Takaisin välilehdelle

Tee: Käy oppitunti läpi loppuun asti

Tasks

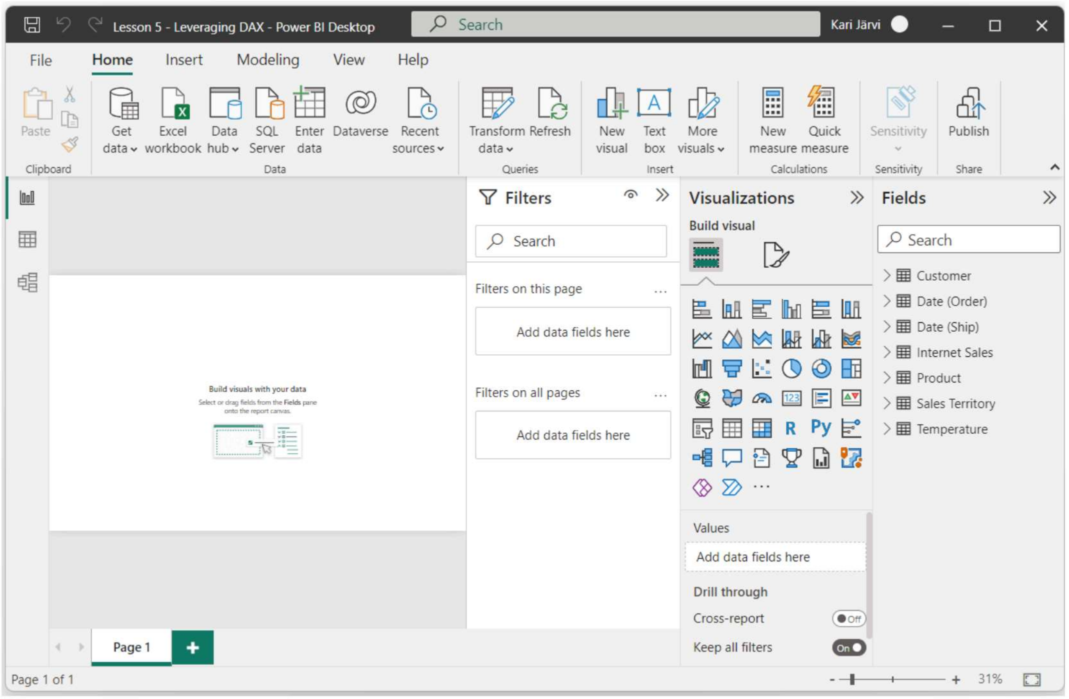
In this exercise, you will learn how to create **calculated columns** in Power BI using DAX. The building of calculated columns is a great way of extending the analytical capability of Power BI and by the end of this chapter, you will feel very comfortable with creating new columns through DAX. The writing of calculated columns logically occurs after the data model has been developed.

Task 1 - Download and open the Lesson 5 .pbix file

Step 1: Click link to download **Lesson 5 – Leveraging DAX.pbix**, and save it to **C:\PBExams**.

Step 2: Open **Power BI Desktop**. Click **File**, and select **Open Report**.

Step 3: Browse to *C:\PBExams*, and select *Lesson 5 – Leveraging DAX.pbix*.



Calculated columns are stored in the table in which they are assigned, and the values are static until the data is refreshed. You will learn more about refreshing data in a later chapter.

There are many use cases for calculated columns, but the two most common are as follows:

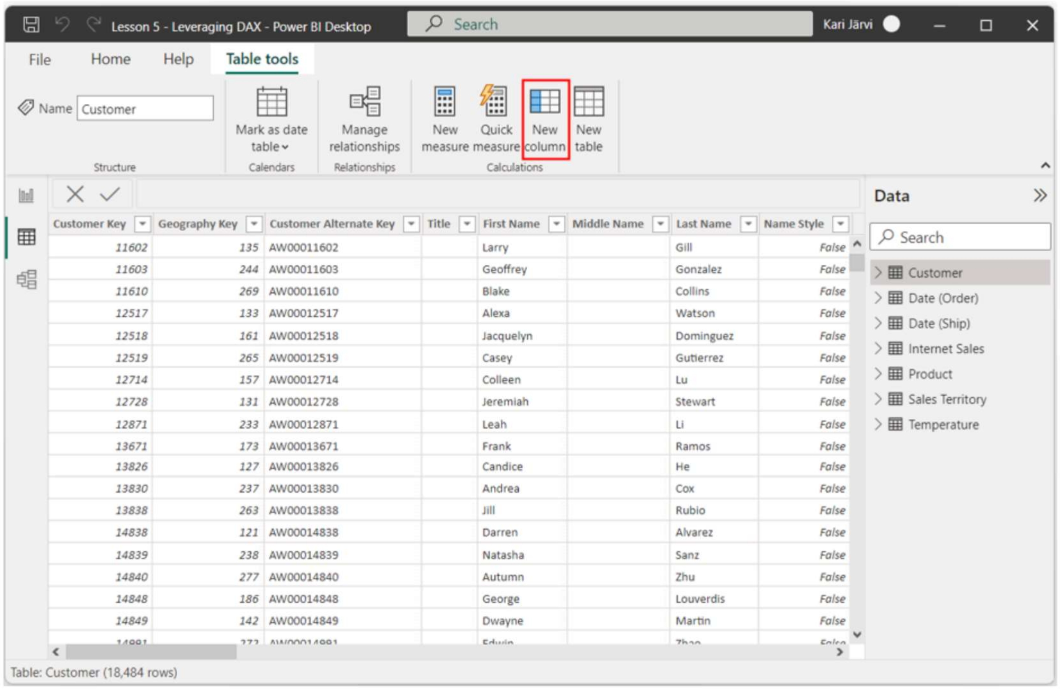
- Adding descriptive columns
- Creating concatenated key columns

