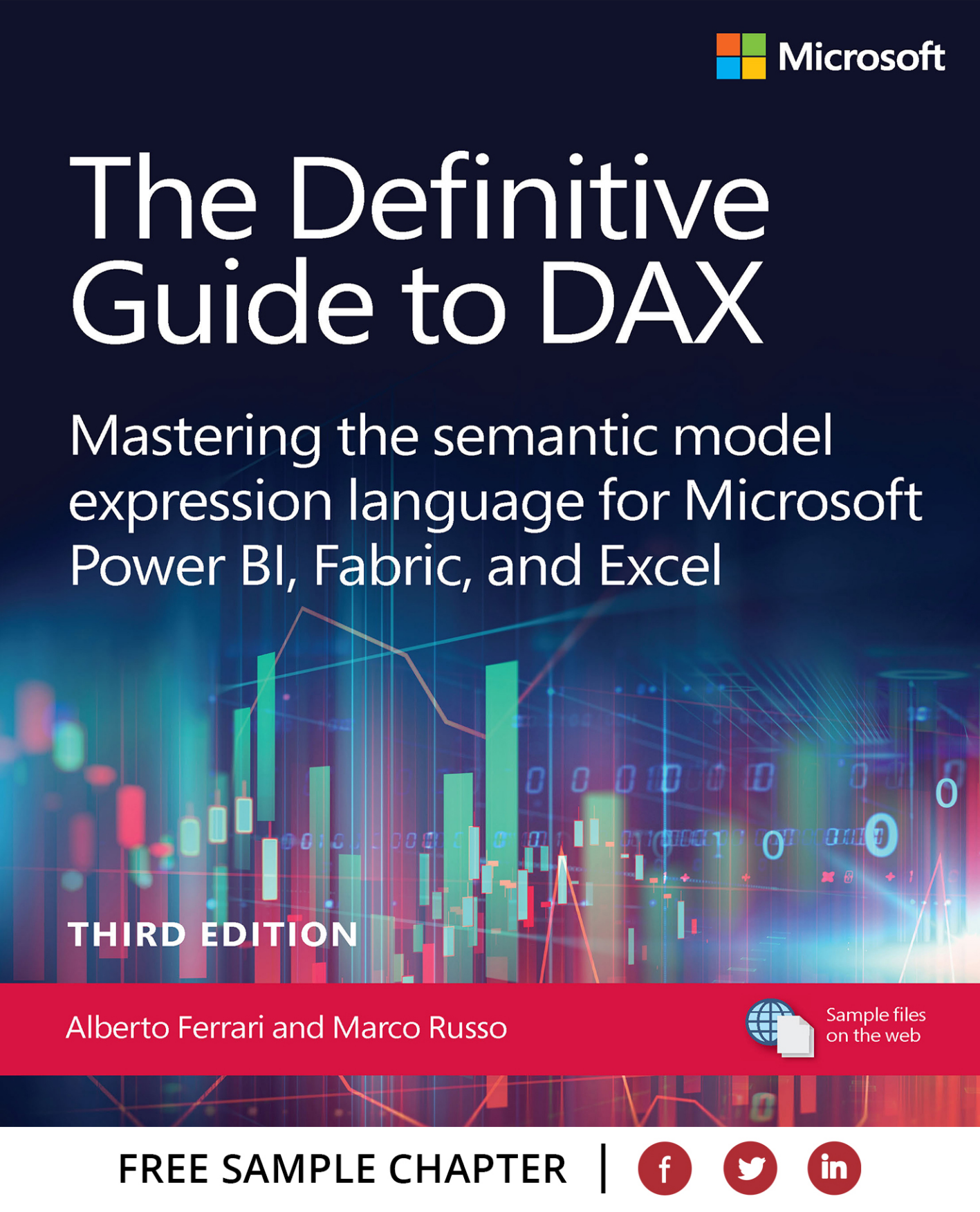
Microsoft
The Definitive Guide to DAX
Mastering the semantic model expression language for Microsoft Power BI, Fabric, and Excel
THIRD EDITION
Alberto Ferrari and Marco Russo
Sample files on the web
FREE SAMPLE CHAPTER

Microsoft

The Definitive Guide to DAX

Mastering the semantic model expression language for Microsoft Power BI, Fabric, and Excel

Third Edition

Alberto Ferrari
Marco Russo

The Definitive Guide to DAX: Mastering the semantic model expression language for Microsoft Power BI, Fabric, and Excel, Third Edition

Published with the authorization of Microsoft Corporation by:

Pearson Education, Inc.

Hoboken, New Jersey

ISBN-13: 978-0-13-824472-9
ISBN-10: 0-13-824472-3

Library of Congress Control Number: On file

$PrintCode

Portfolio Manager
Loretta Yates

Acquisitions Editor
Shourav Bose

Development Editors
Rick Kughen and Charlotte Kughen

Managing Editor
Sandra Schroeder

Senior Project Editor
Tracey Croom

Copy Editor
Charlotte Kughen

Indexer
Rachel Kuhn

Proofreader
Dan Foster

Technical Editor
Daniil Maslyuk

Cover Designer
Twist Creative, Seattle

Cover Illustration
whiteMocca/Shutterstock

Compositor
Danielle Foster

Graphics
Julien Charles-Donatien

Contents at a Glance

Contents

Foreword

Foreword to the 3rd edition

You may not know my name. I help create DAX and Power BI features, building on the shoulders of giants like Marius Dumitru, Cristian Petculescu, John Vulner, Christian Wade, and Jeffrey Wang. I'm honored to follow in their footsteps and contribute to the ongoing evolution of DAX.

Like many of you, I've spent countless hours wrestling with data—trying to make sense of it and learning how to make it speak. And in that journey, one resource has stood out above all: this book.

The *Definitive Guide to DAX* has become more than just a technical manual—it's a rite of passage for data professionals. Whether you're building your first visual calculation or optimizing a complex model, Marco Russo and Alberto Ferrari have been the quiet mentors behind your breakthroughs. Their clarity, precision, and relentless curiosity have shaped how we think about DAX and how we teach it.

Much of my focus has been on making DAX easier to use while expanding its power. That journey has led to features such as visual calculations, user-defined functions, calendar-based time intelligence, and others that I've helped shape. My role is to figure out what we need to build and why. But while I help shape ideas, it's the engineers who bring them to life. Their work is what makes Power BI and DAX truly shine. Without them, there is no Power BI—and no DAX.

As we craft new ideas and features, I get to work closely with Marco and Alberto. I often refer to them as "The Italians"—knowing full well there are many brilliant Italians in the DAX universe and beyond. Marco and Alberto meet regularly with our Power BI team, helping to shape the features you use every day. They challenge us, disagree when necessary (which is often), argue with conviction, and explain with clarity. They are masters of their art. Their impact is massive. They keep us grounded and are among our best eyes and ears in the community.

When I read the first two editions of *The Definitive Guide to DAX*, I admired Marco and Alberto and imagined them as belonging to a rare breed of superhumans. I still admire them—perhaps even more than before—but I no longer see them as superhuman. They are humble, principled, and always fighting for what's best for DAX, for Power BI, and for you. We can discuss life as openly as we do DAX. I never imagined that, and I'm grateful I get to do this.

This third edition of *The Definitive Guide to DAX* is not just a refresh. It reflects years of evolution in the DAX language, including new features such as visual calculations and time intelligence enhancements, and—most importantly—a deeper understanding of how people learn. Marco and Alberto have rethought how to explain abstract concepts, drawing from thousands of classroom interactions and community discussions.

If you're holding this book, you're about to level up. You'll learn not just how DAX works, but why it works the way it does. You'll gain insight into the semantic model, the engine, and the patterns that unlock analytical power. You'll join a global community of professionals who have chosen to go beyond the basics. This book is part of something bigger—a global community of data professionals, authors, trainers, and everyday users who push DAX further every day. You're not just learning from Marco and Alberto—you're joining a movement.

We write the code. They write the books. You? You write the future.

Welcome to the third edition.

Jeroen (Jay) ter Heerdt, Principal Product Manager, Power BI

Foreword to the 1st and 2nd editions

You may not know our names. We spend our days writing the code for the software you use in your daily job: We are part of the development team of Power BI, SQL Server Analysis Services, and—yes, we are among the authors of the DAX language and the VertiPaq engine.

The language you will learn using this book is our creation. We spent years working on this language, optimizing the engine, finding ways to improve the optimizer, and trying to build DAX into a simple, clean, and sound language to make your life as a data analyst easier and more productive.

But hey, this is intended to be the foreword of a book, so no more about us! Why are we writing a foreword for a book published by Marco and Alberto, the SQLBI guys? Well, because when you start learning DAX, it is a matter of a few clicks and searches on the web before you find articles written by them. You start reading their papers, learning the language, and hopefully appreciating our hard work. Having met them many years ago, we have great admiration for their deep knowledge of SQL Server Analysis Services. When the DAX adventure started, they were among the first to learn and adopt this new engine and language.

The articles, papers, and blog posts they publish and share on the web have become the source of learning for thousands of people. We write the code, but we do not spend much time teaching developers how to use it; Marco and Alberto are the ones who spread the knowledge about DAX.

Alberto and Marco's books are among a few bestsellers on this topic, and now with this new guide to DAX, they have truly created a milestone publication about the language we author and love. We write the code, they write the books, and you learn DAX, providing unprecedented analytical power to your business. This is what we love: working all together as a team—we, they, and you—to extract better insights from data.

Marius Dumitru, Architect, Power BI CTO's Office

Cristian Petculescu, Chief Architect of Power BI

Jeffrey Wang, Principal Software Engineer Manager

Christian Wade, Senior Program Manager

Acknowledgments

Writing the three editions of this book required an impressive amount of work. It has been a long and amazing journey, connecting people all around the world in any latitude and time zone to be able to produce the result you are going to read. There are many people who contributed in different ways, making it impractical to list everyone. Blog comments, forum posts, email discussions, chats with attendees and speakers at technical conferences, analyses of customer scenarios, and so much more have been useful to us, and many people have contributed significant ideas to this book. Moreover, big thanks to all the students of our courses: By teaching you, we get better!

That said, there are people we must mention personally, because of their particular contributions to this or previous editions.

We want to start with Edward Melomed: He has inspired us, and we probably would not have started our journey with the DAX language without a passionate discussion we had with him several years ago that ended with the table of contents of our first book about Power Pivot written on a napkin.

We want to thank Microsoft Press and the people who contributed to the project: They all greatly helped us along the process of book writing.

The only job longer than writing a book is the studying you must do in preparation for writing it. We are extremely lucky that many people from Microsoft helped us discover the details of new and old DAX functions. Many developers and PMs spent their time teaching us. However, we give special mention to Marius Dumitru, Jeffrey Wang, Jeroen Ter Heerdt, Akshai Mirchandani, Krystian Sakowski, and Cristian Petculescu. Your help has been priceless, guys!

We also want to thank Amir Netz, Christian Wade, Ashvini Sharma, Kasper De Jonge, and T. K. Anand for their contributions to the many discussions we had about the product. We feel they helped us tremendously in strategic choices we made in this book and in our careers.

We wanted to reserve a special mention to a woman who did an incredible job improving and cleaning up our English. Claire Costa proofread the entire manuscript and made it so much easier to read. Claire, your help is invaluable—Thanks!

We may know DAX, but we surely don't know how to convey information through meaningful and captivating graphics. Julien Charles-Donatien transformed our rough drafts into beautiful graphics, and his invaluable skills are visible in every figure of the book.

The last special mention goes to our technical reviewer: Daniil Maslyuk carefully tested every single line of code, text, example, and reference we had written. He found all kinds of mistakes we would have missed. He rarely made comments that did not require a change in the book. The result is amazing for us. If the book contains fewer errors than our original manuscript, it is only because of Daniil's efforts. If it still contains errors, it is our fault, of course.

Thank you so much, folks!

Introduction

Welcome to the third edition of this book. The first edition of *The Definitive Guide to DAX* dates back to 2015: At the time, 550 pages of DAX seemed a lot. When the second edition hit the shelves in 2020, it expanded to 750 pages. Both previous versions included a few chapters about DAX optimization, which we removed from this third edition because we dedicated an entire book to optimizing DAX in 2024. Hence, once you have completed this book, *Optimizing DAX* may be your next step! However, removing the optimization chapters did not reduce the size of the book: This third edition exceeds 800 pages, all about DAX.

Over the past ten years, DAX has evolved significantly, meeting the market's increasing requests for a powerful language to express business intelligence calculations. Moreover, we—as authors and teachers—have grown a lot, hopefully for the better. We have taught DAX to thousands of users and developers worldwide; we have worked hard with our students, always striving to find the best way to explain complex topics. Eventually, we found different ways of describing this language we love.

The three editions of this book changed with the market. The first edition was very technical, targeting mainly BI professionals. In the second edition, we aimed to broaden our audience by increasing the number of examples, showing practical uses of the functionalities after teaching the theoretical foundation of DAX. We tried to use a more straightforward style, without compromising on precision. In this third edition, we take one more step toward targeting all Power BI users: Not only have we increased the number of examples, but we have also enhanced the learning path. For example, whereas previous editions required readers to fully understand the complex theory of evaluation context before they could learn helpful code, this edition follows a more gradual approach. Instead of presenting the theory all at once, we first introduce theoretical concepts. We use them on several examples, and only later do we move forward, completing topics that were left incomplete during the introduction. We believe the learning experience is now much smoother, and we hope you will appreciate the effort.

