

# 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 |



# **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.

Copyright © 2026 by Pearson Education, Inc.

Hoboken, New Jersey

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms, and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit [www.pearson.com/global-permission-granting.html](http://www.pearson.com/global-permission-granting.html).

No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

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

ISBN-10: 0-13-824472-3

Library of Congress Control Number: On file

\$PrintCode

**Trademarks**

Microsoft and the trademarks listed at <http://www.microsoft.com> on the "Trademarks" webpage are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

Figures 02.09, 03.16, 11.01–11.13, 11.16–11.47, 12.01–12.27, 12.31, 12.36–12.38, 14.06, 17.01–17.12, 17.14–17.16, 17.18–17.20, 18.02, 18.04, 18.09, 18.11, 18.14, 18.28, 18.29 © SQLBI

**Warning and Disclaimer**

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author, the publisher, and Microsoft Corporation shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the programs accompanying it.

**Special Sales**

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at [corpsales@pearsoned.com](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the U.S., please contact [intlcs@pearson.com](mailto:intlcs@pearson.com).

**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

	<i>Foreword</i>	<i>xi</i>
	<i>Acknowledgments</i>	<i>xiv</i>
	<i>Introduction</i>	<i>xvi</i>
CHAPTER 1	Introduction to learning DAX	1
CHAPTER 2	Introducing DAX	25
CHAPTER 3	Introducing the filter context and CALCULATE	81
CHAPTER 4	Manipulating the filter context	113
CHAPTER 5	Introducing the row context and the context transition	157
CHAPTER 6	Understanding basic table functions	195
CHAPTER 7	Understanding variables	237
CHAPTER 8	Understanding the evaluation context	255
CHAPTER 9	Working with the evaluation context	313
CHAPTER 10	Understanding user-defined functions in DAX	363
CHAPTER 11	Working with tables	387
CHAPTER 12	Understanding window functions in DAX	453
CHAPTER 13	Time intelligence calculations	509
CHAPTER 14	Understanding visual calculations	573
CHAPTER 15	Understanding calculation groups	627
CHAPTER 16	Inspecting the filter context and managing hierarchies	651
CHAPTER 17	Authoring queries	689
CHAPTER 18	Advanced DAX concepts	715
CHAPTER 19	Practicing DAX with advanced examples	761
	<i>Index</i>	<i>802</i>



# Contents

<i>Foreword</i> .....	<i>xi</i>
<i>Acknowledgments</i> .....	<i>xiv</i>
<i>Introduction</i> .....	<i>xvi</i>
<b>Chapter 1 Introduction to learning DAX</b>	<b>1</b>
Why is DAX even a thing? .....	2
Understanding how the data model affects your DAX code .....	5
Introducing the demo data model .....	9
Why DAX is different from any other language .....	10
Conclusions .....	24
<b>Chapter 2 Introducing DAX</b>	<b>25</b>
Understanding DAX calculations .....	25
Understanding calculated columns and measures .....	34
Introducing variables .....	44
Formatting DAX code .....	45
Using common DAX functions .....	48
Introducing visual calculations .....	64
Introducing user-defined functions .....	68
Handling errors in DAX expressions .....	71
Conclusions .....	80
<b>Chapter 3 Introducing the filter context and CALCULATE</b>	<b>81</b>
Introducing axis and coordinates .....	81
Introducing the filter context .....	84
Introducing CALCULATE .....	91
Introducing KEEPFILTERS .....	94
VALUES as an alternative to KEEPFILTERS .....	101
Introducing REMOVEFILTERS .....	104
Understanding how totals are computed in Power BI .....	106
Conclusions .....	111

<b>Chapter 4</b>	<b>Manipulating the filter context</b>	<b>113</b>
	Country as a percentage of the continent . . . . .	114
	Percentage over a selected brand . . . . .	118
	Highlighting the best brands . . . . .	124
	Choosing how to remove filters . . . . .	135
	Computing percentage and visually representing growth compared to the previous year . . . . .	142
	Computing returning customers . . . . .	151
	Conclusions . . . . .	156
<b>Chapter 5</b>	<b>Introducing the row context and     the context transition</b>	<b>157</b>
	Introducing the row context . . . . .	158
	Introducing the context transition . . . . .	166
	Using the row context with relationships . . . . .	173
	Understanding nested row contexts . . . . .	187
	Understanding the differences between FILTER and CALCULATE . . . . .	193
	Conclusions . . . . .	194
<b>Chapter 6</b>	<b>Understanding basic table functions</b>	<b>195</b>
	Introducing table functions . . . . .	195
	Introducing the EVALUATE syntax . . . . .	197
	Understanding FILTER . . . . .	199
	Understanding ALL and ALLEXCEPT . . . . .	202
	Understanding VALUES, DISTINCT, and the blank row . . . . .	207
	Understanding SELECTCOLUMNS and ADDCOLUMNS . . . . .	215
	Understanding SUMMARIZE . . . . .	220
	Introducing SUMMARIZECOLUMNS . . . . .	224
	Computing the best store by product . . . . .	226
	Using tables as scalar values . . . . .	229
	Introducing ALLSELECTED . . . . .	232
	Conclusions . . . . .	236

<b>Chapter 7</b>	<b>Understanding variables</b>	<b>237</b>
	Introducing the VAR syntax . . . . .	237
	The importance of variable names . . . . .	239
	Understanding that variables are constant . . . . .	241
	Understanding the scope of variables . . . . .	243
	Using table variables . . . . .	247
	Understanding when variables are evaluated . . . . .	249
	Common patterns using variables . . . . .	251
	Conclusions . . . . .	254
<b>Chapter 8</b>	<b>Understanding the evaluation context</b>	<b>255</b>
	CALCULATE filters are tables . . . . .	255
	Introducing expanded tables . . . . .	261
	Introducing data lineage and TREATAS . . . . .	265
	Understanding the context transition . . . . .	269
	Using the context transition with iterators . . . . .	285
	Introducing CALCULATE modifiers . . . . .	294
	Introducing the ALL* family of functions . . . . .	300
	Understanding the evaluation order of CALCULATE . . . . .	305
	Understanding circular dependencies . . . . .	307
	Conclusions . . . . .	312
<b>Chapter 9</b>	<b>Working with the evaluation context</b>	<b>313</b>
	Using different ways to express a filter . . . . .	313
	Using CALCULATE in different contexts . . . . .	317
	Filtering columns versus filtering tables . . . . .	324
	Understanding the active relationship . . . . .	328
	Understanding KEEPFILTERS with iterators . . . . .	330
	Managing selections using SELECTEDVALUE and VALUES . . . . .	337
	Computing end-of-year gifts for top customers . . . . .	342
	Using many-to-many relationships . . . . .	351
	Conclusions . . . . .	362

<b>Chapter 10</b>	<b>Understanding user-defined functions in DAX</b>	<b>363</b>
	Introducing functions.....	363
	Understanding parameter-passing modes.....	367
	Understanding dynamic binding of columns.....	374
	Model-dependent and model-independent functions.....	376
	Naming conventions for user-defined functions.....	377
	Examples of using functions.....	380
	Conclusions.....	386
<b>Chapter 11</b>	<b>Working with tables</b>	<b>387</b>
	Understanding functions to group and filter.....	387
	Reading Power BI queries.....	405
	Understanding set manipulation functions.....	415
	Understanding functions to extract rows and join tables.....	427
	Using tables as filters.....	440
	Creating calculated tables.....	447
	Conclusions.....	452
<b>Chapter 12</b>	<b>Understanding window functions in DAX</b>	<b>453</b>
	Understanding INDEX.....	454
	Introducing OFFSET.....	464
	Introducing <i>apply semantics</i> .....	466
	Understanding OFFSET.....	476
	Understanding WINDOW.....	482
	Understanding WINDOW and <i>apply semantics</i> .....	487
	Understanding <i>apply semantics</i> .....	494
	Understanding RANK and ROWNUMBER.....	497
	Common window function errors.....	499
	Conclusions.....	508
<b>Chapter 13</b>	<b>Time intelligence calculations</b>	<b>509</b>
	Introducing time intelligence.....	509
	Building a date table.....	522

Using calendars .....	533
Understanding the behavior of time intelligence functions .....	537
Understanding time intelligence functions.....	542
Conclusions .....	570
<b>Chapter 14 Understanding visual calculations</b>	<b>573</b>
Introducing visual calculations .....	573
Understanding the visual shape .....	577
Understanding the visual context .....	587
Understanding visual calculation functions .....	605
Visual calculation examples .....	615
Conclusions .....	626
<b>Chapter 15 Understanding calculation groups</b>	<b>627</b>
Introducing calculation groups.....	628
Understanding calculation group precedence.....	635
Using ISSELECTEDMEASURE and SELECTEDMEASURENAME.....	639
Intercepting multiple selections and no selection .....	641
Activating calculation items in DAX.....	642
Using calculation groups to apply global filters.....	645
Visual calculations and calculation groups .....	649
Conclusions .....	650
<b>Chapter 16 Inspecting the filter context and managing hierarchies</b>	<b>651</b>
Using HASONEVALUE and ISINSCOPE.....	652
Introducing ISFILTERED and ISCROSSFILTERED .....	655
Understanding the differences between VALUES and FILTERS .....	658
Using ISEMPTY .....	660
Understanding arbitrarily shaped filters.....	662
Computing percentages over hierarchies .....	669
Handling parent/child hierarchies .....	674
Conclusions .....	688

<b>Chapter 17</b>	<b>Authoring queries</b>	<b>689</b>
	Understanding EVALUATE.....	689
	Implementing common DAX query patterns.....	699
	Conclusions.....	713
<b>Chapter 18</b>	<b>Advanced DAX concepts</b>	<b>715</b>
	Understanding expanded tables.....	715
	Understanding ALLSELECTED and shadow filter contexts.....	729
	Understanding bidirectional relationships and ambiguity.....	738
	Understanding auto-exists and non-empty.....	750
	Conclusions.....	760
<b>Chapter 19</b>	<b>Practicing DAX with advanced examples</b>	<b>761</b>
	Working with a budget and different granularities.....	761
	Working with semi-additive calculations.....	775
	Computing same-store sales.....	787
	Conclusions.....	801
	<i>Index</i>	802

# Foreword

## Foreword to the 3<sup>rd</sup> 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 1<sup>st</sup> and 2<sup>nd</sup> 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](http://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](https://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/](http://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](http://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](http://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](http://support.microsoft.com)*

# 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.

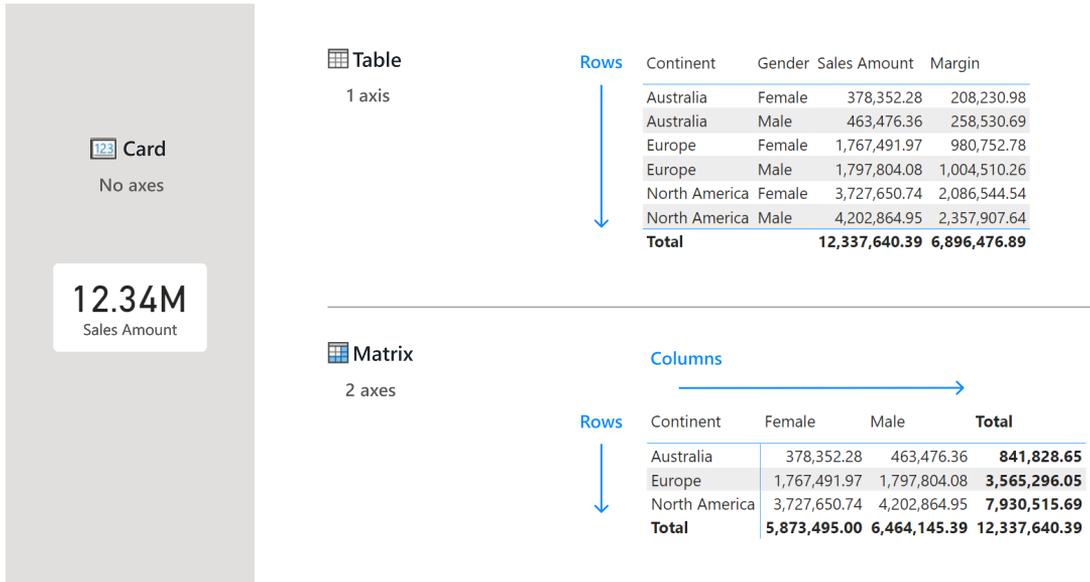
We want to give a few words of warning to our readers. The concept of evaluation context is simple, and you will learn and understand it soon. Nevertheless, you need to thoroughly understand several subtle considerations and details before you really master DAX. Otherwise, you will feel lost at a certain 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.

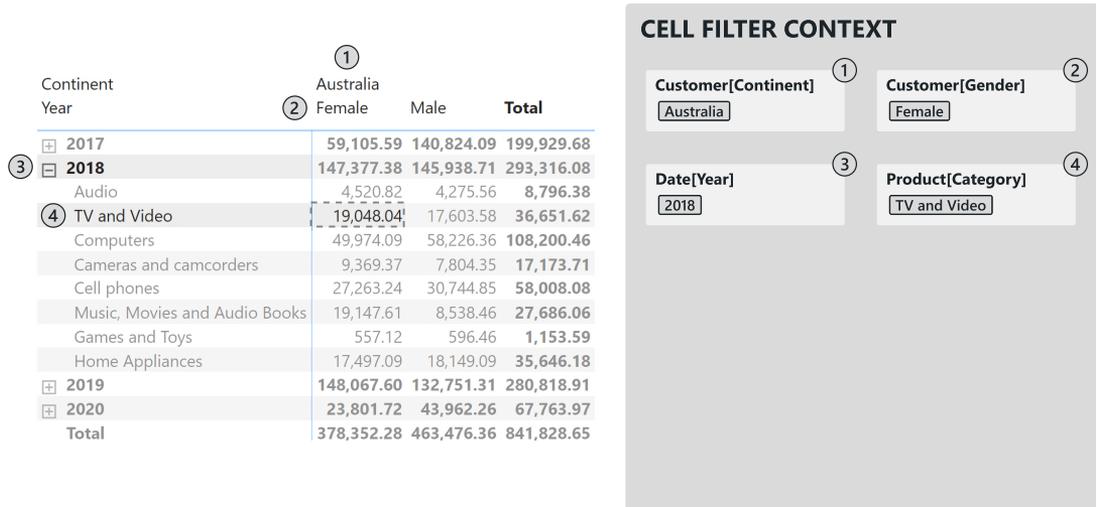


FIGURE 3-2 All the coordinates of a cell contribute to creating the filter context.

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.

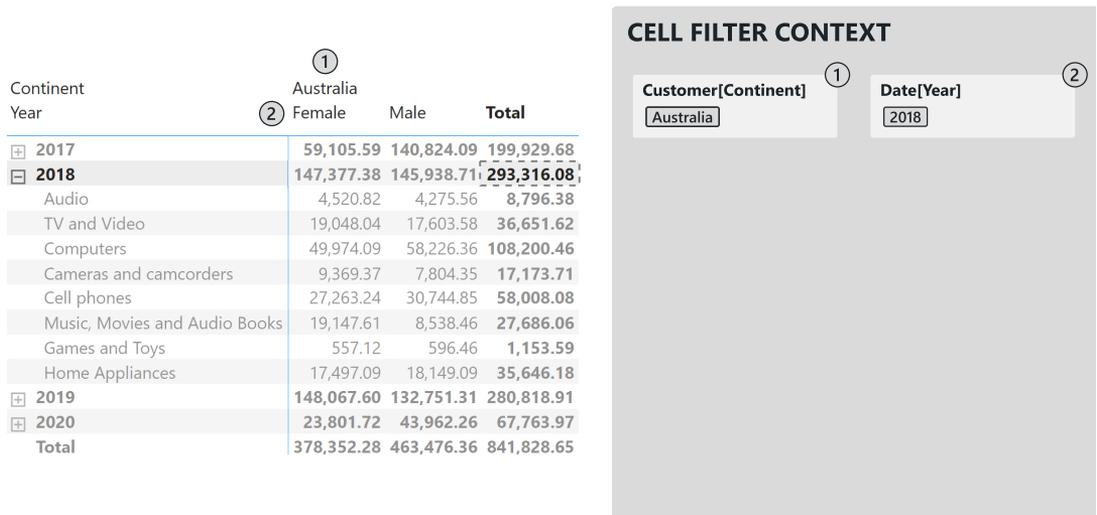


FIGURE 3-3 Coordinates of subtotal cells do not filter all the columns of the axes.

