



# PowerPivot

## Succinctly

by James Beresford



| Technology Resource Portal

# PowerPivot Succinctly

---

By

**James Beresford**

Foreword by Daniel Jebaraj



Copyright © 2015 by Syncfusion Inc.

2501 Aerial Center Parkway

Suite 200

Morrisville, NC 27560

USA

All rights reserved.

## **Important licensing information. Please read.**

This book is available for free download from [www.syncfusion.com](http://www.syncfusion.com) on completion of a registration form.

If you obtained this book from any other source, please register and download a free copy from [www.syncfusion.com](http://www.syncfusion.com).

This book is licensed for reading only if obtained from [www.syncfusion.com](http://www.syncfusion.com).

This book is licensed strictly for personal or educational use.

Redistribution in any form is prohibited.

The authors and copyright holders provide absolutely no warranty for any information provided.

The authors and copyright holders shall not be liable for any claim, damages, or any other liability arising from, out of, or in connection with the information in this book.

Please do not use this book if the listed terms are unacceptable.

Use shall constitute acceptance of the terms listed.

SYNCFUSION, SUCCINCTLY, DELIVER INNOVATION WITH EASE, ESSENTIAL, and .NET ESSENTIALS are the registered trademarks of Syncfusion, Inc.

**Technical Reviewer:** Parikshit Savjani

**Copy Editor:** Courtney Wright

**Acquisitions Coordinator:** Hillary Bowling, marketing coordinator, Syncfusion, Inc.

**Proofreader:** Darren West, content producer, Syncfusion, Inc.

# Table of Contents

<b>The Story behind the <i>Succinctly</i> Series of Books .....</b>	<b>7</b>
<b>About the Author .....</b>	<b>9</b>
<b>Introduction .....</b>	<b>10</b>
In-memory cubes for the desktop .....	10
DAX vs. MDX .....	11
<b>Chapter 1 PowerPivot Model Basics .....</b>	<b>12</b>
Getting started .....	12
Enable PowerPivot.....	12
Opening the PowerPivot data model .....	12
Tables .....	13
Using data within the same workbook .....	13
Using external data from an Enterprise source .....	16
Using external data from the Windows Azure Marketplace.....	21
Other supported data sources .....	24
Data types.....	24
Hierarchies .....	26
Relationships .....	31
Creating relationships manually.....	32
Calculated columns.....	36
DAX.....	36
Creating a simple column .....	36
Lookup up data from another unrelated table.....	40
Lookup up data from another related table.....	42
Calculations on data in another table .....	42
Calculations .....	46
Default aggregations.....	47
Context in calculations .....	49

Time Intelligence.....	49
Context.....	54
Row context .....	54
Query context.....	55
Filter context .....	56
<b>Chapter 2 Using your PowerPivot Model.....</b>	<b>58</b>
Tidying it up.....	58
Number formatting .....	58
Friendly names .....	59
Hiding tables and columns.....	61
Pivot tables .....	62
KPI's.....	62
Slicers .....	65
Power View .....	67
Cards .....	70
Maps .....	72
<b>Chapter 3 Sharing your PowerPivot Model .....</b>	<b>75</b>
Standalone workbook .....	75
SharePoint workbook.....	75
Tabular model .....	76
Security considerations.....	78
<b>Chapter 4 A Note on Instability.....</b>	<b>79</b>
Remove the add-in.....	79
Repair Office .....	79
Try and fix the problem in the data model .....	79
<b>Chapter 5 Deep Dive: The xVelocity engine .....</b>	<b>80</b>
Understanding compression .....	80
Managing very large workbooks .....	82

<b>Additional Resources .....</b>	<b>85</b>
Microsoft.....	85
Independent .....	85

# The Story behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President  
Syncfusion, Inc.

**S**taying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

## Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the Internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

## The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn't everything wonderful born out of a deep desire to change things for the better?

## The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors' tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

## Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

## Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to “enable AJAX support with one click” or “turn the moon to cheese!”

## Let us know what you think

If you have any topics of interest, thoughts or feedback, please feel free to send them to us at [succinctly-series@syncfusion.com](mailto:succinctly-series@syncfusion.com).

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on Twitter and “Like” us on Facebook to help us spread the word about the *Succinctly* series!





# About the Author

James Beresford is a certified Microsoft Business Intelligence (BI) Consultant who has been working with the platform for over a decade. He has worked with all aspects of the stack, his specialty being extraction, transformation, and load (ETL) with *SQL Server Integration Services* (SSIS) and Data Warehousing on SQL Server. He has presented twice at TechEd in Australia and is a frequent presenter at various user groups.

His client experience includes companies in the insurance, education, logistics and banking fields. He first used the HDInsight platform in its preview stage for a telecommunications company to analyse unstructured data, and has watched the platform grow and mature since its early days.

He blogs at [www.bimonkey.com](http://www.bimonkey.com) and tweets [@BI\\_Monkey](https://twitter.com/BI_Monkey). He can be found on LinkedIn at <http://www.linkedin.com/in/jamesberesford>.

# Introduction

Did you know you can handle millions of rows of data on your desktop? Build interactive charts and tables that analyze that data instantly? Create dashboards that bring together related data without needing a huge BI project? Explore your data in real time? Even plot it on a map?

Of course you can. You just do it in Excel. No, seriously. There's a surprisingly unknown tool in Excel called PowerPivot that delivers all of those capabilities in a format that is familiar to Excel users, simple to use, and very powerful.

Microsoft first introduced PowerPivot to the world in beta as "Project Gemini" in 2009. It gave desktop-based data analysts a new way to analyze large amounts of data in memory in the familiar environment of Excel. It was formally introduced as a free add-in for Excel 2010, and made a standard component from 2013. It also found a home within Analysis Services—a core part of Microsoft's SQL Server BI Platform—as a new way of modeling data for the enterprise.

## In-memory cubes for the desktop

PowerPivot is an in-memory cube for the desktop. What that means to you firstly is that your data is modeled with all the power of a cube in a full blown BI solution. That means your data relationships are all managed for you and aggregations by different slicers is something that happens natively, without additional effort on the part of the user. It opens the door to powerful, context-aware calculations that enable analysis without you constantly reworking formulas for different scenarios.

The icing on the cake is that this all happens in your desktop's memory, which is (currently) the fastest way to work with data on a computer. The magic happens in the PowerPivot Data Model, an Excel-like interface where you can work with millions of rows of data in seconds. This lives inside your Excel workbook. There is a small limit on this in terms of memory. You can't let the Excel workbook exceed 2GB in size, and the memory it consumes can't exceed 4GB<sup>1</sup>. That's still a huge workbook – and given how PowerPivot compresses data (more on this later), that means that you could be working with up to 30GB of raw data.

---

<sup>1</sup> Maximum Memory or File Size Exceeded: <http://msdn.microsoft.com/en-us/library/gg399088%28v=sql.110%29.aspx>

## DAX vs. MDX

Cubes have traditionally been accessed using a language called MDX (for MultiDimensional Expressions). MDX is a difficult language to master and has often been one of the barriers to broader adoption of power cube technologies. The query language used by PowerPivot is DAX (for Data Analysis Expressions). The language is deliberately similar to Excel formula-like to make it accessible to business users. For example, a simple addition of two columns looks like this:

```
TwoColumns:=[ColumnA]+[ColumnB]
```

The powerful IF statement looks like this:

```
IFColumn:=IF(A=1,B,C)
```

More significantly, table level aggregations such as SUM look like this:

```
SUMMeasure:=SUM([Column])
```

Not too scary. Part of the appeal of DAX and PowerPivot technologies is that the language is much more accessible than its counterpart from the OLAP Modelled world, MDX. It's often said that the learning curves for the two languages look like this:

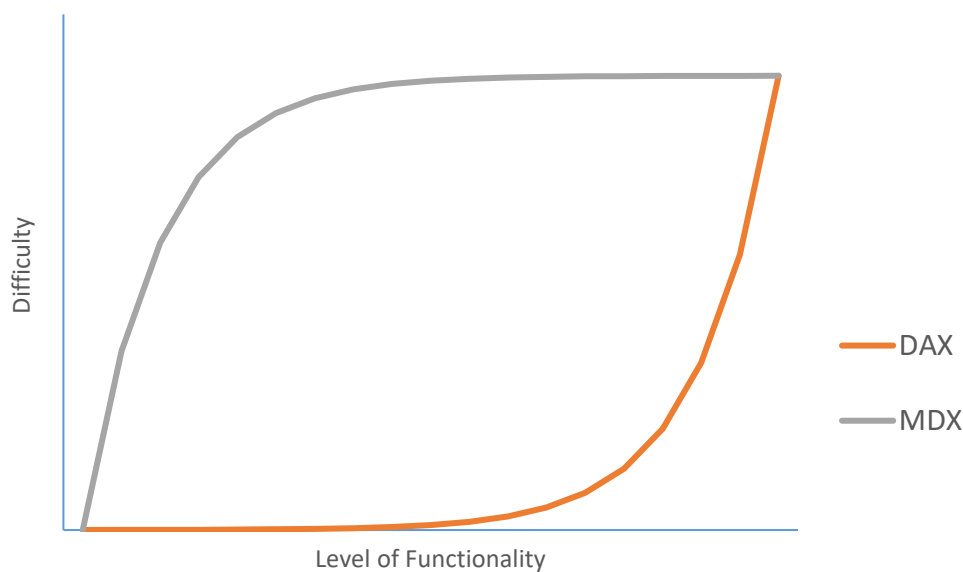


Figure 1: DAX vs MDX learning curve

DAX is easy to get started with, but the advanced capabilities are quite hard to master. In contrast, MDX is hard to get started, but once you have it under your belt it doesn't get much harder. I've listed some [resources](#) at the end of the book for those looking deeper into DAX; they might be helpful when you get to the far end of the learning curve and need some help.

# Chapter 1 PowerPivot Model Basics

To help understand how to use the capabilities of PowerPivot, we will build a model using Microsoft's AdventureWorks sample Data Warehouse for SQL Server as a source of data<sup>2</sup>. Don't worry if you don't have an SQL Server on hand—the [sample workbook](#) that supports this book contains all the data needed for these examples. The workbook is provided in two versions: one with the data imported as tables but none of the features built, and one with a completed model.

## Getting started

### Enable PowerPivot

The first thing to do is to turn it on. Depending on your version of Microsoft Office, it's either baked in, or you'll need to download the add-in. Office 2010 users will need to download the add-in<sup>3</sup>. Then just run the installer, and PowerPivot will appear on your ribbon the next time you open Excel.

If you're using Office 2013 Professional Plus, Office 365 Professional Plus, or standalone Excel, follow these quick steps:

- Go to **File > Options > Add-Ins**
- In the **Manage** box, click **COM Add-ins > Go**
- Check the **Microsoft Office Power Pivot in Microsoft Excel 2013** box, and then click **OK**

The ribbon will now feature a PowerPivot tab.

If you don't have any of the editions mentioned, you'll need to change your version of Office or Excel first.

### Opening the PowerPivot data model

You can access the data model with the PowerPivot tab on the ribbon. Choose the **Manage** option in the Data Model section of the ribbon:

---

<sup>2</sup> Microsoft SQL Server Product Samples: <http://msftdbprodsamples.codeplex.com/>

<sup>3</sup> Download Power Pivot: <http://office.microsoft.com/en-au/excel/download-power-pivot-HA101959985.aspx>

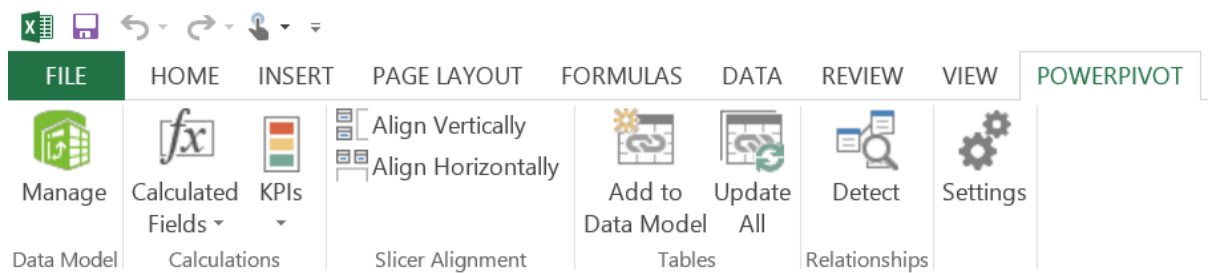


Figure 2: The PowerPivot Ribbon menu in Excel

Clicking on the Manage button launches you into a new window with a new ribbon:

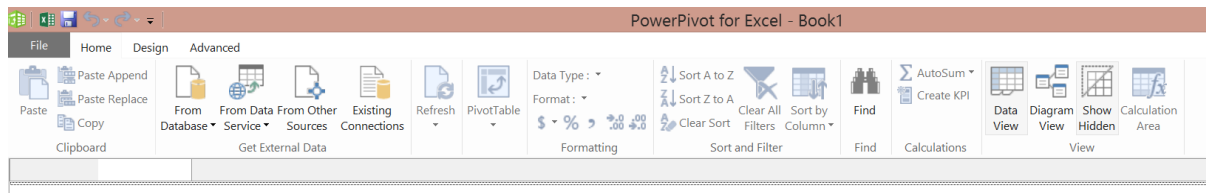


Figure 3: The PowerPivot Data Model Ribbon

The main part of the data model is a blank space to start with. We will fill this out with data and relationships as we explore further.

## Tables

The fundamental ingredients of a PowerPivot model are the tables holding data. This data can come from within the same workbook, or more powerfully from the wide range of systems you can get a connection to.

In the following example we will bring data into our model from three places:

1. Our own workbook
2. An enterprise source
3. Azure Data Market

## Using data within the same workbook

To use data from your workbook, you first need to format it as a table. For this exercise, we are going to create an alternative version of the Product subcategory in the AdventureWorks database. First we will grab the relevant existing data from the database, paste it in, and add a column called **AlternateSubCategory**:

	A	B	C	D
1	ProductSubCategoryKey	SubCategory	AlternateSubCategory	
2		1 Mountain Bikes		
3		2 Road Bikes		
4		3 Touring Bikes		
5		4 Handlebars		
6		5 Bottom Brackets		
7		6 Brakes		

Figure 4: Sample data in our workbook

Then, from the Home Tab of the ribbon, in the Styles section, choose **Format as Table**:

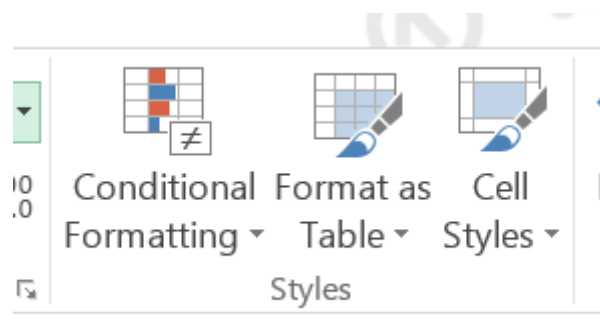


Figure 5: Format as Table

The color scheme you choose does not impact the functionality of the table. After applying a style, Excel will automatically switch to the Design tab of the ribbon for Table Tools. Here it is useful to name your table in the Properties section of the ribbon:

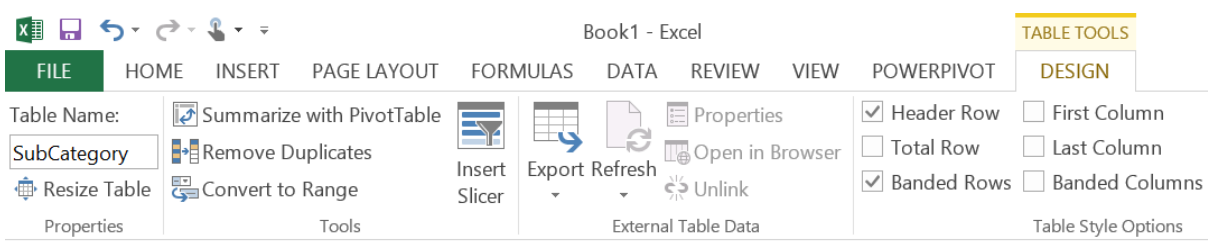


Figure 6: Table design tools

This renaming is optional, but will look better in your data model than a series of references to Table1, Table2, etc., which are the automatically created names for tables.

Now, head to the PowerPivot tab of the ribbon, and with a cell in the table highlighted, click **Add to Data Model** in the **Tables** section:

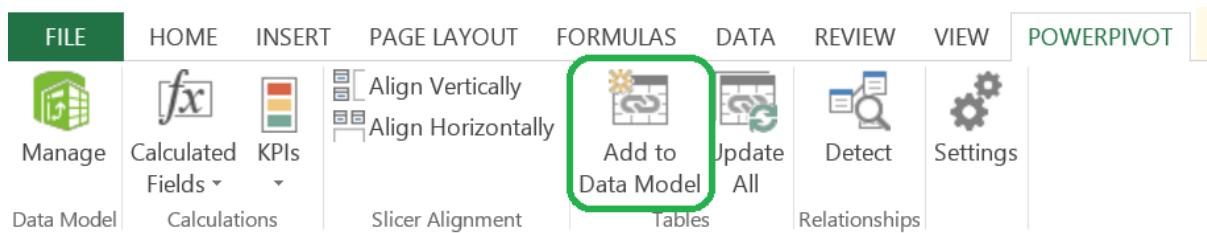


Figure 7: Adding a table to the PowerPivot Data Model

PowerPivot takes over and adds the table to the data model with no further intervention from the user. The data model manager launches and you can view your data instantly.

The screenshot shows the PowerPivot Data Model Manager interface. At the top, there is a tab labeled '[ProductS...'. Below it, a table is displayed with the following columns: ProductSubCategoryKey, SubCategory, AlternateSubCategory, and Add. The table contains 9 rows of data. The first column is highlighted in green. The bottom of the interface shows a 'SubCategory' filter and a 'Record: 1 of 37' indicator.

ProductSubCategoryKey	SubCategory	AlternateSubCategory	Add
1	Mountain Bikes		
2	Road Bikes		
3	Touring Bikes		
4	Handlebars		
5	Bottom Brackets		
6	Brakes		
7	Chains		
8	Cranksets		
9	Derailleurs		

Figure 8: Excel Data in the PowerPivot Data Model

Initially, we have put no values in the AlternateSubCategory column in our source in Excel. How values get updated in PowerPivot depend on your update settings.

The Linked Table section of the ribbon drives how this behaves:

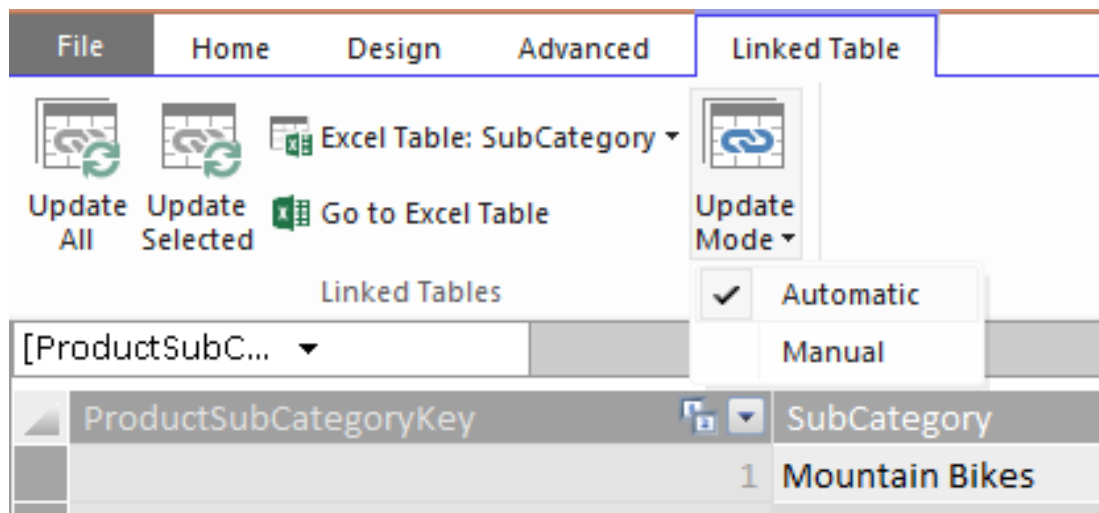


Figure 9: The Linked Table Manager

By default, the Update Mode is set to Automatic, which means as you enter data in the workbook, PowerPivot updates automatically. In the example, the worksheet was updated and the changes were reflected automatically.

It is important to note that you cannot update the data in the Data Model view—what you are looking at is a copy of your data that is reflecting what has been updated in the source—Excel. While the data grid looks a lot like Excel, remember that it is showing a read-only copy of the source data. You can [add calculations](#) later, but this is a layer on top of your source, and not changing it.

If the updates start slowing things down, as often happens in a workbook with many calculations, you can change the Update Mode setting to Manual. Then when you want to update the PowerPivot model with your worksheet updates, you select **Update All** or **Update Selected** to do so.

## Using external data from an Enterprise source

We have now created some alternate categorizations for some data in our warehouse. The next step is to pull in that warehouse data.

From the Home Tab of the PowerPivot Data Model Manager, we have the **Get External Data** section to help us do that. We'll start with the **From Database** option.



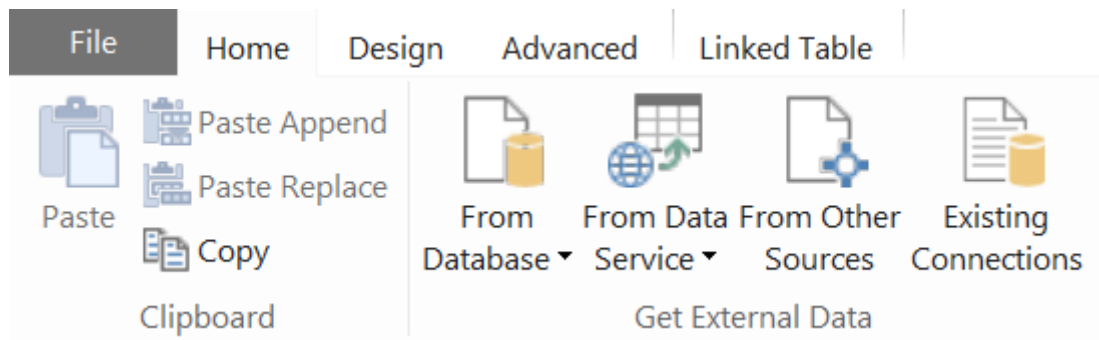


Figure 10: Importing external data

There are three options:

1. SQL Server
2. Access
3. Analysis Services or PowerPivot

If the last option seems confusing, it is because PowerPivot technology is also used in the Enterprise BI solutions—though usually referred to as Tabular models—and these can also be data sources. This will get covered in a little more detail [later](#).