Be mindful: We do not cut any corners. The goal of the book is not to teach you only how; instead, we want you to learn *why*. Here, you will not find a description of how to create a calculated column or which dialog to use to set a property. This is not a step-by-step book that teaches you how to use Power BI. This is a comprehensive exploration of the DAX language, beginning with the fundamentals and progressing to a profound understanding of its inner workings.

We assume no previous knowledge of DAX, even though this is not a book for the casual DAX developer. This book is for individuals who truly want to learn the language and gain a deep understanding of the power and complexity of DAX.

It is important to clarify that this book is structured as a learning path, rather than as a collection of examples designed to address everyday business problems. For practical, ready-to-use DAX solutions, we recommend visiting *www.daxpatterns.com*, our dedicated website. But here, our primary focus is to help you understand the inner workings of DAX, guiding you to truly "think in DAX."

To achieve this, we often use the same model and measures throughout the book. This consistency is intentional: It keeps the focus on the language itself, thus enabling a deeper comprehension of DAX rather than simply solving a variety of specific business scenarios. While you will encounter examples involving common calculations, the purpose is always to illustrate how DAX operates and to teach the underlying concepts, not to provide formulas for direct copy-and-paste use.

Most Power BI developers begin learning DAX by examining examples and demos, which have the primary goal of demonstrating the ease of the language. The thing is, watching someone write DAX will not help you learn how to write DAX any more than watching a professional runner compete in a 100-meter run will teach you how to run faster. To run faster, you need to learn the fundamentals and then practice regularly. The same applies to DAX: You need to learn the fundamentals first and then practice until they become an integral part of your knowledge.

DAX is a mighty language, used in a growing number of analytical tools. It is very powerful, but it encompasses a few concepts that are difficult to understand by just looking at examples. For instance, the evaluation context is a topic that requires a deductive approach: You start with a theory, and then you see a few examples that demonstrate how the theory works.

We really hope you will enjoy spending time with us on this beautiful trip to learn DAX, at least in the same way we enjoyed writing it.

Who this book is for

If you are entirely new to Power BI and plan to use DAX only casually, then this book is probably not the best choice for you. Many articles, videos, and books provide a simple introduction to the tools that implement DAX and to the DAX language itself, starting from the ground up and reaching a basic level of DAX programming. We are well aware of this because we have also produced some of that content!

On the other hand, if you are serious about DAX and you really want to understand every detail of this beautiful language, then this is the book for you. We provide a gradual approach to the language, starting from the basics and slowly moving to the most complex topics. We suggest reading the book from cover to cover in a first read. Then, you will probably review the most complex parts over time, appreciating several details thanks to the experience of practicing the language.

DAX is helpful to different people for various purposes: Power BI users may need to author DAX formulas in their models, Excel users can leverage DAX to author Power Pivot data models, and business intelligence (BI) professionals might need to implement DAX code in BI solutions of any size. In this book, we have attempted to provide information for all these diverse groups of people.

Finally, we wanted to write a book to study, not only a book to read. Initially, we aim to keep it simple and follow a logical path from zero to DAX. However, when the concepts start to become more complex, we stop trying to be simple and remain realistic. DAX is simple, but it is not easy. It took years for us to master it and to understand every detail of the engine. Do not expect to be able to learn all this content by reading casually over the course of a few days. This book requires your attention at a very high level. In exchange for that, we offer an unprecedented depth of coverage of all aspects of DAX, giving you the option to become a real DAX expert.

Assumptions about you

We expect you to have a basic knowledge of Power BI and some experience in analyzing numbers. If you have already had prior exposure to the DAX language, then this will be beneficial for you—you will be able to read the first part more quickly. But, of course, knowing DAX is not necessary.

Organization of this book

The book is designed to progress from introductory chapters to more complex chapters in a logical manner. Each chapter is designed to build upon a thorough understanding of the chapters before it. For this reason, we strongly suggest that you read it gradually from cover to cover and avoid jumping to more advanced chapters too early.

Many concepts are taught through examples. Therefore, please do not skip the examples because they are an essential part of the learning process. Once you have read it for the first time, it becomes useful as a reference: For example, if you are in doubt about

the behavior of KEEPFILTERS after a first thorough read of the book, then you can jump straight to that section. Nevertheless, reading that section without having digested the previous content might result in some frustration or, worse, an incomplete understanding of the concepts.

With that said, here is the content at a glance:

- Chapter 1 provides an introduction to how to learn DAX, with a few sections dedicated to users who already have some knowledge of other languages, such as SQL, Excel, or Python. We do not introduce any new concepts here; we simply point out the differences between DAX and other languages that you may be familiar with, as well as some suggestions on how to learn the language.
- Chapter 2 introduces the DAX language itself. We cover basic concepts such as calculated columns, measures, visual calculations, functions, and error-handling functions. We also list most of the basic functions of the language, providing some examples of their usage. The goal of the chapter is to introduce the main topics of the book, without going into too many details.
- Chapter 3 introduces the critical concepts of the evaluation context, the filter context, and the CALCULATE function. Despite being technically precise, the chapter does not go into the numerous details and intricacies of CALCULATE; it serves as an introduction that is needed to start showcasing useful code.
- Chapter 4 puts the theory introduced in Chapter 3 into practice by showing several examples of filter context manipulation. The goal of the chapter is to consolidate the theory with examples of increasing complexity.
- Chapter 5 introduces the row context and the context transition. Although this chapter serves as an introduction, it also marks the conclusion of the main theoretical overview of DAX.
- Chapter 6 describes in detail the most commonly used table functions in DAX. Having completed the theoretical introduction, each function is explained in detail, along with examples.
- Chapter 7 describes variables. Variables are used throughout the entire book. In this chapter, we provide a detailed description of the syntax and behavior of variables.
- Chapter 8 is the watershed of the book. From this chapter onward, we no longer aim to introduce concepts; instead, we go deeper and explore technical details. From now on, the goal is to understand the inner behavior of complex formulas. CALCULATE modifiers, the CALCULATE algorithm, details about the context transition, and expanded tables are some of the topics covered in the chapter.

- Chapter 9 is a set of complex examples that require advanced DAX knowledge introduced in Chapter 8. Examples are now quite complex, and the formulas conceal a significant amount of complexity. This chapter is necessary to consolidate the knowledge gained in Chapter 8.
- Chapter 10 introduces user-defined functions, along with the parameter passing mode and several examples where the correct parameter passing mode changes the result of a function.
- Chapter 11 is dedicated to functions and features to manipulate tables. Creating temporary tables in measures is an important skill that requires knowledge of several table functions and how to mix them in complex formulas.
- Chapter 12 is a thorough description of window functions. Window functions have some peculiarities that require a more in-depth understanding of concepts such as apply semantics and sorted tables. It is an essential chapter for anybody who wants to use window functions in DAX.
- Chapter 13 is dedicated to time intelligence calculations. In this chapter, we describe the theoretical foundation of time-related calculations and the way calendar-based time intelligence works, comparing it to classic time intelligence. The chapter goes very deep into the details of how time intelligence works, and it is an important chapter to read before implementing any time-related calculations.
- Chapter 14 describes the implementation details of visual calculations. It not only shows how to compute simple visual calculations, but it also goes much deeper to help you understand how visual calculations are implemented through the `VISUAL SHAPE` declaration of table variables in queries.
- Chapter 15 describes in detail how to implement calculation groups. Calculation groups are a powerful modeling tool. This chapter describes how to create and use calculation groups by introducing the basic concepts and showing a few examples.
- Chapter 16 is dedicated to functions that inspect the content of the filter context and create calculations that are dependent on hierarchies, like parent/child hierarchy calculations.
- Chapter 17 describes how to write DAX queries. It displays the complete syntax of the `EVALUATE` statement, along with additional details about functions primarily used in queries, such as `SUMMARIZECOLUMNS`.
- Chapter 18 is a collection of advanced DAX topics: expanded tables, `ALLSELECTED` and shadow filter contexts, the unique auto-exists behavior of `SUMMARIZECOLUMNS`, and the differences between filtering tables and filtering columns in measures.

- Chapter 19 is the final chapter, with advanced examples of DAX calculations. We demonstrate how to perform budget-related calculations, work with semi-additive measures, and conduct same-store calculations. Each example illustrates different approaches to authoring the code and explores the advantages and disadvantages of each choice.

Even after increasing the number of pages and removing the chapters about optimization, we still did not have enough pages to cover all the details we wanted to. Actually, some details are so intricate that learning them would be mostly unproductive. They may be interesting, but you can become a DAX guru without knowing the details of the algorithm used by the classic time intelligence to detect full months, to name an example of unnecessary knowledge. Therefore, we removed details we considered superfluous. We only go as far in the explanation of all the topics as what we deem helpful for most developers. When we had to leave some details out, we referenced articles that further explain the specific topic—because if you are curious like we are, you may want to read that unnecessary detail we skipped in the book to save pages and time!

Conventions

The following conventions are used in this book:

- **Boldface** type is used to highlight important concepts.
- *Italic* type is used to indicate new terms.
- The first letters of the names of dialogs, dialog elements, and commands are capitalized—for example, the Save As dialog.

About the companion content

We have included companion content to enrich your learning experience. The companion content for this book can be downloaded from the following page:

MicrosoftPressStore.com/guideDAX3e/downloads

The companion content includes a separate Power BI Desktop model for each figure of the book. Every figure has its own file. The data model is almost always the same, but you can use these files to follow the steps outlined in the book closely.

The demo files were created using the Contoso Data Generator available at *www.sqlbi.com/tools/contoso-data-generator/*. **There is no need to refresh the content of the Power BI Desktop files, because all the data is already available in the files themselves.** If you want to change the models, you will need to download the database backup from the Contoso Data Generator webpage and restore it on your own SQL Server.

Errata, updates, and book support

We've made every effort to ensure the accuracy of this book and its companion content. You can access updates to this book—in the form of a list of submitted errata and their related corrections—at

MicrosoftPressStore.com/guideDAX3e/errata

If you discover an error that is not already listed, please submit it to us at the same page.

For additional book support and information, please visit

MicrosoftPressStore.com/Support

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to

support.microsoft.com

CHAPTER 3

Introducing the filter context and CALCULATE

In the previous chapter, we introduced the basic concepts of DAX and showcased some of the most used functions. However, the real power of DAX is not in knowing each function; the real power of DAX is in manipulating the filter context, along with a good understanding of the row context. In this chapter, we introduce the filter context and the CALCULATE function in their simplest form. We will not explore the most obscure intricacies of the evaluation context and CALCULATE. However, trust us when we say there are many of those complexities. As with any other journey, the trip to learning the evaluation context starts with the first step.

We want to give a few words of warning to our readers. The concept of evaluation context is simple, and you will learn and understand it soon. Nevertheless, you need to thoroughly understand several subtle considerations and details before you really master DAX. Otherwise, you will feel lost at a certain point on your DAX learning path. We have been teaching DAX to thousands of students in public and private classes, so we know this is normal.

The content of this chapter is not enough to explain the intricacies of the DAX evaluation context. It is a good starting point, but there will be many more details to learn. Hence, if some code is unclear, do not worry too much. Things will become clearer and clearer in the process of reading the first chapters of this book. Then, after the theoretical background is established, you can look at the code with a deep understanding of how it works.

Introducing axis and coordinates

Before introducing the filter context, let's introduce the concept of the axis of a visual and the coordinates of a point in the visual. A Power BI visual may have zero, one, or two axes. A card visual has no axis because the content of the visual is just a number. A table visual has one axis: the rows. A matrix has two axes: rows and columns. In Figure 3-1, you can see these three visuals with their axes.

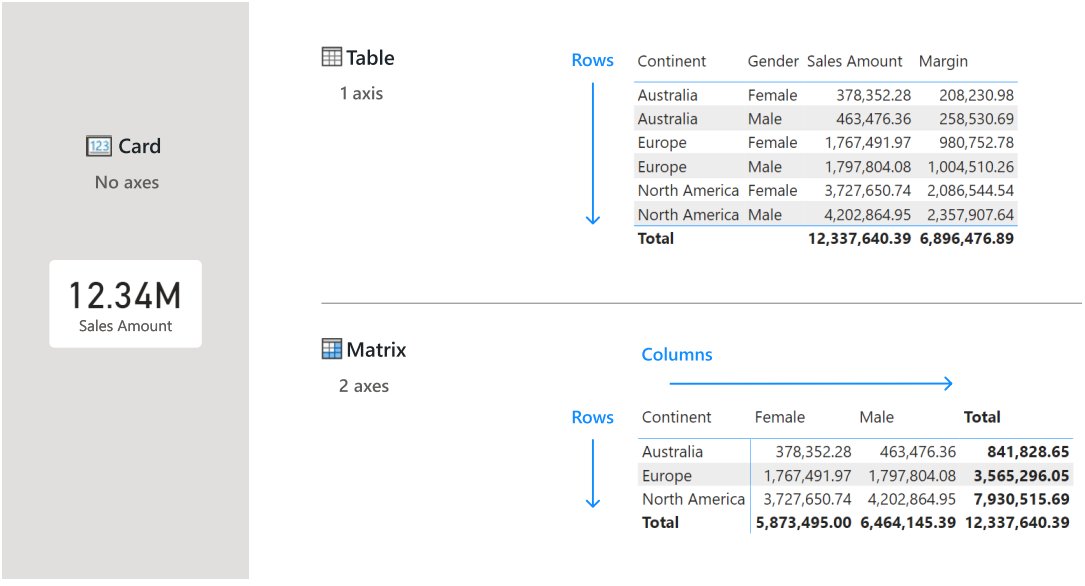

FIGURE 3-1 Power BI visuals can have zero, one, or two axes.