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
<b>Total</b>	<b>12,337,640.39</b>

**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
<b>Total</b>	<b>12,337,640.39</b>

### 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
<b>Total</b>	<b>12,337,640.39</b>

### CELL FILTER CONTEXT

①

Product[Brand]  
Tailspin Toys

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

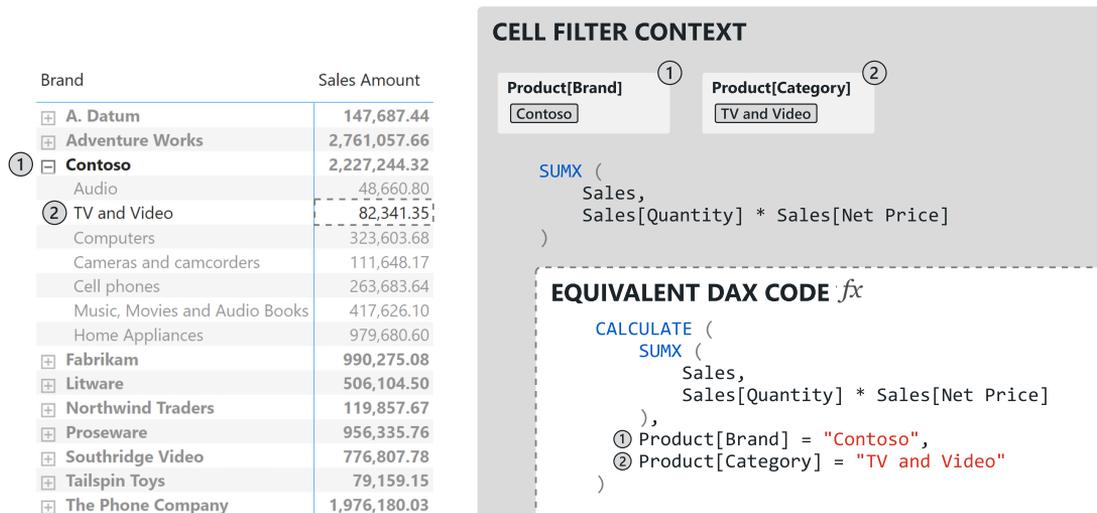
**EQUIVALENT DAX CODE** *fx*

```
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.

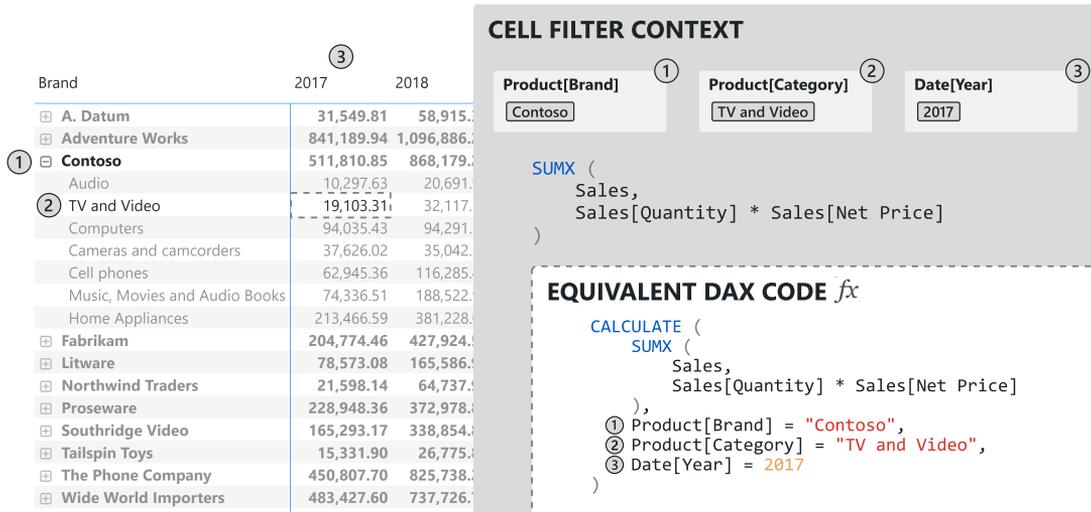
Before we move forward, let's say a few words about CALCULATE. CALCULATE accepts many arguments. The first is the expression to compute. Starting from the second argument, CALCULATE receives the filters to create or modify the filter context. The filters are applied before the first argument can be computed; 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.

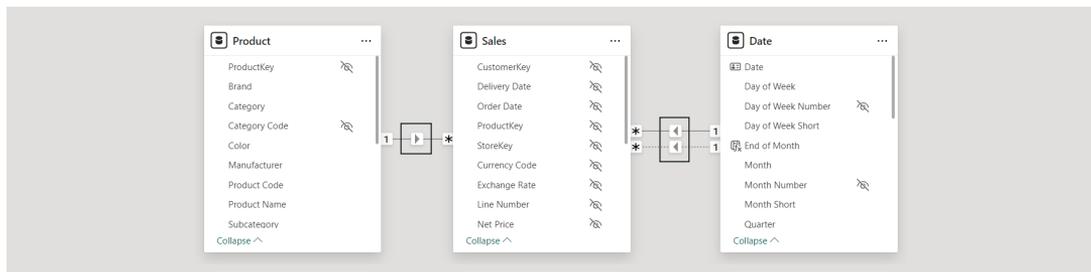
CALCULATE accepts any number of arguments as filters. They are all intersected, meaning that all the filters work together at once. Indeed, if we further modify the matrix by adding the years on the columns, that operation changes the filter but not the mechanism used to create the filter itself. In 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.

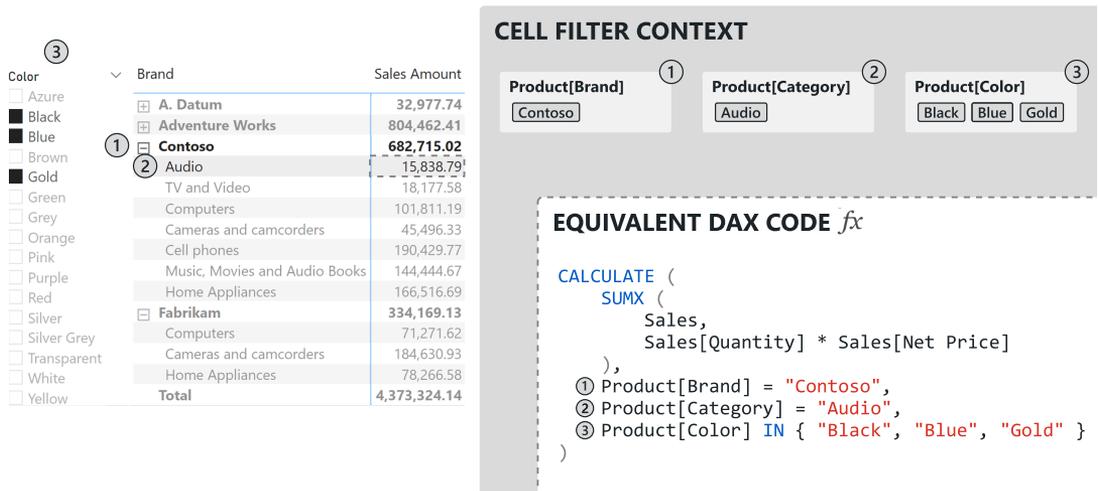


**FIGURE 3-10** Relationships define how the filter context propagates the filters through the cross-filter direction.

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,

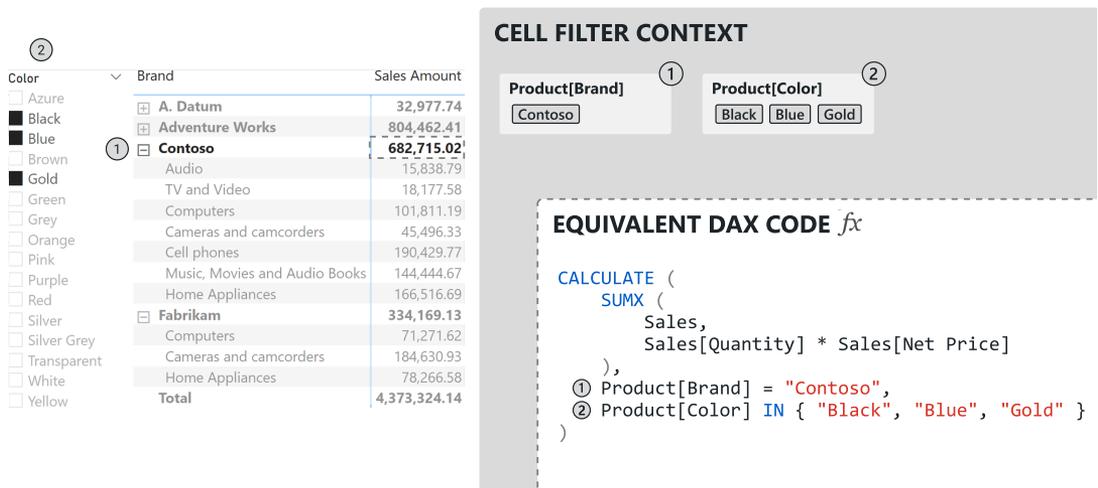
for instance, do this. Whenever you select one or more items in a slicer, the selection generates a filter added to the filter context, as shown in Figure 3-11.



**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.

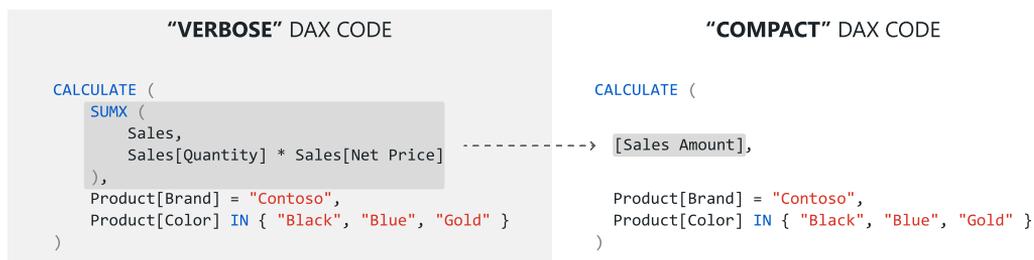


**FIGURE 3-12** At the subtotal level, the more detailed columns do not contribute to the cell filter context.

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.

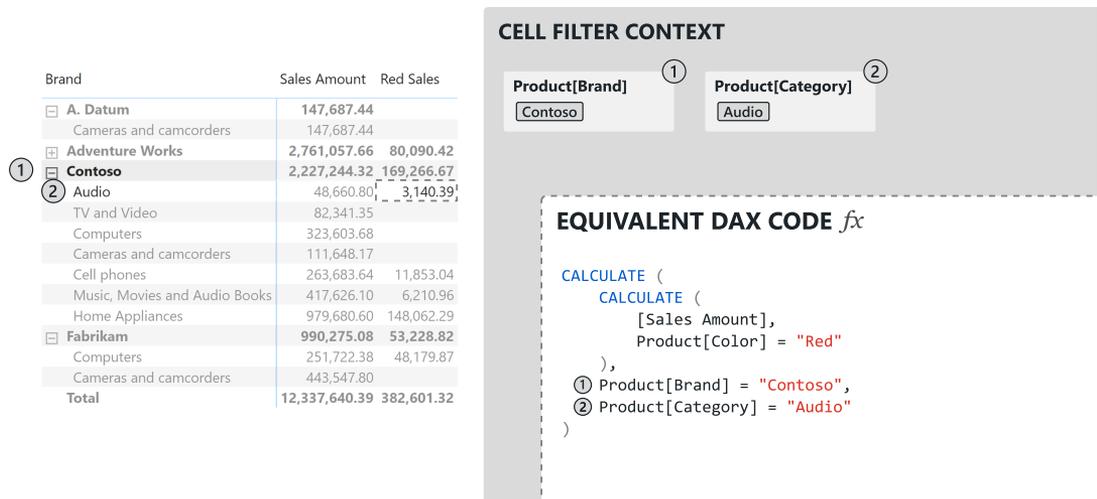


FIGURE 3-14 Because the measure uses CALCULATE, we have two nested CALCULATE functions.

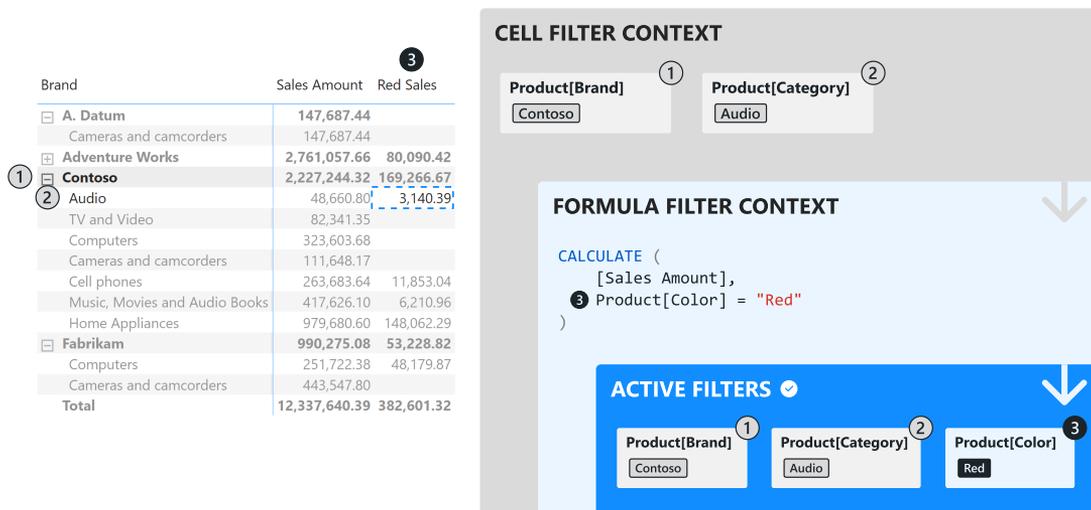
In the figure, CALCULATE shows the cell filter context applied to the evaluation of Red Sales. However, Red Sales contains another CALCULATE. Therefore, the code being executed contains two nested CALCULATE functions. Here is the code with Red Sales expanded:

```

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

```

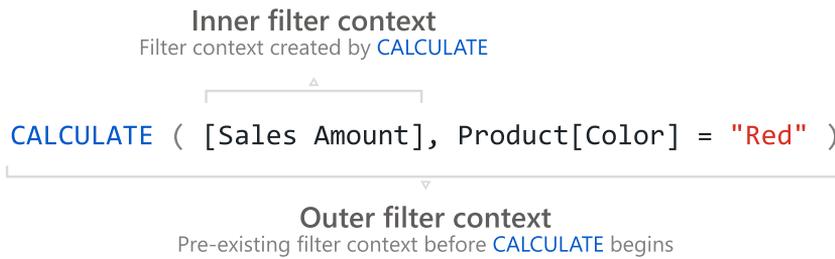
The outer CALCULATE defines the filter context of the cell; the expanded code of Red Sales introduces the inner CALCULATE. In this scenario, the net effect of the two CALCULATE functions is that all the filters are active simultaneously when Sales Amount is being executed. Therefore, the cell shows the 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.

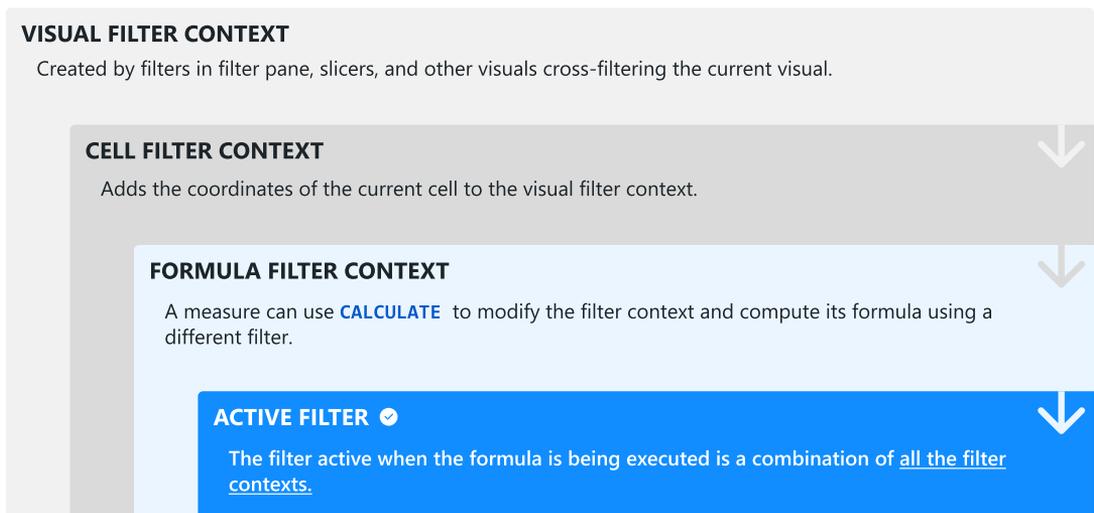
When dealing with CALCULATE, we often refer to two filter contexts: the outer filter context and the inner filter context, as shown in Figure 3-16.



**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

---

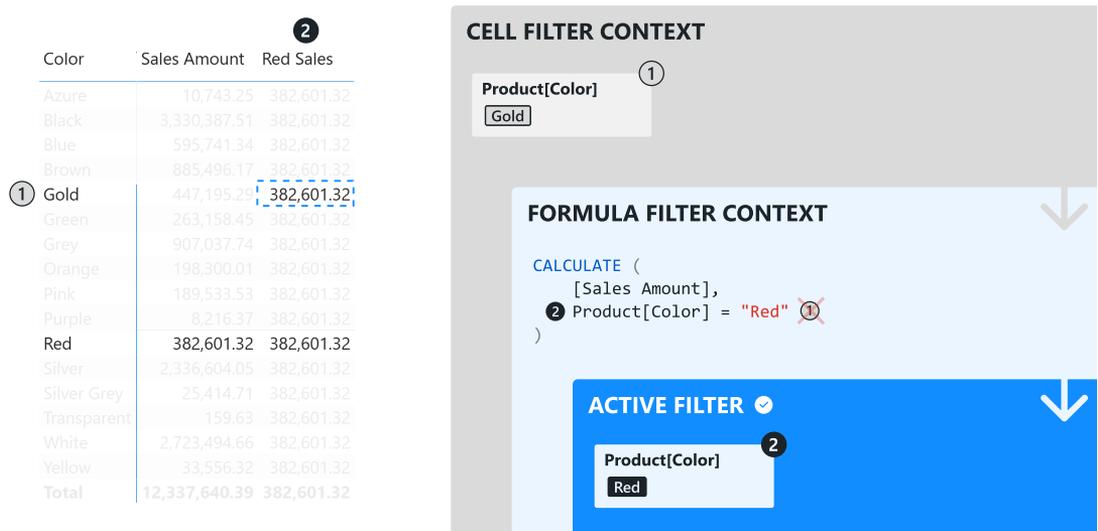
It is interesting to observe what happens when there is a conflict between the cell filter context and the CALCULATE filter context. If we use the `Product[Color]` column on the rows, rather than brand and category, the result shows the same value (sales of red products) on all the rows, as shown in 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
<b>Total</b>	<b>12,337,640.39</b>	<b>382,601.32</b>

**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.

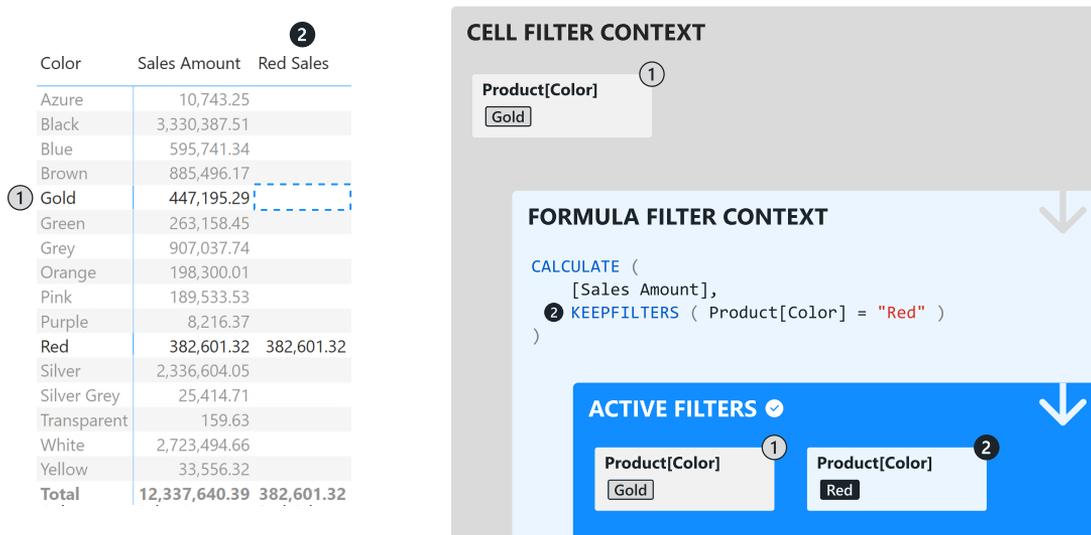
This behavior can be changed by using `KEEPFILTERS`. `KEEPFILTERS` is a `CALCULATE` modifier that changes how `CALCULATE` uses the new filters to generate the inner filter context. When used, `KEEPFILTERS` instructs `CALCULATE` not to remove the outer filter but to maintain it together with the new filter.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

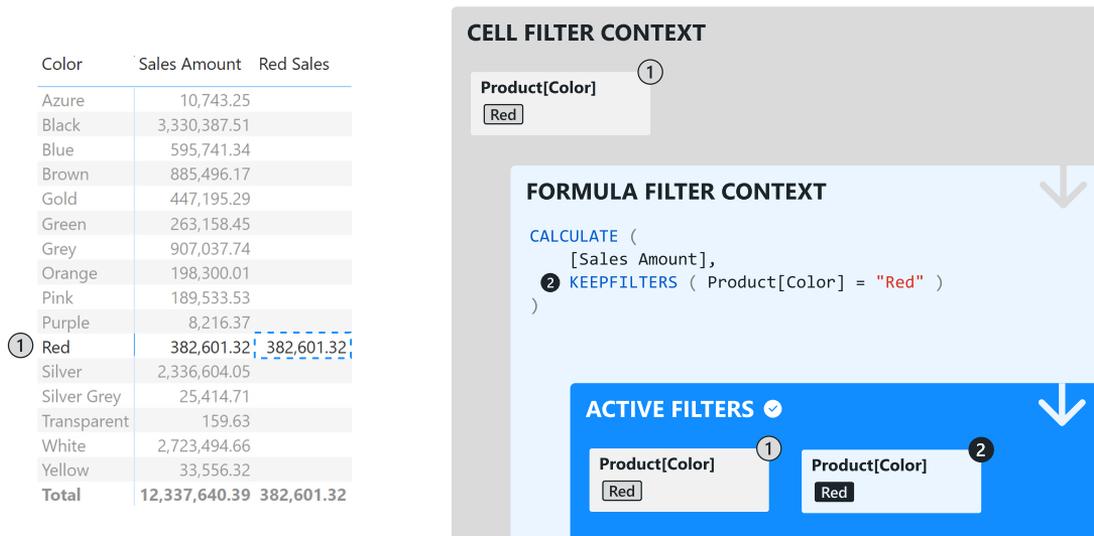
KEEPFILTERS keeps the outer filter context inside the inner filter context. In other words, the code being executed is equivalent to this:

```

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.



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

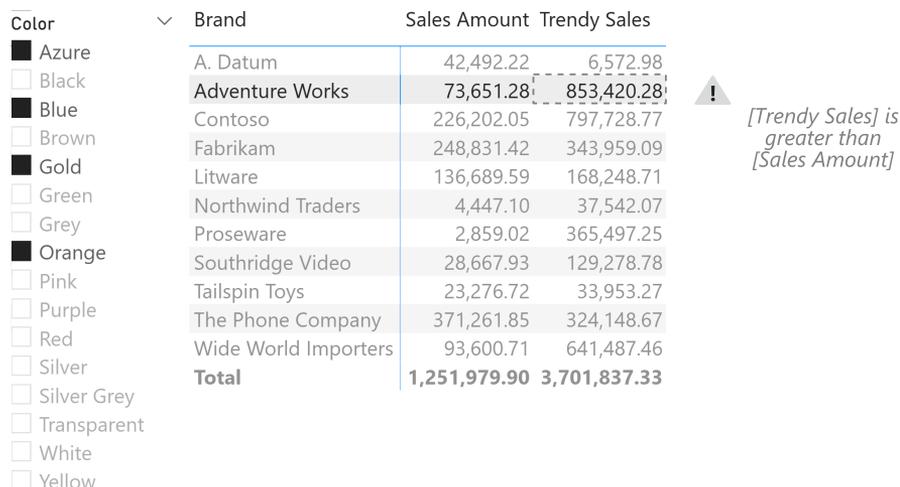
Be mindful that both versions of Red Sales (the one using KEEPFILTERS and the one without KEEPFILTERS) are correct. It all depends on the calculation you want to obtain. In some scenarios, you want to remove the outer filter and replace it with a new one. In other scenarios, you want the opposite behavior.

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`).

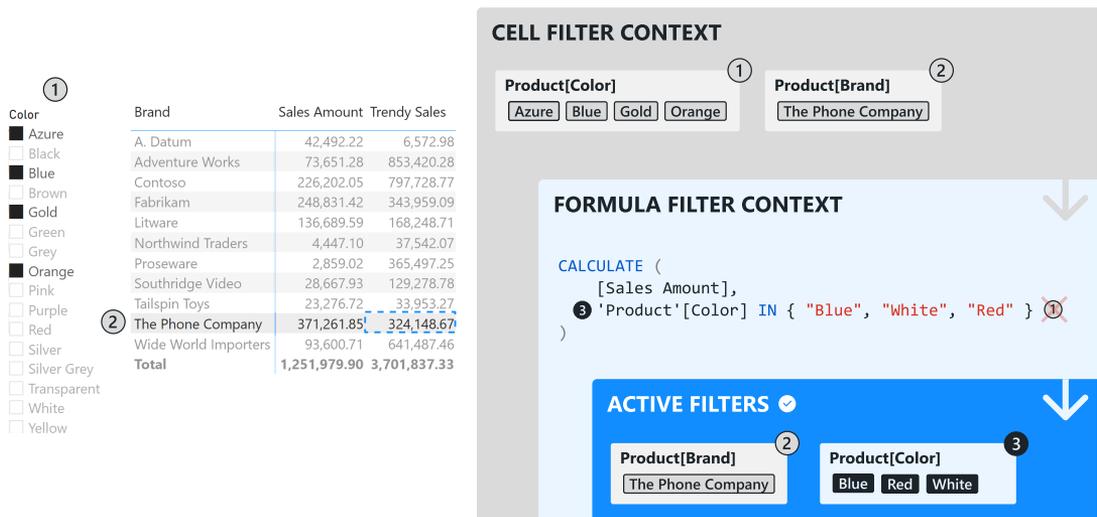


**FIGURE 3-22** The filter replacement produced by `CALCULATE` makes numbers hard to understand.

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

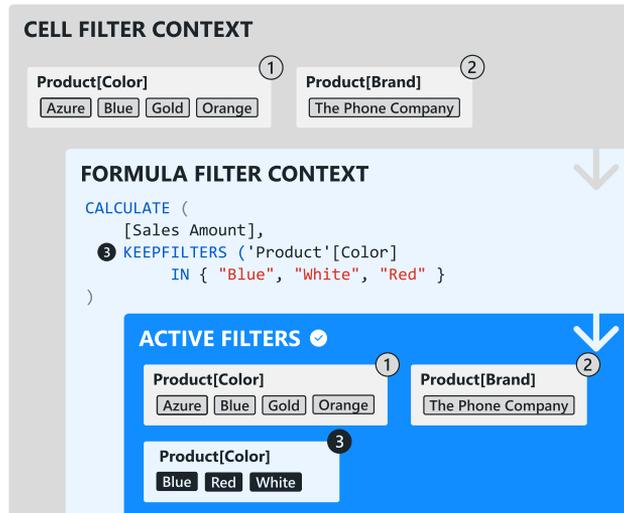
```
Trendy Sales =
CALCULATE (
    [Sales Amount],
    KEEPFILTERS ( 'Product'[Color] IN { "Blue", "White", "Red" } )
)
```

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.

①

Color	Brand	Sales Amount	Trendy Sales
<input type="checkbox"/> Azure	A. Datum	42,492.22	6,572.98
<input type="checkbox"/> Black	Adventure Works	73,651.28	73,651.28
<input type="checkbox"/> Blue	Contoso	226,202.05	177,371.91
<input type="checkbox"/> Brown	Fabrikam	248,831.42	111,322.40
<input type="checkbox"/> Gold	Litware	136,689.59	99,526.87
<input type="checkbox"/> Green	Northwind Traders	4,447.10	4,447.10
<input type="checkbox"/> Grey	Proseware	2,859.02	2,859.02
<input type="checkbox"/> Orange	Southridge Video	28,667.93	6,770.07
<input type="checkbox"/> Pink	Tailspin Toys	23,276.72	22,882.84
<input type="checkbox"/> Purple	<b>The Phone Company</b>	<b>371,261.85</b>	
<input type="checkbox"/> Red	Wide World Importers	93,600.71	90,336.87
<input type="checkbox"/> Silver	<b>Total</b>	<b>1,251,979.90</b>	<b>595,741.34</b>
<input type="checkbox"/> Silver Grey			
<input type="checkbox"/> Transparent			
<input type="checkbox"/> White			
<input type="checkbox"/> Yellow			

②



**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.

Color	Brand	Sales Amount	Trendy Sales
<input type="checkbox"/> Azure	A. Datum	42,492.22	6,572.98
<input type="checkbox"/> Black	Adventure Works	73,651.28	73,651.28
<input type="checkbox"/> Blue	Contoso	226,202.05	177,371.91
<input type="checkbox"/> Brown	Fabrikam	248,831.42	111,322.40
<input type="checkbox"/> Gold	Litware	136,689.59	99,526.87
<input type="checkbox"/> Green	Northwind Traders	4,447.10	4,447.10
<input type="checkbox"/> Grey	Proseware	2,859.02	2,859.02
<input type="checkbox"/> Orange	Southridge Video	28,667.93	6,770.07
<input type="checkbox"/> Pink	Tailspin Toys	23,276.72	22,882.84
<input type="checkbox"/> Purple	<b>The Phone Company</b>	<b>371,261.85</b>	
<input type="checkbox"/> Red	Wide World Importers	93,600.71	90,336.87
<input type="checkbox"/> Silver	<b>Total</b>	<b>1,251,979.90</b>	<b>595,741.34</b>
<input type="checkbox"/> Silver Grey			
<input type="checkbox"/> Transparent			
<input type="checkbox"/> White			
<input type="checkbox"/> Yellow			