In this scenario we will be looking at using SQL Server. I won't detail the wizard here, but the steps are:

1. Provide connection detail
2. Select how to import data—Whole Table/View or custom query
3. Select the tables to import, in this case **FactProductInventory**
4. Opt to select related tables
5. Click **Finish**

I'll point out a couple of things from the steps above. Firstly, when importing data you can specify to pull in an entire table or view, or, if you are comfortable with SQL, write a query that will select a specified amount of data. If you opt to pull in a whole table or view, you can use the Import Wizard to restrict which columns you are pulling in, and apply filters to those columns to restrict how many rows you pull in. As you can see in the following screenshot of the Table Import Wizard where you can select tables, there is a button for **Preview & Filter** at the bottom right-hand corner:

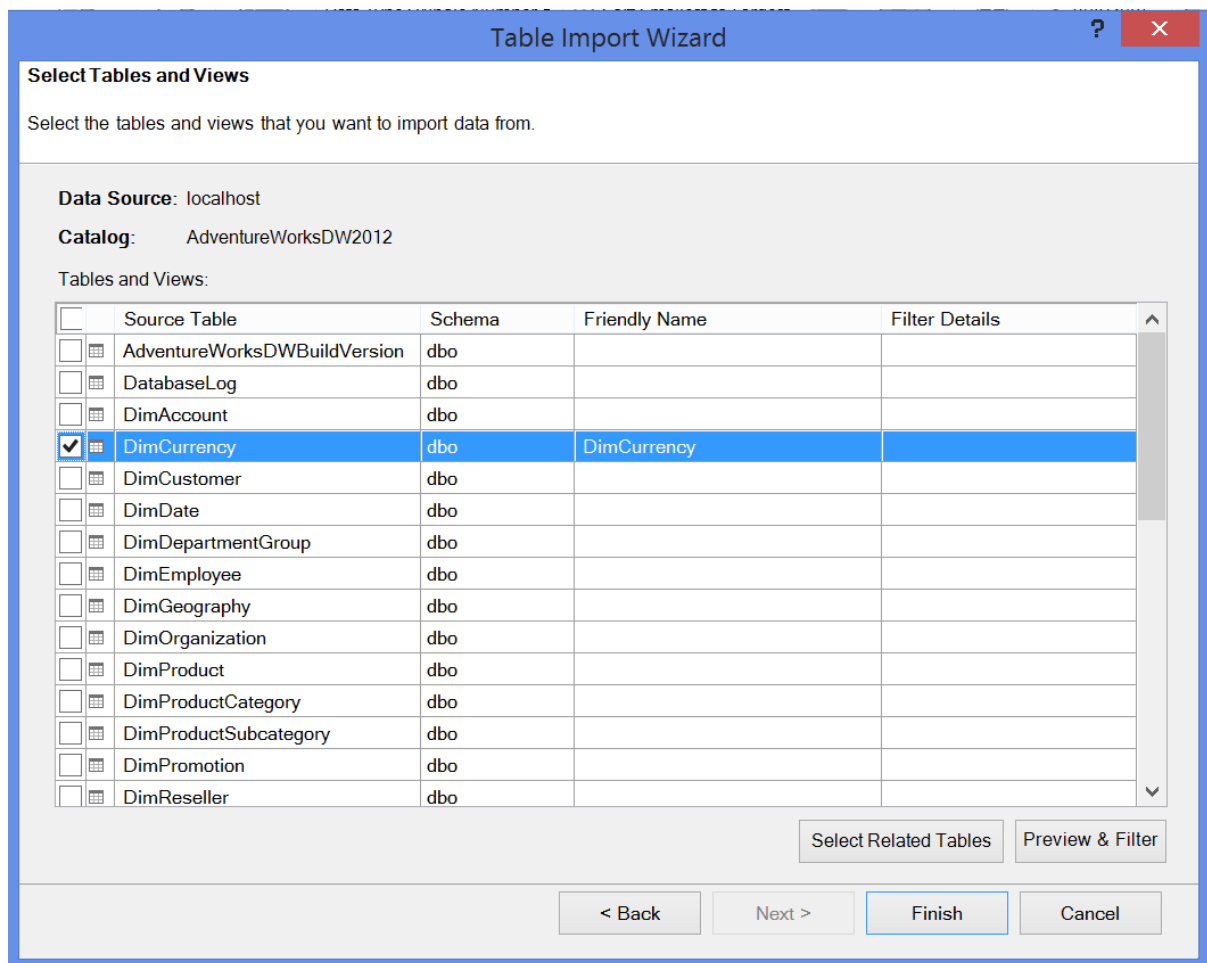


Figure 11: Table Import Wizard

Click this button to launch the Table Import Preview window:

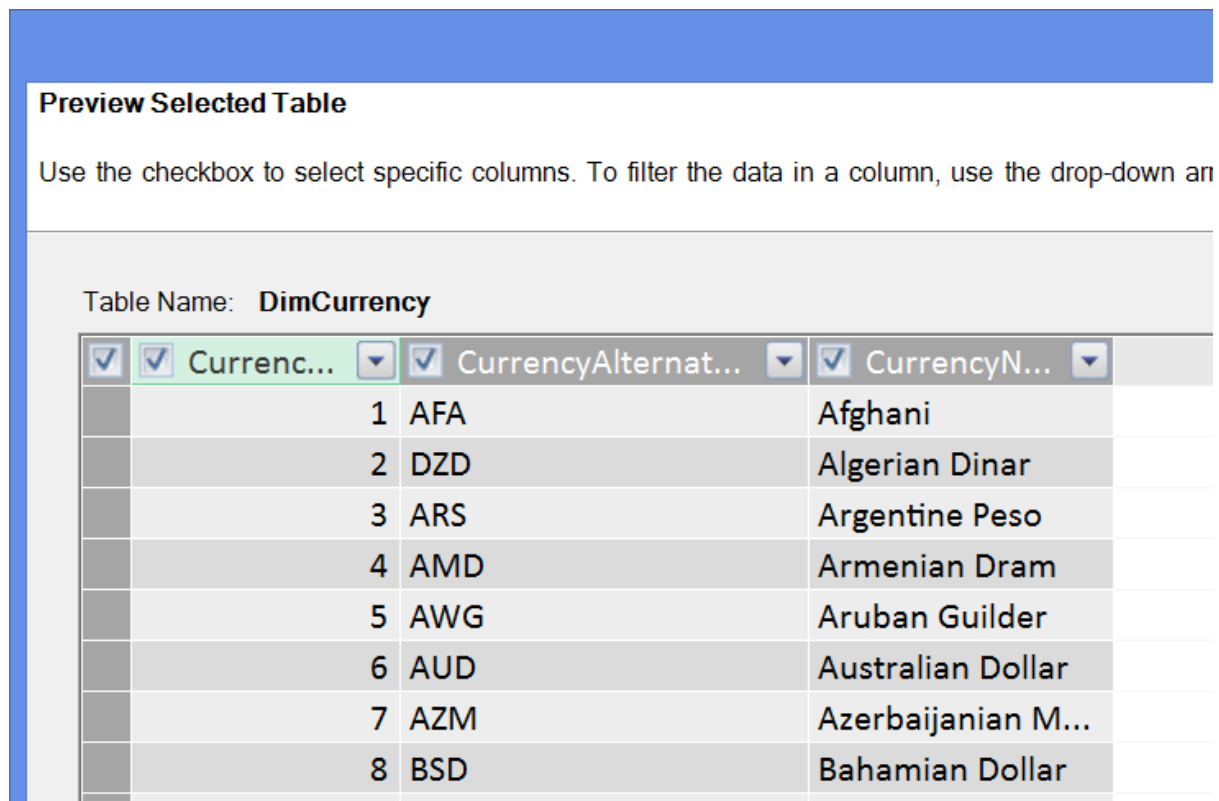


Figure 12: Table Import Preview window

This screen gives you a sample of the data, but also gives you the ability to select or deselect columns from import by using the checkbox by the column name. Further, you can apply a filter by clicking the dropdown arrow to the right of the column name. This is data-type sensitive—like a filter for an Excel Table—so numeric filters can be ranges, whereas text filters can be “Begins with...” or “Contains...”.

The following figure illustrates how this is displayed after applying a filter:

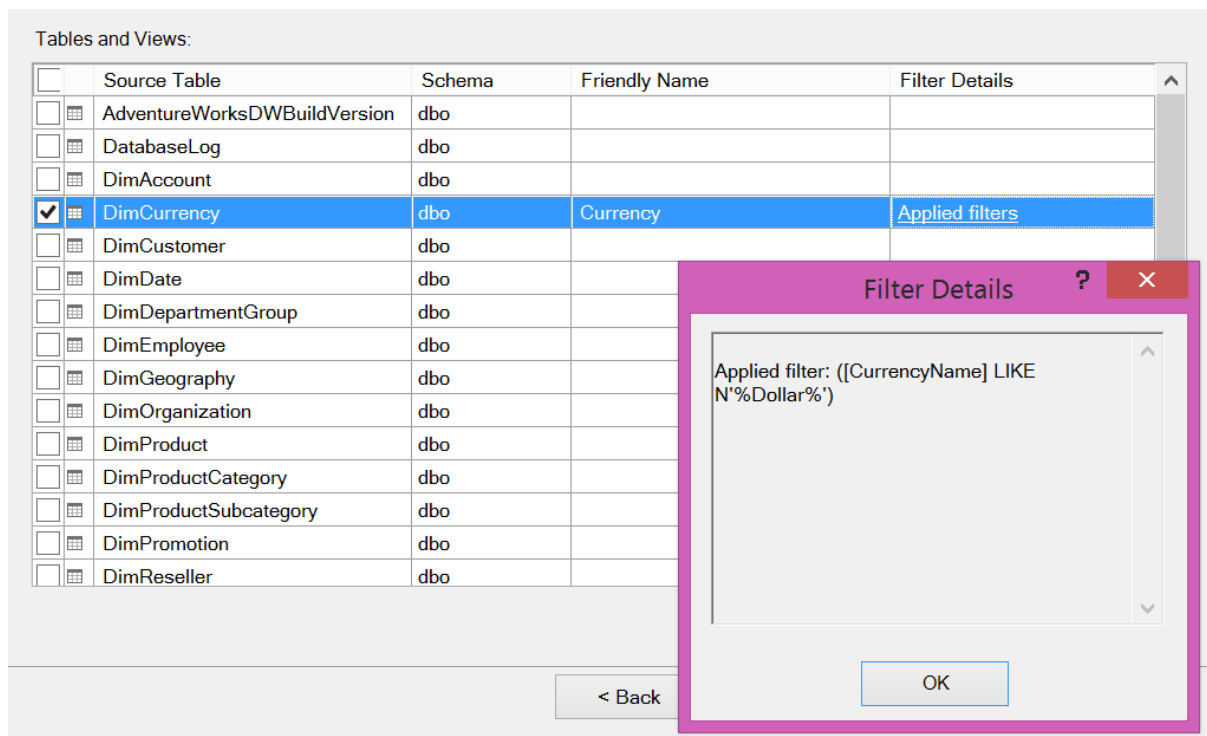


Figure 13: Table Import window with Friendly Name column and Filter Details pop-up

In the Filter Details column, a link appears. Clicking on the link launches a pop-up that shows the filter that is applied.

You can also see that under the Friendly Name column, the default name has been overwritten so the object will have a more business-friendly name in the data model. This renaming can always be [done later](#) in the Data Model Designer if you forget to do it during import, but to save yourself pain later (as name changes don't propagate to your formulas automatically), it's advisable to do it now.

Restricting the amount of data you pull in can be important for bigger sources because even though PowerPivot does offer impressive compression, you do have a finite size of workbook that limits how much data you can pull in. Later on in the book, the section in which we discuss [how compression works](#) will give you the power to work out what columns can be removed to help minimize the size of your workbook. Row filters will make more sense in terms of the context of the data you are looking at—there is no point loading 15 years' worth of sales data, for example, if you are only looking at the last quarter's performance.

The other item to explore is the Select Related Tables button. You may be thinking that PowerPivot can magically detect data that is connected with the table you selected, but unfortunately it can't. What it does is for relational databases: if foreign keys have been coded in the database so that there is a strict relationship between objects (e.g. a fact table and the Date dimension), it can detect these coded relationships. If, like in many data warehouse systems, these haven't been put in place (as they are often difficult to maintain), you will have to work out what tables are related by yourself.

## Using external data from the Windows Azure Marketplace

Another option for external data is Microsoft's market for datasets—Windows Azure Marketplace. In the Get External Data section from the Home tab of the ribbon in the PowerPivot Data Model Designer, there is an option for From Data Service. Selecting the **From Windows Azure Marketplace** option on the dropdown launches a marketplace browser.

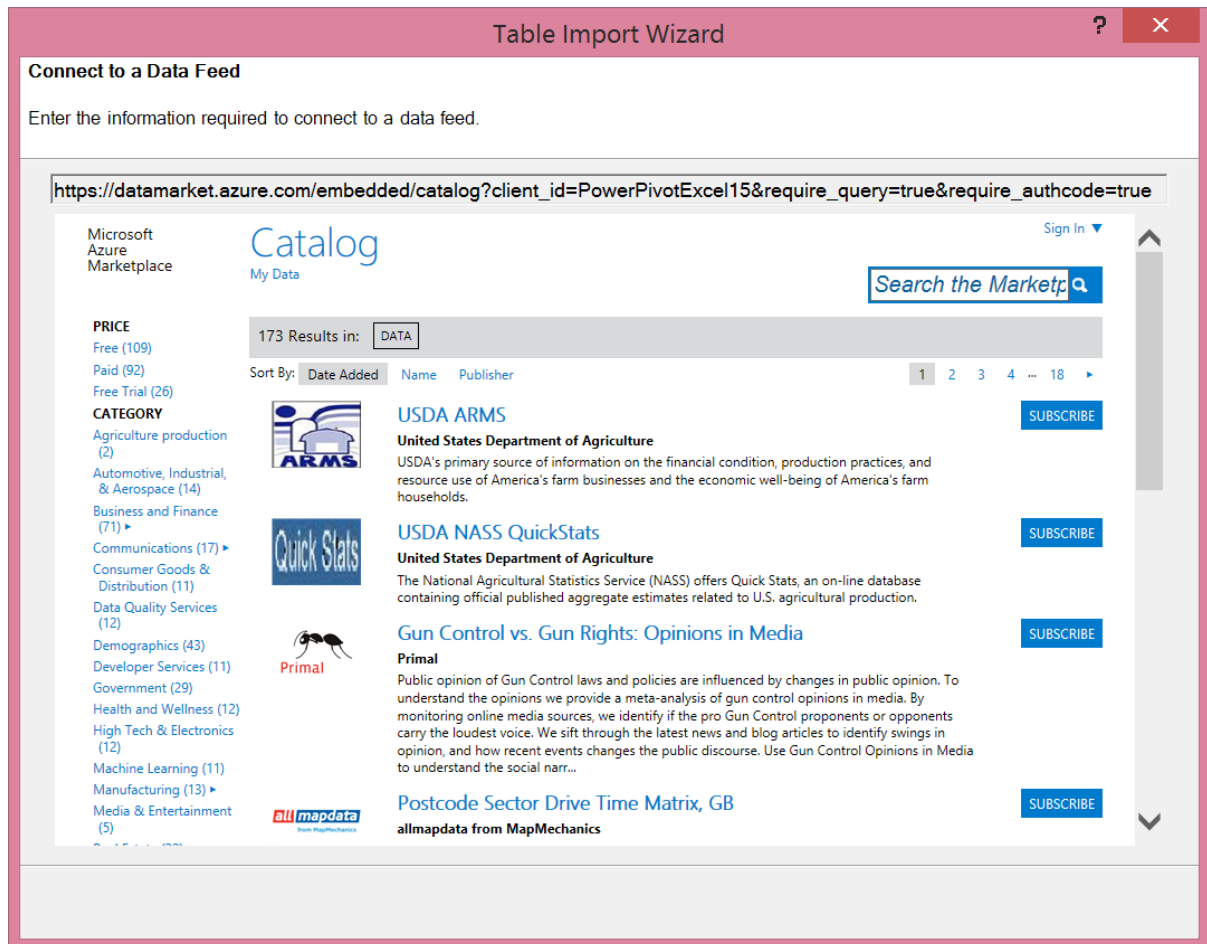


Figure 14: The Windows Azure Marketplace browser

This browser includes a range of interesting—and often free—datasets to bring in to enhance your analysis. You do need to have a Windows Live account to access the service.

In this example we will pull in Boyan Penev's<sup>4</sup> very helpful DateStream dataset, which presents various permutations of date data for slicing up data. We search for "DateStream" and get this result:

---

<sup>4</sup> Boyan Penev's blog: <http://www.bp-msbi.com/>

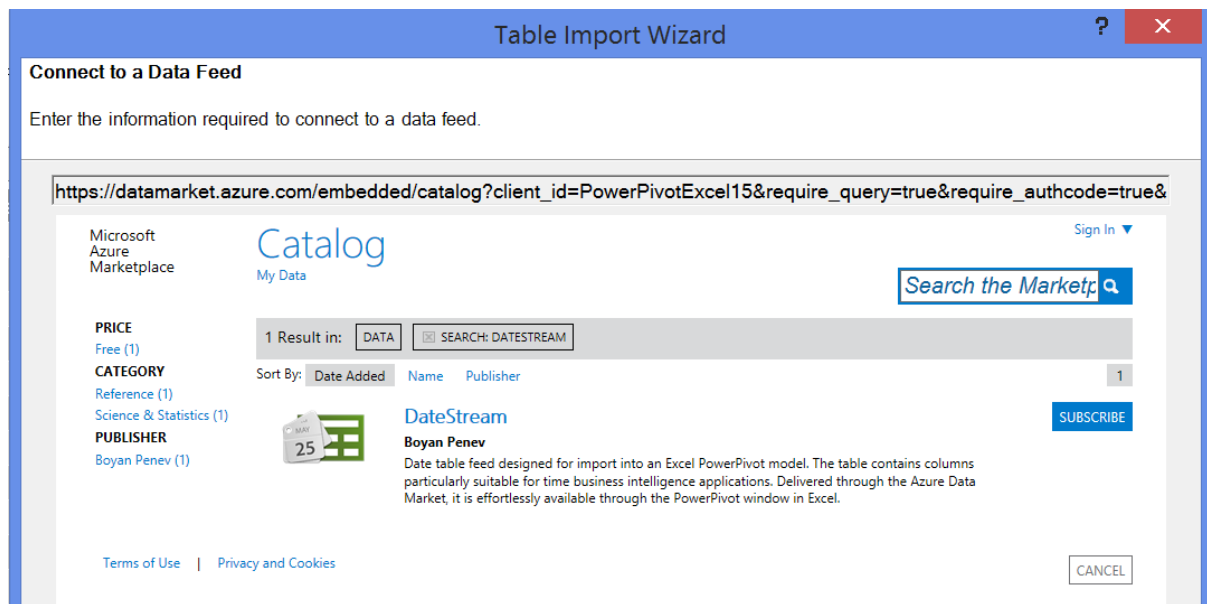


Figure 15: Table Import Wizard - Subscribing to a data set

Click **Subscribe** to open another window where we can select fields and apply very limited filtering:

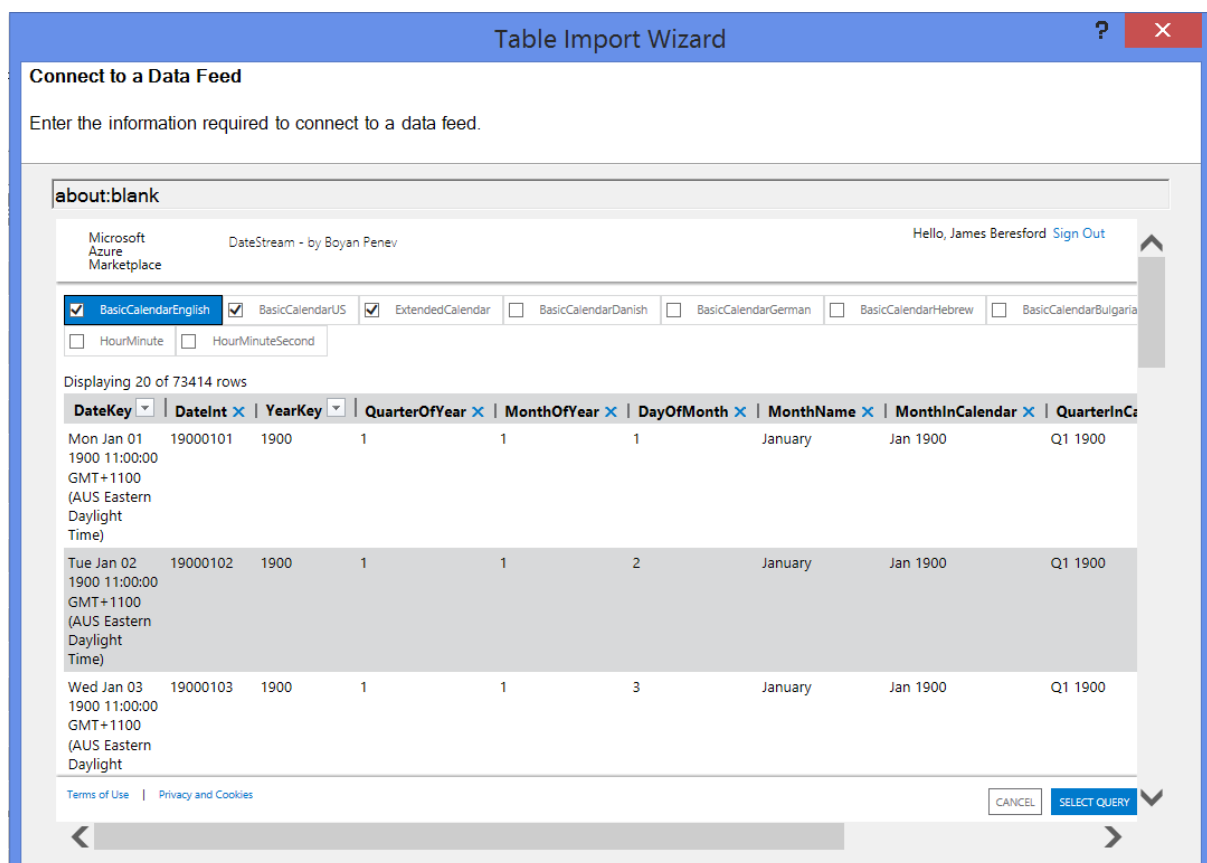


Figure 16: Table Import Wizard - Selecting elements of a data set

We will just unselect some of the non-English language calendars, then click **Select Query**. In the next screen we just name the connection:

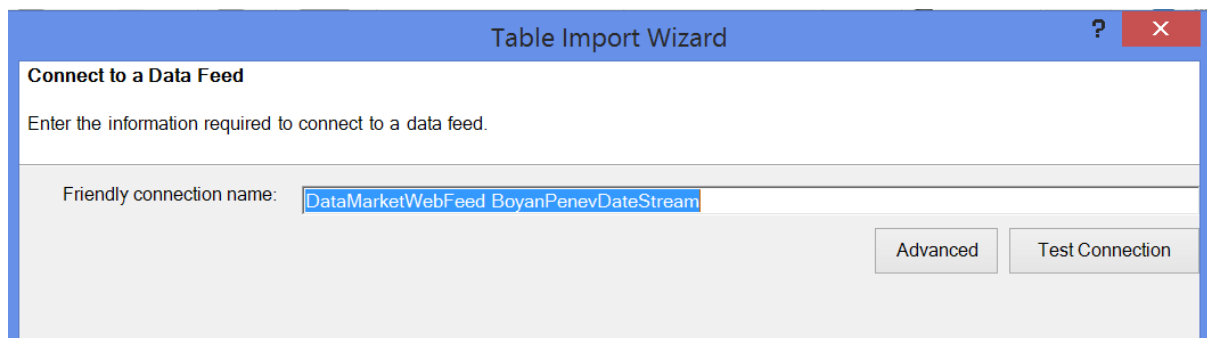


Figure 17: Table Import Wizard - naming the connection

Click **Next** to move to the next screen, where we can select which of the tables we want to import:



Figure 18: Table Import Wizard - selecting tables

Select **BasicCalendarEnglish** and click **Next**, and the import begins:

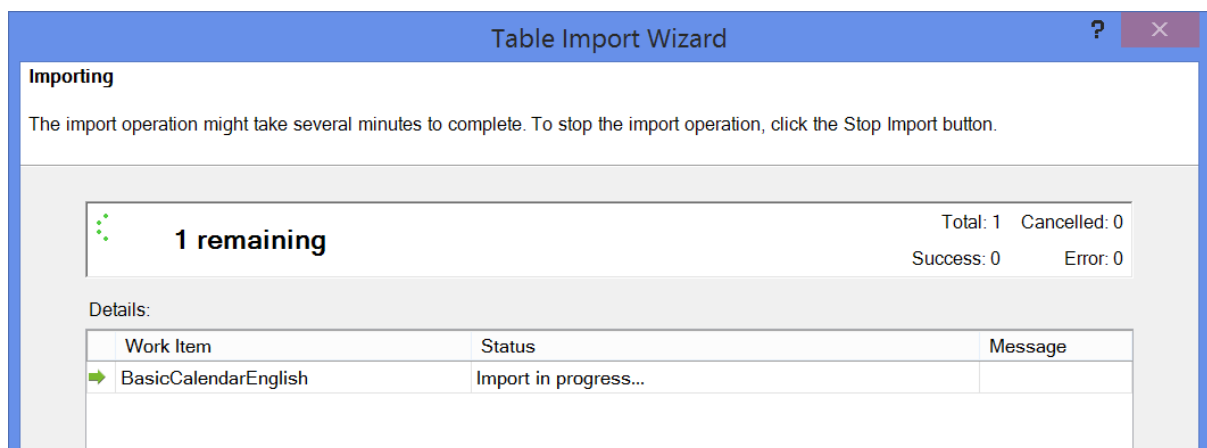


Figure 19: Table Import Wizard - import progress

When complete, we get a new tab in our data model with the data from the Azure Marketplace available to integrate into our model.

## Other supported data sources

There are many other supported data sources, which are available in the Get External Data section of the ribbon via the From Other Sources option.

Supported formats are:

1. Windows SQL Azure
2. Microsoft SQL Server Parallel Data Warehouse
3. Oracle
4. Teradata
5. Sybase
6. Informix
7. IBM/DB2
8. OLEDB/ODBC
9. Microsoft SQL Server Reporting Services
10. URL based Data Feeds
11. Text Feeds

While we won't go into detail about connecting to each of these here, it is clear there is a wide range of data sources available to pull into our data models to support our analysis—all without the need for deep technical knowledge and/or complicated ETL processes.

## Data types

PowerPivot keeps things simple with a limited range of supported data types<sup>5</sup>. These are:

1. Whole Number
2. Decimal Number
3. TRUE/FALSE (aka Boolean)
4. Text
5. Date

---

<sup>5</sup> Data Types Supported in PowerPivot Workbooks: <http://msdn.microsoft.com/en-us/library/gg413463.aspx>



## 6. Currency

When importing your data into PowerPivot, it automatically maps the source data types to the target types. This simplification removes some of the common problems when working with disparate systems with incompatible data types. It also keeps expected calculation results simpler and formatting easier to work with.

One thing to be aware of is that the different types of data compress at different rates. The [xVelocity Deep Dive](#) section towards the end of this book will help you understand why, but the below table indicates which will compress better than others so you can guess at the culprits for hogging space.

Data Type	Compression Rate	Comments
Whole Number	Good	Best compression of the numeric types
Decimal Number	Poor	Decimal numbers do not compress well
TRUE/FALSE	Excellent	Takes up a tiny amount of space
Text	Good	The size of text is not significant
Date	Good	Becomes Poor if it has a time component
Currency	Poor	Marginally Better than Decimal

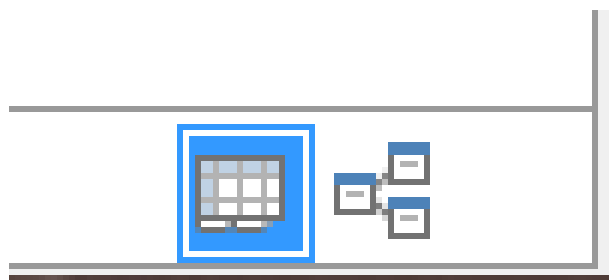
*Figure 20: Data type compression rough guide*

## Hierarchies

Now that we have our date data in our model, we can create a hierarchy to structure how we explore the data. A hierarchy is a logical way of drilling down into a finer grain of detail from a higher level. A date hierarchy is very common when analyzing data. Starting with the Year, we can drill down to Quarter, Month, Week, and Day—and potentially even deeper—though for practical reasons time is usually modeled separately, as the Date dimension gets too big to work with if you include time as well.

To create a hierarchy we first need to look at the data model in a slightly different way. So far we have only been looking at the data model in the Grid view—the Excel-like way of exploring tables and rows of data.

Down at the bottom of the right-hand side of the model is a small pair of icons:



*Figure 21: Switching between Grid and Diagram View*

These icons allow you to switch between the Grid view and the Diagram View. Click on **Diagram View** to open a view of the model like this:

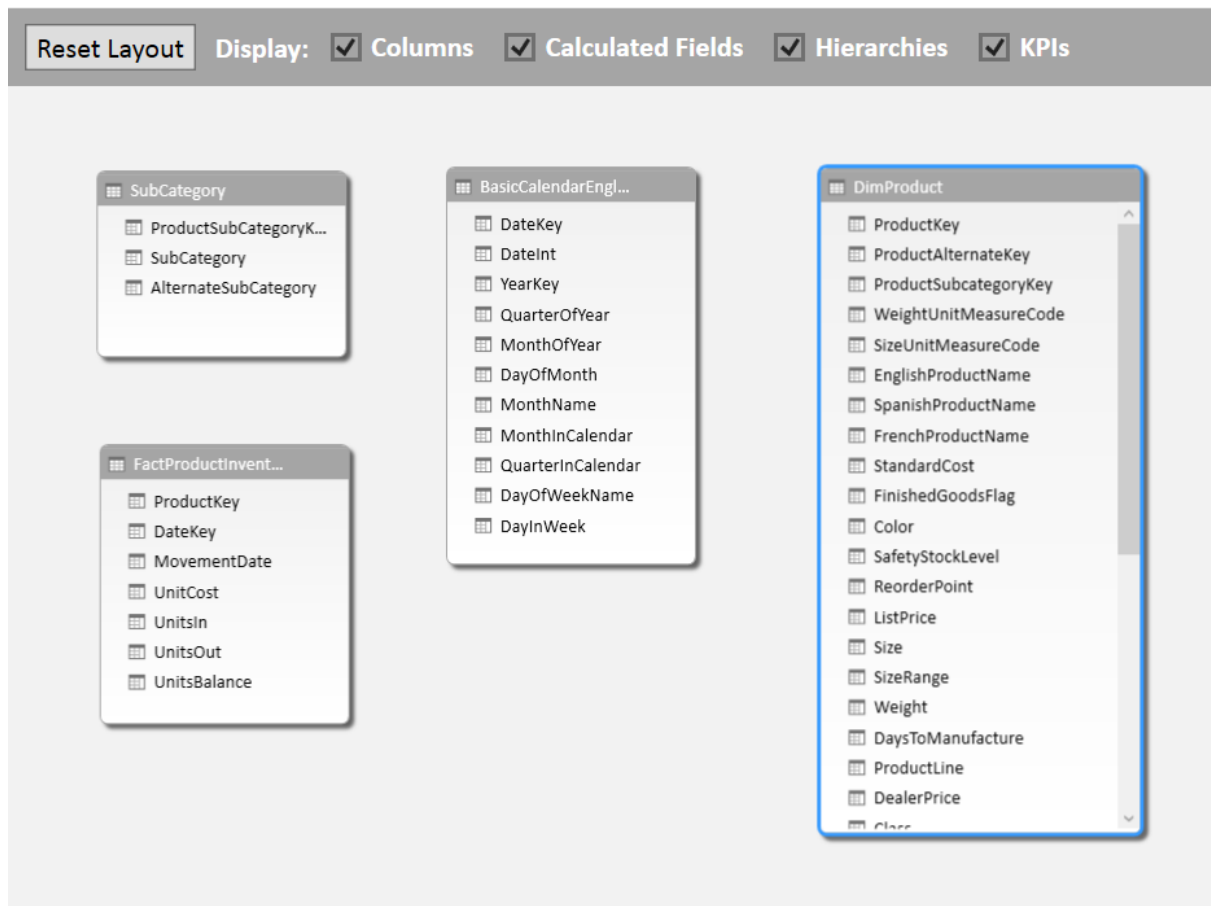


Figure 22: The Diagram view

The Diagram view shows tables and their relationships to each other ([more on this later](#)). In the Diagram view, we can also add hierarchies.

To create a hierarchy in the Date dimension, right-click on the highest intended level of the hierarchy, in this case **YearKey**, and choose **Create Hierarchy**.

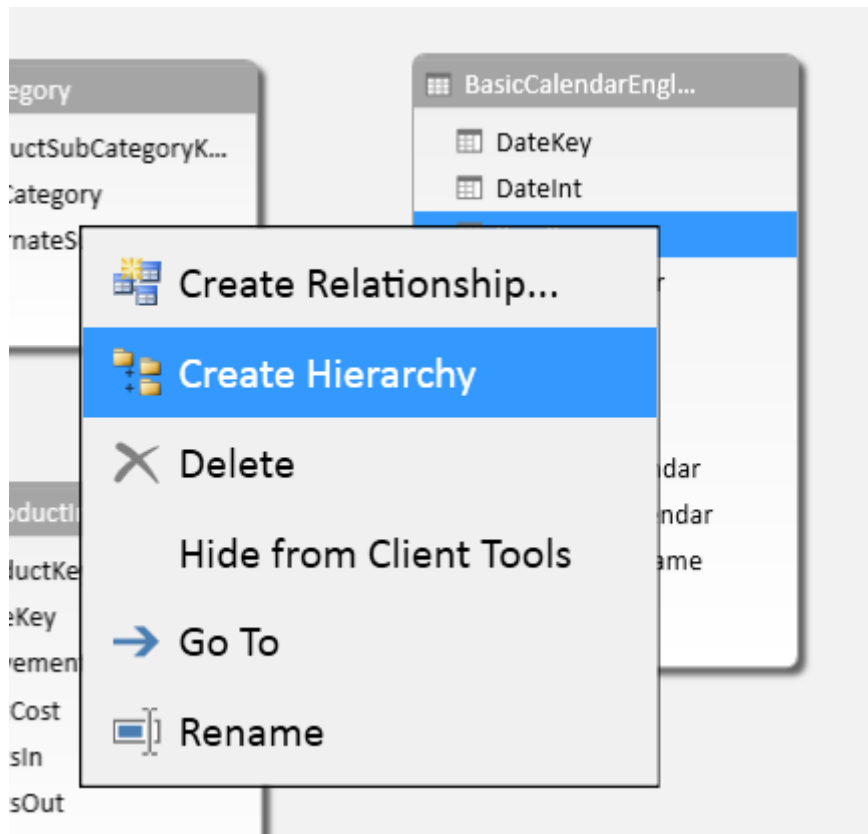


Figure 23: Creating a hierarchy

A new hierarchy is created with a default name:

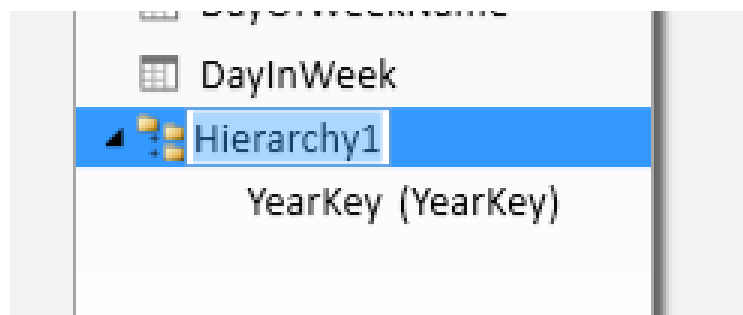


Figure 24: Default hierarchy name

We can change this to something more meaningful, like **Calendar**. We can also rename the levels—double-click on the name to overwrite the field name. The source name is shown in brackets after the field name.

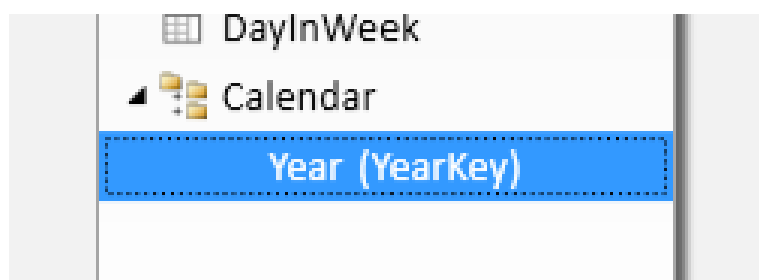


Figure 25: Renaming hierarchies and levels

A single-level hierarchy is not much use, so we need to add levels in. We can just drag fields in below the top level we created, and rename them as we go.

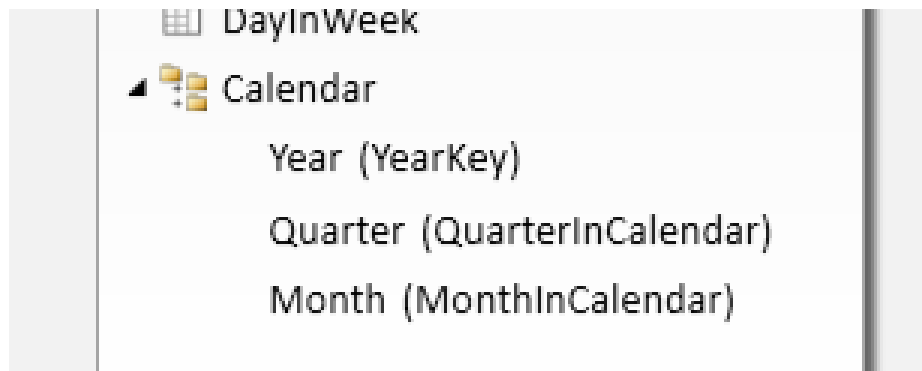


Figure 26: Adding levels to the hierarchy

The practical impact of this is more easily understood if we see what this looks like in Excel.

From the Home tab of the ribbon in the PowerPivot Data Model, click the **Pivot Table** option and choose to create a pivot table in a new worksheet.

On the right-hand side of the screen, the PivotTable field list will appear, just as for a normal pivot table. However, what you see is not all the columns from your source table, but the tables in your PowerPivot data model:

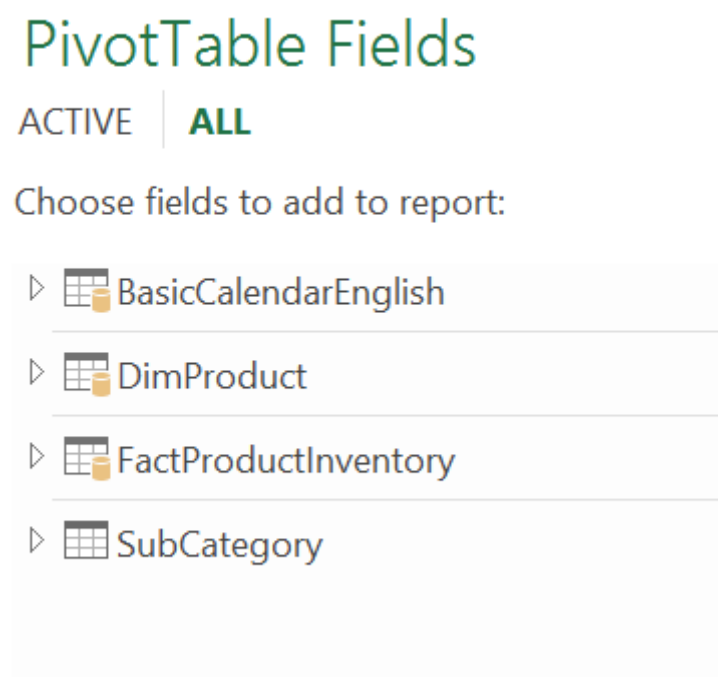


Figure 27: Browsing the model in a pivot table

To explore our newly created hierarchy, expand **BasicCalendarEnglish** and click the **Calendar** checkbox. If you expand the Calendar node, you'll see underneath the levels within the hierarchy that we just created.

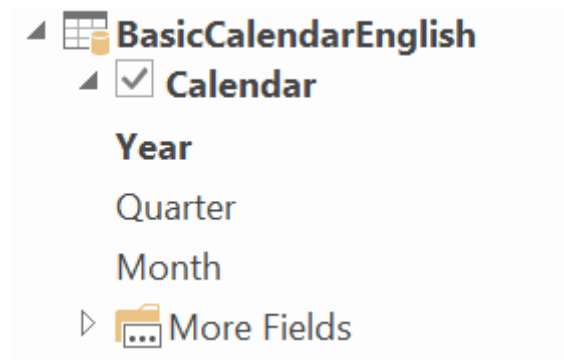


Figure 28: Exploring the hierarchy in a pivot table

This automatically puts the hierarchy into the workbook. We can expand a few nodes in the tree to see how our hierarchy looks:

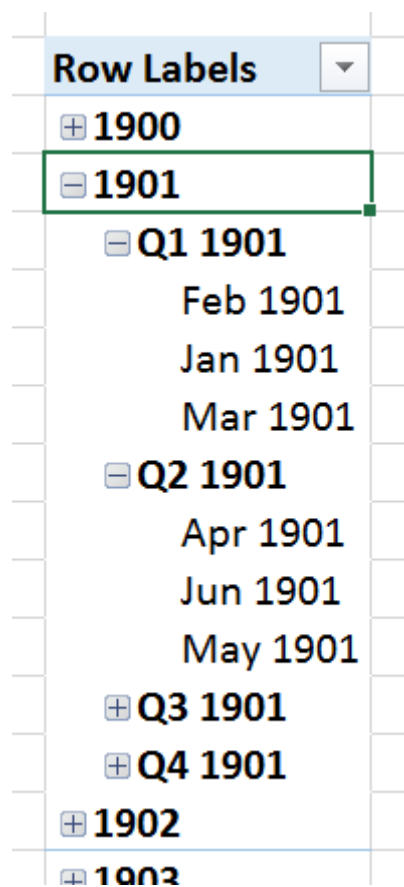


Figure 29: Viewing the hierarchy in Excel

A hierarchy on its own is nice to have, but it only becomes really useful if we use it to slice and dice our data. However, if we were to drag a value from FactProductInventory to analyze, such as UnitsBalance, we would see this in the pivot table:

Row Labels	Sum of UnitsBalance
+ 1900	333869496
- 1901	333869496
- Q1 1901	333869496
Feb 1901	333869496
Jan 1901	333869496
Mar 1901	333869496
+ Q2 1901	333869496
+ Q3 1901	333869496
- Q4 1901	333869496

Figure 30: Analyzing without relationships

And this warning would appear in the Pivot Table Fields window:

Relationships between tables may be needed.

CREATE...



Figure 31: “Relationships may be needed” warning

What is happening is that while FactProductInventory has a date element, PowerPivot hasn't been told that the BasicCalendarEnglish table relates to FactProductInventory by the date keys. As it doesn't know how to tie these pieces of data together, it just adds up all the values for UnitsBalance for the whole of the table for each level you look at in the hierarchy. As an interesting side note, you might notice that the levels of the hierarchy don't total up (i.e. Year is not the sum of Quarters); this is because at each level of the hierarchy you look at, it tries to calculate the right value. But because there is no relationship, every level of the hierarchy ends up returning the same value.

The logical next step in building our model then is to create a relationship so our analysis becomes more meaningful.

## Relationships

One of the most powerful features of PowerPivot is its ability to formally relate data between different tables with a simple drag-and-drop operation, and subsequently pick up data from one table in the context of another. A simple example of this might be using your Date hierarchy in the date table to filter the Product Inventory data in its table.

To do this in basic Excel involves sorting data and doing VLOOKUP formulas to get the related data. Too many VLOOKUPs in a workbook start slowing it down pretty quickly, so you wouldn't want too many of them, and certainly not against millions of rows of data.

For PowerPivot, this is bread and butter stuff, and you'll be able to create relationships between your data that enable powerful slicing and dicing with no need for complex formulas—often without the need for any formulas at all<sup>6</sup>.

