Pro DAX with Power BI

Business Intelligence with PowerPivot and SQL Server Analysis Services Tabular

—

Philip Seamark
Thomas Martens

Apress®

Pro DAX with Power BI

Business Intelligence with PowerPivot and SQL Server Analysis Services Tabular

Philip Seamark
Thomas Martens

Apress®

Pro DAX with Power BI: Business Intelligence with PowerPivot and SQL Server Analysis Services Tabular

Philip Seamark
UPPER HUTT, New Zealand

Thomas Martens
Hamburg, Hamburg, Germany

ISBN-13 (pbk): 978-1-4842-4896-6
ISBN-13 (electronic): 978-1-4842-4897-3
https://doi.org/10.1007/978-1-4842-4897-3

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Joan Murray
Development Editor: Laura Berendson
Coordinating Editor: Jill Balzano

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484248966. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

To William
—Philip Seamark

To #thewomanilove
—Thomas Martens

Table of Contents

About the Authors

Philip Seamark is an experienced data warehouse (DW) and business intelligence (BI) consultant with a deep understanding of the Microsoft stack and extensive knowledge of DW methodologies and enterprise data modeling. Recognized for his analytical, conceptual, and problem-solving abilities, he has more than 25 years of commercial experience delivering business applications across a broad range of technologies. His expertise runs the gamut from project management, dimensional modeling, performance tuning, ETL design, development and optimization, and report and dashboard design to installation and administration.

In 2017, Philip received a Microsoft Data Platform MVP award for his contributions to the Power BI community site, and in 2018 he was named a top 10 influencer in the Power BI space. He can be found speaking at many data, analytic, and reporting events around the world. He is also the founder and organizer of the Wellington Power BI User Group. He can be reached through Twitter @PhilSeamark.

Thomas Martens has more than 20 years of experience in the fields of BI, DW, and analytics. In his current role as business intelligence and analytics principal consultant, he helps large enterprises to implement sustainable analytical applications. In 2018, Thomas received a Microsoft Data Platform MVP award for his contributions to the Power BI community site for the first time. He specializes in the visualization of data and the application of analytical methods to large amounts of data. He recognizes DAX as a powerful query language available in many products and wants to leverage his hard-earned experience in creating solutions to bring DAX users to the next level. He can be reached through Twitter @tommartens68.

PART I

The Foundation

CHAPTER 1

DAX Mechanics

The idea behind this chapter is quite simple. Throughout the last years, we have been asked a lot of questions on how to calculate measures and Calculated Columns. And we also have been asked the following question numerous times: "Why does this not work?"

As soon as we started looking closer to the underlying business problem, we started to often wonder, "Why did they ask this question? It's so simple." Then we realized that things we consider simple are often not that simple for other people.

Providing answers to DAX-related questions on the Power BI forum (`https://community.powerbi.com`) earned me thankful remarks like "You are a legend!" or "Wow, you are a DAX ninja!"

Sure, some of these questions have been challenging to answer, but I'm for sure not a legend and also not a ninja. I do not own and have never owned a black pajama that makes me disappear whenever I want or helps me to hover over the rooftops of the buildings in my hometown. From many conversations with clients and friends of mine, I know that there are many smart people outside who are facing DAX challenges that are beyond their current skills that make them think: "What do I have to do to learn this dark art?"

If you might think DAX is some kind of dark art and reading this book will help you conquer the world, and learn some spells, you'll get disappointed in some way. Yes, you will read some DAX code that will help you conquer the world; at least it will help you create revealing DAX statements, that will help you to discover the full potential that is hidden in your data. But you will certainly not find any spell or curse.

DAX helps tremendously to extract insights from your data. Sometimes this extraction is quite easy, and using one DAX function is sufficient. Unfortunately, there are also those moments where this extraction has to happen forcefully, meaning more than one DAX function is used and the DAX code to create a "simple" measure spans across multiple lines, even multiple pages. It's these DAX statements that may lead to the impression that DAX is an art or some kind of an ancient powerful language spoken by witches, sorcerers, or other mystical folk, but be assured it's not.