**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

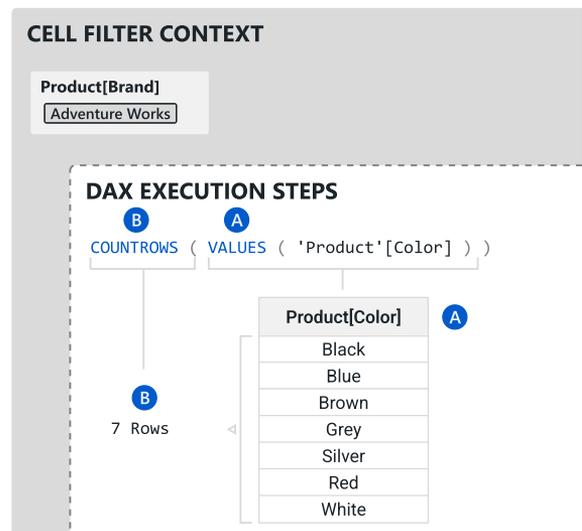
We introduced VALUES in the previous chapter. However, let's recap briefly what we know about the function before discussing how VALUES can be used as an alternative to KEEPFILTERS. The VALUES table function returns the values of a column, as currently visible in the filter context. For example, if you 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
<b>Total</b>	<b>12,337,640.39</b>	<b>16</b>