Based on the axes available for a visual, we can define a coordinate system to describe one individual cell of the visual. For the card visual, there are no coordinates. Therefore, there is no need to define a coordinate system: We can reference the one and only cell of the visual just by the cell or *the value*.

When it comes to the table visual, there is only one dimension: the rows axis. In our example, on the rows axis, there are two model columns from the `Customer` table: `Customer[Continent]` and `Customer[Gender]`. We provide the values for the two columns to define a row. For example, the pair `(Australia, Female)` defines the first row of the table visual. It is useful to note that `(Australia, Female)` defines a row of the visual, not just a single cell. To point out a single cell, we must also provide a measure, like `Sales Amount`. However, the measure does not belong to the coordinate system. The measure defines the calculation happening in that cell. The coordinates of a cell are made up of the model table's column names. In other words, both `Sales Amount` and `Margin` share the same coordinates `(Australia, Female)`.

In a matrix visual, the coordinates are even clearer, mainly because we used only one measure. Both the rows and the columns of the matrix contribute to the coordinates of a cell. In the matrix visual shown, there is only one measure (`Sales Amount`), and the coordinate system is identical to the one used in the table visual: the coordinates of a cell are made up of the model table's column names. The matrix visual would be harder to read if we used two measures. However, as was the case with the table visual, the measure does not contribute to the coordinates; it only defines the algorithm executed in that cell.

It is worth noting that—given the same set of coordinates—a measure computes the same value in any visual. The `(Australia, Female)` combination computes the value of `Sales Amount` identically in the table and matrix visuals.

The coordinates of a cell contribute to the definition of the filter under which a formula is computed. In a more complex scenario, where the coordinate system includes more columns, the filter becomes more complex, as shown in Figure 3-2.

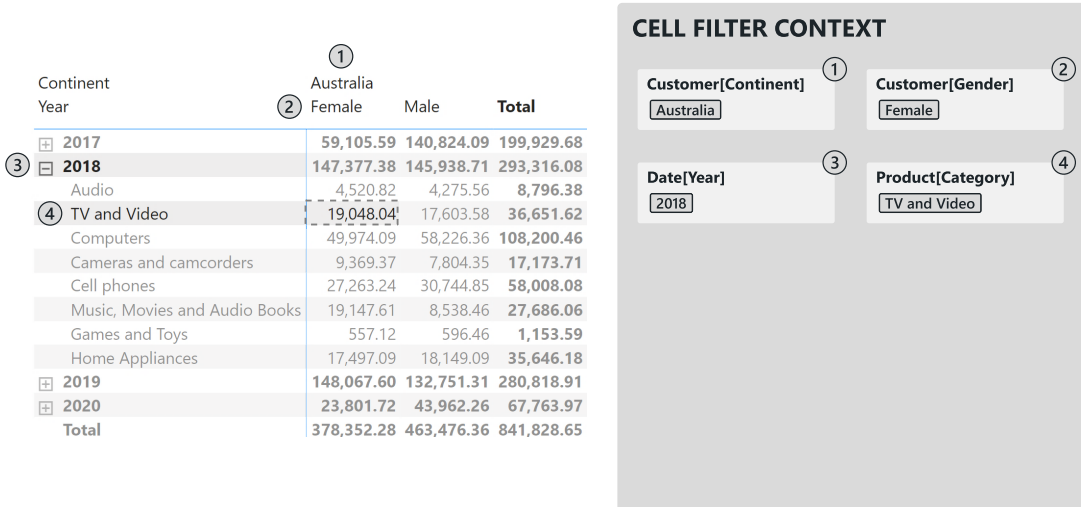

FIGURE 3-2 All the coordinates of a cell contribute to creating the filter context.

The filter operating on a cell is the filter context. Be mindful that the coordinates of a cell are just one part of the filter context. The filter context is a bit more complex than just the set of coordinates, as we will find out in the next section. For now, consider the coordinates and the filter context to be the same thing.

A concept that may not be obvious at first sight is the coordinates of subtotals. The coordinates of subtotals do not include some of the axes' elements. For example, in Figure 3-3, you can see the coordinates of a subtotal for both the rows and columns axes: It does not include either the gender or the category.

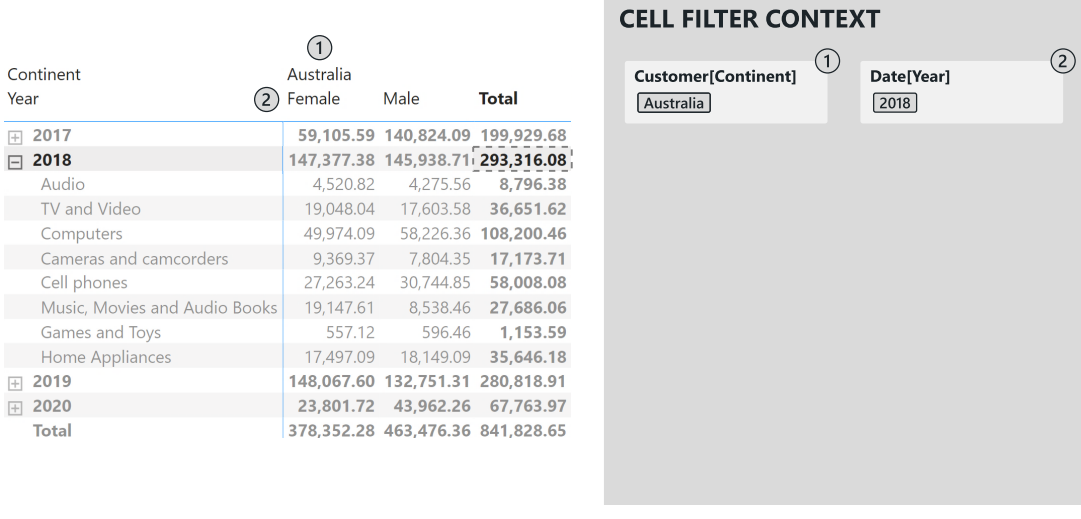

FIGURE 3-3 Coordinates of subtotal cells do not filter all the columns of the axes.

A Power BI visual can be more complex than that. For example, using small multiples, one could create a visual that—by definition—contains multiple visuals. Or it could be that the axes are not immediately recognizable. For example, the visual in Figure 3-4 shows multiple charts by category, each with the year in the x-axis and the customer continent as the legend.

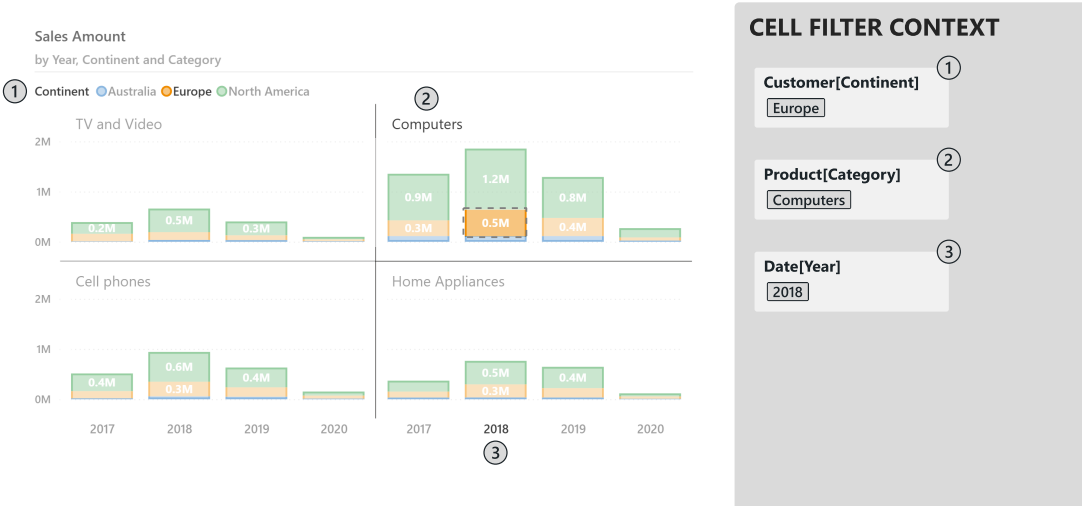

FIGURE 3-4 In a complex visual, each cell has a defined set of filters, even though the rows and columns axes are less recognizable.

Whether the legend should be the rows, the columns, or the filter of the visual context is debatable. The same is true for the column used for small multiples. Despite the concept being somewhat ambiguous, we just rely on an intuitive approach. From a DAX standpoint, it does not matter whether a filter is on the rows, columns, small multiples, legend, or filter. In the end, regardless of where it comes from, it is a filter that contributes to the filter context under which a formula is being evaluated. Indeed, coordinates are mainly useful in defining the filter context, as we will find out in the next section.

Introducing the filter context

The evaluation context has two parts: the filter context and the row context. This chapter introduces the most important of the two: the filter context. We will describe the row context in Chapter 5, "Introducing the row context and the context transition," after we understand the filter context well.

The filter context is the set of filters, mostly created by visuals, under which the DAX code is evaluated. For example, the report in Figure 3-5 shows the `Sales Amount` sliced by `Product[Brand]`.

Brand	Sales Amount
A. Datum	147,687.44
Adventure Works	2,761,057.66
Contoso	2,227,244.32
Fabrikam	990,275.08
Litware	506,104.50
Northwind Traders	119,857.67
Proseware	956,335.76
Southridge Video	776,807.78
Tailspin Toys	79,159.15
The Phone Company	1,976,180.03
Wide World Importers	1,796,930.99
Total	**12,337,640.39**

FIGURE 3-5 The report shows the `Sales Amount` measure sliced by brand.

Each row shows the `Sales Amount` of the given `Brand`. Despite the behavior being very intuitive, it is important to understand how DAX generates the different numbers. Every cell of the matrix computes the same formula: the `Sales Amount` measure. The `Sales Amount` code is a simple `SUMX` over the `Sales` table:

Measure in the Sales table

```
Sales Amount =
SUMX (
    Sales,
    Sales[Quantity] * Sales[Net Price]
)
```

The `Sales Amount` measure does not include any filter. The formula scans `Sales` and computes the multiplication of `Quantity` times `Net Price` as part of the iteration. However, every cell in the matrix shows a different result, indicating that the calculation is happening under a filter. The filter is not inside the formula but outside the DAX code. This is why we call this filter a context: It is a filter that the formula finds itself in. The context filters the model; hence the name, filter context.

Each cell of the matrix is executed in its own filter context. In this first simple example, the filter context is entirely defined by the coordinates of the cell. In each cell, the value of `Sales Amount` is computed by filtering the cell coordinates. When the code of the measure is being executed, the filter is active, thus reducing the number of rows in `Sales` that are visible to `SUMX`. Figure 3-6 shows how the filter originates from the row in the matrix and is transferred to the formula in the code of the measure.

	Brand	Sales Amount
	A. Datum	147,687.44
	Adventure Works	2,761,057.66
	Contoso	2,227,244.32
	Fabrikam	990,275.08
	Litware	506,104.50
	Northwind Traders	119,857.67
	Proseware	956,335.76
	Southridge Video	776,807.78
①	Tailspin Toys	79,159.15
	The Phone Company	1,976,180.03
	Wide World Importers	1,796,930.99
	Total	**12,337,640.39**

CELL FILTER CONTEXT

① **Product[Brand]**
Tailspin Toys

```
SUMX (
    Sales,
    Sales[Quantity] * Sales[Net Price]
)
```

FIGURE 3-6 The filter context originates from the row of the matrix, and it affects the code of the measure.

The filter context determines the calculation in the cell, as it filters which content to apply the formula to. The filter context filters columns in the data model. In our example, the filter context filters the `Product[Brand]` column, allowing only the row for Tailspin Toys to be visible.

Different query languages use different ways of applying filters. In DAX, we use `CALCULATE` to create or modify the filter context. Therefore, a more correct and DAX-like representation of the filter being applied to the measure is the one in Figure 3-7.

	Brand	Sales Amount
	A. Datum	147,687.44
	Adventure Works	2,761,057.66
	Contoso	2,227,244.32
	Fabrikam	990,275.08
	Litware	506,104.50
	Northwind Traders	119,857.67
	Proseware	956,335.76
	Southridge Video	776,807.78
①	Tailspin Toys	79,159.15
	The Phone Company	1,976,180.03
	Wide World Importers	1,796,930.99
	Total	**12,337,640.39**

CELL FILTER CONTEXT

① **Product[Brand]**
Tailspin Toys

```
SUMX (
    Sales,
    Sales[Quantity] * Sales[Net Price]
)
```

EQUIVALENT DAX CODE *fx*

```
CALCULATE (
    SUMX (
        Sales,
        Sales[Quantity] * Sales[Net Price]
    ),
  ① Product[Brand] = "Tailspin Toys"
)
```

FIGURE 3-7 CALCULATE is the function that modifies the filter context in DAX.

