useful data transformations.

Pivot queries are new in SQL Server 2012. Previously you could create a crosstab query as follows:

|  |  |
| --- | --- |
| USE [AdventureWorksDW2012];  SELECT ps.[EnglishProductSubcategoryName]  , SUM(CASE WHEN MONTH([OrderDate]) = 1 THEN [SalesAmount] END) as [Jan]  , SUM(CASE WHEN MONTH([OrderDate]) = 2 THEN [SalesAmount] END) as [Feb]  , SUM(CASE WHEN MONTH([OrderDate]) = 3 THEN [SalesAmount] END) as [Mar]  , SUM(CASE WHEN MONTH([OrderDate]) = 4 THEN [SalesAmount] END) as [Apr]  , SUM(CASE WHEN MONTH([OrderDate]) = 5 THEN [SalesAmount] END) as [May]  , SUM(CASE WHEN MONTH([OrderDate]) = 6 THEN [SalesAmount] END) as [Jun]  , SUM(CASE WHEN MONTH([OrderDate]) = 7 THEN [SalesAmount] END) as [Jul]  , SUM(CASE WHEN MONTH([OrderDate]) = 8 THEN [SalesAmount] END) as [Aug]  , SUM(CASE WHEN MONTH([OrderDate]) = 9 THEN [SalesAmount] END) as [Sept]  , SUM(CASE WHEN MONTH([OrderDate]) = 10 THEN [SalesAmount] END) as [Oct]  , SUM(CASE WHEN MONTH([OrderDate]) = 11 THEN [SalesAmount] END) as [Nov]  , SUM(CASE WHEN MONTH([OrderDate]) = 12 THEN [SalesAmount] END) as [Dec]  FROM [dbo].[DimProduct] as p  INNER JOIN [dbo].[DimProductSubcategory] as ps  ON ps.[ProductSubcategoryKey] = p.[ProductSubcategoryKey]  INNER JOIN [dbo].[FactResellerSales] as rs ON rs.[ProductKey] = p.[ProductKey]  WHERE ps.[EnglishProductSubcategoryName] LIKE '%Bike%'  GROUP BY ps.[EnglishProductSubcategoryName]  ORDER BY ps.[EnglishProductSubcategoryName] | This methodology uses a SELECT CASE approach, which implements the cross tabulation (crosstab report).  Notice that the Case processing is performed within the context of a GROUP BY query.  This same process can be written more succinctly in different programming languages (shown in the Appendix), and was thus complemented in SQL SERVER 2012 by a new PIVOT() approach.  The structure of the code is that the first field in the SELECT statement is for the row headings that run down the first column. Here ProductSubcategory  The 12 subsequent calculated columns create a month column value for each of 12 months. Then each row in the fact table is being analyzed and grouped by Subcategory and month. When the month is 1 then the SalesAmount value is calculated and stored in a column called [Jan].  So this query runs 12 SUM(SalesAmount) queries within the context of the GROUP BY.  So ps.[EnglishProductSubcategoryName] is the row heading, [Jan], [Feb] etc. are the column headings, and the SUM(SalesAmount) phrase provides the total for each row and column intersection. |

