

# 'DIY BI' Tips, Tricks & Techniques

## Power Pivot Edition

By Excelguru Consulting Inc.



03

POWER PIVOT



EXCELGURU

## 'DIY BI' Tips, Tricks & Techniques – Power Pivot Edition

© 2022 Excelguru Consulting Inc.

Excel, Power Query, Power Pivot, Power BI, and their logos are registered trademarks of Microsoft Corporation in the United States and/or other countries, and are property of their respective owners.

Power Query Academy Logo: © Skillwave Training

All rights reserved. No part of this eBook may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information or storage retrieval system without permission from the publisher. Every effort has been made to make this eBook as complete and accurate as possible, but no warranty or fitness is implied. The information is provided on an “as is” basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this eBook.

This eBook is acquired completely free-of-charge by simply signing up to our newsletter on [www.excelguru.ca](http://www.excelguru.ca). If someone charged you for it, we suggest that you request a refund.

Author: Ken Puls  
Layout & Design: Excelguru Consulting Inc.  
Copyediting: Excelguru Consulting Inc.  
Cover & Illustrations: Excelguru Consulting Inc.  
Logos on pages 1, 19 & 25: Skillwave Training

Published online by Excelguru Consulting Inc. on July 3, 2018

Updated version published on February 4, 2020

Updated version published on April 22, 2021

Updated version published on August 9, 2022

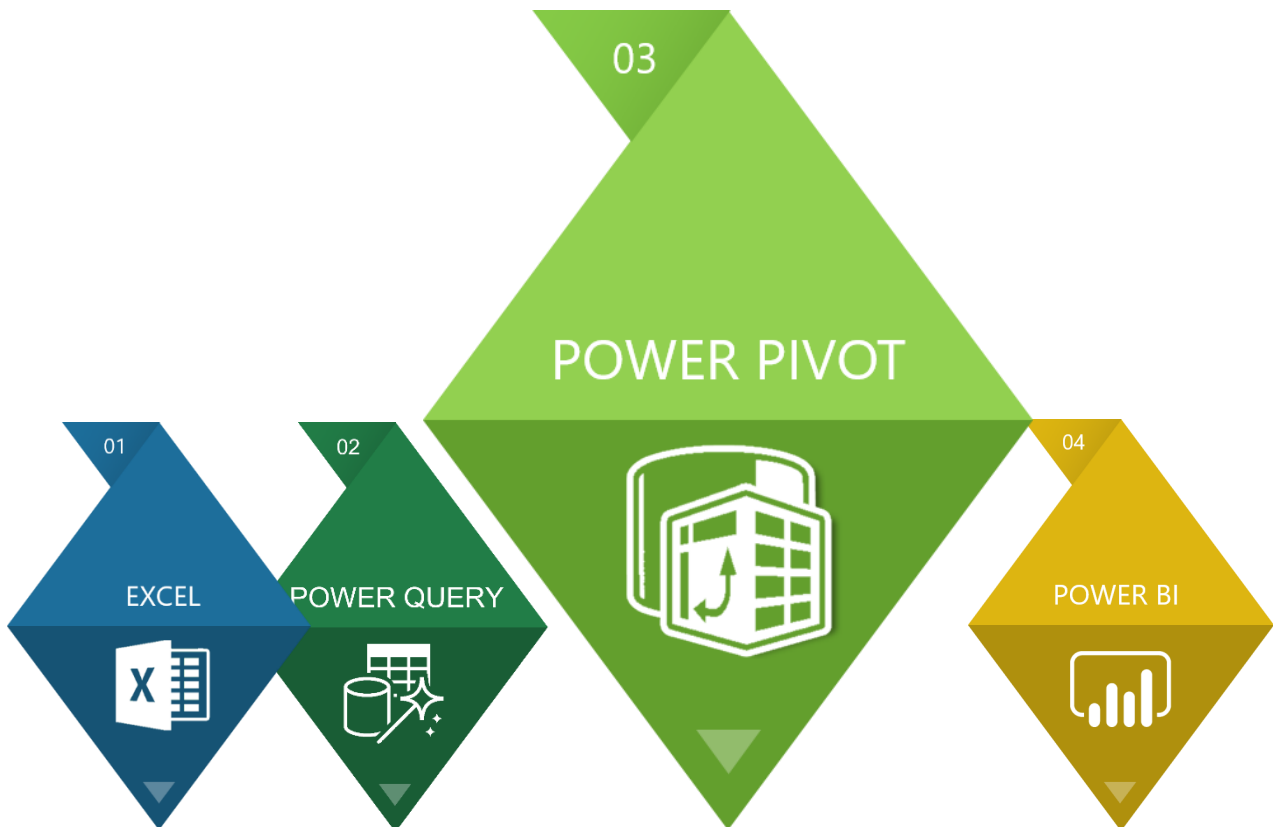
03

POWER PIVOT



## About this Book

This free eBook is the third in a series of four that will be released to our Excelguru email newsletter subscribers. Each book contains five of our favourite tips, tricks, and techniques, with one book each for Excel, Power Query, Power Pivot, and Power BI. The tips and tricks shared in these books have been developed over years of research and real-world experience.



## Get the Entire Series

Don't miss out on the other three books in this series. They're all free, and you can get them as a reward for subscribing to the Excelguru newsletter. In addition to the free book, you'll also get our monthly(ish) email newsletter featuring the latest updates for Excel and Power BI, upcoming training sessions, new products, and other information.

If you are not already a subscriber, sign up today at <http://xlguru.ca/newsletter>.



## Table of Contents

The Demo Model .....	1
Tip 1: Hiding Fields from a User .....	4
Tip 2: Hiding Zeros in a Measure .....	9
Tip 3: DAX Variables .....	11
Tip 4: Retrieve a Value from an Excel Slicer .....	14
Tip 5: One Field, Multiple Slicers, Different Views.....	18



## The Demo Model

### Sample File and Version Requirements

With the exception of one tip, all of the techniques in this eBook can be built in Excel 2010 or higher<sup>1</sup>. Having said this, I also make heavy use of Power Query in the model setup, and I also use Timelines for filtering. This means that the sample files I am providing are only available for Excel 2013 and higher.

**Excel 2013 Users:** I have included a version labelled for your version of Excel. Since you don't have access to DAX variables, you must use this version of the sample file. All other users can use the version labelled Excel 2016+.xlsx.

You can [download the completed sample files by clicking here](#).