P. Seamark and T. Martens, *Pro DAX with Power BI*, https://doi.org/10.1007/978-1-4842-4897-3_1

But if DAX is not a "conjuring" language, why are there so many questions out there on all the forums like Power BI and even on Stack Overflow?

From our point of view, most of the time it's the oversight of the simple things, the moving parts as we call them. People are often asking if they overlook some hidden "context transition" or if they have to consider the "shadow filter" more closely. And they are also talking about mastering the evaluation context as if mastering it will be rewarded with a black belt.

The problem here is sometimes there is a hidden context transition and sometimes the shadow filter plays its role. But most of the times, they forget that mastering the "five-finger death punch" means that first they have to understand and master the parts that are the foundation of DAX. These basic parts form the foundation upon which DAX unfolds its "magic."

Why DAX mechanics

As already mentioned, using DAX to solve analytical questions is not a secret science practiced by some initiated few. Instead, we consider it a craft. This is because we think that everyone who is willing to spend time and does not fear some setbacks is able to master this craft. We think it's legit to compare the writing of a DAX statement with the creation of some pottery. As you might know, it can take some time to successfully finish an apprenticeship, and it can even take a lifetime to become a master of a craft (thinking of some Japanese pottery). Sometimes the intricate workings of a complex DAX statement may remind us even more of a Swiss-made masterpiece measuring valuable time than a simple mug of coffee that helps us keep awake while we figure out how the moving parts are linked together by our DAX statement.

But nevertheless, the pieces that have to work together flawlessly are few and the laws or principles that rule these pieces – the moving parts – are not that complex or difficult as the rules that command the movement of the planets. For this reason, we find it reasonable to try to demystify the writing of DAX and compare it with a craft that can be mastered.

The moving parts

Before we can start to write DAX, we need an environment that is able to execute a DAX statement and that also provides us with an interface where we can write the DAX statement. For the purpose of this book, this environment is set by using Power BI Desktop that can be downloaded at www.powerbi.com.

Power BI Desktop comes with a database (that stores the data and is able to understand/execute DAX queries) and also provides the interface to write these DAX statements. In addition to this and maybe the most obvious part of Power BI Desktop are the many ways to visualize the data stored in the database. Throughout this book, we are referring to a DAX statement or a DAX query, and most of the time it doesn't matter. But there are subtle differences:

- DAX statement

  The term DAX statement is used whenever we refer to a piece of DAX code that is used to define a measure or a Calculated Column.

- DAX query

  The term DAX query is used whenever we are referring to a query that is "automatically" created/composed by Power BI Desktop to retrieve the data from the database to "populate" a visual, no matter if it's a card visual, a table visual, or a clustered column chart.

The database

Much can be said about the database that helps us to find answers to critical questions from various departments throughout the organization, no matter if these organizations are large enterprises or small companies.

This database is Analysis Services Tabular, and its engine is officially called "*xVelocity in-memory analytics engine.*" This engine provides two modes for accessing data:

- Local data

  The data is stored inside the database in an in-memory columnar data store; this mode is commonly known as VertiPaq.

- Remote data

  The data is queried from the data source; this mode is commonly known as DirectQuery .

For the sake of simplicity here, it is called either VertiPaq or even simpler just Analysis Services *Tabular*; this is derived from the term "Business Intelligence Semantic Model Tabular," a name introduced with the release of SQL Server 2012 to differentiate the two analytical engines that have been available since then with SQL Server.

VertiPaq provides its power to the following products inside the Microsoft Business Intelligence offering:

- MSFT SQL Server Analysis Services (SSAS; on premises, since SQL Server 2012)
- Azure Analysis Services
- Power BI Desktop
- Power BI Service
- Power Pivot (in combination with MSFT Excel 2010 until MSFT Excel 2016)

When not explicitly mentioned, we always refer to the version that comes with Power BI Desktop. This book is not meant to cover all the technical details of the VertiPaq engine. This by itself would cover another book, but two points have to be mentioned:

- The data is stored in a *columnar* structure.
- The data is kept in memory.