|  |  |
| --- | --- |
| USE [AdventureWorksDW2012];  SELECT ps.[EnglishProductSubcategoryName]  , SUM(CASE WHEN [SalesTerritoryCountry] = 'United States' THEN [SalesAmount] END) as [USA]  , SUM(CASE WHEN [SalesTerritoryCountry] = 'Germany' THEN [SalesAmount] END) as [Germany]  , SUM(CASE WHEN [SalesTerritoryCountry] = 'Australia' THEN [SalesAmount] END) as [Australia]  , SUM(CASE WHEN [SalesTerritoryCountry] = 'France' THEN [SalesAmount] END) as [France]  , SUM(CASE WHEN [SalesTerritoryCountry] = 'United Kingdom' THEN [SalesAmount] END) as [UK]  FROM [dbo].[DimProduct] as p  INNER JOIN [dbo].[DimProductSubcategory] as ps  ON ps.[ProductSubcategoryKey] = p.[ProductSubcategoryKey]  INNER JOIN [dbo].[FactResellerSales] as rs ON rs.[ProductKey] = p.[ProductKey]  INNER JOIN [dbo].[DimSalesTerritory] as st on st.SalesTerritoryKey = rs.SalesTerritoryKey  GROUP BY ps.[EnglishProductSubcategoryName]  ORDER BY ps.[EnglishProductSubcategoryName] | Here’s another example. Again the first term in the SELECT statement defines the row headings, the subsequent case statements create calculated columns that will be displayed across the page. So 5 Case statements create 5 columns.  Here the SalesAmount field is totaled by country and product subcategory.  A similar approach will be displayed in SSRS, using a matrix control with the improvement of having a dynamic queried list of countries.  You can also make a live data connection in Excel to create a pivot table. |

Pivot()

The PIVOT operator lets you create results that resemble a crosstab report. The data across the page is retrieved using a TSQL SELECT statement that provides totals based on choosing one of the columns as column headers (such as customer, product or territory) and specifying the column to total for each combination of row heading and column heading. There are two main parts of a T-SQL Pivot query; a base query to retrieve the data and a PIVOT expression to specify the row and column headers used to aggregate the data.

One limitation of PIVOT queries is that they aren't dynamic, so all the column headers need to be hard-coded. If the pivoted column has values that vary over time, this part of the expression must be modified each time the data in the pivoted column changes. In this example, the column headers aren't likely to change because they're the months of the year. However, if a query compared sales by territories or sales by salespeople, the query would probably have to be modified frequently. Another option would be to write a stored procedure using dynamic SQL to determine the column headings.   
  
Finish by giving an alias to the PIVOT expression. This is required because the PIVOT functions results are treated as a table. So the PIVOT expression goes after the FROM clause and before the ORDER BY clause if there is one.  
  
Some content adapted from Kathi Kellenberger | SQL Server Pro <http://sqlmag.com/t-sql/create-pivoted-tables-3-steps>.  
Other info from:  
http://ishare-edu.blogspot.com/2012/11/t-sql-pivot-tables-in-sql-server-pivot.html  
Dynamic querying - supporting material at <http://sqlhints.com/2014/03/18/dynamic-pivot-in-sql-server/>

**Important Resources Training**1. Hands-on demonstration training video - [T-SQLPivots.mp4](http://www.cbe.wsu.edu/~mfeatherman/Common/T-SQLPivots.mp4)  
2. [Fantastic Blog Post](http://cooldata.wordpress.com/2014/02/23/really-swell-sql-why-you-must-know-pivot-and-with/)  - Using Pivot(), IN() and WITH()  
3. Web article – [Using PIVOT() and dynamic pivoting](http://stackoverflow.com/questions/19243821/pivot-transformation-using-t-sql)  
  
  
PIVOT () Queries – Here is the basic format of the SQL Statement. To start its best to have just 3 columns selected, row headings, column heading and field for aggregation.