Task 2 - Create your first calculated column

Before you get started though, you need to first know that *Power BI Desktop* provides the **IntelliSense** feature. *IntelliSense* will help you out a lot when writing code. This built-in functionality will autocomplete your code as you go and will also help you explore and discover new functions in the *DAX* language. In order to take advantage of *IntelliSense*, you simply need to start typing in the formula bar.

Step 1: Click on **Data view**.

Step 2: Click on the **Customer** table from the **Fields** list. Once the *Customer* table has been selected, click on **New column**:



Customer Key	Geography Key	Customer Alternate Key	Title	First Name	Middle Name	Last Name	Name Style
11602	135	AW00011602		Larry		Gill	False
11603	244	AW00011603		Geoffrey		Gonzalez	False
11610	269	AW00011610		Blake		Collins	False
12517	133	AW00012517		Alexa		Watson	False
12518	161	AW00012518		Jacquelyn		Dominguez	False
12519	265	AW00012519		Casey		Gutierrez	False
12714	157	AW00012714		Colleen		Lu	False
12728	131	AW00012728		Jeremiah		Stewart	False
12871	233	AW00012871		Leah		Li	False
13671	173	AW00013671		Frank		Ramos	False
13826	127	AW00013826		Candice		He	False
13830	237	AW00013830		Andrea		Cox	False
13838	263	AW00013838		Jill		Rubio	False
14838	121	AW00014838		Darren		Alvarez	False
14839	238	AW00014839		Natasha		Sanz	False
14840	277	AW00014840		Autumn		Zhu	False
14848	186	AW00014848		George		Louverdis	False
14849	142	AW00014849		Dwayne		Martin	False

Table: Customer (18,484 rows)

Step 3: You will now see the text **Column =** in the formula bar. First, name the new column by replacing the default text of Column with *Full Name*. Then, move your cursor to after the equals sign and type a single *quote* character. Immediately after typing the single quote character, a list of autocomplete options will appear underneath the formula bar. This is *IntelliSense* at work. The first option in this list is the name of the table you currently have selected—*Customer*. Press the *Tab* key and the name of the table will automatically be added to the formula bar.

At some point, you will inevitably discover that you can reference just the column name. As a best practice, we recommend always referencing both the table and column name anytime you use a column in your DAX code.

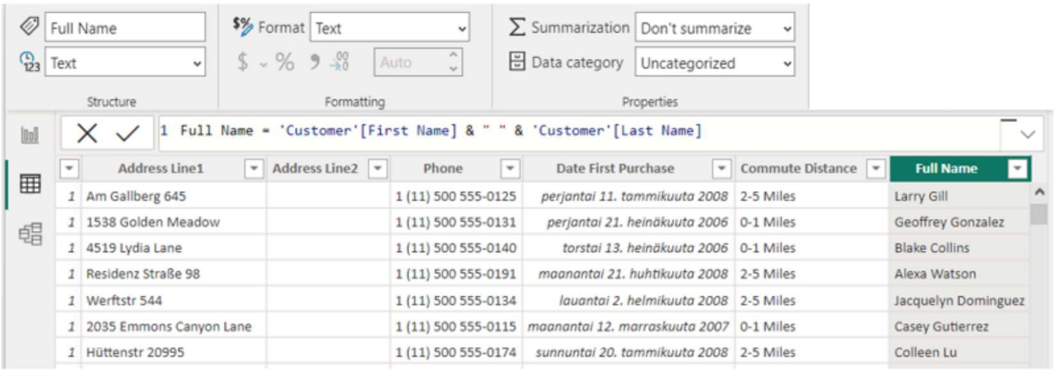
Step 4: Next, type an opening square bracket **[** into the formula bar followed by the capital letter **F**, making "[F". Once again, you will immediately be presented with autocomplete options. The list of options has been limited to only columns that contain the letter *f*, and the first option available from the dropdown is *First Name*. Press the *Tab* key to autocomplete. The formula bar should now contain the following formula:

```
Full Name = 'Customer'[First Name]
```

Step 5: The next step is to add a *space*, followed by the *last name.* There are two options in DAX for combining string values. The first option is the **concatenate** function. Unfortunately, concatenate only accepts two parameters; therefore, if you have more than two parameters, your code will require multiple concatenate function calls. On the other hand, you also have the option of using the **ampersand** sign (**&**) to combine strings.

