Microsoft

Microsoft Excel 2013

Building Data Models with PowerPivot

Alberto Ferrari and Marco Russo

Sample files on the web

Microsoft

Microsoft Excel 2013: Building Data Models with PowerPivot

Alberto Ferrari and Marco Russo

ISBN: 978-0-7356-7634-3

Third Printing: December 2014

Printed and bound in the United States of America.

Acquisitions and Developmental Editor: Kenyon Brown

Production Editor: Christopher Hearse

Editorial Production: S4 Carlisle Publishing Services

Technical Reviewer: Javier Guillen

Indexer: Ellen Troutman Zaig

Cover Design: Twist Creative • Seattle

Cover Composition: Zyg Group, LLC

Illustrator: S4 Carlisle Publishing Services

Contents at a Glance

	Introduction	*xiii*
CHAPTER 1	Introduction to PowerPivot	1
CHAPTER 2	Using the unique features of PowerPivot	31
CHAPTER 3	Introducing DAX	49
CHAPTER 4	Understanding data models	83
CHAPTER 5	Publishing to SharePoint	117
CHAPTER 6	Loading data	133
CHAPTER 7	Understanding evaluation contexts	179
CHAPTER 8	Understanding *CALCULATE*	209
CHAPTER 9	Using hierarchies	233
CHAPTER 10	Using Power View	261
CHAPTER 11	Shaping the reports	289
CHAPTER 12	Performing date calculations in DAX	323
CHAPTER 13	Using advanced DAX	369
CHAPTER 14	Using DAX as a query language	417
CHAPTER 15	Automating operations using VBA	445
CHAPTER 16	Comparing Excel and SQL Server Analysis Services	463
	Index	*473*

Contents

Introduction . . . . *xiii*

Chapter 1 Introduction to PowerPivot 1

Using a PivotTable on an Excel table. 2

Using PowerPivot in Microsoft Office 2013 5

Adding information to the Excel table. 6

Creating a data model with many tables. 8

Understanding relationships 11

Understanding the data model 13

Querying the data model. 14

The PowerPivot add-In. 15

Using OLAP tools and converting to formulas 18

Understanding PowerPivot for Excel 2013. 23

Creating a Power View report. 25

Chapter 2 Using the unique features of PowerPivot 31

Loading data from external sources 31

Creating a PowerPivot PivotTable. 35

Using the DAX language. 37

Creating a calculated column 37

Creating a calculated field 41

Computing complex aggregations like Distinct Count. 44

Refreshing the PowerPivot data model. 48

Chapter 3 Introducing DAX 49

Understanding DAX calculations 49

DAX syntax 49

DAX data types. 50

DAX operators 52

DAX values 53

Understanding calculated columns and fields. 54

Calculated columns 54

Calculated fields 55

Handling errors in DAX expressions 60

Conversion errors. 60

Arithmetical operations 61

Intercepting errors. 63

Formatting DAX code 65

Common DAX functions 68

Aggregate functions 68

Logical functions 71

Information functions 72

Mathematical functions 74

Text functions 74

Conversion functions. 75

Date and time functions 76

Relational functions. 77

Using basic DAX functions 78

Chapter 4 Understanding data models 83

Understanding the basics of data modeling 83

Producing a report without a data model. 84

Building a data model. 88

More about relationships 89

Understanding normalization and denormalization 91

Denormalizing within SQL queries. 95

The PowerPivot query designer. 95

When to denormalize tables 102

Understanding over-denormalization. 104

Understanding OLTP and data marts 106

Querying the OLTP database. 107

Data marts, facts, and dimensions 110

Star schemas 111

Which database is the best to query? 112

Using advanced relationships. 112

Chapter 5 Publishing to SharePoint 117

SharePoint 2013 and PowerPivot integration 117

Licensing and setup 118

Publishing a workbook to SharePoint. 118

Using the PowerPivot Gallery 123

Connecting Excel to a SharePoint Excel data model 125

Creating a Power View report. 127

Managing the PowerPivot data refresh 129

Chapter 6 Loading data 133

Understanding data sources. 133

Loading from a database. 135

Loading from a list of tables 137

Loading relationships 139

Selecting related tables. 140

Loading from a SQL query 141

Loading from views 142

Opening existing connections 143

Loading from Access 144

Loading from SQL Server Analysis Services 146

Using the MDX editor 148

Handling of keys in the OLAP cube 150

Loading from a tabular database 152

Loading from SharePoint. 153

Using linked tables 157

Loading from Excel files. 160

Loading from text files. 163

Loading from the Clipboard 165

Loading from a report 167

Loading from a data feed 171

Loading from Windows Azure Marketplace. 173

  Suggest related data 176

Refreshing connections 176

Chapter 7 Understanding evaluation contexts 179

Introduction to evaluation contexts 180

  Understanding the row context 184

Testing your evaluation context understanding .185

  Using *SUM* in a calculated column 185

  Using fields in a calculated field 186

Creating a row context with iterators 189

Understanding *FILTER*, *ALL*, and context interactions 192

Working with many tables. 195

  Row contexts and relationships. 196

  Filter context and relationships 198

  Introducing *VALUES*. 199

  Introducing *ISFILTERED* and *ISCROSSFILTERED*. 201

Evaluation contexts recap 205

Creating a parameter table. 206

Chapter 8 Understanding CALCULATE 209

Why is *CALCULATE* needed?. 209

CALCULATE examples 213

  Filtering a single column. 213

  Filtering with complex conditions. 218

Using *CALCULATE* inside a row context 222

Understanding circular dependencies. 225

CALCULATE rules. 229

Understanding *ALLSELECTED* 230

Chapter 9 Using hierarchies 233

Understanding hierarchies 233

When to build hierarchies 235

Building hierarchies 235

Creating hierarchies on multiple tables 236

Performing calculations using hierarchies. 238

Using parent/child hierarchies 248

Chapter 10 Using Power View 261

What is Power View? 261

Power View basics. 262

Using the Filters pane 265

Decorating your report. 267

Understanding table, matrix, and cards 268

Using the matrix visualization 269

Using the card visualization 270

Using a table as a slicer. 272

Using charts 272

Using the line chart 273

Using the pie chart. 274

Using the scatter chart 275

Using maps. 277

Understanding drill-down. 281

Using tiles 283

Understanding multipliers. 286

Using Power View effectively 287

Chapter 11 Shaping the reports 289

Key Performance Indicators (KPIs) 289

Creating data models for Power View. 299

Understanding Power View metadata 304

Using Summarize By 305

Using the default field set. 308

Using the Table Behavior dialog box 309

Defining sets 310

Creating dynamic sets with MDX 314

Using perspectives 318

Understanding drill-through. 322

Chapter 12 Performing date calculations in DAX 323

Building a calendar table. 323

Working with multiple calendar tables 326

Calculating working days 329

Aggregating and comparing over time 339

Year-to-Date (YTD), Quarter-to-Date (QTD), and
Month-to-Date (MTD). 339

Time intelligence with *CALCULATE*. 343

Computing periods from the prior year (PY) 346

Computing the moving annual total 348

Using other aggregation functions. 349

Computing difference over a previous year .351

Closing balance over time 353

Semiadditive measures 353

OPENINGBALANCE and *CLOSINGBALANCE* functions. 356

Updating balances by using transactions 360

Computing moving averages 363

Chapter 13 Using advanced DAX 369

Banding. 369

Ranking 374

Using many-to-many relationships 382

Computing new and returning customers 386

Understanding *KEEPFILTERS* 390

Implementing basket analysis. 400

Understanding the power of calculated columns: ABC analysis. 407

Handling currency conversion 411

Chapter 14 Using DAX as a query language 417

Understanding *EVALUATE* 417

Creating an Excel table with *EVALUATE* 418

Using common functions in queries 423

Using *FILTER* 423

Using *CALCULATETABLE* 424

Using *ADDCOLUMNS*. 425

Using *VALUES* with *ADDCOLUMNS* 426

Using SUMMARIZE 427

Using the *ROLLUP* option. 430

Linking back a DAX query 431

Computing ABC analysis with a linked-back table 435

Using *CROSSJOIN* 439

Using *GENERATE* 440

Querying with DAX Studio 443

Chapter 15 Automating operations using VBA 445

Enabling the DEVELOPER tab of the ribbon. 445

Updating a linked-back DAX query through VBA. 451

Using the *Model* object 455

Importing data into the data model using VBA 456

Understanding data connections. 459

Chapter 16 Comparing Excel and SQL Server Analysis Services 463

Understanding the different versions of the engine 463

Feature matrix 464

Securing your data 465

Programmability and flexibility 466

Translations 468

Database size 469

Number of databases. 471

PowerPivot as a prototyping system. 472

Index 473

About the Authors 489

Introduction

Microsoft Excel is the world standard for performing data analysis. Its ease of use and power make the Excel spreadsheet the tool that everybody uses, regardless of the kind of information being analyzed.

You can use Excel to store your personal expenses, your current account information, your customer information or a complex business plan, or even your weight-loss progress during a hard-to-follow diet. The possibilities are infinite—we are not even going to try to start enumerating all the kind of information you can analyze with Excel. The fact is that if you have some data to arrange and analyze, your chances are excellent that Excel will be the perfect tool to use. You can easily arrange data in a tabular format, update it, generate charts, PivotTables, and calculations based on it, and make forecasts with relatively limited knowledge of the software. With the advent of the cloud, now you can use Excel on mobile devices like tablets and smart phones, too, using Internet to have constant access to your information. Also, in earlier versions of Excel, there was a limit of 65,536 rows per single worksheet, and the fact that so many customers asked Microsoft to increase this number (which Microsoft did, raising the limit to 1 million rows in Excel 2007) is a clear indication that users want Excel to store and analyze large amounts of data.

Besides Excel users, there is another category of people dedicating their professional lives to data analysis: business intelligence (BI) professionals. BI is the science of getting insights from large amounts of information, and, in recent years, BI professionals have learned and created many new techniques and tools to manage systems that can handle the range of hundreds of millions or even billions of rows. BI systems require the effort of many professionals and expensive hardware to run. They are powerful, but they are expensive and slow to build, which are serious disadvantages.

Before 2010, there was a clear separation between the analysis of small and large amounts of data: Excel on one side and complex BI systems on the other. A first step in the direction of merging the two worlds was already present in Excel because the PivotTable tool had the ability to query BI systems. By doing that, data analysts could query large BI systems and get the best of both worlds because the result of such a query can be put into an Excel PivotTable, and thus they could use it to perform further analysis.

In 2010, Microsoft made a strong move to break down the wall between BI professionals and Excel users by introducing xVelocity, a powerful engine that drives large BI solutions directly inside Excel. That happened when Microsoft SQL Server 2008

R2 PowerPivot for Excel was released as a free add-in to Excel 2010. The goal was to make the creation of BI solutions so easy that Excel would start to be not only a BI client, but also a BI server, capable of hosting complex BI solutions on a notebook. They called it *self-service BI*.

