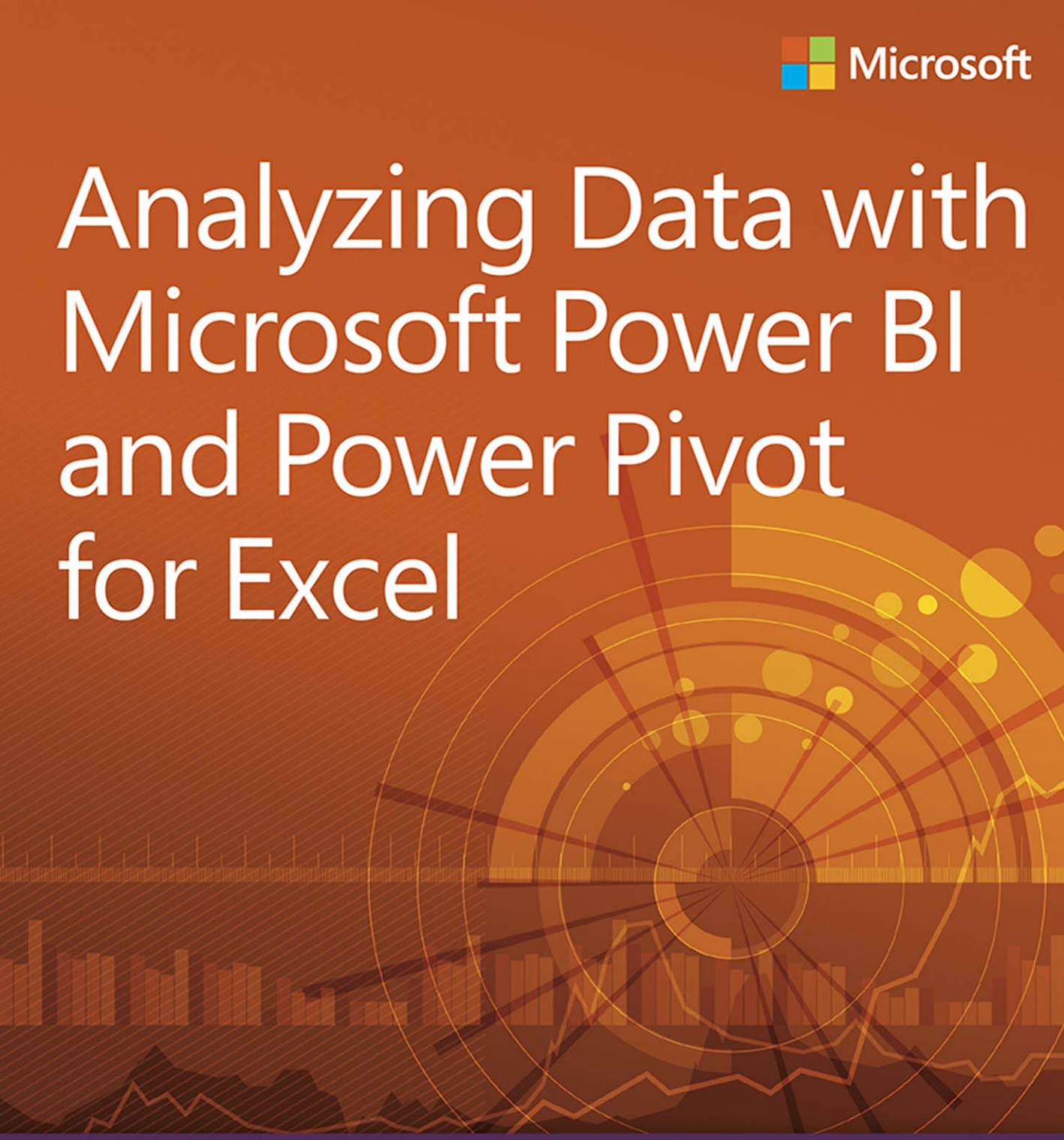
Microsoft

Analyzing Data with Microsoft Power BI and Power Pivot for Excel



Alberto Ferrari and Marco Russo

Sample files on the web

Microsoft

Analyzing Data with Microsoft Power BI and Power Pivot for Excel

Alberto Ferrari
and Marco Russo

Because using inactive relationships does not seem to be the way to go, you should modify the data model by duplicating the dimension. In our example, you can load the Date table three times: once for the order date, once for the due date, and once for the delivery date. You should obtain a non-ambiguous model like the one shown in Figure 4-10.