The columnar and in-memory storage of the data sets the VertiPaq engine apart from SQL Server Relational and from SQL Server Analysis Services Multidimensional (MD). One might think that the in-memory storage limits the size of the dataset that can be stored and analyzed. But in comparison with the row-based data storage of relational database engines, it is possible to compress the data by magnitudes.

But nevertheless, here we will focus on the objects that are more obvious to you, the Power BI user. These objects are

- Tables
- Relationships between these tables
- Measures and Calculated Columns

We use the following picture (Figure 1-1) of a schematic table to explain the workings of certain DAX statements.

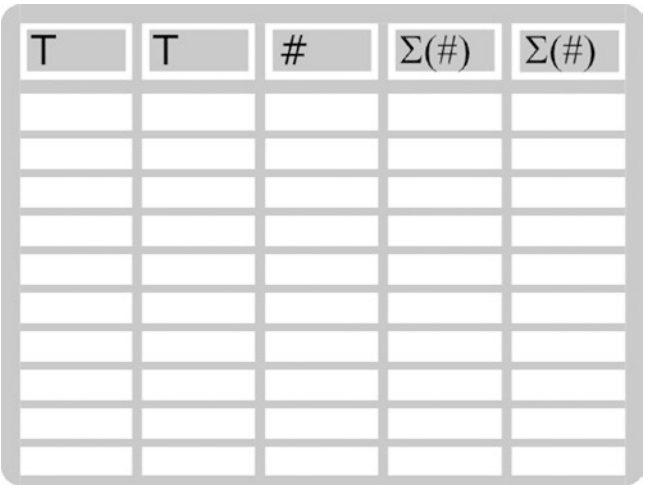

Figure 1-1. *Schematic table*

The preceding table has five columns:

- Two Text Columns (T) – These columns are used to describe the data like *customer name* or *product name.*
- One Numeric Column (#) – This should not be aggregated. This also applies to the column of the data types *datetime* and *date*. These columns often represent key values inside the source system like order *numbers*.
- Two Numeric Columns (Σ(#)) – These columns represent columns of numeric data types, like integers or decimal values. These columns will be aggregated and are most often used in measures.

Relationships are essential for the data analysis and play a vital part for performant DAX statements and will be treated extensively in Chapter 2, "Data Modeling." Basically, they relate the tables within a Tabular data model.

Measures are maybe the most powerful feature inside the xVelocity engine. This is simply due to the fact that whenever the data inside the table is not sufficient to extract the insight that we need, we are using DAX to create a calculation.

Definition A measure returns a scalar value; this means a single value. This scalar value is computed based on the rows of a table, which remain after all the filters have been applied. For this reason, it's safe to claim: A measure is computed by aggregating the filtered rows.

If you find the definition odd, and you are thinking about iterator functions like SUMX, where the expression allows to reference a single value from the current row inside the iteration, don't forget that finally the values are aggregated.

Note Measures can't be used inside slicers, nor as report-level filter and page-level filter in the filter pane of a Power BI report. Here, they can just be used as a Visual level filter.

Calculated Columns add additional analytical power to the table. Using Power BI Desktop to create the data model, one always has to answer the question if an additional column should be created using Power Query or DAX. Sometimes it seems simpler to create the columns using DAX, but there is a price that has to be paid whenever DAX is used for column creation.

- Calculated Columns created by using DAX will extend the duration needed to process the model.
- Calculated Columns created by using DAX will not compress as good as columns created by Power Query.

The preceding text is not a general recommendation for not using DAX to add columns to the data model. There may be situations where adding columns using DAX reduces the overall time spent on data refresh and model processing until the model is ready to be used for analysis.

One has to be aware of the fact that each column adds to the memory footprint of the data model. For this reason, you might consider to create measures in the future whenever possible.