### Model Overview

The model we'll use for the demonstration is a fairly simple one, with four tables. Shown in the Power Pivot diagram view it looks like this:

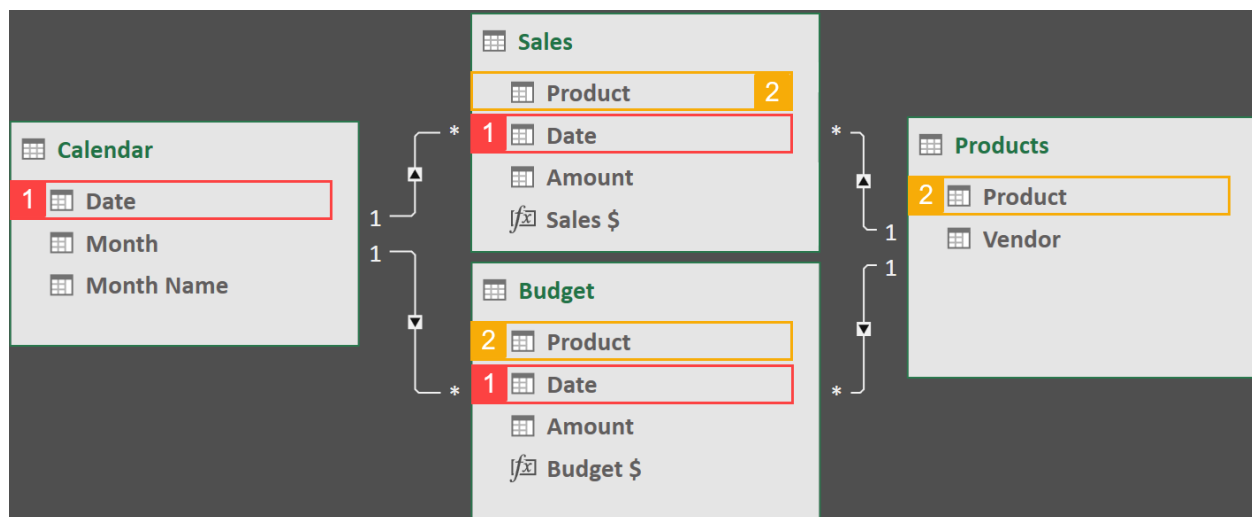



Figure 1 - The Power Pivot model in Diagram view

As you can see, I've highlighted the relationships to make them a bit easier to read. And to give you an idea of the size of this model, it's tiny, containing less than 400 rows of data between all four tables. (In fact, most of those are in the Calendar table!)

It is worth noting that each of these tables has been sourced via Power Query and loaded directly into the Power Pivot data model.



**Extend Your Learning**  
Do you want to truly Master Your Data? You need **Power Query Academy!**  
Get started with our FREE [Power Query Fundamentals](#) course.

<sup>1</sup> Tip 3 shows how to make DAX formulae shorter using the new variables introduced in Excel 2016. This will not prevent earlier versions from making use of the other four tips in this eBook.



## The Calendar Table

The Calendar table is actually the longest table in the model, with 365 rows. There is one record for each day from Jan 1, 2017 through Dec 31, 2017, in order to allow taking advantage of Power Pivot's time intelligence formulae. Here's a quick look at the format of this data:

	Date	Month	Month Name
1	2017-01-01 12:00:00 AM	1	Jan
2	2017-01-02 12:00:00 AM	1	Jan
3	2017-01-03 12:00:00 AM	1	Jan
4	2017-01-04 12:00:00 AM	1	Jan
5	2017-01-05 12:00:00 AM	1	Jan
6	2017-01-06 12:00:00 AM	1	Jan
7	2017-01-07 12:00:00 AM	1	Jan
8	2017-01-08 12:00:00 AM	1	Jan

Figure 2 - The Calendar table

## The Products Table

The Products table is quite short, with only 3 rows; 1 each for Products A, B and C, and their corresponding Vendor name. You can see all the data in Figure 3 below:

	Product	Vendor
1	A	Acme
2	B	Boingo
3	C	Caltrop

Figure 3 - The (entire) Products table

- Sales has 12 rows, listing sales for 4 months as shown in Figure 4 below.
- Budget also only has 12 rows, holding 4 months of budgets by product as shown in Figure 5.

	Product	Date	Amount
1	A	2017-01-31 12:00:00 AM	2367
2	B	2017-01-31 12:00:00 AM	4872
3	C	2017-01-31 12:00:00 AM	3831
4	A	2017-02-28 12:00:00 AM	4098
5	B	2017-02-28 12:00:00 AM	4040
6	C	2017-02-28 12:00:00 AM	4182
7	A	2017-03-31 12:00:00 AM	3305
8	B	2017-03-31 12:00:00 AM	3490
9	C	2017-03-31 12:00:00 AM	2238
10	A	2017-04-30 12:00:00 AM	1790
11	B	2017-04-30 12:00:00 AM	2610
12	C	2017-04-30 12:00:00 AM	1688

Figure 4 - Sales data

	Product	Date	Amount
1	A	2017-01-31 12:00:00 AM	1000
2	B	2017-01-31 12:00:00 AM	5000
3	C	2017-01-31 12:00:00 AM	1000
4	A	2017-02-28 12:00:00 AM	3000
5	B	2017-02-28 12:00:00 AM	3000
6	C	2017-02-28 12:00:00 AM	3000
7	A	2017-03-31 12:00:00 AM	3000
8	B	2017-03-31 12:00:00 AM	4000
9	C	2017-03-31 12:00:00 AM	5000
10	A	2017-04-30 12:00:00 AM	3000
11	B	2017-04-30 12:00:00 AM	3000
12	C	2017-04-30 12:00:00 AM	5000

Figure 5 - Budget data

## The DAX Measures

Finally, the model contains only two simple measures, to summarize the amounts for the Sales and Budget table:

```
Sales $: =SUM(Sales[Amount])
Budget $: =SUM(Budget[Amount])
```



In fairness, these measures are so simple, that I didn't even need to write them. I could have just as easily achieved the same results by dragging the [Amount] column from each table into a PivotTable, creating an Implicit Measure. As the fields are numeric, it would have aggregated them using the SUM function. So why did I explicitly create these? There are a few reasons:

1. Implicit measures are named by prepending the column name with the aggregation type by default (Sum of Amount or Count of Product). If I create an explicit measure, the descriptive, presentation-ready name I choose is used whenever I add the measure to any PivotTable.
2. The raw columns in this case are both called "Amount". If I add both of these to a PivotTable, they both get called "Sum of Amount". So, which is which?
3. When I create a measure, I get to choose the default formatting that the measure uses.

Overall, it's all about investing a little time up front to save me the time and hassle of renaming and reformatting later. But it also adds a bonus in that an explicit measure means I know *exactly* what the measure does. I'm such a believer in this, that I will never use a raw column in the values area of a PivotTable, and always write explicit measures. As an added bonus, I believe that it makes it easier to build more complex measures later by nesting pre-built measures in CALCULATE and other functions.

### Facts vs Dimensions

As a final bit of background here, it's a good idea to quickly review a key concept for PivotTables: the concept of Facts versus Dimensions. What are those? It's actually fairly simple. If a column is going to be aggregated, it's a Fact. If it's going to "Slice" the data, it's a Dimension.

So how do you identify them? It's easy!

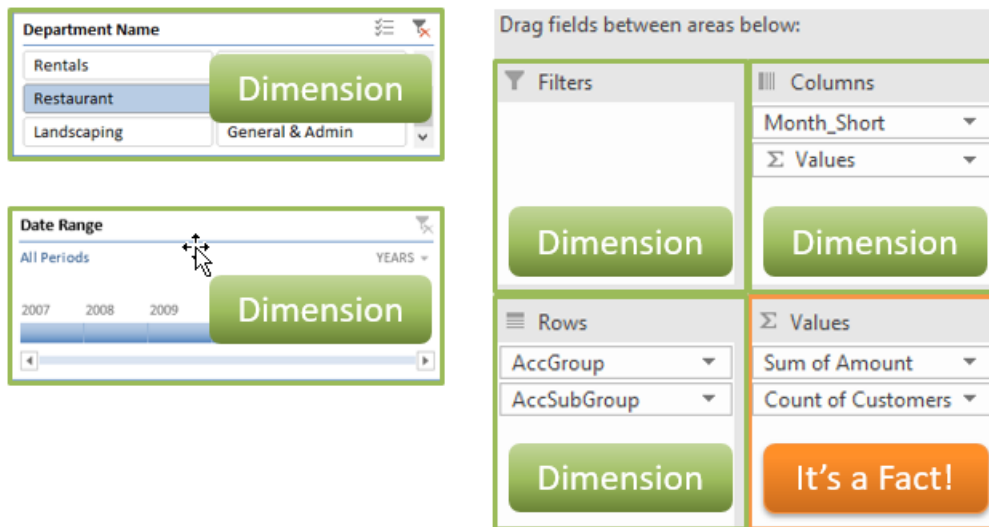


Figure 6 - Identifying Fact and Dimension fields

If the field will live in the Values area of the PivotTable, it's a Fact. If it will live anywhere else in the PivotTable's context (on Rows, Columns, Filters, or in Slicers or Timelines) then it's a Dimension.