**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] )  
)
```

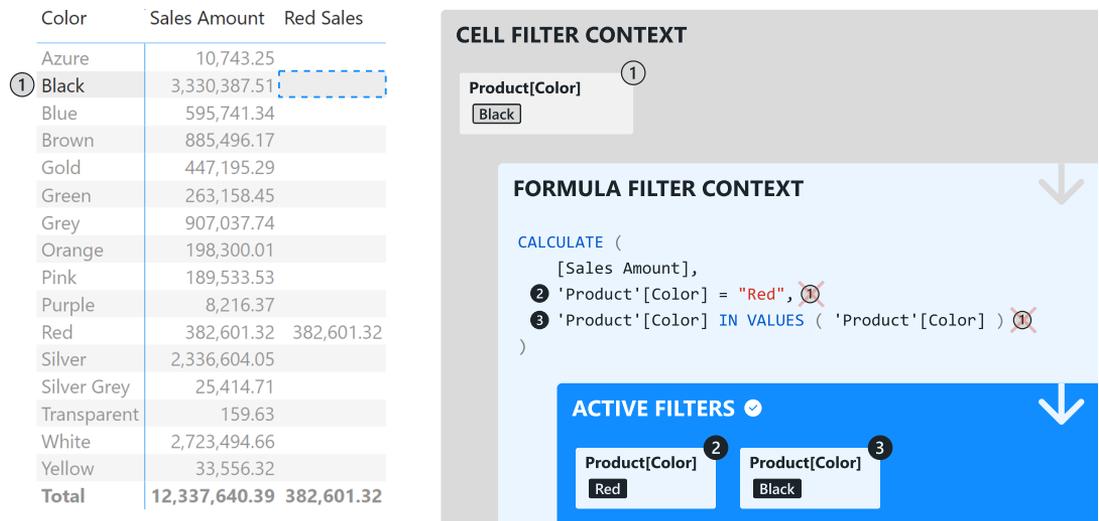
As you can see in Figure 3-27, the result is the same as `Red Sales` using `KEEPFILTERS`: The sales amount value is shown only for the row containing Red.

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	
<b>Total</b>	<b>12,337,640.39</b>	<b>382,601.32</b>

**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.



**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%
<b>Total</b>	<b>12,337,640.39</b>	<b>100.00%</b>

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

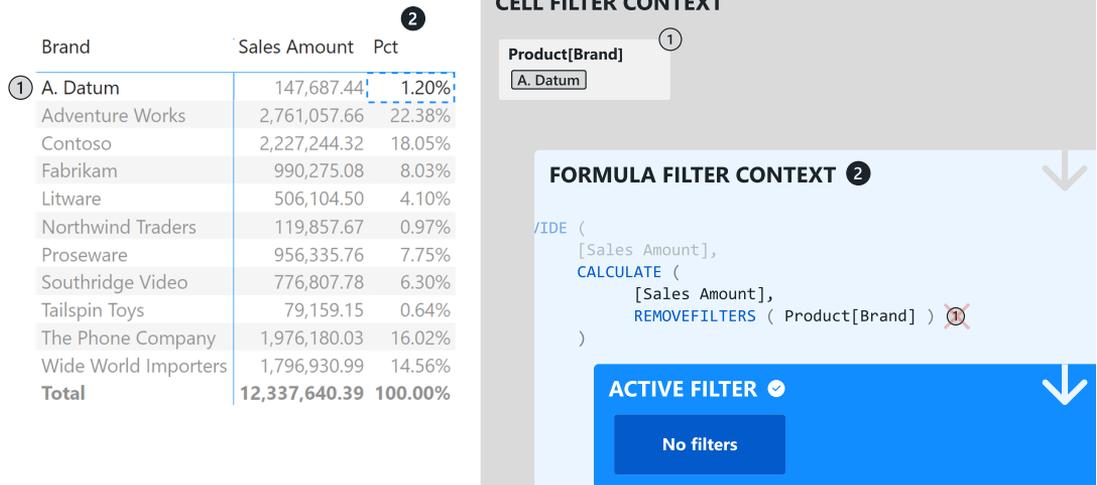
Computing the percentage in itself is not a big deal: It is enough to divide the Sales Amount of the current cell by the Sales Amount at the grand total level and then format the result as a percentage. Computing the sales amount of the current cell is simple because the filter context makes Sales Amount 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

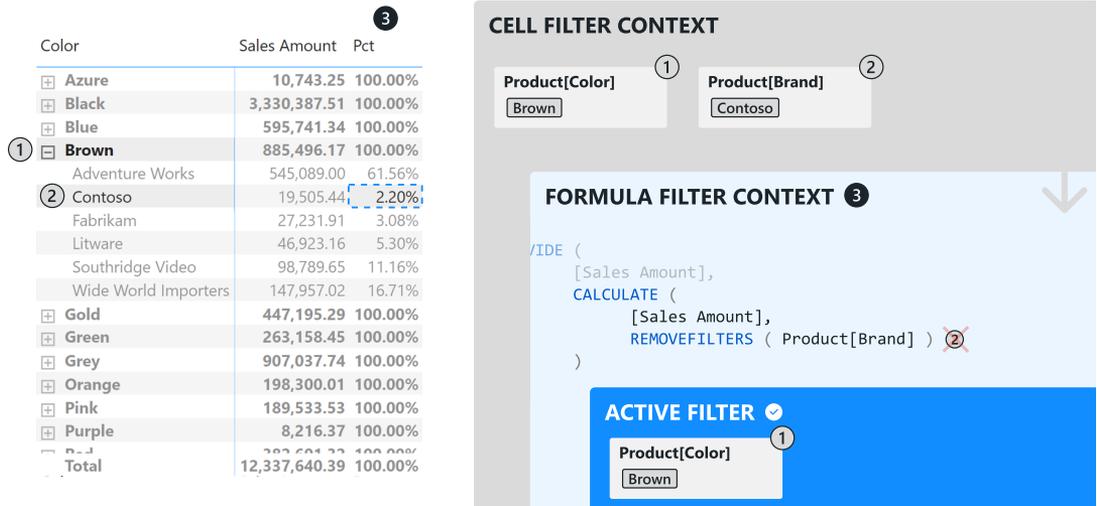
```
Pct =  
DIVIDE (  
    [Sales Amount],  
    CALCULATE (  
        [Sales Amount],  
        REMOVEFILTERS ( 'Product'[Brand] )  
    )  
)
```

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
<b>Azure</b>	<b>10,743.25</b>	<b>100.00%</b>
<b>Black</b>	<b>3,330,387.51</b>	<b>100.00%</b>
<b>Blue</b>	<b>595,741.34</b>	<b>100.00%</b>
<b>Brown</b>	<b>885,496.17</b>	<b>100.00%</b>
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%
<b>Gold</b>	<b>447,195.29</b>	<b>100.00%</b>
<b>Green</b>	<b>263,158.45</b>	<b>100.00%</b>
<b>Grey</b>	<b>907,037.74</b>	<b>100.00%</b>
<b>Orange</b>	<b>198,300.01</b>	<b>100.00%</b>
<b>Pink</b>	<b>189,533.53</b>	<b>100.00%</b>
<b>Purple</b>	<b>8,216.37</b>	<b>100.00%</b>
<b>Total</b>	<b>12,337,640.39</b>	<b>100.00%</b>

**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
<input type="checkbox"/> <b>Azure</b>	<b>10,743.25</b>	
<input type="checkbox"/> <b>Black</b>	<b>3,330,387.51</b>	
<input type="checkbox"/> <b>Blue</b>	<b>595,741.34</b>	
<input type="checkbox"/> <b>Brown</b>	<b>885,496.17</b>	
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%
<input type="checkbox"/> <b>Gold</b>	<b>447,195.29</b>	
<input type="checkbox"/> <b>Green</b>	<b>263,158.45</b>	
<input type="checkbox"/> <b>Grey</b>	<b>907,037.74</b>	
<input type="checkbox"/> <b>Orange</b>	<b>198,300.01</b>	
<input type="checkbox"/> <b>Pink</b>	<b>189,533.53</b>	
<input type="checkbox"/> <b>Purple</b>	<b>8,216.37</b>	
<b>Total</b>	<b>12,337,640.39</b>	

**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
<input type="checkbox"/> <b>Australia</b>	<b>841,828.65</b>	<b>55.45%</b>	<b>401</b>
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
<input type="checkbox"/> <b>Europe</b>	<b>3,565,296.05</b>	<b>55.68%</b>	<b>1,658</b>
<input type="checkbox"/> <b>North America</b>	<b>7,930,515.69</b>	<b>56.04%</b>	<b>3,526</b>
<b>Total</b>	<b>12,337,640.39</b>	<b>55.90%</b>	<b>5,585</b>

**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
<b>Total</b>	<b>12,337,640.39</b>	<b>12,000,000.00</b>

**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
<b>Total</b>	<b>12,337,640.39</b>	<b>11,720,758.37</b>	<b>11,814,485.41</b>

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
<b>Total</b>	<b>12,337,640.39</b>	<b>11,720,758.37</b>	<b>11,877,055.26</b>

**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

ABC Class, 188–189, 191–192

active relationships

- activating inactive relationships, 10, 747, 748

- ambiguity in, 745–747

- bidirectional filtering in, 726

- CALCULATE to change, 719

- overview of, 328–330

- in table expansion, 716, 717

- USERRELATIONSHIP and, 294

- visual depiction of, 6

ADDCOLUMNS

- adding columns via, 222–223, 394, 396

- with CALCULATETABLE, 388–389

- context transition errors with, 319–320

- iteration by, 43, 390–391

- outer/inner, 322

- overview of, 218–220, 390–394

- vs. SELECTCOLUMNS, 218

- shadow filter context created by, 733

- SUMMARIZECOLUMNS and, 225

- SUMMARIZECOLUMNS with, 229

- SUMMARIZE with, 226–229, 400, 401

- as table function, 53

additive calculations, 107–108, 110, 140, 775

ADDMISSINGITEMS, 708–709

aggregation

- ADDCOLUMNS use after initial, 393

- additive, non-additive, and semi-additive, 775

- in calculation groups, 639

- functions for, 40, 48–49, 145

- of grouped rows by GROUPBY, 404

- in iteration, 293

- with many-to-many relationships, 362

- with measures, 34, 37, 38

- in over-denormalized tables, 182

- required by SUMMARIZECOLUMNS, 797

- on a row-by-row basis, 36, 39

- in semi-additive calculations, 785

aggregators

- as bound to filter context, 271

- inside iteration, 169

- inside the row context, 164–165

- iterators vs., 43

- measure aggregators, 37

- overview of, 40–43

- in Python, 19

- statistical functions as, 56

- SUM function, 13, 37, 40

ALL\* functions

- alternate uses in, 729

- as CALCULATE modifier, 294

- overview of, 300–305

- SUMMARIZECOLUMNS and, 700, 760

ALLCROSSFILTERED, 202, 300, 303–305, 729

ALLEXCEPT function

- as CALCULATE modifier, 300, 302

- with expanded tables, 725

- KEEPFILTERS vs., 116

- as a modifier, 318

- overview of, 202, 300

- to remove filters, 317

- REMOVEFILTERS vs., 320, 517–518

- as table function, 300, 302

- tables as required/returned by, 205

- ALL function
    - ALLEXCEPT vs., 205
    - ALLNOBLANKROW vs., 165, 764
    - behavior of, 302
    - blank row returned by, 212
    - for calculated columns, 165
    - as CALCULATE modifier, 301–302
    - extending source table filter context via, 479, 482
    - outer filters ignored by, 256
    - overview of, 202–205, 300
    - parameters, 203
    - as table function, 53
    - two uses for, 203
    - value retrieval with, 120, 126, 207
  - ALLNOBLANKROW
    - ALL vs., 165, 764
    - behavior of, 303
    - and circular dependencies, 212
    - to increase rows for calculation, 202
    - overview of, 300
    - value retrieval with, 207
  - ALLSELECTED, 202, 232–235, 300, 303, 482, 725, 729–738
  - ALL/VALUES, 302
  - alphabetical sorting, 430
  - ambiguity
    - in active relationships, 745–747
    - avoiding data model ambiguity, 738–745
    - in nonactive relationships, 747–750
  - ANCHORED parameter, 547
  - AND condition, 200, 313, 440
  - AND function, 50
  - & operator, 28
  - anonymous tables, 33
  - apply semantics*
    - algorithm for, 494
    - ambiguous row context and failure of, 507–508
    - apply in, 495
    - complexity of, 508
    - current partition* and, 464
    - current row* and, 62, 471–477
    - execution of, 495
    - matching, 494–497
    - as new DAX concept, 454
    - OFFSET behavior and, 464
    - overview of, 466–476
    - PARTITIONBY behavior and, 461
    - WINDOW and, 487–494
  - arbitrarily shaped filters, 330–334, 662–669
  - arithmetic operations errors, 71, 72–73
  - auto-exists, 399–400, 660, 750–753, 755
  - auto-expand, 608–610
  - automatic casting of parameters, 365–366
  - automatic conversions, 28–29
  - Automatic value filter behavior, 755
  - AVERAGE
    - aggregation by, 40, 145
    - as column reference, 163
  - averages
    - computing, 127–129
    - debugging errors in, 131–132
    - highlighting above average items, 124–135
  - AVERAGEX, 174, 286, 334, 335, 370, 404, 483, 665, 667
  - AVG, 635, 637, 639
- ## B
- balance calculations, semi-additivity in, 775–778
  - bar charts, 90
  - bidirectional relationships
    - ambiguity and, 354–356, 738–745, 747
    - cross filtering with, 7–8, 656
    - for expanded tables, 263, 726–727
    - filter propagation and, 352
    - and many-to-many cardinality, 362, 767–768
    - minimizing use of, 8, 738, 747
    - options with, 262
    - USERRELATIONSHIP to change, 297
    - when to use, 747
  - Binary data type, 30
  - binding, dynamic column, 374–376, 494
  - blank row
    - ALL vs. ALLNOBLANKROW and, 165
    - in calculated tables, 358–360
    - dependencies with, 764–765
    - non-empty behavior and, 701

## blank row

- blank row (*continued*)
  - overview of, 207–215
  - and VALUES vs. DISTINCT, 208–210, 214
  - in virtual tables, 585
- BLANKS parameter, 546
- BLANK value, 73–75, 209, 653
- blank values, 460
- Boolean data type, 30
- Boolean logic, 27, 31, 707
- Boolean values, 55, 681
- BOTH, 296, 747
- bound columns*, 471, 474, 476, 489
- brand comparisons, 118–124
- Bravo for Power BI, 570
- budgeting, 761–774
- business intelligence (BI), 1, 2, 27, 351, 425

## C

- CALCULATE
  - to activate calculation items, 642, 644
  - algorithm for, 306–307, 329–330
  - ALLSELECTED to modify, 233
  - to apply global filters, 648
  - applying new filters with, 146, 150
  - arbitrarily shaped filters in, 664–667
  - arguments accepted by, 87
  - automatic, around measures, 170–172, 189
  - brand comparisons with, 120
  - vs. CALCULATETABLE, 387
  - circular dependencies and, 307–311
  - complexity of, 305
  - conflicts with cell filters, 94–100
  - context transition mistakes with, 317–324
  - context transition performed by, 166, 168, 170, 171–172, 272, 281–282, 284, 372–374
  - date lineage for filters placed by, 266–268
  - DATESYTD with, 512–514
  - with duplicated rows, 274
  - explicit, 172, 280–281
  - filter arguments created by, 272–273
  - filter context created/modified by, 86–87, 91, 150, 158, 167, 193, 269–270, 305–307
  - filter evaluation by, 143, 144
  - FILTER function vs., 193–194, 200, 202
  - filtering columns vs. tables in, 324–328
  - filter order and, 144
  - filter propagation and, 88
  - filters and modifiers in, 94, 294–305, 306
  - to ignore filters, 349
  - KEEPFILTERS behavior in, 298, 330, 334–337
  - outer/inner filter context for, 92–93
  - overview of, 91–94, 305–307
  - parameter-passing modes and, 369
  - parameters, 306, 440
  - RELATED inside of, 718
  - removing/restoring filters in, 117, 203
  - row context invalidation with, 270, 272, 284
  - surrounded with CALCULATE, 321–322
  - tables as filters in, 255–260, 265, 355
  - in time intelligence calculations, 515, 521–522, 550, 558, 561, 571
  - UDFs for complex filters in, 381–383
  - with USERRELATIONSHIP, 328
  - VALUES function use in, 101
  - in visual calculations, 577, 589, 602–605, 608–609
- calculated columns
  - ALL function for, 165
  - circular dependencies in, 307–311, 504–507
  - context transition in, 278–281
  - creating, 160
  - filters on temporary, 391
  - measures to avoid, 34, 38, 42
  - model size and, 14, 40
  - nested row contexts on the same table in, 188, 190
  - overview of, 34–36
  - in parent/child hierarchies, 676–680
  - query columns and, 696
  - RELATED in, 175, 718–720
  - removing dependencies with, 764–765
  - row context execution of, 158, 159, 278
  - semantics of, 159
  - table expansion and, 718

- use of `CALCULATE` in, 278, 279
  - value of, 80
  - when to use, 43–44
- calculated tables
  - blank values in, 358–359
  - creating, 120, 197, 226, 447–452
  - defined, 197
  - `EVALUATE` cannot create, 691, 694, 713
  - fixing errors in, 320–324
  - new, 317
  - relationships with, 766
  - removing dependencies with, 764–765
- `CALCULATETABLE`
  - vs. `CALCULATE`, 387
  - `CALCULATE` modifiers supported by, 390
  - context transition executed by, 172, 178
  - to create/modify filter context, 193
  - vs. `FILTER`, 388–390
  - fixing errors by, 321, 323
  - ignoring outer filters with, 314
  - overview of, 387–390
  - `RELATEDTABLE` and, 62
  - vs. `SUMMARIZECOLUMNS`, 759
  - in time intelligence calculations, 521–550, 571
  - time-restricted calculations, 764
  - UDFs for complex filters in, 382–383
  - when to use, 153–154, 388–390
- calculation groups
  - activating calculation items in DAX, 642–644
  - applying global filters with, 645–648
  - multiple/no selections in, 641–642
  - overview of, 628–635
  - in parent/child hierarchies, 684
  - precedence for, 635–639
  - purpose of, 627
  - to select filter algorithm, 799–801
  - for time intelligence calculations, 628
  - and visual calculations, 649–650
- calculations
  - across multiple measures, 384
  - additive and non-additive, 107–110
  - cardinality for correct, 182
  - context transition for concise, 289, 317
  - differing granularities in, 768–770
  - filter context as determining, 86
  - impossibility of generic, 141
  - non-additive, 107, 151, 187, 351, 353, 362
  - via row and filter context, 158
  - table functions for advanced, 199
  - time intelligent. *See* time intelligence calculations
- calendar-based time intelligent functions, 509–511, 514, 520–522, 533, 534, 536, 537–570
- `CALENDAR` function, 523, 524, 571
- calendars
  - behavior of calendar-based functions, 518
  - column categories, 533–535
  - Date table column categories, 533–535, 538
  - defining, 535
  - different-length periods in, 544–547
  - filter clearing in, 540–541
  - Gregorian, 509, 518–519, 538
  - hierarchies in, 519, 520, 523, 541–542
  - 13-month, 519
  - time-related columns, 536
  - types of, 509, 510–511
- camelCase, 378
- cardinality
  - for correct calculations, 182
  - of `CROSSJOIN` result, 416
  - iterator, 286–289, 294
  - many-to-many cardinality relationships, 304, 315, 316, 353, 362, 766, 768, 790
  - of nested iterators, 287
- card visuals, 81, 82
- cell filter context
  - conflicts with `CALCULATE`, 94–100
  - defined, 90, 270
  - errors in manipulation of, 148
  - as original filter context, 111
  - percentage calculation and change of, 145
  - removing, 127
  - `VALUES` evaluated in, 101
- cells, Excel, 11–13
- chains, relationship, 6, 176

## circular dependencies

- circular dependencies
  - blank row and, 212, 764–765
  - in calculated columns, 501, 504–507
  - overview of, 307–312
  - variables as exempt from, 244
- classic time intelligence functions, 509, 512–514, 519, 520, 522, 533, 537–570
- CLOSINGBALANCEMONTH function, 570
- CLOSINGBALANCEQUARTER function, 567, 570
- CLOSINGBALANCEYEAR function, 570
- clustering, 399–400, 750
- Coalesced value filter behavior, 660, 755, 758, 759
- code formatting, 45–48
- COLLAPSEALL function, 589–591
- COLLAPSE function, 67, 589–591, 593, 602, 620, 624, 625
- column filters, 662
- COLUMNS, 580, 594–599, 600, 621
- columns
  - active and inactive, 281–283
  - added to TOPN, 431–432
  - adding new, 216, 218–220, 222–223, 394, 396, 399
  - aggregation functions for, 40–41
  - bound* and *unbound*, 471
  - context transition and hidden, 284
  - context transition and transformation of active, 282, 284
  - cross-filtered, 133, 655–656
  - data lineage and filtering of, 265
  - date columns, 509–510, 529
  - Date table column categories, 533–535, 538
  - in DAX code, 11–13
  - dynamic binding of, 374–376
  - filter expression over multiple, 313–314
  - filter interactions over, 132
  - filter-keep columns, 510, 516–518, 521, 522, 535, 536, 540, 541
  - filtration of, 86, 92, 324–328, 655, 658
  - filtration of expanded table, 720–722
  - formatting references to, 46
  - functions filtering single columns, 92
  - group-by columns, 579, 580, 582, 583, 611, 728, 730, 750, 752
  - grouping, 217, 220, 224, 405
  - headers for, 415, 710
  - iteration of, 167, 668
  - naming, 219–220
  - native and related, 264
  - query columns, 695, 696–697
  - referenced with table names, 172
  - references in visual calculations, 66
  - references vs. values for, 161–164
  - REMOVEFILTERS on tables vs., 140
  - removing filters from, 104, 137–140
  - renaming, 216
  - row context to evaluate values of, 158, 160
  - SELECTCOLUMNS behavior on, 215–216
  - single values for, 123
  - sorting order of, 137–140, 577
  - syntax for referencing, 25–26, 33
  - temporary, 219–220, 391, 399, 401, 405
  - time-related columns, 518, 521, 536, 542, 550–553
  - in virtual tables, 573–576, 583–584
  - in visual shape, 577–583
  - See also* calculated columns
- comments
  - naming UDF, 379
  - for readable code, 26–27
- comparing brands, 116, 118–124
- complete column category, 533, 534
- complex formulas
  - ALLSELECTED in, 235
  - apply semantics* in, 472
  - arbitrarily shaped filters in, 664
  - for calculated columns, 34
  - calculation groups for, 627
  - comments for, 27
  - context transition in, 270
  - error detection in, 126
  - flexibility and complexity, 539
  - formatting readable code for, 45
  - manipulating filter context in, 106, 113
  - multiple filter contexts in, 94
  - reusing existing measures in, 281
  - split into smaller chunks, 229
  - sub-formula detection in, 250

- table filter restrictions in, 792
- UDFs for, 381–383, 386
- VALUES for, 103
- variable definition in, 238
- variables to sub-divide, 44, 130, 142, 238
- for visual calculations, 66, 67
- Composite Models, 159
- computations
  - comments in, 26–27
  - data types for, 27–30
  - of denominators, 104–105
  - like-for-like computations, 787–801
  - operators for, 31–32
  - syntax for, 25–26
  - of totals in Power BI, 106–110
- CONCATENATEX function, 231