Before we move forward, let's say a few words about CALCULATE. CALCULATE accepts many arguments. The first is the expression to compute. Starting from the second argument, CALCULATE receives the filters to create or modify the filter context. The filters are applied before the first argument can be computed; therefore, they must be evaluated first. This makes CALCULATE somewhat counterintuitive. CALCULATE starts evaluating its second argument, and then its third, fourth, and so on; only at the end does CALCULATE compute its first argument after creating the new filter context. Despite this being somewhat counterintuitive, you will use CALCULATE so often that it will become natural once you are used to it.

A matrix can be more complex than the one shown in this first example. Adding more model columns to the rows axis is possible, like in Figure 3-8. However, the mechanism is the same: In the eyes of CALCULATE, each additional column just acts as an additional filter.

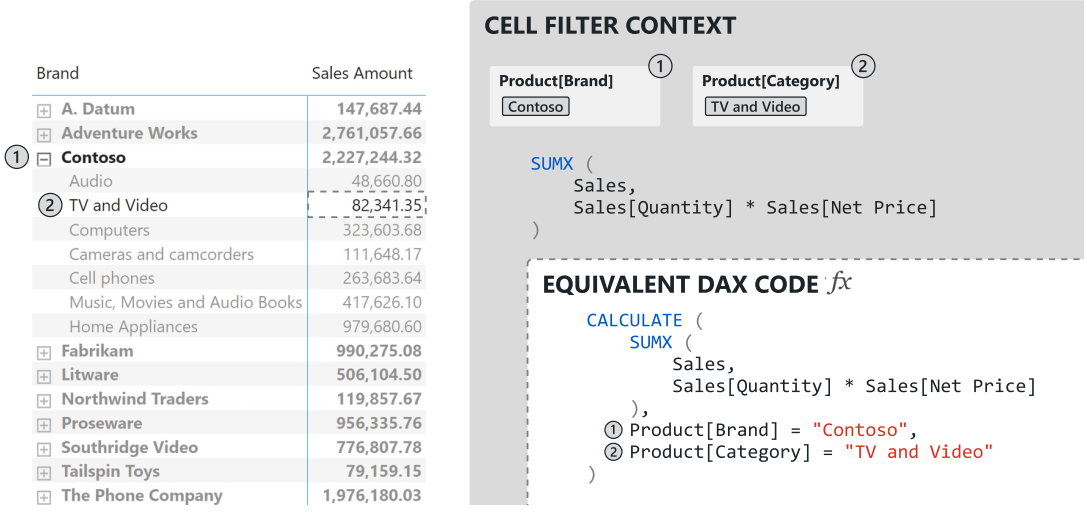

FIGURE 3-8 Multiple filters can be used in CALCULATE, and they add to the filter context.

CALCULATE accepts any number of arguments as filters. They are all intersected, meaning that all the filters work together at once. Indeed, if we further modify the matrix by adding the years on the columns, that operation changes the filter but not the mechanism used to create the filter itself. In Figure 3-9, we simplify the diagram by using one box for the visual representation of the filter context and another box for the equivalent CALCULATE formula without starting from the original measure definition.

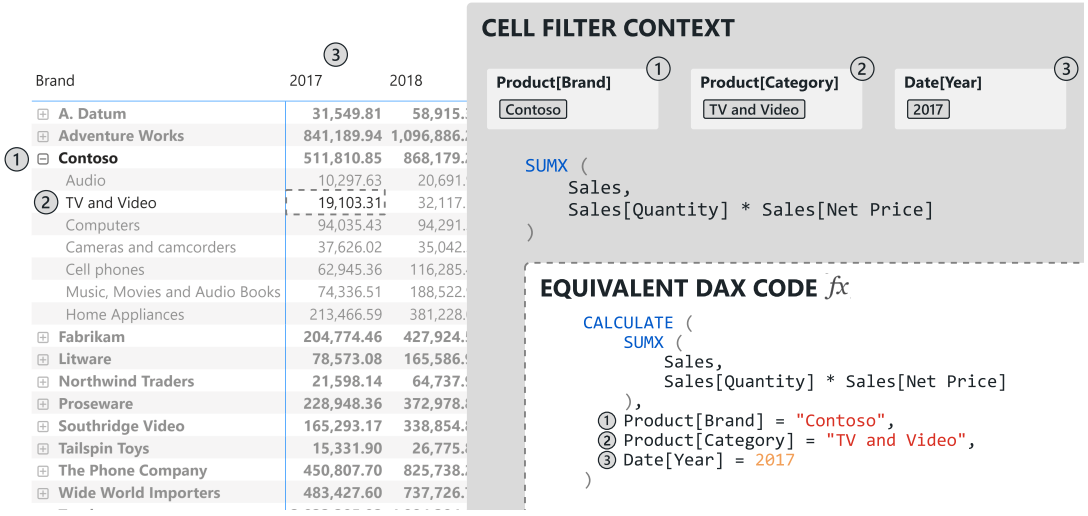

FIGURE 3-9 Values on rows and columns all contribute to the filter context of a cell.

You might notice that the filters applied by CALCULATE operate on `Product` and `Date`, whereas the measure scans `Sales`. At this point, a natural question one might have becomes, how do the filters propagate from one table to another? The answer lies in the presence of relationships.

Placing a filter on one table automatically propagates its effect to any table with a relationship with the original table, following the cross-filter direction of the relationship. The cross-filter direction of a relationship is visible in the diagram view of Power BI as a small arrow in the middle of the relationship itself, as visible in Figure 3-10.

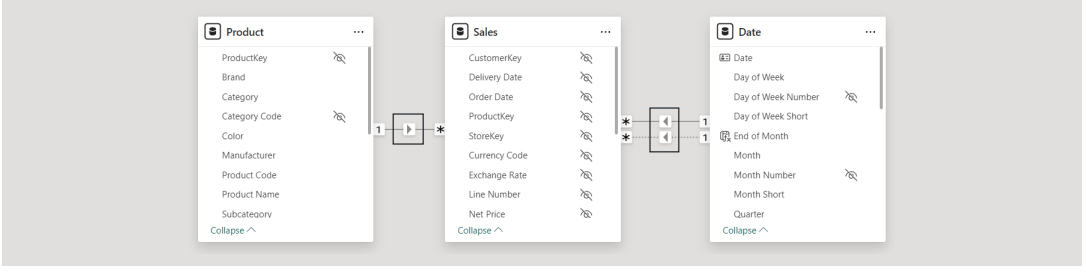

FIGURE 3-10 Relationships define how the filter context propagates the filters through the cross-filter direction.

As you can see, both `Product` and `Date` filter `Sales`. Therefore, when CALCULATE applies a filter on either `Product` or `Date`, the filter is automatically transferred to `Sales`. It is worth remembering that the structure of the data model is an integral part of the DAX code. Changing the model changes the way numbers are computed. Namely, changing the cross-filter direction of a relationship strongly impacts how the filters are transferred.

The filter context is generated not only by the coordinates of the current visual but also by other visuals. In a Power BI report, multiple visuals may contribute to the filter context of one cell—slicers,

for instance, do this. Whenever you select one or more items in a slicer, the selection generates a filter added to the filter context, as shown in Figure 3-11.

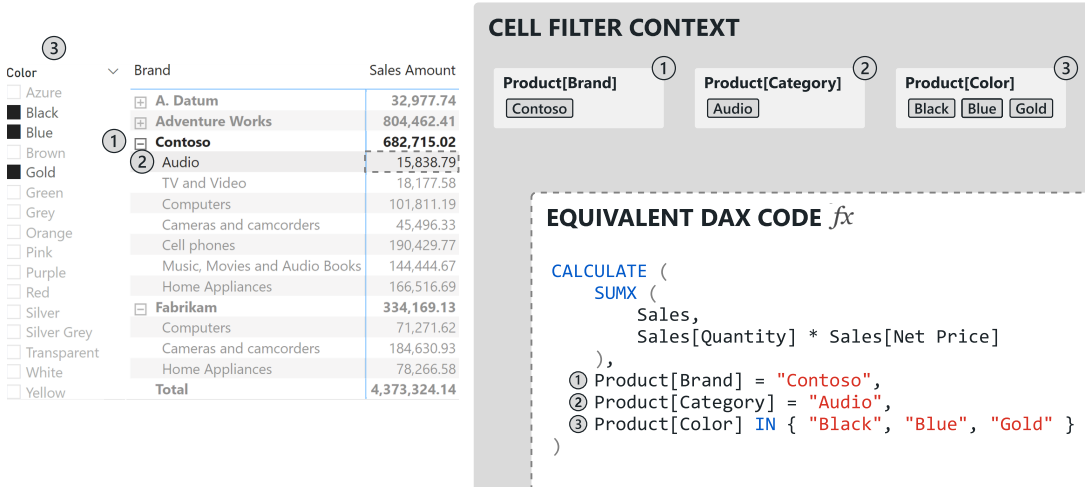

FIGURE 3-11 Slicers and other visuals contribute to the filter context of a cell.

Whenever developers need to focus on a cell and understand its filter context, they should consider both the current visual and any other visual cross-filtering it.

A filter context is active in every cell of the matrix. The columns that contribute to its definition are the columns from the model that contribute to the coordinates of the cell. Think about the subtotals, as shown in Figure 3-12: The coordinates of the subtotals do not include `Product[Category]`. Hence, `Product[Category]` is not being filtered in the cell filter context.

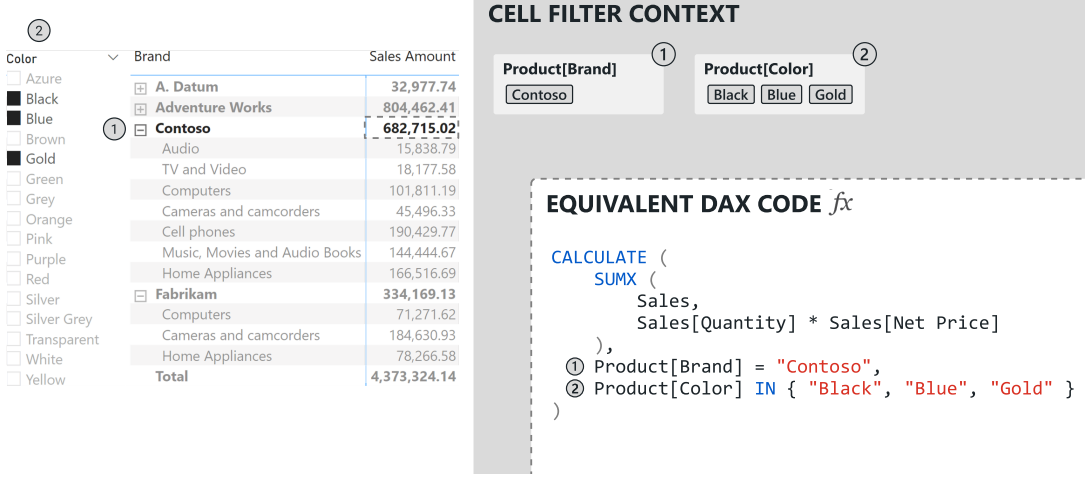

FIGURE 3-12 At the subtotal level, the more detailed columns do not contribute to the cell filter context.

You have now seen several examples, so the rules of the game should be clear by now.

For the sake of clarity, as we expand on the above, we'll refer to the visual we're focusing on and whose filter context we are analyzing as the "current visual." Each visual in the report can cross-filter other visuals and, therefore, add its own filter to the filter context of a cell in the current visual. Most visuals can apply filters. Slicer selection is a standard filter, but for example, the selection of elements in a bar chart can also cross-filter other visuals. A selection you may make in a visual other than the current visual will add filters on all the columns used to create the current visual. In a matrix, both rows and column axes contribute to the filter. In a bar chart, individual bars add their filter context to the filter context of the current visual.

So far, we have expanded the `Sales Amount` code in all the figures to make the code being executed more evident. From now on, we will use a more compact representation, using only the measure name. The compact form is shown in Figure 3-13.

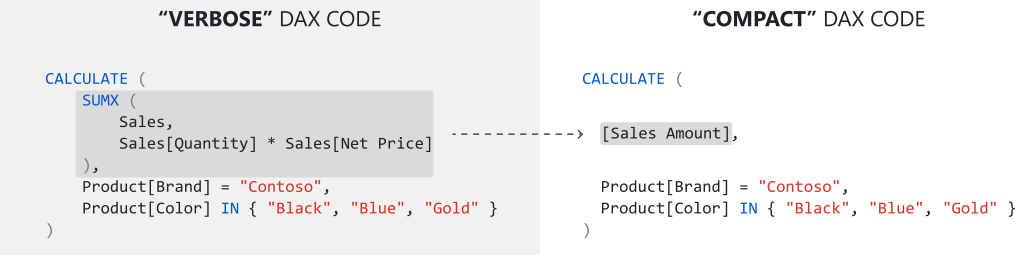

FIGURE 3-13 In this figure, the code executed by CALCULATE is the measure name, no longer its expansion.

We will expand the code occasionally when we need to go into more detail. However, the compact version is shorter and easier to read.