The ampersand will first take both input parameters and convert them into strings. After this data conversion step, the two strings are then combined into one. Let’s continue with the rest of the expression. Remember to use the built-in autocomplete functionality to help you write code.

Step 6: Next, add a **space** and the *last name* column. To add a space—or any string literal value for that matter—into a DAX formula, you will use quotes on both sides of the string. For example, **" "** inserts a space between the first and last name columns. The completed DAX formula will look like Figure below:



```
Full Name = 'Customer'[First Name] & " " & 'Customer'[Last Name]
```

	Address Line1	Address Line2	Phone	Date First Purchase	Commute Distance	Full Name
1	Am Gallberg 645		1 (11) 500 555-0125	perjantai 11. tammikuuta 2008	2-5 Miles	Larry Gill
1	1538 Golden Meadow		1 (11) 500 555-0131	perjantai 21. heinäkuuta 2006	0-1 Miles	Geoffrey Gonzalez
1	4519 Lydia Lane		1 (11) 500 555-0140	torstai 13. heinäkuuta 2006	0-1 Miles	Blake Collins
1	Residenz Straße 98		1 (11) 500 555-0191	maanantai 21. huhtikuuta 2008	2-5 Miles	Alexa Watson
1	Werftstr 544		1 (11) 500 555-0134	lauantai 2. helmikuuta 2008	2-5 Miles	Jacquelyn Dominguez
1	2035 Emmons Canyon Lane		1 (11) 500 555-0115	maanantai 12. marraskuuta 2007	0-1 Miles	Casey Gutierrez
1	Hüttenstr 20995		1 (11) 500 555-0174	sunnuntai 20. tammikuuta 2008	2-5 Miles	Colleen Lu

Task 3 - Create calculated columns with string functions

Now that you have completed your first calculated column, let's build a calculated column that stores the *month-year* value. The goal is to return a month-year column with the two-digit month and four-digit year separated by a dash, making *MM-YYYY*. Let's build this calculation incrementally.

Step 1: Select the **Date (Order)** table and then click on **New** column from the *Table tools* ribbon. Write the following code in the formula bar and then hit **Enter**:

```
Month Year = 'Date (Order)'[Month Number of Year]
```

As you begin validating the code, you will notice that this only returns the single-digit month with no leading zero. Your next attempt may look something like the following:

```
Month Year = "0" & 'Date (Order)'[Month Number of Year]
```

This will work for single-digit months; however, double-digit months will now return three digits.

Step 2: To improve upon this and only return the two-digit month, you can use the **RIGHT** function. The *RIGHT* function returns a specified number of characters from the right side of a string; in the example below, two characters are returned from the expression. Modify your existing DAX formula to look like the following:

```
Month Year = RIGHT("0" & 'Date (Order)'[Month Number of Year], 2)
```

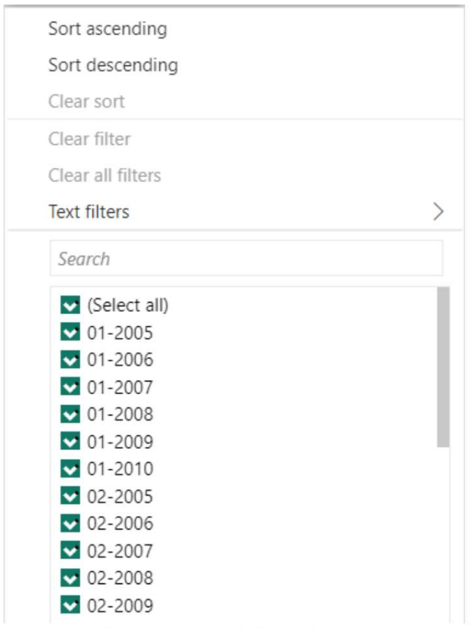
For a full list of text functions in DAX, please go to the following link: https://tinyurl.com/pbiqs-text

The rest of this formula can be completed quite easily. First, to add a dash, the following DAX code can be used:

```
Month Year = RIGHT("0" & 'Date (Order)'[Month Number of Year], 2) & "-"
```

Complete the *Month Year* formula by combining the current string with the *Year* column.

```
Month Year = RIGHT("0" & 'Date (Order)'[Month Number Of Year],2) & "-" & 'Date (Order)'[Year]
```


Sort ascending
Sort descending
Clear sort
Clear filter
Clear all filters
Text filters
Search
(Select all)
01-2005
01-2006
01-2007
01-2008
01-2009
01-2010
02-2005
02-2006
02-2007
02-2008
02-2009

You may have noticed that the *Year* column has a data type of a whole number, and you may have expected that this numeric value would need to be converted to a *String* prior to the combine operation. However, remember that the ampersand operator will automatically convert both inputs into a String before performing the combine operation!

