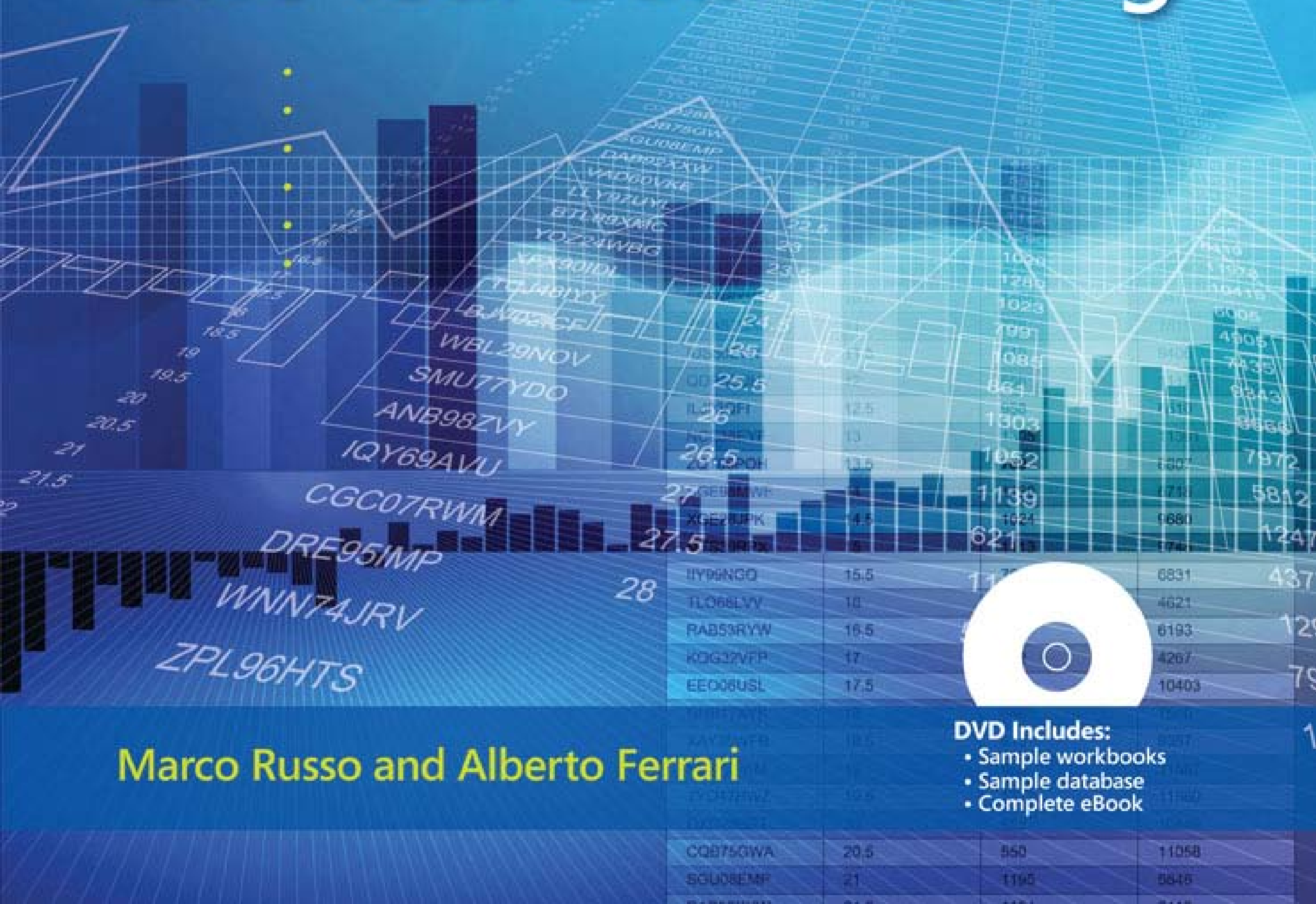
Microsoft

Microsoft®

PowerPivot for Excel® 2010

Give Your Data Meaning



Marco Russo and Alberto Ferrari

DVD Includes:
- Sample workbooks
- Sample database
- Complete eBook

Sample Chapters

To learn more about this book visit:
http://go.microsoft.com/fwlink/?Linkid=200417

Table of Contents

Chapter 2

PowerPivot at Work

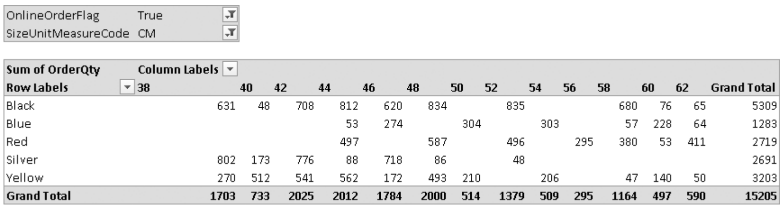
We are now going to introduce some of the most interesting features of Microsoft SQL Server PowerPivot for Excel. The goal of this chapter is to show the most frequently used PowerPivot features for transforming a simple Excel workbook into a complex report that helps you perform analysis on data. This is not yet the place for more advanced topics, such as the DAX programming language or complex relationships. Nevertheless, after you read this chapter, you will be able to perform complex analysis on a relational database and—we hope—still feel the need to go forward in your reading to discover the most advanced uses of PowerPivot.

Please note that we sometimes refer to *the end user* or *the user experience* as if we think that your PowerPivot workbook might be used by somebody else. To make a good report, you always need to think in this way. Even if you are the only user of a specific report, a user-friendly report is easier to read and update even after some time has passed since its creation.

Using the PivotTable to Produce Reports

Let us start with a very simple report, based on the same three tables that you loaded in the previous chapter: Sales Order Header, Sales Order Detail, and Product.

If you create a PivotTable with PowerPivot and put OnlineOrderFlag and SizeUnitMeasureCode on the Report Filter pane, Size on Column Labels, Color on Row Labels and the OrderQty as the value to sum up, you end up with the report shown in Figure 2-1, which you can find in the workbook named CH02-01-FirstSample.xlsx in the companion content.



OnlineOrderFlag	True
SizeUnitMeasureCode	CM

Sum of OrderQty	**Column Labels**													
Row Labels	**38**	**40**	**42**	**44**	**46**	**48**	**50**	**52**	**54**	**56**	**58**	**60**	**62**	**Grand Total**
Black	631	48	708	812	620	834		835			680	76	65	5309
Blue				53	274		304		303		57	228	64	1283
Red				497		587		496		295	380	53	411	2719
Silver	802	173	776	88	718	86		48						2691
Yellow	270	512	541	562	172	493	210		206		47	140	50	3203
Grand Total	**1703**	**733**	**2025**	**2012**	**1784**	**2000**	**514**	**1379**	**509**	**295**	**1164**	**497**	**590**	**15205**

FIGURE 2-1 A simple report using PowerPivot.

Before analyzing more advanced features, let us recall briefly what is going on:

- The OnLineOrderFlag, coming from the SalesOrderHeader table, is a TRUE/FALSE value. PowerPivot found only two possible values for it, so it has been able to fill the combo box of the filter with the values True and False. By choosing True, you selected only the orders placed online.
- The same process happened for the SizeUnitMeasureCode, this time coming from the Product table. It can contain only two distinct values (empty and CM). You have selected CM as the measure unit for the size.
- Placing Color on the rows, Size on the rows, and finally OrderQty as the value, PowerPivot analyzed all the rows containing the value (which is OrderQty, contained in the SalesOrderDetail table). Then it followed the relationship between Order Detail and Products and filtered out all the rows from the detail that do not satisfy the filter condition. In the meantime, it removed all the rows that do not satisfy the condition on the OrderHeader table, which contains the OnlineOrderFlag.
- Having detected the set of rows that you want to analyze, PowerPivot followed the relationship between SalesOrderDetail and Product to find out the color and the size of each product sold, summarized up all the quantities, and displayed the final PivotTable.

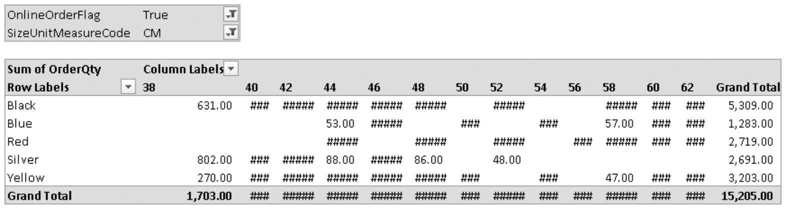
Do not worry if the process described here is not perfectly clear; it will become easier to understand as you continue reading, thanks to the many examples we are going to provide. But remember this important point: the presence of relationships is essential for PowerPivot to detect the set of rows it must take into account from the source tables.

Important You can easily understand why relationships are an important concept in PowerPivot when you remember this significant difference between PowerPivot and the classic Excel PivotTable: the older tool analyzes only one table and so it does not need to relate it with anything else—its analysis is carried on a single object. On the other hand, PowerPivot can analyze more than one table at a time but to do that, it needs to relate the tables to produce useful results.

In the next chapters, we spend several pages in the analysis of different kinds of relationships and how to master them. Nevertheless, before diving into complex analysis, let us solve some minor problems in this sample report to make it more appealing and a smoother introduction to all of the PowerPivot features.

Formatting Numbers

Even if the report shown in Figure 2-1 contains interesting information, it has a problem: it lacks a format for numbers. In Excel worksheets, the formatting of numbers is one of the functions of the worksheet itself. So, to format the numbers properly, you select the data area of the report and choose a proper formatting. If you follow this procedure in a PivotTable, the first result is not very appealing, as you can see in Figure 2-2.



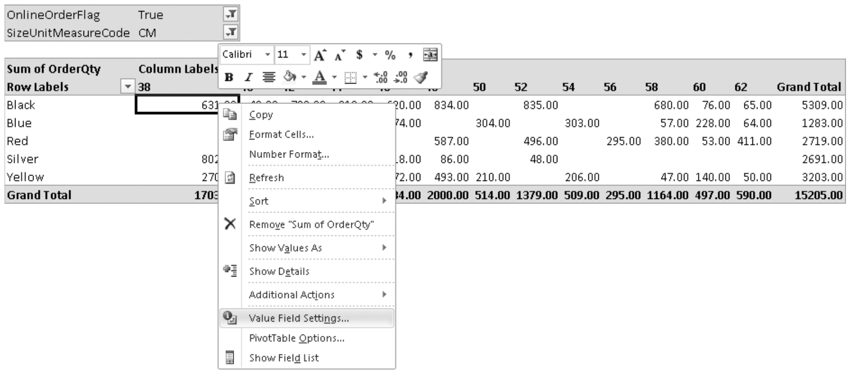
OnlineOrderFlag	True
SizeUnitMeasureCode	CM

Sum of OrderQty	Column Labels													
Row Labels	38	40	42	44	46	48	50	52	54	56	58	60	62	Grand Total
Black	631.00	###	#####	#####	#####	#####		#####			#####	###	###	5,309.00
Blue				53.00	#####		###		###		57.00	###	###	1,283.00
Red				#####		#####		#####		###	#####	###	###	2,719.00
Silver	802.00	###	#####	88.00	#####	86.00		48.00						2,691.00
Yellow	270.00	###	#####	#####	#####	#####	###		###		47.00	###	###	3,203.00
Grand Total	**1,703.00**	**###**	**#####**	**#####**	**#####**	**#####**	**###**	**#####**	**###**	**###**	**#####**	**###**	**###**	**15,205.00**

FIGURE 2-2 Wrong display of numbers if format strings are applied to the PivotTable.

Because you applied the formatting after you created the PivotTable, none of the columns were large enough to accommodate the new representation of numbers, which now contain dots and commas, resulting in larger columns. You can solve this easily by resizing all the columns. Nevertheless, if you decide to change the measure displayed and use a different one (for example, ListPrice), you probably need a different format and different column sizes, and you probably have to resize the entire worksheet.

The correct procedure to follow is to use the PivotTable field settings to define a number format for the OrderQty column. To perform this, you can right-click inside a cell containing the OrderQty value and, from the menu, choose Value Field Settings, as you can see in Figure 2-3.

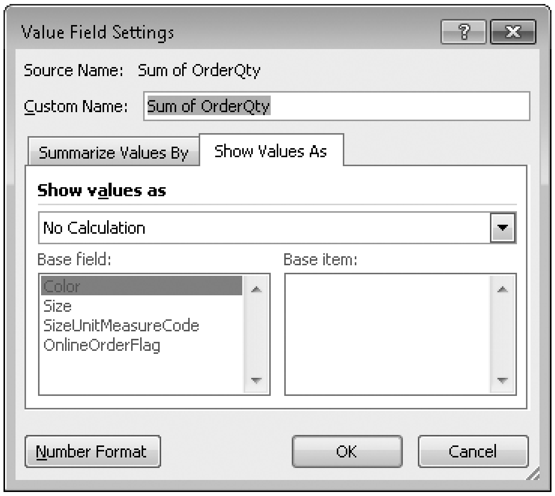


OnlineOrderFlag	True
SizeUnitMeasureCode	CM

Sum of OrderQty	Column Labels											
Row Labels	38			50	52	54	56	58	60	62	Grand Total
Black	631	20.00	834.00		835.00			680.00	76.00	65.00	5309.00
Blue		74.00		304.00		303.00		57.00	228.00	64.00	1283.00
Red			587.00		496.00		295.00	380.00	53.00	411.00	2719.00
Silver	802	8.00	86.00		48.00						2691.00
Yellow	270	72.00	493.00	210.00		206.00		47.00	140.00	50.00	3203.00
Grand Total	**1703**	**4.00**	**2000.00**	**514.00**	**1379.00**	**509.00**	**295.00**	**1164.00**	**497.00**	**590.00**	**15205.00**

FIGURE 2-3 The Value Field Settings menu.

This option opens the Value Field Settings dialog box, shown in Figure 2-4, which contains many options. We are interested, for now, only in the number format, which you can view by clicking the Number Format button.


Value Field Settings

Source Name: Sum of OrderQty

Custom Name: Sum of OrderQty

Summarize Values By | Show Values As

Show values as

No Calculation

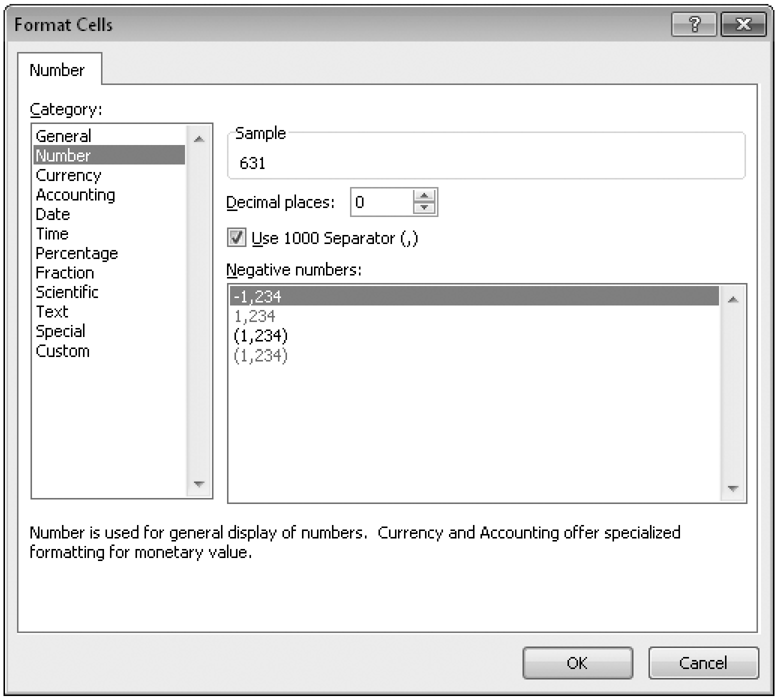
Base field: Color, Size, SizeUnitMeasureCode, OnlineOrderFlag

Base item:

Number Format | OK | Cancel

FIGURE 2-4 The Value Field Settings dialog box.

The Format Cells dialog box (see Figure 2-5) lets you choose a number format for the column in this PivotTable.


Format Cells

Number

Category: General, Number, Currency, Accounting, Date, Time, Percentage, Fraction, Scientific, Text, Special, Custom

Sample: 631

Decimal places: 0

[✓] Use 1000 Separator (,)

Negative numbers:

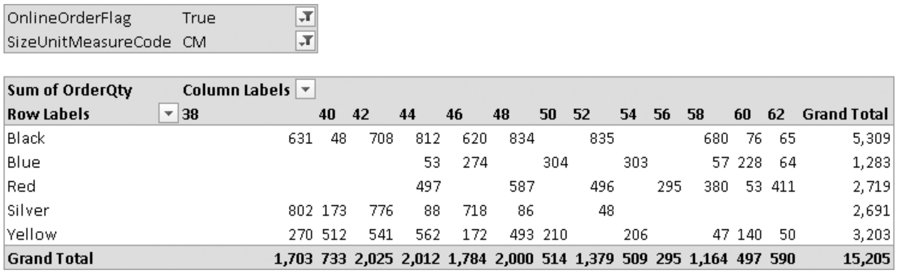
-1,234
1,234
(1,234)
(1,234)

Number is used for general display of numbers. Currency and Accounting offer specialized formatting for monetary value.

OK | Cancel

FIGURE 2-5 The Format Cells dialog box.

When you choose the number format you want (in this case, we have selected a number format with a thousand separator and no decimal places), the PivotTable resizes all the columns automatically, as you can see in Figure 2-6.



OnlineOrderFlag	True
SizeUnitMeasureCode	CM

Sum of OrderQty	Column Labels													
Row Labels	38	40	42	44	46	48	50	52	54	56	58	60	62	Grand Total
Black	631	48	708	812	620	834		835			680	76	65	5,309
Blue				53	274		304		303		57	228	64	1,283
Red				497		587		496		295	380	53	411	2,719
Silver	802	173	776	88	718	86		48						2,691
Yellow	270	512	541	562	172	493	210		206		47	140	50	3,203
Grand Total	**1,703**	**733**	**2,025**	**2,012**	**1,784**	**2,000**	**514**	**1,379**	**509**	**295**	**1,164**	**497**	**590**	**15,205**

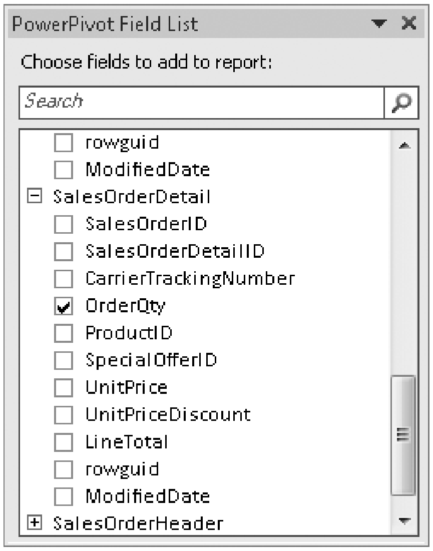
FIGURE 2-6 The PivotTable correctly resized.

This procedure applies number formatting to the current PivotTable only. If the same column is used somewhere else in other PivotTables, your choice in this PivotTable does not affect them.

Please note that, if you change the measure shown, you must repeat the procedure to determine the number format of the new column used.

Hiding or Removing Useless Columns

In the process of making the report more user-friendly, you can now focus on another small problem: the PowerPivot Field List (see Figure 2-7), from which you choose values to put on rows and columns, shows all the columns of all the PowerPivot tables. You see a large set of columns, many of which are not really useful.



PowerPivot Field List

Choose fields to add to report:

Search

- [ ] rowguid
- [ ] ModifiedDate
- ⊟ SalesOrderDetail
  - [ ] SalesOrderID
  - [ ] SalesOrderDetailID
  - [ ] CarrierTrackingNumber
  - [✓] OrderQty
  - [ ] ProductID
  - [ ] SpecialOfferID
  - [ ] UnitPrice
  - [ ] UnitPriceDiscount
  - [ ] LineTotal
  - [ ] rowguid
  - [ ] ModifiedDate
- ⊞ SalesOrderHeader

FIGURE 2-7 The field selector.

Although it is certainly useful to see everything during the process of data discovery, several columns distract us rather than help us. Let us see a couple of examples:

- The column SalesOrderID in SalesOrderDetail is very useful because it makes the relationship between SalesOrderDetail and SalesOrderHeader. Nevertheless, selecting it for reporting purposes produces no useful result. We refer to this kind of column as a technical column—that is, a column mandatory for the data model to work but that has no meaning at all for the reports.
- The columns rowguid and ModifiedDate, in the SalesOrderDetail table, are columns used by the source system that handles the database, but they do not contain any useful information either from the technical or from the reporting point of view. We refer to these columns as useless columns because you can easily remove them without affecting the data model.

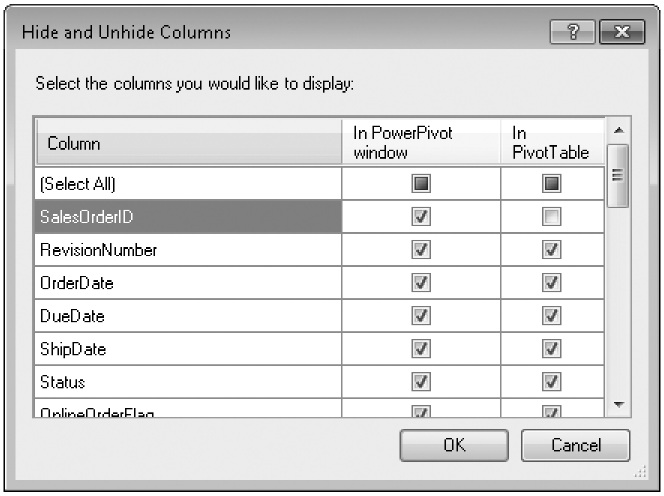
Note Please note that useless columns are not simply columns that you believe are of no use in the current report. Columns like SpecialOfferID, which is a technical column, contain interesting information that seems not to be useful at a certain point in time. If you plan to use the same source for many reports, you need to think twice before tagging a column as useless and maybe deleting it. But even if you remove a column from the PowerPivot table, you can always reload it later by changing the table properties.

To make the user experience better, you should hide technical columns and remove useless ones so that the names shown in the field selector refer only to columns that can be used in the report to provide useful results. When you remove a column from a table, that column is physically deleted from the PowerPivot data model, is no longer available for any operation, and reduces the memory and disk space taken up by the table. On the other hand, when you hide a column, it still exists in the PowerPivot data model, although the user cannot select it in a report. It is now clear why we choose to hide technical columns (if we remove them, we would not be able to use them for relationship, for instance) and remove useless ones.

To perform this task, you can open the Hide And Unhide Columns dialog box, which you can see in Figure 2-8, from the Design tab on the ribbon of the PowerPivot window. From here, you can choose to hide or show any columns of the table.

If you choose to hide a column in PowerPivot, it does not appear in the PowerPivot tables but is still available in the PowerPivot Field List. This decision might help you achieve a cleaner view of data when you are browsing in the PowerPivot window.

On the other hand, we are not interested in hiding any columns from the PowerPivot window because we want to browse all data. We hide columns only in the PowerPivot Field List.



Hide and Unhide Columns

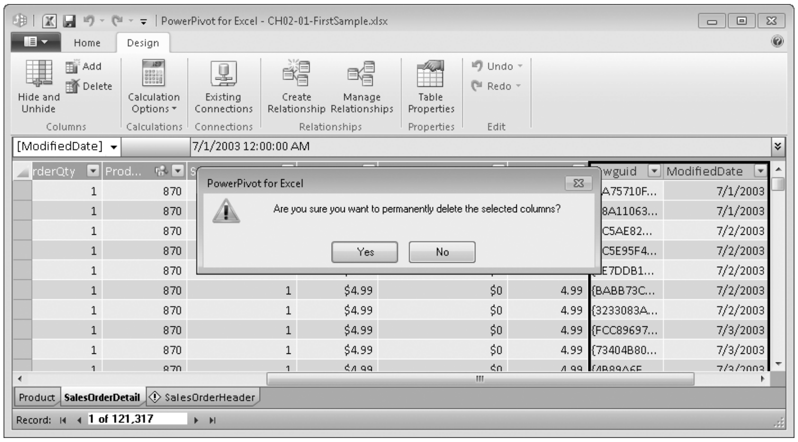
Select the columns you would like to display:

Column	In PowerPivot window	In PivotTable
(Select All)	■	■
SalesOrderID	✓	☐
RevisionNumber	✓	✓
OrderDate	✓	✓
DueDate	✓	✓
ShipDate	✓	✓
Status	✓	✓
OnlineOrderFlag	✓	✓

FIGURE 2-8 Hide and unhide columns.

Note As you might have noted, there is no option that allows you to hide a full table. The PowerPivot Field List hides a table if all of its columns are hidden in the PivotTable. Even if hiding a table might seems useless at this point, later on you will discover several data models that contain not only technical columns but technical tables too. For those models, the option to completely hide a table is useful because it lets you hide the complexity of the model when you are browsing data.

After you have hidden technical columns, you still need to delete the useless ones, which are, in our example, the two columns rowguid and ModifiedDate. To do that, you need to select the column in the PowerPivot window and click the Delete button on the Design tab of the ribbon. After you click it, a confirmation message box appears, like the one in Figure 2-9.



[ModifiedDate] 7/1/2003 12:00:00 AM

rderQty	Prod...	S				wguid	ModifiedDate
1	870					A75710F...	7/1/2003
1	870					8A11063...	7/1/2003
1	870					C5AE82...	7/2/2003
1	870					C5E95F4...	7/2/2003
1	870					E7DDB1...	7/2/2003
1	870	1	$4.99	$0	4.99	{BABB73C...	7/2/2003
1	870	1	$4.99	$0	4.99	{3233083A...	7/2/2003
1	870	1	$4.99	$0	4.99	{FCC89697...	7/3/2003
1	870	1	$4.99	$0	4.99	{73404B80...	7/3/2003
1	870	1	$4.99	$0	4.99	{4B89A6F	7/3/2003

PowerPivot for Excel

Are you sure you want to permanently delete the selected columns?

Yes No

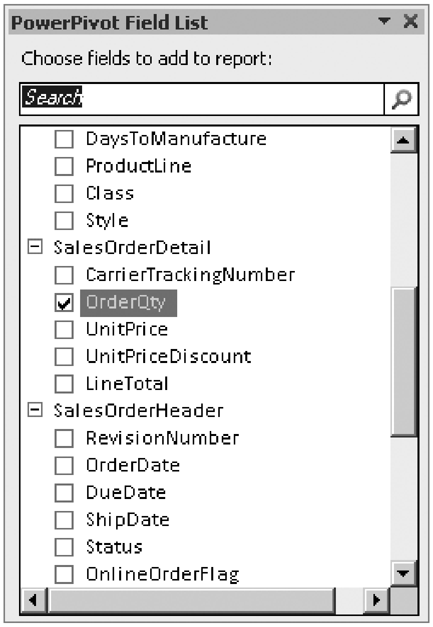
Record: 1 of 121,317

FIGURE 2-9 When you delete columns, PowerPivot raises a message box.

When you delete a column, it is no longer available for any tasks inside PowerPivot. You can still add it back later, if you edit table properties, but this effort requires that you reload the table, which, for large tables, might be a long process.

You should consider the task to hide and delete technical and useless columns as both a cosmetic and operational change in the PowerPivot functionality. It has some very good impact on the usability of PowerPivot, because fewer columns are available for selection. Moreover, the size of the Excel file is reduced by removing useless data from the tables, speeding up all the operations.

After the cleaning of these columns, the field selector, shown in Figure 2-10, looks much better and user friendly.


PowerPivot Field List

Choose fields to add to report:

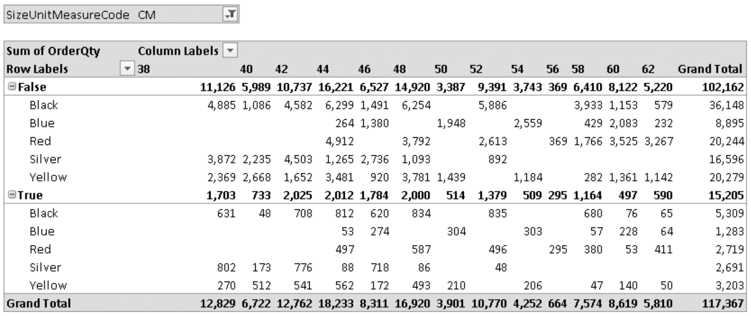
Search

- [ ] DaysToManufacture
- [ ] ProductLine
- [ ] Class
- [ ] Style
- SalesOrderDetail
  - [ ] CarrierTrackingNumber
  - [✓] OrderQty
  - [ ] UnitPrice
  - [ ] UnitPriceDiscount
  - [ ] LineTotal
- SalesOrderHeader
  - [ ] RevisionNumber
  - [ ] OrderDate
  - [ ] DueDate
  - [ ] ShipDate
  - [ ] Status
  - [ ] OnlineOrderFlag

FIGURE 2-10 The field selector with fewer columns.

Adding Calculated Columns

Now that you have hidden or deleted unwanted columns, you can continue the work to make the report easier to use and read. You might notice that there are some fields that have a technical meaning and are not easy to understand at a first glance. The OnlineOrderFlag, for example, is one of them. OnlineOrderFlag is a TRUE/FALSE value and, even if it is pretty understandable by itself, it does not look very nice if used in reports. Take a look, for example, at the report in Figure 2-11, where we simply removed the OnlineOrderFlag from the filter of the report and added it to the rows.



SizeUnitMeasureCode CM

Sum of OrderQty	Column Labels													
Row Labels	38	40	42	44	46	48	50	52	54	56	58	60	62	Grand Total
False	**11,126**	**5,989**	**10,737**	**16,221**	**6,527**	**14,920**	**3,387**	**9,391**	**3,743**	**369**	**6,410**	**8,122**	**5,220**	**102,162**
Black	4,885	1,086	4,582	6,299	1,491	6,254		5,886			3,933	1,153	579	36,148
Blue				264	1,380		1,948		2,559		429	2,083	232	8,895
Red				4,912		3,792		2,613		369	1,766	3,525	3,267	20,244
Silver	3,872	2,235	4,503	1,265	2,736	1,093		892						16,596
Yellow	2,369	2,668	1,652	3,481	920	3,781	1,439		1,184		282	1,361	1,142	20,279
True	**1,703**	**733**	**2,025**	**2,012**	**1,784**	**2,000**	**514**	**1,379**	**509**	**295**	**1,164**	**497**	**590**	**15,205**
Black	631	48	708	812	620	834		835			680	76	65	5,309
Blue				53	274		304		303		57	228	64	1,283
Red				497		587		496		295	380	53	411	2,719
Silver	802	173	776	88	718	86		48						2,691
Yellow	270	512	541	562	172	493	210		206		47	140	50	3,203
Grand Total	**12,829**	**6,722**	**12,762**	**18,233**	**8,311**	**16,920**	**3,901**	**10,770**	**4,252**	**664**	**7,574**	**8,619**	**5,810**	**117,367**

FIGURE 2-11 The values *True* and *False* are difficult to decode without a description.

Ask yourself this question: What do the labels False and True mean in the report? You can assign a meaning to False only if you remember that its value comes from the OnlineOrderFlag but, in the report itself, there is no clear evidence of the fact that the value True means *The order has been placed online*. Clearly, having a different and more understandable description would greatly improve the report readability.

Note Please note that the OnlineOrderFlag is not a technical column. Its description is cryptic, but we definitely want to slice data using this column. So we are not going to hide the column at all. Instead, we will provide it a better description so that the field is more user friendly.

So we are going to describe some standard techniques to show ONLINE ORDER when the value of the column is True and INTERNAL ORDER when the value is False. In order to perform this task, you have two choices:

- You can add a new calculated column to the OrderDetails table, assigning to it a descriptive value for the OnlineOrderFlag. Then you can hide the TRUE/FALSE column and provide only the new column for filtering and slicing.
- You can add a new table to the data model, which has a TRUE/FALSE column as the key and another column that holds the description. Then you can create a relationship between OrderDetails and this new table to let PowerPivot slice the original data with this new table.

Both these operations deal with a much wider topic—*data modeling*. Because they are both interesting and instructive solutions, we describe both of them, first to get a feeling of what a data model is and also to see how a good data model might affect the user experience. We cover data models in full detail in Chapter 4, "Data Models." Nevertheless, this first look now is useful for understanding data models.

The first option is the easier one. To add a column to the SalesOrderHeader table, you need to provide a name and an expression for it so that PowerPivot knows how to select that column and how to compute its values. To add a new column, you need to select the Add Column button on the Design tab of the PowerPivot ribbon, as you can see in Figure 2-12. This operation moves the cursor to the end of the current table and places the cursor inside the formula editor.