Microsoft PowerPivot has no limits on the number of rows it can store: if you need to handle 100 million rows, you can safely do so, and the speed of analysis is amazing. PowerPivot also introduced the DAX language, a powerful programming language aimed to create BI solutions, not only Excel formulas. Finally, PowerPivot is able to compress data in such a way that large amounts of information can be stored in relatively small workbooks. But this was only the first step.

The second definitive step to bring the power of BI to users was the introduction of Excel 2013. PowerPivot is no longer a separate add-in of Excel; now it is an inherent part of the Excel technology and brings the power of the xVelocity engine to every Excel user. The era of self-service BI started in 2010, and it has advanced in 2013.

Because you are reading this introduction, you are probably interested in joining the self-service BI wave, and you want to learn how to master PowerPivot for Excel. You will need to learn the basics of the tool, but this is only the first step. Then, you will need to learn how to shape your data so that you can execute analysis efficiently: we call this *data modeling*. Finally, you will need to learn the DAX language and master all its concepts so you can get the best out of it. If that is what you want, then this is the book for you.

We are BI professionals, and we know from experience that building a BI solution is not easy. We do not want to mislead you: BI is a fascinating technology, but it is also a hard one. This book is designed to help you take the necessary steps to transform you from an Excel user to a self-service BI modeler. It will be a long road that will require time and dedication to travel, and you will find yourself making the adaptations you need to learn new techniques. However, the results you will be able to accomplish are invaluable.

The book is not a step-by-step guide to PowerPivot for Excel 2013. If you are looking for a *PowerPivot for Dummies* book, then this is not the book for you. But if you want a book that will go with you on this long, satisfying journey, from the first simple workbooks to the complex simulations you will be creating soon, then this is your ultimate resource.

When writing this book, we decided to focus on concepts and real-world examples, starting at zero and bringing you to mastering the DAX language. We do not cover every single feature, and we do not explain each operation in a "Click this, and then do that" fashion. On the other hand, we packed in this single book a huge amount of information so that, once you finished studying the book, you will have a great background in the new modeling options of Excel.

This last sentence highlights the main characteristic of this book: it is a book to study, not just to read. Get prepared for a long trip—but we promise you that it will be well worth it.

Note The PowerPivot and Power View features are included only with specific configurations of Office 2013. The PowerPivot feature, which was available in all versions of Excel 2010, is available only in Office 2013 Professional Plus, SharePoint 2013 Enterprise Edition, SharePoint Online 2013 Plan 2, and the E3 or E4 editions of Office 365. The Power View feature, new in Excel 2013, is included with the same versions as PowerPivot. Fortunately, the Excel Data Model is supported in all configurations of Excel 2013. Be aware, however, that the variety of available configurations may change.

Who this book is for

The book is aimed at Excel users, project managers, and decision makers who wish to learn the basics of PowerPivot for Excel 2013, master the new DAX language that is used by PowerPivot, and learn advanced data modeling and programming techniques with PowerPivot.

Assumptions about you

This book assumes that you have a basic knowledge of Excel 2010 or Excel 2013. You do not need to be a master of Excel; just being a regular user is fine. We will cover what is needed to make the transition from Excel to PowerPivot, but we do not cover in any way the fundamentals of Excel, like entering a formula, writing a *VLOOKUP* function, or other basic functionalities.

No previous knowledge of PowerPivot is needed. If you already tried to build a data model by yourself, that is fine, but we will assume that you never opened PowerPivot before reading the book.

Organization of this book

The book is designed to be read from cover to cover. Trying to jump directly to the solution of a specific problem, skipping some content, will probably be the wrong choice. In each chapter, we introduce concepts and functionalities that you will need to understand the subsequent chapters. Moreover, we wrote some chapters knowing that you will need to read them more than once, because the theoretical background they provide is hard to take in at a first read.

The book is divided into 16 chapters:

Chapter 1, "Introduction to PowerPivot," offers a guided tour of the basic features of PowerPivot for Excel 2013. By following a step-by-step guide, we show the main benefits of using PowerPivot for your analytical needs. We show how to create a simple Power View report as well.

Chapter 2, "Using the unique features of PowerPivot," shows the features that are available only if you enable the PowerPivot for Excel add-in. This includes calculated columns, calculated fields, hierarchies, and some other basic features. It is the logical continuation (and conclusion) of Chapter 1.

In Chapter 3, "Introducing DAX," we start covering the DAX language, including its syntax and the most basic functions. We highlight the difference between a calculated column and a calculated field, and at the end, we show a first practical example of DAX usage.

Chapter 4, "Understanding data models," is a theoretical chapter, covering the basics of data modeling and showing the different modeling options in a Power-Pivot database. We describe several concepts that are not evident for Excel users, like normalization and denormalization, the structure of a SQL query, how relationships work and why they are so important, the structure of data marts, and data warehouses.

In Chapter 5, "Publishing to SharePoint," we cover the process of publishing workbooks to Microsoft SharePoint to do team BI. Moreover, we introduce the concept of PowerPivot for SharePoint being a server-side application that you can program and extend using Excel and PowerPivot.

Chapter 6, "Loading data," is dedicated to the many ways to load data inside PowerPivot. For each data source, we show the way it works and provide many hints and best practices for that specific source.

Chapter 7, "Understanding evaluation contexts," and Chapter 8, "Understanding CALCULATE," are the theoretical core of the book. There, we introduce the concepts of evaluation contexts, relationships, and the *CALCULATE* function. These are the pillars of the DAX language, and you will need to master them before writing advanced data models with PowerPivot.

Chapter 9, "Using hierarchies," shows how to create and manage hierarchies. It covers basic hierarchy handling, how to compute values over hierarchies, and finally, it shows how to manage parent/child hierarchies by using the concepts learned in Chapters 7 and 8.

Chapter 10, "Using Power View," is dedicated to the new reporting tool in Excel 2013: Power View. There, we show the main feature of this tool, how to create simple Power View reports, and how to filter data and build reports that are pleasant to look at and provide useful insights in your data.

Chapter 11, "Shaping the reports," covers several advanced topics regarding reporting. It includes Key Performance Indicators (KPIs), how to write them, and how to use them to improve the quality of your reporting system. We also cover the Power View metadata layer in PowerPivot, drill-through, sets in Excel or in MDX, and perspectives.

Chapter 12, "Performing date calculations in DAX," deals with time intelligence. Year to Date (YTD), Quarter to Date (QTD), Month to Date (MTD), working days versus non-working days, semiadditive measures, moving averages, and other complex calculations involving time are all topics covered here.

Chapter 13, "Using advanced DAX," is a collection of scenarios and solutions, all of which share the same background: they are hard to solve using Excel or in any other tool, whereas they are somewhat easier to manage in DAX, once you gain the necessary knowledge from the previous chapters in the book. All these examples come from real-world scenarios and are among the top requests we see when we do consultancy or look at forums on the web.

Chapter 14, "Using DAX as a query language," is dedicated to using DAX as a query language (as you might guess). It covers the various functionalities of DAX when used to query a database. It also shows advanced functionalities, like reverse-linked and linked-back tables, which greatly enhance the capabilities of PowerPivot to build complex data models.

Chapter 15, "Automating operations using VBA," discusses using Microsoft Visual Basic for Applications (VBA) to manage PowerPivot workbooks in a programmatic way, automating a few common tasks. We provide some code examples and show how to solve some of the common scenarios where VBA might be useful.

Chapter 16, "Comparing Excel and SQL Server Analysis Services," compares the functionalities of the three flavors of PowerPivot technology: PowerPivot for Excel, PowerPivot for SharePoint, and SQL Server Analysis Services (SSAS). The goal of this final chapter is to give you a clear picture of what can be done with PowerPivot for Excel, when you need to move a step further and adopt PowerPivot for SharePoint, and what extra features are available only in SSAS.

Conventions

The following conventions are used in this book:

- **Boldface** type is used to indicate text that you type.
- *Italic* type is used to indicate new terms, calculated fields and columns, and database names.
- The first letters of the names of dialog boxes, dialog box elements, and commands are capitalized. For example, the Save As dialog box.
- The names of ribbon tabs are given in ALL CAPS.
- Keyboard shortcuts are indicated by a plus sign (+) separating the key names. For example, Ctrl+Alt+Delete mean that you press Ctrl, Alt, and Delete keys at the same time.

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:

http://www.microsoftpressstore.com/title/9780735676343

The companion content includes the following:

- A Microsoft Access version of the *AdventureWorksDW* databases that you can use to build the examples yourself.
- All the Excel workbooks that are referenced in the text (that is, all the workbooks that are used to illustrate the concepts). Note you need to have Excel 2013 to open the workbooks.

Acknowledgments

We have so many people to thank for this book that we know it is impossible to write a complete list. So thank you so much to all of you who contributed to this book—even if you had no idea that you were doing it. Blog comments, forum posts, email discussions, chats with attendees and speakers at technical conferences, and so much more have been useful to us, and many people have contributed significant ideas to this book. That said, there are people we need to cite personally here because of their particular contributions.

We want to start with Edward Melomed: he inspired us, and we probably would not have started our journey with PowerPivot without a passionate discussion that we had with him several years ago.

We have to thank Microsoft Press, O'Reilly Media, and the people who contributed to the project: Kenyon Brown, Christopher Hearse, and many others behind the scenes.

The only job longer than writing a book is the studying you must do in preparation for writing it. A group of people that we (in all friendliness) call "ssas-insiders" helped us get ready to write this book. A few people from Microsoft deserve a special mention as well because they spent precious time teaching us important concepts about PowerPivot and DAX. Their names are Marius Dumitru, Jeffrey Wang, and Akshai Mirchandani. Your help has been priceless, guys!

We also want to thank Amir Netz, Ashvini Sharma, and T. K. Anand for their contributions to the discussion about how to position PowerPivot. We feel they helped us in some strategic choices we made in this book.

Finishing a book in the age of the Internet is challenging because there is a continuous source of new inputs and ideas. A few blogs have been particularly important to our book, and we want to mention their creators here: Chris Webb, Kasper de Jonge, Rob Collie, Denny Lee, and Dave Wickert.

Finally, a special mention goes to the technical reviewer, Javier Guillen. He double-checked all the content of our original text, searching for errors and giving us invaluable suggestions on how to improve the book. If the book contains fewer errors than our original manuscript, it is because of Javier. If it still contains errors, it is our fault, of course.

Thank you so much, folks!

Support and feedback

The following sections provide information on errata, book support, feedback, and contact information.

Errata

We have made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site:

http://aka.ms/Excel2013DataModelsPP/errata

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at mspinput@microsoft.com.

Note that product support for Microsoft software is not offered through these addresses.

We Want to Hear from You

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at

http://www.microsoft.com/learning/booksurvey

The survey is short, and we will read every one of your comments and ideas. Thanks in advance for your input!