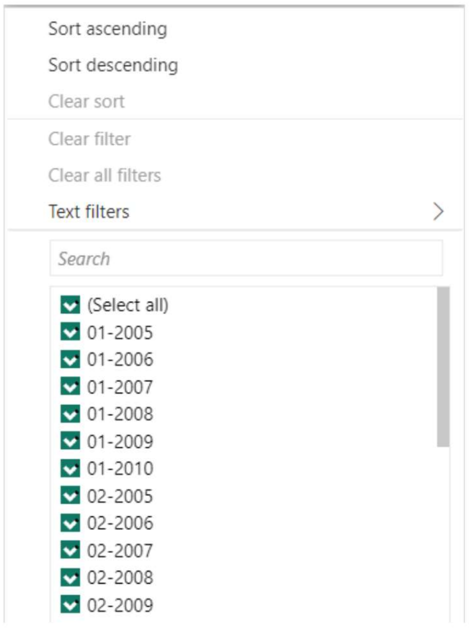
In this example, you learned how to create a *Month Year* column using a combination of the *Right* function with the *ampersand* operator. Next, you will learn an easier method for achieving the same goal with the *FORMAT* function.

Task 4: Using the FORMAT function in DAX

As with any other language, you will find that there are usually multiple ways to do something. Now you are going to learn how to perform the calculation that we saw in the previous section using the **FORMAT** function. The *FORMAT* function allows you to take a number or Date column and customize it in a number of ways. A side effect of the *FORMAT* function is that the resulting data type will be text. Therefore, this may affect numerical-based calculations that you wish to perform on those columns at a later time. Let's perform the preceding calculation again, but this time, using the *FORMAT* function.

Make sure you have the **Date (Order)** table selected, and then click on Create a New Calculated Column by selecting **New** column from the *Table* tools ribbon. In the formula bar, write the following expression:

```
Month Year Format = FORMAT('Date (Order)'[Date], "MM-YYYY")
```


Sort ascending
Sort descending
Clear sort
Clear filter
Clear all filters
Text filters
Search
(Select all)
01-2005
01-2006
01-2007
01-2008
01-2009
01-2010
02-2005
02-2006
02-2007
02-2008
02-2009

If you would like to take a full look at all the custom formatting options available using the *FORMAT* function, please take a look at https://tinyurl.com/pbiqs-format

Now, let's take a look at the DATEDIFF function and how to implement conditional logic with the IF function in DAX.

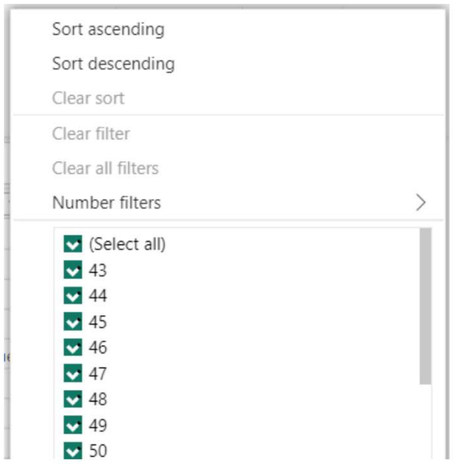
Task 5 - Implementing conditional logic with IF()

Next, you are going to determine each customer's age. The *Customer* table currently contains a column with the birth date of each customer.

Step 1: Make sure you have the Customer order table selected, and then click on **Create a New Calculated Column** by selecting **New column** from the *Table* tools ribbon. Try to use the **DATEDIFF** function in a calculation that looks something like the following:

```
Customer Age = DATEDIFF('Customer'[Birth Date], TODAY(), YEAR)
```

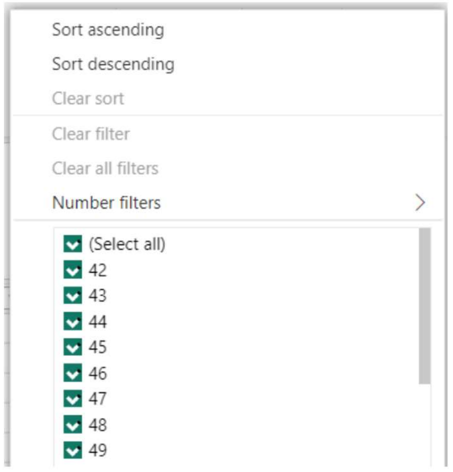
The **TODAY** function returns the current date and time. The **DATEDIFF** function returns the count of the specified interval between two dates; however, it does not look at the day and month and therefore, does not always return the correct age for each customer.



Let's rewrite the previous DAX formula in a different way. In this example, you are going to learn how to use conditional logic and the *FORMAT* function to return the proper customer age. Please keep in mind that there are many ways to perform this calculation.

Step 2: Select the *Customer Age* column from the previous step and rewrite the formula to look like:

```
Customer Age =
IF(
    FORMAT('Customer'[Birth Date],"MMDD") <= FORMAT(TODAY(),"MMDD"), //Logical Test
    DATEDIFF('Customer'[Birth Date],TODAY(),YEAR),                   //Result IF True
    DATEDIFF('Customer'[Birth Date],TODAY(),YEAR)-1)                 //Result IF False
```



Formatting code is very important for readability and maintaining code. Power BI Desktop has built-in functionality to help out with code formatting. When you press *Shift* + *Enter* to navigate down to the next line in your formula bar, your code will be indented automatically where applicable.