Note Calculated Columns can be used on slicers, as report-level filter, and also page-level filters in Power BI reports, and form the content of the categorical axis of the visuals in Power BI.

Power BI Desktop

One of the greatest features of the Analysis Services database is the possibility to add Calculated Columns and measures to the data model. Besides this interface to the database engine, Power BI creates the stage that lets our DAX statements shine, the visuals. These visuals come with their own twist. They provide row and column headers as the Matrix visual does, or an x-axis for categorical values (everything besides fields with a numerical datatype). Sometimes it will become as difficult to show the data that we want to be visualized as it has been to create the measure itself. Chapter 8, "Using DAX to Solve Advanced Reporting Requirements," combines data modeling techniques with DAX statements to create a visual that

- Shows the last N-months in a clustered column chart and the user has to be able to select a certain month that will be used as anchor
- Shows next to the columns of the Top N-customers one additional column that represents the value of all other columns

For now it's sufficient to always remember the "level of interaction" of the objects that we create using DAX, namely, *Calculated Columns* and *measures. What this means is described in* Figure 1-2.

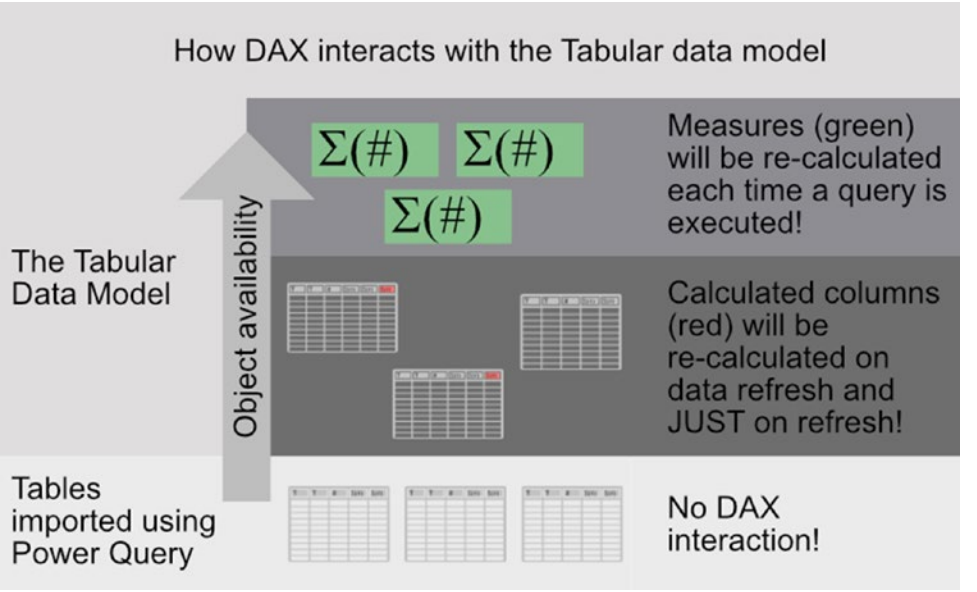

Figure 1-2. *How DAX interacts with the Tabular data model*

What can be learned from Figure 1-2 is the following:

Rule Objects created in the Tabular data model using DAX are not available from Power Query.

Calculated Columns will not be recalculated if a query is executed. The result of the DAX statement will persist in the underlying table if the DAX statement is initially committed and during data refresh.

Calculated tables created outside of a measure definition will be "created" inside the data model, meaning the DAX statement is executed during the initial creation or whenever the definition changed and of course during data refresh.

DAX: First contact

Throughout the book we will use the data model "Wide World Importers" (WWI) that is available on Git to demonstrate DAX statements with a data model, that is not too simple. The dataset used in this chapter is much easier. I think this is necessary to better understand what is really happening.