Let's briefly recap what we have learned so far about the filter context:

- Each visual has its filter context. We call it the visual filter context. The visual filter context is determined by all the other visuals in the report that interact with that current visual and all the report filters applied through the filters pane (visual-, page-, and report-level filters). Slicers, column charts, line charts…most of the visuals in a report contribute to the filter context of the current visual.
- Each cell has its filter context. We call it the *cell filter context*. The coordinates of the cell itself determine the filter context of a cell, plus the visual filter context. The value shown for a cell is determined by the measure used in the cell, which is executed in the cell filter context.
- At the subtotal level, the cell coordinates do not include all the columns. Therefore, the number of columns filtered may be different for different cells.
- At the grand-total level, the cell filter context is the same as the visual filter context because the coordinates of the grand total do not include any column of the visual axes.

Being able to detect the filter context of a cell quickly is of paramount importance when developing DAX formulas. We will analyze several scenarios where the filter context is much more complex than the simple examples provided. However, we proceed one step at a time; for now, the goal is only to get a feeling of what the filter context is.

Introducing CALCULATE

As you may have noticed, we have already introduced CALCULATE in most examples so far. CALCULATE is a function that changes the filter context. It adds or removes filters depending on the developer's needs. In all previous examples, we used CALCULATE in the figures to show the different filter contexts active in different cells. CALCULATE is a DAX function that you can use to manipulate the filter context in any formula.

CALCULATE is a complex function. Its behavior is hard to learn and memorize, as are the side effects of changing the filter context. However, in this section, we want to start using CALCULATE in an easy way so that we can get acquainted with its behavior.

Let's start with a few simple examples. If we want to compute the sales of red items, we can use CALCULATE to modify the filter context inside a measure:

Measure in the Sales table

```
Red Sales =
CALCULATE (
    [Sales Amount],
    'Product'[Color] = "Red"
)
```

This time, we do not use CALCULATE to describe the filter context of a cell in DAX. We are using CALCULATE in a measure to change the cell filter context by adding a filter for the color. When used in a matrix, the `Red Sales` measure shows a smaller number than the original `Sales Amount`, as shown in Figure 3-14.

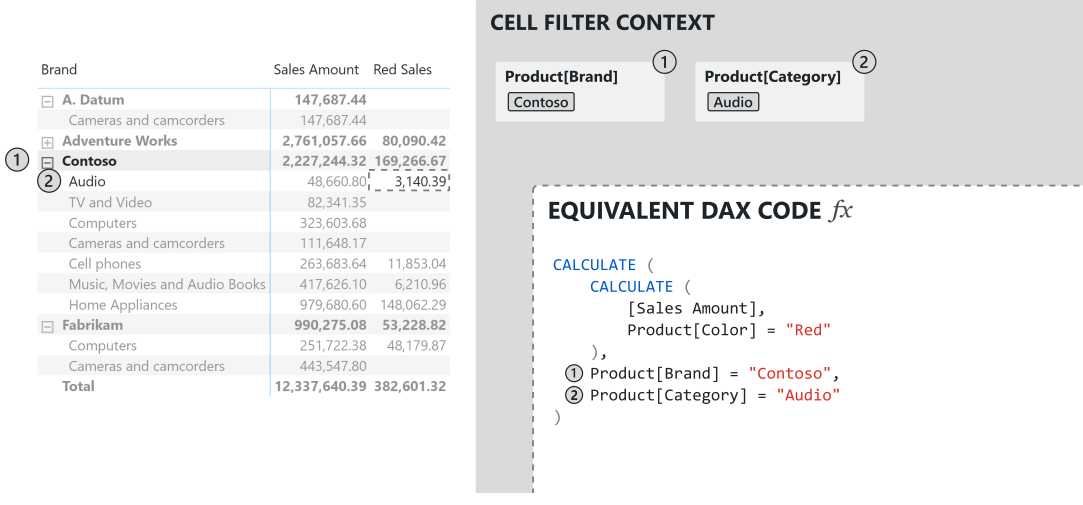

FIGURE 3-14 Because the measure uses CALCULATE, we have two nested CALCULATE functions.

In the figure, CALCULATE shows the cell filter context applied to the evaluation of `Red Sales`. However, `Red Sales` contains another CALCULATE. Therefore, the code being executed contains two nested CALCULATE functions. Here is the code with `Red Sales` expanded:

```
CALCULATE (
    CALCULATE (
        [Sales Amount],
        Product[Color] = "Red"
    ),
    Product[Brand] = "Contoso",
    Product[Category] = "Audio"
)
```

The outer CALCULATE defines the filter context of the cell; the expanded code of `Red Sales` introduces the inner CALCULATE. In this scenario, the net effect of the two CALCULATE functions is that all the filters are active simultaneously when `Sales Amount` is being executed. Therefore, the cell shows the sales of `(Contoso, Audio, Red)`. In Figure 3-15, you can see the effect of combining the two filters.

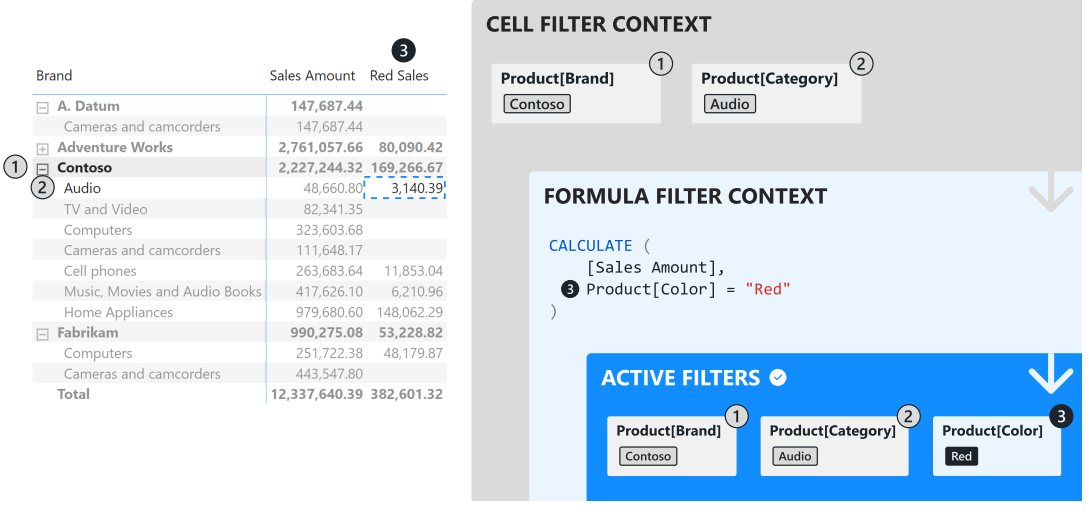

FIGURE 3-15 The filters created by the two CALCULATE functions are combined.

In this example, the two filters are intersected because they operate on different columns. Later in this chapter, we analyze what happens when two CALCULATE functions filter the same column.

When dealing with CALCULATE, we often refer to two filter contexts: the outer filter context and the inner filter context, as shown in Figure 3-16.

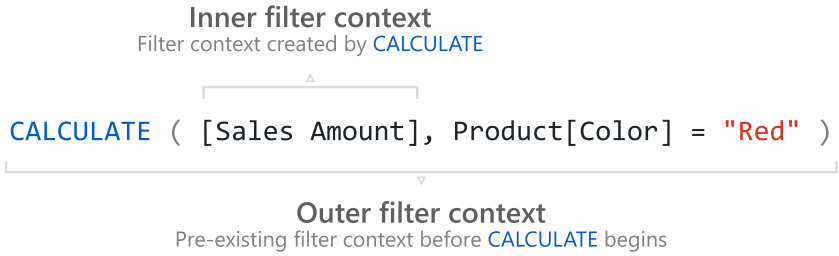

FIGURE 3-16 The inner filter context is generated by CALCULATE.

The outer filter context is the filter context that is active when CALCULATE is invoked. The inner filter context is the new filter context created by CALCULATE and active only during the execution of its first argument (`Sales Amount` in the example).

In general, the concept of inner and outer filter context is broader. When a DAX formula is executed, multiple levels of filter contexts are combined. Figure 3-17 depicts the various filter contexts that—once merged—create the active filter context of any piece of DAX code.

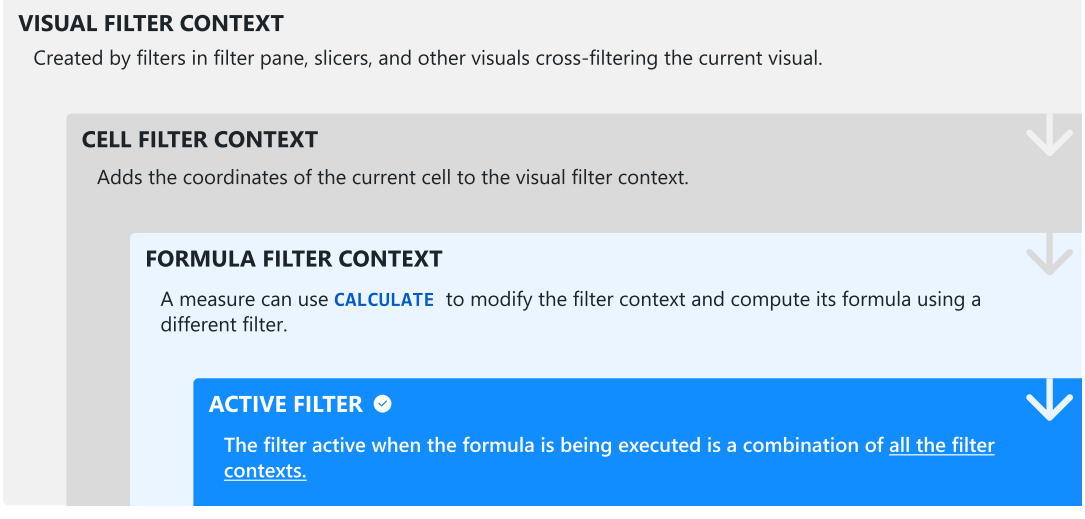

FIGURE 3-17 Various levels of filter contexts are merged to create the filter context of a DAX formula.

As you see, the scenario is more complex once you start diving into the details. Multiple filter contexts are merged to create the filter under which your DAX code is being executed, and this is just the beginning! The more details we are going to learn, the more complex it will be. However, step by step, we are about to uncover all the secrets of DAX evaluations.

Before we move further with CALCULATE, note how you can express filters in CALCULATE. A filter in CALCULATE can be one of three types:

- A condition, like `Product[Color] = "Red"`. This is the most common type of condition, where you specify one or more values for a set of columns.
- A modifier, such as REMOVEFILTERS or USERELATIONSHIP. (We will introduce these in the "Introducing REMOVEFILTERS" section later in this chapter and in the "Understanding USERELATIONSHIP" section in Chapter 8, respectively.) In this case, the modifier instructs CALCULATE to perform an action: REMOVEFILTERS removes filters from the filter context, while USERELATIONSHIP activates an inactive relationship. We will cover other modifiers later on. For now, it is enough to know that they are not filters; they are instructions to CALCULATE to behave in a particular way.
- A table. If you use a table as a filter, the filter is applied to the columns present in the table with an IN condition.

Because we are at the beginning of our DAX journey, we consider a table to be a special type of condition. Hence, we avoid using tables and use an IN syntax to clarify that the filter is indeed a condition. Therefore, we will use formulas like the following:

```
CALCULATE (
    [Sales Amount],
    Product[Color] IN VALUES ( Product[Color] )
)
```

In later chapters, when we learn more about CALCULATE, we will see that tables are not a special type of condition but rather the opposite: conditions are just a shortcut for tables. In CALCULATE, the conditions are tables. However, for a few chapters, we will keep a simplified version of CALCULATE with conditions only. From time to time, when needed, we will use tables to help you start getting used to them.

Introducing KEEPFILTERS

It is interesting to observe what happens when there is a conflict between the cell filter context and the CALCULATE filter context. If we use the `Product[Color]` column on the rows, rather than brand and category, the result shows the same value (sales of red products) on all the rows, as shown in Figure 3-18.

Color	Sales Amount	Red Sales
Azure	10,743.25	382,601.32
Black	3,330,387.51	382,601.32
Blue	595,741.34	382,601.32
Brown	885,496.17	382,601.32
Gold	447,195.29	382,601.32
Green	263,158.45	382,601.32
Grey	907,037.74	382,601.32
Orange	198,300.01	382,601.32
Pink	189,533.53	382,601.32
Purple	8,216.37	382,601.32
Red	382,601.32	382,601.32
Silver	2,336,604.05	382,601.32
Silver Grey	25,414.71	382,601.32
Transparent	159.63	382,601.32
White	2,723,494.66	382,601.32
Yellow	33,556.32	382,601.32
Total	**12,337,640.39**	**382,601.32**

FIGURE 3-18 The Red `Sales` measure always returns the same number, irrespective of the color in the row.

This time, the filter context created by CALCULATE conflicts with the filter context created by the cell. In such scenarios, the inner filter context created by CALCULATE overrides the outer filter, thus replacing the filter on Gold with a filter on Red. As a rule, when a conflict exists, the inner filter context overrides the outer filter context.

This is the standard behavior of CALCULATE. When CALCULATE places a filter on a column that already had filters set by previous functions (or by the Power BI visual), the filter placed by CALCULATE replaces the previous filter, as shown in Figure 3-19.