When completed, the preceding code returns the correct age for each customer. The *FORMAT* function is used to return the two-digit month and two-digit day for each date (the *birth date* and *TODAY's* date). Following the logical test portion of the *IF* statement are two expressions. The first expression is triggered if the logical test evaluates to true, and the second expression is triggered if the result of the test is false. Therefore, if the customer's month and day combo is less than or equal to today's month and day, then their birthday has already occurred this year, and the logical test will evaluate to true, which will trigger the first expression. If the customer's birthday has not yet occurred this year, then the second expression will execute.

In the preceding DAX formula, comments were added by using two forward slashes in the code (**//**). Comments are descriptive and are not executed with the rest of the DAX formula. Commenting code is always encouraged and will make your code more readable and easier to maintain.

In this example, you learned how to implement conditional logic with the *IF* function. Now let's look at the *SWITCH* function, which can be used as an alternative method to *IF*.

Task 6 - Implement conditional logic with SWITCH()

Now that you have the customer's age, it's time to put each customer into an age bucket. For this example, there will be four separate age buckets:

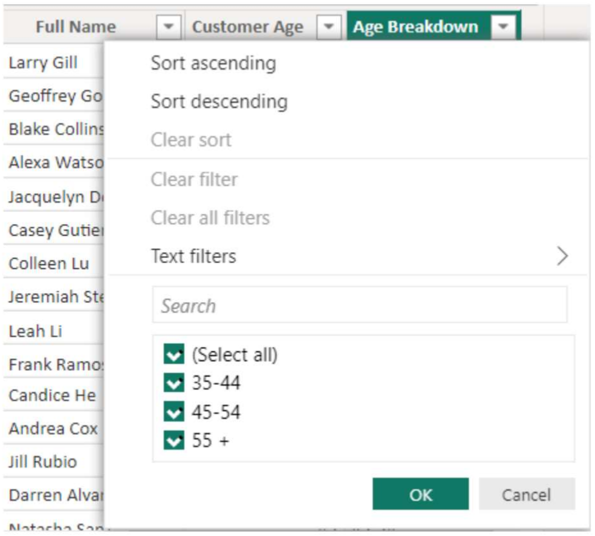
- 18-34
- 35-44
- 45-54
- 55+

The **SWITCH** function is preferable to the **IF** function when performing multiple logical tests in a single DAX formula. This is because the *SWITCH* function is easier to read and makes debugging code much easier.

Step 1: With the *Customer* table selected, click on **New Column**. Type in the completed DAX formula:

```
Age Breakdown =
SWITCH(TRUE(),
    'Customer'[Customer Age] >= 55, "55 +",      //If 55 or older, then 55+
    'Customer'[Customer Age] >= 45, "45-54",     //If 45-54, then 45-54
    'Customer'[Customer Age] >= 35, "35-44",     //If 35-44, then 35-44
    "18-34")
```

The preceding formula is very readable and understandable. There are three logical tests, and if a customer's age does not evaluate to true on any of those logical tests, then that customer is automatically put into the 18-34 age bucket.



You may have noticed that the second and third logical tests do not have an upper range assigned. For example, the second test simply checks whether the customer's age is 45 or greater. Naturally, you may assume that a customer whose age is 75 would be incorrectly assigned to the 45-54 age bucket. However, once a row evaluates to true, it is no longer available for subsequent logical tests. Someone who is 75 would have been evaluated to true on the first logical test (55+) and would no longer be available for any further tests.

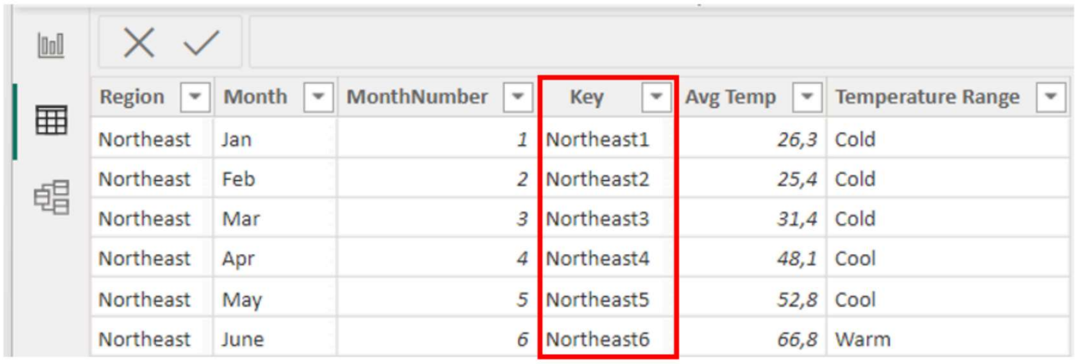
If you would like a better understanding of using the *SWITCH* statement instead of nesting multiple IF statements, then you can check out a blog post by Rob Collie at https://tinyurl.com/pbiqs-switch.