## Creating relationships manually

Creating relationships is, as stated above, a simple drag-and-drop action. There are only a few restrictions that you need to bear in mind when creating a relationship:

- You can only build a relationship from one column to another (i.e. you cannot create a relationship two columns involved unless you combine them together first to make a single column).
- The columns you relate must be of a compatible [Data Type](#).
- For performance reasons, Whole Number to Whole Number are preferable.
- The values in the column on one side of the relationship **must be unique**.
- In data modeling terms, this is a one-to-many relationship
- Advanced workarounds exist for this. If you encounter the need for these, the White Paper “The Many-to-Many Revolution 2.0” by Marco Russo and Alberto Ferrari is recommended reading<sup>7</sup>.

With those factors in mind, we can create a simple relationship between our Date table and our Product Inventory Fact.

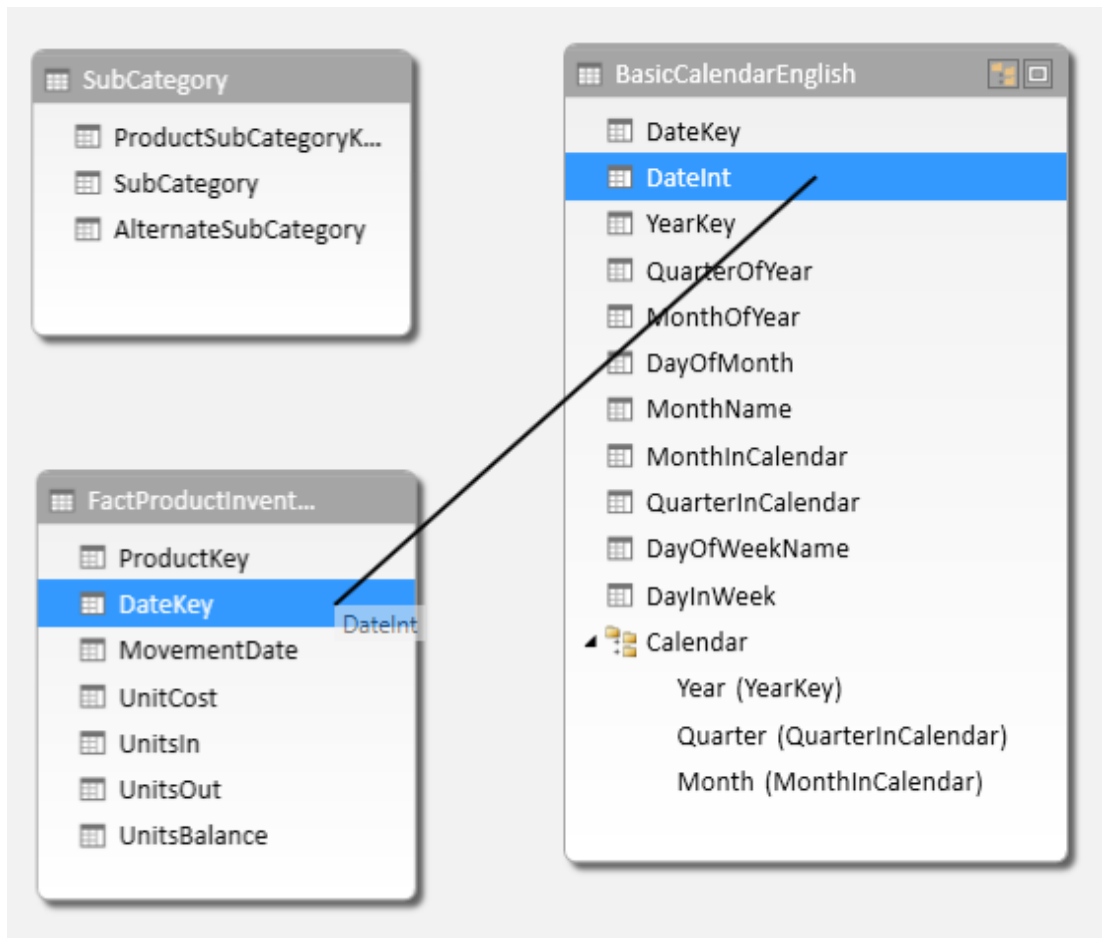
In Diagram View, click on the **DateInt** column in **BasicCalendarEnglish** to highlight it, and drag the cursor over to the **DateKey** column in **FactProductInventory**.

---

<sup>6</sup> Relationships Overview: <http://msdn.microsoft.com/en-us/library/gg399148%28v=sql.110%29.aspx>

<sup>7</sup> The Many-to-Many Revolution 2.0: <http://www.sqlbi.com/articles/many2many/>





*Figure 32: Creating a relationship manually*

When you release the mouse button, the relationship has been formed:

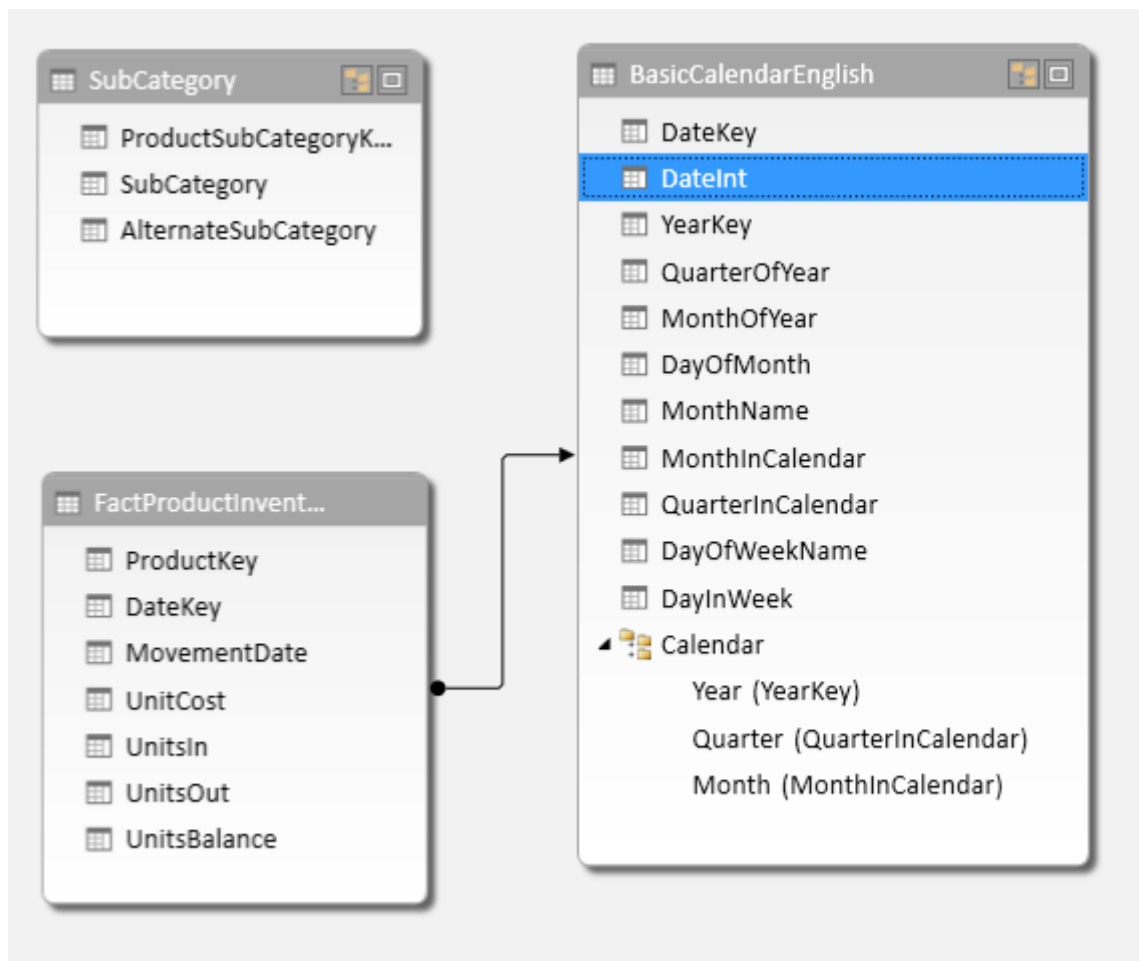


Figure 33: A relationship

Note that the arrow points from the many side of the relationship, FactProductInventory, to the unique side of the relationship, BasicCalendarEnglish.

That is all you need to establish the relationship. To see it in action, we can go back to our pivot table we created earlier— it will have updated automatically. Be careful about clicking Refresh, as that doesn't refresh the pivot table, but actually goes back to refresh all the data from the source.

Row Labels	Sum of UnitsBalance
+ 2005	48540123
- 2006	95598866
- Q1 2006	23617126
Feb 2006	7346456
Jan 2006	8138921
Mar 2006	8131749
+ Q2 2006	23866538
+ Q3 2006	24045927
+ Q4 2006	24069275
+ 2007	94927527
+ 2008	94802980
<b>Grand Total</b>	<b>333869496</b>

Figure 34: Exploring data with a relationship

We can see that a lot has changed now that PowerPivot understands the relationship. There are two big changes. First, the years are now filtered for the years where there is actually data, instead of every year in the calendar hierarchy. Second, each level of the hierarchy has a different number for the Sum of UnitsBalance and the months now add up to the quarter total, and quarters into the year.

To complete our relationships for this example, we will connect up **DimProduct** and **FactProductInventory** using **ProductKey**, and connect **DimProduct** and our Excel table **SubCategory** using **ProductSubCategoryKey**. With a little reorganization, our model should now look like this:

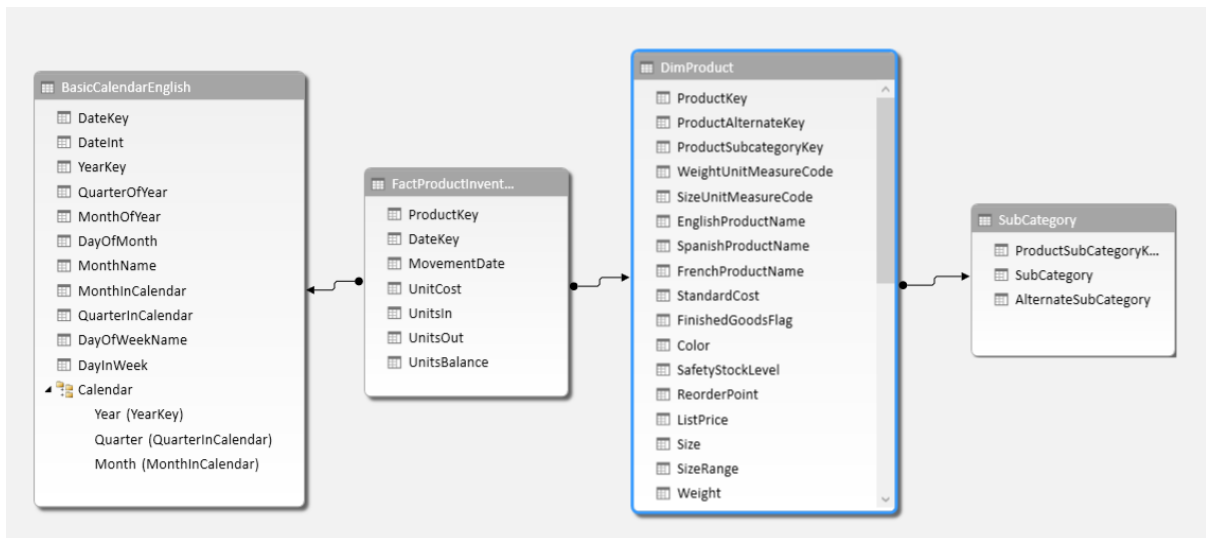


Figure 35: The data model with all relationships

## Calculated columns

Now that we have some connected data we can start extending the model we have with some calculated columns. Calculated columns work pretty much the same way as a calculation in a table in Excel—you enter a formula once, and then it applies to every row in the table. The key difference is that you can't have different values in some rows than others; you can only set one formula, and it will apply to every row without exception—there are no manual overrides for certain rows.

Building a calculated column is a straightforward exercise, but first we have to get familiar with a new formula language called DAX.

## DAX

DAX stands for Data Analysis Expressions. It is a deliberately Excel-like language that makes it easy for advanced Excel users to start creating powerful calculations in PowerPivot models. As mentioned at the start of the book, it's very easy to get started with, but some of the advanced capabilities can stretch the brain quite a bit.

## Creating a simple column

We will get started with DAX by creating a very simple set of columns in the DimProduct table. In Grid view, if we scroll across to the far right-hand side of the model we get the option to **Add Column**.

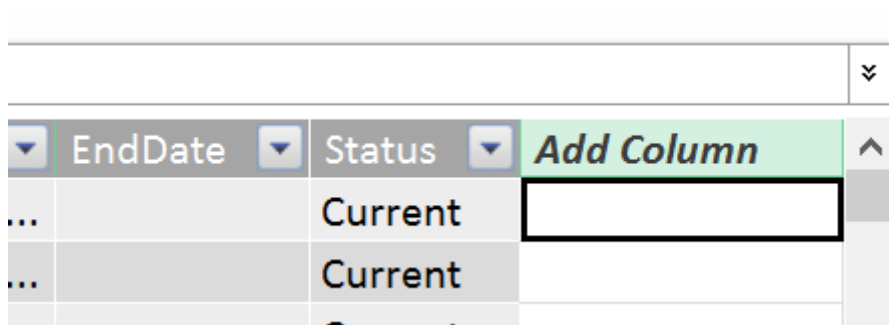


Figure 36: Grid View: Add Column

In our data we have Unit Cost and Weight columns, so we can start by working out the cost of our products per kilo. To create the column, overtype the **Add Column** header in the grid with the new column:

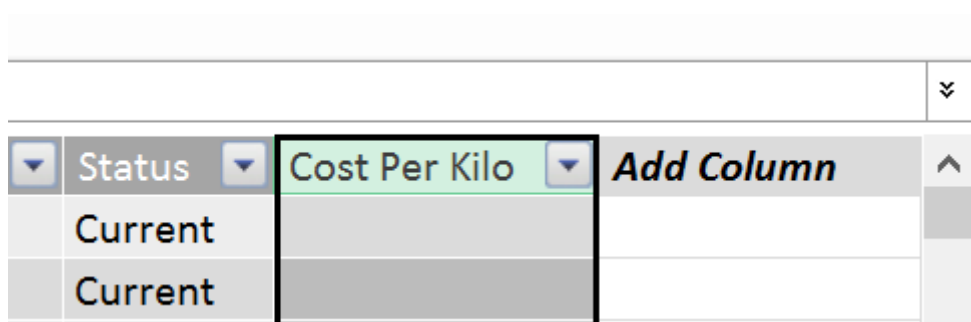


Figure 37: Grid view with an added column

After you click **Enter**, the column is added. You will see the shading of the rows turns to grey to indicate that the column is now in the model, and Add Column reappears so you can add more columns.

Our next step is to create a formula. Over on the left-hand side of the screen we can see the formula bar:

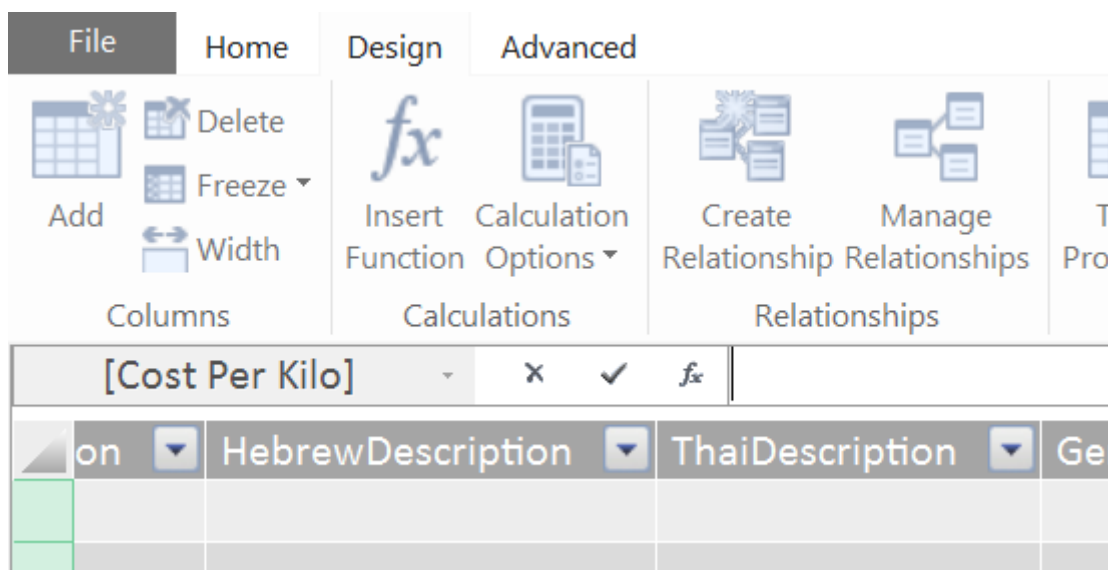


Figure 38: The Formula bar in the data model

To start creating a formula, enter into the formula bar and add an equal sign. Next, enter an opening square bracket [ and a list of fields comes up. When you start typing **StandardCost**, the list will begin to be filtered. You can use the arrow keys to select a value if there are multiple options, and hit **Tab** to select.

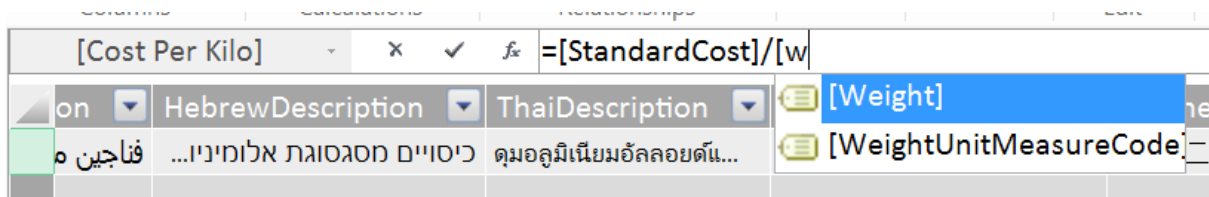


Figure 39: Field prompting in the Formula bar

We can then follow up with a division sign, and then choose the **Weight** field. Hit **Enter** and the formula has been created.

Unfortunately, when we hit **Enter**, this fills the column:

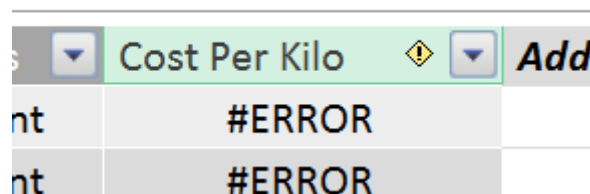


Figure 40: Calculated Field Error

Unlike in Excel where errors are flagged on a row-by-row basis, in PowerPivot if you get one error thrown in the column, the whole column fails to calculate. If we highlight the field a dropdown menu appears at the right-hand side of the cell:

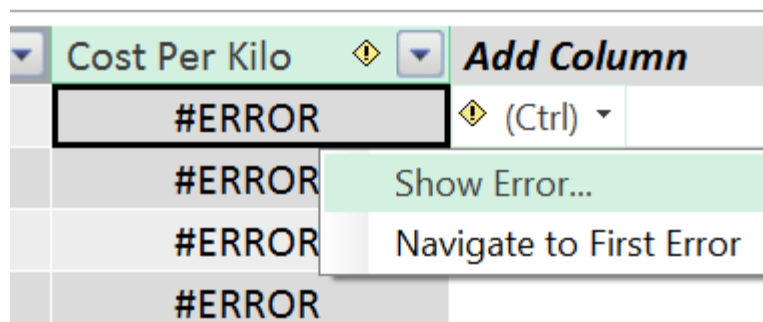


Figure 41: Column Error dropdown

And our error is this:

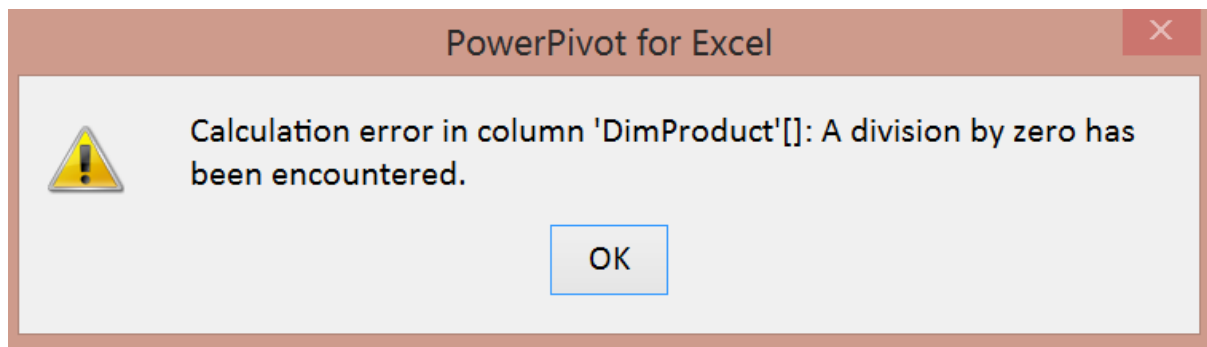


Figure 42: Calculation error message

This should be a simple enough problem to solve. We can quickly look at the data in the Weight column by using the filter on the grid. This is accessed by the drop-down button at the right of the field name, just like in Excel tables:

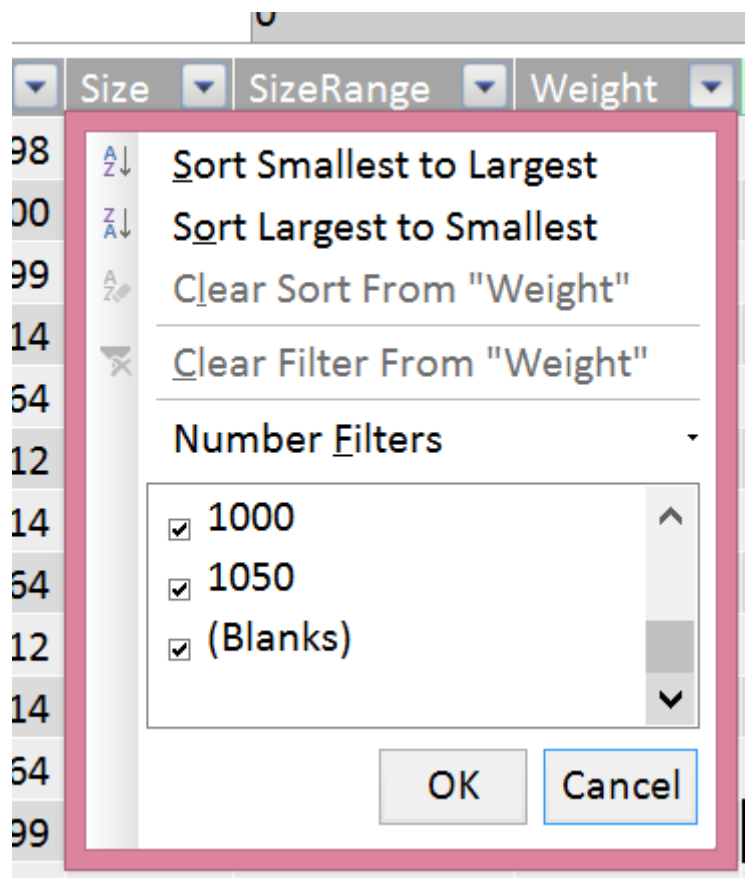


Figure 43: Filtering data in the grid

Scrolling down to the bottom of the list, we see an entry for (Blanks), so it quickly becomes obvious that it isn't populated for every row. We can put in some simple handling using an IF and ISERROR statement just like in Excel:

`=IF(ISERROR([StandardCost]/[Weight]),0,[StandardCost]/[Weight])`

Now, our column calculates without an error whether Weight is populated or not:

Product	Cost Per Kilo	Add
Product 1	\$0.00	
Product 2	\$0.00	
Product 3	\$229.35	
Product 4	\$242.98	
Product 5	\$274.71	

Figure 44: Successful calculation

## Lookup up data from another unrelated table

Exploring our data further, there is actually a column called WeightMeasureCode, which has values of either Blank, LB or G. So our Cost per Kilo calculation isn't all that accurate, as none of the weights are in kilos. We could embed the conversion in a nested IF statement, but there is a much more flexible and powerful option, which shows the real value of PowerPivot.