|  |  |
| --- | --- |
| SELECT \* FROM  (SELECT Field For Row Headings, Field for Column Headings, Field to be aggregated)  FROM table WHERE Conditions)  AS Basetable  PIVOT  (SUM(field to be aggregated ) FOR field for column headings  IN (value1 from column heading to form a column, value2 from column heading to form a column, value3 from column heading to form a column, etc.)  ) AS PivotTable  Optional **ORDER BY** statement | Procedure 1. The first field selected is the field for the row headings (which go down the page on the left hand side  2. Select the field for the column headings which go across the page  3. Select the field that will be aggregated  4. Type AS tablename, then PIVOT, then write the  SUM(field) FOR column headings IN(parameters) Finish with AS Pivot table  *NOTE: It’s best to select only the three fields that you will use when you will use the PIVOT() command.* |
| USE AdventureWorks\_2012DW  SELECT \* FROM  (  SELECT Distinct [StateProvinceName] as State, pc.[EnglishProductCategoryName], [OrderQuantity]  FROM [dbo].[DimGeography] as g  INNER JOIN [dbo].[FactResellerSales] as rs  ON rs.SalesTerritoryKey = g.SalesTerritoryKey INNER JOIN [dbo].[DimProduct] as p  ON p.ProductKey = rs.ProductKey INNER JOIN [dbo].[DimProductSubcategory]as sc  ON sc.ProductSubcategoryKey = p.ProductSubcategoryKey  INNER JOIN [dbo].[DimProductCategory] as pc  ON pc.ProductCategoryKey = sc.ProductCategoryKey  WHERE g.[EnglishCountryRegionName] = 'United States'  )  AS basetable  PIVOT (SUM([OrderQuantity])  FOR [EnglishProductCategoryName]  IN (Bikes, Components, Clothing, Accessories )  )  AS Pivottable | Next we can turn it into a pivot table using the following guideline:  SELECT \* FROM  (SELECT Field For Row Headings, Field for Column Headings, Field to be aggregated)  FROM table  WHERE condition) AS Basetable  PIVOT  (SUM(field to be aggregated ) FOR field for column headings  IN (value1 from column heading to form a column, etc.) ) AS PivotTable  Notice that  a) the GROUP BY is removed b) the aggregate fields are removed c) it takes 3 joins to pull in the product category name |

|  |  |  |
| --- | --- | --- |
| Breakdown of Pivot Query |  | |
| SELECT \* FROM  (SELECT p.[EnglishProductName], [SalesTerritoryCountry], [SalesAmount]  FROM [dbo].[DimSalesTerritory] as st  INNER JOIN [dbo].[FactResellerSales] as rs  on rs.SalesTerritoryKey = st.SalesTerritoryKey  INNER JOIN [dbo].[DimProduct] as p on p.ProductKey = rs.[ProductKey] ) AS BaseTable | **First Step**: It's important that the base query include only the columns that will be needed in the final results. Any columns not pivoted or aggregated will end up as groupings, which might throw off the calculations. So any unnecessary columns may cause extraneous grouping levels and unexpected results. While you can have two row headings, a category and sub-category, this can obfuscate the data making analysis more difficult. This query has no WHERE filtering statement – which is optional. | |
| **PIVOT**  (SUM([SalesAmount] )  FOR [SalesTerritoryCountry] | **Second step**: The next step is to create the PIVOT expression. The first element in the PIVOT expression is an aggregate function. Often this function will be SUM or COUNT (MIN, MAX, AVG will also work), other user-defined functions can also be inserted. The parameter of the aggregate function is the name of the column to be aggregated. The function's results will show up under the pivoted columns.   After the aggregate function, you must type the keyword FOR followed by the name of the pivoted column. To determine the pivoted column, you need to figure out which column contains the values that you want displayed as column headers. In other words, which values that are currently displayed vertically do you want to display horizontally? | |
| IN (Canada, France, Australia, Germany, [United Kingdom], [United States]) )  AS PivotTable | The pivoted column's name is followed by an IN list that's similar to one found in a WHERE clause. This IN list serves two purposes. First, it restricts the rows that are pivoted (has aggregated columns calculated for it). Second, it supplies the pivoted column names. If the values that will end up as column names don't follow the rules for regular identifiers, they must be surrounded by brackets ([ ]). For example, if this example used month numbers instead of month names, you'd need to place each month number inside brackets. | |
|  | | This is the output from the SQL Pivot() query. You can see that the product names are the row headings down the page, the countries are the column headings which appear across the page. You can think of every cell in this cross-tabulated table as showing the results of its own SQL Select GROUP BY statement.  For example in the first row of data the SQL statement for Canada might read: **SELECT** EnglishProductName, **SUM**([SalesAmount]) as Total  **FROM** tablenames,  **WHERE** EnglishProductName = ‘AWC Logo Cap’ **AND** SalesTerritoryCountry = ‘Canada’ **GROUP BY** EnglishProductName |