Stay in Touch

Let’s keep the conversation going! We are on Twitter: *http://twitter.com/MicrosoftPress*.

CHAPTER 3

Introducing DAX

Now that you have seen some of the features of Microsoft PowerPivot for Microsoft Excel 2013, it is time to learn the fundamentals of the DAX language. PowerPivot has its own syntax for defining calculation expressions. It is conceptually similar to an Excel expression, but it has specific functions that allow you to create more advanced calculations on data stored in multiple tables. (The PowerPivot language is called the Data Analysis eXpressions language, but we always use the shorter DAX acronym.)

In this chapter, you will learn the basics of DAX and how to use it to solve some typical problems in business scenarios.

Understanding DAX calculations

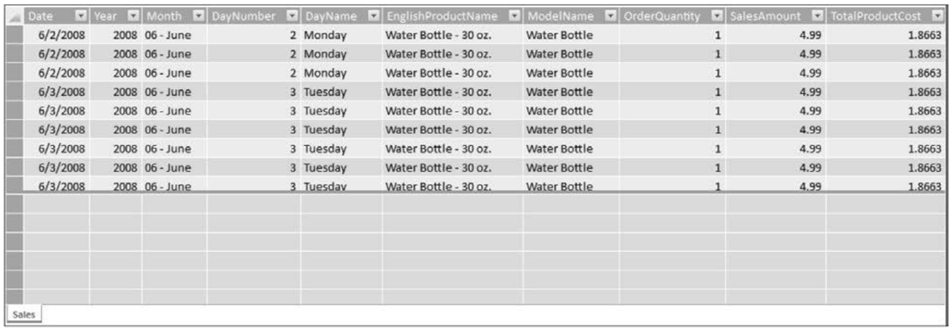
Just like Excel, any calculation in DAX begins with the assignment operator. The main difference is that DAX never uses cell coordinates like A1, C2, and so on. In DAX, you always specify coordinates using column and table names. Moreover, DAX does not support the concept of range as Excel does: to use DAX efficiently, you need to learn to work with columns and tables.

Note In a DAX expression, you can get the value of a column for only a single row or for the whole table—that is, you cannot get access to a specific row inside a table. To get a range, you need to use DAX functions that filter a table, thus returning a subset of the rows of the original table that corresponds to the needed range.

To express complex formulas, you need to learn the basics of DAX, which includes the syntax, the different data types that DAX can handle, the basic operators, and how to refer to columns and tables. In the next few sections, we are going to introduce these concepts.

DAX syntax

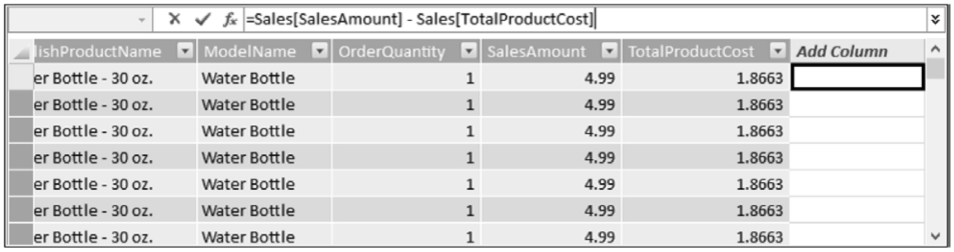
A relatively simple way to understand how DAX syntax works is to start with an example. Suppose that you have a PowerPivot table like the one shown in Figure 3-1. Two columns, SalesAmount and TotalProductCost, are useful in this particular instance. You can find this data model in the companion workbook “CH03-01-SalesExample.xlsx.”



Date	Year	Month	DayNumber	DayName	EnglishProductName	ModelName	OrderQuantity	SalesAmount	TotalProductCost
6/2/2008	2008	06 - June	2	Monday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
6/2/2008	2008	06 - June	2	Monday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
6/2/2008	2008	06 - June	2	Monday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
6/3/2008	2008	06 - June	3	Tuesday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
6/3/2008	2008	06 - June	3	Tuesday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
6/3/2008	2008	06 - June	3	Tuesday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
6/3/2008	2008	06 - June	3	Tuesday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
6/3/2008	2008	06 - June	3	Tuesday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
6/3/2008	2008	06 - June	3	Tuesday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663

FIGURE 3-1 We will use this table to demonstrate DAX syntax.

Using this data, you now want to calculate the margin by subtracting the TotalProductCost from the SalesAmount. To do that, you need to write the DAX formula shown in Figure 3-2 in a new column, which you can call GrossMargin.



```
=Sales[SalesAmount] - Sales[TotalProductCost]
```

lishProductName	ModelName	OrderQuantity	SalesAmount	TotalProductCost	Add Column
er Bottle - 30 oz.	Water Bottle	1	4.99	1.8663	
er Bottle - 30 oz.	Water Bottle	1	4.99	1.8663	
er Bottle - 30 oz.	Water Bottle	1	4.99	1.8663	
er Bottle - 30 oz.	Water Bottle	1	4.99	1.8663	
er Bottle - 30 oz.	Water Bottle	1	4.99	1.8663	
er Bottle - 30 oz.	Water Bottle	1	4.99	1.8663	
er Bottle - 30 oz.	Water Bottle	1	4.99	1.8663	

FIGURE 3-2 You can enter the definition of GrossMargin into the formula bar.

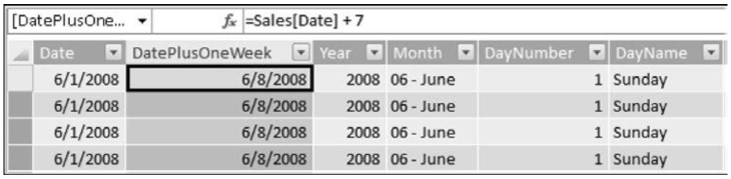
This new formula is repeated automatically for all the rows of the table, resulting in a new column in the table. In this example, you are using a DAX expression to define a calculated column. (Later in this chapter, we will show that DAX is used also to define calculated fields.) This DAX expression handles numeric values and returns a numeric value, too.

DAX data types

You saw in the previous formula how to handle numeric values. DAX can perform computations with seven numeric types:

- *Integer*
- *Real*
- *Currency*
- *Date* (*datetime*)
- *TRUE/FALSE* (Boolean)
- *String*
- *BLOB* (which stands for "binary large object")

PowerPivot has a powerful type-handling system, so you do not have to worry much about data types: when you write a DAX expression, the resulting type is based on the type of the terms used in the expression. You need to be aware of this in case the type returned from a DAX expression is not the expected one; then you must investigate the data type of the terms used in the expression itself. For example, if one of the terms of a sum is a date, the result is also a date; if the same operator is used with integers, the result is an integer. This is known as *operator overloading,* and you can see an example of its behavior in Figure 3-3, where the DatePlusOneWeek column is calculated by adding 7 to the value in the Date column. The result: a date.



```
=Sales[Date] + 7
```

Date	DatePlusOneWeek	Year	Month	DayNumber	DayName
6/1/2008	6/8/2008	2008	06 - June	1	Sunday
6/1/2008	6/8/2008	2008	06 - June	1	Sunday
6/1/2008	6/8/2008	2008	06 - June	1	Sunday
6/1/2008	6/8/2008	2008	06 - June	1	Sunday

FIGURE 3-3 Adding an integer to a date results in a date being increased by the corresponding number of days.

In addition to operator overloading, PowerPivot automatically converts strings into numbers and numbers into strings whenever the operator requires it. For example, if you use the *&* operator, which concatenates strings, PowerPivot automatically converts its arguments into strings. If you look at the formula:

```
= 5 & 4
```

it returns a "54" string result. On the other hand, the formula:

```
= "5" + "4"
```

returns an integer result with the value of 9.

As you have seen, the resulting value depends on the operator and not on the source columns, which are converted following the requirements of the operator. But even if this behavior is convenient, in the "Handling errors in DAX expressions" section later in this chapter, you will see what types of errors might happen during these automatic conversions.

Inside DAX data types

DAX data types might be familiar to people used to working with Excel. However, we need to review a few considerations about two of these data types because of how often they are used in PowerPivot data models.

Date data type

PowerPivot stores dates in a *datetime* data type. This format uses a floating-point number internally, wherein the integer corresponds to the number of days (starting from December 30, 1899) and the decimal identifies the fraction of the day. (Hours, minutes, and seconds are converted to decimal fractions of a day.) Thus, the expression:

```
= NOW () + 1
```

increases a date by one day (exactly 24 hours), returning the date of tomorrow at the same hour/minute/second of the execution of the expression itself.

TRUE/FALSE data type

The data type *TRUE/FALSE* is used to express logical conditions. For example, a calculated column defined by the following expression is of type *TRUE/FALSE:*

```
= Sales[TotalProductCost] > Sales[Amount]
```

The behavior of this data type is similar to the behavior of data types in Excel, and it is usually called a *Boolean data type*. Usually a column of this type is not made visible to the user but is used internally for DAX calculations.

DAX operators

A list of the operators available in DAX is shown in Table 3-1.

Moreover, the logical operators are available also as DAX functions, with syntax very similar to Excel syntax. For example, you can write:

```
AND ([Country] = "USA", [Quantity] > 0)
OR ([Country] = "USA", [Quantity] > 0)
NOT ([Country] = "USA")
```

that corresponds, respectively, to:

```
[Country] = "USA" && [Quantity] > 0
[Country] = "USA" || [Quantity] > 0
! ([Country] = "USA")
```

TABLE 3-1 Operators

Operator type	Symbol	Use	Example
Parenthesis	*( )*	Precedence order and grouping of arguments	`(5 + 2) * 3`

Arithmetic	+
	-
	*
	/
		Addition
		Subtraction/negation
		Multiplication
		Division
			`4 + 2`
			`5 – 3`
			`4 * 2`
			`4 / 2`

Comparison	= <> > >= < <=	Equal to Not equal to Greater than Greater than or equal to Less than Less than or equal to	[Country] = "USA" [Country] <> "USA" [Quantity] > 0 [Quantity] >= 100 [Quantity] < 0 [Quantity] <= 100
Text concatenation	&	Concatenation of strings	"Value is " & [Amount]
Logical	&& || !	*AND* condition between two Boolean expressions *OR* condition between two Boolean expressions *NOT* operator on the Boolean expression that follows	[Country] = "USA" && [Quantity] > 0 [Country] = "USA" || [Quantity] > 0 ! ([Country] = "USA")

DAX values

You have already seen that you can use a value directly in a formula (for example, USA or 0). When such values are used in this way, they are called *literals* and, although using literals is straightforward, the syntax for referencing a column should be looked at. Here is the basic syntax:

```
'Table Name'[Column Name]
```

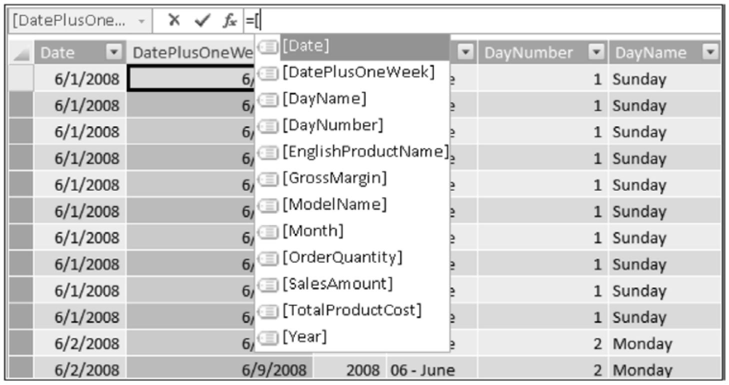
The table name can be enclosed in single quotes. Most of the time, though, the quotes can be omitted if the name does not contain any special characters, such as spaces. In the following formula, for example, the quotes can be omitted:

```
TableName[Column Name]
```

The column name, on the other hand, always needs to be enclosed in square brackets. Note that the table name is optional. If the table name were omitted, the column name would be searched in the current table, which is the one to which the calculated column or measure belongs. However, we strongly suggest that you always specify the complete name (table and column) to avoid any confusion.