First, we will add a new table in Excel called WeightConversions:

	A	B
1	WeightUnitMeasureCode	Conversion factor
2	LB	2.2
3	G	0.001

Figure 45: Weight conversions table

We will then add it to our data model.

Then in DimProduct, add a new column called **Weight Conversion**. To keep things simple, first we will show how to look up the conversion factor with the LOOKUPVALUE<sup>8</sup> formula like this:

```
=LOOKUPVALUE(WeightConversions[Conversion
factor],WeightConversions[WeightUnitMeasure
Code],[WeightUnitMeasure
Code])
```

<sup>8</sup> LOOKUPVALUE function reference: <http://msdn.microsoft.com/en-us/library/gg492170.aspx>



This is equivalent to a VLOOKUP in Excel, but a lot faster and not tied to a fixed data range. If at any point new Weight Measures came in, you could simply add a row to your table in Excel and update the model, and the formula would pick it up.

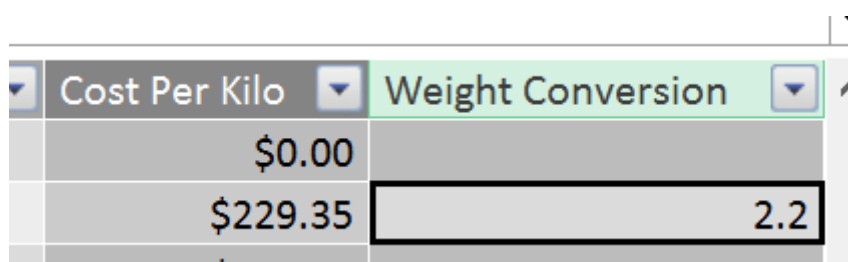
LOOKUPVALUE needs three arguments:

1. The column you want to return—WeightConversions[Conversion factor]
2. The column you are looking up in—WeightConversions[WeightUnitMeasureCode]
3. The column you are providing the value to look up from—[WeightUnitMeasureCode]

There are a few things to note here. First is the addition of a table name into our column syntax. In our first formula everything was in the syntax [Field Name]. Now we are referencing another tables content, so we need to specifically call that out as Table Name[Field Name].

Second is the LOOKUPVALUE formula expects that there is only one value that could be returned per lookup value—it forgives multiple rows in the lookup with the same value, but not different values.

So now we have a lookup for our weight conversion that gives these results:



Cost Per Kilo	Weight Conversion
\$0.00	
\$229.35	2.2

Figure 46: Weight Conversion LOOKUPVALUE result

We can then correct our Cost per Kilo calculation by including this Weight Conversion in our formula:

```
=IF(ISBLANK([WeightUnitMeasureCode]),0,[StandardCost]/([Weight]*[Weight Conversion]))
```

Note also the removal of the “ISERROR” catch all and replacing it with an “ISBLANK”<sup>9</sup> assessing if we have a value in WeightUnitMeasureCode.

Replicating the Weight Conversion value in a column is bit inefficient, albeit a bit easier to understand, so lets shift the formula of that into our Cost Per Kilo formula:

```
=IF(ISBLANK([WeightUnitMeasureCode]),0,[StandardCost]/([Weight]*(LOOKUPVALUE(WeightConversions[Conversion Factor], WeightConversions[WeightUnitMeasureCode],[WeightUnitMeasureCode]))))
```

<sup>9</sup> ISBLANK function reference: <http://msdn.microsoft.com/en-us/library/ee634204.aspx>

So finally we get accurate values in our Cost Per Kilo calculation using some data from another related table:

Status	Cost Per Kilo	A
Current	\$82.47	
Current	\$128.22	
Current	\$194.38	

Figure 47: Revised Cost per Kilo results

## Lookup up data from another related table

We can simplify this process by establishing a relationship between the two tables. In Diagram mode, create a relationship between the new table and the DimProduct table using WeightUnitMeasureCode.

Now, instead of using LOOKUPVALUE, we can use the RELATED<sup>10</sup> function, which uses the relationships in the data model to get related information.

The equivalent to the LOOKUPVALUE formula is this:

`=RELATED(WeightConversions[Conversion factor])`

Which is much simpler—we only need to know which column to retrieve. PowerPivot, being aware of the relationships, can then do all the thinking about how to connect them based on its knowledge of how your data is structured.

So now the Cost per Kilo function can be simplified to:

`=IF(ISBLANK([WeightUnitMeasureCode]),0,[StandardCost]/([Weight]*RELATED(WeightConversions[Conversion factor])))`

The two functions exist because there may not always be a navigable relationship between the table you want to fetch information from and the table you want to reference the lookup value in. It is generally preferable to use the RELATED function, as it is simpler to code and usually less computationally expensive.

## Calculations on data in another table

We have just explored being able to pick up a single value from another table—but PowerPivot can do much more than that. Through the power of relationships, you have full access to the data in related tables.

This is illustrated in the following figure, where the single row for key “B” in the left-hand side table is related to multiple rows in the right-hand table:

<sup>10</sup> RELATED function reference: <http://msdn.microsoft.com/en-us/library/ee634202.aspx>

Table Key	Attribute	Related Key	Attribute	Value
A	Orang Utan	A	North	100
B	Chimpanzee	A	South	200
C	Gorilla	B	East	23
D	Capuchin	B	Central	4556
		B	West	6657
		B	North	785
		C	Central	87

Figure 48: Data in a related table

The immediate challenge is that if you want to calculate a result, you can have only a single value returned. This means one of the key considerations when using related data is that if you want to retrieve data from the Many side of the One-to-Many relationship, something will need to be done to ensure only a single value is returned.

Typically this is done by using an aggregation function, such as SUM or AVERAGE. We can explore this in our model by getting some values from FactProductInventory and capturing them in DimProduct.

Let's start with a simple SUM<sup>11</sup> operation. In DimProduct we can add another column, this time called **Total Units In**. We can then enter the formula:

`=SUM(FactProductInventory[UnitsIn])`

However, we get this result:

▼	Cost Per Kilo ▼	Tot... ▼	A
	\$0.00	270969	
	\$0.00	270969	
	\$0.00	270969	
	\$0.00	270969	
	\$0.00	270969	

Figure 49: SUM without context

This is due to a problem called [Context](#). We will explore this topic more later, but the key problem is that the SUM expression doesn't have any filters applied, so it simply adds every row in the FactProductInventory table.

<sup>11</sup> SUM function reference: <http://msdn.microsoft.com/en-us/library/ee634387.aspx>

If we turn this Calculated Column into a [Calculation](#) at the bottom of the data grid, this problem will go away as the relationships start having an effect again—but we will look at this later, as doing it this way will teach us some lessons about context.

If we want to fix it at the row level, we need to give the SUM expression an idea of how to filter the rows in FactProductInventory. We can use a variation of the SUM expression, SUMX<sup>12</sup>. This allows you to sum over an expression.

This is one of the moments we go for the deep end. The formula to get the result we need is:

```
=SUMX(FILTER(FactProductInventory,FactProductInventory[ProductKey]
=EARLIER([ProductKey])),FactProductInventory[UnitsIn])
```

So as well as introducing SUMX, we are bringing in a very important function, FILTER<sup>13</sup>, and also the EARLIER<sup>14</sup> function, which doesn't get used as often. Let's step through building this up.

SUMX takes two arguments: a table of data, and an expression to sum. So in our first pass, we could try just using FactProductInventory as the table and UnitsIn as the expression:

```
=SUMX(FactProductInventory,FactProductInventory[UnitsIn])
```

However, that just gives us the same result as SUM:

▼	Cost Per Kilo ▼	Tot... ▼	A
	\$0.00	270969	
	\$0.00	270969	
	\$0.00	270969	
	\$0.00	270969	
	\$0.00	270969	

Figure 50: SUMX without context

This is because all we have really done is replace SUM with SUMX. All we did by changing to SUMX was explicitly specify what table UnitsIn was in, when SUM worked it out from the relationship.

What we need to do is apply a filter to FactProductInventory. Here we look to the FILTER function. The FILTER function applies a filter to the table of data you are trying to retrieve a result from. So in our example we want to filter FactProductInventory by the current value of ProductKey in DimProduct. However, if we try with this formula:

<sup>12</sup> SUMX function reference: <http://msdn.microsoft.com/en-us/library/ee634959.aspx>

<sup>13</sup> FILTER function reference: <http://msdn.microsoft.com/en-au/library/ee634966.aspx>

<sup>14</sup> EARLIER function reference: <http://msdn.microsoft.com/en-us/library/ee634551.aspx>

`=SUMX(FILTER(FactProductInventory,DimProduct[ProductKey]),FactProductInventory[UnitsIn])`

We still get the same result. The reason for this is that the filter is applying the filter to FactProductInventory using every value for ProductKey in DimProduct. This is one of those places where we have to start thinking differently than we do with Excel. In most Excel formulas, any arguments you provide are at an individual cell level. In PowerPivot it is far more common to input a table of values. This case is a good illustration—specifying DimProduct[ProductKey] is not requesting the current row's value for that field—it is specifying the contents of the whole column.

In the data we have, the effect of this isn't immediately obvious, as every value of ProductKey in FactProductInventory is also in DimProduct. If there were more values for ProductKey in FactProductInventory than in DimProduct, we would have still seen the same number in every row. However, that number would likely be different than we saw for the SUM function, as the result would have been filtered for only those values of ProductKey in FactProductInventory that were also in DimProduct.

So, difficult concepts aside, how do we get the filter to take the current row's value of ProductKey into account? Here we reach for the EARLIER function (we could also use the EARLIEST<sup>15</sup> function, but EARLIER works fine in this case). What this does is grab the Row Context from an earlier pass of the calculation engine<sup>16</sup>—which for our example means the value of ProductKey in DimProduct for the row the calculation is being performed on.

If we revisit our target formula:

`=SUMX(FILTER(FactProductInventory,FactProductInventory[ProductKey]=EARLIER(Product[ProductKey])),FactProductInventory[UnitsIn])`

The highlighted section uses the EARLIER function to set the filter to be where FactProductInventory[ProductKey] is equal to the ProductKey for the row of DimProduct the formula is being evaluated for. We now get a result like this:

Total Units In	
.00	437
.00	1238
.00	1960
.00	991
.00	3346

Figure 51: SUMX with context provided

<sup>15</sup> EARLIEST function reference: <http://msdn.microsoft.com/en-us/library/ee634779.aspx>

<sup>16</sup> For an explanation, see this blog post Row Context and EARLIER(): <http://www.leehbi.com/2012/02/row-context-and-earlier/>

If you still feel like you are at the deep end, don't worry—EARLIER is one of the harder functions to understand, and it is brought up here more to start raising your awareness of Context in PowerPivot.

## Calculations

Row-level calculations are useful, but where PowerPivot really shines is when calculations are applied at the table level. You may have been wondering why, at the lower portion of the Grid view, there were several rows of grey cells. This is the workspace for adding calculations.

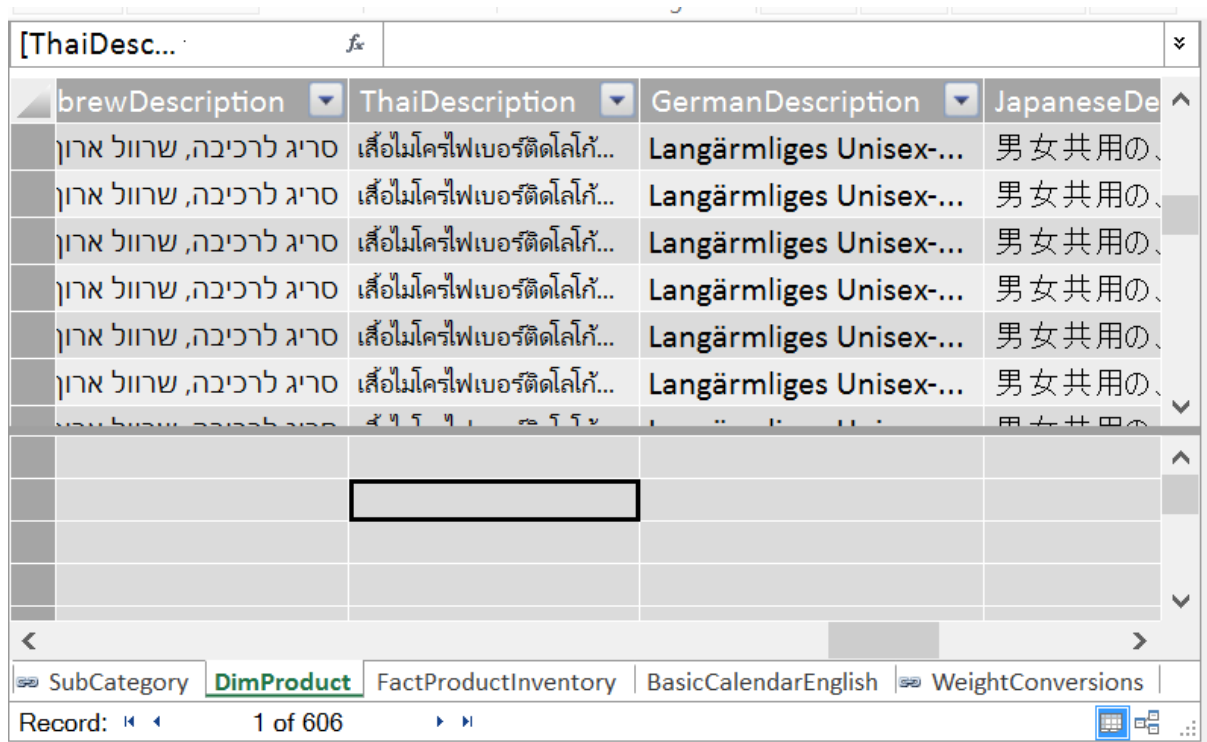


Figure 52: A selected cell in the calculation workspace

In this figure, we have selected a cell. To create a calculation we go to the formula bar, give our calculation a name, and enter the expression. To revisit our earlier calculation, let's SUM the UnitsIn field. Given the sum is applied to FactProductInventory, we'll put it on that table's grid for ease of location. The syntax we need is:

*Total Units In v2:=SUM(FactProductInventory[UnitsIn])*

Note we have to call it "v2" or some variation, as PowerPivot won't allow calculations with the same name as one that already exists. What we see is this:

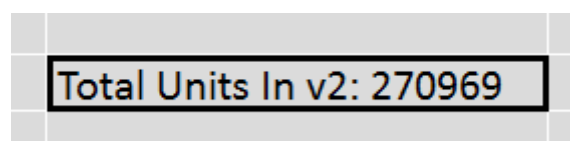


Figure 53: Calculation result

If we now go to Excel, we can see a couple of interesting things. From within the data model, create a pivot table in a new worksheet. In the fields, drag **DimProduct ProductKey** to the rows, and from **FactProductInventory** choose **UnitsIn** and our measure, **Total Units In v2**:

Drag fields between areas below:

<b>FILTERS</b>  	<b>COLUMNS</b> <div>Σ Values ▼</div>
<b>ROWS</b> <div>ProductKey ▼</div>	<b>VALUES</b> <div>Sum of UnitsIn ▼</div> <div>Total Units In v2 ▼</div>

Figure 54: Pivot table settings

What we see in the pivot table is a set of results like this:

Row Labels ▼	Sum of UnitsIn	Total Units In v2
221	1739	1739
222	4343	4343
223	985	985
224	2304	2304
225	5022	5022
226	0	0

Figure 55: Pivot table results

There are a couple of things to pick up on here. First, it seems that our effort in creating our measure is a bit wasted, as PowerPivot seems to add up Units In in the way we want without having created a measure. Second—and more importantly—we haven't had to do any clever calculations to get the right results on a row-by-row basis. We'll look at these next.

## Default aggregations

PowerPivot by default creates a default aggregation calculation on any numeric column it finds. You may have picked this up when building the pivot table if you accidentally clicked the checkbox next to ProductKey instead of dragging it into the Rows area, as it would have automatically added it to the Values section.

This behavior can be changed. On the Advanced tab of the Ribbon in the Data Model, there is an unlabelled section containing an option for Summarize By:

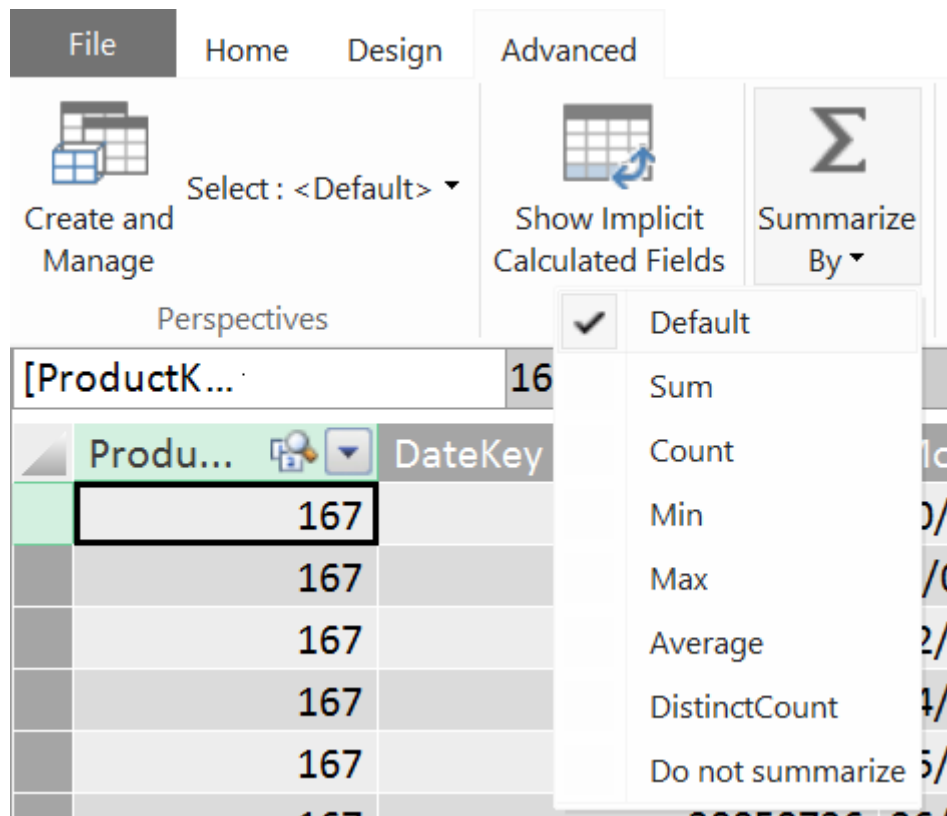


Figure 56: The Summarize By option on the Advanced tab of the data model ribbon

When you click on **Summarize By**, the dropdown menu gives you the following options:

1. Default, which is always Sum
2. Sum
3. Count
4. Min
5. Max
6. Average
7. DistinctCount, a count of unique values
8. Do not summarize

In our case, ProductKey is not actually a meaningful number we can use to sum, average, or otherwise calculate, so it would be a good idea to set it to **Do not summarize**.

With numeric columns, Sum is a handy way to save time in the way of creating calculations.

## AutoSum

Sometimes it is useful to perform multiple operations on a column. Say we were looking at temperatures across time periods—knowing the Min, Max and Average are all useful—though the Sum itself is useless.



In a case like this it would make sense to set the default aggregation to Average. We can then quickly add the Min and Max using the AutoSum option provided. On the Home tab of the Data Model ribbon, to the far right is a section called Calculations, that includes a dropdown for AutoSum:

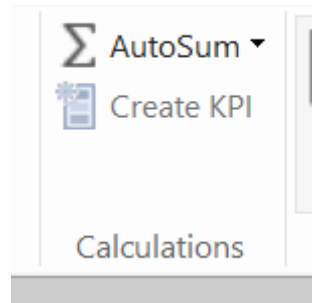


Figure 57: The Calculations section of the Home tab of the data model ribbon

When you have a cell selected in the column you want to AutoSum, choosing a value from this dropdown will automatically add a calculation. If we selected **ProductKey** in the DimProduct table and then selected **Average**, we would see the following measure created:

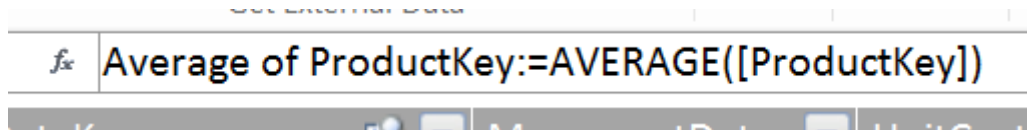


Figure 58: A calculation generated by AutoSum

This isn't going to save you heaps of coding time, but for those who are non-technical it's an easy way to get calculations created.

## Context in calculations

The other big takeaway from creating our calculation is that we didn't have to do anything to get the correct context—the Sum of UnitsIn had the right value per ProductKey without us doing anything particularly clever.