So a PIVOT query returns a crosstab table, or call it a matrix table with many compiled aggregations. You can use a PIVOT() query to build a dataset for reporting or use it to perform a data transformation during an ETL process. If you are exploring data you can more easily use Excel. If you need to produce the analysis every week the Excel-based analyst might spend a good deal of time retrieving data, merging it, and formatting it and publishing it. A saved parameterized PIVOT() query rolled into a stored procedure and called from SSRS can produce a matrix report and chart with one button-click. The downfall of this automated query-report process? It is meant for stable, repetitive information needs and is not dynamic. Only Excel lets you easily flip the data, move and exchange the data values, and column names used to create the pivot table.

|  |  |
| --- | --- |
| ***More in-depth content and examples:***  Look at the table on the right. It’s just a few of the records from a table with > 50,000 records. Often retrieving filtered records that match a criteria is needed to enable analysis. But what if you want to see some totals or aggregates (averages, counts, KPI's, etc.)   What do you do to compile and aggregate a long table of > 50,000 records into a compact cross-table (crosstab) of summary numbers based on some grouping criteria that you specify?  Sure you can use a number of different exciting reporting software, but that requires the analyst to know how to use them.   You the BI specialist can provide the compiled, compacted data in a cross-tab or pivot table. This compacted table of numbers then can serve as the datasource for the reporting in Excel, or a reporting software. The compacted table of numbers can be created   a) 'on-the-fly' with a view (a stored query) b) calling parameterized query - providing a parameter c) you can save the aggregated table of values as a data table on the database server, and just point the spreadsheet or report against that datasource. d) you can use the SELECT query inside EXCEL or the reporting software  This webpage demonstrates this procedure and then extends the code to include parameterization and Visual Studio web form integration so that a reporting solution is provided. | Unprocessed Raw data records from a SQL Server data table http://www.cbe.wsu.edu/~mfeatherman/Common/images/imgC.jpg Compiled set of data - Sales for Region Grouped By Year http://www.cbe.wsu.edu/~mfeatherman/Common/images/imgB.jpg |
| ***Solution: T-SQL Pivot*** In 2005 a new T-SQL query construct was introduced by Microsoft to complement the slightly clumsy and verbose SELECT CASE methodology for creating a cross tab table. Two samples are shown. The first example to the right creates Pivot table shown above.   From [Microsoft Technet](http://technet.microsoft.com/en-us/library/ms177410(v=sql.105).aspx):  You can use the PIVOT relational operators to change a table-valued expression into another table.   PIVOT rotates a table-valued expression by turning the unique values from one column in the expression into multiple columns in the output, and performs aggregations where they are required on any remaining column values that are wanted in the final output.  Example 2--------------------------------------------------- You can flip the row and column headings pretty easily as follows to pivot the data. The pivoting is not dynamic however as in excel (at least not yet).  SELECT \* FROM ( SELECT Year, [Region], [Total\_Sale] FROM [featherman].[Sales] AS s ) AS DataTable   PIVOT ( SUM([Total\_Sale])  FOR Region IN ([North], [South], [East ], [West]) ) AS PivotTable ORDER BY Year | ---Example 1----------  SELECT \* FROM ( SELECT [Region], Year, [Total\_Sale] FROM [featherman].[Sales] AS s )  AS DataTable   PIVOT ( SUM([Total\_Sale])  FOR YEAR IN ([2007], [2008], [2009], [2010], [2011], [2012], [2013]) ) AS PivotTable  In this example, Region is the first pivoted column, Year is second pivoted column, Total\_Sale is the column with the data to be aggregated.   Three column configuration -  1) the column going down the page,  2) then the column going across the page,  3) then the column to be aggregated   This the most simple and common Pivot T-SQL routine  Year is the column that contains the values that will become column headers |
| ***Show me again how you use a T-SQL Pivot?***  Here is the format of the query  -- The base query gets the data to be later aggregated SELECT \* FROM  (SELECT <non-pivoted columns>, [<pivoted column>] AS <column name> FROM table with data) AS <alias for the base data table>  -- The PIVOT expression PIVOT ( <aggregation function>(<column being aggregated>) FOR [<column that contains the values that will become column headers>] IN ( [<first pivoted column>], [<second pivoted column>], ... [<last pivoted column>]) ) AS <alias for the pivoted table>  <optional ORDER BY clause>  The IN statement provides the filter and the values for the columns, unfortunately you have to type these. The columns are totaled for each value in the IN statement. Here they are hard coded in later we provide them via parameter. | |
| ***What if you need columns from another table can you use an Inner Join?***  Yes, you can join tables in different ways. This example adds the inner join to retrieve customer names from a second table. A count of the sales to each customer is provided across the specified years.   SELECT \* FROM  ( SELECT c.[CustomerName], Year, [Total\_Sale]  FROM [featherman].[Sales] AS s INNER JOIN [Customers] as c  ON c.[CustomerID] = s.[CustomerID] )  AS DataTable   PIVOT ( COUNT([Total\_Sale])  FOR YEAR IN ([2007], [2008], [2009], [2010], [2011], [2012], [2013]) )  AS PivotTable | http://www.cbe.wsu.edu/~mfeatherman/Common/images/img16.jpg |
| ***How do you sort a column based on a different column?***  This example is provided to show that if you just rearrange the columns in the first SELECT statement, then you specify a different column as the rows headings (months) and a different column as the columns to be aggregated (years).  USE [AdventureWorks];  SELECT \* FROM  (SELECT Month(OrderDate) AS [Month#] , DATENAME(MONTH,OrderDate) AS [MonthName] , YEAR(OrderDate) AS OrderYear, TotalDue FROM Sales.SalesOrderHeader ORDER BY [Month#]) AS DataTable  PIVOT (SUM(TotalDue)  FOR OrderYear IN([2001], [2002], [2003], [2004]) )  AS PivotTable  http://www.cbe.wsu.edu/~mfeatherman/Common/images/img21.jpg | As shown in the SSIS T-SQL Pivot transformation video - we must first aggregate the data using a GROUP BY query. Whereas the T-SQL Pivot query allows the base query to be unaggregated set of detail data rows, the SSIS PIVIT transformation needs the data to be grouped and sorted.   From Microsoft: "The Pivot transformation compacts an input data flow by pivoting it on a column value, making it less normalized.  The input data should be sorted by the pivot column because a pivot happens each time data in the pivot column changes. Note: duplicate rows will cause this transformation to fail."  USE [AdventureWorks]; SELECT Month(OrderDate) AS [Month#], DATENAME(MONTH,OrderDate) AS [MonthName] , YEAR(OrderDate) AS [OrderYear], SUM(TotalDue) AS [MonthTotal]  FROM Sales.SalesOrderHeader  GROUP BY Month(OrderDate), DATENAME(MONTH,OrderDate), YEAR(OrderDate) ORDER BY Month(OrderDate), YEAR(OrderDate) |