- conditional formatting, 144, 148
- conditional statements, 32
- constant, variables as, 238, 241–243
- CONTAINS, 447
- CONTAINSROW, 447
- context transition
  - as absent from GENERATE, 716
  - automatic, around measures, 170–172, 189, 372
  - avoiding, 253–254, 274–281
  - behavior of, 271–274, 284
  - CALCULATE as executing, 166, 168, 170, 171–172, 272, 281–282, 284, 306, 667, 719, 722
  - in calculated columns, 278–281
  - CALCULATETABLE as executing, 390
  - code optimization with, 181–182
  - common errors in, 317–324
  - complete definition of, 269–270
  - complexity of, 270
  - with duplicated rows, 274–278, 311
  - in expanded tables, 722, 728–729
  - FILTER function use with, 200
  - with iterators, 285–286, 289–294
  - KEEPFILTERS with, 330–337
  - nested iterations and, 288–289
  - overview of, 166–168
  - parameter-passing modes and, 372–374, 386
  - performance of, 274, 284
  - vs. regular iterator use, 178–182
  - table names and, 26
  - as unique to DAX, 10, 22–23
  - when/how to use, 168–170
- conversions
  - automatic vs. explicit, 29
  - conversion functions, 58
  - errors in, 71–72
- coordinates, 81–84, 88
- COUNT, 48
- COUNTBLANK, 49
- COUNTRROWS
  - aggregation by, 49, 169
  - with CALCULATETABLE, 388–389
  - and DISTINCT vs. VALUES, 211
  - DISTINCTCOUNT vs., 134
  - execution of innermost, 170
  - filters invisible to, 353
  - vs. ISEMPTY, 660
  - KEEPFILTERS behavior and, 299
  - for result of ALL, 126
  - in time intelligence calculations, 556
- CROSSFILTER
  - ambiguity and use of, 745
  - as CALCULATE modifier, 294, 306
  - in expanded tables, 727
  - parameters, 296
  - relationships deactivated by, 361
- cross-filter direction
  - changes in, 88, 296
  - in like-for-like computations, 790
  - in relationships, 7
  - single, 173
  - VALUES and retaining of, 302
- cross-filtering
  - direct/indirect, 655–656
  - in inner cell filter context, 133
- CROSSJOIN, 259, 313, 415–418, 494
- currency conversion, 380–381
- Currency data type, 29
- CURRENCY function, 58, 449
- CURRENTGROUP, 402, 405
- current partition, 461, 464, 466

**current row**

- ambiguous row context and, 507, 508
  - apply semantics* and, 471–477, 489
  - OFFSET and, 464–466, 471
  - ranking of, 497
  - subset of source table to find, 496
  - table of candidates for, 495
  - WINDOW and, 482–484, 493
- customer behavior analysis, 425

**D**

database theory, 2

**DataFrame**

- filters applied to, 19
- merge operation on, 17–18

**data lineage**

- column filtration based on, 265–268
- vs. IN filter use, 118
- INTERSECT as retaining, 423
- for query table columns, 695
- SELECTCOLUMNS as maintaining, 396
- in semi-additive calculations, 787
- and TREATAS, 268–269, 440
- UNION as maintaining, 420–422
- as unique to DAX, 22–23

**data model**

- avoiding ambiguity in, 738–745
- avoiding circular dependencies in, 307
- with bidirectional relationships, 8
- column filtration in, 86
- context/relationship interaction in, 173
- correct granularity in, 183
- DAX code as affected by, 5–8, 10
- denormalized, 726
- invalid relationships in, 212
- joining tables in, 439
- native, 674
- relationships in, 262, 439
- vs. semantic model, 2–3
- tables in, 12
- writing measure variations for, 627, 628–629

Data Modeling for Power BI course, 10

data retrieval syntax, 13

DATATABLE, 447, 449–450

data types, 27–30, 365, 449

DATEADD, 542–553

date functions, 59

DATESINPERIOD function, 556

DATESYTD, 512–514, 520, 537, 542, 557, 560

**Date table**

building a, 522–533, 570

for classic vs. calendar-based functions, 511, 533

column categories, 533–535

for fiscal calendars, 528

for a Gregorian calendar, 524–526

for a monthly calendar, 526–527

multiple date columns in, 510, 529

multiple relationships to, 530–531

multiple tables, 510, 531–533

need for separate, 548

for weekly calendars, 528–529

DateTime data type, 27, 29, 30, 58, 59, 511, 522

DATEVALUE, 58, 59

**DAX**

activating calculation items in, 642–644

*apply semantics* in, 454

avoiding code repetitions in, 69

column references vs. column values in, 163

data model effects on, 5–8

details in, 317

filter context creation in, 86–87

formatting code, 45–48

four basic concepts in, 22–23

as a functional language, 13, 16, 17, 21–22, 45

learned via making controlled mistakes, 339–351

leveraging building blocks of, 394

mastering theory of, 14, 22, 23, 289, 305, 312, 715, 801

as measure-driven, 3

measure re-use in, 172

vs. other programming languages, 11–21

Power BI for, 405

query execution, 198–199

- semantics, 261
  - simplicity principle in, 377
  - syntax, 25–33
  - table functions in, 195
  - type-handling system, 27
  - unique features of, 3–5, 10
  - variable evaluation in, 249–251
    - for visual calculations, 65, 66
  - DAX Guide, 801
  - DAX Studio, 199, 226, 695
  - debugging
    - arbitrarily shaped filters and challenges in, 664–665
    - with CALCULATE, 168
    - DateTime data type bugs, 30
    - with DEFINE MEASURE, 198
    - duplicated rows, 15
    - matrix visuals for, 131–135
    - query measures for, 695
    - UDF building blocks for, 386
  - Decimal data type, 29
  - declarative languages, 16
  - DEFAULT, 460
  - DEFINE, 713
  - DEFINE COLUMN, 583–584, 696–697
  - DEFINE FUNCTION, 697–698
  - DEFINE keyword, 690
  - DEFINE MEASURE, 198, 694–695
  - DEFINE MPARAMETER, 698
  - DEFINE TABLE, 695–696
  - DEFINE VAR, 691–694
  - DEFINE VISUAL SHAPE, 698
  - denominator computations, 104–105, 114–118
  - denormalization, 182, 726
  - DENSE argument, 497
  - densification, 580, 584, 585–586
  - dependencies
    - circular, 212, 307–312, 501, 504–507
    - and column/table processing order, 307
    - context transition and unexpected, 284
    - parameters to expose, 375–376
    - in time intelligence calculations, 540
  - developers
    - developer vs. user errors, 122
    - model knowledge by, 214
    - responsibility for functional code, 651
    - understanding intention of, 202
  - dimensions, 9, 766, 771–772
  - direct column filters, 655–656
  - direction, in visual calculations, 600
  - DirectQuery, 159
  - disabling relationships, 296, 361
  - discounts, applying, 109–110
  - DISTINCTCOUNT, 49, 151, 222, 297
  - distinct counts, 107, 221–222, 775
  - DISTINCT function
    - circular dependencies with, 212, 501, 504
    - lineage as removed by, 420–422
    - overview of, 207–215
    - as parameter of iterators, 212
    - removing duplicates via, 184–185, 419, 506
    - as table function, 53
    - value retrieval with, 207
    - vs. VALUES, 210
  - DIVIDE function, 50
  - documentation, variables for, 251–252
  - duplicate code, avoiding, 380–381
  - duplicates, removing, 15, 184–185, 419, 499–504, 506
  - dynamic binding of columns, 374–376
  - dynamic budget allocation, 769–772
  - dynamic format strings, 615, 635
  - dynamic M parameter, 698
  - dynamic reports, 342
  - dynamic user interfaces, 710
- ## E
- EARLIER function
    - to access outer row context, 192–193
    - as legacy function, 193
    - parameters for, 193
  - empty values, 71, 73–75, 77, 100, 753
  - ENDOFMONTH function, 566–570
  - ENDOFQUARTER function, 566–570

## ENDOFYEAR function

- ENDOFYEAR function, 566–570
- ERROR function, 79
- error-handling functions, 78
- errors
  - ambiguous row context, 507–508
  - automatic conversion, 28–29
  - CALCULATE and table syntax errors, 258
  - calculation group, 640, 641–642
  - circular dependency errors, 308, 504–507
  - common context transition errors, 317
  - in comparing percentages, 149
  - correcting, 75–77
  - DateTime data type errors, 30
  - in DAX expressions, 71–80
  - detection of, 78, 80
  - duplicate row errors, 274–276, 499–504
  - failure of simple formulas, 722
  - in filter context, 362
  - in filter expression with SUMMARIZE, 315–316
  - fixed via understanding, 23
  - INDEX function, 459
  - intentional generation of, 79, 641
  - invalidated row context, 272
  - from invalid relationships, 210–212
  - learning from, 145, 362
  - via nested row contexts on the same table, 188
  - no active row context, 160
  - prevented in multi-step measures, 125–126
  - row/filter context, 157
  - syntactical, 185
  - table expansion, 727
  - in totals column, 148–150
  - wrong measure results, 121, 122
  - wrong parameter-passing mode, 367
- EVALUATE statements
  - ALL\* functions and, 301
  - matrix content returned by, 415
  - overview of, 689–698
  - query vs. expression variables in, 691–694
  - in SUBSTITUTEWITHINDEX function, 711
  - syntax for, 197–199, 690–691
  - tables as required/returned by, 699
- evaluation context
  - applying filters to, 19
  - apply semantics* and, 465, 466, 471, 472, 474
  - of CALCULATE, 144, 305, 306
  - for calculated columns, 40
  - filter context in, 84, 157
  - intricacies of, 81
  - mastering theory of, 14
  - for measures, 40
  - row context in, 84, 156, 157
  - SUMX arguments in, 161
  - two components of, 84, 156, 157, 158
  - as unique to DAX, 10, 22–23
  - variable evaluation in, 249
  - vs. visual context, 587
- evaluation order, 305, 307, 308
- Excel formula language
  - DAX vs., 11–15
  - as development language, 10
  - empty values in, 75
- Excel PivotTable, 628
- EXCEPT, 361, 424–426
- EXPANDALL function, 589–591
- expanded tables
  - access to columns in, 715
  - ALLEXCEPT to remove filters from, 205
  - bidirectional expansion, 727
  - context transition in, 274, 728–729
  - definition of, 716
  - filtering of tables/columns in, 720–722
  - vs. filtering tables, 324–328, 726–728
  - and filter propagation, 263, 265, 355, 357
  - on one side of the relationship, 261–263
  - overview of, 261–265, 715–729
  - referencing, 715, 722
  - via regular relationships, 264
  - RELATED with, 715–720
  - REMOVEFILTERS on columns of, 140
  - SUMMARIZE for, 402
  - as unique to DAX, 22–23
  - using table filters in measures, 722–726
- EXPAND function, 589–591, 593, 602, 605, 620, 625

explicit CALCULATE, 280–281  
 explicit conversions, 29  
 explicit filter context, 733  
 explicit table filters, 257  
 expr (expression) parameter, 68, 285  
 Expression parameter, 161, 285

expressions  
   errors in, 71–80  
   expression variables, 691–694  
   GENERATE to merge tables with, 433–437  
   iterator evaluation of, 285  
   iterators to aggregate, 41  
   table, 247, 257–260, 287, 419  
   variables in, 237, 245–247  
   variables to avoid repetition in, 44  
 EXPR parameter (UDF), 364, 367, 368, 383  
 EXTENDING parameter, 545–547

## F

factors, common, 380–381  
 fact tables, 9  
 filter context  
   adding tables to, 722  
   aggregators as bound to, 271  
   and browsing depth of reports, 681  
   CALCULATETABLE to modify, 390  
   CALCULATE to create/modify, 86–87, 91, 150, 167, 193, 305–307, 722  
   calculation group, 629, 641, 646, 648  
   checking for active filters in, 655–658  
   column sort order for, 137–140  
   components of, 158  
   conflicts between contexts, 94–100  
   context transition and, 270–271, 281–282, 284  
   coordinate system and, 83  
   effect of REMOVEFILTERS on, 135–141  
   errors via, 148–149, 362  
   as evaluation context component, 84  
   explicit filter context, 733  
   for expr/value parameters, 69  
   filter interaction in, 132  
   flow from cells to measure, 145

formula behavior and, 651, 652–653, 683  
 inner/outer, 92–93, 95–97  
 ISEMPTY to check for no filters in, 660–662  
 keeping/removing filters in, 116  
 manipulation of, 105, 111, 113, 135–141, 148  
 measures evaluated in, 5, 40  
 multiple, 92–94  
 overview of, 84–90, 270–271  
 row context vs., 157  
 shadow filter contexts, 729  
 simple vs. arbitrarily shaped filters in, 662  
 slicer effect on, 120  
 and SUMX evaluation, 165  
 table expansion in, 717  
 time intelligence function modification of, 512, 515–516, 537, 540, 542  
 VALUES to create new, 143  
 VALUES vs. FILTER filtration in, 658–660

### FILTER function

ADDCOLUMNS with, 391  
 vs. CALCULATE, 193–194, 200, 202  
 vs. CALCULATETABLE, 388–390  
 filtering table expressions via, 20  
 ISONORAFTER with, 706  
 iteration by, 43, 169, 170  
 iteration of SUMMARIZE by, 265  
 matrix query via, 410  
 nesting, 200  
 overview of, 199–202  
 parameter-passing modes and, 370  
 syntax for, 199  
 as table function, 53, 54  
 time-delimited sales with, 180  
 VALUES vs., 658–660  
 filtering tables, 387–405  
 filter-keep columns, 510, 516–518, 521, 522, 535, 536, 540, 541  
 filter propagation  
   to apply global filters, 645–646  
   bidirectional cross filtering and complexity of, 738  
   expanded tables and, 263, 265  
   via relationships, 88, 351  
   SUMMARIZE to control, 357

## filter propagation

filter propagation (*continued*)

- through bidirectional filters, 352–356
- as unique to DAX, 22–23

filters

- ALL as ignoring, 205
  - ALLEXCEPT to remove, 317, 725
  - applied to evaluation context, 19
  - arbitrarily shaped, 330–334, 662–669
  - bidirectional, 352
  - CALCULATE and order of, 144
  - CALCULATE and types of, 94
  - CALCULATE creation of filter arguments, 272–273
  - checking for active, 655
  - color filters, 256
  - complex CALCULATE, 381–383
  - cross-filtering, 133, 655–656
  - expressed over multiple columns, 313–314
  - global, 645–648
  - identical results from different filter handling, 351, 354–355
  - interactions between, 132
  - modifiers vs., 294
  - numeric, 256
  - in Power BI reports, 405
  - propagation of, 88, 173
  - removing, 141, 317
  - restoring, 117
  - for time-delimited computations, 445
  - using tables as, 440–447
  - VALUES to create, 103
  - in visual calculations, 587–589, 602–605
  - visuals as creating, 88–90
- financial functions, 56
- FIRSTDATE function, 562–566
- FIRST function, 606–608
- FIRSTNONBLANK, 568
- fiscal calendars, 510, 528
- foreign keys, 15
- FORMAT, 58
- formatting
- dates, 58
  - DAX code, 45–48
  - moving averages, 619

formulas

- calculated columns via, 34
  - column sort order for, 137–140
  - context transition for concise, 289
  - details in, 23–24, 133, 150, 317
  - filter context and behavior of, 651, 652–653, 683
  - filter context detection for, 90
  - filter contexts in, 93
  - fixing errors in, 125, 131–135
  - formatting, 46
  - as function calls, 16
  - identical results from different filter handling, 351, 443
  - imperfect code in, 113
  - Power BI to create, 405
  - relationship direction in, 177
  - single-line comments and readable, 27
  - single values returned by, 195
  - using row context formulas, 178–182
  - VALUES and semantics of, 214
  - variables as specific to each, 384
  - variations of, 627, 628–629
  - weak vs. strong, 114, 115
- functional languages, 13, 16, 21–22
- functions
- common factors via, 380, 381
  - to create parent/child hierarchies, 674, 676
  - defining, 155
  - dual nature of, 203
  - evaluating innermost first, 197
  - Expression/Value parameters and, 161
  - formatting, 46
  - learning, 23
  - native vs. UDF, 363
  - operators vs., 31
  - overengineering of, 377
  - overview of, 48–64
  - parameters for, 31, 68
  - query functions, 697–698, 699–713
  - simplifying measures with, 155–156
  - user-defined. *See* UDFs (user-defined functions)
  - visual calculation functions, 605–615
- See also specific function by name*

## function signatures

- defined, 162–163
- of SUM, 164
- for time intelligence functions, 509
- for user-defined functions, 363, 364

**G**

## GENERATE

- arguments accepted by, 433
- avoiding invalid combinations via, 314
- context transition as absent from, 716
- iteration by, 43
- to merge tables with expressions, 427, 433–437
- overview of, 433–437

## GENERATEALL, 433–437, 452, 496

## GENERATESERIES, 338, 450–452

## generating errors, 79

## generic calculations, 141

## global filters, 645–648

## global variables, UDFs for, 384–386

## granularity, data, 183, 289, 293, 349, 583, 768–772, 788

## Gregorian calendar

- calendar-based time intelligence and, 518–519
- classic time intelligence and, 538
- Date tables for, 524–526
- time intelligence functions for, 509

## GROUPBY, 402–405

## group-by tuples, 752, 753, 755, 757, 758

## grouped-by operations, 654

## grouping columns, 217, 220, 224, 405

## grouping tables, 387–405

## growth, visual representations of, 142–150, 620

## Growth measure, 4, 147, 622, 623

**H**

## HASONEFILTER function, 660

## HASONEVALUE function, 123, 149, 230, 248, 652–655

## headers, column, 415

## hidden columns, sorting via, 137–140, 284

## Hide Members property, 687

## hierarchies

- COLLAPSE/EXPAND to move visual context in, 589, 590, 591, 605
  - computing percentages over, 669–674
  - creating parent/child hierarchy, 674–687
  - detecting filters in, 658
  - hierarchical shift in time intelligent calculations, 541–542
  - parent/child levels in visual calculations, 591, 593, 599, 651
  - ratio-to-parent calculations, 669–674
  - showing/hiding nodes of, 680