IntelliSense

Whenever you write a formula in Excel or PowerPivot, a special help feature called IntelliSense shows all the possible function names and references that you can use in a formula. When you write a formula in the PowerPivot window, if you type an opening square bracket, IntelliSense shows only the columns of the current table (the one in which you are defining the calculated column or the calculated field). If you type the first letters of a table name, you will see both table name and column names in IntelliSense. In Figure 3-4, you can see the list of columns of the current table displayed when the opening square bracket is typed into the calculated column formula.



```
[DatePlusOne...   =[
```

```
[Date]
[DatePlusOneWeek]
[DayName]
[DayNumber]
[EnglishProductName]
[GrossMargin]
[ModelName]
[Month]
[OrderQuantity]
[SalesAmount]
[TotalProductCost]
[Year]
```

Date	DatePlusOneWe			DayNumber	DayName
6/1/2008	6/			1	Sunday
6/1/2008	6/			1	Sunday
6/1/2008	6/			1	Sunday
6/1/2008	6/			1	Sunday
6/1/2008	6/			1	Sunday
6/1/2008	6/			1	Sunday
6/1/2008	6/			1	Sunday
6/1/2008	6/			1	Sunday
6/1/2008	6/			1	Sunday
6/1/2008	6/			1	Sunday
6/2/2008	6/			2	Monday
6/2/2008	6/9/2008	2008	06 - June	2	Monday

FIGURE 3-4 IntelliSense shows all the fields of the current table when you type the opening square bracket into the PowerPivot window.

Understanding calculated columns and fields

Now that you know the basics of DAX syntax, you need to learn one of the most important concepts in DAX: the difference between calculated columns and calculated fields. Even though they might appear similar initially because you can make some calculations both ways, they are in reality very different. Understanding the differences between them is a key to unlocking the true power of DAX.

Calculated columns

In Chapter 2, “Using the unique features of PowerPivot,” you learned how to define a calculated column. You can do this by using the Add button on the Design tab of the ribbon, or you can simply move to the last column, which is named Add Column, and start writing the formula. The DAX expression has to be inserted into the formula bar, and IntelliSense helps you write the expression.

A calculated column is just like any other column in a PowerPivot table. It can be used in rows, columns, filters, or values of a PivotTable. The DAX expression defined for a calculated column operates in the context of the current row of the table that it belongs to. Any reference to a column returns the value of that column in the current row. You cannot access directly the values of other rows.

Note As you will see in the "Aggregate functions" section later in this chapter, there are DAX functions that aggregate the value of a column for the whole table. The only way to get the value of a subset of rows is to use DAX functions that return a table and then operate on it. In this way, you can aggregate column values for a range of rows, or get a single value from another row and operate on it. Using DAX, you can make a reference to a table and apply filters so that only the desired rows are retrieved.

Calculated columns are easy to create and use. You already saw in Figure 3-2 how to define the Gross Margin column to compute the amount of the gross margin:

```
[Gross Margin] = Sales[SalesAmount] - Sales[TotalProductCost]
```

The expression of the calculated column is evaluated for each row, and its result is stored in the table as if it were a column retrieved from the database. Calculated columns are familiar to Excel users because they behave in a very similar way to Excel table columns.

Calculated fields

You might remember from Chapter 2 that the definition of gross margin as a value works fine with calculated columns, but if you want to compute the gross margin as a percentage, then you will need to define a calculated field. It is now time to deepen your understanding of calculated fields and the syntax needed to write them.

A calculated field is a DAX expression that uses the same syntax as a calculated column; the difference is the context of evaluation. A calculated field is evaluated in the context of the cell of the PivotTable, whereas a calculated column is computed at the row level of the PowerPivot table. The cell context depends on the user selections on the PivotTable. When you use *SUM (SalesAmount)* in a calculated field, you mean the sum of all the rows that are aggregated under the PivotTable cell, whereas when you use *Sales[SalesAmount]* in a calculated column, you mean the value of SalesAmount in this row.

When you create a calculated field, you can define a value that changes according to the filter that the user applies to a PivotTable. In this way, you can solve the problem of calculating the gross margin percentage.

To define a calculated field, you can choose New Calculated Field drop-down menu item displayed by clicking the Calculated Fields button on the PowerPivot tab of the Excel ribbon (see Figure 3-5) when a cell in a PivotTable is selected.

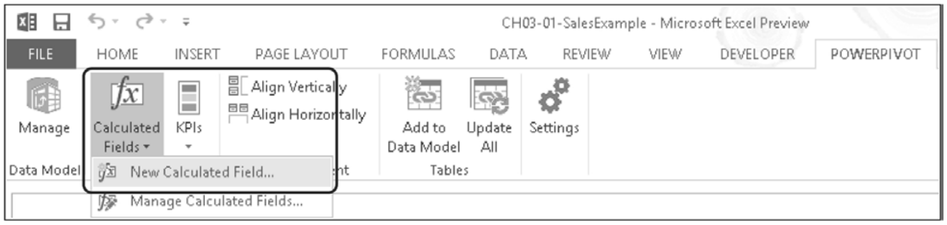


FIGURE 3-5 The New Calculated Field option on the PowerPivot ribbon.

At this point, the Calculated Field dialog box (see Figure 3-6) opens, and you can choose the table name that contains the new calculated field, its name, a description, and finally, the DAX formula and the format string.

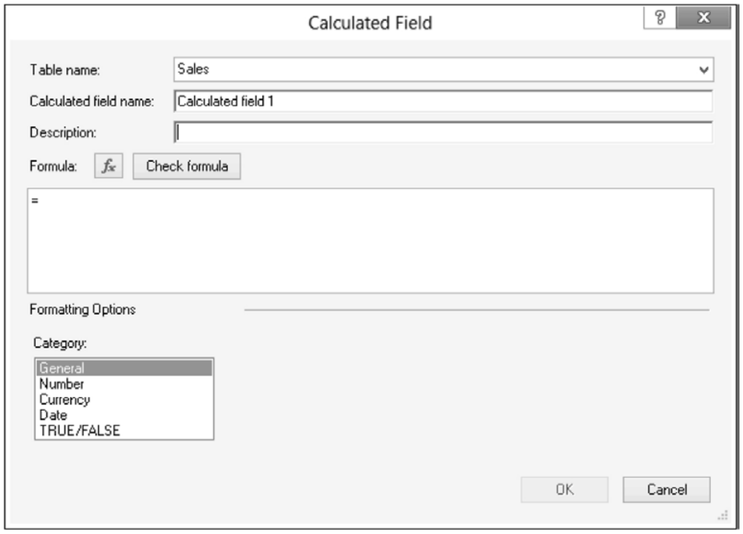

Calculated Field

Table name: Sales

Calculated field name: Calculated field 1

Description:

Formula:

=

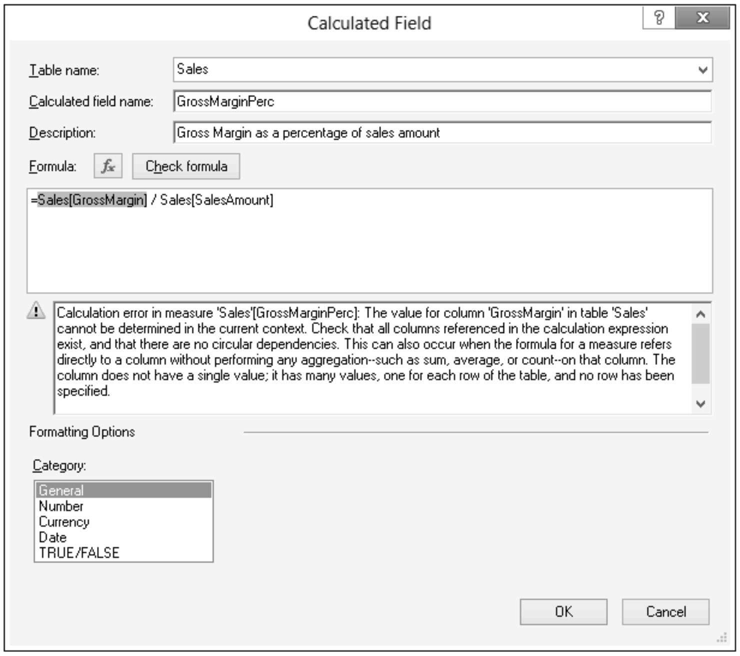
Formatting Options

Category:

General
Number
Currency
Date
TRUE/FALSE

FIGURE 3-6 Here, you can see the Calculated Field window, which is useful for creating a new calculated field.

Now, let's focus on the formula to define a new measure. In a first attempt, you might define it by using the same DAX expression used in the calculated column, but you would get the error shown in Figure 3-7.



```
=Sales[GrossMargin] / Sales[SalesAmount]
```

Calculation error in measure 'Sales'[GrossMarginPerc]: The value for column 'GrossMargin' in table 'Sales' cannot be determined in the current context. Check that all columns referenced in the calculation expression exist, and that there are no circular dependencies. This can also occur when the formula for a measure refers directly to a column without performing any aggregation--such as sum, average, or count--on that column. The column does not have a single value; it has many values, one for each row of the table, and no row has been specified.

FIGURE 3-7 If you define the GrossMarginPerc calculated field with the same DAX expression as a calculated column, you get an error message.

You display the error message by clicking Check Formula. The reason for this error is that the context of execution is not a single row, but a group of rows. That group corresponds to the selection that is implicitly defined by the cell that has to be calculated in the PivotTable. In such a context, which contains multiple rows, there is no way to refer to the value of the column Sales[SalesAmount] because a column has a value only when it is used in the context of a single row.