## Tip 1: Hiding Fields from a User

One of the biggest challenges I see when people are first getting started with Power Pivot is that they inadvertently create relationship issues in their PivotTables (or visuals in Power BI) by pulling fields from the wrong area.

To explain this issue, I created two individual PivotTables using the configuration as follows:

### Sales Table

- Rows: Sales[Date]
- Column: Sales[Product]
- Values: [Sales \$]

### Budget Table

- Rows: Budget[Date]
- Column: Budget[Product]
- Values: [Budget \$]

And the results are shown in Figure 7:

Sales Table					Budget Table				
Sales \$	Product		Total		Budget \$	Product		Total	
Date	A	B	C	Total	Date	A	B	C	Total
2017-01-31	2,367	4,872	3,831	11,070	2017-01-31	1,000	5,000	1,000	7,000
2017-02-28	4,098	4,040	4,182	12,320	2017-02-28	3,000	3,000	3,000	9,000
2017-03-31	3,305	3,490	2,238	9,033	2017-03-31	3,000	4,000	5,000	12,000
2017-04-30	1,790	2,610	1,688	6,088	2017-04-30	3,000	3,000	5,000	11,000
<b>Total</b>	<b>11,560</b>	<b>15,012</b>	<b>11,939</b>	<b>38,511</b>	<b>Total</b>	<b>10,000</b>	<b>15,000</b>	<b>14,000</b>	<b>39,000</b>

Figure 7 - Sales and Budget PivotTables

Everything looks just fine here. The numbers slice up quite nicely, and filter as they should.

But now look what happens when we add [Budget \$] to the Values area of the Sales PivotTable:

Date	A	B		C		Total Sales \$	Total Budget \$
Date	Sales \$	Budget \$	Sales \$	Budget \$	Sales \$	Budget \$	
2017-01-31	2,367	39,000	4,872	39,000	3,831	39,000	11,070
2017-02-28	4,098	39,000	4,040	39,000	4,182	39,000	12,320
2017-03-31	3,305	39,000	3,490	39,000	2,238	39,000	9,033
2017-04-30	1,790	39,000	2,610	39,000	1,688	39,000	6,088
<b>Total</b>	<b>11,560</b>	<b>39,000</b>	<b>15,012</b>	<b>39,000</b>	<b>11,939</b>	<b>39,000</b>	<b>38,511</b>

Figure 8 - A PivotTable with relationship problems

Plainly, the values for [Budget \$] shown in Figure 8 are not correct. In fact, if you look closely, you'll see that the 39,000 being used on every row and column is the grand total from the original budget table. So why is this happening?





The challenge is that this PivotTable pulls its dimensions (the Date and Product) from the Sales table. And while the [Sales \$] measure can understand these (it aggregates the [Amount] column from the Sales table), the [Budget \$] measure cannot.

To understand why, let's look at Figure 9, shown below.

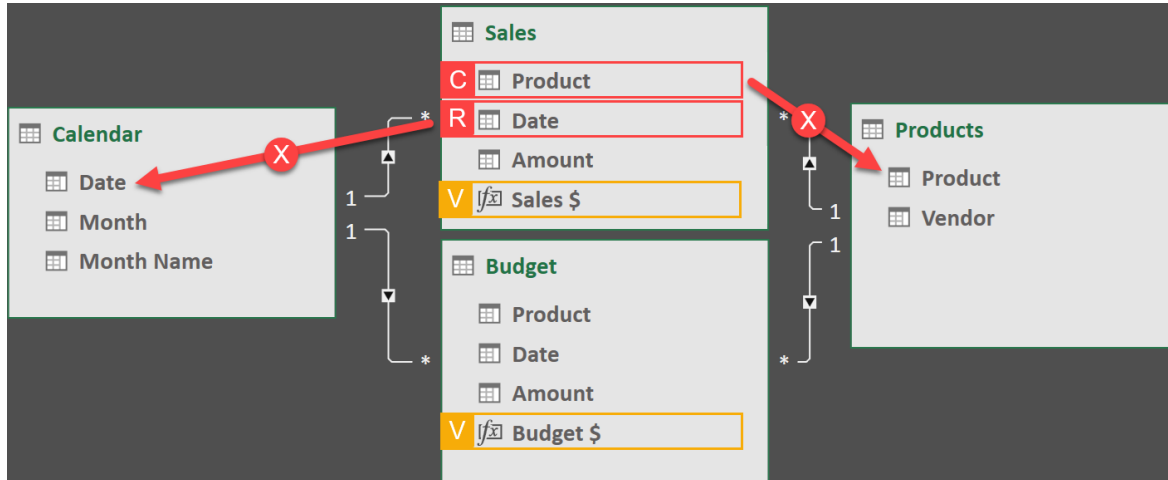


Figure 9 - Poor field choices for a PivotTable

Looking at this, you might think, “What’s the issue?” We know that there is a link between the Calendar[Date] and the Sales[Date], so why can’t a field from the Sales table filter the Calendar table, which in turn filters the Budget table? And likewise, with Products?

If you look at the relationship lines between the tables, you’ll notice that they have arrows on them. They’re similar to one-way streets, and filter context refuses to drive the wrong way down that street! (In short, the filter context will not be passed against the direction of the arrow. Since it can’t get from the Sales table to the dimension, it can’t be passed on to the Budget table from the dimension.)

To avoid a cross filtering problem the PivotTable needs to be configured like this:

- Rows: Calendar[Date]
- Column: Products[Product]
- Values: [Sales \$], [Budget \$]

In other words, we wanted to use the fields shown in Figure 10, below when configuring the PivotTable.



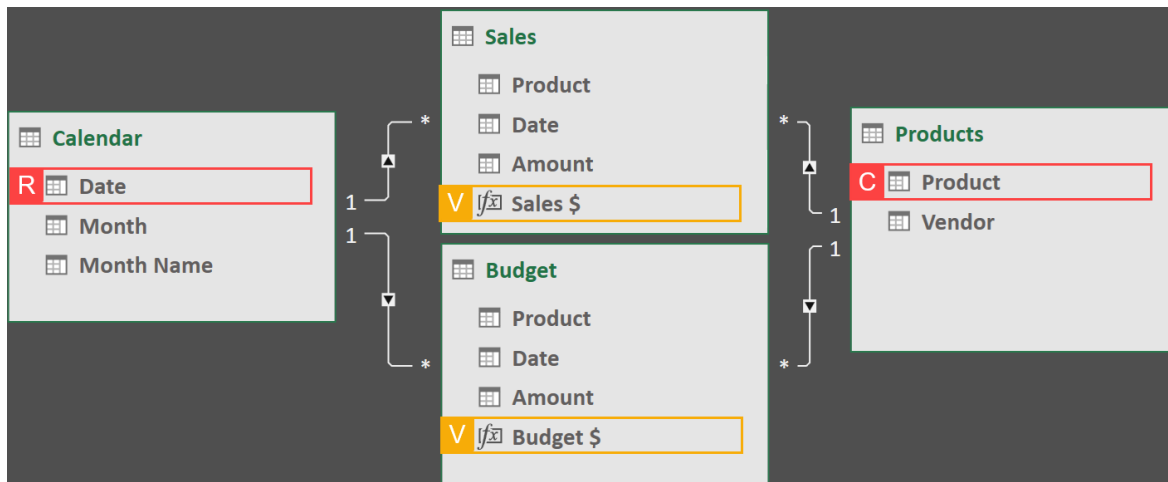


Figure 10 - Proper field choices for a PivotTable

And when we make this modification, pulling the dimensional fields from the Dimension tables (not the Fact tables) the PivotTable will work beautifully, as shown in Figure 11:

	A		B		C		Total Sales \$	Total Budget \$
Date	Sales \$	Budget \$	Sales \$	Budget \$	Sales \$	Budget \$		
2017-01-31	2,367	1,000	4,872	5,000	3,831	1,000	11,070	7,000
2017-02-28	4,098	3,000	4,040	3,000	4,182	3,000	12,320	9,000
2017-03-31	3,305	3,000	3,490	4,000	2,238	5,000	9,033	12,000
2017-04-30	1,790	3,000	2,610	3,000	1,688	5,000	6,088	11,000
<b>Total</b>	<b>11,560</b>	<b>10,000</b>	<b>15,012</b>	<b>15,000</b>	<b>11,939</b>	<b>14,000</b>	<b>38,511</b>	<b>39,000</b>

Figure 11 - A properly functional PivotTable

Truth be told, there is absolutely no reason that this setup should ever have been created this way. In addition, if you follow this simple tip, it will never happen to you.

In order to link these two tables, we had to either create or import a Dimension table that allows a One-to-Many relationship between the Dimension and Fact tables. It's this relationship that allows the filtering in the first place.

My advice to you is simple: when you create the relationship, immediately hide the field on the "many" side of the relationship (the side with the star). To do this, you simply go into Design View in Power Pivot, right click the field on the many side of the relationship and choose "Hide from Client Tools". It will shade the column slightly, to indicate that it's been successful.

If you have a look at Figure 12 below, you'll see that I've hidden the field on the many side of every relationship, as well as other fields that I don't want the user to see. Those include fields that are only in the model to sort other columns, as well as un-aggregated columns.



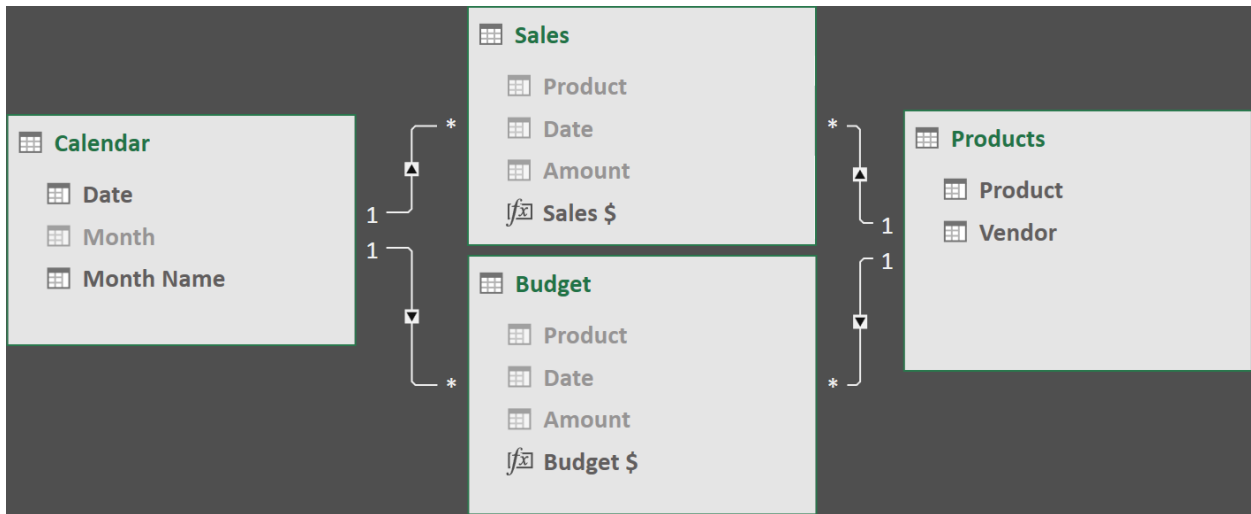


Figure 12 - The Power Pivot model with hidden fields

The question you may be asking now is, “Why? What purpose does this serve?”

Figure 13 shows the effect of making this change. And there are a couple of really cool things here that you should notice:

1. It is now impossible to put the wrong field into a row, column, slicer or timeline. They’re hidden from the model, so a user cannot get it wrong. That’s our primary gain here.
2. By hiding everything except defined measures from my Fact table, the icon changes to the  $\Sigma$  character. This is by design, as these tables contain the fields that should go into the values area of the PivotTable, which also carries the  $\Sigma$  character!

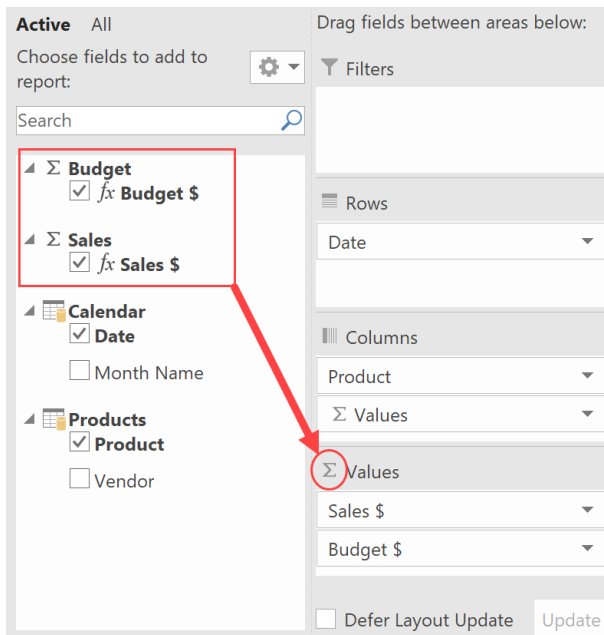


Figure 13 - Hidden fields in play



To be completely honest, I actually wish that Power Pivot (and Power BI Desktop) would automatically hide the “many” side of the relationship upon linking. You can always unhide it, but this would prevent a lot of people from getting this wrong.

If you, like me, think that it would be a great idea for Excel and Power BI to automatically hide the many side when you create a relationship, help me raise this issue with Microsoft. You can [vote for this idea here](#), and hopefully it will eventually save us having to do this manually.



## Tip 2: Hiding Zeros in a Measure

One of the greatest things about PivotTables and Power BI visuals is that they suppress values if they are blank. And that's key, if they are blank. Remember that 0 and blank are very far from the same thing. Take the following example...

In early April, the company purchaser orders products from a new vendor called Delta and begins to sell them. A client purchases one of these products and then, shortly after, they return it as they are unhappy with the quality. A credit note is then issued and posted, making the net of the Sales transaction equal to \$0.00.