Let's take a look at another important task in DAX: retrieving data from a column in another table. In Excel, you would just use the VLOOKUP function, and in SQL, you would use a join. In DAX, you use navigation functions to retrieve this data.

Task 7 -Leveraging existing relationships with navigation functions

It's finally time to create a relationship between the *Temperature* table and the *Internet Sales* table. The key on the *Temperature* table is a combination of the region name and the month number of the year.

This column combination makes a single row unique in this table:



Region	Month	MonthNumber	Key	Avg Temp	Temperature Range
Northeast	Jan	1	Northeast1	26,3	Cold
Northeast	Feb	2	Northeast2	25,4	Cold
Northeast	Mar	3	Northeast3	31,4	Cold
Northeast	Apr	4	Northeast4	48,1	Cool
Northeast	May	5	Northeast5	52,8	Cool
Northeast	June	6	Northeast6	66,8	Warm

Unfortunately, neither of those two columns currently exists in the *Internet Sales* table. However, the *Internet Sales* table has a relationship with the *Sales Territory* table, and the *Sales Territory* table has the region. Therefore, you can determine the region for each sale by doing a simple lookup operation. Well, it should be that simple, but it's not quite that easy. Let's take a look at why.

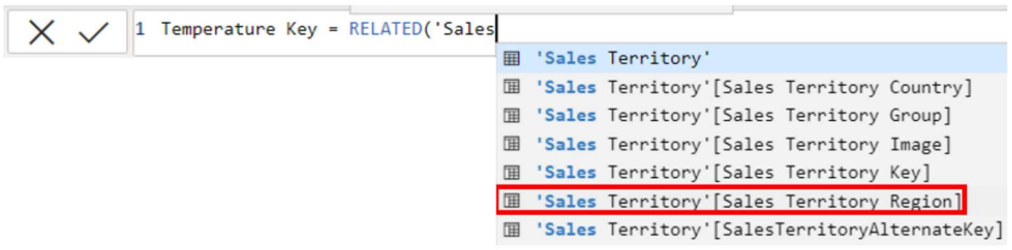
Note! **Calculated columns** do not automatically use the existing relationships in the data model. In contrast to calculated columns, **calculated measures** automatically see and interact with all relationships in the data model. Now let's take a look at why this is important. Calculated measures will be covered in the next section.

Step 1: Try to create a new column **Temperature Key** in the *Internet Sales* table, and try to return the region name from the *Sales Territory* table:

```
Temperature Key = 'Sales Terr
```

Note! There is no IntelliSense and that the autocomplete functionality is unavailable as we type in *Sales Territory*. The reason for this is the calculated column creates a row context and by default does not use the existing relationships in the data model. There is a much more complicated explanation behind all this, but for now, it is sufficient to say that navigation functions (**RELATED** and **RELATEDTABLE**) allow calculated columns to interact with and use existing relationships.

Step 2: Rewrite the following DAX formula with the **RELATED** function, then you will notice that IntelliSense has returned, along with the autocomplete functionality that was previously discussed. The IntelliSense is:



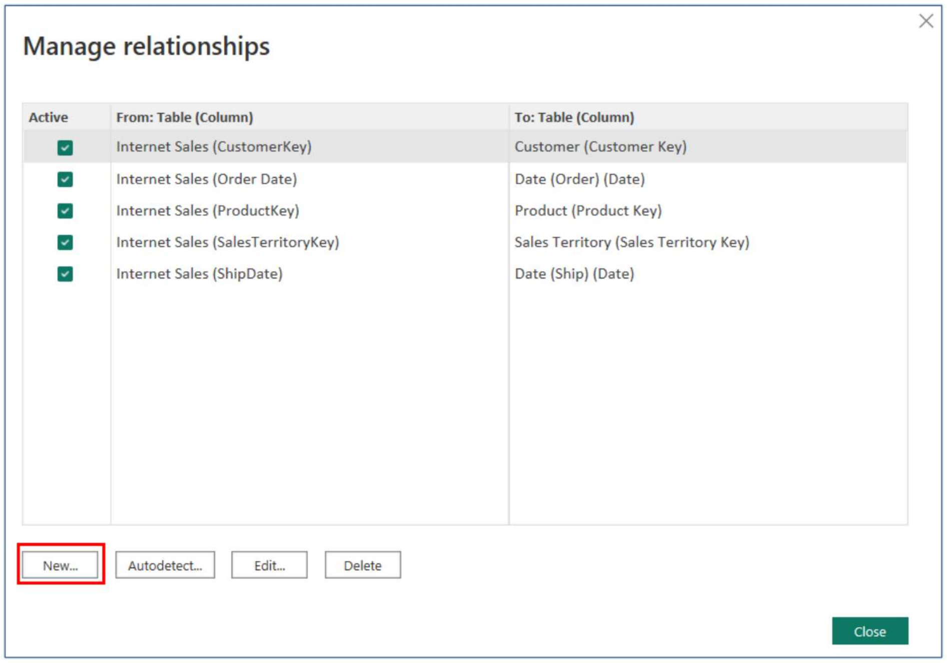
```
Temperature Key = RELATED('Sales
'Sales Territory'
'Sales Territory'[Sales Territory Country]
'Sales Territory'[Sales Territory Group]
'Sales Territory'[Sales Territory Image]
'Sales Territory'[Sales Territory Key]
'Sales Territory'[Sales Territory Region]
'Sales Territory'[SalesTerritoryAlternateKey]
```