To avoid this error, you need to define an expression that divides the sum of the gross margin by the sum of the sales. This expression makes use of the *SUM* function, which aggregates all the rows filtered by the current selection in the PivotTable (which will be the year, as we said before). The following is the correct DAX expression for the measure calculated field; it is also shown in Figure 3-8.

```
= SUM ( Sales[Gross Margin] ) / SUM ( Sales[SalesAmount] )
```

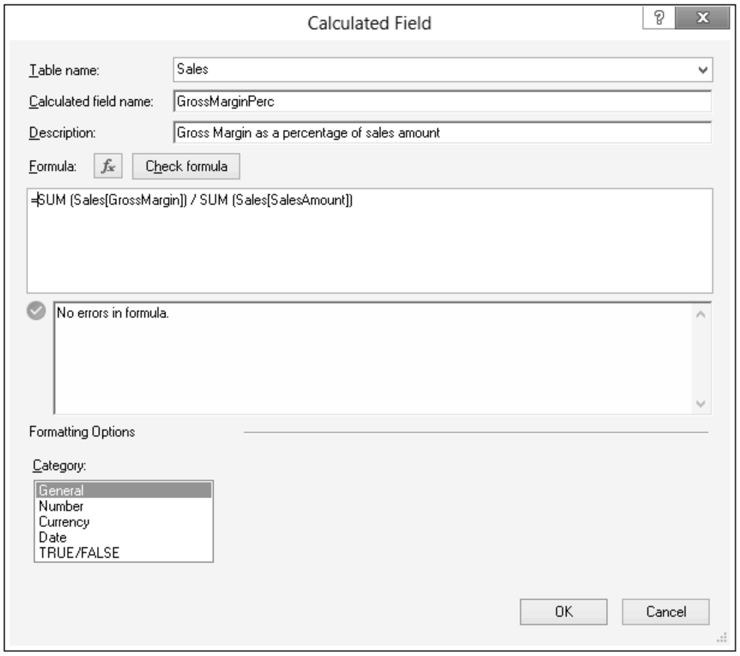

Calculated Field

Table name: Sales

Calculated field name: GrossMarginPerc

Description: Gross Margin as a percentage of sales amount

Formula:

```
=SUM (Sales[GrossMargin]) / SUM (Sales[SalesAmount])
```

No errors in formula.

Formatting Options

Category:
- General
- Number
- Currency
- Date
- TRUE/FALSE

FIGURE 3-8 The correct definition of the GrossMarginPerc measure calculated field.

Differences between calculated columns and calculated fields

Even if they look similar, there is a big difference between calculated columns and calculated fields. The value of a calculated column is computed during data refresh and uses the current row as a context; it does not depend on user activity on the PivotTable. A calculated field operates on aggregations of data defined by the context of the current cell: source tables are filtered according to the coordinates of the cell, and data is aggregated and calculated using these filters. In other words, a calculated field always operates on aggregations of data in the evaluation context, and for this reason the default execution mode does not reference any single row. The evaluation context is explained further in Chapter 7.

Choosing between calculated columns and measures

Now that you have seen the difference between calculated columns and calculated fields, you might be wondering when it is better to use one or the other. Sometimes either is an option, but in most situations, your computation needs determine your choice.

You have to define a calculated column (in the PowerPivot table grid window) whenever you want to do the following:

- Place the calculated results in an Excel slicer or see results in rows or columns in a PivotTable (as opposed to the Values area).

- Define an expression that is strictly bound to the current row. (For example, Price * Quantity cannot work on an average of the two columns.)
- Categorize text or numbers (for example, a range of values for a measure such as customer age, such as 0–18, 18–25, and so on).

However, you must define a calculated field whenever you want to display the resulting calculation values that reflect PivotTable selections made by the user and see them in the Values area of PivotTables. For example:

- When you calculate profit percentage of a PivotTable selection
- When you calculate ratios of a product compared to all products but filter both by year or region

Some calculations can be covered both by calculated columns and calculated fields, even if different DAX expressions must be used in these cases. For example, you can define the GrossMargin as a calculated column as follows:

```
= Sales[SalesAmount] - Sales[TotalProductCost]
```

but it can be defined as a calculated field too:

```
= SUM ( Sales[SalesAmount] ) – SUM ( Sales[TotalProductCost] )
```

The final result is exactly the same. We suggest that you favor the calculated field in this case because, being evaluated at query time, doing so does not consume memory and disk space. However, this factor is really important only in large datasets. When the size of the workbook is not an issue, you can use the method you are more comfortable with.

Cross-references

It is obvious that a calculated field can refer to one or more calculated columns. It might be less intuitive that the opposite is also true: A calculated column can refer to one or more calculated fields. In this way, it forces the calculation of a field for the context defined by the current row. This operation transforms and consolidates the result of a calculated field into a column, which will not be influenced by user actions. Obviously, only certain operations can produce meaningful results because usually a calculated field makes computations that strongly depend on the selection made by the user in the PivotTable.

Handling errors in DAX expressions

Now that you have seen some basic formulas, you should learn how to handle invalid calculations gracefully if (or should we say when?) they happen. A DAX expression might contain invalid calculations because the data that it references is not valid for the formula. For example, you might have a division by zero or a column value that is not a number being used in an arithmetic operation such as multiplication. You must learn how these errors are handled by default and how to intercept these conditions if you want to handle them in a special way.

Before you learn how to handle errors, it is worth describing the different kinds of errors that might appear during a DAX formula evaluation. They are:

- Conversion errors
- Arithmetical operations
- Empty or missing values

Conversion errors

The first kind of error that we analyze is the conversion error. As you have seen before in this chapter, DAX values are automatically converted between strings and numbers whenever the operator requires it. To review the concept, all of these are valid DAX expressions:

```
"10" + 32 = 42
"10" & 32 = "1032"
10 & 32 = "1032"
DATE (2010,3,25) = 3/25/2010
DATE (2010,3,25) + 14 = 4/8/2010
DATE (2010,3,25) & 14 = "3/25/201014"
```

These formulas are always correct because they operate with constant values. But what about the following one?

```
SalesOrders[VatCode] + 100
```

Because the first operator of this sum is obtained by a column (which, in this case, is a text column), you must be sure that all the values in that column are numbers to ascertain whether they will be converted and the expression will be evaluated correctly. If some of the content cannot be converted to suit the operator's needs, you will incur a conversion error. Here are typical situations:

```
"1 + 1" + 0              = Cannot convert value '1+1' of type string to type real
DATEVALUE ("25/14/2010") = Type mismatch
```

To avoid these errors, you need to write more complex DAX expressions that contain error detection logic to intercept error conditions and always return a meaningful result.

Arithmetical operations

The second category of errors is arithmetical operations, such as division by zero or the square root of a negative number. These kinds of errors are not related to conversion; they are raised whenever you try to call a function or use an operator with invalid values.

In PowerPivot, division by zero requires a special handling because it behaves in a way that is not very intuitive (except for mathematicians). When you divide a number by zero, PowerPivot usually returns the special value *Infinity*. Moreover, in the very special cases of 0 divided by 0 or *Infinity* divided by *Infinity,* PowerPivot returns the special *NaN* (not a number) value. Because this is a strange behavior for Excel users to encounter, we have summarized it in Table 3-2.

TABLE 3-2 Special result values for division by zero

Expression	Result
10 / 0	*Infinity*
7 / 0	*Infinity*
0 / 0	*NaN*
(10 / 0) / (7 / 0)	*NaN*

It is important to note that *Infinity* and *NaN* are not errors, but special values in PowerPivot. In fact, if you divide a number by *Infinity,* the expression does not generate an error; rather, it returns 0 (note that in the following expression, 7/0 results in *Infinity*):

```
9954 / (7 / 0)     = 0
```

Apart from this special situation, arithmetical errors might be returned when calling a DAX function with a wrong parameter, such as the square root of a negative number:

```
SQRT ( -1 )     = An argument of function 'SQRT' has the wrong data type
                  or the result is too large or too small
```

If PowerPivot detects errors like this, it blocks any further computation of the DAX expression and raises an error. You can use the special DAX *ISERROR* function to check if an expression leads to an error, something that you will use in the “Intercepting errors” section later in this chapter. Finally, even if special values like *NaN* are displayed in this way in the PowerPivot window, they are displayed as errors when shown in an Excel PivotTable and will be detected as errors by the error detection functions.

Empty or missing values

The third category is not a specific error condition. Rather, it is the presence of empty values, which might result in unexpected results or calculation errors when combining those empty values with other elements in a calculation. You need to understand how these special values are treated in PowerPivot.

DAX handles missing values, blank values, and empty cells in the same way; using the value *BLANK*. *BLANK* is not a real value but a special way to identify these conditions. The value *BLANK* can be obtained in a DAX expression by calling the *BLANK* function, which is different from an empty string. For example, the following expression always returns a blank value, which is displayed as an empty cell in the PowerPivot window:

```
= BLANK ()
```

On its own, this expression is useless, but the *BLANK* function becomes useful every time you want to return an empty value. For example, you might want to display an empty cell instead of 0, as in the following expression that calculates the total discount for a sale transaction and leaves the cell blank if the discount is 0:

```
= IF ( Sales[DiscountPerc] = 0, BLANK (), Sales[DiscountPerc] * Sales[Amount] )
```

If a DAX expression contains a *BLANK,* it is not considered an error, but an empty value. So an expression containing a *BLANK* might return a value or a blank, depending on the calculation required. For example, the following expression:

```
= 10 * Sales[Amount]
```

returns *BLANK* whenever Sales[Amount] is *BLANK*. In other words, the result of an arithmetic product is *BLANK* whenever one or both terms are *BLANK*. This propagation of *BLANK* in a DAX expression happens in several other arithmetical and logical operations, as shown in the following examples:

```
BLANK () + BLANK ()   = BLANK ()
10 * BLANK ()         = BLANK ()
BLANK () / 3          = BLANK ()
BLANK () / BLANK ()   = BLANK ()
BLANK () || BLANK ()  = FALSE
BLANK () && BLANK ()  = FALSE
```

However, the propagation of *BLANK* in the result of an expression is not produced by all formulas. Some calculations do not propagate *BLANK,* but return a value depending on the other terms of the formula. Examples of these are addition, subtraction, division by *BLANK,* and a logical operation between a *BLANK* and a valid value. In the following expressions, you can see some examples of these conditions, along with their results:

```
BLANK () - 10           = -10
18 + BLANK ()           = 18
4 / BLANK ()            = Infinity
0 / BLANK ()            = NaN
FALSE || BLANK          = FALSE
FALSE && BLANK          = FALSE
TRUE || BLANK           = TRUE
TRUE && BLANK           = FALSE
```

Empty values in Excel

Excel has a different way of handling empty values. In Excel, all empty values are considered 0 whenever they are used in a sum or in multiplication, but they return an error if they are part of division or of a logical expression.