A PivotTable is then created with the following configuration:

- Rows: Products[Vendor]
- Column: Calendar[Month]
- Values: [Sales \$]

All is fine with the PivotTable in March, but when it is extended to also show April's sales, it ends up with an unnecessary row:

Sales \$	Column		
Row Labels	Jan	Feb	Mar
Acme	2,367	4,098	3,305
Boingo	4,872	4,040	0
Caltrop	3,831	4,182	2,238

Figure 14 - The March PivotTable suppresses the new vendor

Sales \$	Column			
Row Labels	Jan	Feb	Mar	Apr
Acme	2,367	4,098	3,305	1,790
Boingo	4,872	4,040	0	2,610
Caltrop	3,831	4,182	2,238	1,688
Delta				0

Figure 15 - The April PivotTable includes the new vendor

So how do we fix this? There are actually two steps to it.

### Step 1: Exploiting the BLANK() function

The first step is to take the existing measure and test its value. If the value is equal to zero, rather than show a zero, we need to take advantage of Power Pivot's BLANK() function to return a blank value. Why? Because blanks are suppressed from the PivotTable!

Here's the revised [Sales \$] measure in this case:

```
Sales $: =IF(
    SUM(Sales[Amount])=0,
    BLANK(),
    SUM(Sales[Amount])
)
```



By changing out our function to use this method, the PivotTable suppresses the Delta row, even when April is showing on the PivotTable, as displayed in Figure 16:

Sales \$	Column			
Row Labels	Jan	Feb	Mar	Apr
Acme	2,367	4,098	3,305	1,790
Boingo	4,872	4,040		2,610
Caltrop	3,831	4,182	2,238	1,688

Figure 16 - The April PivotTable now suppresses all zeros

Even though it doesn't look like much, I've met many pros who use an IF test to return a value or 0. This can end up calculating a 0 for each item in your Dimension table. If you want to suppress values, use BLANK() instead, you'll end up way happier.

### Step 2: Make the PivotTable fill BLANK() with 0

So as cool as the last trick was, it makes our PivotTable appear as if it has a hole in it since we've suppressed the March value of zero. To fix that, we actually change the property of the PivotTable.

- Right click the PivotTable → PivotTable Options
- Next to "For Empty Cells Show", enter a 0

At this point the PivotTable will correctly show a zero again, but still suppresses the product that is zero all the way across.

Sales \$	Column			
Row Labels	Jan	Feb	Mar	Apr
Acme	2,367	4,098	3,305	1,790
Boingo	4,872	4,040	0	2,610
Caltrop	3,831	4,182	2,238	1,688

Figure 17 - The final PivotTable

Something that you should also be aware of here: by using the feature to fill cells with 0, we don't necessarily place a zero in the cell. It could be 0.00, or even the – (dash) character. It completely depends upon the number format that you've applied to your PivotTable.



### Tip 3: DAX Variables

Let's assume time has marched forward. We've logged sales for May into our source data, and we've added budgets as far out as June based on the updated tables shown here:

	Product	Date	Amount
1	A	2017-01-31 12:00:00 AM	2367
2	B	2017-01-31 12:00:00 AM	4872
3	C	2017-01-31 12:00:00 AM	3831
4	A	2017-02-28 12:00:00 AM	4098
5	B	2017-02-28 12:00:00 AM	4040
6	C	2017-02-28 12:00:00 AM	4182
7	A	2017-03-31 12:00:00 AM	3305
8	B	2017-03-31 12:00:00 AM	0
9	C	2017-03-31 12:00:00 AM	2238
10	A	2017-04-30 12:00:00 AM	1790
11	B	2017-04-30 12:00:00 AM	2610
12	C	2017-04-30 12:00:00 AM	1688
13	D	2017-04-30 12:00:00 AM	0
14	A	2017-05-31 12:00:00 AM	3184
15	B	2017-05-31 12:00:00 AM	3524
16	C	2017-05-31 12:00:00 AM	2467
17	A	2017-06-30 12:00:00 AM	
18	B	2017-06-30 12:00:00 AM	
19	C	2017-06-30 12:00:00 AM	

	Product	Date	Amount
1	A	2017-01-31 12:00:00 AM	1000
2	B	2017-01-31 12:00:00 AM	5000
3	C	2017-01-31 12:00:00 AM	1000
4	A	2017-02-28 12:00:00 AM	3000
5	B	2017-02-28 12:00:00 AM	3000
6	C	2017-02-28 12:00:00 AM	3000
7	A	2017-03-31 12:00:00 AM	3000
8	B	2017-03-31 12:00:00 AM	4000
9	C	2017-03-31 12:00:00 AM	5000
10	A	2017-04-30 12:00:00 AM	3000
11	B	2017-04-30 12:00:00 AM	3000
12	C	2017-04-30 12:00:00 AM	5000
13	D	2017-04-30 12:00:00 AM	500
14	A	2017-05-31 12:00:00 AM	1000
15	B	2017-05-31 12:00:00 AM	5000
16	C	2017-05-31 12:00:00 AM	1000
17	A	2017-06-30 12:00:00 AM	3000
18	B	2017-06-30 12:00:00 AM	3000
19	C	2017-06-30 12:00:00 AM	3000

Figure 18 - The updated Sales table (left) and Budgets table (right)

Wouldn't it be cool if you could build a little PivotTable which would show the actual revenue up to the date you selected in a slicer, and the budgets after that date? Something like this:

**Show Forecast After:**

Feb 2017 MONTHS ▾

2017

JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC

	Actual	Actual	Budget	Budget	Budget	Budget	
Row Labels	Sales \$ (Jan)	Sales \$ (Feb)	Sales \$ (Mar)	Sales \$ (Apr)	Sales \$ (May)	Sales \$ (Jun)	Sales \$ (Forecast)
Acme	2,367	4,098	3,000	3,000	1,000	3,000	16,465
Boingo	4,872	4,040	4,000	3,000	5,000	3,000	23,912
Caltrop	3,831	4,182	5,000	5,000	1,000	3,000	22,013
Delta	0	0	0	500	0	0	500

Figure 19 - Showing Actuals or Budgets at the click of a Timeline

Now naturally you can do this. To do this on an annual basis, the trick is to create 13 individual DAX measures (one for each month plus the total). It takes a little work, but there are only really three different patterns in use: a formula for the first month, another formula pattern that is reused for the second through final months, and a final formula to add the previous columns.



The measures, as authored in Excel 2010 or 2013, might look like this:

### Formula for the [Sales \$ (Jan)] measure:

```
=CALCULATE([Sales $],ALL('Calendar'),'Calendar'[Month]=1)
```

This is a relatively straightforward measure that removes all the filters from the Calendar table, then resets to only the month of January, then applies that modified filter context to the Sales \$ measure.

### Formulas for the monthly measures, using [Sales \$ (Feb)] as an example:

```
=IF(  
    MAX('Calendar'[Month])>=2,  
    CALCULATE([Sales $],ALL('Calendar'),'Calendar'[Month]=2),  
    CALCULATE([Budget $],ALL('Calendar'),'Calendar'[Month]=2)  
)
```

The purpose of this pattern is to allow us to react to a slicer's selection in order to decide whether to return the Actual or Budget. The IF() statement begins by checking the maximum month that is returned by the filter context (timeline selection). If it's greater than or equal to 2, then the Sales \$ measure should be shown. If it's not (i.e. it's January), then we'll show the Budget.

March through December will use the same formula, only with a different month number in place.

But here's the challenge: in order to create these formulas, you need to update the month end date in three places. As it's tedious work, wouldn't it be nicer if we only had to do it once?