Implicit filters

Before we start creating our first DAX statement, it's necessary to have a look at the underlying data of the simple data model used in this chapter. Figure 1-3 shows the content of the table "*simple table values.*"

DateRunningIndex	Date	ProductIndex	Product Key	Brand	Color	Amount
384	Monday, March 19, 2018	3	P3	B3	white	13
571	Saturday, September 22, 2018	4	P4	B1	white	13
529	Saturday, August 11, 2018	6	P6	B3	red	14
92	Wednesday, May 31, 2017	6	P6	B3	red	12
357	Tuesday, February 20, 2018	5	P5	B2	blue	11
591	Friday, October 12, 2018	5	P5	B2	blue	7
902	Monday, August 19, 2019	8	P8	B4	blue	6
228	Saturday, October 14, 2017	7	P7	B2	blue	6
378	Tuesday, March 13, 2018	4	P4	B1	white	10
912	Thursday, August 29, 2019	1	P1	B1	red	9
555	Thursday, September 6, 2018	7	P7	B2	blue	12
545	Monday, August 27, 2018	2	P2	B2	blue	9
739	Saturday, March 9, 2019	5	P5	B2	blue	9
546	Tuesday, August 28, 2018	7	P7	B2	blue	11
6	Monday, March 6, 2017	7	P7	B2	blue	7
358	Wednesday, February 21, 2018	2	P2	B2	blue	9
441	Tuesday, May 15, 2018	8	P8	B4	blue	11
160	Monday, August 7, 2017	5	P5	B2	blue	11
707	Tuesday, February 5, 2019	1	P1	B1	red	7
89	Sunday, May 28, 2017	5	P5	B2	blue	8
639	Thursday, November 29, 2018	2	P2	B2	blue	13
725	Saturday, February 23, 2019	8	P8	B4	blue	10
567	Tuesday, September 18, 2018	8	P8	B4	blue	12
848	Wednesday, June 26, 2019	4	P4	B1	white	9

Figure 1-3. *Table – simple table values*

If this table is used in a Matrix visual with the following settings

- Brand column as rows
- Color column as columns
- Amount column as values

you will get what is shown in Figure 1-4 (except the circular marks).

Brand	blue	red	white	Total
B1		16	32	**48**
B2	113			**113**
B3		26	13	**39**
B4	39			**39**
Total	**152**	**42**	**45**	**239**

Figure 1-4. *simple table values matrix*

In his book *Beginning DAX with Power BI*, Phil has explained that Power BI adds the values from the row header Brand and from the column header Color to the evaluation context of the measure used on the values band of the Matrix visual.

Note Filters that are derived from column and row headers or slicer selections are called implicit filters. This is also true for the values that are used on the x-axis of the clustered column chart (this logic can be transferred to all other visuals).

I guess you are not surprised about the value displayed at the intersection of B3/red, even if we did not have defined any measure. The result can be checked easily by just filtering the table in the Data view, see Figure 1-5.

DateRunningIndex	Date	ProductIndex	Product Key	Brand	Color	Amount
92	Wednesday, May 31, 2017	6	P6	B3	red	12
529	Saturday, August 11, 2018	6	P6	B3	red	14

Figure 1-5. *simple table values – B3/red filtered*

It's easy to check that the addition of the values from the column *Amount* equals 26. And we can deduce the following:

- A visual filters the table which contains the column used as value, in this case the column *Amount*.
- An aggregation function is used to compute the value 26 from the two remaining rows after filters have been applied.

It's obvious that the aggregation function is SUM. SUM is the default aggregation function that is applied whenever a numeric column is used as value. The default function can be changed for each column and at least should be checked. In the

Properties ribbon of the Modeling menu, the value for Default Summarization can be changed if necessary. The function can be accessed from the Report or Data view; the column that has been checked has to be marked.

Figure 1-6 shows how the default summarization can be changed for the selected field `Amount`.