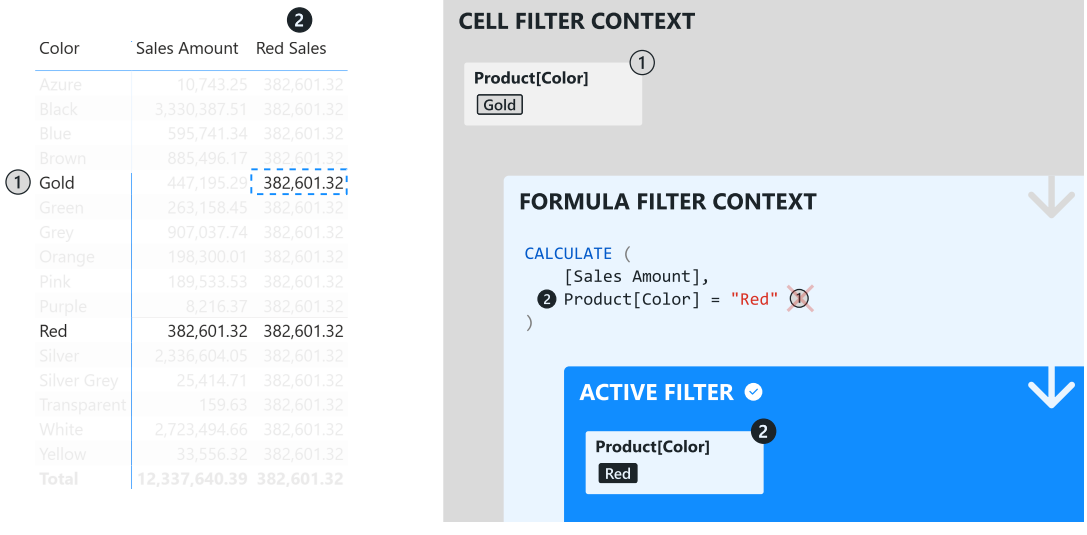

FIGURE 3-19 The inner filter context set by CALCULATE overwrites the outer filter.

There are no special reasons for this behavior. It is what it is. DAX could have had a different semantics where the filters are merged rather than a mere replacement. However, one option had to be chosen, and the choice was filter overwriting.

This behavior can be changed by using KEEPFILTERS. KEEPFILTERS is a CALCULATE modifier that changes how CALCULATE uses the new filters to generate the inner filter context. When used, KEEPFILTERS instructs CALCULATE not to remove the outer filter but to maintain it together with the new filter. Look at the code of the measure, this time with KEEPFILTERS:

Measure in the Sales table

```
Red Sales =
CALCULATE (
    [Sales Amount],
    KEEPFILTERS ( 'Product'[Color] = "Red" )
)
```

Once used in the matrix, it shows the value for Red only when Red is already selected in the matrix. In all other cases, it shows a blank.

In Figure 3-20, you can see the process executed for the row containing Gold, which shows a blank.

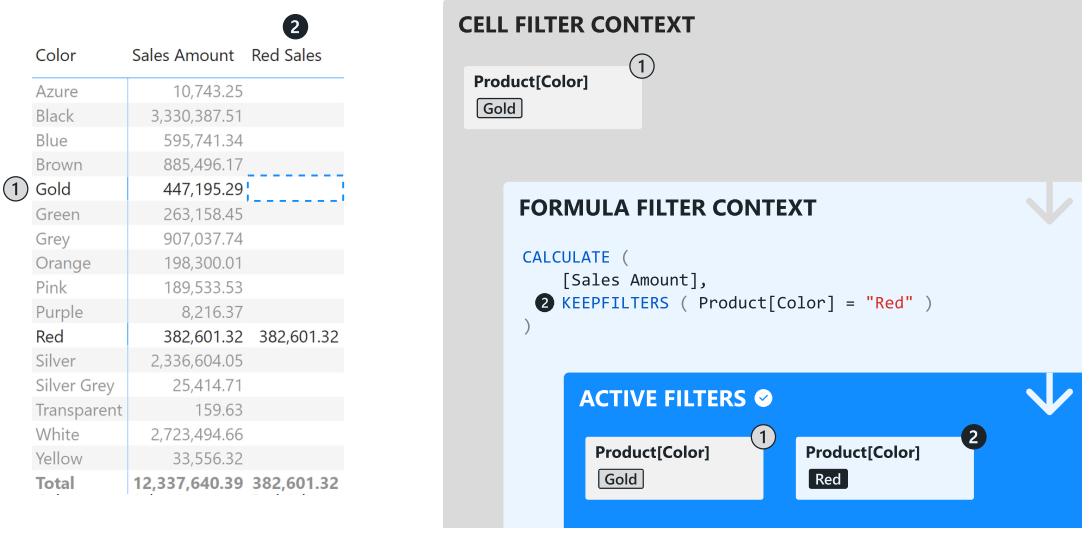

FIGURE 3-20 Using KEEPFILTERS keeps the outer filter inside the inner CALCULATE.

As you can see in Figure 3-20, the filter for Gold is maintained inside the formula filter context. Therefore, two filters are active simultaneously: one for Red and one for Gold. The intersection of the two filters is empty, generating a blank result.

KEEPFILTERS keeps the outer filter context inside the inner filter context. In other words, the code being executed is equivalent to this:

```
CALCULATE (
    [Sales Amount],
    Product[Color] = "Red",
    Product[Color] = "Gold"
)
```

The two filters working together produce an empty result. The only scenario where a number is produced is when both filters (the inner and the outer) filter Red, as shown in Figure 3-21.

	Color	Sales Amount	Red Sales
	Azure	10,743.25	
	Black	3,330,387.51	
	Blue	595,741.34	
	Brown	885,496.17	
	Gold	447,195.29	
	Green	263,158.45	
	Grey	907,037.74	
	Orange	198,300.01	
	Pink	189,533.53	
	Purple	8,216.37	
1	Red	382,601.32	382,601.32
	Silver	2,336,604.05	
	Silver Grey	25,414.71	
	Transparent	159.63	
	White	2,723,494.66	
	Yellow	33,556.32	
	Total	**12,337,640.39**	**382,601.32**

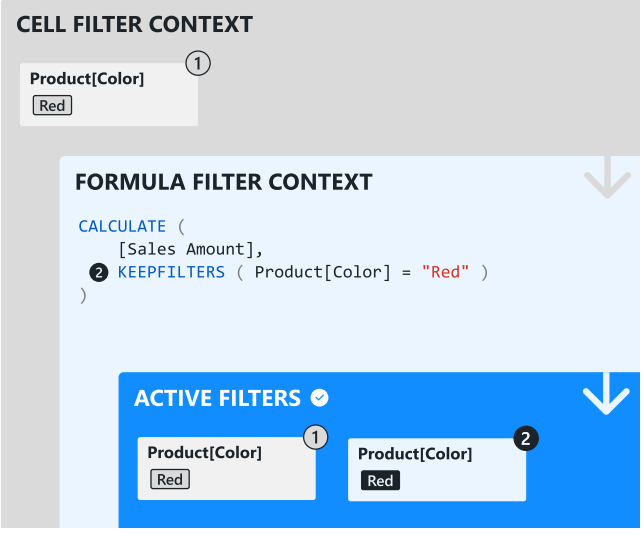

FIGURE 3-21 When both the inner and outer filters are Red, the measure computes a non-blank result.

Be mindful that both versions of `Red Sales` (the one using KEEPFILTERS and the one without KEEPFILTERS) are correct. It all depends on the calculation you want to obtain. In some scenarios, you want to remove the outer filter and replace it with a new one. In other scenarios, you want the opposite behavior.

Let's elaborate further on KEEPFILTERS with another example. We author the `Trendy Sales` measure that computes the sales of trendy colors, which are Blue, White, or Red. The code is quite simple:

Measure in the Sales table

```
Trendy Sales =
CALCULATE (
    [Sales Amount],
    'Product'[Color] IN { "Blue", "White", "Red" }
)
```

The measure works fine if no filter exists for `Product[Color]`. However, when used in conjunction with a slicer over the color, it shows a somewhat unexpected result, as shown in Figure 3-22. The sales of trendy products (`Trendy Sales`) are greater than those filtered by the selected colors (`Sales Amount`).

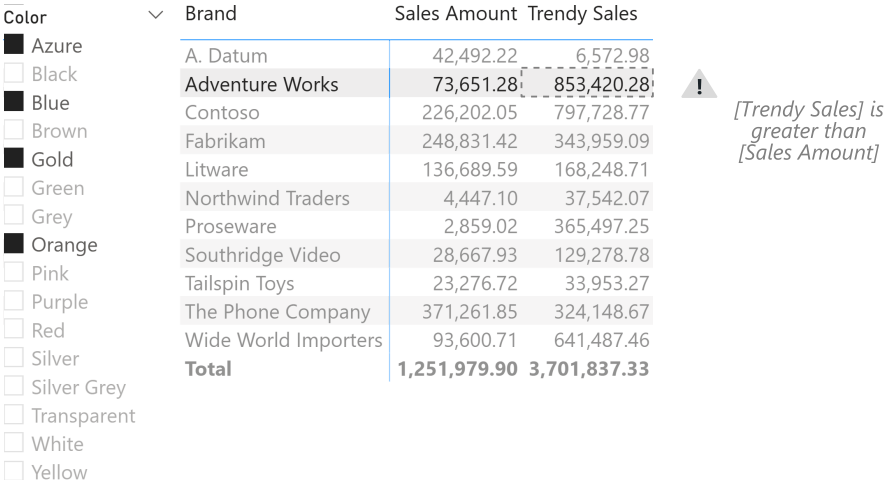

FIGURE 3-22 The filter replacement produced by CALCULATE makes numbers hard to understand.

What happens is that the inner filter on Blue, White, and Red replaces the outer filter on Azure, Blue, Gold, and Orange. In other words, the measure is computing sales for colors that—according to the slicer—should not be part of the calculation.

Even though the problem is more evident with the Adventure Works brand—because `Trendy Sales` is larger than `Sales Amount`, which is nonsensical—the same problem exists for any brand. In the following examples, we focus on "The Phone Company" rather than the "Adventure Works" brand because data distribution makes it easier to understand what happens with the filter context.

In Figure 3-23, you can see how the resulting filter context is computed in the `Trendy Sales` measure for "The Phone Company" brand.

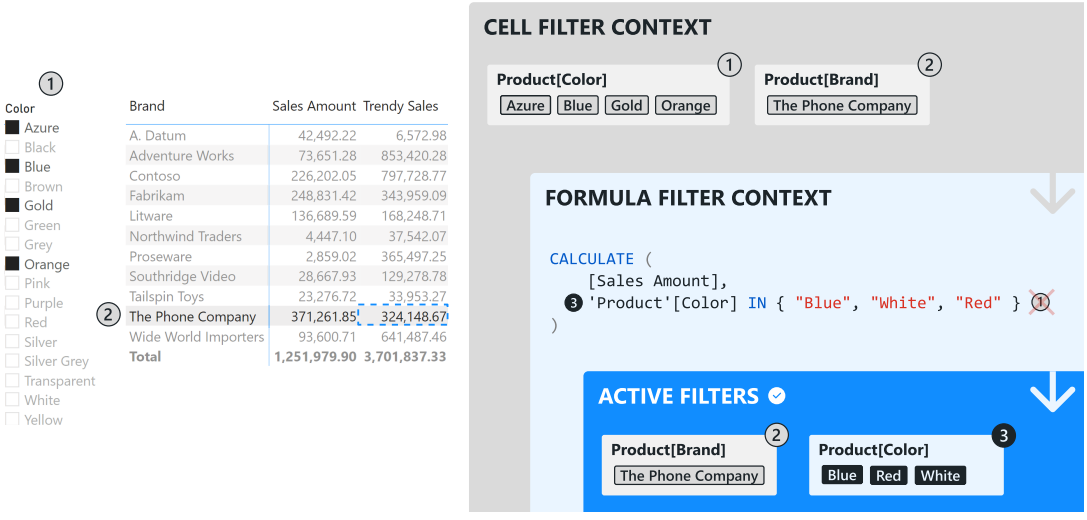

FIGURE 3-23 The filter context for the trendy colors replaces the cell filter context.

Again, this might be what you want to compute: a different set of colors that ignores the `Color` slicer, resulting in a larger amount for Adventure Works and a smaller amount for The Phone Company. However, you most likely want to abide by the slicer filters and add the trendy colors as an additional filter. In that case, `KEEPFILTERS` is the way to go:

Measure in the Sales table

```
Trendy Sales =
CALCULATE (
    [Sales Amount],
    KEEPFILTERS ( 'Product'[Color] IN { "Blue", "White", "Red" } )
)
```

In Figure 3-24, you can see that by using this version of `Trendy Sales`, both filters on color are active at the same time in the resulting filter context. The intersection between the filter coming from the slicer and the additional filter created by `CALCULATE` makes only Blue visible through this filter context because it is the only color that exists in both filters.