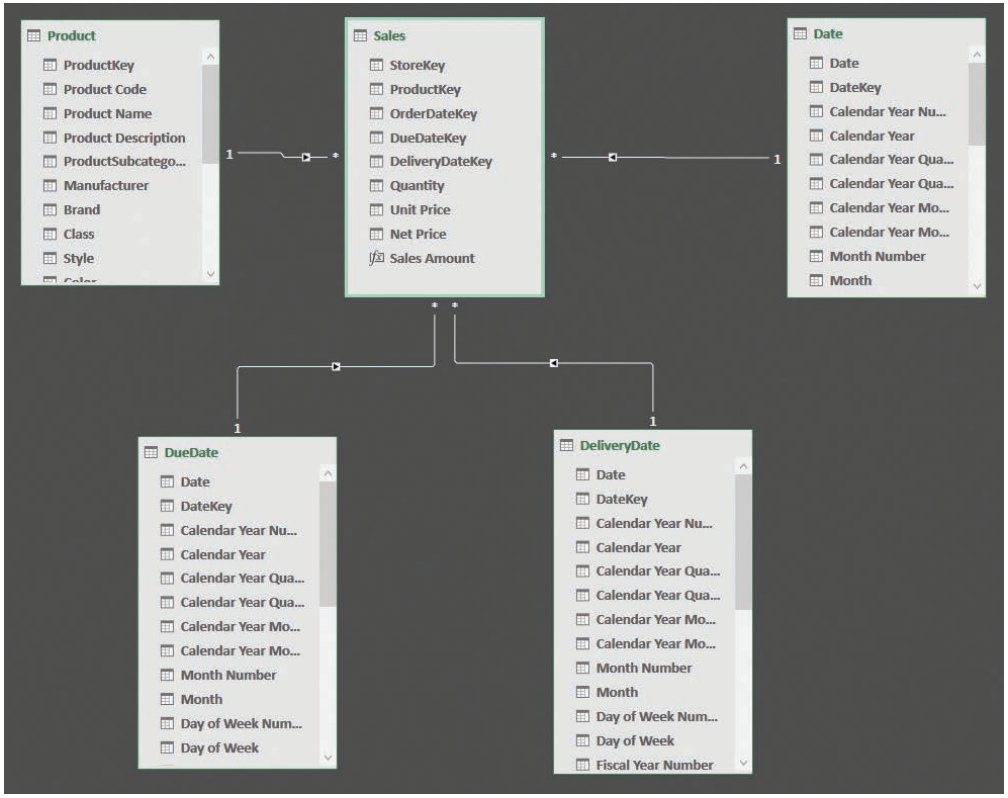


FIGURE 4-10 Loading the Date table multiple times removes ambiguity from the model.

Using this model, it is now possible to create reports that show, for example, the amount sold in one year but delivered in a different one, as shown in Figure 4-11.

Sales Amount	Column Labels				
Row Labels	2007	2008	2009	2010	Grand Total
2007	1,412,267.47	46,948.48			1,459,215.95
2008		1,092,620.19	29,914.86		1,122,535.05
2009			1,193,051.71	49,482.90	1,242,534.61
Grand Total	**1,412,267.47**	**1,139,568.67**	**1,222,966.57**	**49,482.90**	**3,824,285.61**

FIGURE 4-11 The report shows the amount sold in one year and shipped in a different one.