|  |  |
| --- | --- |
| ***Can you filter the results using a WHERE statement with a wildcard?***  USE [AdventureWorks];  select \* from ( select PS.Name, P.Color, PIn.Quantity from Production.Product P inner join Production.ProductSubcategory PS  ON PS.ProductSubcategoryID = P.ProductSubcategoryID  left join Production.ProductInventory PIn  ON P.ProductID = PIn.ProductID  WHERE PS.NAME like '%bikes%'  )  AS [Base DataTable]  PIVOT ( SUM(Quantity) FOR Color IN ([Black],[Blue],[Grey],[Multi],[Red] )  ) AS [Pivoted Table] | http://www.cbe.wsu.edu/~mfeatherman/Common/images/imgD.jpg |
| ***Can you parameterize and filter the results returned  from the PIVOT query? Can I supply the parameters from a webform or an SSRS report?*** You can use the query as the command text of the SQL data Adapter. The query then can accept parameters from the form. You can also save the PIVOT query as a stored procedure that returns the results set.  There is only one easy way to filter the data  1. In the WHERE clause of the base query  There is such a thing as dynamic pivots that can dynamically specify the columns for the IN clause (in green) but that requires heavy-duty programming best done in visual studio. | SELECT \* FROM  (SELECT column names from table 1,2 and 3 as a INNER JOIN table 2 b  ON = b.common field = a.common field  INNER JOIN table 3 as c  ON c.common field = a.common field  WHERE a.field = @p1 And b.field = @p2 )  AS SourceDataTable PIVOT  ( SUM(field to aggregate)  FOR [field to build columns and aggregate by column]  IN ([2005], [2006], [2007], [2008])  ) AS PivotedDataTable |