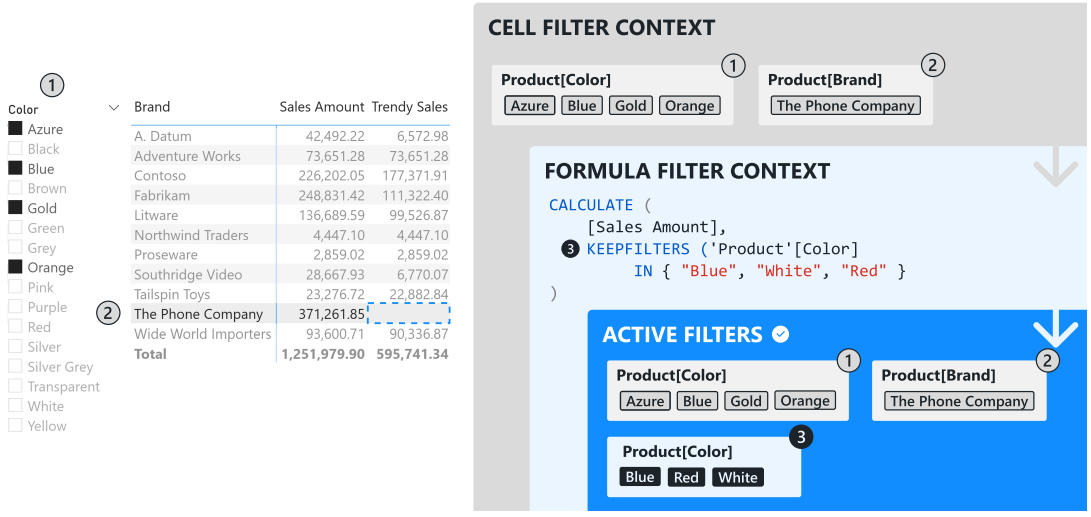

FIGURE 3-24 When both filters over `Product[Color]` are active in the resulting filter context, only Blue remains visible.

Figure 3-25 shows the result of the `Trendy Sales` version that uses `KEEPFILTERS`. The cell for The Phone Company shows an empty value because only blue products are visible in the resulting filter context, and The Phone Company has no Blue products. The cell for Adventure Works shows the same value for `Sales Amount` and `Trendy Sales` because all the selected sales are of Blue products.

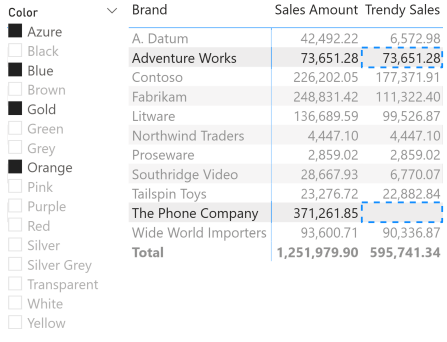

FIGURE 3-25 By using KEEPFILTERS, the cell for The Phone Company now shows a blank value and Adventure Works now shows the same value as `Sales Amount`—which is correct.

VALUES as an alternative to KEEPFILTERS

We introduced VALUES in the previous chapter. However, let's recap briefly what we know about the function before discussing how VALUES can be used as an alternative to KEEPFILTERS. The VALUES table function returns the values of a column, as currently visible in the filter context. For example, if you need to compute the number of colors sold by each brand, you can author a measure that just counts the number of rows in the result of VALUES:

Measure in the Sales table

```
# Colors = COUNTROWS ( VALUES ( 'Product'[Color] ) )
```

When used in a matrix, the measure shows the number of colors of each brand. Indeed, VALUES is evaluated in the cell filter context, which only shows one brand. Despite no explicit filter being placed on the Product[Color] column, only the colors sold by the current brand are visible because Product[Color] is being cross-filtered by the filter on Product[Brand]. In Figure 3-26, you can see that when VALUES is evaluated for Adventure Works, it returns seven values.

Brand	Sales Amount	# Colors
A. Datum	147,687.44	10
Adventure Works	2,761,057.66	7
Contoso	2,227,244.32	15
Fabrikam	990,275.08	12
Litware	506,104.50	12
Northwind Traders	119,857.67	9
Proseware	956,335.76	7
Southridge Video	776,807.78	10
Tailspin Toys	79,159.15	11
The Phone Company	1,976,180.03	6
Wide World Importers	1,796,930.99	12
Total	**12,337,640.39**	**16**

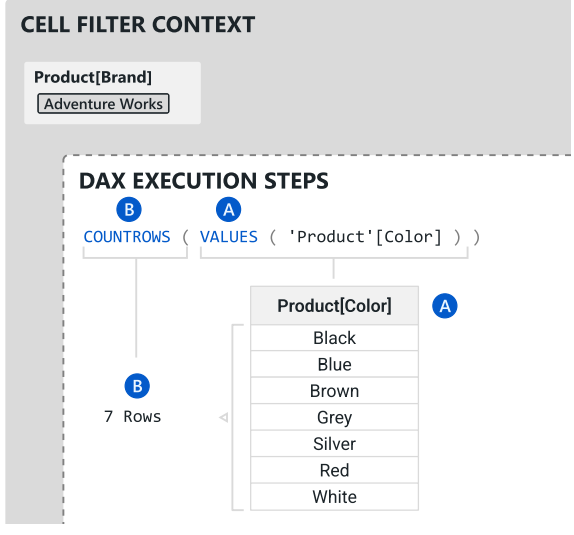

FIGURE 3-26 VALUES is evaluated for the Adventure Works brand, showing only 7 out of 16 colors.

VALUES is extremely useful in CALCULATE because it lets the developer access the values of a column that are visible in the outer filter context. In some scenarios, it is helpful to reproduce a filter that CALCULATE would replace. We now observe how it works by analyzing an alternative version of Red Colors that—instead of using KEEPFILTERS—leverages VALUES to re-create the outer filter context into the inner filter context.

When using KEEPFILTERS, this is the code of Red Sales we created:

Measure in the Sales table

```
Red Sales =
CALCULATE (
    [Sales Amount],
    KEEPFILTERS ( 'Product'[Color] = "Red" )
)
```

KEEPFILTERS aims to ensure that the outer filter over Product[Color] is maintained in the inner filter context. The same goal, which is to re-create the outer filter context in the inner filter context, can be reached by using VALUES with this alternative formulation, where IN checks that the Product[Color] values are within the list of colors visible in the current filter context:

Measure in the Sales table

```
Red Sales =
CALCULATE (
    [Sales Amount],
    'Product'[Color] = "Red",
    Product[Color] IN VALUES ( 'Product'[Color] )
)
```

As you can see in Figure 3-27, the result is the same as Red Sales using KEEPFILTERS: The sales amount value is shown only for the row containing Red.

Color	Sales Amount	Red Sales
Azure	10,743.25	
Black	3,330,387.51	
Blue	595,741.34	
Brown	885,496.17	
Gold	447,195.29	
Green	263,158.45	
Grey	907,037.74	
Orange	198,300.01	
Pink	189,533.53	
Purple	8,216.37	
Red	382,601.32	382,601.32
Silver	2,336,604.05	
Silver Grey	25,414.71	
Transparent	159.63	
White	2,723,494.66	
Yellow	33,556.32	
Total	**12,337,640.39**	**382,601.32**

FIGURE 3-27 Red Sales shows the sales amount value only when Red is already in the cell filter context.

Understanding this latter version of the code is important. CALCULATE's task is to create the inner filter context. Once the new filter context is ready, it becomes the inner filter context, under which `Sales Amount` is to be computed. However, while the new filter context is being prepared, the outer filter context remains active. As part of creating the new filter context, CALCULATE evaluates its filter arguments: In this case, CALCULATE checks that the product color needs to be within the list of colors returned by `VALUES ( Product[Color] )`. When `VALUES ( Product[Color] )` is computed, the active filter context is still the outer filter context; therefore, it contains the seven rows shown earlier.

In Figure 3-28, you can observe the formula's behavior for the Black row in Figure 3-27.

Color	Sales Amount	Red Sales
Azure	10,743.25	
① Black	3,330,387.51	
Blue	595,741.34	
Brown	885,496.17	
Gold	447,195.29	
Green	263,158.45	
Grey	907,037.74	
Orange	198,300.01	
Pink	189,533.53	
Purple	8,216.37	
Red	382,601.32	382,601.32
Silver	2,336,604.05	
Silver Grey	25,414.71	
Transparent	159.63	
White	2,723,494.66	
Yellow	33,556.32	
Total	**12,337,640.39**	**382,601.32**

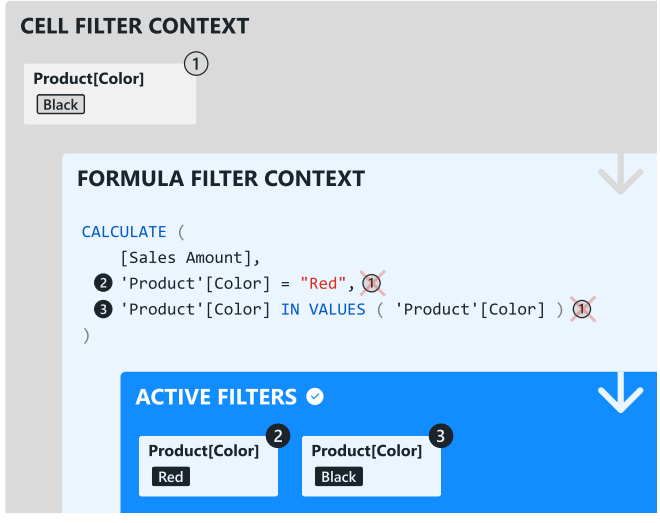

FIGURE 3-28 VALUES is evaluated when the outer filter context only filters Black.

There is a difference between using KEEPFILTERS and using VALUES. As its name implies, KEEPFILTERS keeps existing filters. VALUES creates a filter despite the filter not being there in the first place. Right now, this may seem overly complicated. When KEEPFILTERS is enough, it is also faster than VALUES. However, we will see that VALUES is helpful in more complex calculations. In the following sections, we showcase a scenario where using VALUES is important.

Introducing REMOVEFILTERS

In the examples we have seen, we used CALCULATE to add or replace a filter over a column. In many scenarios, the requirement is different: We want to remove a filter without replacing it with a new one. In these scenarios, we need REMOVEFILTERS.

A very common scenario happens when you need to compute a percentage. For example, let's pretend you want to compute the report in Figure 3-29 that shows the percentage over the total for each brand.

Brand	Sales Amount	Pct
A. Datum	147,687.44	1.20%
Adventure Works	2,761,057.66	22.38%
Contoso	2,227,244.32	18.05%
Fabrikam	990,275.08	8.03%
Litware	506,104.50	4.10%
Northwind Traders	119,857.67	0.97%
Proseware	956,335.76	7.75%
Southridge Video	776,807.78	6.30%
Tailspin Toys	79,159.15	0.64%
The Phone Company	1,976,180.03	16.02%
Wide World Importers	1,796,930.99	14.56%
Total	**12,337,640.39**	**100.00%**

FIGURE 3-29 The Pct measure computes the sales percentage of the current brand over the total.

Computing the percentage in itself is not a big deal: It is enough to divide the Sales Amount of the current cell by the Sales Amount at the grand total level and then format the result as a percentage. Computing the sales amount of the current cell is simple because the filter context makes Sales Amount compute the sales of the current brand in each cell. However, to compute the denominator, we must remove the filter on Product[Brand] to compute the total of more than 12 million.

REMOVEFILTERS does exactly this: It removes any filter from a column or a table. Here is the code of Pct:

Measure in the Sales table

```
Pct =
DIVIDE (
    [Sales Amount],
    CALCULATE (
        [Sales Amount],
        REMOVEFILTERS ( 'Product'[Brand] )
    )
)
```

In Figure 3-30, you can observe the flow of the code execution at the denominator level.

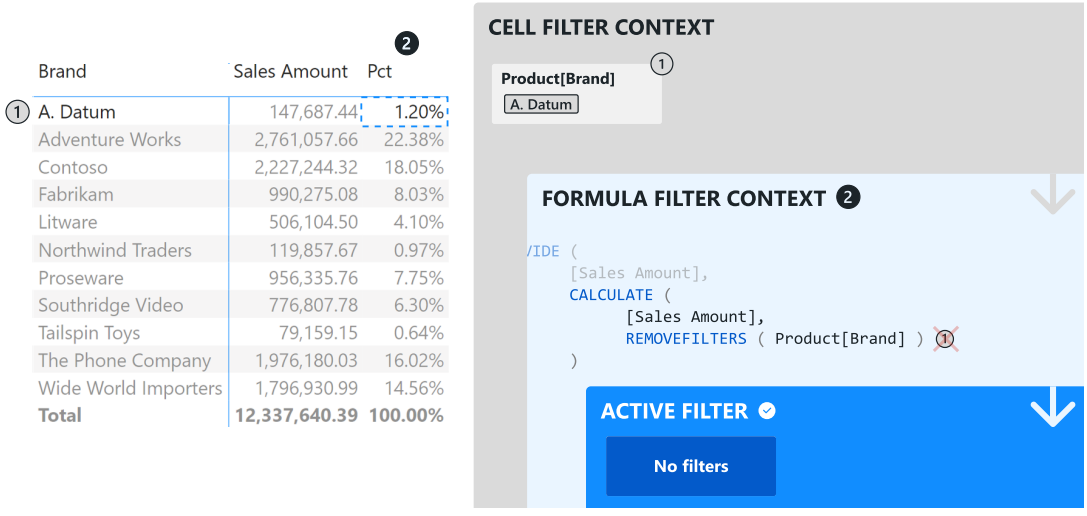

FIGURE 3-30 At the denominator level, the filter context no longer has any filter on the Brand because of `REMOVEFILTERS`.

Note that the previous formula only removes the filter from `Product[Brand]`. If the cell filter context filtered other columns, they would be untouched by `REMOVEFILTERS`. For this reason, the same formula can be used in a matrix that shows—for each product color—the percentage of one brand over all the brands for the same product color, as shown in Figure 3-31.