  - testing inner toward outer levels of, 674
  - in virtual tables, 577, 579, 587
  - via visual shape, 577, 583
- HIGHESTPARENT, 599–600
- highlighting above average items, 124–135

**I**

## IF arguments, 163

## IFERROR, 50, 76–80

## IF function

- conditional statements via, 32
  - in context transition, 279
  - equivalent DAX/Excel syntax for, 12
  - FILTER function vs., 201
  - IFERROR function and, 76
  - a logical function, 50
  - for margin percentages, 639
  - naming columns with, 219
  - order of conditions in, 674
  - parameters, 32
  - SUMX use of, 287
  - SWITCH converted to nested, 52
  - transforming percentages with, 191
- IF statement, 26, 50, 77, 230, 249, 492, 653
- IGNORE, 702, 754
- implicit measures, 36, 37–38
- Import mode, 14, 159
- inactive relationships
- ambiguity in, 747–750
  - leveraging, 10

## inactive relationships

- inactive relationships (*continued*)
    - USERRELATIONSHIP and, 294
    - visual depiction of, 6
  - IN condition, 94
  - Independent value filter behavior, 755, 758, 759, 760
  - INDEX function, 62, 293, 453, 454–464, 797
  - indirect column filters, 655–656
  - IN filter, restoring filters via, 117
  - information functions, 55
  - IN keyword, 447
  - inner filter context
    - CALCULATE as creating, 92–93
    - cross-filtering in, 133
    - inner/outer filter conflicts, 95–97, 192
    - KEEPFILTERS for, 116
    - passing filters from outer to, 101–103, 142
  - inner row context, 161, 192–193, 281, 282, 299, 494
  - IN operator, 33, 267, 268, 269
  - Integer data type, 29
  - intercepting errors, 75–77
  - INTERSECT, 423–424
  - interval-to-date calculations, 557–560
  - INT function, 58
  - invalid relationships, 210–212
  - invalid values, 208, 314–315
  - ISAFter, 705–707
  - ISATLEVEL, 593, 604, 611–612
  - ISBLANK, 74, 77, 163
  - ISCROSSFILTERED, 655–658, 685
  - ISEmpty, 660–662
  - ISFILTERED, 655–658
  - ISINSCOPE, 543, 654–655, 681
  - ISONORAFter, 705–707
  - ISSELECTEDMEASURE, 639–640
  - iteration
    - by ADDCOLUMNS, 223
    - aggregators inside, 169
    - ALLSELECTED in, 734–738
    - arguments accepted in, 285–286
    - context transition with, 285–286, 289–294
    - filter context and, 161
    - by FILTER function, 193
    - by GENERATE, 433
    - granularity and table iteration, 185–186
    - iterator cardinality, 286–289
    - iterators vs. regular aggregators, 43
    - KEEPFILTERS with, 298, 299, 306, 330–337
    - measures inside, 253, 289, 291
    - nested, 187, 227, 287, 288–289, 291
    - order of operations for, 293
    - over *bound columns*, 471
    - over combination of columns, 344–347
    - over VALUES, 231
    - overview of, 40–43
    - parameters for, 41, 285
    - RELATEDTABLE and, 61
    - as row context, 158–159, 271
    - row context with iterators, 160–161
    - ROWS/COLUMNS/ROWPAGES used in, 596
    - SELECTCOLUMNS as iterator, 217
    - in semi-additive calculations, 773–776
    - shadow filter contexts created by, 732–733, 734
    - of single table, 173
    - starting an, 160
    - by SUMX, 271
    - of table function results, 196
    - as unique to DAX, 14, 22–23
    - with VALUES vs. DISTINCT, 214
    - variable use with, 241
- ## J
- joining tables, 427, 437–440
- ## K
- ### KEEPFILTERS
- for active columns in outer row context, 282–283
  - in applying global filters, 648
  - around time intelligence functions, 517
  - as filter argument modifier, 294, 298, 306, 774
  - filtering columns with, 328
  - with iterators, 298, 299, 306, 330–337, 668
  - in like-for-like computations, 797
  - overview of, 94–100
  - percentage calculations with, 116

SUMMARIZECOLUMNS and, 760  
VALUES as alternative to, 101–103

## keys

parent key, 675, 676  
primary keys, 15, 274  
relationship key, 6, 516  
SQL, 15

Kimball methodology, 9

**L**

LASTDATE function, 562–566, 778–779  
LAST function, 606–608, 617  
LASTNONBLANK, 568  
lateral-shifting functions, 536  
lateral shift in time intelligence calculations, 541–542  
lattice navigation, 592–593, 623  
leaf nodes, 675, 686, 687  
LEFT OUTER JOINS, 15  
like-for-like computations, 787–801  
limited relationships, 208, 264, 315  
lineage. *See* data lineage  
local columns, 494  
logical functions, 50–52  
LOOKUP function, 605, 608–610  
LOOKUPVALUE, 677, 719–720  
LOOKUPWITHTOTALS function, 608–610  
Lotus 1-2-3, 30

**M**

many side of a relationship  
    bidirectional filtering from, 7  
    expansion and, 263  
    syntax for, 6  
    table expansion from, 262  
many-to-many relationships, 6, 176, 262, 315,  
    351–362, 715, 790  
many-to-one relationships, 60, 173, 176, 221, 261,  
    262, 352, 356, 742, 766–768  
margin percentages, 107, 639, 645  
Mark As Date Table, 516, 517, 522, 571  
MATCHBY, 476, 495, 496–497, 501–504

matching, in *apply semantics*, 494–497  
mathematical functions, 55–56  
mathematical theory, 23

## matrix visual

axes in, 81  
coordinates in, 82  
debugging with, 131–135  
filters applied to, 411–412  
generic calculations for, 141  
queries to analyze, 406–415  
in visual calculations, 577, 579

## MAX

aggregation by, 40, 145, 293  
as calculation item, 635  
parameter calculations with, 41

## MDX, 3, 10, 20

MeasureCheck variable, 123, 124

## measures

adding filters based on, 411–412  
additive, 775  
aggregation with, 37, 40  
automatic CALCULATE around, 170–172, 189  
avoiding table names in front of, 172, 219  
bidirectional relationships in, 742  
calculated columns vs., 34  
calculation groups for variations on,  
    627, 628–629, 632  
coordinates and value computation by, 82  
creating row context for, 160  
DAX definition of, 3, 4  
evaluating performance of, 249–250  
filter context effects on, 85–86  
FILTER function use on, 200  
fixing errors in, 125  
fixing errors via, 323–324  
formatting, 46  
global calculation changes via, 36  
inside iteration, 253  
minimizing ALLSELECTED in, 734  
misspelling errors, 640  
model access for, 622  
non-additive, 775  
overview of, 37–40

## measures

### measures (*continued*)

- parameter tables to alter, 120
  - query measures, 694
  - reusing existing, 172, 281, 289
  - semi-additive, 777
  - for stand-alone tables, 339
  - static vs. dynamic strings in, 635
  - step by step building of, 113
  - syntax for referencing, 26
  - table filters in, 324–328, 722–726
  - temporary tables in, 195, 266
  - unexpected results via incorrect use of, 137–138
  - universal functionality for, 121, 472
  - use of CALCULATE in, 91
  - use of SUMMARIZE in, 401
  - use of WINDOW in, 486
  - using functions with, 155–156
  - using table filters in, 722–726
  - variable definition with, 237
  - variables in multi-step, 125–126, 130
  - variables vs., 242
  - visuals and, 141
  - when to use, 43–44
- memory, RAM, 34, 43
- metadata, 3
- metric parameters, 383
- Microsoft Analysis Services
- calculated tables in, 197
  - data types for, 29
  - DAX as used in, 1
  - parent/child hierarchies in, 674
- Microsoft Fabric, 1, 510
- Microsoft Learn, 801
- Microsoft Power BI
- axes and coordinates in, 81, 84
  - calculated tables in, 197
  - calculation groups in, 628
  - date tables in, 27, 29, 510, 570
  - DAX for, 20–21
  - implicit measures, 36
  - query view in, 197, 199, 226, 689, 695
  - reading queries in, 405–415
  - semantic models, 1, 2–3

SUMMARIZECOLUMNS in, 224

table functions in, 53

### Microsoft Power Pivot

- calculated tables as absent from, 197
- data types in, 27, 29
- date tables in, 510, 570
- roots of DAX in, 1, 11

### MIN

- aggregation by, 40, 48, 145
- as calculation item, 635
- parameter calculations with, 41

MINX, 43, 179, 180, 635

missing values, 71, 73–75

model columns, 265, 266, 494–495

model-dependent/-independent UDFs, 376–377, 378

### modifiers

- ALL as, 203
- ALLSELECTED as, 233
- of CALCULATE, 294–305
- as CALCULATE filter type, 94
- KEEPFILTERS as, 96–97
- result modifiers in EVALUATE, 690

monthly calendars, 511, 526–527

MOVINGAVERAGE, 612–613, 614, 617

moving averages, 489, 615–620

M parameters, 698

multidimensional databases, 674

multiline comments, 26–27

MyDatesYTD function, 513–514

## N

### naming

- camelCase for, 378
- of columns, 219–220
- names reserved by Microsoft, 378
- of parent/child hierarchies, 674
- PascalCase for, 364
- of query measures, 695
- of variables, 239–241

native columns, 264

native functions, 363, 364

NATURALINNERJOIN, 437–440

NATURALLEFTOUTERJOIN, 415, 437–440  
 navigating the virtual table lattice, 592–593, 623  
 nested functions  
   evaluation of innermost first in, 197  
   nesting CALCULATE, 91–93  
   nesting FILTER, 200  
   parameter limits and, 31  
   performance and order of, 16  
 nested iterators, 287, 288–289, 291  
 nested row contexts, 180, 187–193, 252, 281  
 nesting VAR/RETURN statements, 244  
 NEXT function, 606–608  
 nodes, depth of hierarchical, 680–681, 683, 685–686, 687  
 nonactive relationships, 6, 10, 294, 747–750  
 non-additivity, 107–110, 151, 187, 351, 353, 362, 775  
 NONE, 296, 747  
 non-empty behavior, 701, 752, 753–754  
 NOT function, 50  
 NULL value, 75  
 number/string conversions, 28  
 numeric filters, 256

## O

OFFSET, 464–482  
 one side of the relationship  
   blank row on, 208, 214, 215  
   expansion from, 261, 263, 726  
   filtering as beginning from, 7  
   invalid blank value on, 209  
   iteration of, 175  
   syntax for, 6  
 one-to-many relationships, 62, 174, 176, 262, 310, 352, 356, 742  
 one-to-one (1:1) relationships, 6, 174, 176, 261, 262, 263, 727  
 ONEWAY, 296  
 ONEWAY\_RIGHTFILTERSLEFT, 296  
 ONEWAY\_RIGHTFILTERSRIGHT, 296  
 OPENINGBALANCEMONTH function, 570  
 OPENINGBALANCEQUARTER function, 567, 570  
 OPENINGBALANCEYEAR function, 570

opening/closing balance calculations, 566–570  
 operator overloading, 28  
 operators, DAX, 31–32  
 optimization  
   calculation groups for, 629  
   with context transition, 181–182  
   of filter expression in large models, 316  
   ISFILTERED for advanced, 658  
   with nested iterators, 289, 291  
   primary key dependencies and, 312  
   writing efficient code, 374  
   writing equivalent alternatives for, 443  
 OR condition, 313, 416, 440–443  
 ORDERBY, 453, 457, 476, 498  
 OR function, 50  
 original evaluation context, 305  
 original filter context, 111  
 outer filter context  
   ALL as ignoring, 256  
   CALCULATE as overriding, 146  
   inner/outer filter conflicts, 92–93, 95–97, 192  
   KEEPFILTERS to retain, 116  
   as valid in row context, 161  
   VALUES in, 320  
   VALUES to re-create, 101–103, 142  
 outer row context, 161, 192–193, 281, 282, 299  
 over-denormalized data, 182–187

## P

PARALLELPERIOD, 553–557, 569  
 parameters  
   added to what-if analysis, 452  
   automatic casting of, 365–366  
   evaluation of, 16  
   function parameters, 31, 68  
   iterator parameters, 41, 285  
   minimum/maximum of, 41  
   UDF parameters, 363, 364–365, 378–379  
   value vs. expression, 368  
   window function parameters, 454  
   *See also specific function by name*  
 parameter table, 120

## parent/child hierarchies

- parent/child hierarchies
  - creation of, 674–687
  - flattened into column-based hierarchies, 674, 675
  - naming conventions for, 674
  - ratio-to-parent calculations, 669–674
  - in visual context, 591, 593, 598, 599, 600, 602, 624, 625
- partial column category, 533, 534
- PARTITIONBY, 453, 460–464, 476, 485, 600
- PascalCase, 377
- PATH function, 676, 679
- PATHITEM parameter, 677
- percentages
  - aggregating values for, 38–39
  - ALLSELECTED to compute, 234–235
  - for brand comparisons, 119
  - comparison errors, 149
  - computing percentages over hierarchies, 669–674
  - of current year, 144–146, 149
  - discount percentage slicers, 337
  - dividing current cells by total, 114
  - in dynamic budget allocation, 769–772
  - expanded tables and incorrect, 724
  - margin percentages, 107, 639, 645
  - of previous years, 144–147, 149, 620
  - REMOVEFILTERS to compute, 104, 135–141
  - of sales calculations, 317–324
  - static budget allocation based on, 763
  - of total growth, 142–150
  - visual calculations for, 66–67
  - year-by-year comparisons of, 603
- performance
  - calculated columns and slowed, 34, 38
  - of CALCULATE vs. FILTER, 194
  - of context transition, 274, 284
  - of CROSSJOIN, 416
  - of error-handling functions, 78
  - evaluating measure performance, 249–250
  - of EXCEPT, 426
  - of GROUPBY, 403
  - iterator, 43
  - iterator cardinality and, 287, 294
  - of nested iteration, 187, 289, 291
  - query measures to tune, 695
  - in semi-additive calculations, 785
  - variables and improved, 239, 254
  - visual calculation, 573
  - writing efficient code to optimize, 374
- Performance Analyzer, 406
- + operator, 28
- Power BI. *See* Microsoft Power BI
- Power Query, 44, 338, 358, 523, 527, 698
- Power Query Editor, 698
- PRECISE parameter, 545
- PREVIOUS function, 606–608, 621
- PREVIOUSYEAR function, 555
- primary keys, 15, 274, 312, 457
- PRODUCT, 48
- product categories, 6
- programming languages, 10, 24
- Python vs. DAX, 11, 17–20

## Q

- queries
  - authoring, 199, 415
  - complexity of, 406
  - CROSSJOIN in, 416
  - EVALUATE for executing, 197–198, 689
  - to find invalid relationships, 214–215
  - optimizing, 16
  - Power BI query view, 197, 199, 226, 689
  - power of, 199
  - primary structure of, 407
  - query functions, 699–713
  - query variables, 691–694
  - readable code and improved, 202
  - reading Power BI, 405–415, 675
- query expression, 690

## R

- RAM, 34
- RANGE, 614–615

- range lookups, 376
- RANK DENSE, 497, 498
- RANK function, 62, 497–499, 586
- RANK SKIP, 497, 498
- RANKX, 23, 286, 432
- ratio-to-parent calculations, 669–674
- readability
  - of CALCULATE, 194
  - comments to enhance, 26–27
  - DAX syntax sugar for, 255–256
  - DIVIDE for, 72
  - of explicit code, 165
  - of HASONEVALUE, 123
  - of IFERROR, 76
  - importance of human, 45
  - of nested iteration, 187, 188
  - of SWITCH, 52
  - for temporary columns, 220
  - variable definition for, 692
  - variables to improve, 44, 193, 237, 251–252
  - writing equivalent alternatives for, 443
- reading Power BI queries, 405–415
- recursion, sideways, 633–634
- recursive definitions, 244
- recursive relationships, 679
- redundant data, 182
- refactoring, 377
- referencing columns, 66, 161–164, 172
- referencing tables, 25–26, 196, 212, 661, 715, 722
- related columns, 264
- RELATED function
  - in expanded tables, 715–720, 727
  - to navigate relationships, 60
  - when to use, 174–175
- RELATEDTABLE function
  - as alias for CALCULATETABLE, 172, 178
  - AVERAGEX iteration over result of, 404
  - to navigate relationships, 60
  - for nested row contexts on the same table, 189
  - relationship chains and, 177
  - time-delimited sales with, 180
  - when to use, 175
- relational functions, 60–62
- relationships
  - active and inactive, 6, 10, 294, 328–330, 738–739, 745–750
  - ambiguity and, 738–745
  - and the blank row, 208
  - calculated columns to define, 34
  - calculated table, 766
  - CALCULATE modifiers and, 294–297
  - with calculation groups, 642
  - chains of, 6, 176
  - Date table column category relationships, 534
  - disabling, 296, 361
  - expansion and table relationships, 261–262
  - filter propagation with, 88, 173
  - invalid, 210–212
  - joining tables with temporary, 435
  - limited, 208, 264, 315
  - many side of a relationship, 6, 7, 262, 263
  - multiple Date table, 530–531
  - one side of the relationship, 6, 7, 175, 208, 209, 214, 215, 261, 263, 726
  - and RELATED function use, 174–175
  - relationship handling, 15–16
  - role of, in the model, 158
  - row context use with, 173–178
  - self-referencing, 675, 679
  - in table expansion, 716, 717
  - table relationships, 5, 174, 764
  - temporary, 435
- REMOVEFILTERS
  - vs. ALLEXCEPT, 320
  - ALL function and, 203
  - ambiguity avoided via, 746
  - automatic Date table, 514–516, 521, 522, 540
  - as CALCULATE modifier, 94, 294, 306, 329–330
  - with expanded tables, 722, 724
  - with explicit CALCULATE, 280–281
  - ignoring outer filters with, 314
  - on multiple columns, 138
  - with no arguments, 141, 142
  - overview of, 104–106

## REMOVEFILTERS

REMOVEFILTERS (*continued*)

- percentage calculations with, 116, 135–141
- to remove cell filter context, 127
- SUMMARIZECOLUMNS and, 700, 760
- on table vs. table columns, 140

renaming columns, 216

reports

- and behavior of measures, 5
- dynamic, 342
- Power BI, 405
- unique DAX features for, 5
- visuals in, 577

Reset argument, 599–602

Result, 239

RETURN

- query variables and, 691
- variable definition after, 237, 239
- variable types in, 238

returning customers, computing number of, 151–156

ROLLUPADISSUBTOTAL, 407, 702, 703

ROLLUPADDSUBTOTAL, 702

ROLLUPGROUP, 702, 704

rounding numbers, 108

ROW, 448–449, 699–700

ROW CONTEXT, 285

row context

- active/inactive columns in, 281–283, 284