At first glance, the PivotTable in Figure 4-11 is hard to read. It is very difficult to quickly grasp whether the delivery year is in rows or in columns. You can deduce that the delivery year is in columns by analyzing the numbers because delivery always happens after the placement of the order. Nevertheless, this is not evident as it should be in a good report.

It is enough to change the content of the year column using the prefix *OY* for the order year and *DY* for the delivery year. This modifies the query of the Calendar table and it makes the report much easier to understand, as shown in Figure 4-12.

Sales Amount	Column Labels				
Row Labels	DY 2007	DY 2008	DY 2009	DY 2010	Grand Total
OY 2007	1,412,267.47	46,948.48			1,459,215.95
OY 2008		1,092,620.19	29,914.86		1,122,535.05
OY 2009			1,193,051.71	49,482.90	1,242,534.61
Grand Total	**1,412,267.47**	**1,139,568.67**	**1,222,966.57**	**49,482.90**	**3,824,285.61**

FIGURE 4-12 Changing the prefix of the order and delivery years makes the report much easier to understand.

So far, it looks like you can easily handle multiple dates by duplicating the date dimension as many times as needed, taking care to rename columns and add prefixes to the column values to make the report easy to read. To some extent, this is correct. Nevertheless, it is important to learn what might happen as soon as you have multiple fact tables. If you add another fact table to the data model, like Purchases, the scenario becomes much more complex, as shown in Figure 4-13.

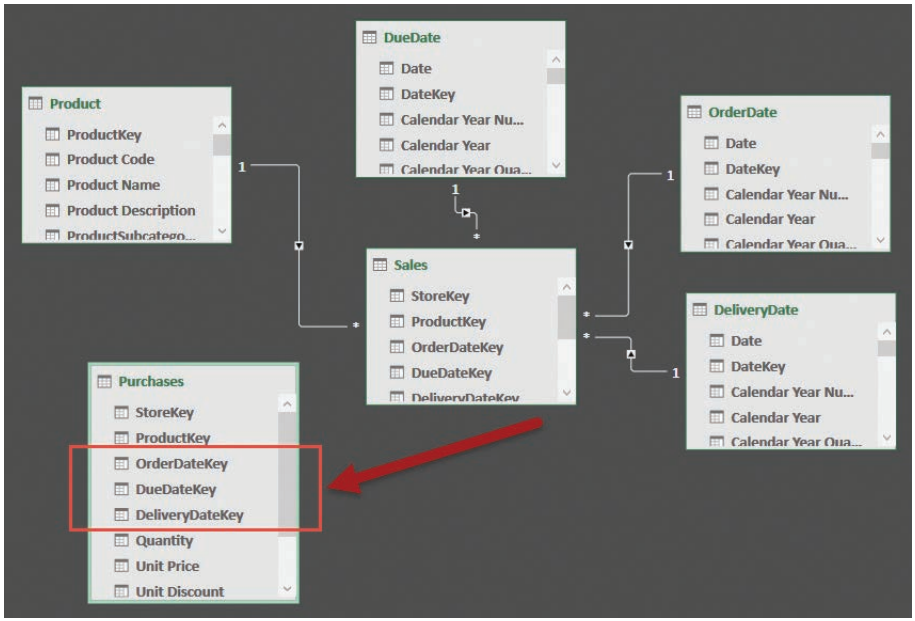


FIGURE 4-13 The Purchases table has three additional dates.

The simple addition of the Purchases table to the model generates three additional dates because purchases also have an order, delivery, and due date. The scenario now requires more skill to correctly design it. In fact, you can add three additional date dimensions to the model, reaching six dates in a single model. Users will be confused by the presence of all these dates. Therefore, although the model is very powerful, it is not easy to use, and it is very likely to lead to a poor user experience. Besides that, can you imagine what happens if, at some point, you need to add further fact tables? This explosion of date dimensions is not good at all.