Understanding the behavior of empty or missing values in a DAX expression and using *BLANK* to return an empty cell in a calculation are also important skills that control the results of a DAX expression. You can often use *BLANK* as a result when you detect wrong values or other errors, as you are going to learn in the next section.

Intercepting errors

Now that you have seen the various kinds of errors that can occur, you can learn a technique to intercept errors and correct them or, at least, show an error message that contains some meaningful information. The presence of errors in a DAX expression frequently depends on the value contained in tables and columns referenced in the expression itself. So you might want to control the presence of these error conditions and return an error message. The standard technique is to check whether an expression returns an error and, if so, replace the error with a message or a default value. A few DAX functions have been designed to do this.

The first of them is the *IFERROR* function, which is very similar to the *IF* function, but instead of evaluating a *TRUE/FALSE* condition, it checks whether an expression returns an error. You can see two typical uses of the *IFERRROR* function here:

```
= IFERROR ( Sales[Quantity] * Sales[Price], BLANK () )
= IFERROR ( SQRT ( Test[Omega] ), BLANK () )
```

In the first expression, if either Sales[Quantity] or Sales[Price] are strings that cannot be converted into a number, the returned expression is an empty cell; otherwise the product of Quantity and Price is returned.

In the second expression, the result is an empty cell every time the Test[Omega] column contains a negative number.

When you use *IFERROR* this way, you follow a more general pattern that requires the use of *ISERROR* and *IF:*

```
= IF (
    ISERROR ( Sales[Quantity] * Sales[Price] ),
    BLANK (),
    Sales[Quantity] * Sales[Price]
  )

= IF (
    ISERROR ( SQRT ( Test[Omega] ) ),
    BLANK (),
    SQRT ( Test[Omega] )
  )
```

You should use *IFERROR* whenever the expression that has to be returned is the same one that is being tested for an error; you do not have to duplicate the expression in two places, and the resulting formula is more readable and easier to fix later without introducing errors. You should use *IF,* however, when you want to return the result of a different expression when there is an error.

For example, *ISNUMBER* can be used to detect whether a string (the price in the first line) can be converted to a number and then calculate the total amount; otherwise, an empty cell can be returned.

```
= IF ( ISNUMBER ( Sales[Price] ), Sales[Quantity] * Sales[Price], BLANK () )
= IF ( Test[Omega] >= 0, SQRT ( Test[Omega] ), BLANK () )
```

The second example simply detects whether the argument for *SQRT* is valid or not, calculating the square root only for positive numbers and returning *BLANK* for negative ones.

A particular case is the test against the empty value. The *ISBLANK* function detects an empty value condition, returning *TRUE* if the argument is *BLANK*. This is important especially when a missing value has a meaning different from a value set to 0. In the following example, we calculate the cost of shipping for a sales transaction, using a default shipping cost for the product if the weight is not specified in the sales transaction:

```
= IF (
    ISBLANK ( Sales[Weight] ),
    RELATED ( Product[DefaultShippingCost] ),
    Sales[Weight] * Sales[ShippingPrice]
)
```

If we had just multiplied product weight and shipping price, we would have gotten an empty cost for all the sales transactions with missing weight data.

Formatting DAX code

Before continuing with the explanation of the DAX language, it is useful to cover a very important aspect of DAX: formatting the code. DAX is a functional language, meaning that no matter how complex it is, a DAX expression is always a single function call with some parameters. The complexity of the code is reflected in the complexity of the expressions that are passed as parameters to the outermost function.

For this reason, it is normal to see expressions that span 10 lines or more. Seeing a 20-line DAX expression may seem strange to you, but it is normal, and you will get used to it. Nevertheless, as formulas start to grow in length and complexity, it is extremely important that you learn how to write them correctly so that they are readable by humans.

There is no “official” standard to format DAX code, yet we believe that it is important to describe the standard that we used with our code. It is probably not perfect, and you might prefer something different. The only thing you need to remember when formatting your code is: “Do not write everything on a single line, or you will get in trouble before you know it.”

To demonstrate why formatting is so important, we show here a formula that you will use in Chapter 12, “Performing date calculations in DAX” It is somewhat complex to learn, but not the most complex formula you will create. Here is what the expression looks like if you do not format it in some way:

```
IF (COUNTX (BalanceDate, CALCULATE (COUNT( Balances[Balance] ), ALLEXCEPT
( Balances, BalanceDate[Date] ))) > 0, SUMX (ALL ( Balances[Account] ), CALCULATE
(SUM( Balances[Balance] ), LASTNONBLANK (DATESBETWEEN (BalanceDate[Date],
BLANK(),LASTDATE( BalanceDate[Date] )), CALCULATE ( COUNT( Balances[Balance]
))))), BLANK ())
```

Trying to understand what this formula computes is nearly impossible because you have no idea which is the outermost function and how the different function calls are merged to create the complete flow of execution. We have seen too many examples of formulas written this way by customers that, at some point, ask for help in understanding why the formula returns incorrect results. Guess what? The first thing we do is format the expression—only then can we start working on it.

The same expression, properly formatted, looks like this:

```
=IF (
    COUNTX (
        BalanceDate,
        CALCULATE (
            COUNT ( Balances[Balance] ),
            ALLEXCEPT ( Balances, BalanceDate[Date] )
        )
    ) > 0,
    SUMX (
        ALL ( Balances[Account] ),
        CALCULATE (
            SUM ( Balances[Balance] ),
            LASTNONBLANK (
                DATESBETWEEN (
                    BalanceDate[Date],
                    BLANK (),
                    LASTDATE ( BalanceDate[Date] )
                ),
                CALCULATE ( COUNT ( Balances[Balance] ) )
            )
        )
    ),
    BLANK ()
)
```

The code is the same, but now it is much easier to identify the three parameters of *IF* and, most important, to follow the blocks that raise naturally from the indented lines and see how they create the complete flow of execution. Yes, the code is still hard to read, but now the problem is with DAX, not the formatting. And you are shortly going to learn how to read and manage DAX.

Thus, this is the set of rules that we use in this book and the associated workbooks:

- Keywords like *IF, COUNTX,* and *CALCULATE* are always separated from any other term with a space, and they are always written in uppercase.
- All column references are written in the form TableName[ColumnName], with no space between the table name and the opening square bracket.
- Commas are always followed by a space and are never preceded by a space.
- If the formula fits one single line, then no other rule need to be applied.

- If the formula does not fit on a single line, then the following applies:
    - The function name stands on a line by itself, with the opening parenthesis.
    - All the parameters are on separate lines, indented with four spaces and with a comma at the end of the expression.
    - The closing parenthesis is aligned with the function call and stands on a line by itself.

These are the basic rules that we use. If you find a way to express formulas that best fits your reading attitude, then use it. The goal of formatting is to make the formula easier to read, so use the way that works best for you. The most important thing to remember when defining your personal set of formatting rules is that you always need to see errors as soon as possible. If, on the unformatted code shown earlier, DAX alerts you to a missing closing parenthesis, you will have a very hard time spotting the error. On the other hand, it is much easier to see how the closing parenthesis matches the opening function calls in the formatted code.

Help with formatting DAX

Formatting DAX is not easy because you need to write it using a small font in a text box, and unfortunately, Excel does not provide an editor for it. Nevertheless, a few hints might help you when writing your DAX code:

- If you want to make the text bigger, you can use Ctrl + the mouse wheel to increase the font size, making it easier to see the code.
- If you want to add a new line to the formula, you can press Shift+Enter.
- If editing in the text box is really difficult, you can always copy the code into another editor, like Notepad, make your changes, and then paste the formula into the text box again.

Finally, whenever you look at a DAX expression, it is hard to understand at first glance whether it is a calculated column or a calculated field. Thus, we use an equals sign (=) in this book to define a calculated column and the assignment operator (:=) to define calculated fields:

```
CalcCol = SUM ( Sales[SalesAmount] )       is a calculated column
CalcFld := SUM ( Sales[SalesAmount] )      is a calculated field
```

Common DAX functions

Now that you have learned about the fundamentals of DAX and how to handle error conditions, let's take a brief tour through its most commonly used functions and expressions. Writing a DAX expression is often similar to writing an Excel expression because many functions are similar, if not identical. Excel users often find using PowerPivot very intuitive, thanks to their previous knowledge of Excel. In the remaining part of this chapter, you will see some of the most frequently used DAX functions, which you are likely to use to build your own PowerPivot data models.

You can see all the formulas shown in this section in the companion workbook "CH03-02-Aggregation Functions.xlsx."

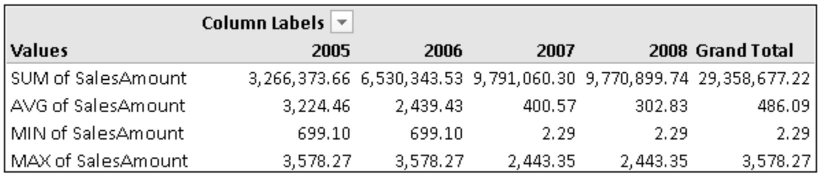
Aggregate functions

Almost every PowerPivot data model needs to work on aggregated data. DAX offers a set of functions that aggregate the values of a column in a table and return a single value. We call this group of functions *aggregate functions*. For example, the expression:

```
= SUM ( Sales[Amount] )
```

calculates the sum of all the numbers in the Amount column of the Sales table. This expression aggregates all the rows of the Sales table if it is used in a calculated column, but it considers only the rows that are filtered by slicers, row, columns, and filter conditions in a PivotTable whenever it is used in a measure.

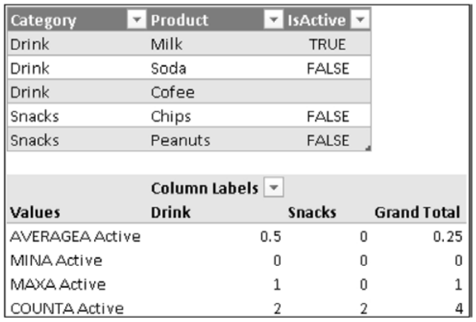
The main four aggregate functions (*SUM, AVERAGE, MIN,* and *MAX*) operate only on numeric values. These functions are identical to the corresponding Excel functions both in name and in behavior: any data that is not numeric is ignored in the operation. In PowerPivot, these functions work only if the column passed as an argument is of numeric or date type. In Figure 3-9, you can see an example of calculated fields defined by these aggregate functions.



	Column Labels				
Values	2005	2006	2007	2008	Grand Total
SUM of SalesAmount	3,266,373.66	6,530,343.53	9,791,060.30	9,770,899.74	29,358,677.22
AVG of SalesAmount	3,224.46	2,439.43	400.57	302.83	486.09
MIN of SalesAmount	699.10	699.10	2.29	2.29	2.29
MAX of SalesAmount	3,578.27	3,578.27	2,443.35	2,443.35	3,578.27

FIGURE 3-9 In this PivotTable, you can see different calculated fields using statistical functions that aggregate SalesAmount values.