- aggregators inside, 164–165
- apply semantics* and ambiguous, 507–508
- calculated columns evaluated in, 278
- CALCULATE to create filter context from, 167, 171–172, 269–270, 272, 284
- and column reference vs. value, 162–164
- components of, 158
- context transition and, 270–271, 284
- EARLIER function to access outer, 192–193
- as evaluation context component, 84
- filter context vs., 157
- FILTER function operation in, 193
- formula using, 178–182
- inner as hiding outer, 161, 192, 281–282, 299
- iteration as, 158–159, 271

nested, 180, 187–193, 252, 281

overview of, 158–160, 270–271

RELATED function in, 728

SUMX for, 41

use with iterators, 160–161

use with relationships, 173–178

ROWNUMBER function, 62, 497–499

ROWPAGES, 580, 594–599

ROWS, 580, 594–599, 600, 617, 625

rows

calculated column rows, 34

column values to define, 82

context transition and duplicated, 274–278

duplicate source table rows, 499–504

filter context origination from, 85–86

rows axis, 81, 82

syntax for, 33

TOPN to extract rows from tables, 427–433

in visual shape, 577–583, 587

RUNNINGSUM, 576, 610–611

## S

Sales Amount measure, 85, 90, 125

same-distance-from-parent algorithm, 548–550

SAMEPERIODASLASTYEAR, 511, 536–542, 550–553, 560, 561

SAMPLE, 712–713

scalar expressions, 195, 248

scalar functions, 52, 53, 589

scalar values

returned by UDFs, 363

table-scalar conversion, 248

using tables as, 229–232

variables storing, 238, 247

SELECTCOLUMNS function

adding columns via, 394, 396

data lineage maintained by, 396

in like-for-like computations, 792, 795

overview of, 215–220, 394–397

SUMMARIZE vs., 395

as table function, 53

- in visual shape, 581
- SELECTEDMEASURE, 633, 635
- SELECTEDMEASURENAME, 639–640
- SELECTEDVALUE
  - checking columns for values with, 231
  - with context transition, 728
  - managing selections with, 337–340
  - retrieving slicer values with, 343
- self-referencing relationships, 675, 679
- semantic models
  - analyzing queries of, 406–415
  - avoiding code repetitions in, 69–70
  - calculation group precedence in, 639
  - components of, 158
  - data model vs., 2–3
  - date columns in, 509–510
  - measures as component of, 3
  - Power BI to create, 1, 2, 405
  - relationships in, 351
  - table relationships in, 15, 18
  - UDFs to share code across, 363
  - visual calculation as separate from, 573–574, 622
  - writing measures in, 313
- semi-additive calculations, 491, 562–566, 775–787
- set manipulation functions, 259, 415–426
- shadow filter contexts
  - ALLSELECTED use of, 729, 732–734
  - iteration creating, 732
  - overview of, 732–734
- sideways recursion, 633–634
- simple filters, 662–663
- single-line comments, 26–27
- slicers
  - at any granularity, 771
  - brand selection with, 118, 119
  - for calculation group items, 630, 644
  - creating, 120
  - errors introduced by, 130
  - filters created by, 88–89, 354, 358
  - information needed for, 762
  - reading queries of, 406–410
  - SELECTEDVALUE and VALUES for, 337–342
  - snowflake schema, 726
  - Sort by Column, 138
  - source tables
    - ALL to extend filter context of, 479
    - apply semantics* and, 473
    - bound* and *unbound* columns in, 474, 476
    - boundaries for, 482
    - created by OFFSET, 470, 471, 472, 478
    - duplicate rows in, 499–504
    - model and local columns in, 494
    - omitting the, 462–464, 465
    - WINDOW evaluation of, 489, 492, 493
  - speed. *See* performance
  - SQL
    - DAX vs., 11, 15–17
    - empty values in, 75
    - join functions in, 433
    - JOIN in, 18
    - queries, 5, 44
    - SELECT statement, 394
    - UNION ALL operation, 418
  - SQLBI, 801
  - star schemas, 9, 726
  - STARTOFMONTH function, 566–570
  - STARTOFQUARTER function, 566–570
  - STARTOFYEAR function, 566–570
  - static allocation, 763–766
  - static tables, 448–449
  - statistical functions, 56
  - STDEV, 40
  - strings
    - calculation items to change, 633, 634
    - case sensitivity of, 30
    - dynamic format strings, 615, 635
    - String data type, 30
    - string/number conversions, 28
    - transforming percentages to, 191–192
  - subcategory calculations, 206
  - sub-formula detection, 250
  - subqueries, 17
  - SUBSTITUTEWITHINDEX function, 415, 710–712

## subtotals

### subtotals

- cell filter context filtration of, 89, 90
- coordinates of, 83
- incorrect, in semi-additive calculations, 776
- SUMMARIZE for, 401, 700
- in visual shape, 577–578

### SUM function

- aggregating values with, 37, 48, 164
- behavior of, 40
- column names in, 13
- in context transition, 273, 276
- signature of, 164
- SUMX and, 43

### SUMMARIZECOLUMNS

- ADDCOLUMNS and, 229
- ADDMISSINGITEMS with, 708–709
- aggregation logic in, 18
- ALLSELECTED in, 730–732, 734
- arbitrarily shaped filters and, 669
- auto-exists in, 750–753
- best practices for, 759–760
- in expanded tables, 728
- filters and modifiers of, 700, 760
- filters as tables in, 268
- grouping via, 217, 224
- in like-for-like computations, 791, 792
- non-empty behavior of, 750, 751, 753–754
- overview of, 224–225, 397–402, 751
- queries using, 407, 700–705, 713
- vs. SUMMARIZE, 315–316, 402
- with TOPN, 428
- value filter behavior of, 751, 755–759
- in visual calculations, 579, 580, 583
- WINDOW function vs., 487

### SUMMARIZE function

- ADDCOLUMNS use with, 226–229, 400
- adding columns via, 399
- behavior of, 220–223
- clustering by, 399–400
- column grouping via, 185
- to correct itemization formula, 134–135
- distinct counts via, 221

- do not add columns via, 222–223
- in expanded tables, 727
- filter expression via, 314, 357
- FILTER to iterate, 265
- vs. GROUPBY, 402
- grouping via, 217, 220
- non-empty behavior of, 401
- within OFFSET, 470
- overview of, 397–402
- vs. SELECTCOLUMNS, 395
- semantics of, 401
- SUMMARIZECOLUMNS vs., 402
- as table function, 53
- in time intelligence calculations, 556

### SUMX

- arguments accepted by, 286
- and column values, 165
- in context transition, 273, 276, 298–299
- creation of row context, 159–161
- in filter context, 85
- first and second parameters of, 160–161
- iteration by, 41, 166, 171, 271, 285
- and iterator cardinality, 287

### SWITCH, 50, 51

### syntax

- column/measure distinctions via, 219–220
- overview of DAX syntax, 25–33
- string case sensitivity, 30
- syntactical errors, 185
- table syntax errors, 258
- user-defined function syntax, 364
- variable syntax, 237, 240
- visual calculation syntax, 576, 580
- See also specific function by name*

## T

table constructor, 423, 449, 450

### table filters

- vs. column filters, 720–722
- in measures, 722–726
- restricting calculations with, 792
- SUMMARIZECOLUMNS and, 760

## table functions

- ADDCOLUMNS, 218–220
- advanced calculations with, 199
- ALL\* functions as, 300–305
- ALL and ALLEXCEPT, 202–205
- ALLSELECTED, 232–235
- authoring queries with, 689, 713
- common factors via, 381
- DISTINCT, 207–215
- EVALUATE, 689
- FILTER, 199–202
- limiting use of, 447
- overview of, 52–55, 195–197
- parameter-passing modes and, 370
- SELECTCOLUMNS, 215–218
- SUMMARIZE, 220–223
- SUMMARIZECOLUMNS, 224–225
- vs. table references, 196
- time intelligence functions as, 519–522
- VALUES, 207–215

*See also specific function by name*

## tables

- as arguments, 374–375
- ascending/descending sort order, 429–432
- avoiding table names in front of measures, 172, 219
- blank row in, 208
- as CALCULATE filters, 255–260, 265
- calculating scalar values via, 195
- calculation groups as, 642
- cells vs., 11–13
- changing lineage of, 268–269
- and circular dependencies, 307–311
- column references vs. column values in, 162
- combining, 18, 418–423
- creating multi-row, 449
- in a data model, 5
- in DAX code, 11–15
- defining measures in, 40
- expanded version of, 261
- expressions, 247, 257–260, 287, 690
- filtering columns vs., 324–328
- filtering/grouping functions for, 387–405

- filter interactions among, 132
- filter propagation between, 88
- as filters, 94
- iteration over, 158, 161, 248, 285, 286
- joining two, 427, 437–440
- nested row contexts on different, 188–192
- nested row contexts on the same, 187–188
- over-denormalized, 182–187
- primary keys of, 312
- query tables, 695–696
- referencing, 212, 661, 715, 722
- referencing columns in, 25–26
- relationships, 5, 174
- relationships and expansion tables, 261–262
- removing filters from, 104
- returned by UDFs, 363
- ROW to create static tables, 448–449
- as scalar values, 229–232
- set functions to manipulate, 415–426
- syntax for anonymous tables, 33
- TOPN function to extract rows from, 427–433
- using REMOVEFILTERS on, 140
- using tables as filters, 440–447
- variables, 247–248
- virtual, for visual calculations, 65–66
- visuals, 81, 82
- See also* calculated tables, expanded tables, virtual tables

Tabular Editor, 48, 254, 311, 570

Tabular models

- DAX in, 1
- defining calendars in, 535
- Hide Members property in, 687
- implicit measures in, 36
- referencing data models as, 25
- and self-referencing relationships, 674

temporary columns, 219–220, 391, 399, 405

temporary relationships, 435

text functions, 56–57

*The Definitive Guide to DAX*, 715

theory, DAX, 22, 23, 24, 312

13-month calendar, 511, 519

ties, with INDEX, 457–459

## time-delimited sales computations

- time-delimited sales computations, 178–182, 443–445
  - time intelligence calculations
    - automatic REMOVEFILTERS in, 514–516, 521
    - calculation groups for, 628
    - calendar types, 509, 510–511
    - complex/hidden behavior in, 522
    - date columns in semantic model, 509–510
    - Date table building, 522–533
    - filter clearing in, 540–541
    - filter-keep columns in, 516–518
    - functions in, 59, 63–64, 509, 512–514, 518–522, 537–570
    - HASONEVALUE function in, 653
    - interval-to-date calculations, 557–560
    - lateral and hierarchical shifts in, 541–542
    - using calendars, 533–536
  - time intelligence functions
    - DATEADD, 542–553
    - FIRSTDATE, 562–566
    - for interval-to-date calculations, 557–560
    - LASTDATE, 562–566
    - mixing, 560–561
    - for opening/closing balances, 566–570
    - overview of, 63–64
    - PARALLELPERIOD and whole period functions, 553–557
    - SAMEPERIODASLASTYEAR, 511, 536–542, 550–553, 561
    - tailoring, 571
  - time-related columns, 518, 521, 536, 542, 550–553
  - TMDL (Tabular Model Definition Language), 535, 536, 570, 573
  - top category calculations, 206
  - TOPN function, 427–433, 434, 456, 457, 705–706
  - totals
    - ALL to compute, 234
    - cell filter context for grand total, 90
    - computing percentages of, 104, 138, 139, 234–235
    - contribution of each row to, 114–118
    - error-correction procedure in, 148–150, 344, 346
    - growth percentages of yearly totals, 142–150
    - learning from errors in, 145
    - in like-for-like computations, 789
    - for over-denormalized tables, 182–187
    - Power BI computation of, 106–110
    - in semi-additive calculations, 775
    - in time intelligence calculations, 512
    - in visual calculations, 604
    - in visual shape, 577–578
  - TOTALYTD, 512, 557
  - TREATAS
    - filter creation by, 268–269, 408
    - filtering individual values with, 410
    - as replacing INTERSECT, 423
    - to restore data lineage, 422–423, 440
    - in semi-additive calculations, 787
  - trigonometric functions, 56
  - tuple syntax, 33
  - type-handling system, 27
- ## U
- UDFs (user-defined functions)
    - avoiding duplication with, 797
    - calculation groups and, 627
    - dynamic binding of columns, 374–376
    - examples of using, 380–386
    - Expression and Value parameters, 161, 364, 367–374
    - in like-for-like computations, 797, 799, 801
    - model-dependent/-independent, 376–377
    - naming conventions for, 377–379
    - organization via, 363
    - overview of, 68–70, 363–366
    - parameter-passing modes, 363–374, 386
    - syntax for, 364
  - unbound columns*, 471, 474, 476, 489
  - unidirectional relationships
    - vs. bidirectional cross filters, 7–8
    - options with, 262
    - USERRELATIONSHIP to change, 297
  - UNION, 314, 361, 418–423

## USERRELATIONSHIP

- ambiguity and use of, 745
- behavior of, 294–296
- as CALCULATE modifier, 94, 294, 306, 328–330
- in expanded tables, 719

**V**

VAL parameter (UDF), 364, 367

value filter behavior, 402, 660, 669, 688, 751, 755–759, 760

value parameter, 68, 69, 161, 285, 364, 367, 371, 632

## values

- blank row and invalid, 208, 212
- blank vs. incorrect, 651
- calculated table, 197
- column reference vs. column value, 161–164
- of columns/measures, 80
- cross-filtering and invisible, 133
- CROSSJOIN for nonexistent, 418
- empty/missing value errors, 73–75
- functions to aggregate, 40
- hidden, 655, 658
- hiding rows by blanking, 679, 682, 684
- implicit measures to compute, 36
- INDEX sorting of blank values, 460
- invalid values, 314–315
- ISEMPTY to test for no, 660
- measures to aggregate, 37–39
- returned by ALL vs. VALUES, 207
- row-by-row aggregation of, 34, 36
- row context to evaluate column values, 158, 160
- scalar, 195, 229–232
- IN to search for, 447
- single column values, 123
- stored by variables, 238
- SUMMARIZE for retrieving groups of, 397–398
- visual computation of, 64

## VALUES function

- as alternative to KEEPFILTERS, 101–103
- apply semantics* evaluation of, 494
- behavior of, 101

vs. DISTINCT, 210

effect on formula semantics, 214

filter creation by, 103, 320

FILTERS vs., 658–660

managing selections with, 340–342

overview of, 207–215

as parameter of iterators, 212, 214

scalar value computation with, 229

table evaluation with, 116, 120, 302

as table function, 53, 54, 320

value retrieval with, 207

in visual calculations, 617

VAR block, 237–239

## variables

- as constant values, 238, 241–243
- debugging via, 131–135
- defining, 44, 243–245, 248, 691–694
- defining multiple, 237
- documentation via, 251–252
- evaluation of, 249–251
- expression vs. query, 691–694
- joining tables of, 437
- measures vs., 242
- in multi-step measures, 125–126, 130
- naming, 239–241
- for nested row contexts, 190–191, 252
- in parent/child hierarchies, 687
- in Power BI queries, 407
- readability via, 46
- as replacing EARLIER function, 193
- scope of, 243–247
- storing scalar values, 237
- syntax for, 237, 240
- table expansion variables, 716
- table variables, 247–248
- UDFs for global variables, 384–386

Variant data type, 30

VAR keyword, 44

VAR/RETURN blocks, 237, 238, 244, 247

VAR/RETURN statements, 243, 244, 245, 692

VAR statements, 243, 692

## virtual tables

### virtual tables

- calculation group effects on, 649
- densification and additions to, 585–586
- functions for, 605–615
- functions to move visual context in, 589
- navigating the lattice of, 592–593, 620, 623
- placing filters on, 602
- ROWS/COLUMNS/ROWPAGES to access, 594–596
- visual calculation references to, 573–576, 620
- visual calculations as new columns in, 583–584
- visual context to filter, 587–589
- visual shape of, 577–583

### visual calculation context

- collapsing/expanding, 589–591
- parent/child levels of, 591, 593, 598, 599, 600, 602, 624, 625
- for visual calculations, 587

### visual calculations

- CALCULATE in, 602–605
- and calculation groups, 649–650
- COLLAPSE/COLLAPSEALL, 589–591
- densification and, 585–586
- direction in, 600
- examples of, 615–625
- EXPAND/EXPANDALL, 589–591
- filters in, 602
- functions for, 605–615
- lack of data model access for, 573–574, 621, 623
- limitations of, 620, 622, 623
- measures and, 141, 574
- navigating the table lattice, 592–593
- as new virtual table columns, 583–584
- for non-additive calculations, 109–110
- overview of, 64–67, 573–576
- Reset argument, 599–602
- ROWS/COLUMNS/ROWPAGES for row access in, 594–599
- syntax for, 576, 580
- tracking changes in, 573
- visual context for. *See* visual calculation context
- visual shape in, 577–583, 698
- and window functions, 576–577

### visual filter context

- ALLSELECTED and, 232–235
- vs. visual calculations context, 157

### visuals

- axes and coordinates in, 81–84
- filters created by, 84, 88–90
- reading queries of, 406–415
- SUMMARIZECOLUMNS to create, 224
- volume of sales, computing highest, 226–229

## W

weekly calendars, 510, 528–529

what-if analysis, 452

### WINDOW function

- and *apply semantics*, 482, 487–494
- current row* and, 482–484
- overview of, 482–487
- used in measures, 486
- visual calculations and, 577

### window functions

- apply semantics* implemented by, 454
- common errors with, 499–508
- complexity of, 508
- INDEX, 454–464
- OFFSET, 464–482
- optional arguments in, 454
- order as irrelevant in, 454
- overview of, 62–63
- purpose of, 453, 508
- as replacing TOPN, 427
- source table omission in, 462–464
- visual calculations and, 576–577
- WINDOW, 482–487

Wrong Amt, 274, 275, 277

## Y

year-to-date calculation, 512