Another option would be to use the three dimensions that are already present in the model to slice the purchases and sales. Thus, Order Date filters the order dates of both Sales and Purchases. The same thing happens for the other two dimensions, too. The data model becomes the one shown in Figure 4-14.

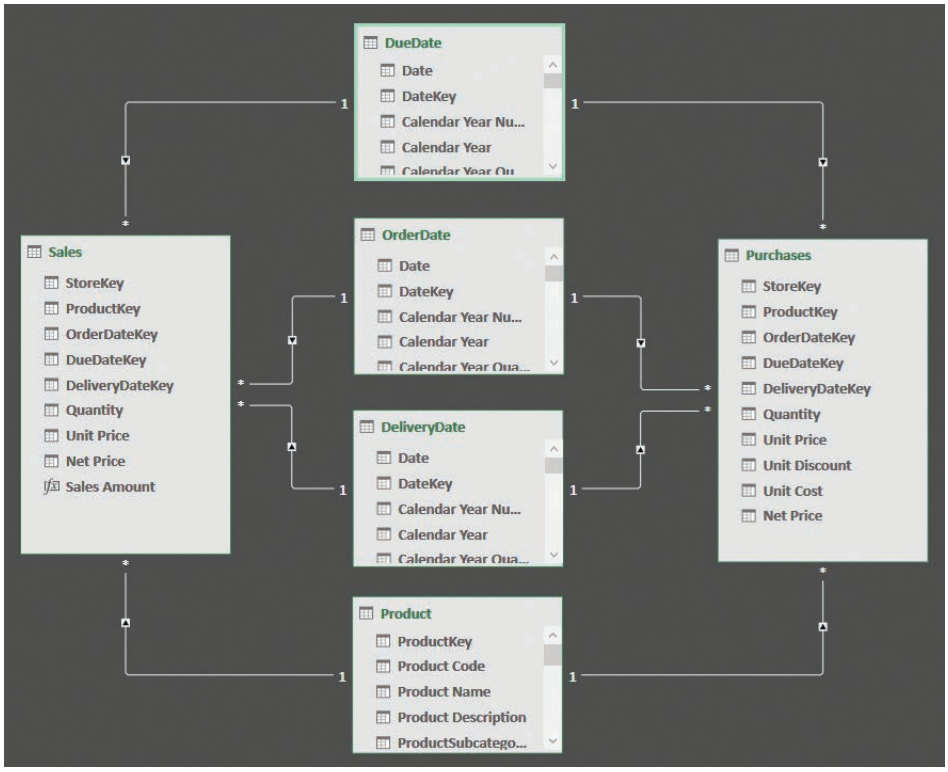


FIGURE 4-14 Using the same dimensions to filter two fact tables makes the model easier to use.

The model in Figure 4-14 is much easier to use, but it is still too complex. Moreover, it is worth noting that we were very lucky with the dimension we added. Purchases had the same three dates as Sales, which is not very common in the real world. It is much more likely that you will add fact tables with dates that have nothing to share with the previous facts. In such cases, you must decide whether to create additional date dimensions, making the model harder to browse, or to join the new fact table to one of the existing dates, which might create problems in terms of the user experience because the names are not likely to match perfectly.

The problem can be solved in a much easier way if you resist the urge to create multiple date dimensions in your model. In fact, if you stick with a single date dimension, the model will be much easier to browse and understand, as shown in Figure 4-15.