We will revisit this later, but the important thing to be aware of is that for calculations—as opposed to calculated columns—PowerPivot applies the correct filters to the rows of data so you get only the values you are expecting. It understands what context you are viewing the data in based on the relationships you have created.

## Time Intelligence

There's a lot of functionality within DAX for performing powerful calculations, but one of the most important is the capability that Microsoft calls Time Intelligence<sup>17</sup>.

---

<sup>17</sup> Time Intelligence function reference: <http://technet.microsoft.com/en-us/library/ee634763.aspx>

To give you a taste of this, we will look at a couple of DAX functions in this category, PREVIOUSDAY and TOTALMTD. First we need to tell PowerPivot which table should be used as the Date Table on which it will base its Time Intelligence calculations.

## Creating the Date table

We have [already loaded in a Date table](#), but we didn't tell PowerPivot the contents of the table. If we select our **BasicCalendarEnglish** table in the Grid view, and head to the design tab, we can choose **Mark as Date Table** from the dropdown menu.



Figure 59: Mark as Date Table option on the Design tab

This brings up a dialog in which you need to select the date column. This has to be a date data type and contain no duplicates. We will choose the **DateKey** column:

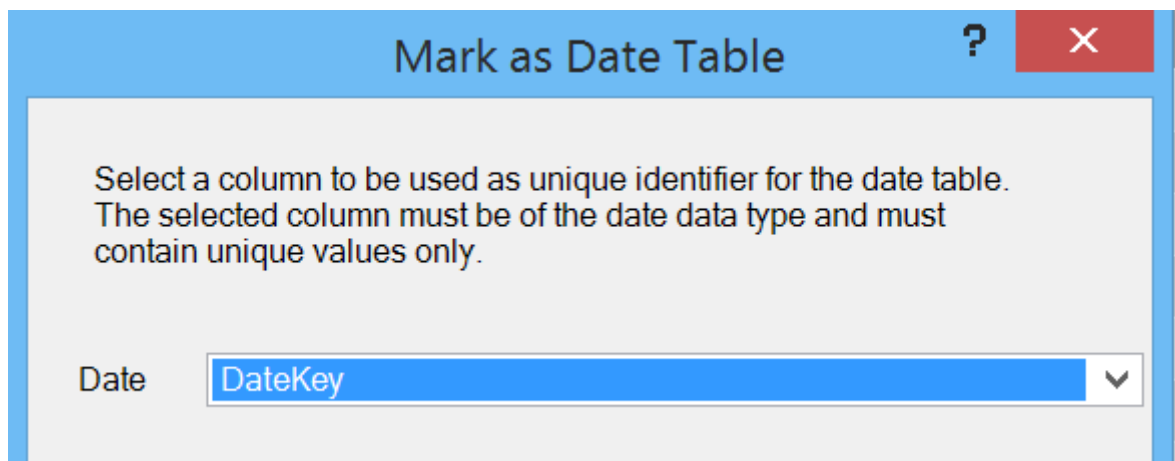


Figure 60: Selecting the Date Column for the date table

PowerPivot now knows this table can be used as a reference when building up Time Intelligence calculations. A constraint we have skipped over is that the date ranges must also be continuous—there must be an entry for every day between the lowest and the highest dates in your date table. Because we have used a reliable external source, this wasn't a problem for us. If you build your own, be sure it contains contiguous dates in the range—otherwise your calculations could return incorrect results.

Now let's look at a couple of functions to understand how time intelligence works.

## PREVIOUSDAY

The PREVIOUSDAY<sup>18</sup> function unsurprisingly returns the previous day in your Date Dimension to the one currently selected. This is useful for calculating daily movements in data.

In our example, we will create a calculation that works out what UnitsIn was on the previous day. To start with, we will just calculate what the previous day was:

*PreviousDay:=PREVIOUSDAY('Date'[DateKey])*

In the calculation area, this shows the result:

<b>PreviousDay: (blank)</b>
-----------------------------

Figure 61: PREVIOUSDAY calculation result

This is not a concern however, and once again is down to context. At this level, the Date table is not part of the filters you are applying to the data, so in this context there is no Date that the function can work out a previous day for.

If we create a simple pivot table that uses the Date table with the Date on rows, and our measure as the values, we see the following result:

Calendar	2005
Row Labels	PreviousDay
1/01/2005	31/12/2004 0:00
2/01/2005	1/01/2005 0:00
3/01/2005	2/01/2005 0:00
4/01/2005	3/01/2005 0:00
5/01/2005	4/01/2005 0:00
6/01/2005	5/01/2005 0:00
7/01/2005	6/01/2005 0:00

Figure 62: PREVIOUSDAY calculation viewed in a pivot table

As we would expect, the value of the PreviousDay calculation is the previous calendar day.

Now, to leverage this to create a movement in data values from the previous day, we need to retrieve the value of UnitsIn for the previous day. The formula we need to do this is:

---

<sup>18</sup> PREVIOUSDAY function reference: <http://technet.microsoft.com/en-us/library/ee634803.aspx>

```
PreviousDayUnits:=CALCULATE(SUM([UnitsIn]),PREVIOUSDAY('Date'[DateKey]))
```

This introduces a new function, CALCULATE<sup>19</sup>, and a new concept, filters. The CALCULATE function's power is in being able to modify the context of the formula. If we want the sum of UnitsIn, we can just create a SUM on that column. If we want to get that SUM for the previous day, we need to override the context provided by the Date table with a new context—in this case the context of the previous day, as calculated by the PREVIOUSDAY function.

Again, in the calculation area we don't get a meaningful results, as there is no Date context provided:

PreviousDay: (blank)
PreviousDayUnits: (blank)

Figure 63: PREVIOUSDAY without context

But if we add our new measure into our pivot table, plus the UnitsIn for comparison, we see this:

Calendar	2006		
Row Labels	PreviousDay	Sum of UnitsIn	PreviousDayUnits
31/05/2006	30/05/2006 0:00	0	0
1/06/2006	31/05/2006 0:00	14	0
2/06/2006	1/06/2006 0:00	2494	14
3/06/2006	2/06/2006 0:00	0	2494
4/06/2006	3/06/2006 0:00	12	0

Figure 64: The PREVIOUSDAY impact in the CALCULATE function

Here we can see that the results for PreviousDayUnits are now offset by one day from the result from UnitsIn.

To then create a movement, we just need to subtract the previous day's figure from today's figure:

```
DailyUnitsMovement:=SUM([UnitsIn])-
CALCULATE(SUM([UnitsIn]),PREVIOUSDAY('Date'[DateKey]))
```

Which yields the result:

<sup>19</sup> CALCULATE function reference: <http://msdn.microsoft.com/en-us/library/ee634825.aspx>

Row Labels	PreviousDay	Sum of UnitsIn	PreviousDayUnits	DailyUnitsMovement
31/05/2006	30/05/2006 0:00	0	0	0
1/06/2006	31/05/2006 0:00	14	0	14
2/06/2006	1/06/2006 0:00	2494	14	2480
3/06/2006	2/06/2006 0:00	0	2494	-2494
4/06/2006	3/06/2006 0:00	12	0	12

Figure 65: Using the PREVIOUSDAY and CALCULATE functions to create a day to day movement

Obviously these figures are a bit nonsensical, but the power to create this kind of calculation is very helpful if you need to monitor time-based changes in numbers, such as account balances, average temperatures, and so on.

## TOTALMTD

The TOTALMTD<sup>20</sup> function calculates a Month to Date Total. All you need to supply is the value you want to total and the Date to use. So if we wanted to create a Month to Date sum of UnitsIn, we would simply add a calculation with this formula:

*UnitsMTD:=TOTALMTD(SUM([UnitsIn]),'Date'[DateKey])*

This gives the following result in a pivot table:

Calendar 2007		
Row Labels	Sum of UnitsIn	UnitsMTD
1/10/2007	70	70
2/10/2007	7249	7319
3/10/2007	79	7398
4/10/2007	68	7466
5/10/2007	99	7565
6/10/2007	207	7772
7/10/2007	212	7984

Figure 66: TOTALMTD in action

Here we can see the UnitsMTD calculation slowly increasing in value with the UnitsIn that are added each day.

This isn't a very complex formula, but it is something that would require some complex manipulation in Excel. The great thing about Time Intelligence functions is that they make it very easy to do time-based analysis with a very low level of effort.

<sup>20</sup> TOTALMTD function reference: <http://technet.microsoft.com/en-us/library/ee634560.aspx>

## Context

We have been skirting around the subject of context throughout these last few sections<sup>21</sup>. Now it's time to dig deeper. The context defines what data is available when you do a calculation, so it's a mix of the impact of filters, relationships, and location of your formula.

There are three different types of context:

1. Row context
2. Query context
3. Filter context

We'll explore these one by one.

### Row context

We experienced row context when we were creating [Calculated Columns](#). In row context you have access to all the values in that row, plus any related data from any tables that are on the “one” side of a one-to-many relationship with that row.

This helps explain some of the issues we experienced building those formulas. The initial part of the formula we built—cost per kilo—only needed values from the row we were operating in, so it worked fine:

```
=IF(ISERROR([StandardCost]/[Weight]),0,[StandardCost]/[Weight])
```

All the highlighted values we wanted to use were available in the row context.

As we extended this, we pulled in related data from the “one” side of a one-to-many relationship from our Weight Conversions table using the RELATED function:

```
=IF(ISBLANK([WeightUnitMeasureCode]),0,[StandardCost]/([Weight]*RELATED(WeightConversions[Conversion factor])))
```

However, once we tried to pull in data from the “many” side of a one-to-many relationship, things stopped behaving quite so well and we ended up with this formula:

```
=SUMX(FILTER(FactProductInventory,FactProductInventory[ProductKey]  
=EARLIER([ProductKey])),FactProductInventory[UnitsIn])
```

In this case what we are effectively doing is creating our own context within that cell and largely ignoring the row context, except for where we use EARLIER to obtain a value from the current row to act as a filter.

So as quick summary, when we are in a calculated column, the following data is available within that row context:

---

<sup>21</sup> Context in DAX formulas: <http://msdn.microsoft.com/en-us/library/gg413423%28v=sql.110%29.aspx>

1. Values from columns in that row
2. Values from related tables that are on the “one” side of a one-to-many relationship

## Query context

When we created [Calculations](#) and worked with them in pivot tables, we began to work with query context. Whenever we create a measure such as a SUM of a column, you’ll note that the value that appears against the formula in the calculations is always the total of everything in that column. This is because at that time the query context for that measure is the whole table.

When we move that calculation into the pivot table, the way we slice and dice the data changes the context for each cell we calculate. Let’s explore this using a sample pivot table:

FILTERS		COLUMNS	
Calendar		AlternateSubCategory	
ROWS		VALUES	
Color		Sum of UnitsIn	

Figure 67: Query context sample pivot table settings

Here we have applied a filter, sliced on rows and columns and summed UnitsIn. Viewing this in the pivot table:

Calendar		2007	
Sum of UnitsIn		Column Labels	
Row Labels	Accessories Partners Clothing Pa		
Black			2850
Blue			2888
Grey			0
Multi			

Figure 68: Query Context Sample pivot table

To get our value of 2888 for the SUM of UnitsIn – highlighted in green, we get the context from Calendar as the pivot table filter (highlighted in blue) to year 2007. Context also comes from the color Blue (in purple) and SubCategory (in orange).

The application of this context is something that PowerPivot does automatically. Any time you look at a calculation in a client tool like Excel, when it returns an individual cell (such as the one highlighted in green above), it looks at all the applied filters to work out the context and applies that to the measure to give you the correct result.

To summarize, when we are looking at a calculation, the context comes from the slicers and filters you apply to the cell of data you are looking at.

## Filter context

Last of all is filter context, which we have explored when using the CALCULATE function to work out a Previous Day value. Filter context applies on top of query or row context (or in some cases can deliberately override it).

The CALCULATE function is an indirect way of looking at this effect, so we will look at it using a direct application. The following formula applies a FILTER function to a SUMX of UnitsIn:

```
FilteredSumOfUnitsIn:=SUMX(FILTER(FactProductInventory,RELATED(DimProduct[Color])="Blue"), FactProductInventory[UnitsIn])
```

It is using the RELATED function to identify if the Product is Blue in color, and filtering the Product Inventory table for those products that meet that criteria. If we examine this in a PivotTable, where we have a Calendar filter and use the product line on the rows:

Calendar	2006	
Row Labels	FilteredSumOfUnitsIn	Sum of UnitsIn
(blank)		1049
M		17463
R		23614
S	1365	18038
T	0	0
Grand Total	1365	60164

Figure 69: Filter Context pivot table

We can see here that the FilteredSum of UnitsIn is taking into account the filters applied by the query context (i.e. Calendar Year and ProductLine) and overlaying the filter context of the product having to be blue.

Filter context is a modification of query context. However as mentioned, it can also override it.



Using the ALL<sup>22</sup> function, you can override every filter. For example, in this formula:

```
UnFilteredSumOfUnitsIn:=SUMX(ALL(FactProductInventory),
FactProductInventory[UnitsIn])
```

Here we are asking the expression to disregard all filters on FactProductInventory when applying the SUMX. If we add this to our pivot table:

Calendar	2006		
Row Labels	FilteredSumOfUnitsIn	Sum of UnitsIn	UnFilteredSumOfUnitsIn
(blank)		1049	270969
M		17463	270969
R		23614	270969
S	1365	18038	270969
T	0	0	270969
<b>Grand Total</b>	<b>1365</b>	<b>60164</b>	<b>270969</b>

Figure 70: ALL function overriding query context

What we see is that all filters—including those from the Query context—have been removed, and we are just seeing the grand total of UnitsIn for the whole table. The use of this function in isolation is limited, but it often serves as a useful means of clearing filters in more advanced calculations, such as calculating percentages of totals.

<sup>22</sup> ALL function reference: <http://msdn.microsoft.com/en-us/library/ee634802.aspx>

# Chapter 2 Using your PowerPivot Model

Creating a rough and ready PowerPivot model is a great way to explore data, but most of the time those initial explorations need a little polishing if you are thinking about putting them in front of management or sharing them with your colleagues.

## Tidying it up

There are a few simple things you can do to make the model itself a little more presentable.

### Number formatting

First of all, any numeric or data column can have formatting applied to the numbers. On the Home tab of the data model there is a Formatting section:

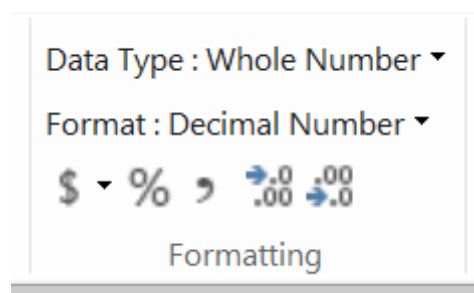


Figure 71: The Formatting section of the ribbon on the Home tab

Depending on your data type, you get different options—effectively you can only apply formatting to numeric and date columns. For example, we could apply the Decimal Number format to UnitsIn, removing the decimals, so you would get thousand separators in the numbers in that column.

The important thing is that any formatting you apply in the data model then flows on to several places. Firstly, any simple calculations directly on that column will inherit the format, so your SUM of UnitsIn will also have that format (our complex formula at a row level, however, will not).

More importantly, that formatting then also flows on to anything consuming that data, such as a pivot table or chart. This is demonstrated in the following figure:

Row Labels	Sum of UnitsIn	Sum of Total Units In
Black	80,928	80928
Blue	23,512	23512
Grey	0	0
Multi	25,057	25057
NA	47,771	47771

Figure 72: Formatting flowing through from the data model

This is a quick and easy way to avoid applying lots of formatting in tables and charts.

## Friendly names

The next thing you can easily adjust is the names of tables and fields. Most data sources have names that work well for databases or other data storage systems, but not so great for people. For example, our starting view for browsing our model in a pivot table looks like this:

Choose fields to add to report:

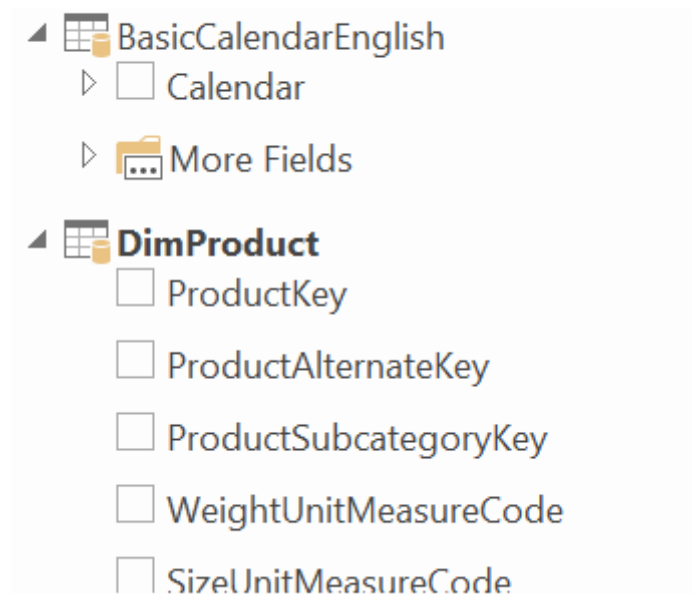


Figure 73: Browsing the data with its original names

Most business users would probably prefer to see “Product” rather than “DimProduct,” for example. Renaming the tables is simple. At the bottom of the grid view each table has its name on a tab:

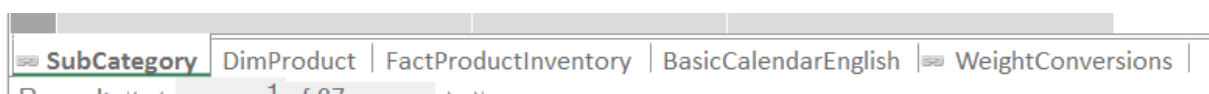


Figure 74: Table name in the tabs on Grid view

We can quickly double-click on each tab and rename the tables:



Figure 75: Renamed tables in the Grid View

These names are a bit more business friendly. However, note that on the Product Tab we are being given a formula error warning. Renaming tables doesn't flow through to the formulas automatically, so you can make a mess of your model pretty easily. It's generally easier to set the friendly name as you import the data.

Columns can also be given friendly names. You can do this in a couple of places—either within the Grid view by double-clicking on the column header, or in the Diagram View by double-clicking on the column name. Using the Diagram View is usually more efficient, as you can see all the columns in a smaller amount of screen space.

The following image shows the effect of renaming columns in the Diagram View:

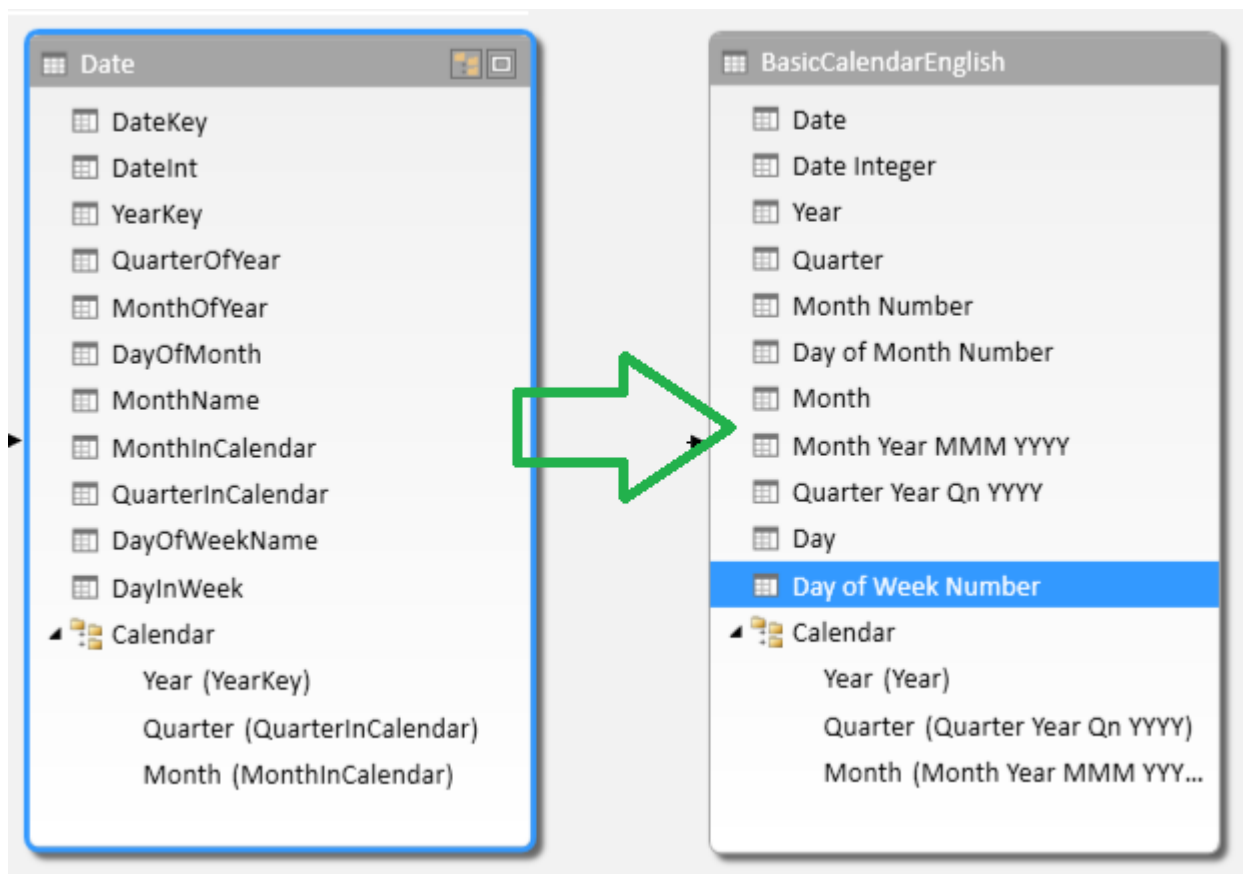


Figure 76: Renaming columns

This makes model more presentable, as you can turn technical names into terms business users will be able to understand.

## Hiding tables and columns

The next step is to conceal components of the model that will complicate the experience for the users the model will be shared with. This can be done at the table level and at the individual column level.