T-SQL Pivot vs. Pivot tables vs. GROUP BY queries

Summary

|  |  |  |  |
| --- | --- | --- | --- |
| Transformation | Usage | Advantages | Disadvantages |
| GROUP BY() | used to provide calculations based on groups (uses the GROUP BY query) | Can have many different aggregated values for any row (or dimension). Good to build result sets for reports & spreadsheets | GROUP BY will not solve all aggregation needs. A higher level of aggregation may be needed. |
|  |  |  |  |
| PIVOT() | Total, Sum, Count, Min, Max, ?? - used to aggregate large amounts of data into a crosstab format | Compacts a ton of data down into a small footprint table, which is the perfect source for a column or line chart. | Harder to parameterize. Typically can only have one aggregate |

|  |  |  |
| --- | --- | --- |
| ***Feature*** | ***GROUP BY Query*** | ***PIVOT Query*** |
| USE | Group BY queries can provide many columns of aggregated results such as counts, totals, averages. Moving averages, ranks, etc.) The aggregations can be calculated for different grouping levels, where the chosen grouping outputs one row of aggregated data per grouping of dimension attributes (such as geographic region, product line, employee, project, etc.).  If you have one level of grouping (say country) then the totals will be at a high level (CEO level). This minimal grouping may be visualized on an executive’s dashboard.  using If you have 2 or 3 levels of grouping then you have more rows of more detailed data (i.e., within each country you have customers grouped within cities, where one record per customer is calculated and provided in the dataset.  If you are creating a stored procedure or view which will provide a dataset of query results (aka resultset) for subsequent reporting, pivoting. You can also include fewer levels of grouping (lower granular data) in your query flatten out the dimension and show for example all levels of geographic data (continent, country, region) or levels of product groupings (product category, product line, product ID) | Pivot queries provide a static compacted table of results displaying ONE aggregated value in a tabular format. The aggregation for the ONE calculated value is based on two dimensions.  For example production units can be shown for month and machine Total Sales units sold can be totaled for city and product line |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | **GROUP BY()** | **PIVOT()** |
| Advantages | 1. Great tool to create new tables of compacted information - useful for subsequent reporting or analysis.  2. Can include many columns of aggregated values - useful for creation of multiple KPI columns that are be the datasource for a KPI dashboard  3. While data formatting usually occurs in the top level reporting software (excel, SSRS, Tableau) you can use the T-SQL FORMAT(SUM(FieldName),'N0') | 1. Great tool to compact a lot of data into one tabular format that looks and feels like a spreadsheet.   2. The data format is known and therefore easy for the analyst or manager to understand the data very quickly |
| Disadvantages | If you use more than one term in the GROUP BY clause you will receive duplicates in that column (down the page). Several columns of highly duplicate data (the first couple of columns for example) is hard to look at in the result set IF you are directly copying the resultset from SQL Server to an Excel sheet.  But if you use the result set as the data source for an Excel pivot table or SSRS report, then the duplicates can be hidden and the granularity actually calculations on different dimensions and collection of dimensions.. | 1. Typically you aggregate on one dimension down the rows and one dimension across the columns and one data metric (one sum or one count, etc.) displayed as the cells inside the resultset. The same calculation is performed on each cell of data for example a total by the column and the row values (i.e. east 2001, east 2002).  Adding a second dimension to the rows down the page is fine and useful, but also can be confusing for the report reader (SSRS conquers this problem with drill-down pivot reports, where you can show/hide columns and rows).  Adding a second metric in the cells may be possible, but then the table gets MUCH harder to read. Better to have two pivot tables.  2. If you need to group more than one value in a pivot table format – it’s easier to use Excel or SSRS. |
|  |  |  |