As in Excel formulas, DAX offers an alternative syntax to these functions to make the calculation on columns that can contain both numeric and non-numeric values, such as text columns. That syntax simply adds the suffix *A* to the name of the function to get the same name and behavior as the same function in Excel. However, these functions are useful only for columns containing *TRUE/FALSE* values because *TRUE* is evaluated as 1 and *FALSE* as 0. Any value for a text column is always considered to be 0. Empty cells are never considered in the calculation. So even if these functions can be used in non-numeric columns without retuning an error, their results are not always the same as Excel because there is no automatic conversion to numbers for text columns. These functions are named *AVERAGEA*, *COUNTA*, *MINA*, and *MAXA*, and Figure 3-10 displays an example of their usage in measures operating in a *TRUE/FALSE* column of the sample table shown in the same worksheet. The table is used as a linked table in PowerPivot, and the lower part of the screenshot is a PivotTable based on that PowerPivot data.



Category	Product	IsActive
Drink	Milk	TRUE
Drink	Soda	FALSE
Drink	Cofee	
Snacks	Chips	FALSE
Snacks	Peanuts	FALSE

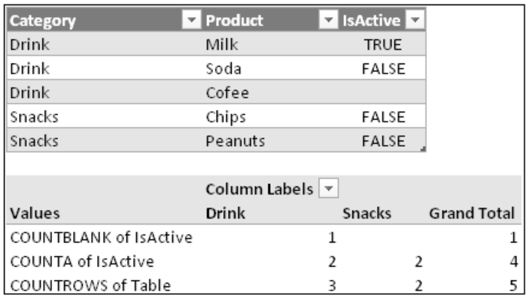
	Column Labels		
Values	Drink	Snacks	Grand Total
AVERAGEA Active	0.5	0	0.25
MINA Active	0	0	0
MAXA Active	1	0	1
COUNTA Active	2	2	4

FIGURE 3-10 *TRUE/FALSE* is evaluated as 1/0 in *A*-suffixed statistical functions.

Even though these aggregate functions have the same name, there is a difference in the way they are used in DAX and Excel. In PowerPivot, a column has a type, and its type determines the behavior of aggregate functions in that column. Excel handles a type for each cell, whereas PowerPivot handles a type for each column. PowerPivot deals with data in tabular form (technically called *relational data*) with well-defined types for each column, whereas Excel formulas work on heterogeneous cell values without well-defined types. If a column in PowerPivot is of a number type, all the values can be only numbers or empty cells. If a column is of a text type, these functions behave as if it were always 0 (except for *COUNTA*), even if the text can be converted to a number, whereas in Excel, the value is considered a number on a cell-by-cell basis. For these reasons, these DAX functions are not very useful for text-type columns.

The only interesting function in the group of *A*-suffixed functions is *COUNTA*. It returns the number of cells that are not empty and works on any type of column. If you are interested in counting all the cells in a column containing an empty value, you can use the *COUNTBLANK* function. Finally, if you want to count all the cells of a column regardless of their content, you want to count the number of rows of the table, which can be obtained by calling the *COUNTROWS* function. (It gets a table as a parameter, not a column.) In other words, the sum of *COUNTA* and *COUNTBLANK* for the same column of a table is always equal to the number of rows of the same table, as shown in Figure 3-11.

```
COUNTROWS( Sales ) = COUNTA ( Sales[SalesPersonID] )
                     + COUNTBLANK ( Sales[SalesPersonID] )
```



Category	Product	IsActive
Drink	Milk	TRUE
Drink	Soda	FALSE
Drink	Cofee	
Snacks	Chips	FALSE
Snacks	Peanuts	FALSE

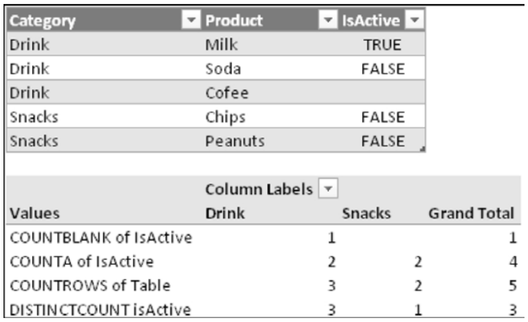
	Column Labels		
Values	Drink	Snacks	Grand Total
COUNTBLANK of IsActive	1		1
COUNTA of IsActive	2	2	4
COUNTROWS of Table	3	2	5

FIGURE 3-11 The *COUNTROWS* function returns the sum of *COUNTA* and *COUNTBLANK* of the same column.

So you can use four functions to count the number of elements in a column or table:

- *COUNT* operates only on numeric columns
- *COUNTA* operates on any type of columns
- *COUNTBLANK* returns the number of empty cells in a column
- *COUNTROWS* returns the number of rows in a table

There is still another very important counting function. *DISTINCTCOUNT,* which does exactly what its name suggests: it counts the distinct values of a column, which it takes as its only parameter. *DISTINCTCOUNT* counts the *BLANK* value as one of the possible values. Thus, as shown in Figure 3-12, the *DISTINCTCOUNT* of the IsActive column for category Drink is 3 because *BLANK* is a possible value.



Category	Product	IsActive
Drink	Milk	TRUE
Drink	Soda	FALSE
Drink	Cofee	
Snacks	Chips	FALSE
Snacks	Peanuts	FALSE

	Column Labels		
Values	Drink	Snacks	Grand Total
COUNTBLANK of IsActive	1		1
COUNTA of IsActive	2	2	4
COUNTROWS of Table	3	2	5
DISTINCTCOUNT isActive	3	1	3

FIGURE 3-12 *DISTINCTCOUNT* counts the *BLANK* value as a valid value.

Note *DISTINCTCOUNT* is a function introduced in the 2012 version of PowerPivot. To compute the number of distinct values of a column in the previous version of PowerPivot, you had to use *COUNTROWS(DISTINCT(ColName)).* The two patterns return the very same result, but *DISTINCTCOUNT* is easier to read, requiring only a single function call.

A last set of statistical functions can apply an expression to each row of a table and then operate an aggregation on that expression. This set of functions is very useful, especially when you want to make calculations using columns of different related tables. For example, if a Sales table contains all the sales transactions and a related Product table contains all the information about a product, including its cost, you might calculate the total internal cost of a sales transaction by defining a calculated field with this expression:

```
Cost := SUMX ( Sales, Sales[Quantity] * RELATED ( Product[StandardCost] ) )
```

This function calculates the product of Quantity (from the Sales table) and the StandardCost of the sold product (from the related Product table) for each row in the Sales table, and it returns the sum of all these calculated values.

Generally, all the aggregate functions ending with an *X* behave in the following way: they calculate an expression (the second parameter) for each of the rows of a table (the first parameter) and return a result obtained by the corresponding aggregate function (*SUM, MIN, MAX,* or *COUNT*) applied to the result of those calculations.

Evaluation context is important for understanding how this calculation works. You will learn more about this behavior in Chapter 7, "Understanding evaluation contexts." The *X*-suffixed functions available are *SUMX, AVERAGEX, COUNTX, COUNTAX, MINX,* and *MAXX*.

Logical functions

Sometimes you may want to build a logical condition in an expression—for example, to implement different calculations depending on the value of a column or to intercept an error condition. In these cases, you can use one of the logical functions in DAX. In the section, "Handling Errors in DAX Expressions," earlier in this chapter, you learned the two most important functions of this group, which are *IF* and *IFERROR*. All of these functions are very simple and do what their names suggest: *AND, FALSE, IF, IFERROR, SWITCH, NOT, TRUE,* and *OR*. If, for example, you want to compute the Amount as Quantity multiplied by Price only when the Price column contains a correct numeric value, you can use the following pattern:

```
Amount := IFERROR ( Sales[Quantity] * Sales[Price], BLANK () )
```

If you did not use *IFERROR* and the *Price* value contains an invalid number, the result for the calculated column would be an error because if a single row generates a calculation error, that error is propagated to the whole column. The usage of *IFERROR*, however, intercepts the error and replaces it with a blank value.

Another interesting function in this category is *SWITCH*, which is useful when you have a column containing a low number of distinct values and you want to get different behaviors depending on the value. For example, the Size column in the DimProduct table contains L, M, S, and XL, and you might want to decode this value in a more meaningful column. You can obtain the result by using nested *IF* calls:

```
SizeDesc :=
    IF ( DimProduct[Size] = "S", "Small",
        IF ( DimProduct[Size] = "M", "Medium",
            IF ( DimProduct[Size] = "L", "Large",
              IF ( DimProduct[Size] = "XL", "Extra Large", "Other" ) ) ) )
```

A more convenient way to express the same formula using *SWITCH* is:

```
SizeDesc :=
    SWITCH (
        DimProduct[Size],
       "S", "Small",
       "M", "Medium",
       "L", "Large",
       "XL", "Extra Large",
       "Other"
    )
```

The code in this expression is more readable, even though it is not faster, because, internally, *SWITCH* statements are translated into nested *IF*.

TIP Because *SWITCH* is converted into a set of nested *IF,* where the first one that matches takes precedence, you can test multiple conditions in the same expression using this pattern:

```
SWITCH (
    TRUE (),
    DimProduct[Size] = "XL" && DimProduct[Color] = "Red", "Red and XL",
    DimProduct[Size] = "XL" && DimProduct[Color] = "Blue", "Blue and XL",
    DimProduct[Size] = "L" && DimProduct[Color] = "Green", "Green and L"
)
```

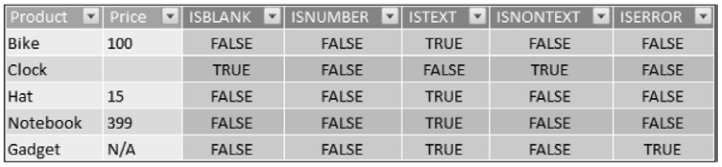
Using *TRUE* as the first parameter means "Return the first result where the condition evaluates to true."

Information functions

Whenever you need to analyze the type of an expression, you can use one of the information functions in DAX. All of these functions return a *TRUE/FALSE* value and can be used in any logical expression. They are *ISBLANK, ISERROR, ISLOGICAL, ISNONTEXT, ISNUMBER,* and *ISTEXT.*

It is important to note that when a table column is passed as a parameter, the functions *ISNUMBER, ISTEXT,* and *ISNONTEXT* always return *TRUE* or *FALSE,* depending on the data type of the column and on the empty condition of each cell. In Figure 3-13, you can see how the column Price (which is of Text type) affects the result of these calculated columns.

```
ISBLANK          = ISBLANK ( Sales[Price] )
ISNUMBER         = ISNUMBER ( Sales[Price] )
ISTEXT           = ISTEXT ( Sales[Price] )
ISNONTEXT        = ISNONTEXT ( Sales[Price] )
ISERROR          = ISERROR ( Sales[Price] + 0 )
```



Product	Price	ISBLANK	ISNUMBER	ISTEXT	ISNONTEXT	ISERROR
Bike	100	FALSE	FALSE	TRUE	FALSE	FALSE
Clock		TRUE	FALSE	FALSE	TRUE	FALSE
Hat	15	FALSE	FALSE	TRUE	FALSE	FALSE
Notebook	399	FALSE	FALSE	TRUE	FALSE	FALSE
Gadget	N/A	FALSE	FALSE	TRUE	FALSE	TRUE

FIGURE 3-13 The results from information functions are based on column type.