### Formula for the [Sales \$ (Forecast)] Measure:

Finally, as the individual months do all the hard work, this simple formula can just add the appropriate Actual and Budget amounts together to return a forecasted total:

```
= [Sales $ (Jan)]+[Sales $ (Feb)]+[Sales $ (Mar)]+[Sales $ (Apr)]+[Sales $ (May)]+[Sales $ (Jun)]
```

### The Power of DAX Variables

Starting in Excel 2016, the Power Pivot team added variables to the DAX programming language. The same feature also exists in Power BI, but unfortunately won't be ported back to earlier versions of Excel.

The concept of a DAX variable is that it allows you to define a value at the beginning of the code that you can re-use elsewhere. This makes it much easier to copy and paste patterns which require minor modification to work.

To implement a variable, we preface our regular function with the following:

```
=  
  
VAR  
    <Variable Name> = DAX Formula  
  
RETURN  
    <The original DAX formula>
```





You can have as many variables as you want to create, and then you can use them in place of any values in the original DAX formula. The [Sales \$ (Feb)] measure, for example, would look like this:

```
=VAR MthEnd=2
RETURN
  IF (
    MAX('Calendar'[Month])>=MthEnd,
    CALCULATE([Sales $],ALL('Calendar'), 'Calendar'[Month]=MthEnd),
    CALCULATE([Budget $],ALL('Calendar'), 'Calendar'[Month]=MthEnd)
  )
```

In this case, we have created a variable called MthEnd, assigned it a value of 2, and then we call that variable throughout the function. The benefit is that we only have to declare the value once, and it will be used in all three places, making it much less likely to create an error.

We can also rewrite the Forecast \$ measure to make use of these variables as well:

```
=VAR
  MthEnd=MAX('Calendar'[Month])
RETURN
  CALCULATE (
    [Sales $],
    ALL('Calendar'),
    'Calendar'[Month]<=MthEnd
  )
+ CALCULATE (
  [Budget $],
  ALL('Calendar'),
  'Calendar'[Month]>MthEnd
)
```

The benefit of the DAX variable here is that we can evaluate the final month, returning a single value. That value can then be passed into the CALCULATE() statement, meaning that it doesn't require us to include an additional FILTER() function. (If you tried to use MAX('Calendar'[Month]) in place of the MthEnd variables, you would need to do so.) In addition, as this runs only two calculate formulae, it should perform much faster than adding each column as they each run 6.



## Tip 4: Retrieve a Value from an Excel Slicer

If you work with slicers and timelines in Excel at all, you've probably wanted to return the value of your slicer to a cell so that you can use it in a formula. There are a couple of ways to do this:

### Version 1: Standard (non-Power) PivotTable

The easiest way to do this for a non-power PivotTable is to simply:

- Create a new PivotTable on a new worksheet
- Place the field you are using in your slicer into the Filters area
- Select your slicer and go to Slicer Tools → Report Connections
- Link the existing slicer to your new PivotTable

Why put it on a different PivotTable than your original. Because it means you don't need to mess up your current design. This will be very important to you if you need this field shown elsewhere on your PivotTable, as you can only have a field on the table once. Keep in mind that this new PivotTable can even be hidden on a different worksheet if you like.

Overall, it's super easy, and you can now point a formula against the filter field to read the value in the slicer. The only caveat here is that you can only get one value at a time, even if there are multiple items showing:

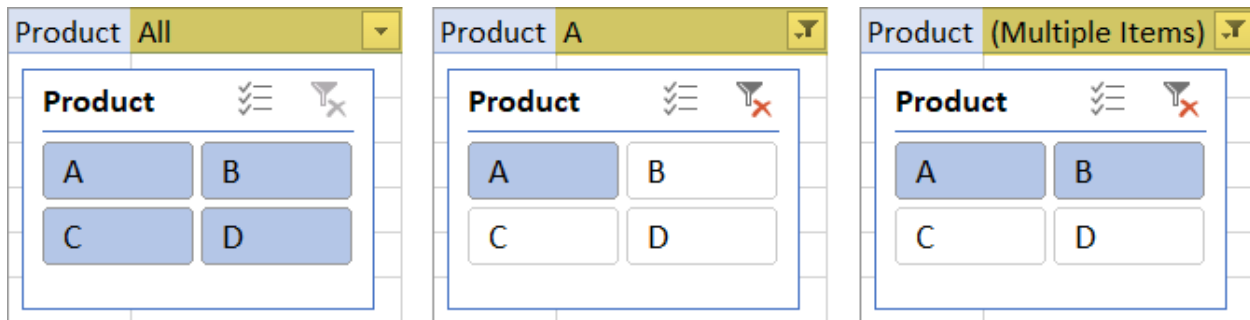


Figure 20 - We can reference the PivotTable's Filter field using a formula

The one that kills me here is the Multiple Items piece, as you can't easily pull out the first/last item, or a list of the items. But we can do either with DAX.

### Version 2: Retrieving the First/Last Selection from a Slicer

Let's assume that we want to build upon our last solution to drive the Actual and Budget showing above each column of the PivotTable by formulae, based on the selection in the timeline. How do we do it?



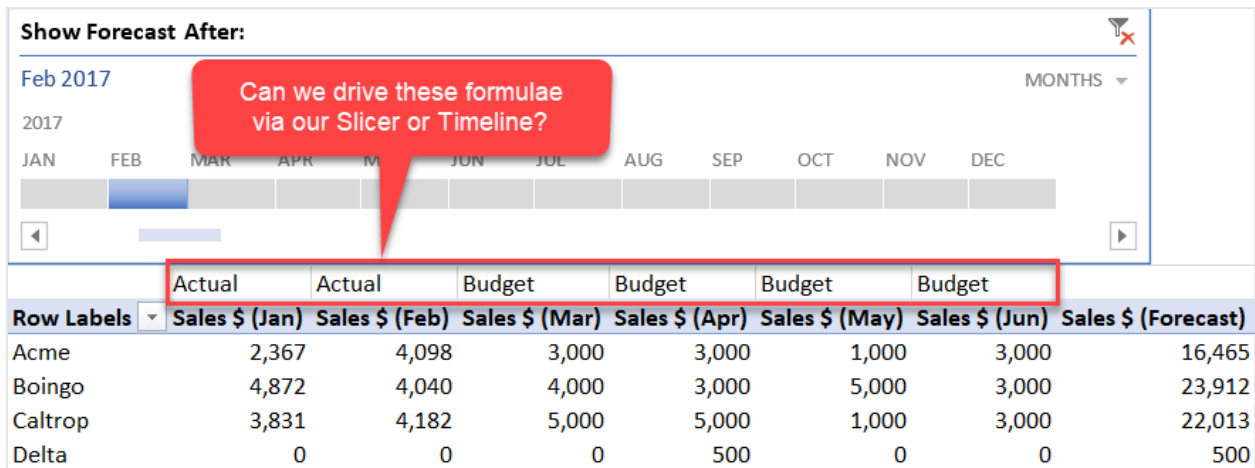


Figure 21 - Showing Actuals or Budgets at the click of a Timeline

The first step is that we need to create a measure to work out the month number that is selected. We'll call that measure MaxMonth and use the following formula to create it:

```
=CALCULATE (
    MAX ('Calendar' [Month]),
    LASTDATE ('Calendar' [Date])
)
```

That should give us the ability to work out a single value representing the month number for the last date selected on the timeline, even if multiple months are chosen.