|  |  |
| --- | --- |
| ***FAQ:   Q1: Why not just use an Excel Pivot table - they are easier to deal with.***  ***Answer:*** Many times an Excel pivot table is the way to go, if you are the analyst interacting with the data *in a live OLAP session*. You might even build an Excel spreadsheet and post it to Sharepoint for an analyst or manager to interact with. Other times you build production reports and dashboards that are consumed by others in regular time intervals. In that instance you need the data to become automagically pre-aggregated into the tabular format, so that it can serve as the data source for production reports, and charts. Other times the PIVOT query is used in the SSIS process of building a data warehouse...wherein you implement several data transformation steps to aggregate the data for different levels of management.  WHY NOT JUST USE A GROUP BY QUERY?  Group by queries produce tall tables with many rows that you have to scroll down. Pivots give a more compact, spreadsheet-like response.  GROUP BY queries do have an advantage though, they allow you to aggregate different calculations for each group (here A COUNT, SUM AND AVERAGE). Pivots just aggregate one number.  Because so many columns and aggregations can be provided, GROUP BY queries are a good pre-processing vehicle which result in good base data for Pivot tables enabling further processing. | |
| USE [AdventureWorks2012];  SELECT YEAR(OrderDate) AS [OrderYear], Month(OrderDate) AS [Month#]  , DATENAME(MONTH,OrderDate) AS [Month] , COUNT([SalesOrderID]) AS [# Orders] , FORMAT(AVG(TotalDue), 'N0') AS [Avg. Sale for Month] , FORMAT(SUM(TotalDue),'N0') AS [Monthy Total] FROM Sales.SalesOrderHeader AS s  GROUP BY YEAR(OrderDate), Month(OrderDate), DATENAME(MONTH,OrderDate)  ORDER BY OrderYear, [Month#] | Sample Results of a GROUP BY Query http://www.cbe.wsu.edu/~mfeatherman/Common/images/img18.jpg |

|  |
| --- |
| Compare the format of the above GROUP BY query and the results of a T-SQL Pivot query, below. The main difference is that GROUP BY queries can give back hundreds of rows, and many columns, but PIVOTS give back more compacted data.  HOWEVER, the drawback is that only one number can be calculated for each cell (like a count or a sum).  You typically only use three fields in the SELECT statement 1) the row heading from a dimension table 2) the column heading from a different dimension and 3) the measure that is being aggregated. Take a look at the select statement below, and also notice there is no GROUP BY statement. The pivot means that the group by is being performed for each cell using the intersection of the row and column attributes to calculate each cell.  USE [AdventureWorks]; SELECT \* from  (SELECT YEAR(OrderDate) AS OrderYear, DATENAME(MONTH,OrderDate) AS [MonthName], TotalDue FROM Sales.SalesOrderHeader ) AS BaseDataTable  PIVOT (SUM(TotalDue)  FOR [MonthName] IN(January,February,March,April,May, June,July, August,September,October, November,December)  ) AS PivotTable  Similar results shown in this pivot table - but only sales are totaled by month and year, there are no counts or average sales for the month.  There aren’t 2 or 3 columns for each month.  http://www.cbe.wsu.edu/~mfeatherman/Common/images/img1F.jpg |

Appendix

This code example would show how a modern programming language would implement a SELECT CASE procedure

SELECT CASE MONTH([OrderDate])

CASE 1 THEN SUM([SalesAmount] as [Jan]  
CASE 2 THEN SUM([SalesAmount] as [Feb]  
CASE 3 THEN SUM([SalesAmount] as [Mar]  
…

END SELECT