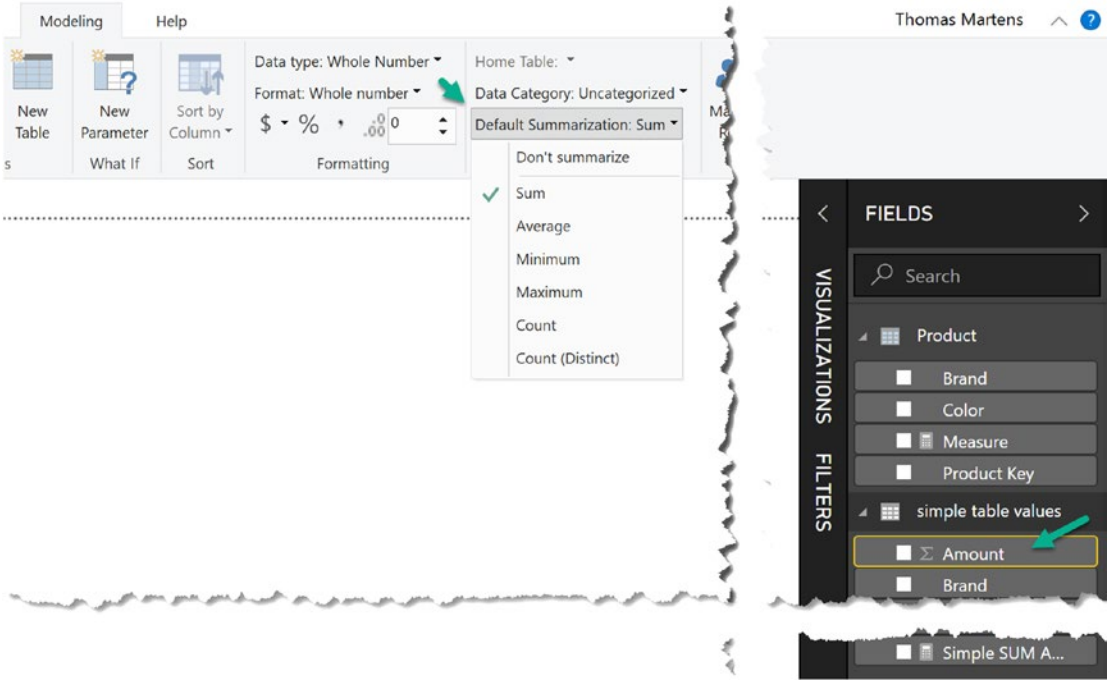

Figure 1-6. *Column default aggregation function*

Note If you change the default summarization of a column, you have to remove the column from the visual and add it back.

I have to admit that I have been thinking for quite some time that for each cell inside the Matrix visual including the total values on rows and columns, a separate DAX query is created, because the filter context changes for each cell. Fortunately, this is not the case. What really happens behind the covers can be controlled using DAX Studio, an open source tool that is mainly developed by the guys from sqlbi.com and Darren Gosbell. DAX Studio is an essential tool for the creation of DAX statements (the formatting is much smarter) and whenever you are not satisfied with the

performance of your DAX statement. For this reason, some of the capabilities of DAX Studio are described in Chapter 12, "DAX Studio" – at least the functions that are necessary to optimize slow-performing DAX statements.

To discover what's going behind the scenes, DAX Studio is able to catch the DAX query/queries that is/are created by Power BI Desktop, to retrieve the data to populate the visual.

The following DAX (see Listing 1-1) code is created when the report page that contains the simple matrix is activated.