To hide a table, right-click on the name of the table and select **Hide from Client Tools**:

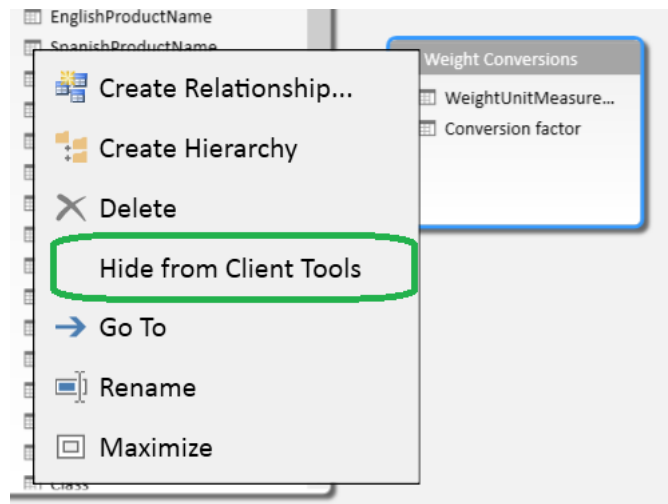


Figure 77: Hiding a table

This will leave the table visible in Diagram view (slightly greyed out) and in Grid view (with the tab name greyed out), but it won't be available when creating a pivot table. This can be useful when you have working tables—like the Weight Conversions table—that in themselves are not useful for end user analysis, but support calculations or relationships that are needed elsewhere.

Similarly, for columns in Diagram view, you can right-click on an individual column and choose **Hide from Client Tools**.

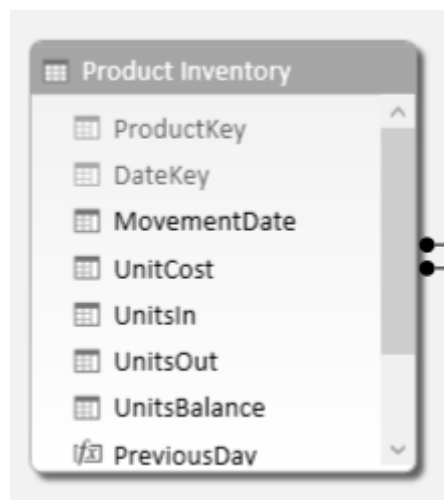


Figure 78: Hiding Columns

The column name will be shaded and as with hiding a table, will now not be available in a pivot table for analysis. This has a similar use case as there are columns, such as the Key columns in the above example, that have no value for end user analysis but are needed to support relationships and calculations.

The end result flows through to our pivot tables and charts, so we get a much cleaner experience when browsing our data:

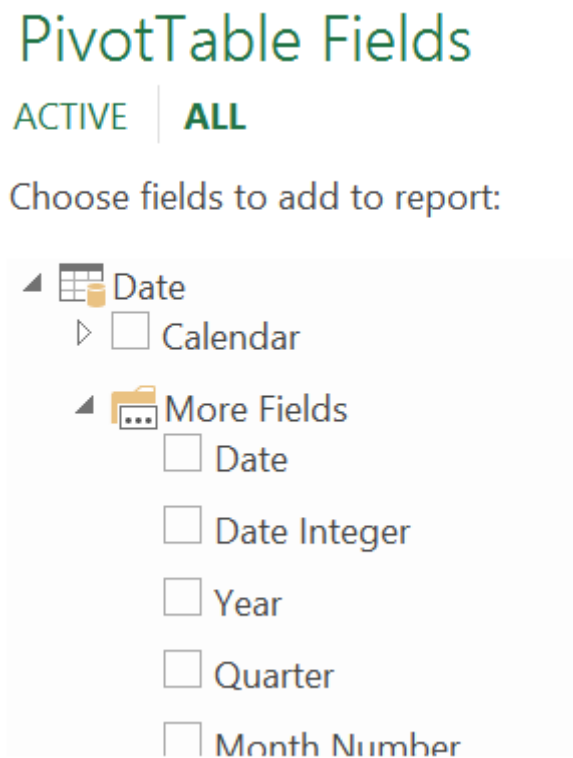


Figure 79: Name changes flowing through to the pivot table field list

## Pivot tables

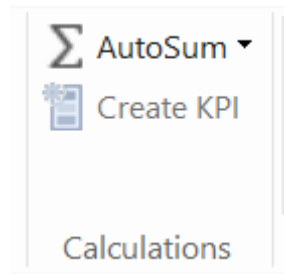
Once we have tidied up our data model we can use it to create some nice looking reports using pivot tables and charts.

We have already used pivot tables to do some basic exploration, so let's look at a couple of key features that PowerPivot enables or enhances in Excel. Pivot charts are pretty much unaltered in function so we won't look at those in this book.

## KPI's

Key Performance Indicators are a standard feature of any dashboard—usually in a traffic-light format, so dashboard users can rapidly take in which KPI's are performing well, and which ones are not.

PowerPivot natively supports KPI's as part of a variation of calculations. If we choose a cell in the area below the grid, we can add a KPI. To start with, we need to create a calculation; as you will note in the Calculations section of the Home tab of the ribbon, the "Create KPI" option is shaded:

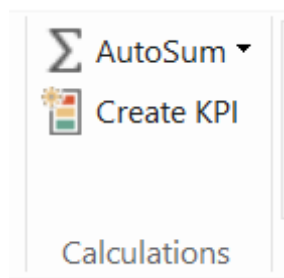


*Figure 80: The Calculations tab of the ribbon with dimmed Create KPI option*

We will pick an arbitrary calculation on Units Balance—in this case an average. The Autosum option gives us this formula:

*Average of Units Balance:=AVERAGE([Units Balance])*

Importantly, we now see the Create KPI option light up:



*Figure 81: The Calculations tab of the ribbon with Create KPI enabled*

Clicking on the **Create KPI** option while having the calculation highlighted gives us a dialog to configure the KPI:

Key Performance Indicator (KPI)

KPI base field (value): Average of Units Balance

**KPI Status**

Define target value:

☐ Calculated Field: No target fields have been defined

☒ Absolute value: 300

Define status thresholds:

200 400

Target

Select icon style:

OK Cancel

Figure 82: Configuring a KPI

There are three key sections to work with:

1. Define Target Value: This is the KPI goal, so this would be the target sales volume, for example.
2. Define Status Thresholds: This is where you set the boundaries for when the KPI turns red, yellow, or green.
3. Select Icon Style: This is where you choose what graphics your client tool will present (if supported).

Our metric is just an example, so our thresholds don't carry any meaning other than to demonstrate the effect. We update the settings as before, and then click **OK**.

The calculation now gets a little KPI adornment in the Grid view:

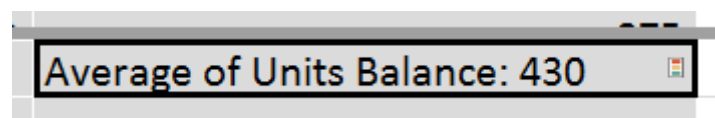


Figure 83: KPI adornment on a calculation

To demonstrate what this looks like in Excel, we'll create a simple pivot table with color on the rows. When we go to the pivot table fields to look for the calculation, we can see the way that it is displayed has changed:



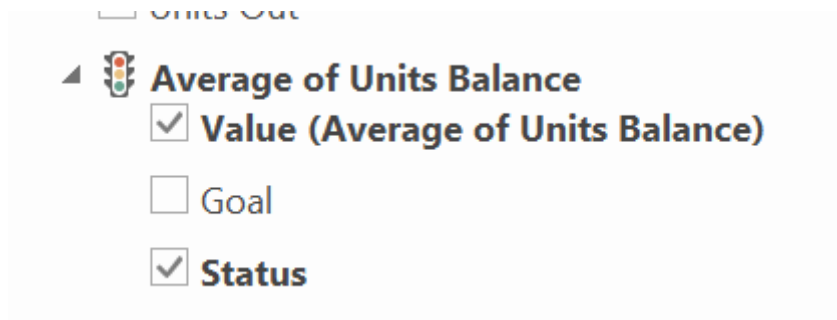


Figure 84: A KPI in the pivot table fields list

Under the traffic light icon we can see we have an option to see the measure's value, its goal (which can matter if you've set a dynamic target), and the status (represented by icons). Selecting all three gives us this in our pivot table:

Row Labels	Average of Units Balance	Average of Units Balance Status	Average of Units Balance Goal
Black	252		300
Blue	180		300
Grey	4		300
Multi	3		300
NA	690		300
Red	261		300
Silver	331		300
Silver/Black	430		300
White	3		300
Yellow	220		300
<b>Grand Total</b>	<b>430</b>		<b>300</b>

Figure 85: KPI's in pivot tables

Effective display of KPI's used to be reserved for high-end dashboarding tools, but courtesy of PowerPivot, you can manage them in Excel!

## Slicers

Slicers are a basic feature of Pivot tables<sup>23</sup> that have been available since 2010. They allow for rapid and intuitive filtering of data that works better with PowerPivot than native pivot tables.

In the pivot table Fields section of a pivot table pointing at our data model, expand the Calendar hierarchy, right-click on **Year** and choose the **Add as Slicer** option:

<sup>23</sup> Use slicers to filter PivotTable data: <http://office.microsoft.com/en-au/excel-help/use-slicers-to-filter-pivottable-data-HA010359466.aspx>

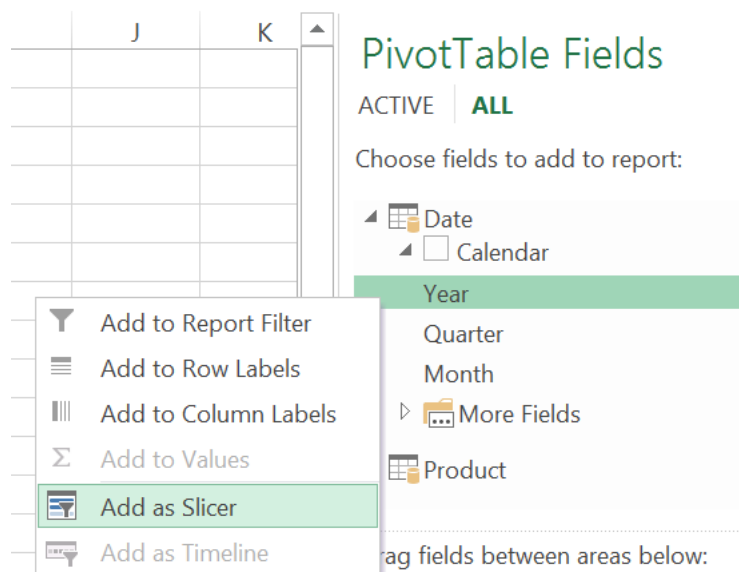


Figure 86: Adding a Slicer to a pivot table

To our pivot table, add **Alternate SubCategory** on the Rows, and **Units Balance** as the value. What we will see is this pivot table with this slicer:

Row Labels	Sum of Units Balance
Accessories Partners	826,317
Clothing Partners	33,388
Core Product	8,404,903
Engineering Partners	1,480
New Product	6,684,435
Parts Partners	2,187,810
(blank)	30,401,790
<b>Grand Total</b>	<b>48,540,123</b>

**Year**

2005

2006

2007

2008

1900

1901

Figure 87: Pivot table with slicer

There are a couple of things to note from this. First, the slicer lists the valid values for that pivot table at the top. When adding the slicer to start with, before the pivot table has any other data, it just listed all years from 1900 to 2100. Now that it has some context, the valid values (2005-2008) are listed at the top. All other values are still displayed, but shaded. The second thing to notice is how fast it is when you select different values—even though the data set has three quarters of a million rows in it, response times are instant.

Slicers are pretty slick in other ways, especially because you can attach them to multiple pivot tables and charts to drive some flashy interactive reporting. The interactivity slicers bring—particularly with the smarts that filters out irrelevant values—makes it easy to build impressive and intuitive interactive reporting.

## Power View

Another lesser known feature of Excel (2013 & Office 365 Professional Plus editions) is the data visualization tool Power View<sup>24</sup>. This is a rich, interactive tool that makes it easy to explore your data and create some very impressive reports.

To create a basic report, click the **Power View** button on the **Insert** tab of the ribbon. This launches Power View in a new tab on the workbook (you may need to install Silverlight on first use).

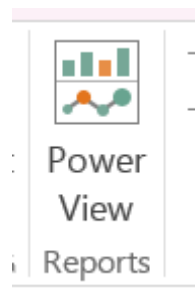


Figure 88: Power View on the ribbon

The Power View report surface is divided into three functional areas:

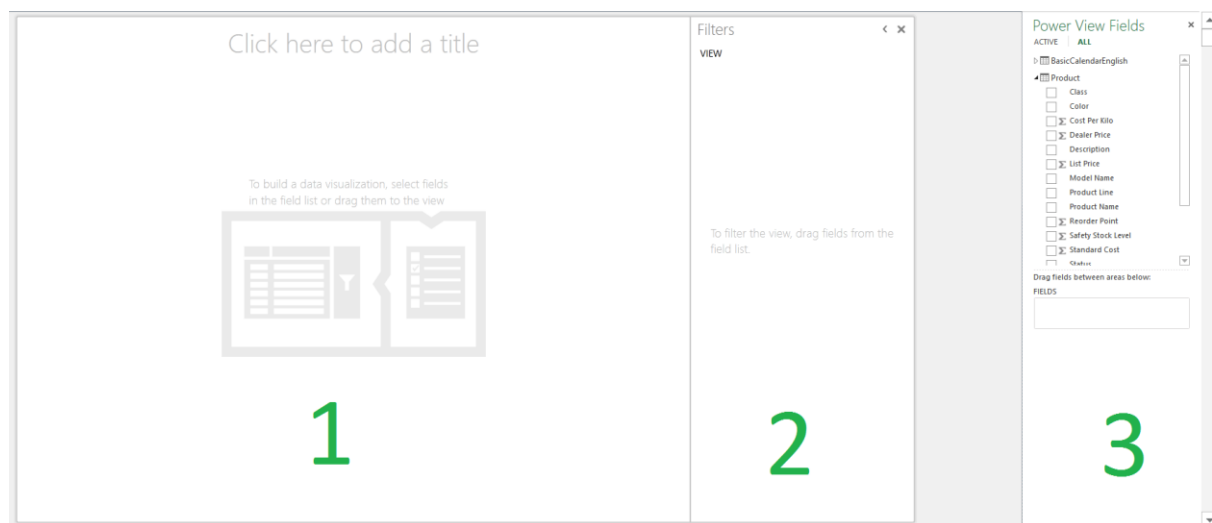


Figure 89: Power View report surface

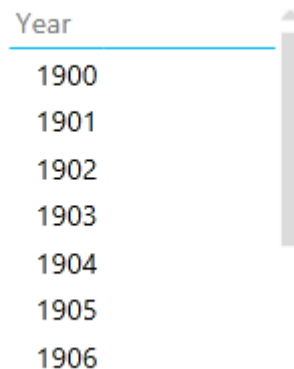
Area 1 is the actual report canvas, where data will be displayed. Area 2 is for applying filters to the whole report (if you wanted to cut data down to just one region or time period, for example). Area 3 is the Field list, which functions very similarly to the Field List in pivot tables in terms of selecting data items to work with.

Power View is designed to be much simpler to build reports in—anything you do should never take more than two clicks, and most operations are drag-and-drop.

---

<sup>24</sup> Power View: Explore, visualize, and present your data: <http://office.microsoft.com/en-au/excel-help/power-view-explore-visualize-and-present-your-data-HA102835634.aspx>

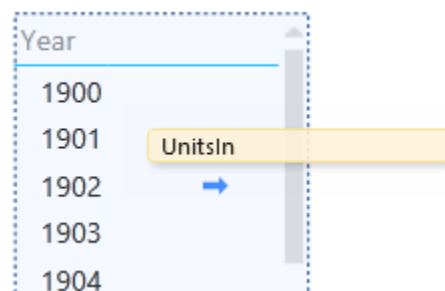
To create a simple data table, drag the **Year** level of the Calendar hierarchy onto the report design surface:



Year
1900
1901
1902
1903
1904
1905
1906

Figure 90: The Year hierarchy on the PowerView surface

Then into the same box, drag the **Units In** measure from Product Inventory:



Year
1900
1901
1902
1903
1904

Figure 91: Adding a measure to a table on the PowerView report surface

The basic table is created. Note how the formatting specified for the Units In measure is carried across to the report.

Year	Units In
2005	11,698
2006	60,164
2007	122,075
2008	77,032
<b>Total</b>	<b>270,969</b>

Figure 92: A basic PowerView table

Now we can have some fun. Let's convert this to a bar chart first. Select the table, and on the **Design** tab of the ribbon, click **Bar Chart** in the switch visualization section and choose **Stacked Bar**:

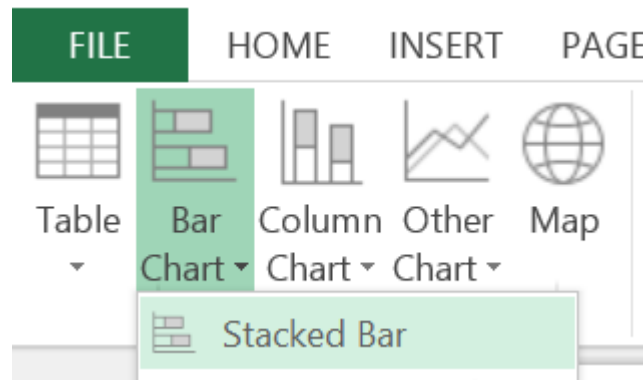


Figure 93: Changing chart type in Power View

The generated chart is a bit squashed, so by dragging on the corner on lower right-hand side, we can resize it to something more readable:

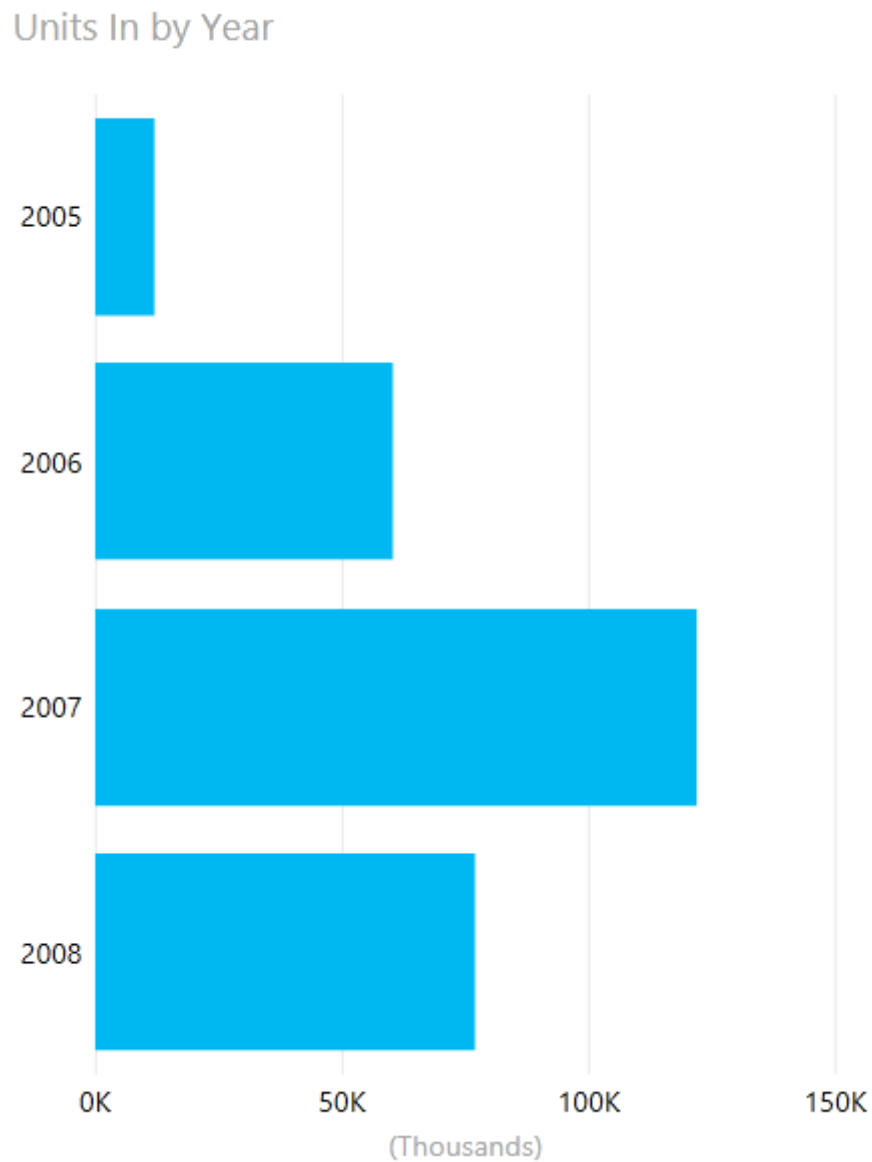


Figure 94: A bar chart in Power View

So, far so good. You can see the chart has also given itself a meaningful title. Now we can demonstrate the interactivity of Power View. On the right-hand side of the design surface, drag the **Color** attribute from Product, and then drag on to the table that appears Units In from Product Inventory. Then select an item on that chart and switch the visualization to **Pie Chart**.

This is where it gets interesting. Click on the slice of the pie chart for the largest category, Black. You'll see the chart to the left is then automatically filtered for black products:

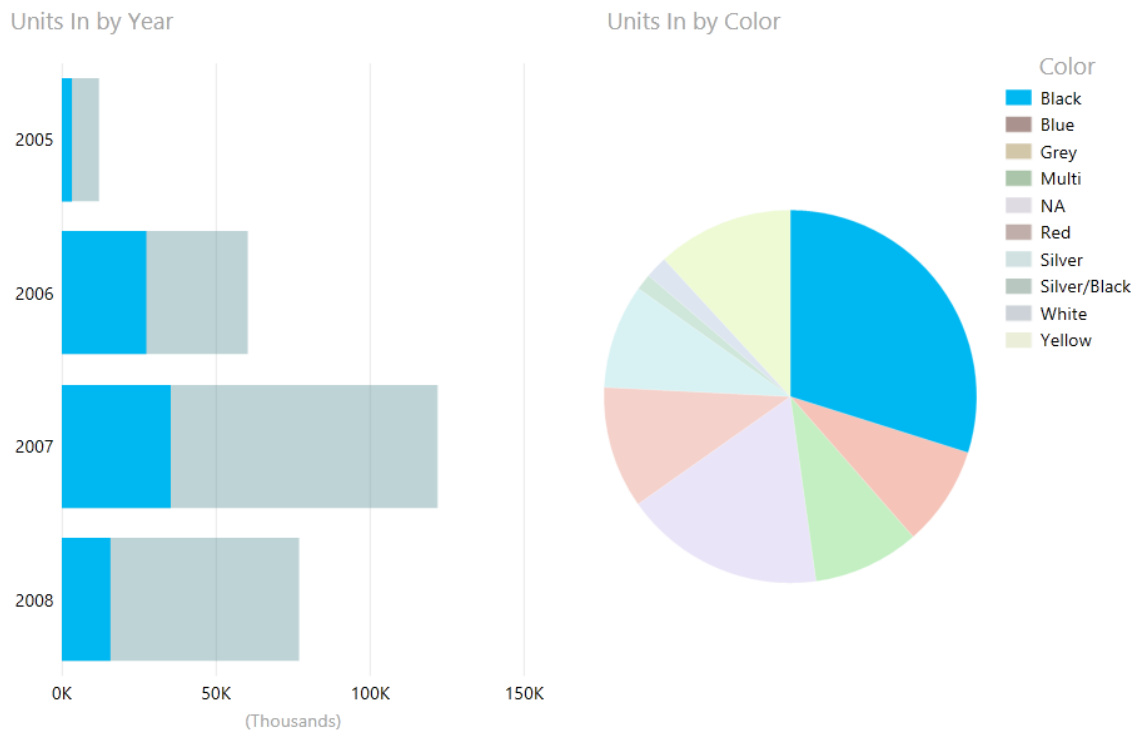


Figure 95: Cross-filtering report items in Power View

It goes the other way as well—if you click on one of the bars in the left-hand chart, the pie chart's results will be filtered for that year. The relationships in the underlying data model enable this kind of user-friendly cross-filtering with no expertise for the user at all. Plus it looks pretty slick!