Step 3: Now it's time to finish creating a *Temperature Key* column in the **Internet Sales** table: Type in the DAX formula:

```
Temperature Key =
RELATED('Sales Territory'[Sales Territory Region]) &  //the Region from Sales Territory Table
RELATED('Date (Order)'[Month Number Of Year])         //the Month number of Year from Date Table
```

You may have observed that unlike a VLOOKUP in Excel or a join in SQL, we did not have to specify the join columns when using the RELATED function! This is because the built-in navigation functions automatically leverage the existing relationships in the model.

Step 4: Now that the *Temperature Key* column has been created in the *Internet Sales* table, let's create the relationship. Click on **Manage Relationships** from the *Home* ribbon and then click on **New...** to open the *Relationship editor* window.

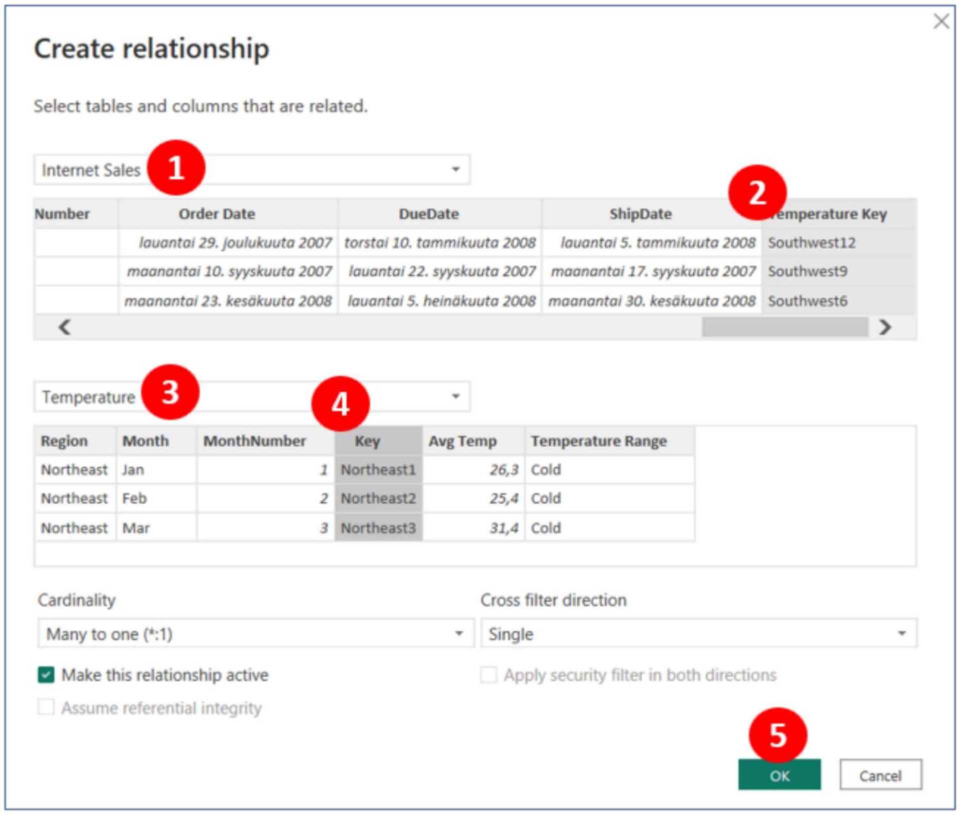


Manage relationships

Active	From: Table (Column)	To: Table (Column)
✓	Internet Sales (CustomerKey)	Customer (Customer Key)
✓	Internet Sales (Order Date)	Date (Order) (Date)
✓	Internet Sales (ProductKey)	Product (Product Key)
✓	Internet Sales (SalesTerritoryKey)	Sales Territory (Sales Territory Key)
✓	Internet Sales (ShipDate)	Date (Ship) (Date)

Step 2: Complete the following steps to create a new relationship:

1. Select *Internet Sales* from the first drop-down selection list.
2. Select *Temperature Key* from the list of columns (scroll right).
3. Select *Temperature* from the second drop-down selection list.
4. Select *Key* from the list of columns.
5. Click **OK** to save your new relationship.



Create relationship

Select tables and columns that are related.