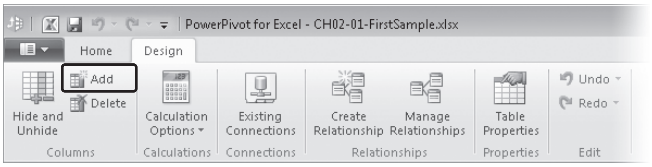
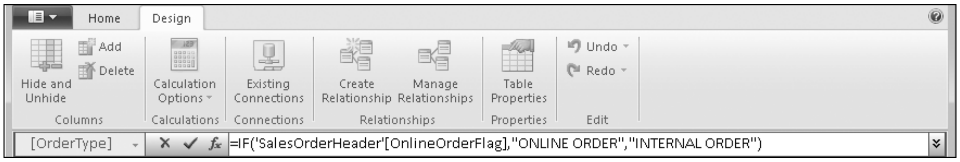


FIGURE 2-12 The Add button creates a new calculated column.

You can now write the formula for the new column in the formula bar of PowerPivot. The formula bar looks very similar to the formula bar of Excel. Nevertheless, formulas for PowerPivot are very different from formulas in Excel. PowerPivot does not use the Excel formula language. Instead, it uses a new language called DAX, which we introduce in Chapter 3, "Introduction to DAX." But for this simple example, we can ignore the complexities of DAX and enter a simple formula, which is understandable by itself (moreover, it looks similar to Excel) and is shown in Figure 2-13.

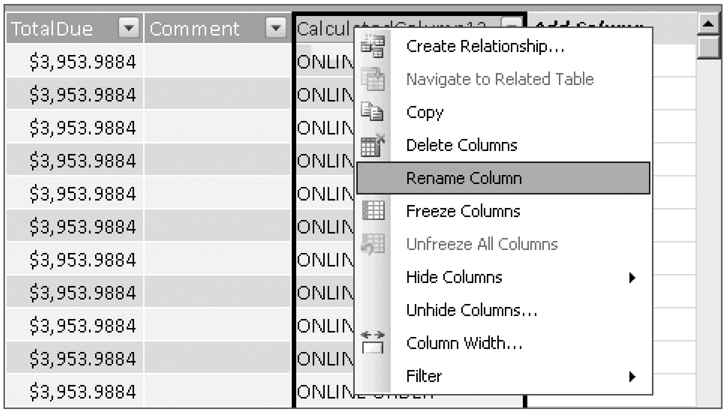


```
=IF('SalesOrderHeader'[OnlineOrderFlag],"ONLINE ORDER","INTERNAL ORDER")
```

FIGURE 2-13 The formula for the new column, shown in the formula bar.

This code uses the DAX function IF, which looks and works like the IF function in Excel. If the value of the first parameter evaluates to True, it returns its second parameter; otherwise, it returns the third one. In other words, if the OnlineOrderFlag is true, the formula returns ONLINE ORDER; otherwise, its value is INTERNAL ORDER.

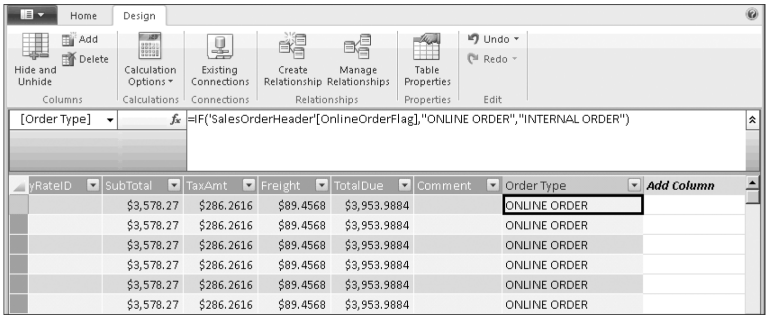
The newly added column has been named CalculatedColumn1 by PowerPivot, which is not really user friendly. To rename the column, it is enough to select the column, right-click the column name, and choose Rename from the menu, as you can see in Figure 2-14. You can choose, for example, to name it OrderType.



TotalDue	Comment	Calcul
$3,953.9884		ONLIN
$3,953.9884		ONLIN
$3,953.9884		ONLIN
$3,953.9884		ONLIN
$3,953.9884		ONLIN
$3,953.9884		ONLIN
$3,953.9884		ONLIN
$3,953.9884		ONLIN
$3,953.9884		ONLIN
$3,953.9884		ONLIN
$3,953.9884		ONLIN

FIGURE 2-14 The Rename Column option of the column menu, available with a right-click.

In Figure 2-15, you can see the final result, showing both the new column and the formula bar for the calculated column.



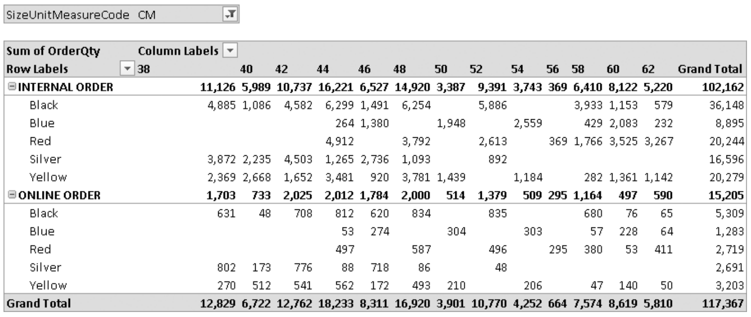
[Order Type]

```
=IF('SalesOrderHeader'[OnlineOrderFlag],"ONLINE ORDER","INTERNAL ORDER")
```

yRateID	SubTotal	TaxAmt	Freight	TotalDue	Comment	Order Type
	$3,578.27	$286.2616	$89.4568	$3,953.9884		ONLINE ORDER
	$3,578.27	$286.2616	$89.4568	$3,953.9884		ONLINE ORDER
	$3,578.27	$286.2616	$89.4568	$3,953.9884		ONLINE ORDER
	$3,578.27	$286.2616	$89.4568	$3,953.9884		ONLINE ORDER
	$3,578.27	$286.2616	$89.4568	$3,953.9884		ONLINE ORDER
	$3,578.27	$286.2616	$89.4568	$3,953.9884		ONLINE ORDER

FIGURE 2-15 The formula bar with the calculated column.

Now that you have a good description of the order type, you can safely hide the column OnlineOrderFlag, which now has become a technical column. (It contains a value needed to compute the description, but you want to browse and slice data using the description defined in the new Order Type column and not the original value.) After you do that, you can use the new OrderType field in a PivotTable, and the result looks like Figure 2-16.


SizeUnitMeasureCode CM

Sum of OrderQty	Column Labels													
Row Labels	38	40	42	44	46	48	50	52	54	56	58	60	62	Grand Total
INTERNAL ORDER	**11,126**	**5,989**	**10,737**	**16,221**	**6,527**	**14,920**	**3,387**	**9,391**	**3,743**	**369**	**6,410**	**8,122**	**5,220**	**102,162**
Black	4,885	1,086	4,582	6,299	1,491	6,254		5,886			3,933	1,153	579	36,148
Blue				264	1,380		1,948		2,559		429	2,083	232	8,895
Red				4,912		3,792		2,613		369	1,766	3,525	3,267	20,244
Silver	3,872	2,235	4,503	1,265	2,736	1,093		892						16,596
Yellow	2,369	2,668	1,652	3,481	920	3,781	1,439		1,184		282	1,361	1,142	20,279
ONLINE ORDER	**1,703**	**733**	**2,025**	**2,012**	**1,784**	**2,000**	**514**	**1,379**	**509**	**295**	**1,164**	**497**	**590**	**15,205**
Black	631	48	708	812	620	834		835			680	76	65	5,309
Blue				53	274		304		303		57	228	64	1,283
Red				497		587		496		295	380	53	411	2,719
Silver	802	173	776	88	718	86		48						2,691
Yellow	270	512	541	562	172	493	210		206		47	140	50	3,203
Grand Total	**12,829**	**6,722**	**12,762**	**18,233**	**8,311**	**16,920**	**3,901**	**10,770**	**4,252**	**664**	**7,574**	**8,619**	**5,810**	**117,367**

FIGURE 2-16 Sample report with OnlineOrderFlag decoded.

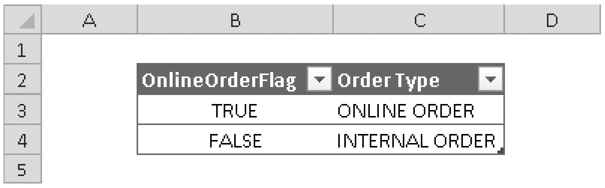
You can see that no value has changed but the report is now easier to understand because you have a clear knowledge of the meaning of the rows. In other words, the values are now self-explanatory, so the report is easier to use.

Important Whenever you create a PowerPivot workbook, you need to remember that textual descriptions of columns are always much easier to understand when compared to the underlying code. It is a bad idea to use code inside PivotTables because it makes the final PivotTable harder to use.

After you delete useless and hidden technical columns, you normally need to decode some columns and create new ones with better descriptions. Then you need to hide the original columns to allow users to choose only among self-describing columns.

We do not want, at this point, to investigate further how to define more complex columns because this first technique is straightforward. We prefer to spend some time showing a different solution to the same issue, which is to create a new related table. This second technique is interesting to study because, in developing it, you are going to change the data model, something you should learn as soon as possible.

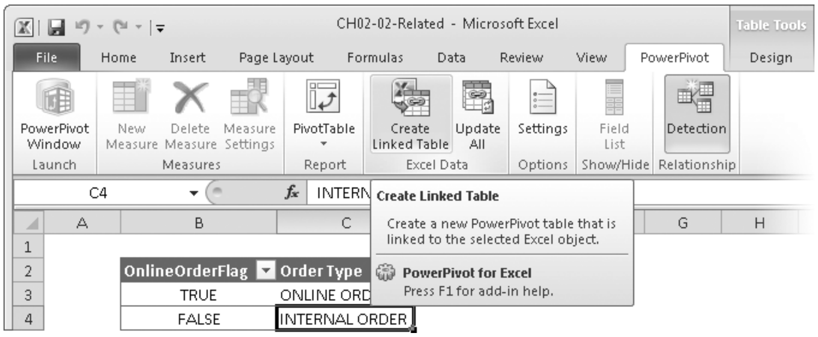
There is no table in the database that provides a description for the OnlineOrderFlag, so you need to create an Excel worksheet that contains the table and then make PowerPivot aware of this new information. To create the table, simply type in an Excel worksheet the information (see Figure 2-17) and then, after having selected the six cells, choose Format As Table on the Home tab of the Excel ribbon. You can find the example in the workbook CH02-02-Related.xlsx in the companion content.



	A	B	C	D
1				
2		OnlineOrderFlag	Order Type	
3		TRUE	ONLINE ORDER	
4		FALSE	INTERNAL ORDER	
5				

FIGURE 2-17 Decoding table for the OnlineOrderFlag.

Now that you have an Excel table, all you need to do is to let PowerPivot know of its existence. From the PowerPivot tab on the Excel ribbon, choose the Create Linked Table button with the cursor inside the table (see Figure 2-18). Please note that, if the cursor is not in the table, you need to manually provide the table boundaries, a tedious task that PowerPivot carries out for you if the cursor is inside the table.



Create Linked Table

Create a new PowerPivot table that is linked to the selected Excel object.

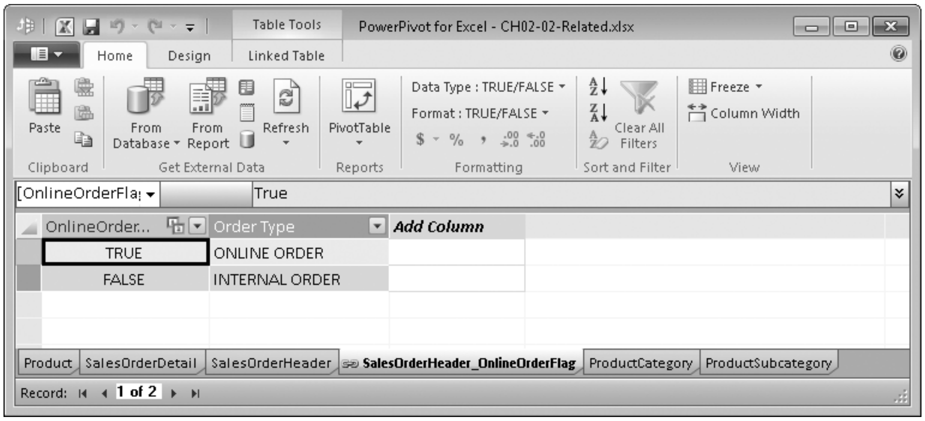
PowerPivot for Excel

Press F1 for add-in help.

OnlineOrderFlag	Order Type
TRUE	ONLINE ORD
FALSE	INTERNAL ORDER

FIGURE 2-18 The Create Linked Table command imports an Excel table inside PowerPivot.

This operation opens the PowerPivot window in which you can see your Excel table exactly as if it were a standard imported table. The only difference is in the small chain before its name, indicating that this is an Excel linked table and not an imported one. The table can be renamed a more appropriate name if you need to do that—for example, you can rename it SalesOrderHeader_OnlineOrderFlag. You can see this in Figure 2-19.

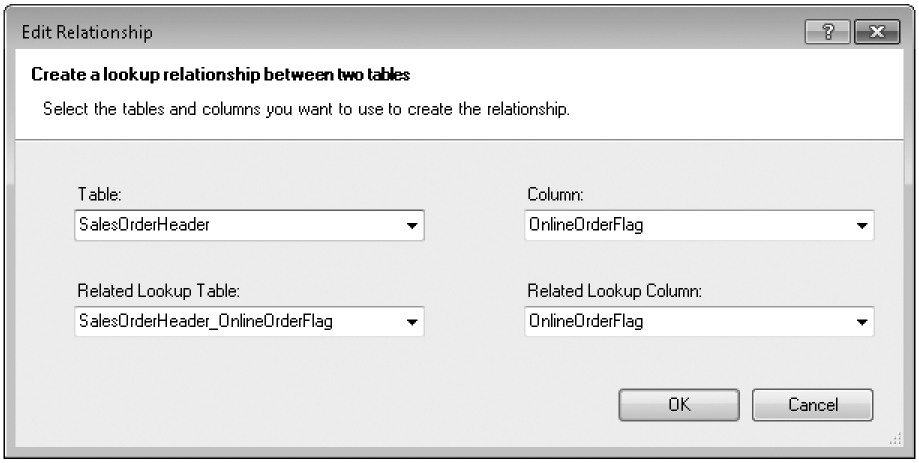


OnlineOrder...	Order Type	Add Column
TRUE	ONLINE ORDER	
FALSE	INTERNAL ORDER	

FIGURE 2-19 The decoding table imported in PowerPivot.

Maybe the power of what we are doing is not immediately evident, so it is worth spending some words on it. We are mixing, in the same PowerPivot model, tables coming from a SQL database with an ad-hoc table created in Excel to suit our needs. In other words, we are extending the existing model with our personal information. This simple fact helps us build complex and interesting data models.

The only missing point is a relationship between the SalesOrderHeader and this new table. To create the relationship, you need to go to the SalesOrderHeader table, choose the column OnlineOrderFlag, and click the Create Relationship button on the Design tab of the ribbon. This action opens a dialog box in which you describe the relationship. It should look like Figure 2-20.


Edit Relationship

Create a lookup relationship between two tables

Select the tables and columns you want to use to create the relationship.

Table: SalesOrderHeader

Column: OnlineOrderFlag

Related Lookup Table: SalesOrderHeader_OnlineOrderFlag

Related Lookup Column: OnlineOrderFlag

FIGURE 2-20 Definition of the relationship with the decoding table in PowerPivot.

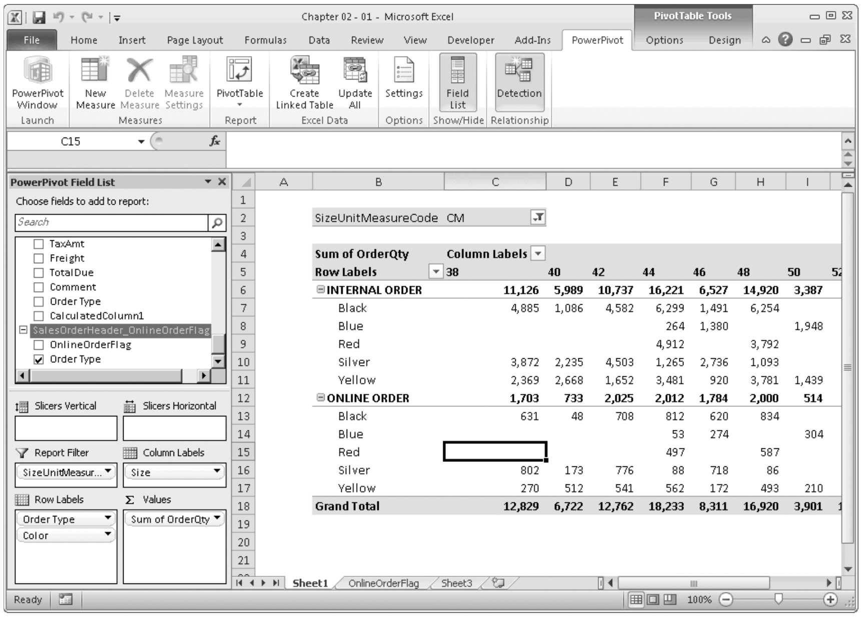
In Figure 2-20, you are stating that the OnlineOrderFlag in the SalesOrderHeader table is related to the column OnlineOrderFlag in the SalesOrderHeader_OnlineOrderFlag table. Because the related columns have the same type (a TRUE/FALSE value), the relationship can be created. Clicking Create is enough to make PowerPivot analyze data and create the relationship.

Lookup Tables

This kind of table, which contains keys and values describing them, is normally called a *Lookup Table* because it allows you to give a name to a code by looking up the code in the table.

Lookup tables are very similar to the Excel VLOOKUP functions. If, in a standard Excel worksheet, we want to provide a description to a particular code, we might use a VLOOKUP function in a cell that refers to a decoding area. Lookup relationships work much the same way even if, with PowerPivot, we use relationships to create much more complex models.

Now that you have completed the creation of a linked table and defined the relationship with this new table, it is time to return to the PivotTable and click the Refresh button to see what has changed (see Figure 2-21).



SizeUnitMeasureCode CM

Sum of OrderQty	Column Labels						
Row Labels	38	40	42	44	46	48	50
INTERNAL ORDER	**11,126**	**5,989**	**10,737**	**16,221**	**6,527**	**14,920**	**3,387**
Black	4,885	1,086	4,582	6,299	1,491	6,254	
Blue				264	1,380		1,948
Red				4,912		3,792	
Silver	3,872	2,235	4,503	1,265	2,736	1,093	
Yellow	2,369	2,668	1,652	3,481	920	3,781	1,439
ONLINE ORDER	**1,703**	**733**	**2,025**	**2,012**	**1,784**	**2,000**	**514**
Black	631	48	708	812	620	834	
Blue				53	274		304
Red				497		587	
Silver	802	173	776	88	718	86	
Yellow	270	512	541	562	172	493	210
Grand Total	**12,829**	**6,722**	**12,762**	**18,233**	**8,311**	**16,920**	**3,901**

FIGURE 2-21 The new decoding table in the PowerPivot selector.

You now see a new table inside the Field List, named SalesOrderHeader_OnlineOrderFlag, with two columns (one of which is a technical one that you should hide later). The Order Type column in this new table does the same task that was accomplished by the Order Type in the SalesOrderHeader table. The difference now is that it is much easier to change the descriptions because they are not hard-coded in a DAX formula but contained in a table hosted inside the Excel workbook. This means that changing the descriptions is now a simple task for anyone, and no one has to understand what is going on with the code.

Let us stop for a few seconds and think about what you have done.

- You have been able to mix different sources of data into a single coherent view of information, shaping the data model to make it fit your needs.
- You have provided your users (and yourself) an easy way to give descriptions to technical values, making the process of browsing the PivotTable and producing reports more intuitive.
- You created your first piece of a data model—that is, a model of data that describes the entities you intend to browse.

The only drawback of this solution is that, if many fields require a lookup table, the field selector of the PivotTable might become a little messy because too many tables start to appear inside it. Luckily, there is a simple solution to this: you can use the RELATED function in DAX, something you are going to learn later in this chapter.

Adding Measures

Even if you can perform many interesting calculations working at the row level of the tables, some calculations cannot be defined at this level because they depend on the query context. (In other words, they depend on the selection made by the user in the PivotTable.)

You will explore many of these calculations later in this book (see Chapters 3, 6, 7, and 8), and to do that, you need to learn the DAX language. But right now we want to show you a simple example of the differences between a calculated column and a measure in PowerPivot. We also briefly investigate why measures are sometimes needed.

You are going to implement a column that calculates the distinct count of products sold. A *distinct count* computes the number of distinct values of a specific column and is very useful, for example, for customers or products that happen to appear several times inside a table such as SalesOrderDetails. This formula cannot be computed at the row level because, for each sale, its value is 1 (one product sold) while, for many sales, its value is not the sum of all the values at the row level. Instead, it needs to be computed based on user selection. Such types of calculation cannot be defined at the row level, so they are called *measures* and need to be defined at the PivotTable level.

To create a new measure, we need to right-click the PowerPivot Field List and choose Add New Measure as in Figure 2-22. You can find this example in the companion file CH02-03-Measures.xlsx.

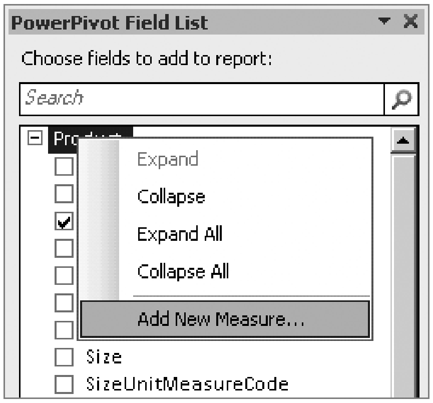


FIGURE 2-22 The context menu with which you add a new measure to the PowerPivot model underlying the PivotTable.

At this point, a new dialog box appears (see Figure 2-23) in which you need to provide the new measure properties. Type a name for the new measure—say, **DistinctProducts,** and then you need to write the DAX formula that calculates the value.

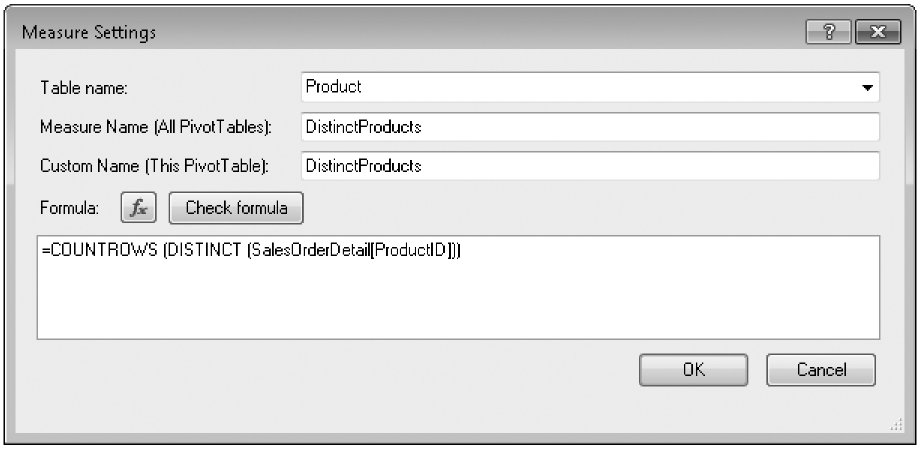


Table name: Product
Measure Name (All PivotTables): DistinctProducts
Custom Name (This PivotTable): DistinctProducts
Formula:

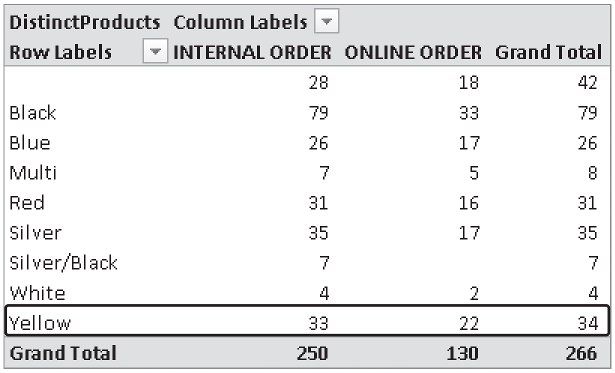
```
=COUNTROWS (DISTINCT (SalesOrderDetail[ProductID]))
```

FIGURE 2-23 The dialog box in which you add a new measure to the PowerPivot model.

Although easy to read, the DAX formula actually hides much of the power of DAX. We are not interested in understanding now the details of how it works. Let us just take a look at it:

```
COUNTROWS (DISTINCT (SalesOrderDetail[ProductID]))
```

You can read it as “count the number of rows that are in a table containing only the distinct values of the column ProductID of the table SalesOrderDetail”. To compute the value of this measure, PowerPivot makes the calculation in the context defined by the PivotTable query and provides the correct distinct count of products for each cell of the PivotTable. For example, you can produce an interesting report like the one in Figure 2-24, which computes the number of distinct products sold, slicing data by color.



DistinctProducts	Column Labels		
Row Labels	INTERNAL ORDER	ONLINE ORDER	Grand Total
	28	18	42
Black	79	33	79
Blue	26	17	26
Multi	7	5	8
Red	31	16	31
Silver	35	17	35
Silver/Black	7		7
White	4	2	4
Yellow	33	22	34
Grand Total	**250**	**130**	**266**

FIGURE 2-24 Query of the distinct products.

As you can see, for each cell there is a calculation of the number of unique products sold. Moreover, it is worth noting that the aggregation of distinct count is not the sum. If you look, for example, at the row for the color Yellow, we have sold 33 distinct products with internal orders, 22 online, but the grand total of distinct products is 34. In other words, out of 34 yellow products sold, 22 were sold online, 33 were sold directly, and only one of them has been sold online and not directly, thus giving you the grand total of 34. This might seems confusing at first glance, but it is indeed the correct behavior to expect when you are using distinct counts.

From the PivotTable point of view, measures and columns look very similar even if, for what concerns the internal engine of PowerPivot, they are completely different items. Starting in Chapter 3, you begin to learn the DAX language and the exact difference between calculated columns and measures.

Adding More Tables

All the reports shown up to now relied on three tables only, and they already have shown some interesting data. Nevertheless, the AdventureWorks database contains a lot of other tables that you can add to the PowerPivot data model to improve reporting. You might have noticed, for example, that the Products table contains a ProductSubcategoryID. This column is a key in the ProductSubcategory table, which we have not loaded yet. It happens, in turn, that the ProductSubcategory table contains a key, called ProductCategoryID, which relates to the ProductCategory table. This chain of relationships lets us retrieve the product category, by means of walking two steps, from a product to its subcategory and then from the subcategory to its category.

Graphically, the relationship can be seen in Figure 2-25.

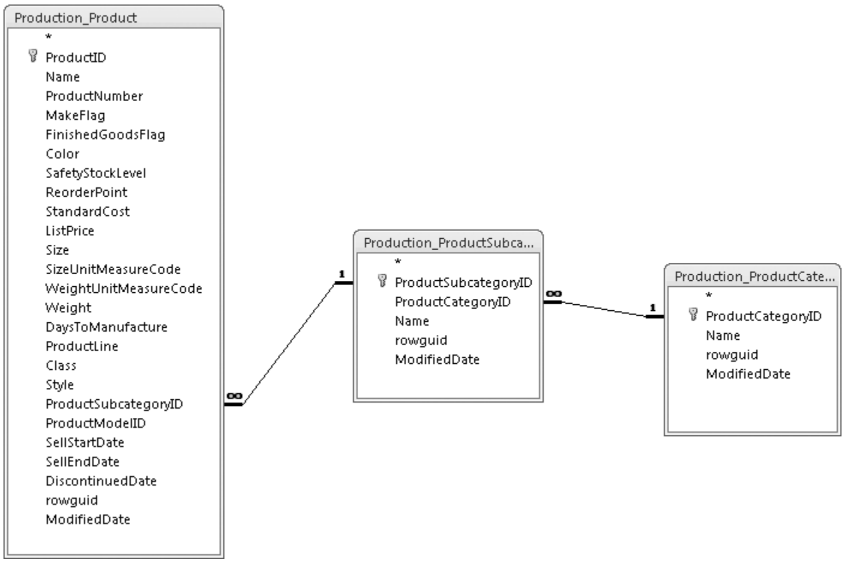
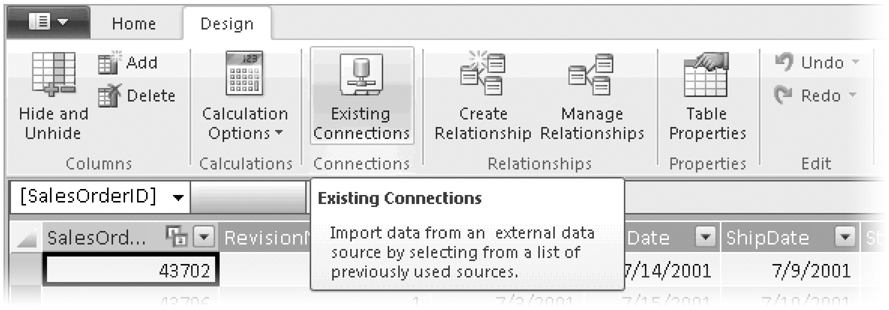


FIGURE 2-25 The chained (or cascading) relationship between three tables.

These kinds of relationships, which appear very often in the database world, are called *chained relationships* because they form a chain that you can follow from the beginning to the end to relate many tables.

Note Please note a curious phenomenon that often appears in the world of databases. Even if we plan to slice data by category first, then by subcategory, and finally by products, following a very natural path, in reality the chain of the relationships is reversed, starting from the more detailed table and going into the less detailed one. This is absolutely normal—it concerns how data is modeled in relational databases. Throughout the book, we discover many other relationships that need to be read in this reversed way.

To make PowerPivot allow you to slice data with the columns of these new tables, you need to import them into your data model. To do that, it is enough to repeat the loading process you did before to import the first three tables. This time, instead of using the From Database button, you should use the Existing Connections button, which is located on the Design tab of the PowerPivot ribbon, as you can see in Figure 2-26. You can do this because the connection to the database has been already saved inside the Excel workbook and you can now use it to import all the useful tables without needing to create a new connection. You can find this example, with the tables already loaded, in the workbook CH02-04-NewTables.xlsx.



Existing Connections

Import data from an external data source by selecting from a list of previously used sources.

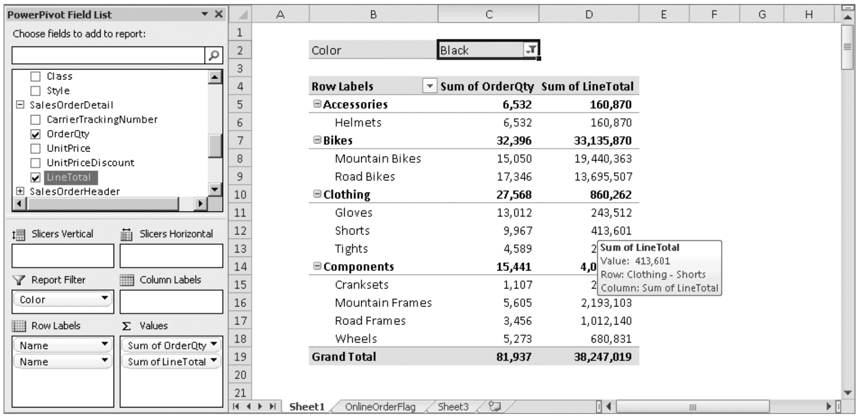
FIGURE 2-26 The Existing Connection button opens a connection to a previously used database.

You already know that during the loading process PowerPivot detects the relationship between Subcategory and Category. Moreover, you also know that you need to hide technical columns (ProductSubcategoryID, ProductCategoryID) and to remove the useless ones (rowguid and UpdatedDate in both tables) to create a clean data model.

Now, if you try to add Category or Subcategory to the PivotTable, PowerPivot detects the need for new relationships and, when asked to, it automatically detects the relationship between the SubcategoryID in the Products table and the column with the same name in the ProductSubcategory one.

Note You might wonder why PowerPivot is so good at finding relationships during the loading of tables and did not detect the relationship between Product and ProductSubcategory, even if this relationship is already stored in the database metadata. The reason is that, during loading of data, PowerPivot searches for relationships among the tables it is currently loading, ignoring tables that are already present in the PowerPivot data model. These other relationships (between existing and new tables) need to be detected later, through the relationship-detection algorithm.

Now that you have empowered the PowerPivot data model with these two tables, you easily can produce complex reports like the one shown in Figure 2-27, in which we mix columns from products, categories, subcategories and orders, letting PowerPivot resolve the complex relationships that make the browsing process possible.