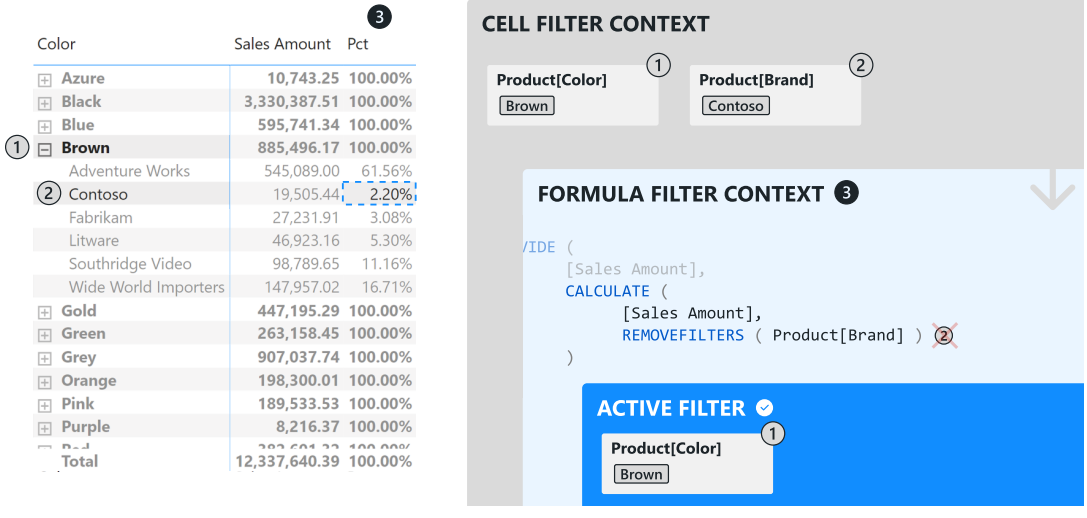

FIGURE 3-31 The filter on `Product[Color]` is not removed by `REMOVEFILTERS ( Product[Brand] )`.

REMOVEFILTERS can remove filters from one column, from multiple columns (just add them as further arguments), from an entire table, or it can also remove all the filters from the filter context. All these possibilities are handy in different scenarios.

REMOVEFILTERS is extremely helpful. However, before using it freely in your code, we suggest you wait for the end of the next chapter. REMOVEFILTERS in itself is not complex. What is complex is the manipulation of the filter context—at least when done right. In Chapter 4, "Manipulating the filter context," we provide several examples of filter context manipulation, including the use of REMOVEFILTERS.

Understanding how totals are computed in Power BI

Let's look in Figure 3-32 at the matrix we used for the last example.

Color	Sales Amount	Pct
⊞ **Azure**	**10,743.25**	**100.00%**
⊞ **Black**	**3,330,387.51**	**100.00%**
⊞ **Blue**	**595,741.34**	**100.00%**
⊟ **Brown**	**885,496.17**	**100.00%**
Adventure Works	545,089.00	61.56%
Contoso	19,505.44	2.20%
Fabrikam	27,231.91	3.08%
Litware	46,923.16	5.30%
Southridge Video	98,789.65	11.16%
Wide World Importers	147,957.02	16.71%
⊞ **Gold**	**447,195.29**	**100.00%**
⊞ **Green**	**263,158.45**	**100.00%**
⊞ **Grey**	**907,037.74**	**100.00%**
⊞ **Orange**	**198,300.01**	**100.00%**
⊞ **Pink**	**189,533.53**	**100.00%**
⊞ **Purple**	**8,216.37**	**100.00%**
Total	**12,337,640.39**	**100.00%**

FIGURE 3-32 The matrix shows the percentage of sales of the current brand over the total for each color.

It is interesting to question the meaning of each color row. It always shows 100%, so it is somewhat useless, and we might think about removing it from the report. However, as humans, we find that value reassuring. Seeing 100% gives us the nice feeling that the numbers in all the cells under it are correct. And, if we want to double-check this, we can just sum the Pct value for all the brands within a color to confirm that the total is indeed 100%.

Just look at the same report without the totals in Figure 3-33. Do you not feel an immediate need to check that the values in Pct account for 100% of Brown?

Color	Sales Amount	Pct
⊞ **Azure**	**10,743.25**	
⊞ **Black**	**3,330,387.51**	
⊞ **Blue**	**595,741.34**	
⊟ **Brown**	**885,496.17**	
Adventure Works	545,089.00	61.56%
Contoso	19,505.44	2.20%
Fabrikam	27,231.91	3.08%
Litware	46,923.16	5.30%
Southridge Video	98,789.65	11.16%
Wide World Importers	147,957.02	16.71%
⊞ **Gold**	**447,195.29**	
⊞ **Green**	**263,158.45**	
⊞ **Grey**	**907,037.74**	
⊞ **Orange**	**198,300.01**	
⊞ **Pink**	**189,533.53**	
⊞ **Purple**	**8,216.37**	
Total	**12,337,640.39**	

FIGURE 3-33 Removing the subtotals hides the 100% values and reduces our trust in the numbers.

When the total is computed by summing individual rows, its presence helps us trust the report. Nonetheless, this only applies to additive calculations. An additive calculation is indeed a calculation where the totals are computed by summing the rows. Most of the calculations are additive, and over time, we might get used to checking that a measure is correct by adding rows and checking the result. Despite all this, many interesting calculations, such as distinct counts and margin percentages, do not have the additive property.

For example, in Figure 3-34, you can easily see that `Sales Amount` is additive, whereas both `Margin %` and `# Customers` are not. `# Customers` is the distinct count of `Sales[CustomerKey]`.

Continent	Sales Amount	Margin %	# Customers
⊟ **Australia**	**841,828.65**	**55.45%**	**401**
Audio	16,877.99	55.38%	41
TV and Video	87,620.17	55.62%	67
Computers	358,969.18	55.99%	185
Cameras and camcorders	55,526.76	56.06%	41
Cell phones	139,611.95	54.36%	173
Music, Movies and Audio Books	66,871.79	58.84%	148
Games and Toys	3,755.99	49.82%	74
Home Appliances	112,594.82	52.81%	65
⊞ **Europe**	**3,565,296.05**	**55.68%**	**1,658**
⊞ **North America**	**7,930,515.69**	**56.04%**	**3,526**
Total	**12,337,640.39**	**55.90%**	**5,585**

FIGURE 3-34 Out of three measures, two are non-additive. Only `Sales Amount` shows the sum of rows in the subtotals.

Non-additivity is a property of a formula. Forcing a non-additive measure to be additive is—usually—a mistake. Because DAX does not know whether a calculation is additive or not (to be honest, it knows to some extent, but this would be out of the scope of this section), it chooses to stay on the safe side. It computes subtotals by computing the same formula again and again in different filter contexts.

This behavior guarantees that the additive or non-additive totals are always computed correctly. Unfortunately, newbies may conclude that their measure is wrong just because the total is not the sum of individual rows. That would be an incorrect conclusion, at least for non-additive calculations. For additive measures, the total must be the sum of the rows. For non-additive measures, it is correct to show a total that is not the sum of individual rows.

There are techniques to force additivity for non-additive calculations, but we cannot introduce them now because they mainly rely on the concept of context transition—which we will see later. In this introductory chapter, it is useful to showcase a couple of scenarios where developers would expect additivity, whereas DAX is correct in performing a non-additive calculation.

One simple example is rounding. If we want a measure that computes the sales amount rounded to the nearest million, we can author a measure like the following:

Measure in the Sales table

```
Rounded Sales = MROUND ( [Sales Amount], 1000000 )
```

Once projected in a matrix, the result is non-additive because the rounding happens on the value of `Sales Amount` computed at the total level and not by summing the value rounded for each Continent, as shown in Figure 3-35.

Continent	Sales Amount	Rounded Sales
⊞ Australia	841,828.65	1,000,000.00
⊞ Europe	3,565,296.05	4,000,000.00
⊞ North America	7,930,515.69	8,000,000.00
Total	**12,337,640.39**	**12,000,000.00**

FIGURE 3-35 Rounded `Sales` is non-additive because the total is rounded separately as opposed to it being the sum of the rounded values.

The sum of the rounded values would be 13,000,000. However, because `Sales Amount` evaluates to 12,337,640.39, its rounded value is 12,000,000, not 13,000,000. Despite being somewhat exaggerated, this example shows how the calculations happen in a matrix visual. DAX computes the total by executing the formula of the measure in a different filter context, where some (if not all) filters are missing.

This behavior is desirable and needed. Indeed, if Power BI were to compute the total by summing individual values, not only would it not work for non-additive calculations, but it would also create a situation where the result may differ depending on the number of rows in the matrix.

Let's see another example where the formula's behavior looks somewhat counterintuitive. Beware, the example would actually make little sense in a business context, but it does show you how DAX works.

We write a calculation that computes the sales amount and applies a discount of 5% if the sales amount is larger than one million. We can tell that it will not make sense to try to add together the values we find, as they are sometimes discounted and sometimes not.

The simplest way to demonstrate the behavior of this formula is to use two visual calculations. We will discuss visual calculations in detail later in the book. However, here is the code of the first, which just computes the discounted sales:

Visual calculation

```
Discounted Sales =
IF (
    [Sales Amount] >= 1000000,
    [Sales Amount] * 0.95,
    [Sales Amount]
)
```

The second visual calculation forces additivity over `Discounted Sales`:

Visual calculation

```
Additive Calc = EXPAND ( SUM ( [Discounted Sales] ), ROWS )
```

This formula sums the values of `Discounted Sales` for all the rows of the visual when the hierarchy of Country and Category attributes is expanded. At the total level, `Additive Calc` sums the rows of the visual for the column of the model you may be slicing by—in our example below, either categories or countries. While the value is the same as `Discounted Sales` for each row of the visual, the total is computed by summing individual rows in the total. In Figure 3-36, you can see two matrixes: one is sliced by `Product[Category]`, and the other by `Customer[Country]`. The total of `Discounted Sales` is the same for both matrixes. However, the total of `Additive Calc` is different in the two matrixes because the individual rows have different values. Indeed, some will hit the discount threshold, and some will not.

When changing the contents of the visual, the very same formula ends up giving different results.

Category	Sales Amount	Discounted Sales	Additive Calc
Audio	238,356.06	238,356.06	238,356.06
TV and Video	1,554,379.92	1,476,660.92	1,476,660.92
Computers	4,773,083.99	4,534,429.79	4,534,429.79
Cameras and camcorders	702,883.41	702,883.41	702,883.41
Cell phones	2,239,863.67	2,127,870.49	2,127,870.49
Music, Movies and Audio Books	844,851.77	844,851.77	844,851.77
Games and Toys	88,449.68	88,449.68	88,449.68
Home Appliances	1,895,771.88	1,800,983.28	1,800,983.28
Total	**12,337,640.39**	**11,720,758.37**	**11,814,485.41**

Country	Sales Amount	Discounted Sales	Additive Calc
Australia	841,828.65	841,828.65	841,828.65
Canada	1,432,719.68	1,361,083.69	1,361,083.69
France	409,131.89	409,131.89	409,131.89
Germany	1,281,186.84	1,217,127.50	1,217,127.50
Italy	249,829.30	249,829.30	249,829.30
Netherlands	649,742.90	649,742.90	649,742.90
United Kingdom	975,405.11	975,405.11	975,405.11
United States	6,497,796.01	6,172,906.21	6,172,906.21
Total	**12,337,640.39**	**11,720,758.37**	**11,877,055.26**

FIGURE 3-36 Forcing additivity provides different results, depending on the model column used to slice the values.

At the total level for `Discounted Sales`, we notice that the value obtained is 95% of the total for `Sales Amount`, simply because the total of `Sales Amount` is far above one million, and therefore, the formula applies the 5% discount to the total.

At the total level for `Additive Calc`, the value obtained is the actual sum of all the `Additive Calc` values for each row of the matrix (each `Category` or each `Country`). So, this total does indeed represent our overall revenue after the selective discount, depending on what we sliced by.

The `Discounted Sales` measure we authored is non-additive because it applies the discount if `Sales Amount` exceeds one million. The non-additive behavior can happen for several rows in the matrix; hence, the discount is applied to individual rows exceeding one million. However, individual values are not summed up in the total. The `Discounted Sales` measure is computed again, and because the total of `Sales Amount` largely exceeds one million, the discount is applied to the entire value of `Sales Amount`.

Despite not being intuitive, it is the correct behavior. Developers always have the option of creating additive calculations, as we did in this demo through a visual calculation, but they need to do so explicitly.

Conclusions

In this chapter, we briefly introduced the concept of the filter context and CALCULATE.

Each cell in a matrix has its unique filter context, the filter context your measure starts with. We sometimes call the cell filter context the *original filter context*, the first context in which the measure is being evaluated. Whenever a measure should direct the calculation to another area of the model, developers can rely on CALCULATE to change the filter context.

CALCULATE can add or remove filters. When CALCULATE creates a filter, it overrides any existing filter in the same column. Developers can change this behavior by using KEEPFILTERS or IN VALUES. REMOVEFILTERS removes filters from the filter context and can remove filters from columns, tables, or the entire data model.

There are still a lot of details that we need to cover before we can master the filter context and CALCULATE. Before diving into more complex details in the next chapter, we manipulate the filter context by creating some calculations, leaving the more intricate technical details for later in the book.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y