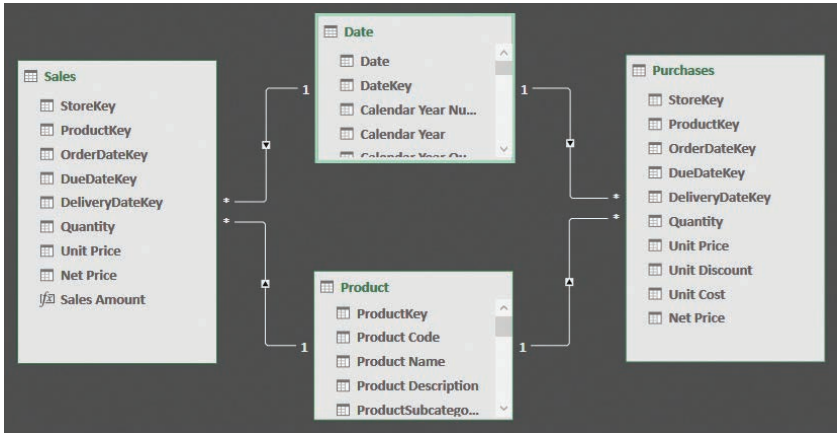


FIGURE 4-15 In this simpler model, there is a single date dimension that is linked to OrderDate in both fact tables.

Using a single date dimension makes the model extremely easy to use. In fact, you intuitively know that Date will slice both Sales and Purchases using their main date column, which is the date of the order. At first sight, this model looks less powerful than the previous ones, and to some extent, this is true. Nevertheless, before concluding that this model is in fact less powerful, it is worth spending some time analyzing what the differences are in analytical power between a model with many dates and one with a single date.

Offering many Date tables to the user makes it possible to produce reports by using multiple dates at once. You saw in a previous example that this might be useful to compute the amount sold versus the amount shipped. Nevertheless, the real question is whether you need multiple dates to show this piece of information. The answer is no. You can easily address this issue by creating specific measures that compute the needed values without having to change the data model.

If, for example, you want to compare the amount sold versus the amount shipped, you can keep an inactive relationship between Sales and Date, based on the DeliveryDateKey, and then activate it for some very specific measure. In our case, you would add one inactive relationship between Sales and Date, obtaining the model shown in Figure 4-16.

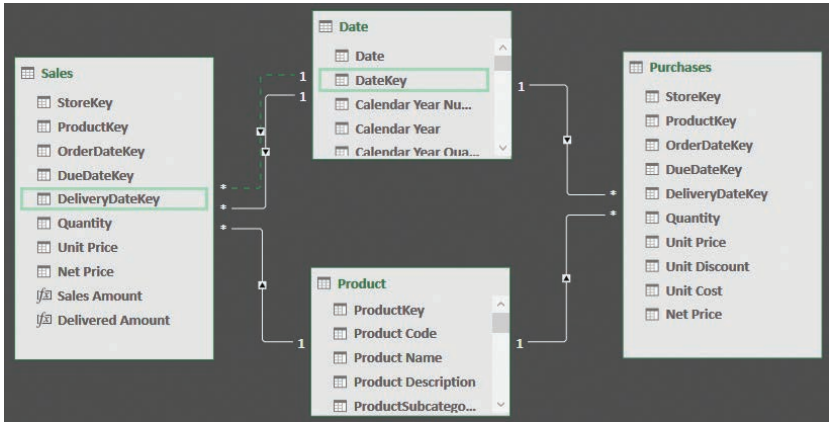


FIGURE 4-16 The relationship between DeliveryDateKey and DateKey is in the model, but it is disabled.

Once the relationship is in place, you can write the Delivered Amount measure in the following way:

```
Delivered Amount :=
CALCULATE (
    [Sales Amount],
    USERELATIONSHIP ( Sales[DeliveryDateKey], 'Date'[DateKey] )
)
```

This measure enables the inactive relationship between Sales and Date for only the duration of the calculation. Thus, you can use the Date table to slice your data, and you still get information that is related to the delivery date, as shown in the report in Figure 4-17. By choosing an appropriate name for the measure, you will not incur any ambiguity when using it.

Row Labels	Sales Amount	Delivered Amount
CY 2007	1,459,215.95	1,410,787.80
CY 2008	1,122,535.05	1,145,421.73
CY 2009	1,242,534.61	1,221,566.90
CY 2010		46,509.18
Grand Total	**3,824,285.61**	**3,824,285.61**

FIGURE 4-17 Delivered Amount uses the relationship based on a delivery date, but its logic is hidden in the measure.