Color	Black	
Row Labels	**Sum of OrderQty**	**Sum of LineTotal**
Accessories	**6,532**	**160,870**
Helmets	6,532	160,870
Bikes	**32,396**	**33,135,870**
Mountain Bikes	15,050	19,440,363
Road Bikes	17,346	13,695,507
Clothing	**27,568**	**860,262**
Gloves	13,012	243,512
Shorts	9,967	413,601
Tights	4,589	2
Components	**15,441**	**4,0**
Cranksets	1,107	2
Mountain Frames	5,605	2,193,103
Road Frames	3,456	1,012,140
Wheels	5,273	680,831
Grand Total	**81,937**	**38,247,019**

Sum of LineTotal
Value: 413,601
Row: Clothing - Shorts
Column: Sum of LineTotal

FIGURE 2-27 Sample report with categories and subcategories.

The report, as it looks now, is quite nice. Nevertheless, because you are surely striving for perfection, you notice a couple of small issues:

- Both tables (ProductCategory and ProductSubcategory) have the same description for the column name, as you can see in the Row Labels list in Figure 2-27. This is not very user friendly because it is hard to understand whether you correctly put subcategory under category or vice versa (apart, clearly, from the rule of common sense as soon as you see wrong data).
- ProductCategory and ProductSubcategory are separated from the Products table, even if they are strictly related to products. In a small pivot table like this one, this is not a big issue. However, as the data model gets larger, you should try to reduce the number of tables shown to the user as much as possible, to make it easier to find the columns. A rule of thumb in the Business Intelligence world dictates that you should never browse more than 15 different tables. If you let tables spread at a speed of one table per lookup, you reach that limit very quickly.

Although the solution of the first point is straightforward (it is enough to change the name of the columns as displayed by PowerPivot in the PowerPivot window), the second one is much more interesting because it lets us introduce a very simple yet powerful DAX formula: RELATED.

You are now going to remove the tables ProductCategory and ProductSubcategory from the field list in the PivotTable editor, replacing them with two new columns in the Product table, named Category and Subcategory. Moreover, in doing this, you will solve both the points stated above.

The problem you need to face is that the original Product table does not contain the textual description of category or subcategory, it only contains the ProductSubcategoryID, which is a technical column used to create the relationship with the ProductSubcategory table. You definitely need a way to create a calculated column in a table that contains the value of a column in another table, following a relationship. This is exactly what the RELATED function has been built for.

The RELATED function returns the value of a column from another table if it has a valid relationship with the current one. You can define two new columns inside the Product table using these formulas, as shown in Table 2-1.

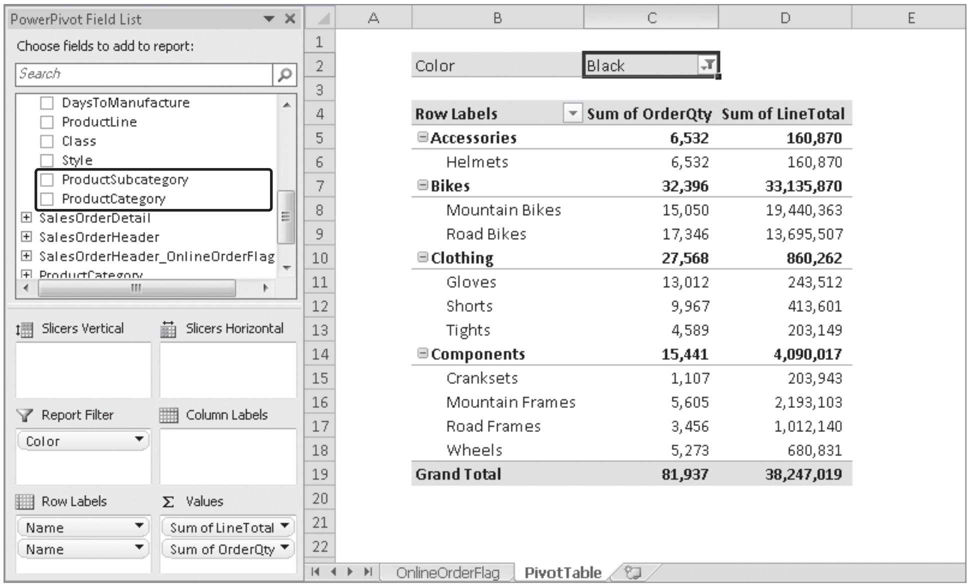
TABLE 2-1 Using the RELATED function.

Column	Formula
SubCategory	=RELATED (ProductSubcategory[Name])
Category	=RELATED (ProductCategory[Name])

The SubCategory calculated column contains the value of the Name column in the ProductSubcategory table, while the Category calculated column contains the name of the category, taken from the ProductCategory table.

Note We do not need to worry about the fact that the relationship between Products and ProductCategory is a chained one, which makes it necessary for PowerPivot to follow two relationship steps to gather the correct category value. PowerPivot already knows about the existence of chained relationships and handles this complexity by itself.

This simple definition leads to a much better user experience because now you see two new columns inside the Products table that hold the value of the category and subcategory. So you can safely hide all the columns in the lookup tables (which, in turn, makes both tables disappear from the field list). The report looks like the one in Figure 2-28.



Report Filter: Color
Row Labels: Name, Name
Values: Sum of LineTotal, Sum of OrderQty

Color	Black	
Row Labels	**Sum of OrderQty**	**Sum of LineTotal**
Accessories	**6,532**	**160,870**
Helmets	6,532	160,870
Bikes	**32,396**	**33,135,870**
Mountain Bikes	15,050	19,440,363
Road Bikes	17,346	13,695,507
Clothing	**27,568**	**860,262**
Gloves	13,012	243,512
Shorts	9,967	413,601
Tights	4,589	203,149
Components	**15,441**	**4,090,017**
Cranksets	1,107	203,943
Mountain Frames	5,605	2,193,103
Road Frames	3,456	1,012,140
Wheels	5,273	680,831
Grand Total	**81,937**	**38,247,019**

FIGURE 2-28 Sample report with categories and subcategories tied to the product.

Although this might seem a simple enhancement, it is a very important one because it lets us introduce the concept of Data Modeling, to which we are going to dedicate the whole of Chapter 4. The user queries the Data Model, and the simpler it is, the better will be his experience. One of the most complex abilities of a data modeler is to create models that, even if complex in their implementation, look very easy to the end user.

Note Even if the original data model of AdventureWorks has two distinct tables for category and subcategory (which is the right choice for a standard database system), the data model is much easier to query if you hide these two tables and transform their content into columns inside the Product table, which is exactly what you have done. By modifying the data model, you have reduced the number of tables shown to the user and given a meaningful name to the columns.

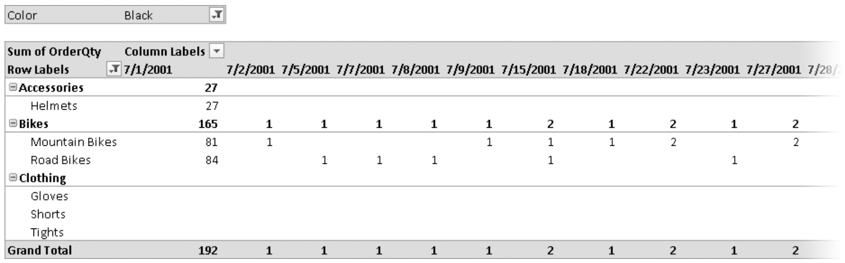
The two tables still exist in the data model, but they are hidden from the user, who can have access to their values through the new computed columns. This is the first sample situation in which you use an internal data model while showing a different one to the user. You will get acquainted with this technique because we use it throughout the book.

An interesting exercise, which we leave to you to try, is to use the RELATED function to remove the technical table we previously used to give a description of the online order flag. The technique is exactly the same one you used in this section and the exercise gives you greater confidence using the RELATED function. Moreover, you should use the same technique whenever the purpose of a table is to provide lookup values and that table is not a part of the data model you want to show to the user.

Working with Dates

Up to now, you have used columns that contain a relatively small set of distinct values, such as the color or the category of products, to slice data in the PivotTable. When, on the other hand, a column contains a lot of distinct values, the resulting PivotTable gets harder to use. We are now going to describe this problem in greater detail and provide a solution for it.

The SalesOrderHeader table contains a column, OrderDate, which records the date of the order. The details in this column are important but, for the purpose of reporting, the column contains too much information. If you simply put the OrderDate data in columns, you end up with a report that contains all the information you need but is very difficult to read (see Figure 2-29) because of the high fragmentation of values. In technical terms, we say that the order date column is not a good aggregator because it does not let us focus on interesting information. A good aggregator, on the other hand, groups together a huge number of distinct elements of information, leading to interesting results. You can find this example in the workbook CH02-05-WorkingWithDates.xlsx.



Color	Black

Sum of OrderQty	Column Labels											
Row Labels	7/1/2001	7/2/2001	7/5/2001	7/7/2001	7/8/2001	7/9/2001	7/15/2001	7/18/2001	7/22/2001	7/23/2001	7/27/2001	7/28/
Accessories	**27**											
Helmets	27											
Bikes	**165**	**1**	**1**	**1**	**1**	**1**	**2**	**1**	**2**	**1**	**2**	
Mountain Bikes	81	1				1	1	1	2		2	
Road Bikes	84		1	1	1		1			1		
Clothing												
Gloves												
Shorts												
Tights												
Grand Total	**192**	**1**	**1**	**1**	**1**	**1**	**2**	**1**	**2**	**1**	**2**	

FIGURE 2-29 Date columns are not good aggregators; the report is sparse.

As you can see, browsing information at the date level produces a sparse report. A much better aggregator would be the year or the month level. Both of those aggregators greatly reduce the fragmentation of the report and result in a better understanding of the data.

PowerPivot can aggregate data, but to do that, it needs columns. So you need to add new columns to the SalesOrderHeader table that contains the year and to the table that contains the month of the order. Aggregating for these columns produces the result you want.

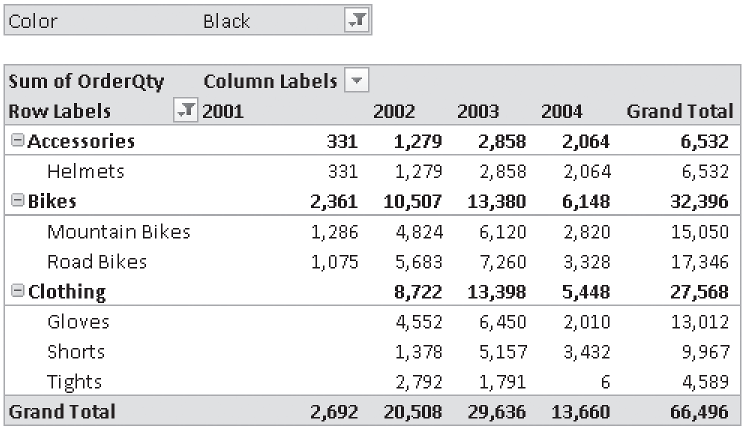
Note As you can see, we are shifting our concern from the problem of the sparse report to that of adding new calculated columns, and because we already know how to add new calculated columns, we are now finding our way to the solution of our problem.

You can add two new calculated columns to the SalesOrderHeader table in the PowerPivot window, following the already described procedure, and use Table 2-2 to get the formulas.

TABLE 2-2 Using the date/time functions.

Column	Formula
Order Year	=YEAR (SalesOrderHeader[OrderDate])
Order Month	=MONTH (SalesOrderHeader[OrderDate])

You are using two DAX functions: YEAR and MONTH, which, as their name suggests, return the year and the month of the date they receive as the parameter. Now you have two new columns that let you slice the data by year and month. You can use these columns to produce interesting reports, like the one in Figure 2-30, which aggregates at the year level.

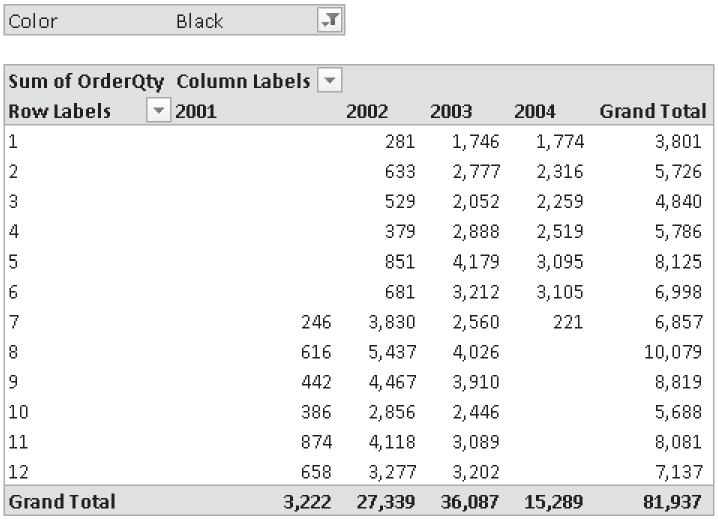


Color	Black

Sum of OrderQty	Column Labels				
Row Labels	2001	2002	2003	2004	Grand Total
⊟Accessories	**331**	**1,279**	**2,858**	**2,064**	**6,532**
Helmets	331	1,279	2,858	2,064	6,532
⊟Bikes	**2,361**	**10,507**	**13,380**	**6,148**	**32,396**
Mountain Bikes	1,286	4,824	6,120	2,820	15,050
Road Bikes	1,075	5,683	7,260	3,328	17,346
⊟Clothing		**8,722**	**13,398**	**5,448**	**27,568**
Gloves		4,552	6,450	2,010	13,012
Shorts		1,378	5,157	3,432	9,967
Tights		2,792	1,791	6	4,589
Grand Total	**2,692**	**20,508**	**29,636**	**13,660**	**66,496**

FIGURE 2-30 Aggregating by year produces more interesting reports.

Or combining month and years, you can produce the report shown in Figure 2-31.



Color	Black

Sum of OrderQty	Column Labels				
Row Labels	2001	2002	2003	2004	Grand Total
1		281	1,746	1,774	3,801
2		633	2,777	2,316	5,726
3		529	2,052	2,259	4,840
4		379	2,888	2,519	5,786
5		851	4,179	3,095	8,125
6		681	3,212	3,105	6,998
7	246	3,830	2,560	221	6,857
8	616	5,437	4,026		10,079
9	442	4,467	3,910		8,819
10	386	2,856	2,446		5,688
11	874	4,118	3,089		8,081
12	658	3,277	3,202		7,137
Grand Total	**3,222**	**27,339**	**36,087**	**15,289**	**81,937**

FIGURE 2-31 Combining years and months on the same report.

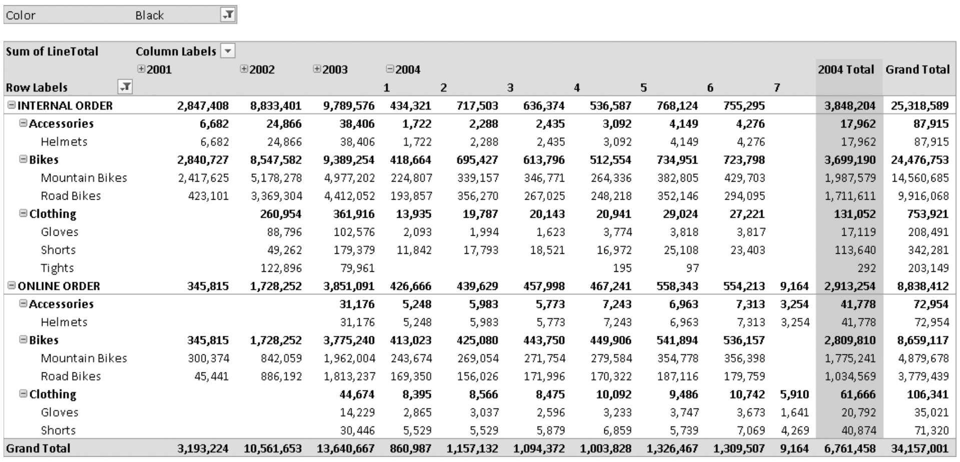
You might have noticed that the report shows the numbers of the months and not their names, which surely needs to be fixed. In Chapter 7, "Date Calculations in DAX," where we cover date handling in much more detail, you learn how to show the month names. Nevertheless, faster help comes in Chapter 3, where you find a "Date and Time Functions" section with a list of the functions available to you for manipulating dates. You also find in Chapter 3 a simple formula for getting both the number and the name of a month from a date.

The technique of adding more columns to the table to produce aggregates when the data inside the table is too detailed is frequently used with dates. Moreover, dates are so important a topic in the BI analysis that we will spend all of Chapter 7 on it. That said, there are many columns that are not good aggregators. For example, in Chapter 10, "PowerPivot Data Model Patterns," you see a complete banding system that performs aggregations by price bands and, as you will learn, the technique is similar to the one we just used: whenever a column is not a good aggregator, you need to add new columns that group more data so that aggregated values get interesting.

Refreshing Data

Now that we have scratched the surface of some of the many PowerPivot features, it is time to understand what happens to your reports when the underlying data changes, something that happens rapidly because of the normal life cycle of data.

A report is nothing but an Excel file with a PivotTable that queries data and provides interesting results. Our current model is already a good source that produces nice reports like the one in Figure 2-32. As you might notice, we used the date year and month in columns, so we can expect this report to change over time.



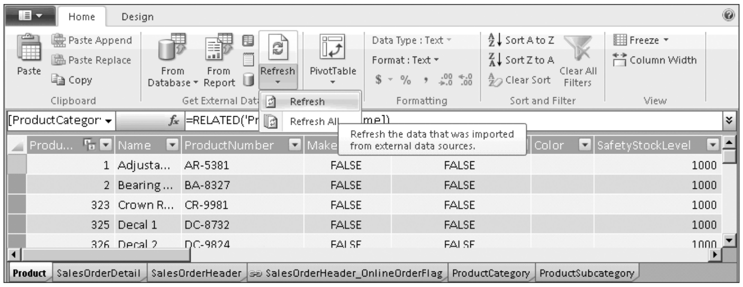
Color	Black

Sum of LineTotal	Column Labels											
	2001	2002	2003	2004							2004 Total	Grand Total
Row Labels				1	2	3	4	5	6	7		
INTERNAL ORDER	**2,847,408**	**8,833,401**	**9,789,576**	**434,321**	**717,503**	**636,374**	**536,587**	**768,124**	**755,295**		**3,848,204**	**25,318,589**
Accessories	**6,682**	**24,866**	**38,406**	**1,722**	**2,288**	**2,435**	**3,092**	**4,149**	**4,276**		**17,962**	**87,915**
Helmets	6,682	24,866	38,406	1,722	2,288	2,435	3,092	4,149	4,276		17,962	87,915
Bikes	**2,840,727**	**8,547,582**	**9,389,254**	**418,664**	**695,427**	**613,796**	**512,554**	**734,951**	**723,798**		**3,699,190**	**24,476,753**
Mountain Bikes	2,417,625	5,178,278	4,977,202	224,807	339,157	346,771	264,336	382,805	429,703		1,987,579	14,560,685
Road Bikes	423,101	3,369,304	4,412,052	193,857	356,270	267,025	248,218	352,146	294,095		1,711,611	9,916,068
Clothing		**260,954**	**361,916**	**13,935**	**19,787**	**20,143**	**20,941**	**29,024**	**27,221**		**131,052**	**753,921**
Gloves		88,796	102,576	2,093	1,994	1,623	3,774	3,818	3,817		17,119	208,491
Shorts		49,262	179,379	11,842	17,793	18,521	16,972	25,108	23,403		113,640	342,281
Tights		122,896	79,961				195	97			292	203,149
ONLINE ORDER	**345,815**	**1,728,252**	**3,851,091**	**426,666**	**439,629**	**457,998**	**467,241**	**558,343**	**554,213**	**9,164**	**2,913,254**	**8,838,412**
Accessories			**31,176**	**5,248**	**5,983**	**5,773**	**7,243**	**6,963**	**7,313**	**3,254**	**41,778**	**72,954**
Helmets			31,176	5,248	5,983	5,773	7,243	6,963	7,313	3,254	41,778	72,954
Bikes	**345,815**	**1,728,252**	**3,775,240**	**413,023**	**425,080**	**443,750**	**449,906**	**541,894**	**536,157**		**2,809,810**	**8,659,117**
Mountain Bikes	300,374	842,059	1,962,004	243,674	269,054	271,754	279,584	354,778	356,398		1,775,241	4,879,678
Road Bikes	45,441	886,192	1,813,237	169,350	156,026	171,996	170,322	187,116	179,759		1,034,569	3,779,439
Clothing			**44,674**	**8,395**	**8,566**	**8,475**	**10,092**	**9,486**	**10,742**	**5,910**	**61,666**	**106,341**
Gloves			14,229	2,865	3,037	2,596	3,233	3,747	3,673	1,641	20,792	35,021
Shorts			30,446	5,529	5,529	5,879	6,859	5,739	7,069	4,269	40,874	71,320
Grand Total	**3,193,224**	**10,561,653**	**13,640,667**	**860,987**	**1,157,132**	**1,094,372**	**1,003,828**	**1,326,467**	**1,309,507**	**9,164**	**6,761,458**	**34,157,001**

FIGURE 2-32 A report that contains current dates probably needs to be refreshed periodically.

You probably want to produce reports like this one and then refresh their values periodically to get the newest data and cover the latest periods in time. Nevertheless, when you import data to PowerPivot, you do not create a live link between the source table and the Excel workbook. Instead, you copy data to PowerPivot, which stores the information in its columnar database and works separately from the original data source. For this reason, if you want to refresh data, you need to reload the information directly from the source.

To reload data, you need to click the Refresh button on the Home tab of the PowerPivot ribbon, shown in Figure 2-33.



[ProductCategor =RELATED('Pr me])

Refresh the data that was imported from external data sources.

Produ...	Name	ProductNumber	Make		Color	SafetyStockLevel
1	Adjusta...	AR-5381	FALSE	FALSE		1000
2	Bearing ...	BA-8327	FALSE	FALSE		1000
323	Crown R...	CR-9981	FALSE	FALSE		1000
325	Decal 1	DC-8732	FALSE	FALSE		1000
326	Decal 2	DC-9824	FALSE	FALSE		1000

FIGURE 2-33 To refresh data, we need to use the PowerPivot window.

Because refreshing a table really means reloading it from the database, it can sometimes take time. It is a pretty fast operation for small tables (less than one million rows, for instance), but it gets more time-consuming as the table becomes larger.

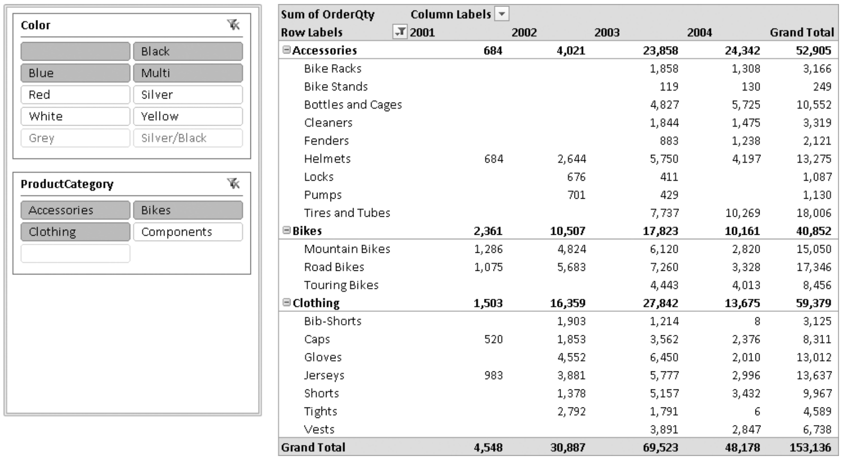
Note This PowerPivot operation works very differently from the behavior of an Excel PivotTable linked to a SQL Server Analysis Services database. When you connect Excel to Analysis Services, Excel stores only the results, not the original data. So whenever the underlying dataset in the Analysis Services database changes, you can simply refresh the PivotTable to make Excel query the database again and retrieve the new information. The basic difference is that a PowerPivot workbook stores data whereas a classic PivotTable is just a presentation layer over data stored somewhere else (in the example, in the Analysis Services database).

The only kind of tables that are automatically refreshed are Linked Tables—that is, tables that exist in the Excel workbook and are imported inside PowerPivot through the Create Linked Table button. You can switch automatic updates of linked tables on or off, using the options available on the Linked Table tab of the PowerPivot ribbon. The option Update Mode on the Linked Table tab is set to Automatic by default, and there are very rare cases where it might be useful to turn it off.

Using Slicers

So far, we have looked at PowerPivot features. Now we would like to end this chapter by discussing a standard Excel feature that is very useful when you are working with PowerPivot. Excel 2010 can add slicers to a PivotTable. Although slicers have been introduced mainly for PivotTables linked to PowerPivot data, they are a feature with range: you can define slicers for PivotTables linked to Analysis Services databases or simple PivotTables linked to data in the same Excel workbook. Because slicers are so useful for reports, it is surely worth mentioning them in a PowerPivot book.

Slicers are graphical items that let the user easily define filters on a PivotTable. Take a look at the report in Figure 2-34, which contains a PivotTable and a couple of slicers. You can find this example in the workbook CH02-06-Slicers.xlsx.



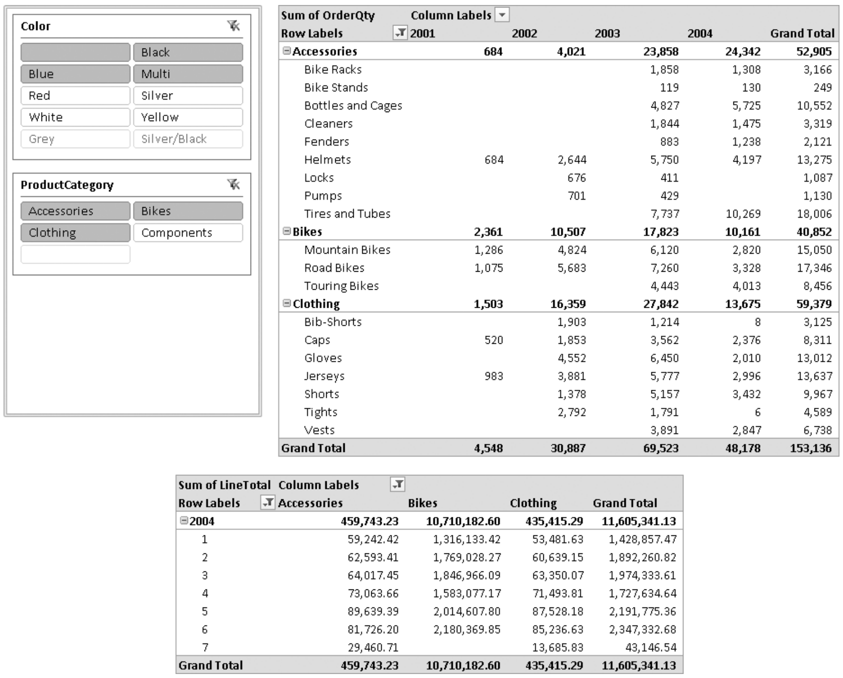
Sum of OrderQty	Column Labels				
Row Labels	2001	2002	2003	2004	Grand Total
Accessories	**684**	**4,021**	**23,858**	**24,342**	**52,905**
Bike Racks			1,858	1,308	3,166
Bike Stands			119	130	249
Bottles and Cages			4,827	5,725	10,552
Cleaners			1,844	1,475	3,319
Fenders			883	1,238	2,121
Helmets	684	2,644	5,750	4,197	13,275
Locks		676	411		1,087
Pumps		701	429		1,130
Tires and Tubes			7,737	10,269	18,006
Bikes	**2,361**	**10,507**	**17,823**	**10,161**	**40,852**
Mountain Bikes	1,286	4,824	6,120	2,820	15,050
Road Bikes	1,075	5,683	7,260	3,328	17,346
Touring Bikes			4,443	4,013	8,456
Clothing	**1,503**	**16,359**	**27,842**	**13,675**	**59,379**
Bib-Shorts		1,903	1,214	8	3,125
Caps	520	1,853	3,562	2,376	8,311
Gloves		4,552	6,450	2,010	13,012
Jerseys	983	3,881	5,777	2,996	13,637
Shorts		1,378	5,157	3,432	9,967
Tights		2,792	1,791	6	4,589
Vests			3,891	2,847	6,738
Grand Total	**4,548**	**30,887**	**69,523**	**48,178**	**153,136**

FIGURE 2-34 Slicers are graphical items that perform one-click filtering.

The two slicers on the left show all the possible values of the Color and Category columns. You can click a single cell and activate a filter for a specific value or press the Ctrl key and click to activate the filter for multiple values. Slicers are clearly useful when a column contains a small number of different values, such as color and category. Slicers for columns that have a lot of different possible values are difficult to use. Nevertheless, because the filtering normally happens on columns with a small number of distinct values, slicers are graphically appealing and very easy to use.

Slicers behave exactly like filters, but they are more elegant and easier to use. It is worth noting, moreover, that slicers can show columns that already appear in the PivotTable, as is the case in our example for the category column. We put the category on rows, in the first place, and then put the category on the slicers. The same column can appear in slicers and in the report. This is a feature that standard filters of a PivotTable do not support.

Moreover, slicers have another significant difference with filters: whereas filters apply to a single PivotTable (they are, after all, part of the query sent to the data source), slicers can be tied to more than one PivotTable, filtering them all with a single click. Let us look at the report in Figure 2-35.



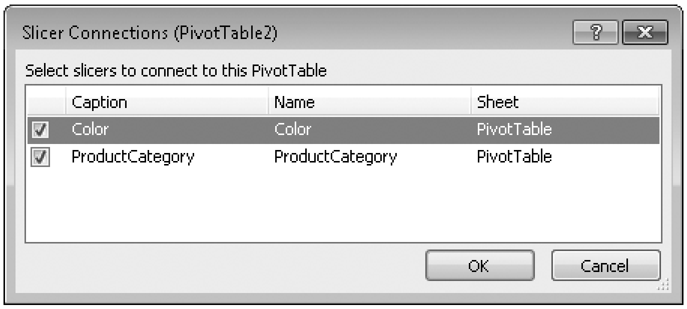
Sum of OrderQty	Column Labels				
Row Labels	2001	2002	2003	2004	Grand Total
⊟Accessories	**684**	**4,021**	**23,858**	**24,342**	**52,905**
Bike Racks			1,858	1,308	3,166
Bike Stands			119	130	249
Bottles and Cages			4,827	5,725	10,552
Cleaners			1,844	1,475	3,319
Fenders			883	1,238	2,121
Helmets	684	2,644	5,750	4,197	13,275
Locks		676	411		1,087
Pumps		701	429		1,130
Tires and Tubes			7,737	10,269	18,006
⊟Bikes	**2,361**	**10,507**	**17,823**	**10,161**	**40,852**
Mountain Bikes	1,286	4,824	6,120	2,820	15,050
Road Bikes	1,075	5,683	7,260	3,328	17,346
Touring Bikes			4,443	4,013	8,456
⊟Clothing	**1,503**	**16,359**	**27,842**	**13,675**	**59,379**
Bib-Shorts		1,903	1,214	8	3,125
Caps	520	1,853	3,562	2,376	8,311
Gloves		4,552	6,450	2,010	13,012
Jerseys	983	3,881	5,777	2,996	13,637
Shorts		1,378	5,157	3,432	9,967
Tights		2,792	1,791	6	4,589
Vests			3,891	2,847	6,738
Grand Total	**4,548**	**30,887**	**69,523**	**48,178**	**153,136**

Sum of LineTotal	Column Labels			
Row Labels	Accessories	Bikes	Clothing	Grand Total
⊟2004	**459,743.23**	**10,710,182.60**	**435,415.29**	**11,605,341.13**
1	59,242.42	1,316,133.42	53,481.63	1,428,857.47
2	62,593.41	1,769,028.27	60,639.15	1,892,260.82
3	64,017.45	1,846,966.09	63,350.07	1,974,333.61
4	73,063.66	1,583,077.17	71,493.81	1,727,634.64
5	89,639.39	2,014,607.80	87,528.18	2,191,775.36
6	81,726.20	2,180,369.85	85,236.63	2,347,332.68
7	29,460.71		13,685.83	43,146.54
Grand Total	**459,743.23**	**10,710,182.60**	**435,415.29**	**11,605,341.13**

FIGURE 2-35 Slicers can be tied to more than one PivotTable, creating interactive reports.

The upper PivotTable shows the number of items sold over time, whereas the lower one shows a detail of the year 2004 and displays the money value of sales. It would be nice to be able to link the slicer to both PivotTables so that we produce an interactive report in which you can select color and category and update both PivotTables quickly.

To link a slicer to more than one PivotTable, you can move the cursor inside the PivotTable you want to link to a slicer and then choose the Slicer Connections option from the Insert Slicer button on the Options tab of the Excel ribbon. The dialog box in Figure 2-36 appears.



Select slicers to connect to this PivotTable

	Caption	Name	Sheet
✓	Color	Color	PivotTable
✓	ProductCategory	ProductCategory	PivotTable

FIGURE 2-36 With this dialog box, we can tie one slicer to more than one PivotTable.