Listing 1-1. Simple matrix

```
DEFINE
  VAR __DSOCore =
    SUMMARIZECOLUMNS(
      ROLLUPADDISSUBTOTAL('simple table values'[Brand],
      "IsGrandTotalRowTotal"),
      ROLLUPADDISSUBTOTAL('simple table values'[Color],
         "IsGrandTotalColumnTotal"),
      "SumAmount", CALCULATE(SUM('simple table values'[Amount]))
    )

  VAR __DSOPrimary =
    TOPN(
      102,
      SUMMARIZE(__DSOCore, 'simple table values'[Brand],
      [IsGrandTotalRowTotal]),
      [IsGrandTotalRowTotal],
      0,
      'simple table values'[Brand],
      1
    )

  VAR __DSOSecondary =
    TOPN(
      102,
      SUMMARIZE(__DSOCore, 'simple table values'[Color],
      [IsGrandTotalColumnTotal]),
```

```
      [IsGrandTotalColumnTotal],
      1,
      'simple table values'[Color],
      1
    )

EVALUATE
  __DSOSecondary

ORDER BY
  [IsGrandTotalColumnTotal], 'simple table values'[Color]

EVALUATE
  NATURALLEFTOUTERJOIN(
    __DSOPrimary,
    SUBSTITUTEWITHINDEX(
      __DSOCore,
      "ColumnIndex",
      __DSOSecondary,
      [IsGrandTotalColumnTotal],
      ASC,
      'simple table values'[Color],
      ASC
    )
  )

ORDER BY
  [IsGrandTotalRowTotal] DESC, 'simple table values'[Brand], [ColumnIndex]
```

We will delve into such DAX queries in much more detail in Chapter 12, "DAX Studio." But for now I just want to direct your attention to the first section of the DAX query. Here will see the following code snippet as part of the DAX function SUMMARIZECOLUMNS.

Listing 1-2. Implicit measure definition

```
"SumAmount", CALCULATE(SUM('simple table values'[Amount]))
```

If we would have written the measure, it would look almost the same. For this reason, we create a measure inside the table *simple table values* using this DAX statement without using the function CALCULATE (see Listing 1-3).

Listing 1-3. Measure – Simple SUM Amount

```
Simple SUM Amount =
SUM('simple table values'[Amount])
```

If you want to check the result, you will realize that we retrieve the same values as from the first query, but this is what we have been expecting.

But what we are looking for is the appearance of our first measure in the query created by Power BI. The following listing shows the first part of the query. And we can see that a "natural" column is treated like an explicitly defined measure:

```
VAR __DSOCore =
    SUMMARIZECOLUMNS(
      ROLLUPADDISSUBTOTAL('simple table values'[Brand],
      "IsGrandTotalRowTotal"),
      ROLLUPADDISSUBTOTAL('simple table values'[Color],
      "IsGrandTotalColumnTotal"),
      "SumAmount", CALCULATE(SUM('simple table values'[Amount])),
      "Simple_SUM_Amount", 'simple table values'[Simple SUM Amount]
    )
```

Until now, you have to believe me that both definitions are the same even if it seems that a CALCULATE function is missing that is wrapped around our measure, unless you have already read Phil's book or some other great DAX books that are available. But we could also prove that both measures yield exactly the same result just by removing the CALCULATE and executing the query in DAX Studio.

Before I will explain what an explicit filter is, I just want to slightly modify the Matrix visual. You will find this matrix on the report page "Chapter 1 – implicit filters – b." In addition to the implicit filters that will be applied to the query execution, I also added the column *Brand* as a page-level filter (just do avoid unwanted interference with other report pages). See Figure 1-7 for the filter settings.

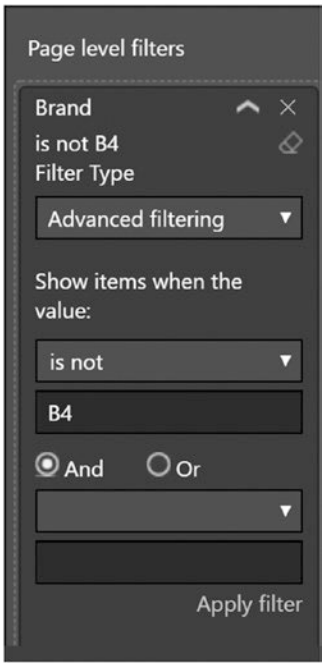

Figure 1-7. *Page-level filter Brand not like B4*

This is an important task because there are some misunderstandings about the behavior of the report-level filter, page-level filter, and also visual-level filter. The following is configured: Remove rows where the value of the column *Brand* equals *B4*. Listing 1-4 shows the query created by Power BI Desktop (at least the important part).