Thus, the simple rule is to create a single date dimension for the whole model. Obviously, this is not a strict rule. There are scenarios where having multiple date dimensions makes perfect sense. But there must be a powerful need to justify the pain of handling multiple Date tables.

In our experience, most data models do not really require multiple Date tables. One is enough. If you need some calculations made using different dates, then you can create measures to compute them, leveraging inactive relationships. Most of the time, adding many date dimensions comes from some lack in the analysis of the requirements of the model. Thus, before adding another date dimension, always ask yourself whether you *really* need it, or if you can compute the same values using DAX code. If the latter is true, then go for more DAX code and fewer date dimensions. You will never regret that.

Handling date and time

Date is almost always a needed dimension in any model. Time, on the other hand, appears much less frequently. With that said, there are scenarios where both the date and the time are important dimensions, and in those cases, you need to carefully understand how to handle them.

The first important point to note is that a Date table cannot contain time information. In fact, to mark a table as a Date table (which you must do if you intend to use any time-intelligence functions on a table), you need to follow the requirements imposed by the DAX language. Among those requirements is that the column used to hold the `datetime` value should be at the day granularity, without time information. You will not get an error from the engine if you use a Date table that also contains time information. However, the engine will not be able to correctly compute time-intelligence functions if the same date appears multiple times.

So what can you do if you need to handle time too? The easiest and most efficient solution is to create one dimension for the date and a separate dimension for the time. You can easily create a time dimension by using a simple piece of M code in Power Query, like the following:

```
Let
    StartTime = #datetime(1900,1,1,0,0,0),
    Increment = #duration(0,0,1,0),
    Times = List.DateTimes(StartTime, 24*60, Increment),
    TimesAsTable = Table.FromList(Times,Splitter.SplitByNothing()),
    RenameTime = Table.RenameColumns(TimesAsTable,{{"Column1", "Time"}}),
    ChangedDataType = Table.TransformColumnTypes(RenameTime,{{"Time", type time}}),
    AddHour = Table.AddColumn(
        ChangedDataType,
        "Hour",
        each Text.PadStart(Text.From(Time.Hour([Time])), 2, "0" )
    ),
    AddMinute = Table.AddColumn(
        AddHour,
        "Minute",
        each Text.PadStart(Text.From(Time.Minute([Time])), 2, "0" )
    ),
    AddHourMinute = Table.AddColumn(
        AddMinute,
        "HourMinute", each [Hour] & ":" & [Minute]
    ),
    AddIndex = Table.AddColumn(
        AddHourMinute,
        "TimeIndex",
        each Time.Hour([Time]) * 60 + Time.Minute([Time])
    ),
    Result = AddIndex
in
    Result
```

The script generates a table like the one shown in Figure 4-18. The table contains a TimeIndex column (with the numbers from 0 to 1439), which you can use to link the fact table, and a few columns to slice your data. If your table contains a different column for the time, you can easily modify the previous script to generate a time as the primary key.

Time	Hour	Minute	TimeIndex	HourMinute
00.00.00	00	00	0	00:00
00.01.00	00	01	1	00:01
00.02.00	00	02	2	00:02
00.03.00	00	03	3	00:03
00.04.00	00	04	4	00:04
00.05.00	00	05	5	00:05
00.06.00	00	06	6	00:06
00.07.00	00	07	7	00:07
00.08.00	00	08	8	00:08
00.09.00	00	09	9	00:09
00.10.00	00	10	10	00:10
00.11.00	00	11	11	00:11
00.12.00	00	12	12	00:12
00.13.00	00	13	13	00:13
00.14.00	00	14	14	00:14
00.15.00	00	15	15	00:15

FIGURE 4-18 This is a simple time table that is generated with Power Query.