You can link both slicers (Category and Color) or make a selection of which slicer to link to the PivotTable and which ones not to use. If you link both slicers to both PivotTables, both tables are updated to reflect the filters when you make a selection in the slicers. This simple characteristic of slicers makes them a great option when you need to create interactive reports.

Caution Whenever you place more than one PivotTable on the same worksheet, you need to guard against overlapping them. PivotTables, by nature, change their size depending on the selections you make. (As you change the number of categories, the PivotTable containing the categories increases its height to accommodate the new categories selected.) If you want to place PivotTables side by side, for example, you need to make sure that they cannot grow to the point of overlapping. If this happens, Excel raises an error.

Summary

In this chapter, you learned some of the most useful features of PowerPivot:

- Format strings should be set in the PivotTable field settings so that resizing of columns is carried on by the PivotTable itself.
- Useless columns should be deleted from the PowerPivot data model, and technical ones should be hidden so that you have fewer columns to search to produce the report.
- You can add simple calculated columns or use more complex functions, such as RELATED, to enrich a table with information computed on related tables. RELATED is a very useful function because it allows us to reduce the number of tables and move columns to locations the user expects them.
- You created your first linked table, which enriches the original data model with other information directly stored inside the Excel workbook.
- You learned the basic difference between calculated columns and measures, something we discuss in greater detail in the next chapter.
- Whenever the data inside the data model is too detailed, you should create aggregating columns to avoid slicing too many details. You saw a demonstration of this technique on date columns, but it can be easily extended to other types.
- Data that needs to be refreshed needs to be reloaded from the database because PowerPivot tables are a copy of the original data, not a link to them. This might be an issue with very large databases.
- Slicers are graphical tools to make filters for one or more PivotTables. They are useful and good looking, and they allow you to build interactive reports by combining more than one PivotTable in the same Excel workbook.

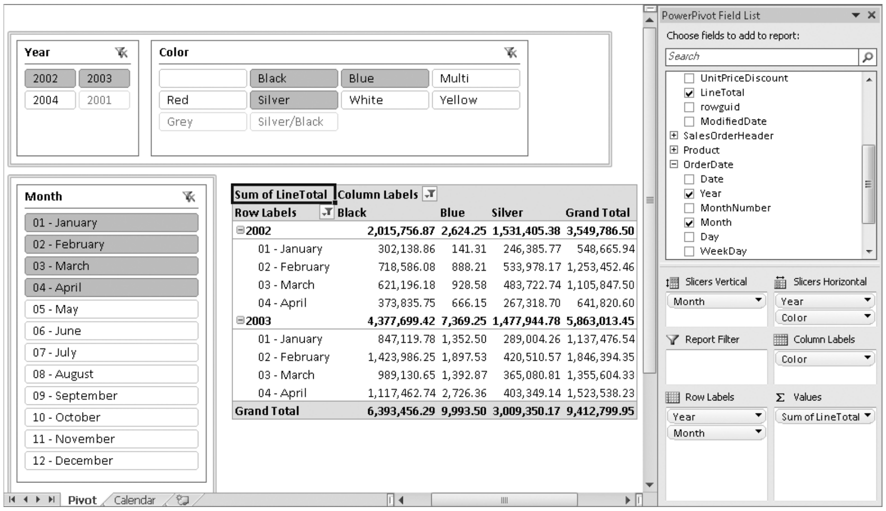
Chapter 7

Date Calculations in DAX

Many analyses of data have to deal with dates. Microsoft SQL Server PowerPivot for Excel offers a number of functions that simplify many calculations on dates that are typical in a business scenario, but using the right function in the right way requires some explanation. As you see in this chapter, the first step in date calculations is to create a separate Dates table that supports most of the requirements.

Working with a Dates Table

In some examples in the previous chapters, we defined calculated columns that extracted parts of the date that we used to group dates, such as the year and the month. This technique might be applied to each table containing a date, but it would quickly become hard to manage. It is better to create a separate table containing a row for each date, using the date as a key to link that Dates table with other tables that contain data related to a date. In this way, you obtain a model wherein all attributes about dates are included in a separate table and are easy to access when you browse data with a PivotTable, as you can see in Figure 7-1.



Sum of LineTotal	Column Labels			
Row Labels	**Black**	**Blue**	**Silver**	**Grand Total**
2002	**2,015,756.87**	**2,624.25**	**1,531,405.38**	**3,549,786.50**
01 - January	302,138.86	141.31	246,385.77	548,665.94
02 - February	718,586.08	888.21	533,978.17	1,253,452.46
03 - March	621,196.18	928.58	483,722.74	1,105,847.50
04 - April	373,835.75	666.15	267,318.70	641,820.60
2003	**4,377,699.42**	**7,369.25**	**1,477,944.78**	**5,863,013.45**
01 - January	847,119.78	1,352.50	289,004.26	1,137,476.54
02 - February	1,423,986.25	1,897.53	420,510.57	1,846,394.35
03 - March	989,130.65	1,392.87	365,080.81	1,355,604.33
04 - April	1,117,462.74	2,726.36	403,349.14	1,523,538.23
Grand Total	**6,393,456.29**	**9,993.50**	**3,009,350.17**	**9,412,799.95**

FIGURE 7-1 PivotTable browsing Order data by using a Dates table named OrderDate.

A Dates table is also useful for making calculations using special DAX functions that operate on Dates. These functions, of which DATEADD is an example, often require that all the days in a given range exist in the data table—otherwise, a missing day might result in a wrong calculation. You might have no sales for a day (in fact, it is pretty common to have no sales on nonworking days), so the separate Dates table allows you to make the right calculations without requiring any modification of the original table that contains measures to analyze.

The only side effect of this technique is that you need to create a Dates table in PowerPivot for each date attribute you want to analyze in a single table because there can be only one relationship between two tables in PowerPivot.

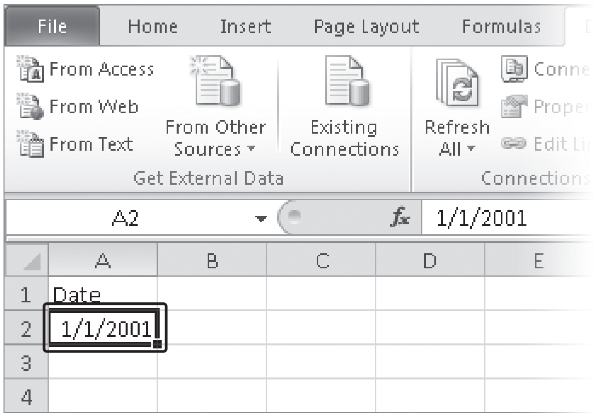
Tip Creating multiple relationships with the same lookup table is not supported in PowerPivot. For this reason, you must duplicate the Dates table whenever you have more date columns that you want to analyze in the same table, such as Order Date and Ship Date attributes in a Sales table.

How to Build a Dates Table

To create a Dates table in PowerPivot, you need a data source that contains at least a column with all days included in the period of time you want to analyze. For example, if the minimum and maximum date contained in Sales data is July 3, 2001 and July 27, 2004, respectively, the range of dates you should consider is between January 1, 2001 and December 31, 2004. In this way, you have all the days for all the years containing sales data.

In Chapter 3, "Introduction to DAX," you saw how to create Day, Month, and Year calculated columns for a Calendar table that has just the Date column as existing data. However, if you do not have an external source providing you with a valid Dates table (such as a corresponding table in SQL Server), we suggest that you create all the calculated columns for a Dates table in Excel. In this way, it will be easier to copy and paste the entire contents of that table into a new one when you have to handle more dates—for example, Order Date and Ship Date—in your PowerPivot model.

To create your Dates table, you can start by typing Date in a cell and 1/1/2001 in the cell below it, as you can see in Figure 7-2.



Date
1/1/2001

FIGURE 7-2 Creating a Dates table in Excel.

Then in the bottom-right corner of the cell containing the 1/1/2001 date (which is highlighted in Figure 7-2), you can drag down until you reach the date of 12/31/2004, as you can see in Figure 7-3.

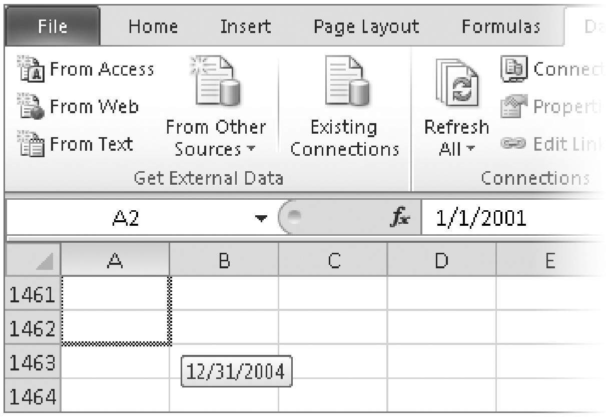
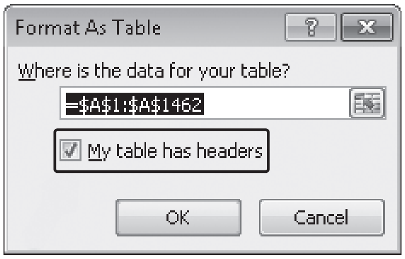


FIGURE 7-3 Selecting end range for creating a Dates table in Excel.

At this point, you can release the mouse. You just created a list of all the days included from the beginning of 2001 to the end of 2004. Now you can click the Date cell, click the Format As Table button on the Home tab of the ribbon, and then confirm that your table has headers, as you can see in Figure 7-4.



Format As Table

Where is the data for your table?

=$A$1:$A$1462

[✓] My table has headers

FIGURE 7-4 Confirming the range of the Table and confirming that your table has headers.

In Figure 7-5, you can see how to give the Calendar name to the table by using the text box available on the Design tab of the Table Tools contextual tab of the Excel ribbon and how to start adding new columns by right-clicking a cell in the table and selecting the InsertTable Column To The Right item from the Insert context menu.

In Figure 7-6, you can see how to define a formula in an empty cell of the new column to calculate the Year. After you type **=YEAR(**, you can click the Date column to get the right syntax to read that column, as shown in Figure 7-6.

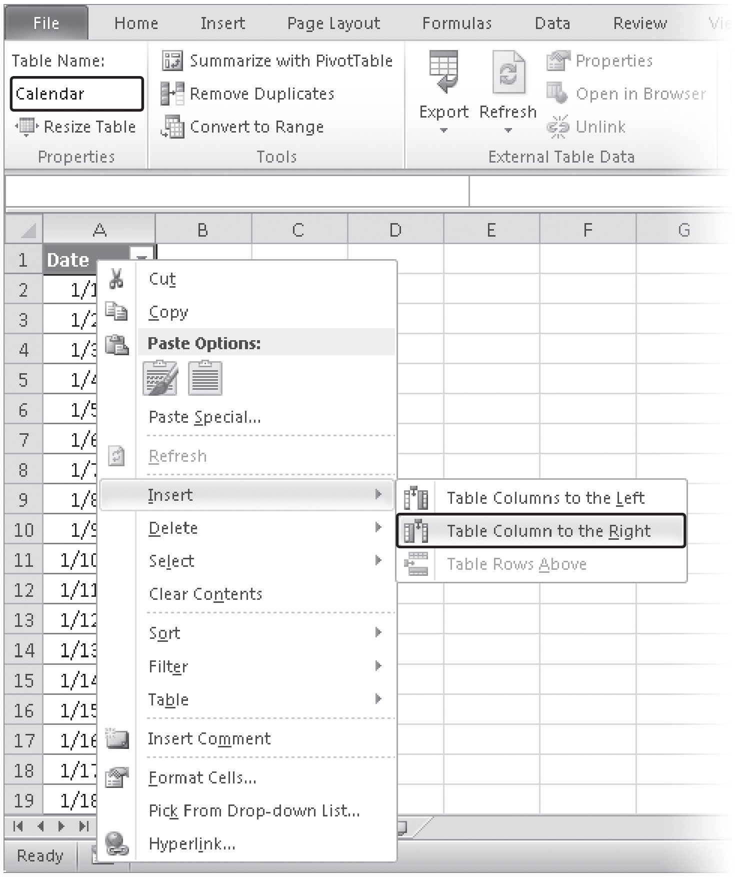
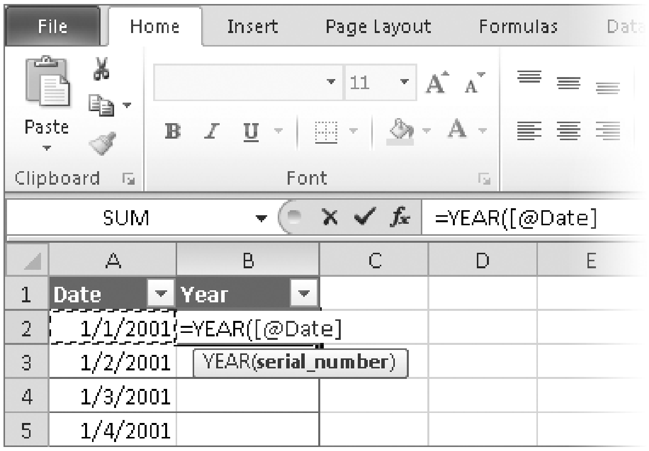


FIGURE 7-5 Inserting a new column in the Dates table.

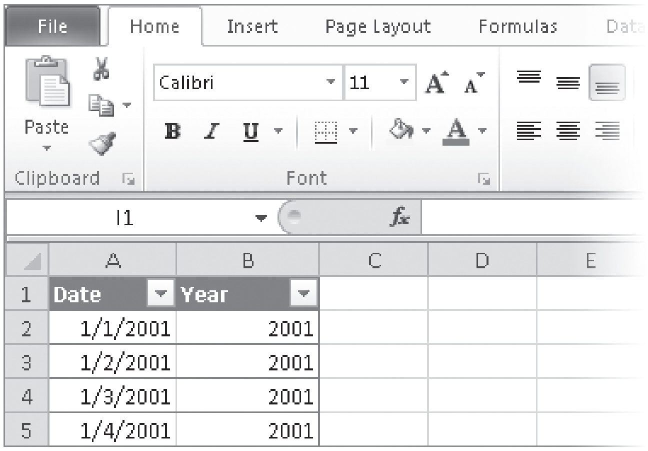


=YEAR([@Date])

	A	B
1	Date	Year
2	1/1/2001	=YEAR([@Date]
3	1/2/2001	
4	1/3/2001	
5	1/4/2001	

FIGURE 7-6 Defining the formula for the Year column in Excel.

At this point, you can type the closing parenthesis and then press Enter. The formula is automatically copied for all the rows of the table in the same column, with the result that you can see in Figure 7-7 (after you adjust the format of the Year column to General in case it was a different format that you copied from the Date column).

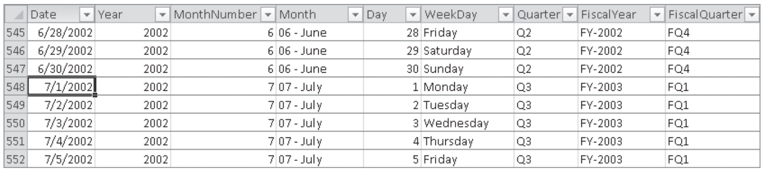


	A	B
1	Date	Year
2	1/1/2001	2001
3	1/2/2001	2001
4	1/3/2001	2001
5	1/4/2001	2001

FIGURE 7-7 The Year column calculated for all the rows.

With this technique, you can define all the columns that are useful for navigating the data that aggregate date in several ways.

Figure 7-8 shows the final result of a complete Dates table with fiscal year starting on July 1. You can find this table in the CH07-01-Calendar.xlsx workbook included on the companion DVD. Table 7-1 contains the formula definitions for all of the columns.



	Date	Year	MonthNumber	Month	Day	WeekDay	Quarter	FiscalYear	FiscalQuarter
545	6/28/2002	2002	6	06 - June	28	Friday	Q2	FY-2002	FQ4
546	6/29/2002	2002	6	06 - June	29	Saturday	Q2	FY-2002	FQ4
547	6/30/2002	2002	6	06 - June	30	Sunday	Q2	FY-2002	FQ4
548	7/1/2002	2002	7	07 - July	1	Monday	Q3	FY-2003	FQ1
549	7/2/2002	2002	7	07 - July	2	Tuesday	Q3	FY-2003	FQ1
550	7/3/2002	2002	7	07 - July	3	Wednesday	Q3	FY-2003	FQ1
551	7/4/2002	2002	7	07 - July	4	Thursday	Q3	FY-2003	FQ1
552	7/5/2002	2002	7	07 - July	5	Friday	Q3	FY-2003	FQ1

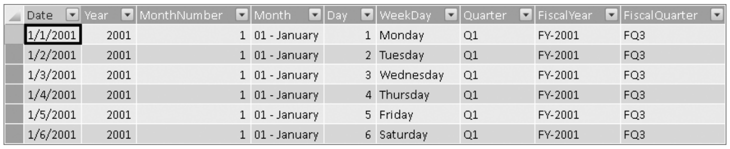
FIGURE 7-8 A complete Dates table with fiscal year starting on July 1.

TABLE 7-1 Formula definitions for the Dates table in Excel.

Column	Formula
Year	=YEAR([@Date])
MonthNumber	=MONTH([@Date])
Month	=TEXT([@Date],"MM - mmmm")
Day	=DAY([@Date])
WeekDay	=TEXT([@Date],"dddd")
Quarter	="Q" & ROUNDUP(MONTH([@Date]) /3,0)
FiscalYear	="FY-" & [@Year]+IF([@MonthNumber]<7,0,1)
FiscalQuarter	="FQ" &MOD(CEILING(22+[@MonthNumber]-6-1,3)/3,4)+1

Note Table 7-1 does not include a column for week number. The table omits it because there are several techniques for calculating the week number in a year, and different businesses have different ways to make this calculation. More important, sometimes a week belongs to a year that is different from the calendar year—the fiscal year, for example—even if only for a few days of a year. In that case, you also need to define a WeekYear column that must be used for browsing the weeks in a meaningful way. We preferred not to include a specific week calculation to keep the Dates table simple and to avoid possible confusion introducing an algorithm that might be different than that used in your company.

Now you can import this table in PowerPivot as a linked table. The result is shown in Figure 7-9.



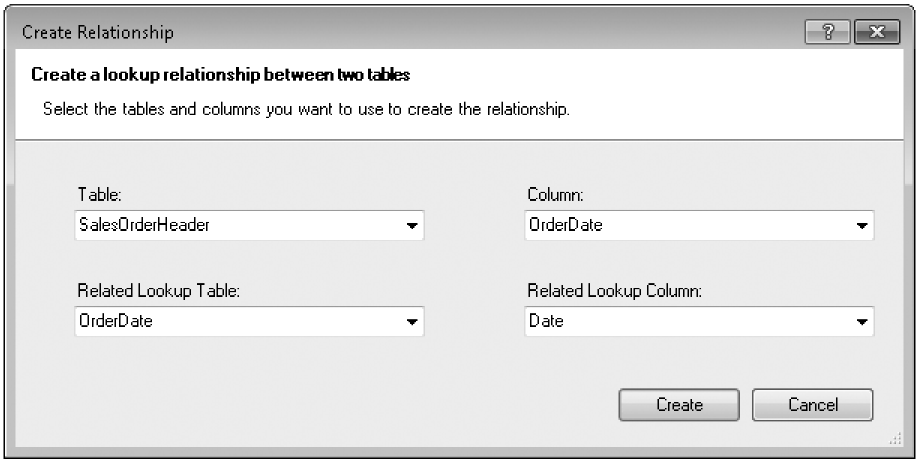
Date	Year	MonthNumber	Month	Day	WeekDay	Quarter	FiscalYear	FiscalQuarter
1/1/2001	2001	1	01 - January	1	Monday	Q1	FY-2001	FQ3
1/2/2001	2001	1	01 - January	2	Tuesday	Q1	FY-2001	FQ3
1/3/2001	2001	1	01 - January	3	Wednesday	Q1	FY-2001	FQ3
1/4/2001	2001	1	01 - January	4	Thursday	Q1	FY-2001	FQ3
1/5/2001	2001	1	01 - January	5	Friday	Q1	FY-2001	FQ3
1/6/2001	2001	1	01 - January	6	Saturday	Q1	FY-2001	FQ3

FIGURE 7-9 The Dates table imported in PowerPivot as a linked table.

You can see that the month name contains the month number in front of it, so December is described as *12 – December*. It is useful to have the month names automatically sorted. However, if you want to sort month names but also want to avoid the initial number, please take a look at the section "Custom Sorting in PivotTables" in Chapter 8, "Mastering PivotTables," where we describe how to sort columns of a Dates table in a PivotTable.

You might want to change the data types of some columns in the Dates table. Whenever you import the Excel table into PowerPivot, columns like Year, MonthNumber, and Day are usually defined as Whole Number data types. For this reason, when you select one of these columns in the PivotTable, the selected attribute is placed by default in the Values area of the PivotTable and is aggregated when you use the Sum function. You might prefer to change the data types of these columns to Text so that by default they are used to group data in rows.

If you want to test your new Calendar table, you should now import the SalesOrderHeader, SalesOrderDetail, Customer, and Product tables from the AdventureWorks database into the same PowerPivot model. Relationships between these tables are automatically detected during the import. At this point, you need to create a relationship between the OrderDate field of the SalesOrderHeader table and the Date field of the Calendar table you just imported. Before starting, in PowerPivot you have to rename the Calendar table to OrderDate so that it expresses the dates it represents. Then you click the Create Relationship button on the Design tab of the ribbon and fill in the dialog box, as shown in Figure 7-10.



Create Relationship

Create a lookup relationship between two tables

Select the tables and columns you want to use to create the relationship.

Table: SalesOrderHeader

Column: OrderDate

Related Lookup Table: OrderDate

Related Lookup Column: Date

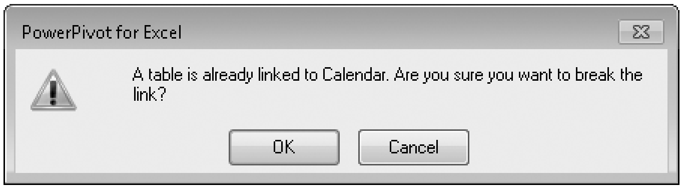
FIGURE 7-10 Create a relationship between the SalesOrderHeader and OrderDate tables.

At this point, the model is ready to browse data, as you saw at the beginning of this chapter, in Figure 7-1.

Working with Multiple Dates Tables

In the model you saw in the previous section, each Order has several dates. In case you want to analyze not only the Order Date but also the Ship Date, you need to define a second table in PowerPivot because the same table (that is, the Dates table) cannot have more than one relationship with a given table (SalesOrderHeader).

At this point, you have two options. You can either create a new linked table starting from the same table you used before (shown in Figure 7-8) or copy that table into Excel and create the linked table starting from this copy. The first option is not the best one because in PowerPivot you can have only one linked table for a given Excel table. If you try to create a linked table starting from the same Calendar table you defined before, the warning message shown in Figure 7-11 appears.



PowerPivot for Excel

A table is already linked to Calendar. Are you sure you want to break the link?

FIGURE 7-11 A warning against trying to create a linked table for an Excel table already used as a linked table.

If you continue creating a linked table this way, you cannot update the OrderDate table anymore. If you create a model that must be refreshed over time and that is likely to have a life cycle longer than the current year, you are better off using another way, which allows future updates.

The second option requires you to copy and paste the existing Calendar table in Excel. Before you do that, you should rename the Calendar table in Excel, using the same name we used for the corresponding linked table in PowerPivot, which is OrderDate. To do that, you can type the OrderDate name into the Table Name text box available on the Design tab of the Table Tools contextual tab of the Excel ribbon, as you can see in Figure 7-12.

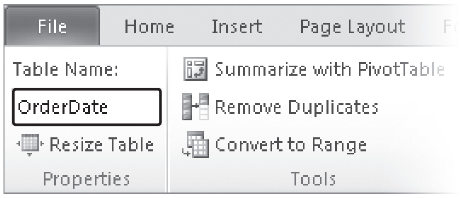
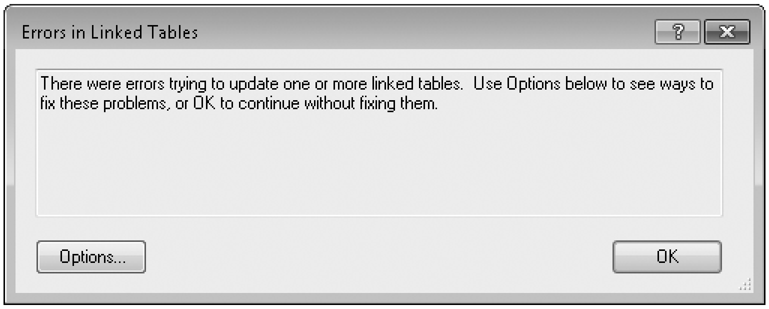


FIGURE 7-12 Renaming the table OrderDate, in Excel.

At this point, if you try to Update the OrderDate linked table in PowerPivot, you get the error message shown in Figure 7-13.



There were errors trying to update one or more linked tables. Use Options below to see ways to fix these problems, or OK to continue without fixing them.

FIGURE 7-13 The error message you get when you try to update OrderDate after changing the name of the underlying Excel table.

When you click the Options button, you can select the Change Excel Table Name option. Then you choose the OrderDate table in the combo box that shows the available tables in Excel, as you can see in Figure 7-14.

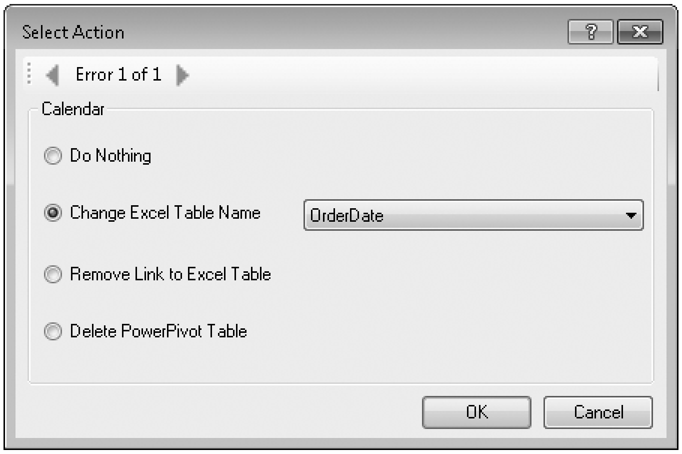


FIGURE 7-14 Fixing the error in LinkedTable by selecting the correct underlying Excel table.

Now you can copy the OrderDate table in Excel into a new one that we call ShipDate. You might do this by selecting the whole table, copying it, and then pasting it into an empty space of your Excel workbook. However, another option is to use a single dedicated Excel worksheet for each table like these so that you can simply duplicate the worksheet into a new one. In this way, whenever you need to add columns or rows to the table, you never have to move other existing tables. Moreover, tables are easily accessible when you click the corresponding worksheet name in Excel.

To create a copy of the worksheet containing the OrderDate table, you have to right-click the OrderDate label and select Move Or Copy from the context menu that you can see in Figure 7-15.

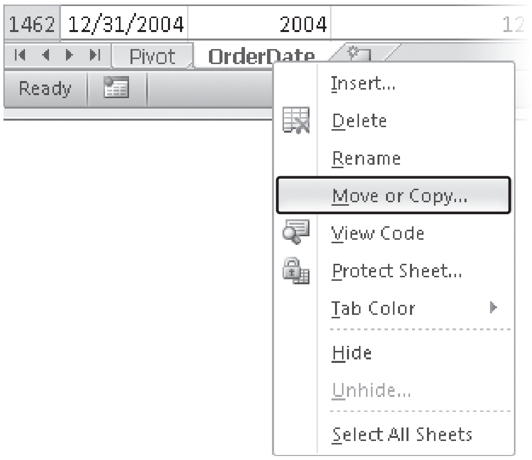
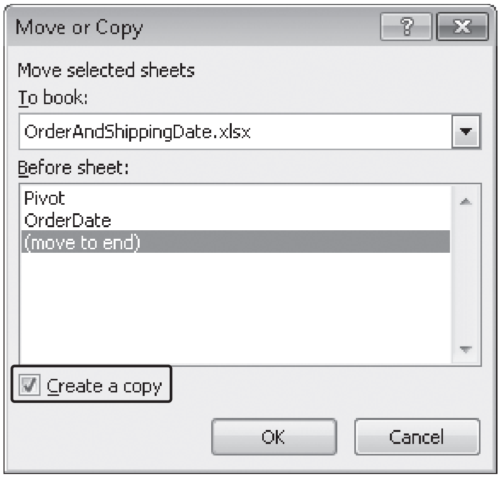


FIGURE 7-15 Choosing Move Or Copy from the context menu.

The selection displays the dialog box shown in Figure 7-16, in which you have to select the Create A Copy check box and choose the position of the new sheet.



Move or Copy

Move selected sheets

To book:

OrderAndShippingDate.xlsx

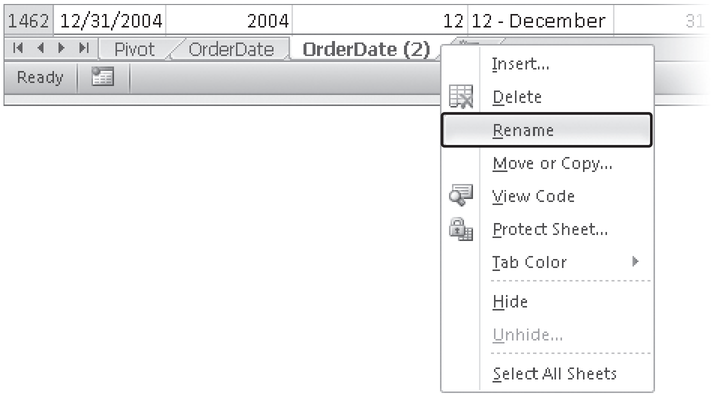
Before sheet:

Pivot
OrderDate
(move to end)

☑ Create a copy

FIGURE 7-16 Selecting the option to create a copy of the worksheet to place at the end of the list.

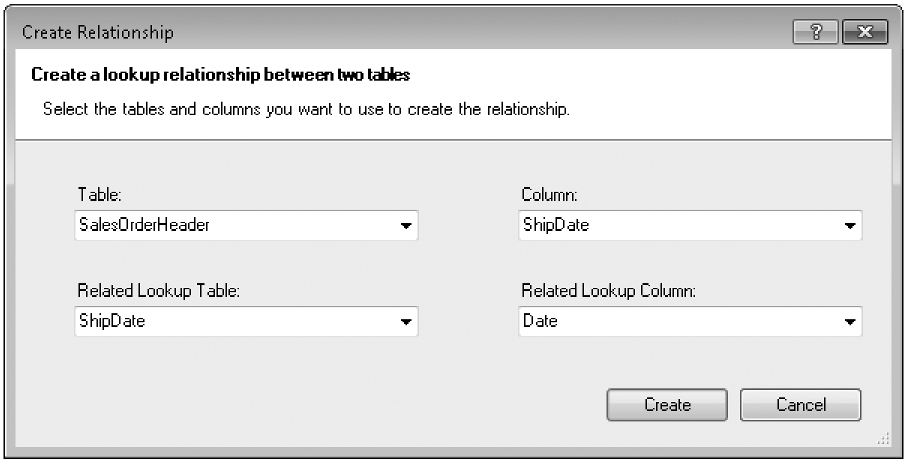
At this point, you rename both the table (using the same procedure you saw already in Figure 7-12) and the worksheet (by right-clicking on the OrderDate (2) label and then selecting Rename from the context menu that you can see in Figure 7-17); you use the new ShipDate name.



1462	12/31/2004	2004	12	12 - December	31

FIGURE 7-17 The Rename option in the context menu.

Finally, you can create a linked table for the ShipDate table by clicking the Create Linked Table button on the PowerPivot ribbon. Again, you have to create a relationship in PowerPivot between the SalesOrderHeader and ShipDate tables, by using the ShipDate column of the SalesOrderHeader table this time, as you can see in Figure 7-18.



Select the tables and columns you want to use to create the relationship.

Table: SalesOrderHeader
Column: ShipDate
Related Lookup Table: ShipDate
Related Lookup Column: Date

FIGURE 7-18 Creating a relationship between the SalesOrderHeader and OrderDate tables.

You can find the resulting model in the CH07-02-OrderAndShippingDate.xlsx workbook included on the companion DVD. However, as you can see in the next section, duplicating tables might not be enough. Because Excel does not show the table name to which a column belongs when you use it for Slicers and Filters, you might want to add a prefix to your columns. It is better to do that directly in the source Excel table rather than renaming the columns in PowerPivot only so that the overall model is simpler to understand.