Next, we need to create a new PivotTable which will hold only the MaxMonth measure:

- Create a new PivotTable
- Place the MaxMonth measure in the Values area

The next step is to link the Timeline to this PivotTable:

- Select the Timeline that is linked to your existing PivotTable.
- Go to Timeline Tools → Report Connections → Select the PivotTable that holds MaxMonth

At this point, you want to make sure that MaxMonth is working correctly with the new Pivot and that the value is changing. In the case of the sample model, it would look as follows:

MaxMonth
2

Figure 22 - The linked PivotTable holding the MaxMonth measure



It's now time to make some magic...

- Select one of the cells in this PivotTable
- Go to PivotTable Tools → Analyze → OLAP Tools → Convert to Formulas

If you now go and select the cell that holds the MaxMonth value, you'll see that you have a formula that reads as follows:

```
=CUBEVALUE (  
    "ThisWorkbookDataModel",  
    <cell reference>,  
    <timeline name>  
)
```

If and you check the cell that is referenced in the middle, you'll see that it also contains a formula like this:

```
=CUBEMEMBER ("ThisWorkbookDataModel", "[Measures].[MaxMonth] ")
```

We can consolidate this by copying the formula in the referenced cell and pasting it into the formula that was being used to display the MaxMonth value as follows:

```
=CUBEVALUE (  
    "ThisWorkbookDataModel",  
    CUBEMEMBER ("ThisWorkbookDataModel", "[Measures].[MaxMonth] ") ,  
    <timeline name>  
)
```

We've now got a formula that will return the month number for the highest month selected in the timeline, even if multiple months are contained in that selection. The beauty here is that you don't need the whole PivotTable to return this single value. You can do it via a single formula that you can now just hide in a cell behind the timeline!

From here it's just a matter of building a formula for each column's header to see if the month selected in the timeline is greater than the month the column is representing. So, assuming we placed this formula in cell C1, the Excel formulas could be as simple as:

```
B2: =IF($C$1>1, "Budget", "Actual")  
C2: =IF($C$1>2, "Budget", "Actual")
```

And as you can see it works a treat:



	A	B	C	D	E	F	G	H
1	<b>Show Forecast After:</b> Mar 2017 MONTHS ▾ 2017 JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC							
2		Actual	Actual	Actual	Budget	Budget	Budget	
3	Row Labels ▾	Sales \$ (Jan)	Sales \$ (Feb)	Sales \$ (Mar)	Sales \$ (Apr)	Sales \$ (May)	Sales \$ (Jun)	Sales \$ (Forecast)
4	Acme	2,367	4,098	3,305	3,000	1,000	3,000	16,770
5	Boingo	4,872	4,040	0	3,000	5,000	3,000	19,912
6	Caltrop	3,831	4,182	2,238	5,000	1,000	3,000	19,251
7	Delta	0	0	0	500	0	0	500

Figure 23 - March's column header responding to the Timeline selection

### Version 3: Retrieving All Selected Values from a Slicer

You already know how to pull the values from a slicer using either the PivotTable filter or a formula based on the tricks above. The magic behind this trick is really all in the DAX.

Assuming that you'd like a measure to return all the Product names from this model, we'd build a Selected Products measure that looks like this:

```
=CONCATENATEX (
    VALUES (Products [Product]),
    Products [Product],
    ", "
)
```

And that's it really. Drop it into the Values area of a separate PivotTable and link your formula to that, (as shown in Version 1 above).

Product (Multiple Items) ▾	Selected Products
Product <input checked="" type="checkbox"/> A <input type="checkbox"/> B <input checked="" type="checkbox"/> C <input type="checkbox"/> D	A,C

Figure 24 - Our new measure returns the actual items selected, not just "Multiple Items"

And if you'd then prefer the formula version, just convert the new PivotTable to values (as shown in Version 2) in order to generate the CUBE function, you need to call it directly. (Of course, if you're really good, you can just write the CUBE function without even creating the PivotTable first!)



## Tip 5: One Field, Multiple Slicers, Different Views

For our final tip, I thought it might be cool so show you how you can use slicers to drive comparisons between selected values. Wait... is that a trick? I'll let you determine this. Have a look at this scenario:

Compare these...		against these...	
A	B	A	B
Month	Product(s) A	Product(s) C	Difference
Jan	2,367	3831	-1,464
Feb	4,098	4182	-84
Mar	3,305	2238	1,067
Apr	1,790	1688	102
May	3,184	2467	717
<b>Grand Total</b>	<b>14,744</b>	<b>14406</b>	<b>338</b>

Figure 25 - Comparing Product A vs Product C

No big deal, right? We can just create a measure for Product A, another measure for Product C and put them both on the PivotTable. Or can we? What happens when you have your Product A measure on the PivotTable and the right-hand slicer is attached to that table? Unless you've got an ALL() nested in Product A's measure, it's going to remove all Product A values from the table.

What if we change the slicer selection?

Compare these...		against these...	
A	B	A	B
Month	Product(s) A	Product(s) B,C	Difference
Jan	2,367	8703	-6,336
Feb	4,098	8222	-4,124
Mar	3,305	2238	1,067
Apr	1,790	4298	-2,508
May	3,184	5991	-2,807
<b>Grand Total</b>	<b>14,744</b>	<b>29452</b>	<b>-14,708</b>

Figure 26 - Changing the slicer selection to compare product A vs products B and C

Do you see this? We've got two slicers that are independently driving two different measure columns. The first thing you must be wondering is how we can even get two slicers hooked to the same PivotTable that have different selections in them, as slicers stay in sync with each other.

Of course, these are different fields, but even at that, how is the first selection not removing the rows from the underlying table that would be needed to calculate these?

The secret to this requires a minor change to the structure of the data model, two new relationships, and a bit of DAX measure craftiness.

03

POWER PIVOT



## Step 1: Add new tables

Before we can do anything else, we need separate tables to drive each of our slicers. To do that I quickly created some new tables using Power Query via the following steps:

- Reference the Products query
- Right click the Product column → Remove Other Columns
- Right click the Product column → Remove Duplicates
- Rename the new table to Product1
- Load it to the Data Model


I then duplicated this new Products1 query:

- Duplicate the Products2 query
- Change the name to Product2
- Load it to the Data Model

This gave me two new short tables which each look like this:

	Product
1	A
2	B
3	C
4	D

Figure 27 - One of the new Product tables



**Extend Your Learning**  
Do you want to truly Master Your Data? You need **Power Query Academy!**  
Get started with our FREE [Power Query Fundamentals](#) course.

## Step 2: Relate the tables

This is the tricky part. Because the Products table and each of the two new tables have unique items in the Product column, it is possible that Power Pivot could get this relationship backwards. It's also critical that we end up with an "inactive" relationship, as an active relationship will remove critical records that we need to summarize.