## Cards

Want to create highly visually appealing reports using images? Power View supports the incorporation of images from the data model into reports as cards<sup>25</sup>, which look like this:

<sup>25</sup> Cards in Power View: <http://office.microsoft.com/en-au/excel-help/cards-in-power-view-HA104013489.aspx>

## Cards Example




	Mountain-300 Black, 48 Product Name	112,464.00 Units Balance
	Mountain-400-W Silver, 38 Product Name	110,485.00 Units Balance
	Mountain-400-W Silver, 40 Product Name	109,738.00 Units Balance

Figure 96: Power View cards incorporating images

Here we have brought in the images that are embedded in the product dataset and surfaced them on the report alongside some of the data sets attributes and metrics.

To get this functionality, a few tweaks need to be made to the table holding the images. On the Advanced tab of the Data Model, there is a section for Reporting Properties. Click on the **Table Behavior** option:

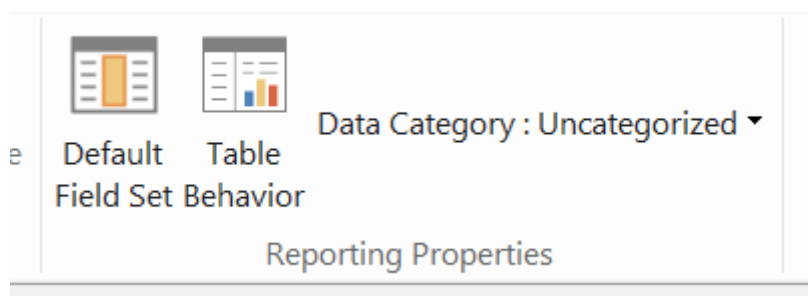


Figure 97: Reporting Properties on the Advanced tab of the ribbon

This launches the Table Behavior dialog box, which needs a few settings configured. The following example has them filled out for our needs:

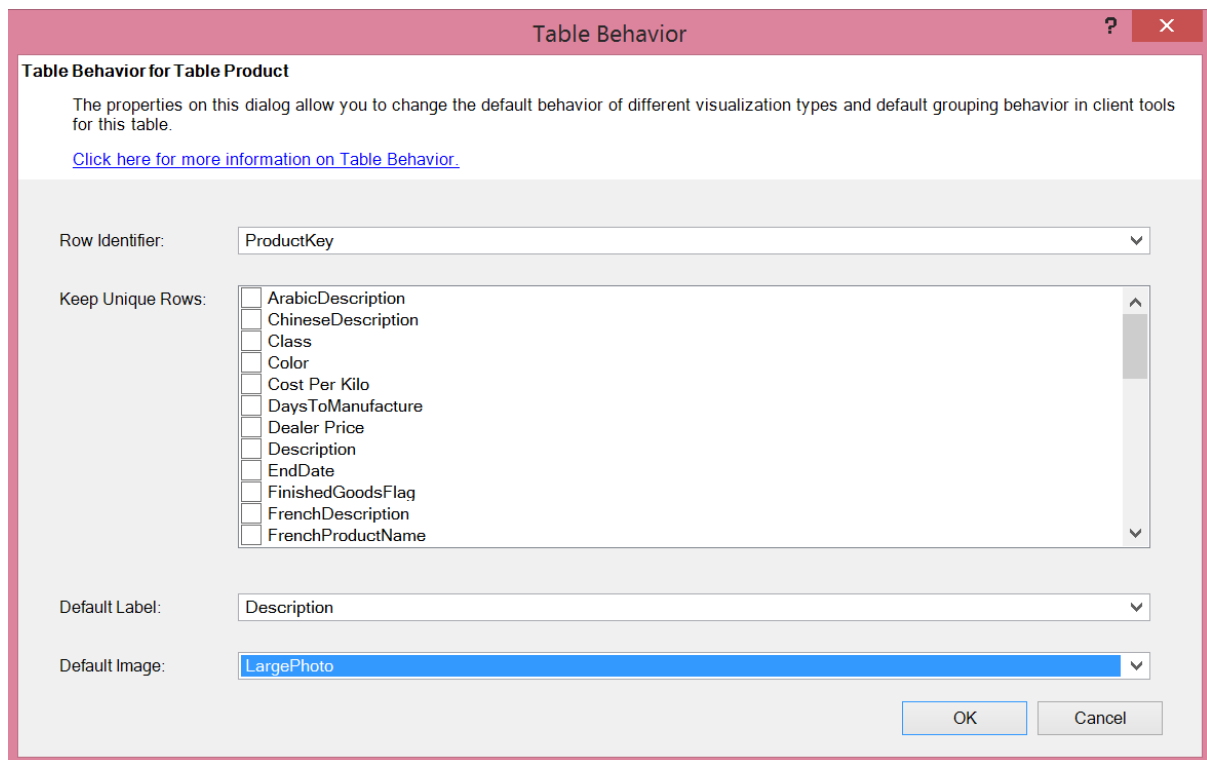


Figure 98: Table Behavior dialog box

The key settings are to set a unique row identifier—ProductKey works for us—and a default image. In the model the LargePhoto column contains the image, though it displays in the Grid view like so:

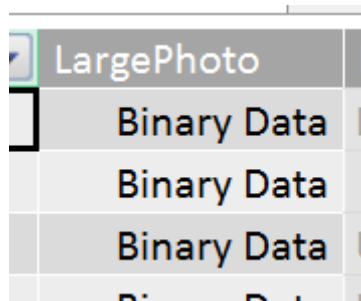


Figure 99: Binary Data in the Grid View

With these settings, the LargePhoto column can be dragged onto the Power View report surface and display as an image. Using the Cards visualizations then enables the kind of reports demonstrated at the start of this section.

## Maps

Another impressive visualization available is maps data. Here is a simple example:



Value by Suburb, and Category



Figure 100: PowerView maps example

This map was built off of the following simple data set:

Suburb	Value	Category
Sydney, NSW, Australia	10000	East
Brisbane, Australia	2000	East
Hobart, Australia	3000	Unknown
Melbourne, Australia	7600	South
Perth, Australia	5600	West
Cairns, Australia	1200	East
Adelaide, Australia	8900	South

Figure 101: Maps example data set

One thing to immediately notice is that the data hasn't been geocoded. Power View takes the location data and sends it to Bing to geocode it—so as long as Bing understands it, it can create a point on the map for it. This means that sometimes you need to add some extra detail to your data to get the correct results. The town of Birmingham, for example, can be found most famously in the United Kingdom, but also exists in Alabama, U.S. It's usually advisable to extend address data with state and country data to get the right results.

These maps can be zoomed in and out to give an appropriate level of scale. They do have some limitations—mainly that the visualization is limited to circles on the map (you can subdivide the circles into pie charts but this kind of visualization is largely useless for conveying information except in rare cases). You can color the circles by categories (as we did in the previous example using the Category column), which is the most meaningful way to apply some differentiation to the dots. This however relies on each dot only belonging to one category; otherwise the pie chart effect just discussed comes into play.

# Chapter 3 Sharing your PowerPivot Model

Often once a useful PowerPivot model has been created, it needs to move off the desktop and to places where it can be accessed by other people. Next we will explore the means for doing so.

## Standalone workbook

A PowerPivot workbook is a standalone, complete data model. All the data from the sources has been stored in the workbook. You can simply share it by putting it on a file share or emailing it, or any option you have for passing a file around.

The only constraint is that if there is a need to refresh the data, and the recipient may not be able to do so if they do not have valid credentials for the data source.

## SharePoint workbook

Uploading a PowerPivot model into SharePoint can open up some interesting sharing options if PowerPivot for SharePoint features have been enabled<sup>26</sup>.

These features include:

1. The workbook data model becomes an accessible data source
2. Excel services integration for providing visualization in SharePoint pages
3. Automated data refresh
4. Access control
5. Workbook monitoring

Let's dig into these features a little more.

First, uploading the workbook into a suitable, enabled SharePoint environment allows some behind-the-scenes wizardry to occur with Analysis Services. The workbook's data model becomes accessible as a data source in its own right. This works (at a very simplified level) by pushing the data model in the workbook back to a set of specially configured Analysis Services servers, which are part of the SharePoint farm. This then enables it to operate as a standalone data model with the servers supporting requests for data, also enabling scalability. Subsequently PowerView, reporting services, and Excel services can then build reports on the uploaded data model.

---

<sup>26</sup> PowerPivot for SharePoint: <http://technet.microsoft.com/en-us/library/ee210682%28v=sql.105%29.aspx>

Second, as just alluded to, Excel Services integration allows the embedding of Excel content such as charts and pivot tables into SharePoint web pages—and Excel Services will be able to use the PowerPivot data model as a source. This means if you have created a set of useful dashboards and reports as part of building the PowerPivot workbook, you can now share them broadly as part of a SharePoint site.

As these data models would be of limited use if the data had to be manually refreshed, SharePoint enables the scheduled refresh of the data in the model<sup>27</sup>. This is limited to a once-per-day refresh (as of SharePoint 2013).

Next, putting the workbook in SharePoint enables the usual level of access control that you get for any document managed in that environment. We will cover this further in the [Security considerations](#) section.

Finally for the IT Administrators, there is the ability to monitor<sup>28</sup> which workbooks are being accessed, whether data refreshes are failing, and how server resources are being utilized. This can be useful for identifying candidates for models that could be turned into fully supported BI solutions (as well as any unloved models that are draining resources and could be removed).

In summary, if you are in an enterprise environment that leverages data heavily, these features greatly enable the ideals of self-service BI. That is, users can create their own reporting and content, can be flexible about the data sources they include, and yet still have access to the horsepower provided by a full BI solution. The IT Administrators have a window to keep an eye on things to manage the self-service capabilities without making users be subject to normal deployment and release restrictions. That does not set aside some of the other risks of self-service models, such as poor quality data or bad formulas, but it at least gets the heavily used solutions under watch.

## Tabular model

Your model might need further capabilities, such as:

- more frequent or managed refreshing
- more granular security
- capacity to handle larger amounts of data
- very wide distribution
- centralized management

In these cases, it may be beneficial to consider upgrading the model from a workbook to a full business intelligence solution as part of SQL Server Analysis Services.

---

<sup>27</sup> Automatically Refresh PowerPivot Data in SharePoint: <http://msdn.microsoft.com/en-us/library/gg413467%28v=sql.110%29.aspx>

<sup>28</sup> PowerPivot Management Dashboard and Usage Data: <http://msdn.microsoft.com/en-us/library/ee210626.aspx>

The migration path is very simple. In SQL Server Data Tools<sup>29</sup> (part of Visual Studio), from the **File** menu choose **New Project**, and in the Analysis Services templates is the **Import from PowerPivot** template:

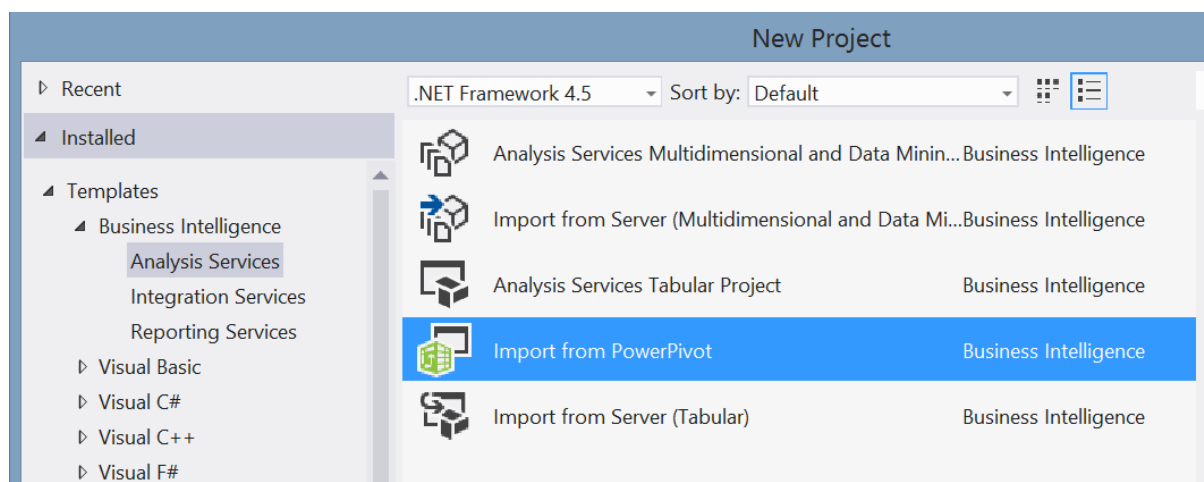


Figure 102: Importing a PowerPivot workbook into Visual Studio

Choosing this template will create a new Analysis Services Tabular project. As part of the creation process it will ask for a PowerPivot workbook to import. Once you select it, the project will create a complete tabular model from your workbook. It will have all the tables, relationships, calculations and other settings from the workbook. The design surfaces are very similar—there is the Grid view and the Diagram view to choose from, and the same formula language, DAX, is still used. Some work may be required to ensure all the data connections will work when the model is deployed to a server, but the model is otherwise good to go. This adds the following capabilities:

- More frequent or managed refreshing
- Scheduled refreshes through SQL Agent and/or SSIS
- Scheduled data feeds from SSIS
- More granular security
- Security roles restrict access at a row level, allowing different users to see different aspects of the data model
- Capacity to handle larger amounts of data
- The sky is effectively the limit—the 2GB limit is gone. The largest model I have heard of in production was 2TB in size!
- Very wide distribution
- Being part of an Enterprise server infrastructure gives the performance and scalability a workbook simply can't

---

<sup>29</sup> Microsoft SQL Server Data Tools: <http://msdn.microsoft.com/en-au/data/tools.aspx>

- Centralized management
- Again, being part of Enterprise infrastructure gives access to skilled IT resources to keep the model up and running.

The ease with which you can leap from workbook to fully capable BI solution is to me one of the greatest features. It enables BI professionals to rough out proofs of concept very quickly in Excel before taking the leap and industrializing—and users get what they saw in the POC, just more robust.

## Security considerations

PowerPivot models created in Excel have a very basic security model, which is worth bearing in mind if you share workbooks, as it is very easy to share sensitive data this way.

Standalone PowerPivot models in an Excel workbook are fully readable by anyone you give the file to. That means if you build a customer analytics model and share it around the organization, if you embedded customer personal data in that model then anyone with the workbook can see it. Once the data has been imported from the source system using your credentials into PowerPivot, it is no longer secured through any form of authentication. The only constraint is that (unless you embedded your credentials in the workbook) the recipient will not be able to refresh the data.

PowerPivot models uploaded into SharePoint are similarly insecure. Authorization to access SharePoint is required to get to the file, but once it is there, if someone with access downloads and shares the workbook, the data is exposed again. In terms of accessing it as a data source, SharePoint authentication does provide some protection.

As a broad rule, if you are concerned about the sensitivity of the data in your workbook, then focus on sharing the results of your analysis, but not your analysis itself.

However, if your data is very sensitive and you do require proper access control, then moving your model into a SQL Server Tabular model is the most secure way to do so. SQL Server security will control data model access and also can extend security to rows of data. So for example, you can secure whole departments' data to just their employees given a suitable filter.

# Chapter 4 A Note on Instability

Unfortunately, while PowerPivot is an amazing desktop tool, it sometimes isn't a terribly well-behaved one. If you start experiencing unhelpful errors such as "Object not set to a reference of an Object" when working with PowerPivot, there are three actions you can take to resolve this:

## Remove the add-in

Step one is to remove the add-in (via **File > Options > Add Ins > Manage COM Add Ins**), restart Excel and add the add-in back in again. Try reopening your workbook and seeing if it behaves. If so, all good. If not, escalate to the next step.

## Repair Office

The next option is to close all Office programs and go to **Add/remove programs** in the Control Panel. Locate your installation of Office, right-click, and choose **Change**. An option will come up to **Repair**. Choose this and go and get a coffee. After this process, reopen your workbook and if it works, the problem is resolved (until next time).

## Try and fix the problem in the data model

If the data model will load in some form, you may have another option, though it involves deleting data and calculations—so take a backup first!

If it loads the grid but gets stuck on a particular table, go to Diagram view and hunt for calculations with error markers, and delete them. Unfortunately, the code is irretrievable.

If the grid still won't load, try deleting relationships to the problem table in Diagram view. If it still doesn't load, try deleting the table.

After that the options are getting slim, and rebuilding the model is often the most effective way to get back to square one.

## Chapter 5 Deep Dive: The xVelocity engine

At the heart of PowerPivot's amazing capabilities is the xVelocity engine. This is the powerful technology that enables PowerPivot workbooks to compress data by such an impressive margin, allowing huge data sets to be stored in relatively small amounts of memory. Understanding what data compresses well and why will help you understand how your workbook is using memory.

### Understanding compression

The xVelocity engine in PowerPivot achieves compression primarily through a feature called Dictionary Encoding. This works by creating an integer index to represent your data values, creating a dictionary to store the index/value relationships, and only storing the index in the data.

To spell this out in simpler terms, consider a collection of names in our data:

Name	Value
John Smith	10
Ann Smith	23
Marco Russo	35
Chris Webb	7
Alberto Ferrari	8
Ann Smith	378
Marco Russo	5543
Chris Webb	54

*Figure 103: Sample data for understanding compression*

We can see several of these names are repeated, so there is some storage overhead in repeating data. PowerPivot will create a dictionary of these with an index:



ID	Name
0	John Smith
1	Ann Smith
2	Marco Russo
3	Chris Webb
4	Alberto Ferrari

Figure 104: Dictionary for the Name column

Then, this gets stored in the workbook as:

Name ID	Value
0	10
1	23
2	35
3	7
4	8
1	378
2	5543
3	54

Figure 105: Data stored with the dictionary index

This allows for a significant amount of data compression. Even the shortest name on our list, Ann Smith, consumes 9 bytes of data. In the small dictionary we described, we need 1 byte to store up to 10 entries in the list. Even if we had 10,000 names in the list, we could get away with a 5-byte number to store it.

In practical terms, this means that data that is repeated more frequently in your dataset will get compressed more effectively. For example, product codes that are repeated across sales transactions will compress very well. In the reverse, some data will not compress well at all—for example the sales transaction amounts, which will vary from line to line with very little repeatability.

A small caveat to all this: the previous description covers how the compression algorithm usually works and is broadly agreed to behave. However, the actual compression engine is not publically exposed and sometimes will behave differently, though the precise circumstances for this will never be exposed to an end user.

## **Managing very large workbooks**

If your workbook is getting too big, there are a couple of tricks to bring it back down in size, both of which require only a little knowledge of how compression works.

The first technique is simply to get rid of some columns. Data not stored in your workbook will obviously take up less space! Understanding compression helps you understand which data you can get rid of for the most impact. Removing columns with lots of repeating data, such as product and transaction codes, won't save you much space—even if the codes themselves are very long. However non-repeating columns—such as timestamps, currency amounts and so on—can make a big difference.

The second technique requires a little bit of thinking. If you are stuck with non-repeating data, such as a timestamp, currency amount, or even people's names, you could try breaking those columns up into chunks that do repeat more often, and only combining them when you do calculations.

Consider transaction amounts, which are commonly rounded to two decimal places:

Amount

7.99

8.99

10.95

12.95

7.49

10.49

15.49

7.75

*Figure 106: Sample transaction amounts*

This list currently has eight unique values. If we split out the decimal portion from the main number, then the repeatability changes:

Amount	Dollars	Cents
7.99	7	99
8.99	8	99
10.95	10	95
12.95	12	95
7.49	7	49
10.49	10	49
15.49	15	49
7.75	7	75

*Figure 107: Sample transaction amounts split out*

Now we have five unique dollar amounts and four unique cents amounts. At this tiny scale, this would increase the space needed—but consider the effect over thousands of transactions. The cents portion will only ever have 100 permutations, which can be stored in just 2 bytes. Depending on how variable your dollar amounts are, this could save quite a lot of space. Dates with timestamps, and even first and last names can be looked at in this way.

# Additional Resources

If you want to learn more, here are some excellent starting points:

## Microsoft

PowerPivot for Excel Tutorial Introduction: <http://msdn.microsoft.com/en-us/library/gg413497%28v=sql.110%29.aspx>

QuickStart: Learn DAX Basics in 30 Minutes - <http://office.microsoft.com/en-au/excel-help/quickstart-learn-dax-basics-in-30-minutes-HA102891601.aspx?CTT=5&origin=HA102836100>

Understanding Date Tables and Time Intelligence: <http://office.microsoft.com/en-au/excel-help/understand-and-create-date-tables-in-power-pivot-in-excel-2013-HA104139621.aspx>

## Independent

Chris Webb's BI blog: <http://cwebbbi.wordpress.com/> Chris is a deep technical expert who has spent much of his career digging into Analysis Services.

SQLBI: <http://www.sqlbi.com/> Run by Alberto Ferrari and Marco Russo, two vocal figures in the PowerPivot world. Their site has a large number of in-depth articles and white papers for those pushing their models to the limit.

BI Monkey: <http://www.bimonkey.com> My blog, while not as authoritative as the other sources mentioned, often deals with complex modeling scenarios and some of the day-to-day realities of working with PowerPivot.