Dates Columns in Different Tables

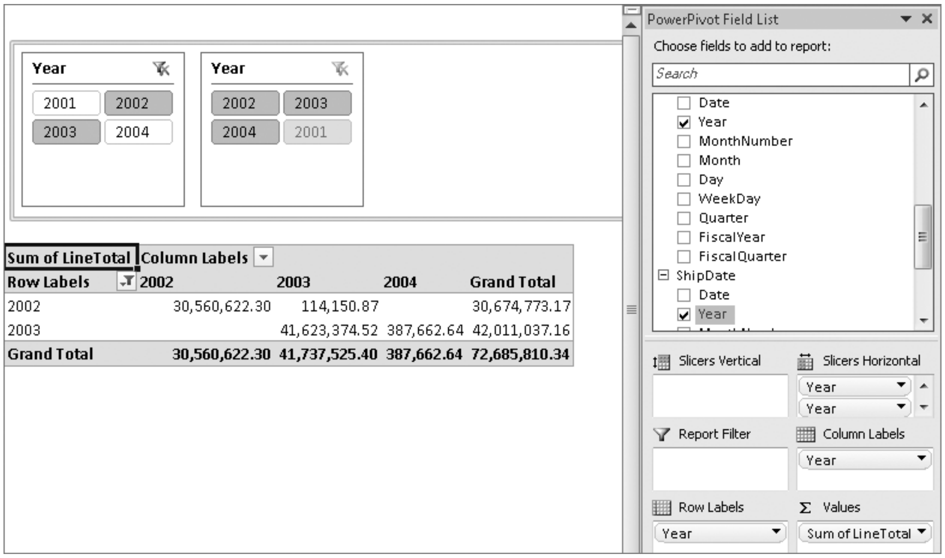
You must define a separate Dates table to distinguish the semantics of different dates in your data. This is certainly true whenever different date columns belong to the same table, as in the case of the OrderDate and ShipDate columns in the SalesOrderHeader table. However, when you have dates columns in different tables, you have to evaluate whether the semantics of these dates is the same or not.

Every time you have a different role for a date, you have to create separate Dates tables to browse data, just as you saw in this section. On the other hand, you have to use the same Dates table whenever these dates have the same meaning, at least for your analysis.

For example, if you have an OrderDate in the Sales table and a CallDate in a CallCenterCalls table, you might decide to create two separate data tables named OrderDate and CallDate. But you might also want to create a single Dates table that connects both events, which would ease the browsing over time of data from both tables in the same report. If you have no other dates in your model, no ambiguities arise from that arrangement, but if there are other dates involved in the same model, you should consider a separate model for doing correlation analysis, avoiding misleading names in your model.

Differentiating Columns in Multiple Dates Tables

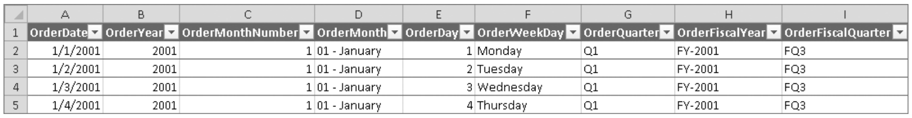
Duplicating the same table, such as a Dates table, multiple times in a PowerPivot model makes the resulting PivotTable difficult to read whenever the same attributes are used from different tables. For example, in Figure 7-19, you can see a PivotTable in which the Year from OrderDate has been put in rows and in the first slicer, and the Year from ShipDate has been put in columns and in the second slicer. The problem is that there is no evidence of the table that a column belongs to whenever it is moved into slicers, filters, rows, or columns of the PivotTable. The final model for the example of this section is available in the CH07-03-PrefixedDateColumns.xlsx workbook included on the companion DVD.



Sum of LineTotal	Column Labels			
Row Labels	2002	2003	2004	Grand Total
2002	30,560,622.30	114,150.87		30,674,773.17
2003		41,623,374.52	387,662.64	42,011,037.16
Grand Total	**30,560,622.30**	**41,737,525.40**	**387,662.64**	**72,685,810.34**

FIGURE 7-19 Columns with the same name from different tables are not recognizable in a PivotTable.

So in case you create a model with multiple copies of the same tables, you should differentiate the names of the columns so that they are immediately recognizable in a report. You can edit the table names in Excel by adding a prefix to each column. In Figure 7-20, you can see the heading of the OrderDate table, wherein each column has been prefixed with the word *Order*. You can do the same for the ShipDate column by using the *Ship* prefix.

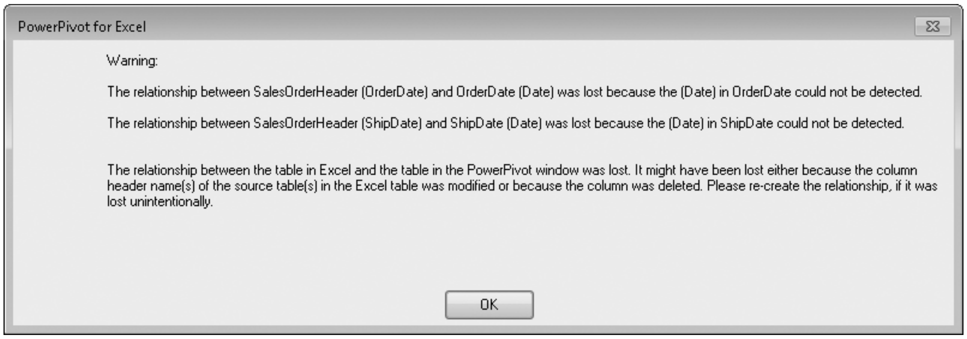


	A	B	C	D	E	F	G	H	I
1	OrderDate	OrderYear	OrderMonthNumber	OrderMonth	OrderDay	OrderWeekDay	OrderQuarter	OrderFiscalYear	OrderFiscalQuarter
2	1/1/2001	2001	1	01 - January	1	Monday	Q1	FY-2001	FQ3
3	1/2/2001	2001	1	01 - January	2	Tuesday	Q1	FY-2001	FQ3
4	1/3/2001	2001	1	01 - January	3	Wednesday	Q1	FY-2001	FQ3
5	1/4/2001	2001	1	01 - January	4	Thursday	Q1	FY-2001	FQ3

FIGURE 7-20 The columns of OrderDate prefixed with *Order*.

Broken Relationships After Columns Are Renamed

Renaming a column that is part of a relationship breaks that relationship. For example, if you try to update the linked tables after renaming the Date columns, you receive the error message shown in Figure 7-21. The existing relationships were based on a column (Date) that does not exist anymore.


PowerPivot for Excel

Warning:

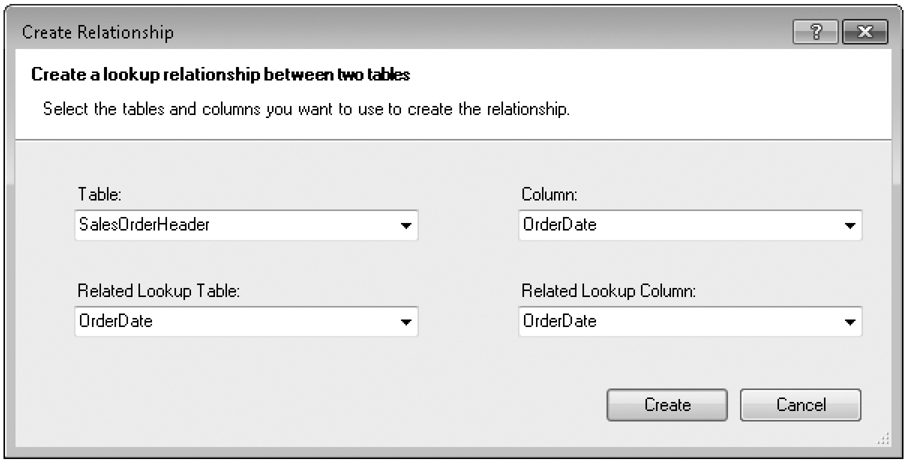
The relationship between SalesOrderHeader (OrderDate) and OrderDate (Date) was lost because the (Date) in OrderDate could not be detected.

The relationship between SalesOrderHeader (ShipDate) and ShipDate (Date) was lost because the (Date) in ShipDate could not be detected.

The relationship between the table in Excel and the table in the PowerPivot window was lost. It might have been lost either because the column header name(s) of the source table(s) in the Excel table was modified or because the column was deleted. Please re-create the relationship, if it was lost unintentionally.

FIGURE 7-21 Relationships lost when a column is renamed.

In this example, you need to re-create the relationships between SalesOrderHeader and the OrderDate and ShipDate tables (through the columns OrderDate and ShipDate, respectively). In Figure 7-22, you can see the definition of the relationship for OrderDate; the one for ShipDate simply uses the corresponding names for the lookup table name and columns used to define the relationships.


Create Relationship

Create a lookup relationship between two tables

Select the tables and columns you want to use to create the relationship.

Table: SalesOrderHeader

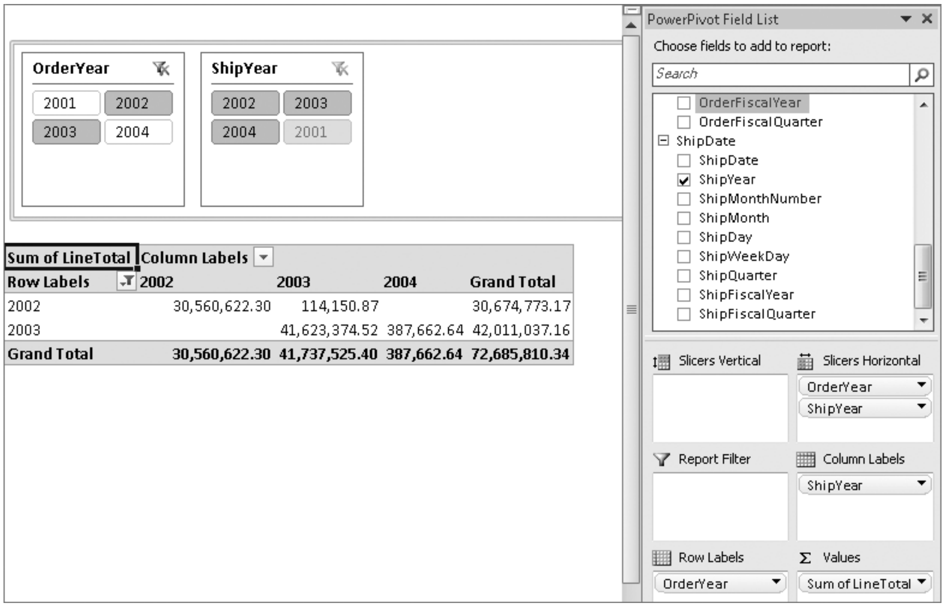
Column: OrderDate

Related Lookup Table: OrderDate

Related Lookup Column: OrderDate

FIGURE 7-22 Recreating relationships by using the new lookup column name.

After you rename your column, you can create a report similar to the one you saw in Figure 7-19, but this time, use more meaningful names for columns that were ambiguous before. You can see the result of such a process in Figure 7-23.



Sum of LineTotal	Column Labels			
Row Labels	2002	2003	2004	Grand Total
2002	30,560,622.30	114,150.87		30,674,773.17
2003		41,623,374.52	387,662.64	42,011,037.16
Grand Total	30,560,622.30	41,737,525.40	387,662.64	72,685,810.34

PowerPivot Field List

Choose fields to add to report:

Search

- ☐ OrderFiscalYear
- ☐ OrderFiscalQuarter
- ⊟ ShipDate
  - ☐ ShipDate
  - ☑ ShipYear
  - ☐ ShipMonthNumber
  - ☐ ShipMonth
  - ☐ ShipDay
  - ☐ ShipWeekDay
  - ☐ ShipQuarter
  - ☐ ShipFiscalYear
  - ☐ ShipFiscalQuarter

Slicers Vertical

Slicers Horizontal: OrderYear, ShipYear

Report Filter

Column Labels: ShipYear

Row Labels: OrderYear

Values: Sum of LineTotal

FIGURE 7-23 Column prefixes are more recognizable both in the slicers and in the PowerPivot Field List.

We suggest that you use column prefixes every time you have the same column name in different tables—not just for Dates tables.

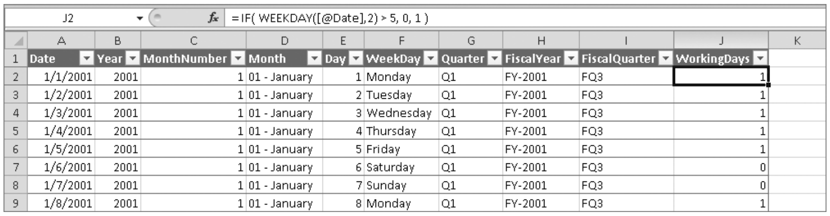
Calculating Working Days

Now that you have learned how to create a calendar table, it is worth pointing out some columns that can be very useful in data analysis and that can be conveniently stored in the calendar table. For example, you might be interested in defining a measure that calculates the average of sales per working days in a given period. (You can find the complete example in the CH07-04-WorkingDays.xlsx workbook included on the companion DVD.) To do that, you have to calculate the number of working days, which in turn requires knowing whether a day is a working day. The simpler way to do this is to add a WorkingDays column to the Excel OrderDate table. That column should have the value 1 for working days, and 0 for holidays, weekends, and other nonworking days. Instead of compiling this column by hand, you might define it by using the following Excel formula that assigns 1 to all week days between Monday and Friday, leaving 0 to Saturday and Sunday:

```
= IF( WEEKDAY([@Date],2) > 5, 0, 1 )
```

Tip You might want to use a separate NonWorkingDays table to configure the working days in the week. In that case, you use the VLOOKUP Excel function in the preceding expression. You see a similar example later in this section, when we discuss how to create a table that defines public holidays that must be differentiated from working days.

This formula is automatically copied into all the rows of the OrderDate table, as you can see in Figure 7-24.

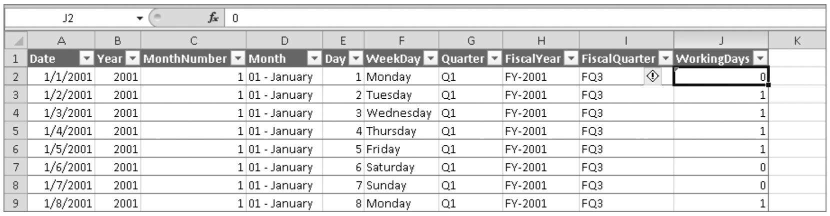


J2 = IF(WEEKDAY([@Date],2) > 5, 0, 1)

Date	Year	MonthNumber	Month	Day	WeekDay	Quarter	FiscalYear	FiscalQuarter	WorkingDays
1/1/2001	2001	1	01 - January	1	Monday	Q1	FY-2001	FQ3	1
1/2/2001	2001	1	01 - January	2	Tuesday	Q1	FY-2001	FQ3	1
1/3/2001	2001	1	01 - January	3	Wednesday	Q1	FY-2001	FQ3	1
1/4/2001	2001	1	01 - January	4	Thursday	Q1	FY-2001	FQ3	1
1/5/2001	2001	1	01 - January	5	Friday	Q1	FY-2001	FQ3	1
1/6/2001	2001	1	01 - January	6	Saturday	Q1	FY-2001	FQ3	0
1/7/2001	2001	1	01 - January	7	Sunday	Q1	FY-2001	FQ3	0
1/8/2001	2001	1	01 - January	8	Monday	Q1	FY-2001	FQ3	1

FIGURE 7-24 The WorkingDays column added to the OrderDate table in Excel.

You can modify single values for other nonworking days, such as public holidays, overriding the formula with a forced fixed value (usually 0) just for these days. For example, in Figure 7-25, you can see the value for January 1, 2001 overridden by a 0 value, whereas the following dates are still evaluated by the formula we defined before.



J2 0

Date	Year	MonthNumber	Month	Day	WeekDay	Quarter	FiscalYear	FiscalQuarter	WorkingDays
1/1/2001	2001	1	01 - January	1	Monday	Q1	FY-2001	FQ3	0
1/2/2001	2001	1	01 - January	2	Tuesday	Q1	FY-2001	FQ3	1
1/3/2001	2001	1	01 - January	3	Wednesday	Q1	FY-2001	FQ3	1
1/4/2001	2001	1	01 - January	4	Thursday	Q1	FY-2001	FQ3	1
1/5/2001	2001	1	01 - January	5	Friday	Q1	FY-2001	FQ3	1
1/6/2001	2001	1	01 - January	6	Saturday	Q1	FY-2001	FQ3	0
1/7/2001	2001	1	01 - January	7	Sunday	Q1	FY-2001	FQ3	0
1/8/2001	2001	1	01 - January	8	Monday	Q1	FY-2001	FQ3	1

FIGURE 7-25 The value 0 overriding the formula for January 1, 2001.

Note The warning shown in Figure 7-25 to the left of cell J2 indicates a possible inconsistency in a column that contains a formula. You can click the Ignore Error item in the context menu to turn off the warning.

You can update the OrderDate linked table in PowerPivot, and the WorkingDays column shows up in the PivotTable too. At this point, you can define a measure belonging to the SalesOrderDetail table named DailySales, which divides the sum of LineTotal by the sum of working days, as you can see in Figure 7-26.

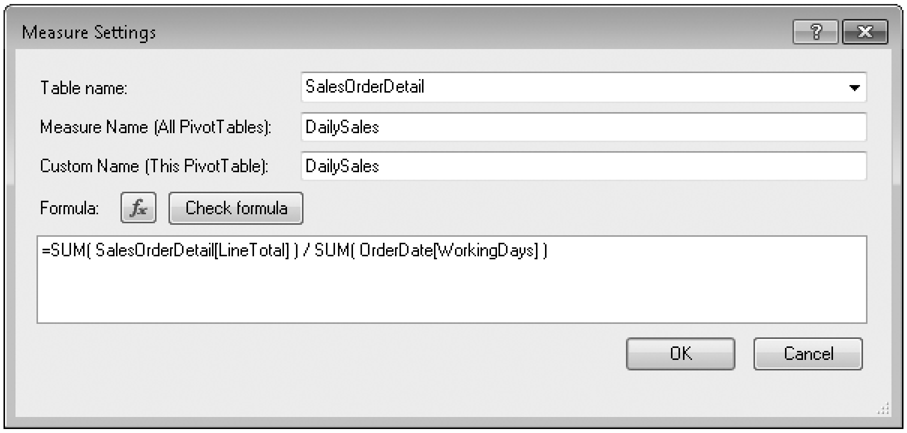


Table name: SalesOrderDetail

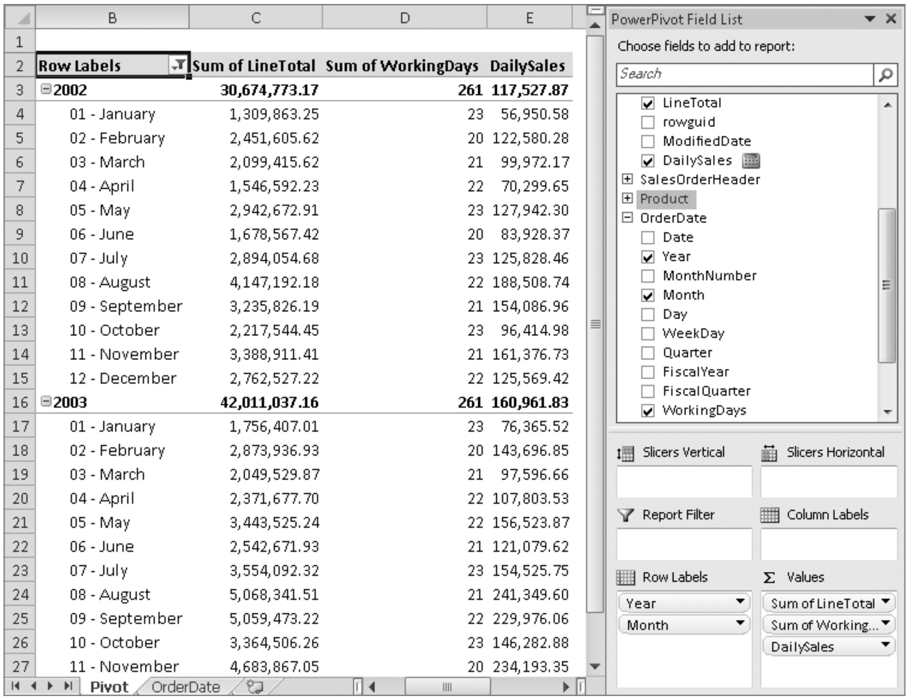
Measure Name (All PivotTables): DailySales

Custom Name (This PivotTable): DailySales

```
=SUM( SalesOrderDetail[LineTotal] ) / SUM( OrderDate[WorkingDays] )
```

FIGURE 7-26 The definition of DailySales measure.

The final result is shown in Figure 7-27, where both WorkingDays and DailySales measures are exposed in the PivotTable. However, in a real report, you usually do not show the working days number but just the average measures, such as Daily Sales.



Row Labels	Sum of LineTotal	Sum of WorkingDays	DailySales
2002	**30,674,773.17**	**261**	**117,527.87**
01 - January	1,309,863.25	23	56,950.58
02 - February	2,451,605.62	20	122,580.28
03 - March	2,099,415.62	21	99,972.17
04 - April	1,546,592.23	22	70,299.65
05 - May	2,942,672.91	23	127,942.30
06 - June	1,678,567.42	20	83,928.37
07 - July	2,894,054.68	23	125,828.46
08 - August	4,147,192.18	22	188,508.74
09 - September	3,235,826.19	21	154,086.96
10 - October	2,217,544.45	23	96,414.98
11 - November	3,388,911.41	21	161,376.73
12 - December	2,762,527.22	22	125,569.42
2003	**42,011,037.16**	**261**	**160,961.83**
01 - January	1,756,407.01	23	76,365.52
02 - February	2,873,936.93	20	143,696.85
03 - March	2,049,529.87	21	97,596.66
04 - April	2,371,677.70	22	107,803.53
05 - May	3,443,525.24	22	156,523.87
06 - June	2,542,671.93	21	121,079.62
07 - July	3,554,092.32	23	154,525.75
08 - August	5,068,341.51	21	241,349.60
09 - September	5,059,473.22	22	229,976.06
10 - October	3,364,506.26	23	146,282.88
11 - November	4,683,867.05	20	234,193.35

FIGURE 7-27 PivotTable showing results for WorkingDays and DailySales.

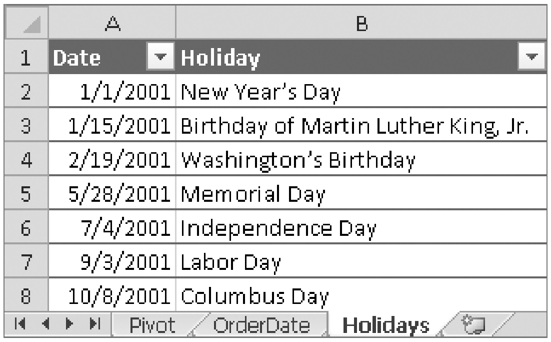
Working Days in Different Countries

In the examples included in this section, we are making some wrong assumptions for a database like AdventureWorks, but that might be good for your own data. In fact, because we are accounting for sales in stores located in different countries, we should consider a different number of working days for each of these countries. This would make the DailySales measure harder to calculate. In fact, we should accomplish all this as well:

- Define a separate table to calculate working days, based on country and date.
- Make a calculation of the required DailySales average by country.
- Aggregate that number for all countries by using a weighted average based on the sales amount for that country in a given period.

Although this calculation is still possible, it is very complex and it is seldom used because this measure is probably not the same for different countries in the same report.

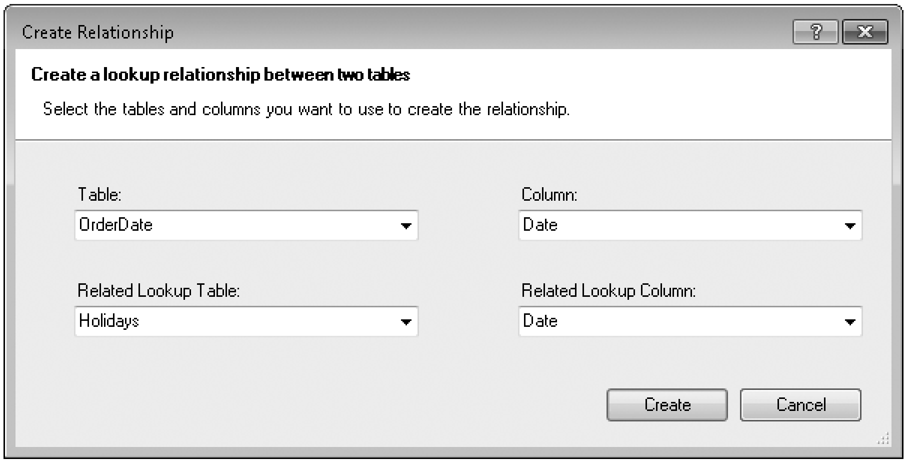
Our technique up to now is really error prone because we write directly into a cell a value of 0 to indicate a holiday, without any further explanation. If we make an error, it is really hard to identify; furthermore, we make no distinction between weekend days (which are automatically calculated) and holidays. A better solution is to define a separate Holidays table, which is easier to check and to maintain because it moves into a single calculated column the logic to merge weekend evaluation and holiday definition using a single formula. In Figure 7-28, you can see such a Holidays table, defined in Excel.



	A	B
1	Date	Holiday
2	1/1/2001	New Year's Day
3	1/15/2001	Birthday of Martin Luther King, Jr.
4	2/19/2001	Washington's Birthday
5	5/28/2001	Memorial Day
6	7/4/2001	Independence Day
7	9/3/2001	Labor Day
8	10/8/2001	Columbus Day

FIGURE 7-28 Holidays table in Excel.

You can import this Holidays table as a linked table in PowerPivot and define a relationship between the OrderDate and Holidays tables, as shown in Figure 7-29.



Create Relationship

Create a lookup relationship between two tables

Select the tables and columns you want to use to create the relationship.

Table: OrderDate

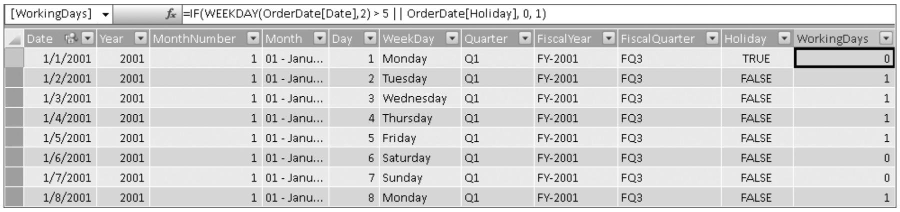
Column: Date

Related Lookup Table: Holidays

Related Lookup Column: Date

FIGURE 7-29 The relationship between the OrderDate and Holidays tables.

You can remove the WorkingDays column because you are moving the whole logic into PowerPivot. Because you need to use the RELATE function to get holiday information in PowerPivot, you should move all the business logic into one simple place: avoid splitting it half and half between Excel and and PowerPivot. After you update the OrderDate table in PowerPivot by removing the WorkingDays data column, you can define a new WorkingDays calculated column by using the DAX formula that you can see in Figure 7-30.



[WorkingDays] =IF(WEEKDAY(OrderDate[Date],2) > 5 || OrderDate[Holiday], 0, 1)

Date	Year	MonthNumber	Month	Day	WeekDay	Quarter	FiscalYear	FiscalQuarter	Holiday	WorkingDays
1/1/2001	2001	1	01 - Janu...	1	Monday	Q1	FY-2001	FQ3	TRUE	0
1/2/2001	2001	1	01 - Janu...	2	Tuesday	Q1	FY-2001	FQ3	FALSE	1
1/3/2001	2001	1	01 - Janu...	3	Wednesday	Q1	FY-2001	FQ3	FALSE	1
1/4/2001	2001	1	01 - Janu...	4	Thursday	Q1	FY-2001	FQ3	FALSE	1
1/5/2001	2001	1	01 - Janu...	5	Friday	Q1	FY-2001	FQ3	FALSE	1
1/6/2001	2001	1	01 - Janu...	6	Saturday	Q1	FY-2001	FQ3	FALSE	0
1/7/2001	2001	1	01 - Janu...	7	Sunday	Q1	FY-2001	FQ3	FALSE	0
1/8/2001	2001	1	01 - Janu...	8	Monday	Q1	FY-2001	FQ3	FALSE	1

FIGURE 7-30 The WorkingDays calculated column in the OrderDate table.

Let us examine the DAX formula in Figure 7-30. First of all, you can see a new Holiday calculated column defined by the following formula:

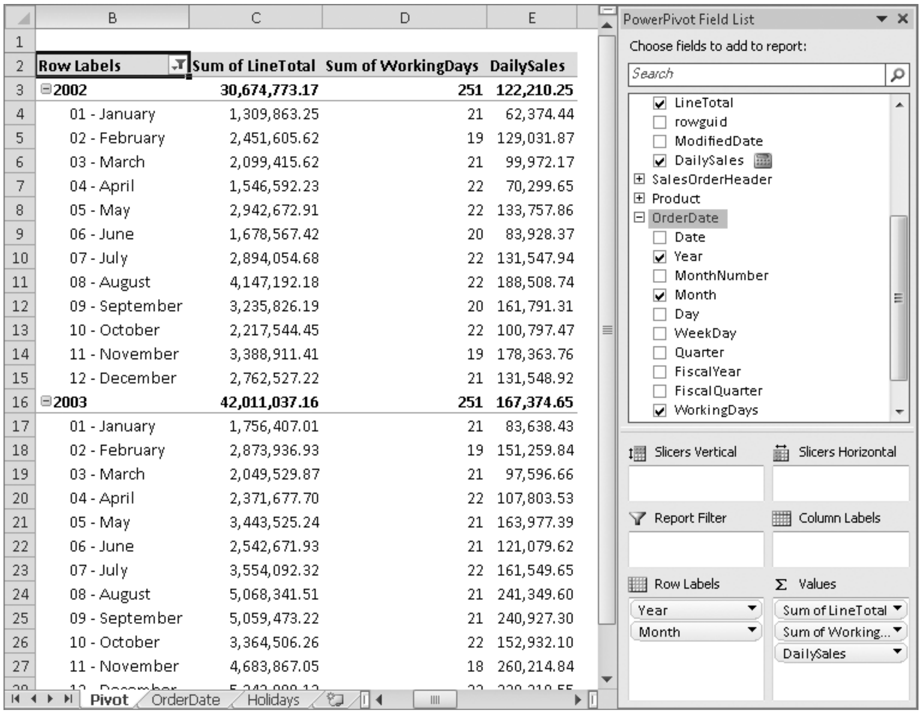
```
Holiday = IF( ISBLANK( RELATED(Holidays[Date]) ), FALSE, TRUE )
```

The Holiday column has a TRUE value for every day that corresponds to a holiday in the Holidays table. Using this information, we extend the previously defined Excel formula that considers whether a nonworking day is a Saturday, a Sunday, or a holiday by using the following DAX formula:

```
WorkingDays = IF( WEEKDAY( OrderDate[Date], 2 ) > 5 || OrderDate[Holiday],
                  0, 1)
```

Tip Remember that the || operator corresponds to the OR Boolean operator, which also can be written using the OR function, both in PowerPivot and Excel.

Finally, you can browse data with the right calculation of WorkingDays, according to the Holidays table we included in the model. In Figure 7-31, you can see the resulting PivotTable, which you can find along with the complete model in the CH07-05-WorkingDays-HolidaysTable.xlsx workbook included on the companion DVD.



Row Labels	Sum of LineTotal	Sum of WorkingDays	DailySales
2002	**30,674,773.17**	**251**	**122,210.25**
01 - January	1,309,863.25	21	62,374.44
02 - February	2,451,605.62	19	129,031.87
03 - March	2,099,415.62	21	99,972.17
04 - April	1,546,592.23	22	70,299.65
05 - May	2,942,672.91	22	133,757.86
06 - June	1,678,567.42	20	83,928.37
07 - July	2,894,054.68	22	131,547.94
08 - August	4,147,192.18	22	188,508.74
09 - September	3,235,826.19	20	161,791.31
10 - October	2,217,544.45	22	100,797.47
11 - November	3,388,911.41	19	178,363.76
12 - December	2,762,527.22	21	131,548.92
2003	**42,011,037.16**	**251**	**167,374.65**
01 - January	1,756,407.01	21	83,638.43
02 - February	2,873,936.93	19	151,259.84
03 - March	2,049,529.87	21	97,596.66
04 - April	2,371,677.70	22	107,803.53
05 - May	3,443,525.24	21	163,977.39
06 - June	2,542,671.93	21	121,079.62
07 - July	3,554,092.32	22	161,549.65
08 - August	5,068,341.51	21	241,349.60
09 - September	5,059,473.22	21	240,927.30
10 - October	3,364,506.26	22	152,932.10
11 - November	4,683,867.05	18	260,214.84

PowerPivot Field List

Choose fields to add to report:

- [✓] LineTotal
- [ ] rowguid
- [ ] ModifiedDate
- [✓] DailySales
- SalesOrderHeader
- Product
- OrderDate
  - [ ] Date
  - [✓] Year
  - [ ] MonthNumber
  - [✓] Month
  - [ ] Day
  - [ ] WeekDay
  - [ ] Quarter
  - [ ] FiscalYear
  - [ ] FiscalQuarter
  - [✓] WorkingDays

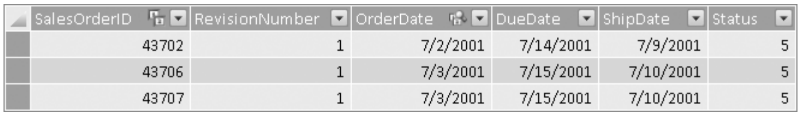
Row Labels: Year, Month

Values: Sum of LineTotal, Sum of Working..., DailySales

FIGURE 7-31 The PivotTable showing final results using the Holidays table support.