Listing 1-4. FilterTable

```
VAR __DSOFilterTable =
    FILTER(
      KEEPFILTERS(VALUES('simple table values'[Brand])),
      'simple table values'[Brand] <> "B4"
    )

  VAR __DSOCore =
    SUMMARIZECOLUMNS(
      ROLLUPADDISSUBTOTAL('simple table values'[Brand],
      "IsGrandTotalRowTotal"),
      ROLLUPADDISSUBTOTAL('simple table values'[Color],
      "IsGrandTotalColumnTotal"),
```

```
    __DSOFilterTable,
    "SumAmount", CALCULATE(SUM('simple table values'[Amount]))
  )
```

What's important to notice here is the creation of a variable called *__DS0FilterTable* that will be used in all subsequent sections of the query.

To make things much more exciting, I will utilize a third report page "Chapter 1 – implicit filters – c." This report page also has the same page-level filter, but additionally there is also a slicer that is also using the column Brand. Here I will select the Brands B1 and B2. Figure 1-8 shows how the report will look like.

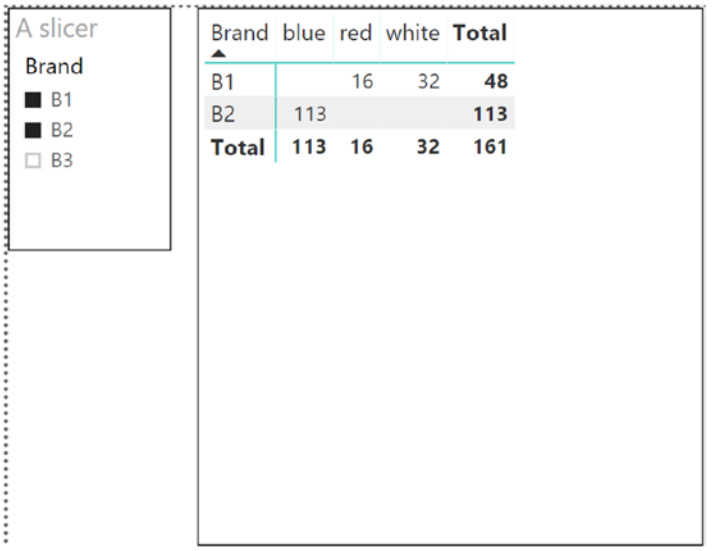

Figure 1-8. *Page-level filter and slicer*

Listing 1-5 shows the important parts of the query.

Listing 1-5. FilterTable page-level filter and slicer

```
VAR __DSOFilterTable =
    FILTER(
      KEEPFILTERS(VALUES('simple table values'[Brand])),
      AND(
        'simple table values'[Brand] IN {"B2",
          "B1"},
```

```
        'simple table values'[Brand] <> "B4"
      )
    )

VAR __DSOCore =
  SUMMARIZECOLUMNS(
    ROLLUPADDISSUBTOTAL('simple table values'[Brand],
    "IsGrandTotalRowTotal"),
    ROLLUPADDISSUBTOTAL('simple table values'[Color],
    "IsGrandTotalColumnTotal"),
    __DSOFilterTable,
    "SumAmount", CALCULATE(SUM('simple table values'[Amount]))
  )
```

You may wonder why this gets me so excited or why I find the variable __DS0FilterTable so interesting. But the answer to this is quite simple.

It's not possible to create a measure that shows the SUM of Amount for the three brands. What would be necessary is to remove the filter that has been implicitly added from the slicer but keep the filter that is coming from the page-level filter. To create such a measure, it's necessary to already take some precautions in the data model.

Rule It's not possible to remove filters coming from the slicers but keep the filter coming from the report-level filter or page-level filter.

Explicit filters

Whenever we are tasked with the writing of a measure, we have to tackle the challenge of the evaluation context. The evaluation context describes the context that is present when the measure (but also Calculated Columns) gets evaluated.

There are two components that determine this context:

- Filter context
- Row context