To that end, we need to make sure of two things when creating these relationships:

1. The relationships are pointing in the correct direction (the arrow should point towards the Products table unless you are using Excel 2010/2013 where it goes the other way).
2. The relationship uses a dotted line (indicating that it is inactive) as shown below:



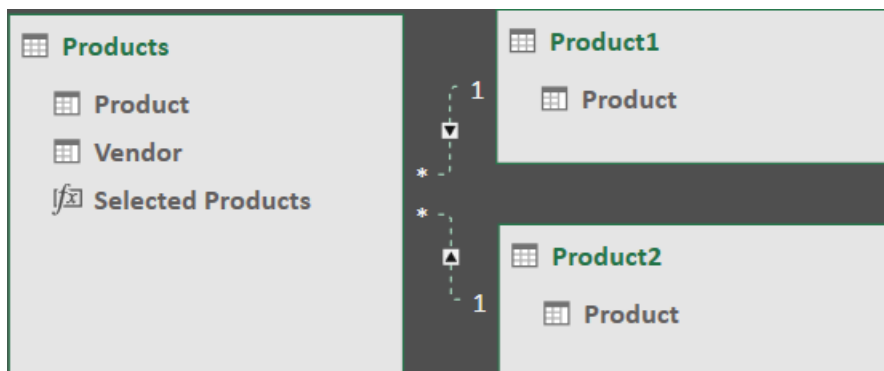


Figure 28 - The Product1 and Product2 tables linked into the data model

So how do we do this?

I would not rely on the drag and drop interface here to create the relationships. To get this correct, I would recommend the following steps:

- Right click Products[Product] → Create Relationship
- Choose Product1 as the other table
- Uncheck the “Active” box at the bottom of the relationships window

When you click OK, check that the relationship arrow is flowing the right way and that it is using a dotted line.

- If the direction is correct but the line is solid: Double click the line → uncheck the Active box → click OK
- If the direction is backwards: Select the line → press Delete → try again changing the order you selected the fields

Make sure you create both relationships before you drop back to Excel!

### Step 3: Create your PivotTable and Slicers

With the relationships created, we are getting closer, but we still aren't quite there yet. Let's create a PivotTable using the following configuration:

Rows: Calendar[Month]

Values: Sales[Sales \$]

And two slicers:

Product1[Product] and change the caption to “Compare these...”

Product2[Product] and change the caption to “against these...”

And the output should now look similar to what you see here:





	A	B	C	D
1	Compare these...		against these...	
2				
3	A	B	C	D
4				
5	<b>Row Labels</b>	<b>Sales \$</b>		
7	Jan	11070		
8	Feb	12,320		
9	Mar	5,543		
10	Apr	6,088		
11	May	9,175		
12	<b>Grand Total</b>	<b>44,196</b>		

Figure 29 - The new PivotTable with "disconnected" slicers

The key thing to notice here? The slicers have been filtered but have no effect on the measure in the PivotTable at all.

#### Step 4: Creating the DAX measures

Now it's time to create a little DAX. In this case, we're going to write a CALCULATE() statement that will override the relationship in use. We'll need one for each comparison case.

Our first measure will take our existing [Sales \$] measure and instruct Power Pivot to use the inactive relationship between Products[Product] and Product1[Product] as shown here:

```
Product 1 Sales $:
=CALCULATE (
    [Sales $],
    USERRELATIONSHIP (Products [Product], Product1 [Product])
)
```

The second measure is virtually identical, except that it uses the relationship between the Products and Product 2 table (instead of Product1):

```
Product 2 Sales $:
=CALCULATE (
    [Sales $],
    USERRELATIONSHIP (Products [Product], Product2 [Product])
)
```



Finally, in order to work out a variance between the two, we'll also add one more measure:

```
Sales $ Var - 1 vs 2: = [Product 1 Sales $]-[Product 2 Sales $]
```

With those measures created, you can drop them on the PivotTable, and you'll see that these new measures do react to our slicers:

Compare these...		against these...					
A	B	C	D	A	B	C	D
Row Labels	Sales \$	Product 1 Sales \$	Product 2 Sales \$	Sales \$ Var - 1 vs 2			
Jan	11,070	2,367	4,872	-2,505			
Feb	12,320	4,098	4,040	58			
Mar	5,543	3,305	0	3,305			
Apr	6,088	1,790	2,610	-820			
May	9,175	3,184	3,524	-340			
<b>Grand Total</b>	<b>44,196</b>	<b>14,744</b>	<b>15,046</b>	<b>-302</b>			

Figure 30 - Measures iterating over inactive relationships

Notice that [Product 1 Sales \$] selections will update every time you select one or more values from the “Compare these...” slicer. [Product 2 Sales \$] will update using the “against these...” slicer, and [Sales \$] will remain unchanged since the relationships used to link these tables to the model are inactive. (If one of those relationships was active, it would filter the entire model!)

### Step 5: The final polish

I love this trick as it allows my clients to choose which fields to compare. But the next question that comes up is: if they select multiple items, how can we get those items listed on the report? As it happens, we've already seen that trick in action in Tip 4: Retrieve a Value from an Excel Slicer. We just need to make a small modification (or rather addition) to our measure.

Remember this measure?

```
Selected Products:
=CONCATENATEX (
    VALUES (Products [Product]),
    Products [Product],
    ", "
)
```

We just need to create two measures which will force our new relationships on it.



Here is the measure I used to return the selected products from the first slicer:

```
Compare X:
=CALCULATE (
    [Selected Products],
    USERRELATIONSHIP (Products [Product], Product1 [Product])
)
```

And the measure for the second slicer:

```
Compare Y:
=CALCULATE (
    [Selected Products],
    USERRELATIONSHIP (Products [Product], Product2 [Product])
)
```

I then created a new PivotTable and linked both the slicers as shown here:

	Compare X	Compare Y		
	A	B,C		
	Compare these...	against these...		
	A B C D	A B C D		
Mar	3,305	2,238	1,067	
Apr	1,790	4,298	-2,508	
May	3,184	5,991	-2,807	
<b>Grand Total</b>	<b>14,744</b>	<b>29,452</b>	<b>-14,708</b>	

Figure 31 - Returning multiple slicer selections to a PivotTable for later use

In this case I didn't even bother to convert it to the formulaic equivalent. I only did a small bit of creative formula work to write up some dynamic headers (joining text to the body area cells), put the slicers on top of the PivotTable to hide it, and hid the header row of the PivotTable as shown below:



Compare these...		against these...	
A	B	A	B
Month	Product(s) A	Product(s) B,C	Difference
Jan	2,367	8,703	-6,336
Feb	4,098	8,222	-4,124
Mar	3,305	2,238	1,067
Apr	1,790	4,298	-2,508
May	3,184	5,991	-2,807
<b>Grand Total</b>	<b>14,744</b>	<b>29,452</b>	<b>-14,708</b>

Figure 32 - The final output using slicers to compare products

Job done, looks great and comparisons can be performed with an easy-to-use interface.

The only thing left to do is return to the Power Pivot Data Model and hide the Product1 and Product2 tables as well as the [Compare X] and [Compare Y] measures on the Products table. Not only will that prevent anyone from accidentally using the wrong fields when they build their own PivotTable, but it makes the solution look like pure magic.

Naturally this technique can be extended to other types of data as well. The first time I ever used this technique was actually to give my client the ability to compare different years against each other and they LOVED it!





# Extend Your Learning

Our Skillwave Training learning platform provides the BEST online video training created by world-class experts:

[View Course Catalogue](#)

**Power Query Academy**  
Intermediate to Advanced

**Excel Fundamentals**  
Boot Camp

**Self Service BI Boot Camp**  
for Excel and Power BI

**Dimensional Modeling**  
for the Excel & Power BI Pro

**Building Financial Statements**  
in Excel with Power Pivot

**Power Query Academy**  
Fundamentals

## Get started for FREE

Sign up for our FREE [Power Query Fundamentals course](#) and start becoming a power user of Excel and Power BI today!





## 'DIY BI' Tips, Tricks & Techniques

©2022 Excelguru Consulting Inc.

Don't miss out on the other books in this series!

Register for our newsletter today at

<http://xlguru.ca/newsletter>.