Another common calculation involving working days is the delta between two dates. For example, in the SalesOrderHeader table of the model used in this chapter, there are three dates, which you can also see in Figure 7-32:

- OrderDate: The date of the order
- DueDate: When the customer expects the order to be delivered
- ShipDate: The date of order shipment



SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status
43702	1	7/2/2001	7/14/2001	7/9/2001	5
43706	1	7/3/2001	7/15/2001	7/10/2001	5
43707	1	7/3/2001	7/15/2001	7/10/2001	5

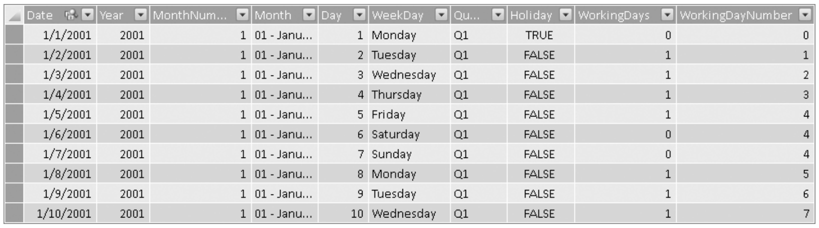
FIGURE 7-32 The Dates column in the SalesOrderHeader table.

Calculating whether an order has been shipped on time seems pretty easy: you should just compare the DueDate and ShipDate columns. However, if you consider a standard delivery time of four working days, you should calculate how many orders have been shipped after DueDate minus four working days. This calculation requires the support of the Dates table. The complete model of the following example is available in the CH07-06-DeliveryDays.xlsx workbook included on the companion DVD.

To make the calculation, we need to add a calculated column in the SalesOrderHeader table that calculates for each order the difference (in working dates) between the two dates. You can create a WorkingDayNumber calculated column in the Dates table that has the following formula:

```
WorkingDayNumber =SUMX( FILTER( OrderDate,
                                OrderDate[Date] <= EARLIER(OrderDate[Date]) ),
                        OrderDate[WorkingDays] )
```

This number calculates for each day the number of working days elapsed since the first date in the Dates table. In Figure 7-33, you can see how this number is calculated for a few rows.



Date	Year	MonthNum...	Month	Day	WeekDay	Qu...	Holiday	WorkingDays	WorkingDayNumber
1/1/2001	2001	1	01 - Janu...	1	Monday	Q1	TRUE	0	0
1/2/2001	2001	1	01 - Janu...	2	Tuesday	Q1	FALSE	1	1
1/3/2001	2001	1	01 - Janu...	3	Wednesday	Q1	FALSE	1	2
1/4/2001	2001	1	01 - Janu...	4	Thursday	Q1	FALSE	1	3
1/5/2001	2001	1	01 - Janu...	5	Friday	Q1	FALSE	1	4
1/6/2001	2001	1	01 - Janu...	6	Saturday	Q1	FALSE	0	4
1/7/2001	2001	1	01 - Janu...	7	Sunday	Q1	FALSE	0	4
1/8/2001	2001	1	01 - Janu...	8	Monday	Q1	FALSE	1	5
1/9/2001	2001	1	01 - Janu...	9	Tuesday	Q1	FALSE	1	6
1/10/2001	2001	1	01 - Janu...	10	Wednesday	Q1	FALSE	1	7

FIGURE 7-33 The WorkingDayNumber calculation.

At this point, you can define the number of working days between two dates using the difference of WorkingDayNumber for the correspondent dates. Because you might not want to add too many tables to the PowerPivot model, you may reuse the same OrderDate table already imported into the model to get the WorkingDayNumber for both DueDate and ShipDate dates of an order. For example, this number for ShipDate can be obtained by using the following DAX expression:

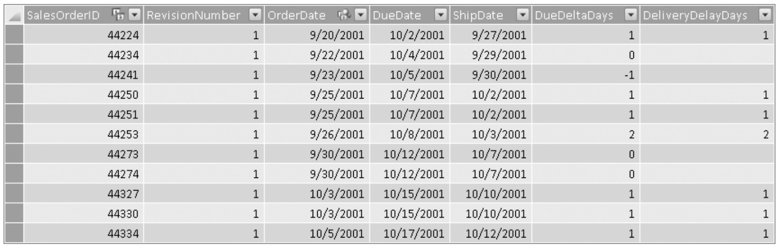
```
CALCULATE( VALUES( OrderDate[WorkingDayNumber] ),
           FILTER( OrderDate,
                   OrderDate[Date] = SalesOrderHeader[ShipDate] ) )
```

The FILTER call filters only the ShipDate row in the OrderDate table. Using this filter, the CALCULATE function returns the value of WorkingDayNumber for that row. The use of VALUES grants that an error message is raised if the FILTER returns more than one row (in which case, the filter condition contains an error).

So using this DAX expression for both ShipDate and DueDate, we can define a DueDeltaDays calculated column in SalesOrderHeader by using the following formula:

```
DueDeltaDays = CALCULATE( VALUES( OrderDate[WorkingDayNumber] ),
                          FILTER( OrderDate,
                                  OrderDate[Date] = SalesOrderHeader[ShipDate])) + 4
               - CALCULATE( VALUES( OrderDate[WorkingDayNumber] ),
                            FILTER( OrderDate,
                                    OrderDate[Date] = SalesOrderHeader[DueDate]))
```

The DueDeltaDays column shows a positive number in the case of a delay, representing the number of delay days. Negative numbers indicate an early delivery (measured always in days). In Figure 7-34, you can see values for this column and for another calculated column named DeliveryDelayDays, which displays a value only for delayed orders.



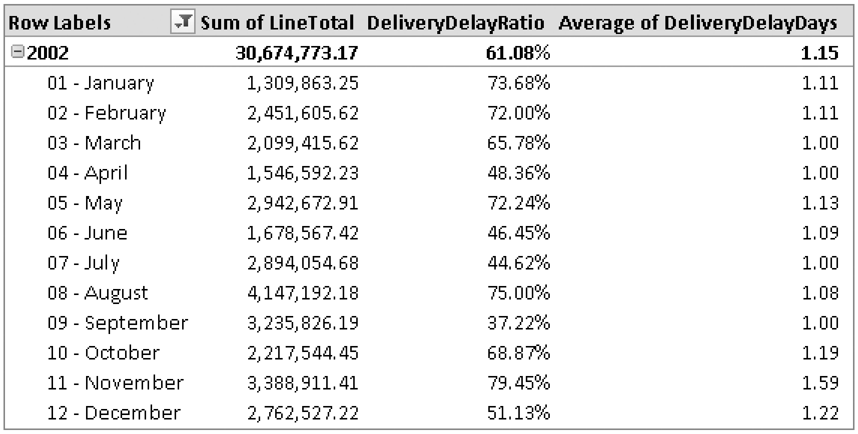
SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	DueDeltaDays	DeliveryDelayDays
44224	1	9/20/2001	10/2/2001	9/27/2001	1	1
44234	1	9/22/2001	10/4/2001	9/29/2001	0	
44241	1	9/23/2001	10/5/2001	9/30/2001	-1	
44250	1	9/25/2001	10/7/2001	10/2/2001	1	1
44251	1	9/25/2001	10/7/2001	10/2/2001	1	1
44253	1	9/26/2001	10/8/2001	10/3/2001	2	2
44273	1	9/30/2001	10/12/2001	10/7/2001	0	
44274	1	9/30/2001	10/12/2001	10/7/2001	0	
44327	1	10/3/2001	10/15/2001	10/10/2001	1	1
44330	1	10/3/2001	10/15/2001	10/10/2001	1	1
44334	1	10/5/2001	10/17/2001	10/12/2001	1	1

FIGURE 7-34 The DueDeltaDays and DeliveryDelayDays calculated columns in the SalesOrderHeader.

With this information, you can calculate some measures in the PivotTable, such as the ratio of delayed deliveries:

```
DeliveryDelayRation = COUNT( SalesOrderHeader[DeliveryDelayDays] )
                      / COUNTROWS( SalesOrderHeader )
```

You can also calculate the average delay (in days) for delayed orders, by simply selecting the Summarize By Average item on the DeliveryDelayDays column. In Figure 7-35, you can see a PivotTable displaying both these measures.



Row Labels	Sum of LineTotal	DeliveryDelayRatio	Average of DeliveryDelayDays
⊟2002	30,674,773.17	61.08%	1.15
01 - January	1,309,863.25	73.68%	1.11
02 - February	2,451,605.62	72.00%	1.11
03 - March	2,099,415.62	65.78%	1.00
04 - April	1,546,592.23	48.36%	1.00
05 - May	2,942,672.91	72.24%	1.13
06 - June	1,678,567.42	46.45%	1.09
07 - July	2,894,054.68	44.62%	1.00
08 - August	4,147,192.18	75.00%	1.08
09 - September	3,235,826.19	37.22%	1.00
10 - October	2,217,544.45	68.87%	1.19
11 - November	3,388,911.41	79.45%	1.59
12 - December	2,762,527.22	51.13%	1.22

FIGURE 7-35 The DeliveryDelayRatio and average of DeliveryDelayDays columns in PivotTable.

Calculating WorkingDays by Using Table Relationships

You might wonder why we have not used table relationships to relate SalesDate and DueDate tables with corresponding SalesDates and DueDates tables; why have we used a more complicated DAX expression instead? We have two reasons for that. First, we would have needed to duplicate these two tables both in Excel and in PowerPivot. Second, the need to duplicate would have been propagated to the Holidays table too, requiring three Holidays tables in Excel that would have been imported in three different linked tables in PowerPivot. You can look at an example of such a model in the CH07-07-DeliveryDays-UsingRelationships.xlsx workbook included on the companion DVD.

Because PowerPivot does not support more than one relationship between two tables, it follows in an indirect way that you cannot relate the same Holiday table from two or more different tables (such as SalesDates and DueDates) if these tables are both related to the same table (such as SalesOrderHeader), even if the relationship is through different columns.

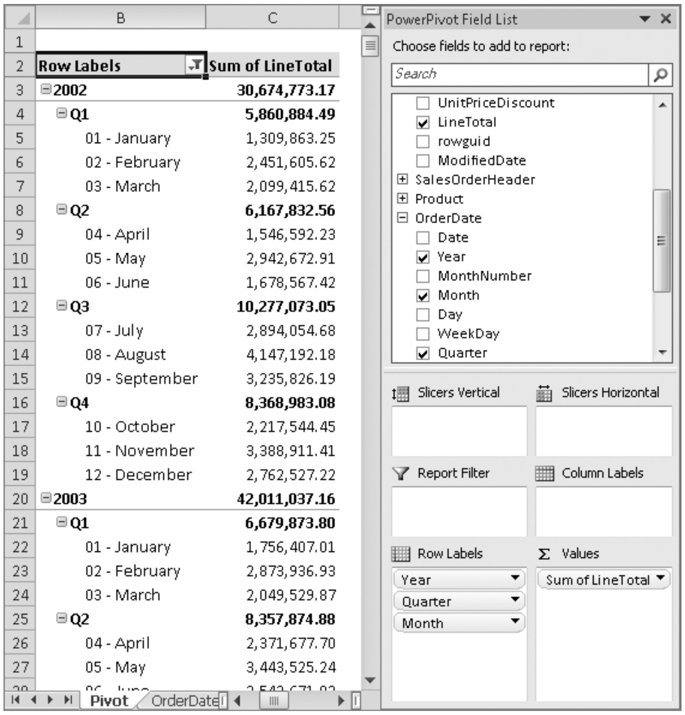
Aggregating and Comparing over Time

Working days calculation is only the first step in the benefits that you can obtain by using a calendar table. In the next sections, we introduce other useful techniques. It is often required that you analyze particular aggregations of values over time. For example, you might want to calculate the aggregated value of a measure from the beginning of the year up to the period you are selecting. (This is commonly called *year-to-date aggregation*.) You might want to look at the Sales Amount for the month of March but also want to look at the total Sales Amount from January to March. Having a Dates table is an important prerequisite for making this calculation in PivotTable.

Year-to-Date, Quarter-to-Date, and Month-to-Date

The calculation of year-to-date (YTD), quarter-to-date, (QTD) and month-to-date (MTD) are all very similar. Obviously, month-to-date is meaningful only when you are looking at data at the day level, whereas year-to-date and quarter-to-date calculations are often used to look at data at the month level.

For example, in Figure 7-36, you can see the LineTotal measure aggregated by year, quarter, and month.



Row Labels	Sum of LineTotal
2002	30,674,773.17
Q1	5,860,884.49
01 - January	1,309,863.25
02 - February	2,451,605.62
03 - March	2,099,415.62
Q2	6,167,832.56
04 - April	1,546,592.23
05 - May	2,942,672.91
06 - June	1,678,567.42
Q3	10,277,073.05
07 - July	2,894,054.68
08 - August	4,147,192.18
09 - September	3,235,826.19
Q4	8,368,983.08
10 - October	2,217,544.45
11 - November	3,388,911.41
12 - December	2,762,527.22
2003	42,011,037.16
Q1	6,679,873.80
01 - January	1,756,407.01
02 - February	2,873,936.93
03 - March	2,049,529.87
Q2	8,357,874.88
04 - April	2,371,677.70
05 - May	3,443,525.24

PowerPivot Field List

Choose fields to add to report:

- [ ] UnitPriceDiscount
- [✓] LineTotal
- [ ] rowguid
- [ ] ModifiedDate
- SalesOrderHeader
- Product
- OrderDate
  - [ ] Date
  - [✓] Year
  - [ ] MonthNumber
  - [✓] Month
  - [ ] Day
  - [ ] WeekDay
  - [✓] Quarter

Row Labels: Year, Quarter, Month

Values: Sum of LineTotal

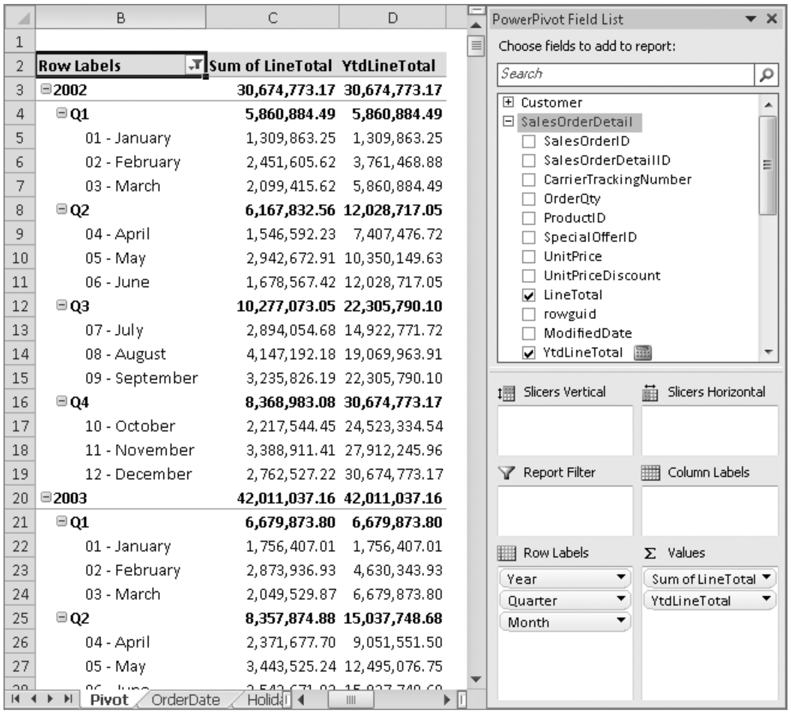
FIGURE 7-36 The LineTotal measure aggregated by the corresponding period in a row.

You can calculate the year-to-date value of LineTotal for each month and quarter by using a measure that operates on the filter context, modifying the filter context on dates for a range that starts on January 1 and ends on the month corresponding to the calculated cell. You can define a YtdLineTotal measure by using the following DAX formula:

```
YtdLineTotal = CALCULATE( SUM( SalesOrderDetail[LineTotal] ),
                          DATESYTD( OrderDate[Date] ) )
```

The CALCULATE function receives in its second parameter a table that contains the dates of the year-to-date period that has to be considered in the aggregation. This set of dates is returned by the built-in DATESYTD function, which is a Time Intelligence function that returns a list of all the dates from the beginning of the year until the last date included in the current filter context.

You can see the new measure in action in Figure 7-37.



	B	C	D
1			
2	Row Labels	Sum of LineTotal	YtdLineTotal
3	⊟2002	30,674,773.17	30,674,773.17
4	⊟Q1	5,860,884.49	5,860,884.49
5	01 - January	1,309,863.25	1,309,863.25
6	02 - February	2,451,605.62	3,761,468.88
7	03 - March	2,099,415.62	5,860,884.49
8	⊟Q2	6,167,832.56	12,028,717.05
9	04 - April	1,546,592.23	7,407,476.72
10	05 - May	2,942,672.91	10,350,149.63
11	06 - June	1,678,567.42	12,028,717.05
12	⊟Q3	10,277,073.05	22,305,790.10
13	07 - July	2,894,054.68	14,922,771.72
14	08 - August	4,147,192.18	19,069,963.91
15	09 - September	3,235,826.19	22,305,790.10
16	⊟Q4	8,368,983.08	30,674,773.17
17	10 - October	2,217,544.45	24,523,334.54
18	11 - November	3,388,911.41	27,912,245.96
19	12 - December	2,762,527.22	30,674,773.17
20	⊟2003	42,011,037.16	42,011,037.16
21	⊟Q1	6,679,873.80	6,679,873.80
22	01 - January	1,756,407.01	1,756,407.01
23	02 - February	2,873,936.93	4,630,343.93
24	03 - March	2,049,529.87	6,679,873.80
25	⊟Q2	8,357,874.88	15,037,748.68
26	04 - April	2,371,677.70	9,051,551.50
27	05 - May	3,443,525.24	12,495,076.75

FIGURE 7-37 The LineTotal year-to-date measure side-by-side with a regular measure.

This approach requires that you deal with the CALCULATE function, but because this pattern (using a CALCULATE and a DATESYTD function) is very common, PowerPivot offers a dedicated DAX function that simplifies (and makes more readable) the syntax of the YTD calculation, TOTALYTD:

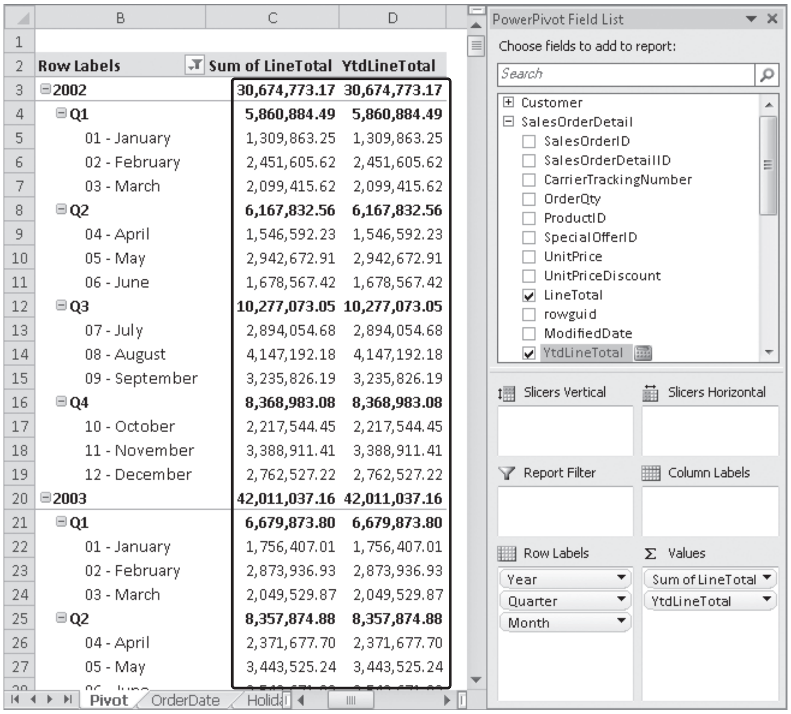
```
YtdLineTotal = TOTALYTD( SUM( SalesOrderDetail[LineTotal] ),
                         OrderDate[Date] )
```

As you can see, the syntax requires the hoped-for aggregation as the first parameter and then just the date column as the second parameter. The behavior is identical to the original measure, but the name of the TOTALYTD function immediately communicates the intention of the formula. However, you need to know the behavior of the original CALCULATE syntax because it allows a more complex calculation that you define later in this chapter.

What Date Column to Use

Keep in mind that the date column that you must use when calling TOTALYTD (and other similar functions) is the date column of the Dates table, and not the date column of the table that is the object of analysis. In this case, the OrderDate[Date] column was used instead of the SalesOrderHeader[OrderDate] column. If we had used the latter, the calculation would have been wrong. You can see in Figure 7-38 the result that would have been produced by using the following formula for the YTD measure:

```
YtdLineTotal = TOTALYTD( SUM( SalesOrderDetail[LineTotal] ),
                         SalesOrderHeader[OrderDate] )
```



	B	C	D
1			
2	Row Labels	Sum of LineTotal	YtdLineTotal
3	**2002**	**30,674,773.17**	**30,674,773.17**
4	**Q1**	**5,860,884.49**	**5,860,884.49**
5	01 - January	1,309,863.25	1,309,863.25
6	02 - February	2,451,605.62	2,451,605.62
7	03 - March	2,099,415.62	2,099,415.62
8	**Q2**	**6,167,832.56**	**6,167,832.56**
9	04 - April	1,546,592.23	1,546,592.23
10	05 - May	2,942,672.91	2,942,672.91
11	06 - June	1,678,567.42	1,678,567.42
12	**Q3**	**10,277,073.05**	**10,277,073.05**
13	07 - July	2,894,054.68	2,894,054.68
14	08 - August	4,147,192.18	4,147,192.18
15	09 - September	3,235,826.19	3,235,826.19
16	**Q4**	**8,368,983.08**	**8,368,983.08**
17	10 - October	2,217,544.45	2,217,544.45
18	11 - November	3,388,911.41	3,388,911.41
19	12 - December	2,762,527.22	2,762,527.22
20	**2003**	**42,011,037.16**	**42,011,037.16**
21	**Q1**	**6,679,873.80**	**6,679,873.80**
22	01 - January	1,756,407.01	1,756,407.01
23	02 - February	2,873,936.93	2,873,936.93
24	03 - March	2,049,529.87	2,049,529.87
25	**Q2**	**8,357,874.88**	**8,357,874.88**
26	04 - April	2,371,677.70	2,371,677.70
27	05 - May	3,443,525.24	3,443,525.24

PowerPivot Field List

Choose fields to add to report:

Search

- Customer
- SalesOrderDetail
  - SalesOrderID
  - SalesOrderDetailID
  - CarrierTrackingNumber
  - OrderQty
  - ProductID
  - SpecialOfferID
  - UnitPrice
  - UnitPriceDiscount
  - ✓ LineTotal
  - rowguid
  - ModifiedDate
  - ✓ YtdLineTotal

Slicers Vertical | Slicers Horizontal | Report Filter | Column Labels

Row Labels: Year, Quarter, Month

Values: Sum of LineTotal, YtdLineTotal

FIGURE 7-38 The wrong year-to-date calculation returns the same value as the LineTotal measure.

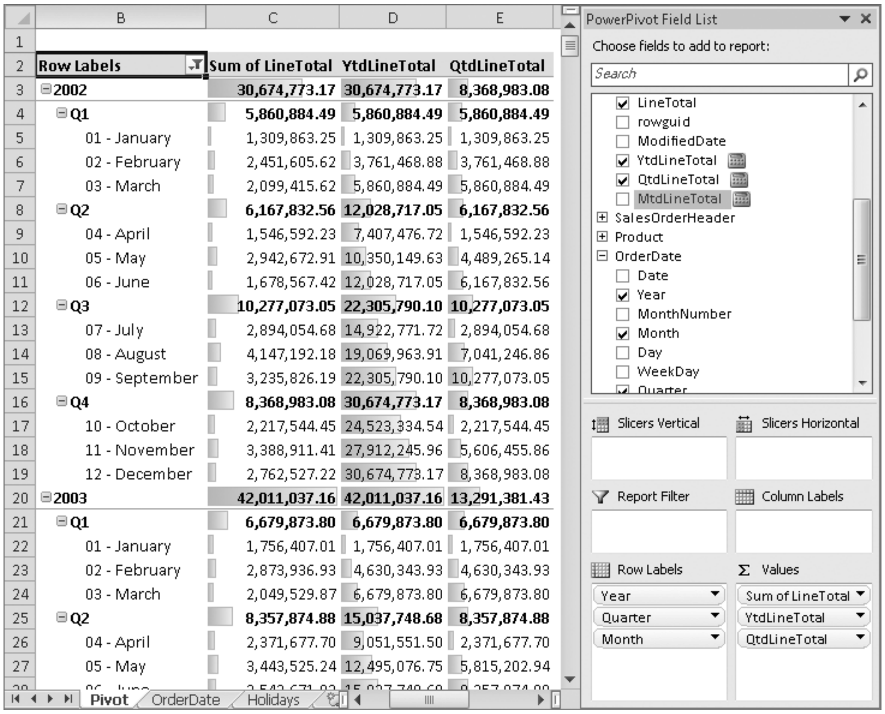
The problem is that the existing filter on year and month would be still applied. There are possible workarounds that you might use, but our suggestion is to always define and use a Dates table. Further details on the issue and possible workarounds are available in this blog post written by Kasper de Jonge: *http://tinyurl.com/DaxTimeAll*.

As you can for the year-to-date calculation, you can also define quarter-to-date and month-to-date calculations with built-in functions, as in these measures:

```
QtdLineTotal = TOTALQTD( SUM( SalesOrderDetail[LineTotal] ),
                         OrderDate[Date] )

MtdLineTotal = TOTALMTD( SUM( SalesOrderDetail[LineTotal] ),
                         OrderDate[Date] )
```

In Figure 7-39, you can see the year-to-date and quarter-to-date measures used in a PivotTable. Note that the quarter-to-date measure makes the year total equal to the last quarter of the year.



Row Labels	Sum of LineTotal	YtdLineTotal	QtdLineTotal
2002	**30,674,773.17**	**30,674,773.17**	**8,368,983.08**
Q1	**5,860,884.49**	**5,860,884.49**	**5,860,884.49**
01 - January	1,309,863.25	1,309,863.25	1,309,863.25
02 - February	2,451,605.62	3,761,468.88	3,761,468.88
03 - March	2,099,415.62	5,860,884.49	5,860,884.49
Q2	**6,167,832.56**	**12,028,717.05**	**6,167,832.56**
04 - April	1,546,592.23	7,407,476.72	1,546,592.23
05 - May	2,942,672.91	10,350,149.63	4,489,265.14
06 - June	1,678,567.42	12,028,717.05	6,167,832.56
Q3	**10,277,073.05**	**22,305,790.10**	**10,277,073.05**
07 - July	2,894,054.68	14,922,771.72	2,894,054.68
08 - August	4,147,192.18	19,069,963.91	7,041,246.86
09 - September	3,235,826.19	22,305,790.10	10,277,073.05
Q4	**8,368,983.08**	**30,674,773.17**	**8,368,983.08**
10 - October	2,217,544.45	24,523,334.54	2,217,544.45
11 - November	3,388,911.41	27,912,245.96	5,606,455.86
12 - December	2,762,527.22	30,674,773.17	8,368,983.08
2003	**42,011,037.16**	**42,011,037.16**	**13,291,381.43**
Q1	**6,679,873.80**	**6,679,873.80**	**6,679,873.80**
01 - January	1,756,407.01	1,756,407.01	1,756,407.01
02 - February	2,873,936.93	4,630,343.93	4,630,343.93
03 - March	2,049,529.87	6,679,873.80	6,679,873.80
Q2	**8,357,874.88**	**15,037,748.68**	**8,357,874.88**
04 - April	2,371,677.70	9,051,551.50	2,371,677.70
05 - May	3,443,525.24	12,495,076.75	5,815,202.94

PowerPivot Field List

Choose fields to add to report:

- ✓ LineTotal
- rowguid
- ModifiedDate
- ✓ YtdLineTotal
- ✓ QtdLineTotal
- MtdLineTotal
- SalesOrderHeader
- Product
- OrderDate
  - Date
  - ✓ Year
  - MonthNumber
  - ✓ Month
  - Day
  - WeekDay

Row Labels: Year, Quarter, Month

Values: Sum of LineTotal, YtdLineTotal, QtdLineTotal

FIGURE 7-39 The year-to-date and quarter-to-date measures side-by-side with a regular measure.

To calculate a year-to-date measure over the fiscal year, you need to use an optional parameter that specifies the end day of the fiscal year. For example, you can calculate the fiscal year-to-date for LineTotal by using the following expression:

```
FiscalYtdLineTotal = TOTALYTD( SUM( SalesOrderDetail[LineTotal] ),
                               OrderDate[Date],
                               “06-30” )
```

The last parameter corresponds to June 30, which in our OrderDate table corresponds to the end of the fiscal year. You can find several Time Intelligence functions that have a last, optional YE_Date parameter for this purpose: STARTOFYEAR, ENDOFYEAR, PREVIOUSYEAR, NEXTYEAR, DATESYTD, TOTALYTD, OPENINGBALANCEYEAR, and CLOSINGBALANCEYEAR.

Periods from the Prior Year

People commonly need to get a value from a period of the prior year (PY). This can be useful for making comparisons of trends, during a period last year to the same period this year, as you can see in the CH07-08-Aggregation.xlsx workbook included on the companion DVD. This is the DAX expression you need to calculate that value:

```
PyLineTotal = CALCULATE( SUM( SalesOrderDetail[LineTotal] ),
                         SAMEPERIODLASTYEAR( OrderDate[Date] ) )
```

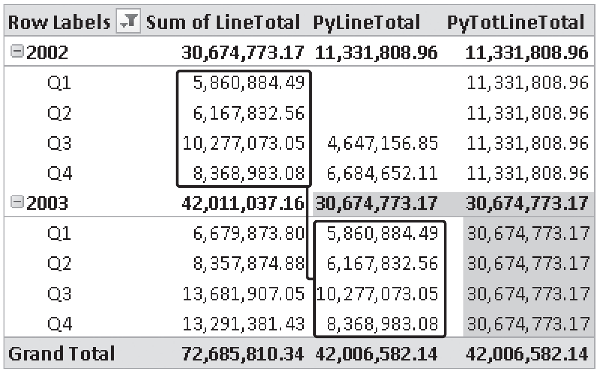
The CALCULATE function changes the filter by using the SAMEPERIODLASTYEAR function, which returns a set of dates shifted one year back in time. The SAMEPERIODLASTYEAR function is a specialized version of the more generic DATEADD function, which can be used by specifying the number and type of periods to shift. For example, the same PyLineTotal measure can be defined by this equivalent expression:

```
PyLineTotal = CALCULATE( SUM( SalesOrderDetail[LineTotal] ),
                         DATEADD( OrderDate[Date], -1, YEAR ) )
```

Sometimes you must look at the total amount of a measure for the previous year, usually to compare it with the year-to-date total. To do that, you can use the PARALLELPERIOD function, which is similar to DATEADD but returns the full period specified in the third parameter instead of the partial period returned by DATEADD. The PyTotLineTotal measure that calculates the total sum of LineTotal for the previous year can be defined this way:

```
PyTotLineTotal = CALCULATE( SUM( SalesOrderDetail[LineTotal] ),
                            PARALLELPERIOD( OrderDate[Date], -1, YEAR ) )
```

In Figure 7-40, you can see the result of the PyLineTotal and PyTotLineTotal measures. The quarters data in 2002 for the Sum Of LineTotal column has been copied into the respective quarters of year 2003 in the PyLineTotal column. The PyTotLineTotal simply reports for every period the total amount of the LineTotal column for the year before.