Internet Sales 1

2

Number	Order Date	DueDate	ShipDate	Temperature Key
	lauantai 29. joulukuuta 2007	torstai 10. tammikuuta 2008	lauantai 5. tammikuuta 2008	Southwest12
	maanantai 10. syyskuuta 2007	lauantai 22. syyskuuta 2007	maanantai 17. syyskuuta 2007	Southwest9
	maanantai 23. kesäkuuta 2008	lauantai 5. heinäkuuta 2008	maanantai 30. kesäkuuta 2008	Southwest6

Temperature 3

4

Region	Month	MonthNumber	Key	Avg Temp	Temperature Range
Northeast	Jan	1	Northeast1	26,3	Cold
Northeast	Feb	2	Northeast2	25,4	Cold
Northeast	Mar	3	Northeast3	31,4	Cold

Cardinality: Many to one (*:1)

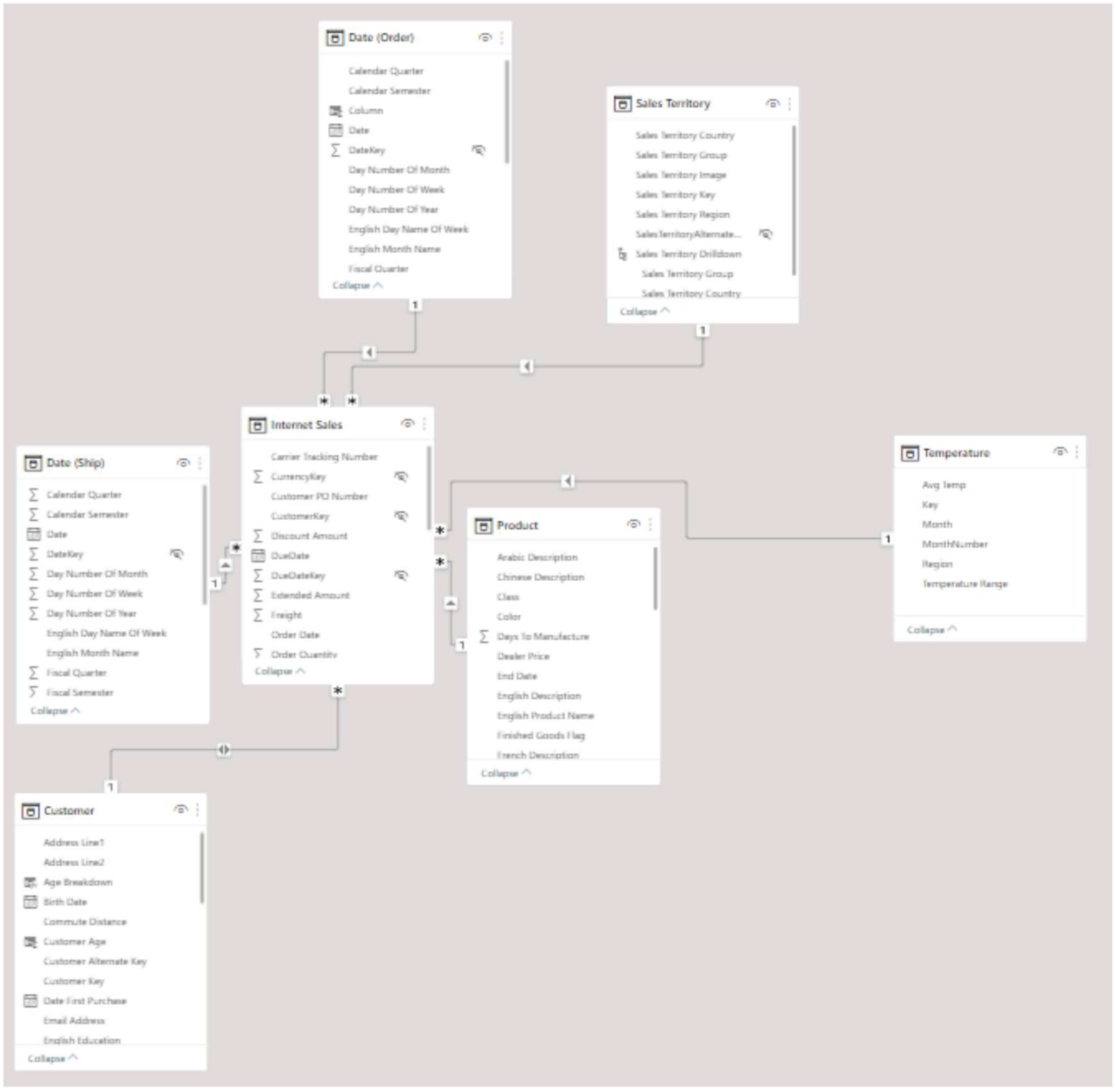
Cross filter direction: Single

[✓] Make this relationship active

[ ] Apply security filter in both directions

[ ] Assume referential integrity

5



So far in this chapter, you have learned how to create calculated columns in tables. These calculated columns increase the analytical capabilities of your data models by adding columns that can be leveraged to describe your metrics. You also learned how to create a concatenated key column in order to build a relationship between the Temperature and Internet Sales tables.

End-of-Exercise

Olet suorittanut 0 % oppitunnista

◄ Calculated columns with DAX

Siirry...

Exercise 12 - Creating calculated measures ►