Row Labels	Sum of LineTotal	PyLineTotal	PyTotLineTotal
⊟2002	**30,674,773.17**	**11,331,808.96**	**11,331,808.96**
Q1	5,860,884.49		11,331,808.96
Q2	6,167,832.56		11,331,808.96
Q3	10,277,073.05	4,647,156.85	11,331,808.96
Q4	8,368,983.08	6,684,652.11	11,331,808.96
⊟2003	**42,011,037.16**	**30,674,773.17**	**30,674,773.17**
Q1	6,679,873.80	5,860,884.49	30,674,773.17
Q2	8,357,874.88	6,167,832.56	30,674,773.17
Q3	13,681,907.05	10,277,073.05	30,674,773.17
Q4	13,291,381.43	8,368,983.08	30,674,773.17
Grand Total	**72,685,810.34**	**42,006,582.14**	**42,006,582.14**

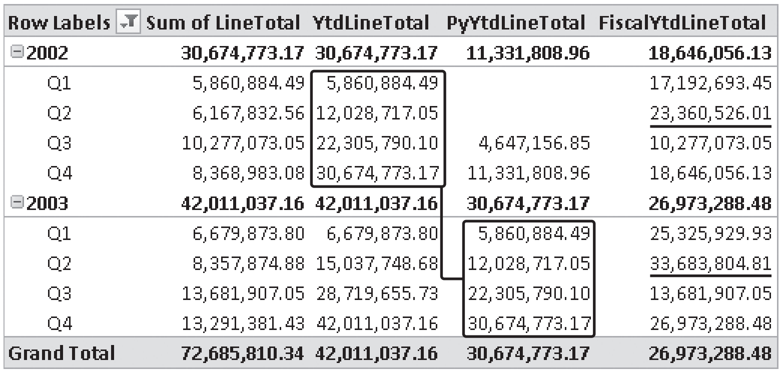
FIGURE 7-40 Prior Year simple calculations.

When you want to calculate the year-to-date of the prior year because, typically, you want to compare it with the current year-to-date measure, you have to mix the two techniques. Instead of passing the OrderDate[Date] parameter to SAMEPERIODLASTYEAR, which corresponds to the list of dates that are active in the current filter context, you can use the DATESYTD function to make a transformation of these dates, defining the year-to-date group first. However, you might also invert the order of these calls. In fact, the two following definitions of PyYtdLineTotal are equivalent:

```
PyYtdLineTotal = CALCULATE( SUM( SalesOrderDetail[LineTotal] ),
                            SAMEPERIODLASTYEAR( DATESYTD( OrderDate[Date] ) ) )

PyYtdLineTotal = CALCULATE( SUM( SalesOrderDetail[LineTotal] ),
                            DATESYTD( SAMEPERIODLASTYEAR( OrderDate[Date] ) ) )
```

You can see the results of the PyYtdLineTotal in Figure 7-41. The values of YtdLineTotal are reported for PyYtdLineTotal shifted by one year. In the same screen, you can also see the FiscalYtdLineTotal measure that you saw at the end of the previous section: the horizontal lines between Q2 and Q3 in that column highlight the points at which the year-to-date calculation restarts.



Row Labels	Sum of LineTotal	YtdLineTotal	PyYtdLineTotal	FiscalYtdLineTotal
⊟2002	**30,674,773.17**	**30,674,773.17**	**11,331,808.96**	**18,646,056.13**
Q1	5,860,884.49	5,860,884.49		17,192,693.45
Q2	6,167,832.56	12,028,717.05		23,360,526.01
Q3	10,277,073.05	22,305,790.10	4,647,156.85	10,277,073.05
Q4	8,368,983.08	30,674,773.17	11,331,808.96	18,646,056.13
⊟2003	**42,011,037.16**	**42,011,037.16**	**30,674,773.17**	**26,973,288.48**
Q1	6,679,873.80	6,679,873.80	5,860,884.49	25,325,929.93
Q2	8,357,874.88	15,037,748.68	12,028,717.05	33,683,804.81
Q3	13,681,907.05	28,719,655.73	22,305,790.10	13,681,907.05
Q4	13,291,381.43	42,011,037.16	30,674,773.17	26,973,288.48
Grand Total	**72,685,810.34**	**42,011,037.16**	**30,674,773.17**	**26,973,288.48**

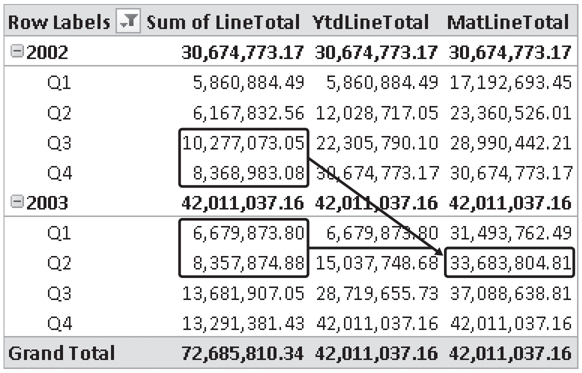
FIGURE 7-41 The year-to-date calculation for Prior Year and Fiscal Year.

Another commonly requested calculation that eliminates seasonal changes in sales is the moving annual total (MAT), which always considers the last 12 months. For example, the value of MatLineTotal for March 2002 is calculated by summing the range of dates from April 2001 to March 2002. Consider the following MatLineTotal measure definition, which calculates the moving annual total for LineTotal:

```
MatLineTotal = CALCULATE( SUM( SalesOrderDetail[LineTotal] ),
                          DATESBETWEEN(
                              OrderDate[Date],
                                  NEXTDAY(
                                      SAMEPERIODLASTYEAR(
                                          LASTDATE( OrderDate[Date] ) ) ),
                                  LASTDATE( OrderDate[Date] ) ) )
```

The implementation of this measure requires some attention. You need to use the DATESBETWEEN function, which returns the dates from a column included between two specified dates. Because this calculation is always made at the day level, even if the PivotTable is browsing data at the month level, you must calculate the first day and the last day of the interval you want. The last day can be obtained by calling the LASTDATE function, which returns the last date of a given column (always considering the current filter context). Starting from this date, you can get the first day of the interval by requesting the following day (by calling NEXTDAY) of the corresponding last date one year before. (You can do this by using SAMEPERIODLASTYEAR, as we did before.)

In Figure 7-42, you can see a PivotTable using the moving annual total calculation. For example, the 2003 Q2 data has been calculated by summing Q3 and Q4 of 2002, plus Q1 and Q2 of 2003. In the middle, you see the classic year-to-date calculation, which has the same value of moving annual total only for the last period of each year (in this case Q4).



Row Labels	Sum of LineTotal	YtdLineTotal	MatLineTotal
2002	**30,674,773.17**	**30,674,773.17**	**30,674,773.17**
Q1	5,860,884.49	5,860,884.49	17,192,693.45
Q2	6,167,832.56	12,028,717.05	23,360,526.01
Q3	10,277,073.05	22,305,790.10	28,990,442.21
Q4	8,368,983.08	30,674,773.17	30,674,773.17
2003	**42,011,037.16**	**42,011,037.16**	**42,011,037.16**
Q1	6,679,873.80	6,679,873.80	31,493,762.49
Q2	8,357,874.88	15,037,748.68	33,683,804.81
Q3	13,681,907.05	28,719,655.73	37,088,638.81
Q4	13,291,381.43	42,011,037.16	42,011,037.16
Grand Total	**72,685,810.34**	**42,011,037.16**	**42,011,037.16**

FIGURE 7-42 The moving Annual Total vs. year-to-date calculation.

Other Aggregation Functions and the CALCULATE Syntax

In all the examples, we have used the SUM aggregation function. You might need to use other aggregation functions, such as AVERAGE, or more complex formulas. Whenever you saw SUM(SalesOrderDetail[LineTotal]) in the previous example, consider that you can always replace such expressions with another DAX formula, also by simply replacing the aggregation function.

In case your calculation becomes more complex, you might prefer to specify that calculation, which can be shared across several other measures, into a separated measure, containing just this operation. In this way, you can avoid the duplication of the aggregation function in all the formulas for measures that make special calculations for dates.

For example, you might want to calculate the weighted average price with the following formula, assigned to measure AveragePrice:

```
AveragePrice = SUM( SalesOrderDetail[LineTotal] ) / SUM( SalesOrderDetail[OrderQty] )
```

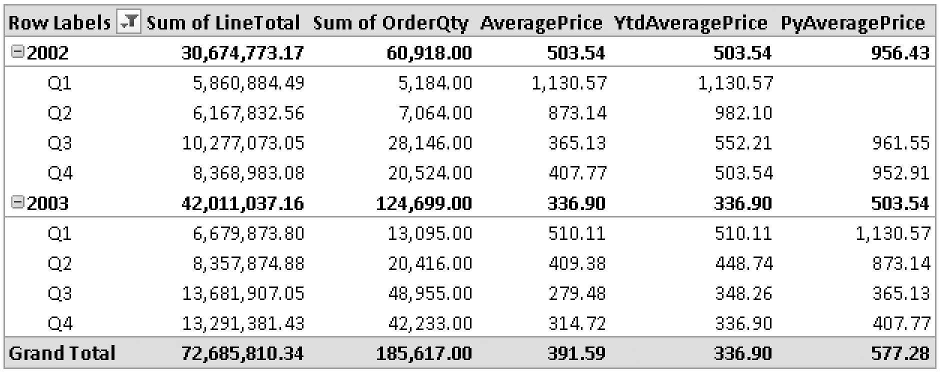
You can use a direct reference to that measure, without using an aggregation function, whenever you use formulas such as CALCULATE or special Time Intelligence functions that behave like CALCULATE, as in the year-to-date calculation defined in YtdAveragePrice measure:

```
YtdAveragePrice = TOTALYTD( SalesOrderDetail[AveragePrice], OrderDate[Date] )
```

The calculation for the prior year can be written by using the CALCULATE function:

```
PyAveragePrice = CALCULATE( SalesOrderDetail[AveragePrice],
                            SAMEPERIODLASTYEAR( OrderDate[Date] ) )
```

In Figure 7-43, you can see the results of these formulas in a PivotTable.



Row Labels	Sum of LineTotal	Sum of OrderQty	AveragePrice	YtdAveragePrice	PyAveragePrice
⊟2002	**30,674,773.17**	**60,918.00**	**503.54**	**503.54**	**956.43**
Q1	5,860,884.49	5,184.00	1,130.57	1,130.57	
Q2	6,167,832.56	7,064.00	873.14	982.10	
Q3	10,277,073.05	28,146.00	365.13	552.21	961.55
Q4	8,368,983.08	20,524.00	407.77	503.54	952.91
⊟2003	**42,011,037.16**	**124,699.00**	**336.90**	**336.90**	**503.54**
Q1	6,679,873.80	13,095.00	510.11	510.11	1,130.57
Q2	8,357,874.88	20,416.00	409.38	448.74	873.14
Q3	13,681,907.05	48,955.00	279.48	348.26	365.13
Q4	13,291,381.43	42,233.00	314.72	336.90	407.77
Grand Total	**72,685,810.34**	**185,617.00**	**391.59**	**336.90**	**577.28**

FIGURE 7-43 The year-to-date and prior year calculations for Average Price.

In case you want to make a monthly average of the total sales, you should use the number of months in the denominator of the ratio, as in the following expression that you can use to define the MonthlyAverage measure:

```
MonthlyAverage = IF( COUNTROWS( VALUES( OrderDate[Month] ) ) > 0,
                     SUM( SalesOrderDetail[LineTotal] )
                         / COUNTROWS( VALUES( OrderDate[Month] ) ),
                     BLANK() )
```

Please note that this definition does not work with a selection of more than one year. To avoid this issue, you need to create a calculated column in the OrderDate table with a concatenation of year and month so that the distinct number of values over a period take account also of the year and not just the month. If you do that, you must replace that column to the Month column used in the formula above.

To make a monthly average of the year-to-date sales, you have to replace the corresponding YtdLineTotal measure at the numerator and you need a CALCULATE expression at the denominator so that you can calculate the number of months included in the year-to-date calculation:

```
YtdMonthlyAverage = IF( COUNTROWS( VALUES( OrderDate[Month] ) ) > 0,
                           SalesOrderDetail[YtdLineTotal]
                               / CALCULATE( COUNTROWS( VALUES( OrderDate[Month] ) ),
                                            DATESYTD( OrderDate[Date] ) ),
                        BLANK() )
```

As you can see, the expression might be different according to the calculation you have to do. You have to pay particular attention to the calculation necessary for numerators and denominators of any ratio and average measure.

Difference over Previous Year

A common operation that compares a measure with its value in the prior year is to calculate the difference of these values. That difference might be expressed as an absolute value or by using a percentage, as you can see in the CH07-09-Yoy.xlsx workbook included on the companion DVD. To make these calculations, you need the value for the prior year that you already defined in PyLineTotal:

```
PyLineTotal = CALCULATE( SUM( SalesOrderDetail[LineTotal] ),
                         SAMEPERIODLASTYEAR( OrderDate[Date] ) )
```

The absolute difference of LineTotal over previous year (year-over-year, YOY) is a simple subtraction. You can define a YoyLineTotal measure with the following expression:

```
YoyLineTotal = SUM( SalesOrderDetail[LineTotal] ) - SalesOrderDetail[PyLineTotal]
```

You calculate the value of the selected year by using the SUM aggregation; the measure corresponding to the value of the prior year does not need to be summed because the aggregation is already done as part of the underlying measure expression.

The analogous calculation for comparing the year-to-date measure with a corresponding value in the prior year is a simple subtraction of two measures, YtdLineTotal and PyYtdLineTotal, which you saw in the previous section; we report it here just as reminder:

```
PyYtdLineTotal = CALCULATE( SUM( SalesOrderDetail[LineTotal] ),
                            SAMEPERIODLASTYEAR( DATESYTD( OrderDate[Date] ) ) )

YoyYtdLineTotal = SalesOrderDetail[YtdLineTotal] - SalesOrderDetail[PyYtdLineTotal]
```

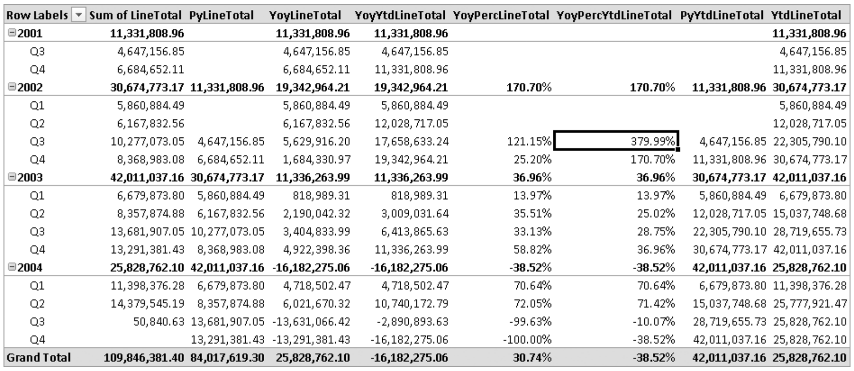
Most of the time, the year-over-year difference is better expressed as a percentage in a report. You can define this calculation by dividing YoyLineTotal by the PyLineTotal; in this way, the difference uses the prior read value as a reference for the percentage difference (100 percent corresponds to a value that is doubled in one year). In the following expression that defines the YoyPercLineTotal measure, the IF statement avoids a divide-by-zero error in case there is no corresponding data in the prior year:

```
YoyPercLineTotal = IF( SalesOrderDetail[PyLineTotal] = 0,
                       BLANK(),
                       SalesOrderDetail[YoyLineTotal] / SalesOrderDetail[PyLineTotal] )
```

A similar calculation can be made to display the percentage difference of a year-over-year comparison for the year-to-date aggregation. You can define YoyPercYtdLineTotal by using the following formula:

```
YoyPercYtdLineTotal = IF( SalesOrderDetail[PyYtdLineTotal] = 0,
                          BLANK(),
                          SalesOrderDetail[YoYYtdLineTotal]
                              / SalesOrderDetail[PyYtdLineTotal] )
```

In Figure 7-44, you can see the results of these measures in a PivotTable.



Row Labels	Sum of LineTotal	PyLineTotal	YoyLineTotal	YoyYtdLineTotal	YoyPercLineTotal	YoyPercYtdLineTotal	PyYtdLineTotal	YtdLineTotal
⊟2001	**11,331,808.96**		**11,331,808.96**	**11,331,808.96**				**11,331,808.96**
Q3	4,647,156.85		4,647,156.85	4,647,156.85				4,647,156.85
Q4	6,684,652.11		6,684,652.11	11,331,808.96				11,331,808.96
⊟2002	**30,674,773.17**	**11,331,808.96**	**19,342,964.21**	**19,342,964.21**	**170.70%**	**170.70%**	**11,331,808.96**	**30,674,773.17**
Q1	5,860,884.49		5,860,884.49	5,860,884.49				5,860,884.49
Q2	6,167,832.56		6,167,832.56	12,028,717.05				12,028,717.05
Q3	10,277,073.05	4,647,156.85	5,629,916.20	17,658,633.24	121.15%	379.99%	4,647,156.85	22,305,790.10
Q4	8,368,983.08	6,684,652.11	1,684,330.97	19,342,964.21	25.20%	170.70%	11,331,808.96	30,674,773.17
⊟2003	**42,011,037.16**	**30,674,773.17**	**11,336,263.99**	**11,336,263.99**	**36.96%**	**36.96%**	**30,674,773.17**	**42,011,037.16**
Q1	6,679,873.80	5,860,884.49	818,989.31	818,989.31	13.97%	13.97%	5,860,884.49	6,679,873.80
Q2	8,357,874.88	6,167,832.56	2,190,042.32	3,009,031.64	35.51%	25.02%	12,028,717.05	15,037,748.68
Q3	13,681,907.05	10,277,073.05	3,404,833.99	6,413,865.63	33.13%	28.75%	22,305,790.10	28,719,655.73
Q4	13,291,381.43	8,368,983.08	4,922,398.36	11,336,263.99	58.82%	36.96%	30,674,773.17	42,011,037.16
⊟2004	**25,828,762.10**	**42,011,037.16**	**-16,182,275.06**	**-16,182,275.06**	**-38.52%**	**-38.52%**	**42,011,037.16**	**25,828,762.10**
Q1	11,398,376.28	6,679,873.80	4,718,502.47	4,718,502.47	70.64%	70.64%	6,679,873.80	11,398,376.28
Q2	14,379,545.19	8,357,874.88	6,021,670.32	10,740,172.79	72.05%	71.42%	15,037,748.68	25,777,921.47
Q3	50,840.63	13,681,907.05	-13,631,066.42	-2,890,893.63	-99.63%	-10.07%	28,719,655.73	25,828,762.10
Q4		13,291,381.43	-13,291,381.43	-16,182,275.06	-100.00%	-38.52%	42,011,037.16	25,828,762.10
Grand Total	**109,846,381.40**	**84,017,619.30**	**25,828,762.10**	**-16,182,275.06**	**30.74%**	**-38.52%**	**42,011,037.16**	**25,828,762.10**

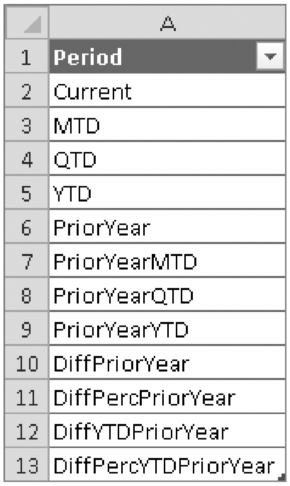
FIGURE 7-44 The year-over-year (YOY) measures used in a PivotTable.

Simplifying Browsing with a Period Table

In this chapter, you have seen how to create single measures with special calculations over time, such as year-to-date, year-over-year, and so on. One drawback of this approach is that you have to define one measure for each of these calculations, and the list of the measures in your model might grow too long.

A possible solution to this issue, which is also an interesting generic modeling solution, is to create a special table containing one line for each of the calculations you might want to apply to a measure. In this way, the end user has a shorter list of measures and possible operations on them, instead of having the Cartesian product of these two sets. However, you can also see that this solution has its own drawbacks, and maybe it is better to create just the measures you really want to use in your model, trying to expose only the combinations of measures and calculations that are meaningful for the expected analysis of your data.

First of all, you create a Period table in Excel, which contains the list of possible calculations that should be applied to a measure, as you can see in Figure 7-45. The complete model used for this example is available in the CH07-10-PeriodTable.xlsx workbook included on the companion DVD.



	A
1	Period
2	Current
3	MTD
4	QTD
5	YTD
6	PriorYear
7	PriorYearMTD
8	PriorYearQTD
9	PriorYearYTD
10	DiffPriorYear
11	DiffPercPriorYear
12	DiffYTDPriorYear
13	DiffPercYTDPriorYear

FIGURE 7-45 A Period table in Excel.

The same table has to be imported as a linked table into PowerPivot. However, you do not have to define any relationships between this table and other tables in your model because you use the selected member of the Period table to change the behavior of a measure through its DAX definition. Nevertheless, PowerPivot warns you of a missing relationship when you browse data in a PivotTable, as you can see in Figure 7-46.

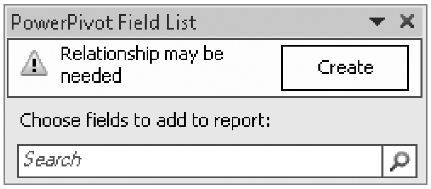


FIGURE 7-46 A warning about a missing relationship caused by the Period table.

This warning is provoked by the Period table, which does not have any relationships with other tables in the model. You can disable this warning by pressing the Detection button on the PowerPivot tab of the ribbon, which is the one highlighted in Figure 7-47.

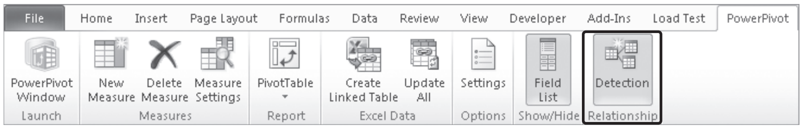


FIGURE 7-47 The Detection button on the PowerPivot tab of the ribbon disables the detection of missing relationships.

At this point, you can define a single measure that checks the selected value of the Period table and uses a DAX expression to return the corresponding calculation. Because there are no relationships with the Period table, the selected value in the Period table is always the one chosen by the user whenever that table is used as a filter, or the selected value is the corresponding value in a row or a column whenever Period is used in Row or Column labels. In general, we follow this generic pattern:

```
= IF( COUNTROWS( VALUES( Period[Period] ) ) = 1,
      IF( VALUES( Period[Period] ) = "Current", <expression>,
      IF( VALUES( Period[Period] ) = "MTD", <expression>,
      …
```

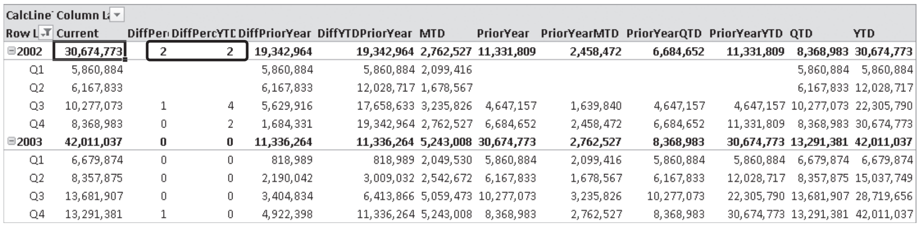
The first condition checks that there are not multiple values active in the filter context. In such a case, you should avoid any calculation because of the ambiguity of having multiple active values; otherwise, you should generate an error in the calculation, instead of returning a wrong value without warning the user. Then in the next step, each value is checked by a different IF statement, which evaluates the correct expression corresponding to the Period value. Assuming you have all the measures previously defined in this chapter, you need to replace the expression tag with the corresponding specific measure. For example, you can define a generic CalcLineTotal measure, which is used to apply one or more of the operations described in the Period table to the LineTotal measure:

```
CalcLineTotal = IF( COUNTROWS( VALUES( Period[Period] ) ) = 1,
      IF( VALUES( Period[Period] ) = "Current", SUM( SalesOrderDetail[LineTotal] ),
      IF( VALUES( Period[Period] ) = "MTD", SalesOrderDetail[MtdLineTotal],
      IF( VALUES( Period[Period] ) = "QTD", SalesOrderDetail[QtdLineTotal],
      IF( VALUES( Period[Period] ) = "YTD", SalesOrderDetail[YtdLineTotal],
      IF( VALUES( Period[Period] ) = "PriorYear", SalesOrderDetail[PyLineTotal],
      IF( VALUES( Period[Period] ) = "PriorYearMTD", SalesOrderDetail[PyMtdLineTotal],
      IF( VALUES( Period[Period] ) = "PriorYearQTD", SalesOrderDetail[PyQtdLineTotal],
      IF( VALUES( Period[Period] ) = "PriorYearYTD", SalesOrderDetail[PyYtdLineTotal],
      IF( VALUES( Period[Period] ) = "DiffPriorYear", SalesOrderDetail[YoyLineTotal],
      IF( VALUES( Period[Period] ) = "DiffPercPriorYear",
                                                  SalesOrderDetail[YoyPercLineTotal],
      IF( VALUES( Period[Period] ) = "DiffYTDPriorYear", SalesOrderDetail[YoyYtdLineTotal],
      IF( VALUES( Period[Period] ) = "DiffPercYTDPriorYear",
                                                  SalesOrderDetail[YoyPercYtdLineTotal],
  BLANK() ) ) ) ) ) ) ) ) ) ) ) ) ),
      BLANK() )
```

You have to repeat this definition for each of the measures to which you want to apply the Period calculations. You might avoid defining all the internal measures by replacing each reference to a measure with its corresponding DAX definition. This would make the CalcLineTotal definition longer and hard to maintain, but it is a design choice you might follow.

Tip Remember that you cannot hide a measure from a PivotTable (you can hide only calculated columns). So if you do not want to expose internal calculations, you should expand all the measures included in the preceding CalcLineTotal expression.

At this point, you can browse data by using the Period values crossed with the CalcLineTotal measure. In Figure 7-48, only the CalcLineTotal measure has been selected; the Period values are in the columns, and a selection of years and quarters is in the rows.



CalcLine	Column La											
Row L	Current	DiffPer	DiffPercYTD	DiffPriorYear	DiffYTDPriorYear	MTD	PriorYear	PriorYearMTD	PriorYearQTD	PriorYearYTD	QTD	YTD
⊟2002	**30,674,773**	**2**	**2**	**19,342,964**	**19,342,964**	**2,762,527**	**11,331,809**	**2,458,472**	**6,684,652**	**11,331,809**	**8,368,983**	**30,674,773**
Q1	5,860,884			5,860,884	5,860,884	2,099,416					5,860,884	5,860,884
Q2	6,167,833			6,167,833	12,028,717	1,678,567					6,167,833	12,028,717
Q3	10,277,073	1	4	5,629,916	17,658,633	3,235,826	4,647,157	1,639,840	4,647,157	4,647,157	10,277,073	22,305,790
Q4	8,368,983	0	2	1,684,331	19,342,964	2,762,527	6,684,652	2,458,472	6,684,652	11,331,809	8,368,983	30,674,773
⊟2003	**42,011,037**	**0**	**0**	**11,336,264**	**11,336,264**	**5,243,008**	**30,674,773**	**2,762,527**	**8,368,983**	**30,674,773**	**13,291,381**	**42,011,037**
Q1	6,679,874	0	0	818,989	818,989	2,049,530	5,860,884	2,099,416	5,860,884	5,860,884	6,679,874	6,679,874
Q2	8,357,875	0	0	2,190,042	3,009,032	2,542,672	6,167,833	1,678,567	6,167,833	12,028,717	8,357,875	15,037,749
Q3	13,681,907	0	0	3,404,834	6,413,866	5,059,473	10,277,073	3,235,826	10,277,073	22,305,790	13,681,907	28,719,656
Q4	13,291,381	1	0	4,922,398	11,336,264	5,243,008	8,368,983	2,762,527	8,368,983	30,674,773	13,291,381	42,011,037

FIGURE 7-48 The Period calculations applied to the CalcLineTotal measure.

As we anticipated, this solution has several drawbacks.

- After you put Period in rows or columns, you cannot change the order of its members. Actually, you can do this by using some Excel features, but it is not as immediate and intuitive as moving the list of measures into the Values list in the PowerPivot Field List panel.
- The number format of the measure cannot change for particular calculations requested through some Period values. For example, in Figure 7-48, you can see that the DiffPercPriorYear and DiffPercYTDPriorYear calculations do not display the CalcLineTotal value as a percentage because you can define a single number format for a measure in a PivotTable. A possible workaround is to change the number format directly in Excel cells, but this change is lost as soon as you navigate into the PivotTable.
- If you use more than one measure in the PivotTable, you must create a set based on column items in Excel, choosing only the combination of measures and Period values that you really want to see in the PivotTable. You can see an example of how to create these sets in the "Defining Sets" section of Chapter 8, "Mastering PivotTables."
- You have to create a specific DAX expression for each combination of Period calculations and measures that you want to support. This is not flexible and scalable as a more generic solution could be.

You have to evaluate case by case whether or not these drawbacks make the implementation of a Period table a good option.

Calculation Parameters Using an Unrelated Table

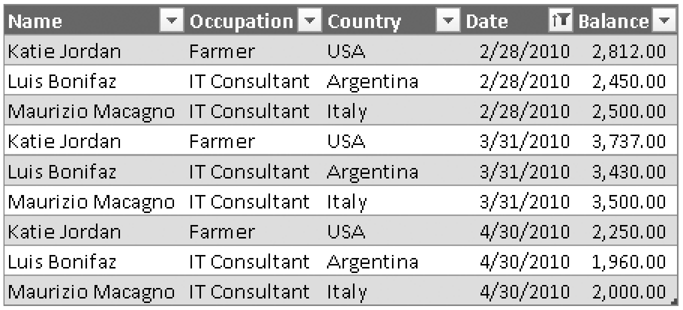
In the last section, you saw us apply a technique that depends on a table in the PowerPivot model that does not have any relationships with other tables in the same model. In general, this technique might be useful as a way to pass information to a measure as if it were a parameter. For example, imagine that you define a table in Excel that contains all the integers from 1 to 10, and then you import this table into PowerPivot, calling it SimulationParameter. At this point, you might use the value selected in this table in the DAX expressions of your measures, using that number as if it were a parameter passed to your formula. Moving that table into a slicer would be a convenient way for an end user to look at results by changing the selected value.

Closing Balance over Time

In a PivotTable, each cell contains the result of applying an aggregation function to a measure. Whenever that function is SUM, the measure is called an additive measure because SUM is applied over all dimensions. Whenever another function is applied, such as AVERAGE, MIN, or MAX, the measure is called a nonadditive measure because an aggregation function other than SUM is applied over all dimensions. However, it is important to note that both for additive and nonadditive measures, the same aggregation function is always applied over all dimensions, without exception.

Semiadditive Measures

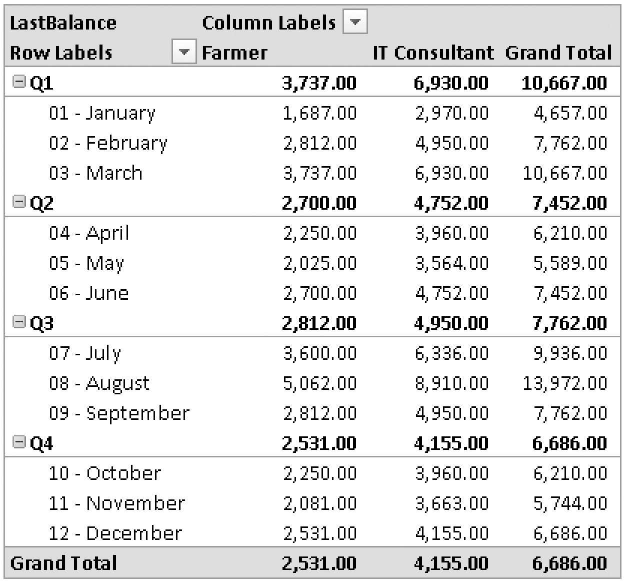
Some measures should behave in a different way. For example, think about the balance for a bank account. If you consider several accounts, you can calculate the total balance for an occupation by summing up all the balances of customers grouped by occupation. However, you cannot sum the same balance twice, and you probably have several balances of the same account that measure it over time. For example, in Figure 7-49, you can see a Balance table in Excel: the same account has a balance value for each date. This type of measure is called a semiadditive measure, because it can be aggregated using SUM over some dimensions but requires a different aggregation algorithm over other dimensions. You can find the following example in the CH07-11-SemiAdditive.xlsx workbook included on the companion DVD.



Name	Occupation	Country	Date	Balance
Katie Jordan	Farmer	USA	2/28/2010	2,812.00
Luis Bonifaz	IT Consultant	Argentina	2/28/2010	2,450.00
Maurizio Macagno	IT Consultant	Italy	2/28/2010	2,500.00
Katie Jordan	Farmer	USA	3/31/2010	3,737.00
Luis Bonifaz	IT Consultant	Argentina	3/31/2010	3,430.00
Maurizio Macagno	IT Consultant	Italy	3/31/2010	3,500.00
Katie Jordan	Farmer	USA	4/30/2010	2,250.00
Luis Bonifaz	IT Consultant	Argentina	4/30/2010	1,960.00
Maurizio Macagno	IT Consultant	Italy	4/30/2010	2,000.00

FIGURE 7-49 The raw balance account data.

In the case of account balance data, the only dimension that cannot be summed is the Date. With the term *dimension Date,* we include all the attributes of a Dates table related to the table containing the real measures. The logic that has to be implemented for the Date attributes is to consider only the values belonging to the last date in the evaluated period. In other words, you must implement a logic that can produce the same results that you see in Figure 7-50.



LastBalance	Column Labels		
Row Labels	Farmer	IT Consultant	Grand Total
Q1	**3,737.00**	**6,930.00**	**10,667.00**
01 - January	1,687.00	2,970.00	4,657.00
02 - February	2,812.00	4,950.00	7,762.00
03 - March	3,737.00	6,930.00	10,667.00
Q2	**2,700.00**	**4,752.00**	**7,452.00**
04 - April	2,250.00	3,960.00	6,210.00
05 - May	2,025.00	3,564.00	5,589.00
06 - June	2,700.00	4,752.00	7,452.00
Q3	**2,812.00**	**4,950.00**	**7,762.00**
07 - July	3,600.00	6,336.00	9,936.00
08 - August	5,062.00	8,910.00	13,972.00
09 - September	2,812.00	4,950.00	7,762.00
Q4	**2,531.00**	**4,155.00**	**6,686.00**
10 - October	2,250.00	3,960.00	6,210.00
11 - November	2,081.00	3,663.00	5,744.00
12 - December	2,531.00	4,155.00	6,686.00
Grand Total	**2,531.00**	**4,155.00**	**6,686.00**

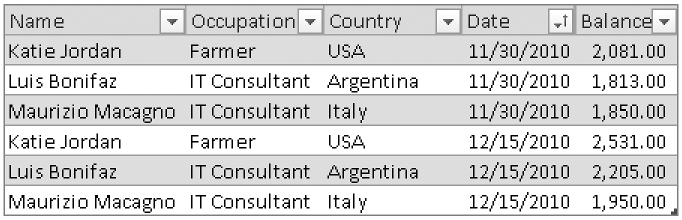
FIGURE 7-50 The result of applying the LastBalance measure.

The LastBalance measure used in Figure 7-50 calculates the total of a quarter by using just the last month available in that period. For each month, only the last date for that month is considered. So the total of a quarter is calculated using only the last day of that quarter. You can define the LastBalance measure in this way:

```
LastBalance = CALCULATE( SUM( Balances[Balance] ), LASTDATE( BalanceDate[Date] ) )
```

The definition of the LastBalance measure uses the LASTDATE function to keep just the last date that is active in the current filter context. So only the last date in the selected period is considered in the CALCULATE call.

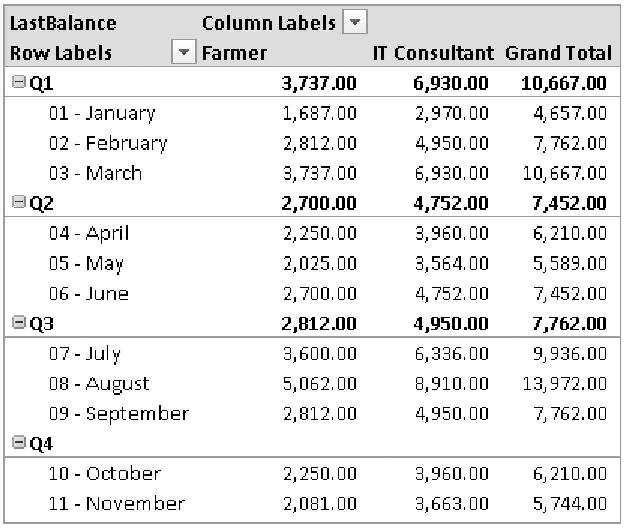
As usual, you must use a separate Dates table. Remember that the last date in a period is the last date available in the BalanceDate table (mentioned in the preceding formula) and not the last date for which there is raw data. This might have unwanted consequences. If your data does not have values for the last day of a month and the Dates table contains all the days for that month, the LastBalance formula you have used returns no data (a blank value) for that month. Consider the last two months available in the Balances table, as shown in Figure 7-51.



Name	Occupation	Country	Date	Balance
Katie Jordan	Farmer	USA	11/30/2010	2,081.00
Luis Bonifaz	IT Consultant	Argentina	11/30/2010	1,813.00
Maurizio Macagno	IT Consultant	Italy	11/30/2010	1,850.00
Katie Jordan	Farmer	USA	12/15/2010	2,531.00
Luis Bonifaz	IT Consultant	Argentina	12/15/2010	2,205.00
Maurizio Macagno	IT Consultant	Italy	12/15/2010	1,950.00

FIGURE 7-51 The last two months of balance account data.

The Balances table contains a balance for each account and each last day of the month, but for December the last day available is December 15. If the BalanceDate table contains all the days for year 2010, including 31 days for December, the LastBalance measure tries to filter balance data for December 31, which is not available, resulting in a PivotTable like the one shown in Figure 7-52, where the row for December is missing.



LastBalance	Column Labels		
Row Labels	Farmer	IT Consultant	Grand Total
⊟Q1	**3,737.00**	**6,930.00**	**10,667.00**
01 - January	1,687.00	2,970.00	4,657.00
02 - February	2,812.00	4,950.00	7,762.00
03 - March	3,737.00	6,930.00	10,667.00
⊟Q2	**2,700.00**	**4,752.00**	**7,452.00**
04 - April	2,250.00	3,960.00	6,210.00
05 - May	2,025.00	3,564.00	5,589.00
06 - June	2,700.00	4,752.00	7,452.00
⊟Q3	**2,812.00**	**4,950.00**	**7,762.00**
07 - July	3,600.00	6,336.00	9,936.00
08 - August	5,062.00	8,910.00	13,972.00
09 - September	2,812.00	4,950.00	7,762.00
⊟Q4			
10 - October	2,250.00	3,960.00	6,210.00
11 - November	2,081.00	3,663.00	5,744.00

FIGURE 7-52 December and 4th Quarter totals are missing.

A possible solution is to delete rows from the BalanceDate from December 16 through December 31. In this way, the LastBalance measure returns values as previously shown in Figure 7-50. Another option is to use the LASTNONBLANK function, which returns the last date for which a particular expression is not blank. The use of this function is not very intuitive when the Dates column and the expression you want to evaluate manage different tables. First of all, this is a formula for a LastBalanceNonBlank measure that works also with the BalanceDate complete with all the dates through December 31.

```
LastBalancaNonBlank = CALCULATE( SUM( Balances[Balance] ),
                                 LASTNONBLANK( BalanceDate[Date],
                                               COUNTROWS( RELATEDTABLE(Balances) ) ) )
```

The preceding formula produces exactly the result you saw in Figure 7-50, without your needing to remove rows from the BalanceDate table.

Using FIRSTNONBLANK and LASTNONBLANK Functions

The LASTNONBLANK function you have just seen has a particular behavior, shared also by FIRSTNONBLANK. The syntax of these functions is the following one:

```
FIRSTNONBLANK( <column>, <expression> )
LASTNONBLANK( <column>, <expression> )
```

These functions return the first or last value in <column>, filtered by the current context, wherein the <expression> is not blank. So these functions behave like SUMX or similar functions in this regard. They set a row context for a value of <column> and then evaluate the <expression> by using that row context. If <expression> and <column> manage data of the same table, everything works fine. However, whenever <expression> uses columns of tables other than the one to which <column> belongs, you need to transform a row context into a filter context by using RELATEDTABLE or CALCULATE. This is a very common situation every time you have a separate Dates table, which is the best practice for every date-related calculation.

To get the right value for the last nonblank date for a given measure/table, you have to use something like this:

```
=LASTNONBLANK( Dates[Date], CALCULATE( COUNT( Balances[Balance] ) ) )
```

It returns the last date (in the current filter context) for which there are values for the Balance column in the Balances table. You can also use an equivalent formula:

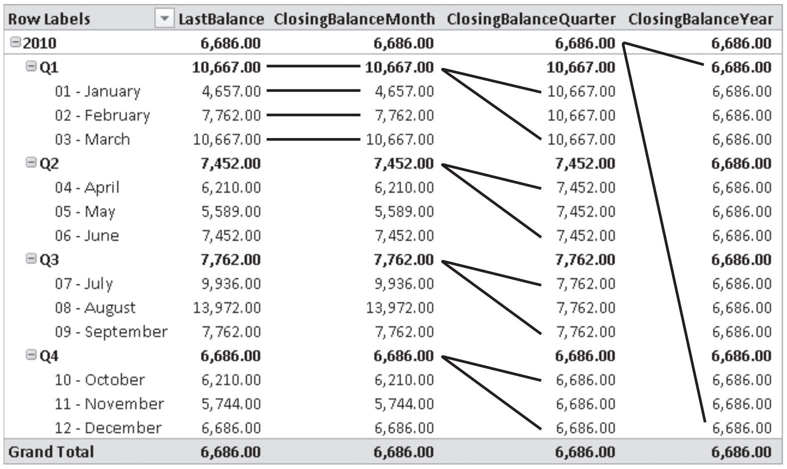
```
=LASTNONBLANK( Dates[Date], COUNTROWS( RELATEDTABLE( Sales ) ) )
```

This formula returns the last date (in the current filter context) for which there is a related row in the Sales table.

OPENINGBALANCE and CLOSINGBALANCE Functions

DAX provides several functions to get the first and last date of a period (year, quarter, or month) that are useful whenever you need to get that value of a selection that is smaller than the whole period considered. For example, looking at the month level (which may be displayed in rows), you might want to display also the value of the end of the quarter and the end of the year in the same row, as you can see in Figure 7-53. (The examples shown in this section are also available in the CH07-12-ClosingBalance.xlsx workbook included on the companion DVD.)

Note Please note that raw data used in this example includes balances for dates through December 31. For this reason, the DAX function we are going to use provides complete results because the data based on the LASTDATE function would not work if the last day of a period (such as month, quarter, or year) were missing.



Row Labels	LastBalance	ClosingBalanceMonth	ClosingBalanceQuarter	ClosingBalanceYear
2010	**6,686.00**	**6,686.00**	**6,686.00**	**6,686.00**
Q1	**10,667.00**	**10,667.00**	**10,667.00**	**6,686.00**
01 - January	4,657.00	4,657.00	10,667.00	6,686.00
02 - February	7,762.00	7,762.00	10,667.00	6,686.00
03 - March	10,667.00	10,667.00	10,667.00	6,686.00
Q2	**7,452.00**	**7,452.00**	**7,452.00**	**6,686.00**
04 - April	6,210.00	6,210.00	7,452.00	6,686.00
05 - May	5,589.00	5,589.00	7,452.00	6,686.00
06 - June	7,452.00	7,452.00	7,452.00	6,686.00
Q3	**7,762.00**	**7,762.00**	**7,762.00**	**6,686.00**
07 - July	9,936.00	9,936.00	7,762.00	6,686.00
08 - August	13,972.00	13,972.00	7,762.00	6,686.00
09 - September	7,762.00	7,762.00	7,762.00	6,686.00
Q4	**6,686.00**	**6,686.00**	**6,686.00**	**6,686.00**
10 - October	6,210.00	6,210.00	6,686.00	6,686.00
11 - November	5,744.00	5,744.00	6,686.00	6,686.00
12 - December	6,686.00	6,686.00	6,686.00	6,686.00
Grand Total	**6,686.00**	**6,686.00**	**6,686.00**	**6,686.00**

FIGURE 7-53 The balance data at end of month, quarter, and year for each month.

The formulas used to calculate ClosingBalanceMonth, ClosingBalanceQuarter, and ClosingBalanceYear measures are the following:

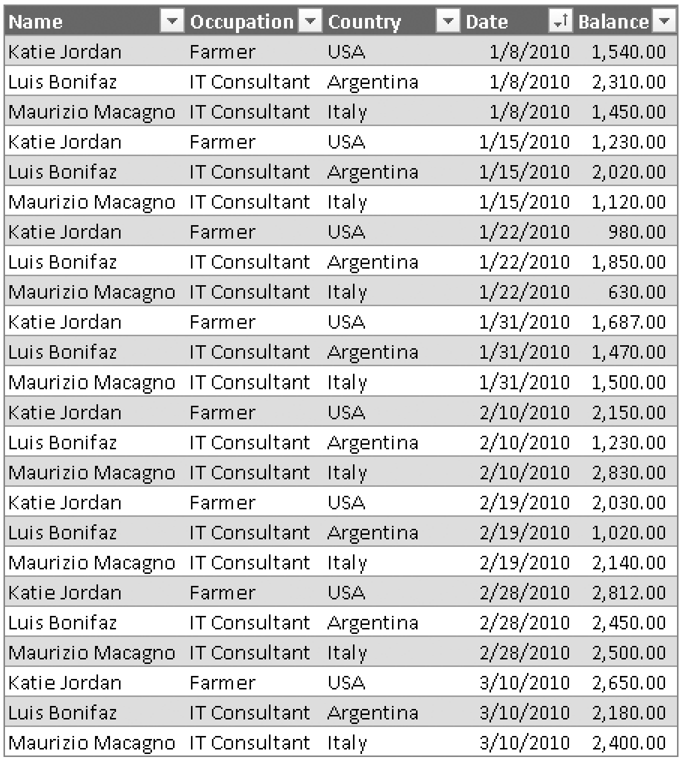
```
ClosingBalanceMonth = CLOSINGBALANCEMONTH( SUM( Balances[Balance] ), BalanceDate[Date] )
ClosingBalanceQuarter = CLOSINGBALANCEQUARTER( SUM( Balances[Balance] ), BalanceDate[Date] )
ClosingBalanceYear = CLOSINGBALANCEYEAR( SUM( Balances[Balance] ), BalanceDate[Date] )
```

These formulas use the LASTDATE function internally, but they operate on a set of dates that can extend the current selection in the PivotTable. For example, the CLOSINGBALANCEYEAR function considers the LASTDATE of Balance[Date], which is applied to the last year period of the dates included in the filter context. So for February 2010 (and for any month or quarter of 2010), this date is December 31, 2010. The CLOSINGBALANCEYEAR function behaves like a CALCULATE expression using the ENDOFYEAR function as a filter. As usual, the use of CALCULATE is more generic and flexible, but specific DAX functions like CLOSINGBALANCEYEAR better express the intention of the measure designer. The following are measures equivalent to the ones previously shown using CALCULATE syntax.

```
ClosingBalanceEOM = CALCULATE( SUM( Balances[Balance] ), ENDOFMONTH( BalanceDate[Date] ) )
ClosingBalanceEOQ = CALCULATE( SUM( Balances[Balance] ), ENDOFQUARTER( BalanceDate[Date] ) )
ClosingBalanceEOY = CALCULATE( SUM( Balances[Balance] ), ENDOFYEAR( BalanceDate[Date] ) )
```

Tip The DAX functions OPENINGBALANCEMONTH, OPENINGBALANCEQUARTER, and OPENINGBALANCEYEAR use the FIRSTDATE internally instead of the LASTDATE of the considered period. They correspond to the CALCULATE formula, which uses STARTOFMONTH, STARTOFQUARTER, and STARTOFYEAR internally as its filter, respectively.

An important consideration has to be made about dates for which there is available data in your model. You can see this if you drill down to data at the day level in the PivotTable. Before doing that, consider the raw data set we used in this example, shown in Figure 7-54. As you can see, there are more balances for each month. For example, in January there are balances for days 8, 15, 22, and 31.

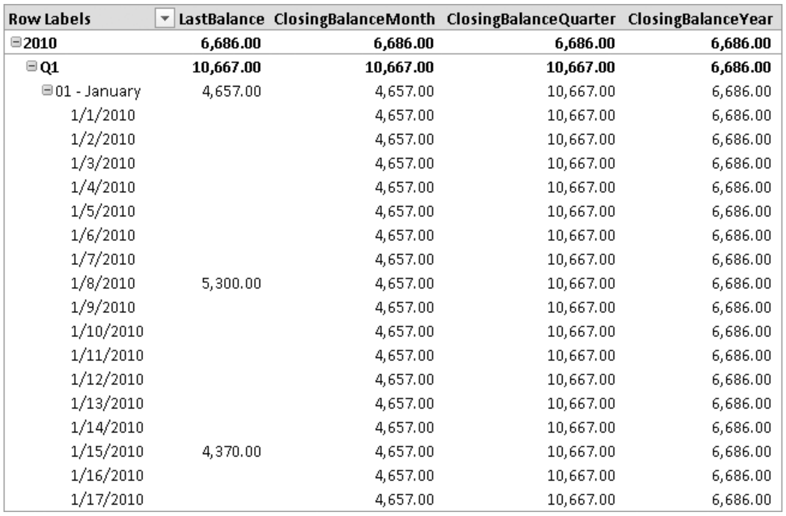


Name	Occupation	Country	Date	Balance
Katie Jordan	Farmer	USA	1/8/2010	1,540.00
Luis Bonifaz	IT Consultant	Argentina	1/8/2010	2,310.00
Maurizio Macagno	IT Consultant	Italy	1/8/2010	1,450.00
Katie Jordan	Farmer	USA	1/15/2010	1,230.00
Luis Bonifaz	IT Consultant	Argentina	1/15/2010	2,020.00
Maurizio Macagno	IT Consultant	Italy	1/15/2010	1,120.00
Katie Jordan	Farmer	USA	1/22/2010	980.00
Luis Bonifaz	IT Consultant	Argentina	1/22/2010	1,850.00
Maurizio Macagno	IT Consultant	Italy	1/22/2010	630.00
Katie Jordan	Farmer	USA	1/31/2010	1,687.00
Luis Bonifaz	IT Consultant	Argentina	1/31/2010	1,470.00
Maurizio Macagno	IT Consultant	Italy	1/31/2010	1,500.00
Katie Jordan	Farmer	USA	2/10/2010	2,150.00
Luis Bonifaz	IT Consultant	Argentina	2/10/2010	1,230.00
Maurizio Macagno	IT Consultant	Italy	2/10/2010	2,830.00
Katie Jordan	Farmer	USA	2/19/2010	2,030.00
Luis Bonifaz	IT Consultant	Argentina	2/19/2010	1,020.00
Maurizio Macagno	IT Consultant	Italy	2/19/2010	2,140.00
Katie Jordan	Farmer	USA	2/28/2010	2,812.00
Luis Bonifaz	IT Consultant	Argentina	2/28/2010	2,450.00
Maurizio Macagno	IT Consultant	Italy	2/28/2010	2,500.00
Katie Jordan	Farmer	USA	3/10/2010	2,650.00
Luis Bonifaz	IT Consultant	Argentina	3/10/2010	2,180.00
Maurizio Macagno	IT Consultant	Italy	3/10/2010	2,400.00

FIGURE 7-54 The raw balance data with more balances for each month.

Note In this example, we always have a balance value for each account, as if we took a snapshot on a certain date for every account available, even if it has not changed its value since the previous date. We see in the next section what to do whenever this condition is not true.

If you browse this data at the day level in the PivotTable by using the same measures as the previous example, you see the results shown in Figure 7-55.



Row Labels	LastBalance	ClosingBalanceMonth	ClosingBalanceQuarter	ClosingBalanceYear
2010	**6,686.00**	**6,686.00**	**6,686.00**	**6,686.00**
Q1	**10,667.00**	**10,667.00**	**10,667.00**	**6,686.00**
01 - January	4,657.00	4,657.00	10,667.00	6,686.00
1/1/2010		4,657.00	10,667.00	6,686.00
1/2/2010		4,657.00	10,667.00	6,686.00
1/3/2010		4,657.00	10,667.00	6,686.00
1/4/2010		4,657.00	10,667.00	6,686.00
1/5/2010		4,657.00	10,667.00	6,686.00
1/6/2010		4,657.00	10,667.00	6,686.00
1/7/2010		4,657.00	10,667.00	6,686.00
1/8/2010	5,300.00	4,657.00	10,667.00	6,686.00
1/9/2010		4,657.00	10,667.00	6,686.00
1/10/2010		4,657.00	10,667.00	6,686.00
1/11/2010		4,657.00	10,667.00	6,686.00
1/12/2010		4,657.00	10,667.00	6,686.00
1/13/2010		4,657.00	10,667.00	6,686.00
1/14/2010		4,657.00	10,667.00	6,686.00
1/15/2010	4,370.00	4,657.00	10,667.00	6,686.00
1/16/2010		4,657.00	10,667.00	6,686.00
1/17/2010		4,657.00	10,667.00	6,686.00

FIGURE 7-55 Browsing data at the day level displays rows with no balance data.

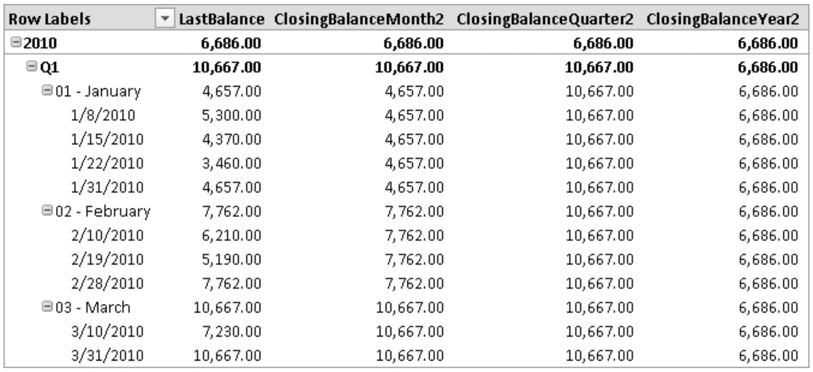
As you can see, the measures defined to display values at the end of the period suffer an unpleasant side effect: all the dates are visible, even those for which there are no balance data available. If you want to display just the rows corresponding to dates with balance data defined, you have to modify the measures, checking the existence of data in the Balances table, in this way:

```
ClosingBalanceMonth2
    = IF( COUNTROWS( Balances ) > 0,
          CLOSINGBALANCEMONTH( SUM( Balances[Balance] ), BalanceDate[Date] ),
          BLANK() )

ClosingBalanceQuarter2
    = IF( COUNTROWS( Balances ) > 0,
          CLOSINGBALANCEQUARTER( SUM( Balances[Balance] ), BalanceDate[Date] ),
          BLANK() )

ClosingBalanceYear2
    = IF( COUNTROWS( Balances ) > 0,
          CLOSINGBALANCEYEAR( SUM( Balances[Balance] ), BalanceDate[Date] ),
          BLANK() )
```

Browsing data using these measures results in a report like the one shown in Figure 7-56.



Row Labels	LastBalance	ClosingBalanceMonth2	ClosingBalanceQuarter2	ClosingBalanceYear2
⊟2010	**6,686.00**	**6,686.00**	**6,686.00**	**6,686.00**
⊟Q1	**10,667.00**	**10,667.00**	**10,667.00**	**6,686.00**
⊟01 - January	4,657.00	4,657.00	10,667.00	6,686.00
1/8/2010	5,300.00	4,657.00	10,667.00	6,686.00
1/15/2010	4,370.00	4,657.00	10,667.00	6,686.00
1/22/2010	3,460.00	4,657.00	10,667.00	6,686.00
1/31/2010	4,657.00	4,657.00	10,667.00	6,686.00
⊟02 - February	7,762.00	7,762.00	10,667.00	6,686.00
2/10/2010	6,210.00	7,762.00	10,667.00	6,686.00
2/19/2010	5,190.00	7,762.00	10,667.00	6,686.00
2/28/2010	7,762.00	7,762.00	10,667.00	6,686.00
⊟03 - March	10,667.00	10,667.00	10,667.00	6,686.00
3/10/2010	7,230.00	10,667.00	10,667.00	6,686.00
3/31/2010	10,667.00	10,667.00	10,667.00	6,686.00

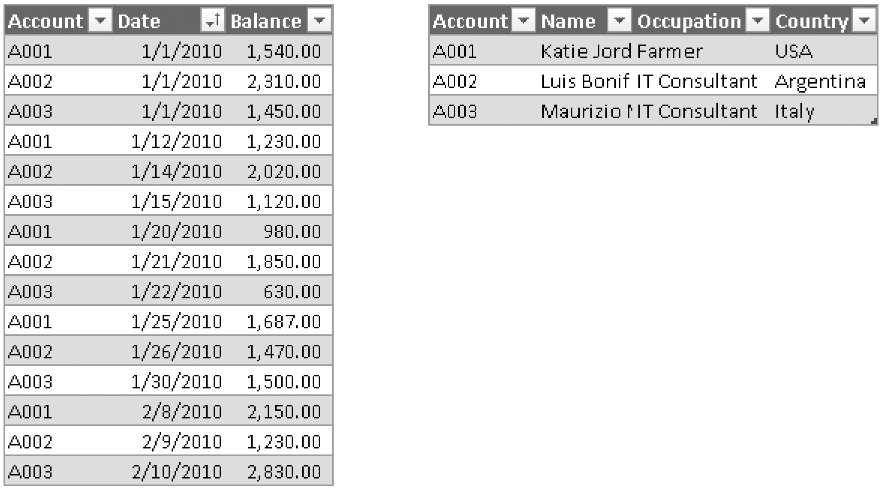
FIGURE 7-56 Using measures that display only days for which there is balance data.

By default, the PivotTable in Excel does not display empty rows and columns. For this reason, the days containing no balance date are not shown: all the measures used in the PivotTable return BLANK for those days, removing them from the report.

Updating Balances by Using Transactions

The balance account model you saw in the previous section makes an important assumption: for a given date, either data is not present at all or all the accounts have a balance value for that date. In case an account does not have a balance value for a date that other accounts are measured, that account is considered to have a zero balance for that date. This assumption is good for certain data structures, which are generated by a system that makes a snapshot of the situation (all balance accounts values) on a given date.

However, some scenarios have a different data model in which the previous assumption is not valid. For example, consider this other way to collect data about balance accounts. In the Balances table shown in Figure 7-57, data has been normalized by means of an Accounts table, which can be seen on the right side of the same figure. (The model used in this section is available in the CH07-13-ClosingTransaction.xlsx workbook included on the companion DVD.) Moreover, you can find a balance row for an account only for dates when a transaction made some changes in the account balance.



Account	Date	Balance
A001	1/1/2010	1,540.00
A002	1/1/2010	2,310.00
A003	1/1/2010	1,450.00
A001	1/12/2010	1,230.00
A002	1/14/2010	2,020.00
A003	1/15/2010	1,120.00
A001	1/20/2010	980.00
A002	1/21/2010	1,850.00
A003	1/22/2010	630.00
A001	1/25/2010	1,687.00
A002	1/26/2010	1,470.00
A003	1/30/2010	1,500.00
A001	2/8/2010	2,150.00
A002	2/9/2010	1,230.00
A003	2/10/2010	2,830.00

Account	Name	Occupation	Country
A001	Katie Jord	Farmer	USA
A002	Luis Bonif	IT Consultant	Argentina
A003	Maurizio	IT Consultant	Italy

FIGURE 7-57 Raw balance account data updated for transactions and not in snapshots.

As you can see, account A001 changes its value on January 1, 12, 20, and 25; account A002 changes on January 1, 14, 21, and 26; and account A003 changes on January 1, 15, 22, and 30. There is no data at the end of month (January 31), and there is no data for all accounts on a given date (for example, January 12 has an account balance only for account A001). So neither LastBalance nor ClosingBalance measures we have seen before can work with this data because their initial assumptions are not valid anymore. We must create a more complex calculation.

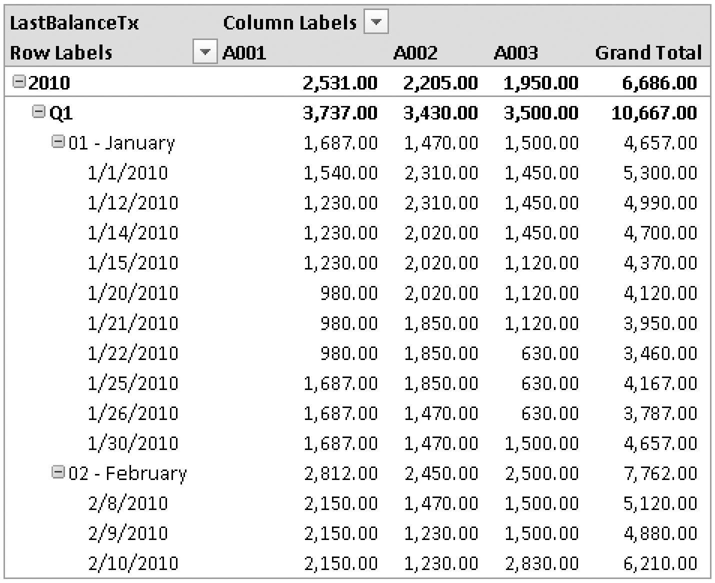
The basic idea is that, for each account, you must get the last nonblank date included in the selected period. The calculation for a single account can be made by using the CALCULATE function and by filtering data on the LASTNONBLANK date included in the period between the first date available and the last date in the period. Notice that the date range considered begins even outside the period: you might request the balance for February and there might be no rows in that month, so previous dates also must be considered for the interval. You use a SUMX function to iterate all the available accounts.

```
SUMX( ALL( Balances[Account] ),
      CALCULATE( SUM( Balances[Balance] ),
                 LASTNONBLANK( DATESBETWEEN( BalanceDate[Date],
                                             BLANK(),
                                             LASTDATE( BalanceDate[Date] ) ),
                               CALCULATE( COUNT( Balances[Balance] ) ) ) ) )
```

This expression calculates a value for each date in the BalanceDate table. To get the calculation only for dates that have at least one transaction (for any account), you must make a test similar to the one you saw already in the previous section for ClosingBalance measures. Finally, you can define the complete LastBalanceTx measure by using this DAX formula:

```
LastBalanceTx
  = IF( COUNTX( BalanceDate,
                CALCULATE( COUNT( Balances[Balance] ),
                           ALLEXCEPT( Balances, BalanceDate[Date] ) ) ) > 0,
        SUMX( ALL( Balances[Account] ),
              CALCULATE( SUM( Balances[Balance] ),
                         LASTNONBLANK( DATESBETWEEN( BalanceDate[Date],
                                                     BLANK(),
                                                     LASTDATE( BalanceDate[Date] ) ),
                                       CALCULATE( COUNT( Balances[Balance] ) ) ) ) ),
        BLANK() )
```

This formula produces the result shown in Figure 7-58, in which you can see the balance updated for each account (one for each column) only for days in which at least one new balance is present in the Balances table.



LastBalanceTx	Column Labels			
Row Labels	**A001**	**A002**	**A003**	**Grand Total**
⊟2010	**2,531.00**	**2,205.00**	**1,950.00**	**6,686.00**
⊟Q1	**3,737.00**	**3,430.00**	**3,500.00**	**10,667.00**
⊟01 - January	1,687.00	1,470.00	1,500.00	4,657.00
1/1/2010	1,540.00	2,310.00	1,450.00	5,300.00
1/12/2010	1,230.00	2,310.00	1,450.00	4,990.00
1/14/2010	1,230.00	2,020.00	1,450.00	4,700.00
1/15/2010	1,230.00	2,020.00	1,120.00	4,370.00
1/20/2010	980.00	2,020.00	1,120.00	4,120.00
1/21/2010	980.00	1,850.00	1,120.00	3,950.00
1/22/2010	980.00	1,850.00	630.00	3,460.00
1/25/2010	1,687.00	1,850.00	630.00	4,167.00
1/26/2010	1,687.00	1,470.00	630.00	3,787.00
1/30/2010	1,687.00	1,470.00	1,500.00	4,657.00
⊟02 - February	2,812.00	2,450.00	2,500.00	7,762.00
2/8/2010	2,150.00	1,470.00	1,500.00	5,120.00
2/9/2010	2,150.00	1,230.00	1,500.00	4,880.00
2/10/2010	2,150.00	1,230.00	2,830.00	6,210.00

FIGURE 7-58 Results of LastBalanceTx measure.

Tip This scenario requires a particularly complex DAX calculation, which becomes much more complicated if other tables are added to the model. The document available in the Microsoft PowerPivot for Excel 2010 Data Analysis Expressions Sample (which can be downloaded at *http://tinyurl.com/DaxSample*) shows a similar example in the Time Intelligence Functions section involving an Inventory Scenario with two tables other than the Dates table. Take a look at that document if you have a similar scenario.

Keep in mind that the Balances[Account] column used to make the relationship with the Accounts table is used in the LastBalanceTx formula and should not be selected in the PivotTable. Instead of that, you should use the Accounts[Account] column; otherwise, you could see wrong data in the PivotTable. The reason is similar to the case for which we suggest you use a Dates table instead of denormalizing all dates information in the same table that contains the measures. So a best practice is to hide in PivotTable all the columns that you use to relate Balances (the table containing measures) to other tables such as BalanceDate and Accounts (which are the tables containing attributes for browsing data).

Summary

In this chapter, you saw how to create a Dates table for a PowerPivot model and how to use that table to support several types of calculations: number of working days, aggregation and comparison over time, and closing balance over time.