You might be wondering whether *ISNUMBER* can be used with a text column just to check whether a conversion to a number is possible. Unfortunately, you cannot use this approach; if you want to test whether a text value can be converted to a number, you must try the conversion and handle the error if it fails. For example, to test whether the column Price (which is of type *String*) contains a valid number, you must write:

```
IsPriceCorrect = ISERROR ( Sales[Price] + 0 )
```

To get a *TRUE* result from the *ISERROR* function, for example, DAX tries to add a zero to the Price to force the conversion from a text value to a number. The conversion fails for the N/A price value, so you can see that *ISERROR* is *TRUE.*

Suppose, however, that you try to use *ISNUMBER,* as in the following expression:

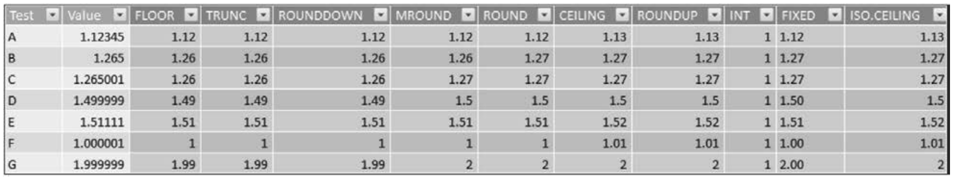
```
IsPriceCorrect = ISNUMBER ( Sales[Price] )
```

In this case, you will always get *FALSE* as a result because, based on metadata, the Price column is not a number but a string.

Mathematical functions

The set of mathematical functions available in DAX is very similar to the same set in Excel, with the same syntax and behaviors. The most commonly used mathematical functions are *ABS, EXP, FACT, LN, LOG, LOG10, MOD, PI, POWER, QUOTIENT, SIGN,* and *SQRT*. Random functions include *RAND* and *RANDBETWEEN*. Finally, there are several functions to round numbers that deserve an example; in fact, you might use several approaches to get the same result. Consider these calculated columns, along with their results in Figure 3-14:

```
FLOOR        = FLOOR ( Tests[Value], 0.01 )
TRUNC        = TRUNC ( Tests[Value], 2 )
ROUNDDOWN    = ROUNDDOWN ( Tests[Value], 2 )
MROUND       = MROUND ( Tests[Value], 0.01 )
ROUND        = ROUND ( Tests[Value], 2 )
CEILING      = CEILING ( Tests[Value], 0.01 )
ISO.CEILING  = ISO.CEILING ( Tests[Value], 0.01 )
ROUNDUP      = ROUNDUP ( Tests[Value], 2 )
INT          = INT ( Tests[Value] )
FIXED        = FIXED ( Tests[Value], 2, TRUE )
```



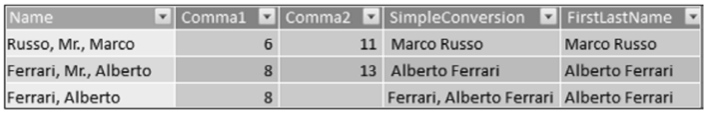
Test	Value	FLOOR	TRUNC	ROUNDDOWN	MROUND	ROUND	CEILING	ROUNDUP	INT	FIXED	ISO.CEILING
A	1.12345	1.12	1.12	1.12	1.12	1.12	1.13	1.13	1	1.12	1.13
B	1.265	1.26	1.26	1.26	1.26	1.27	1.27	1.27	1	1.27	1.27
C	1.265001	1.26	1.26	1.26	1.27	1.27	1.27	1.27	1	1.27	1.27
D	1.499999	1.49	1.49	1.49	1.5	1.5	1.5	1.5	1	1.50	1.5
E	1.51111	1.51	1.51	1.51	1.51	1.51	1.52	1.52	1	1.51	1.52
F	1.000001	1	1	1	1	1	1.01	1.01	1	1.00	1.01
G	1.999999	1.99	1.99	1.99	2	2	2	2	1	2.00	2

FIGURE 3-14 A summary of different rounding functions.

As shown, *FLOOR, TRUNC,* and *ROUNDDOWN* are very similar, except in the way that you can specify the number of digits to round to. *CEILING* and *ROUNDUP* are very similar in their results, but in the opposite way. You can see a few differences in the way the rounding is done between the *MROUND* and *ROUND* functions. Finally, it is important to note that *FLOOR* and *MROUND* functions do not work on negative numbers, whereas other functions do.

Text functions

Almost all the text functions available in DAX are similar to those available in Excel, with only a few exceptions: *CONCATENATE, EXACT, FIND, FIXED, FORMAT, LEFT, LEN, LOWER, MID, REPLACE, REPT, RIGHT, SEARCH, SUBSTITUTE, TRIM, UPPER,* and *VALUE*. These functions are useful for manipulating text and extracting data from strings that contain multiple values. For example, in Figure 3-15, you can see an example of the extraction of first and last name from a string containing these values separated by commas, with the title in the middle that we want to remove.



Name	Comma1	Comma2	SimpleConversion	FirstLastName
Russo, Mr., Marco	6	11	Marco Russo	Marco Russo
Ferrari, Mr., Alberto	8	13	Alberto Ferrari	Alberto Ferrari
Ferrari, Alberto	8		Ferrari, Alberto Ferrari	Alberto Ferrari

FIGURE 3-15 Here, you can see an example of extracting first and last names using text functions.

You start by calculating the position of the two commas, and then you use these numbers to extract the right part of the text. The SimpleConversion column implements a formula that might return wrong values if there are fewer than two commas in the string (and it raises an error if there are no commas at all), whereas the FirstLastName column implements a more complex expression that does not fail in the case of missing commas. In fact, you can see in Figure 3-15 that the last row shows an incorrect value in the SimpleConversion column, whereas the FirstLastName column shows a correct conversion.

```
Comma1 = IFERROR ( FIND ( ",", People[Name] ), BLANK () )
Comma2 = IFERROR ( FIND ( ",", People[Name], People[Comma1] + 1 ), BLANK () )
SimpleConversion = MID ( People[Name], People[Comma2] + 1, LEN ( People[Name] ) )
                   & " " & LEFT ( People[Name], People[Comma1] - 1 )
FirstLastName = TRIM (
                    MID (
                        People[Name],
                        IF (
                            ISNUMBER ( People[Comma2] ),
                            People[Comma2],
                            People[Comma1]
                        ) + 1,
                        LEN ( People[Name] )
                    )
                )
                & IF (
                  ISNUMBER ( People[Comma1] ),
                  " " & LEFT ( People[Name], People[Comma1] - 1 ),
                  ""
                )
```

The FirstLastName column is defined by a long DAX expression, but you must use it to avoid possible errors that would propagate to the whole column if even a single value generated an error.

Conversion functions

You learned at the beginning of this chapter that DAX converts data types automatically to adjust them to the needs of the operators. Even if this happens automatically, a set of functions still can perform explicit conversion of types.

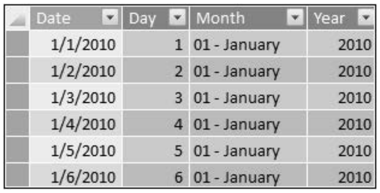
CURRENCY can transform an expression into a currency type, whereas *INT* transforms an expression into an integer. *DATE* and *TIME* take the date and time parts as parameters and return a correct *DATETIME*. *VALUE* transforms a string into a numeric format, whereas *FORMAT* gets a numeric value as the first parameter and a string format as its second one and can transform numeric values into strings.

Date and time functions

In almost every type of data analysis, handling time and date is an important aspect. PowerPivot has a large number of functions that operate on date and time. Some of them correspond to similar functions in Excel and make simple transformations to and from a *datetime* data type. The date and time functions are *DATE, DATEVALUE, DAY, EDATE, EOMONTH, HOUR, MINUTE, MONTH, NOW, SECOND, TIME, TIMEVALUE, TODAY, WEEKDAY, WEEKNUM, YEAR,* and *YEARFRAC*. To make more complex operation on dates, such as comparing aggregated values year over year or calculating the year-to-date value of a measure, there is another set of functions called Time Intelligence functions, which will be described later in the book in Chapter 12.

As mentioned before in this chapter, a *datetime* data type internally uses a floating-point number wherein the integer part corresponds to the number of the day (starting from December 30, 1899) and the decimal part indicates the fraction of the day in time. (Hours, minutes, and seconds are converted into decimal segments of the day.) So adding an integer number to a *datetime* value increments the value by a corresponding number of days. However, most of the time the conversion functions are used to extract day, month, and year from a date. The following example demonstrates how to extract this information from a table containing a list of dates (see Figure 3-16 for the result of the code):

```
Day    = DAY ( Calendar[Date] )
Month  = FORMAT ( Calendar[Date], "MM - mmmm" )
Year   = YEAR ( Calendar[Date] )
```



Date	Day	Month	Year
1/1/2010	1	01 - January	2010
1/2/2010	2	01 - January	2010
1/3/2010	3	01 - January	2010
1/4/2010	4	01 - January	2010
1/5/2010	5	01 - January	2010
1/6/2010	6	01 - January	2010

FIGURE 3-16 Here, you can see an example of extracting date information using date and time functions.

As Figure 3-16 shows, the Month column is calculated using the *FORMAT* function, which is classified as a text function but is very useful for building a string that keeps the right sort order of the months by placing the month number before the month name. (The Day and Year columns are sorted in the right order because of their numeric data type.)

Relational functions

Two useful functions that enable you to navigate through relationships inside a DAX formula are *RELATED* and *RELATEDTABLE*. In Chapters 7 and 8, you learn all the details of how these functions work, but for now, it is worthwhile to discuss them briefly here.

You already know that a calculated column can reference column values of the table in which it is defined. Thus, a calculated column defined in FactResellerSales can reference any column of the same table. But what can you do if you must refer to a column in another table? In general, you cannot use columns in another table unless a relationship is defined between the two tables. However, if the two tables are in a relationship, then the *RELATED* function enables you to access columns in the related table.

For example, you might want to compute a calculated column in the FactResellerSales table that checks whether the product that has been sold is in the Bikes category and, if so, apply a reduction factor to the standard cost. To compute such a column, you must write an *IF* function that checks the value of the product category, which is not in the FactResellerSales table. Nevertheless, a chain of relationships starts from FactInternetSales, reaching DimProductCategory through DimProduct and DimProductSubcategory, as Figure 3-17 shows.

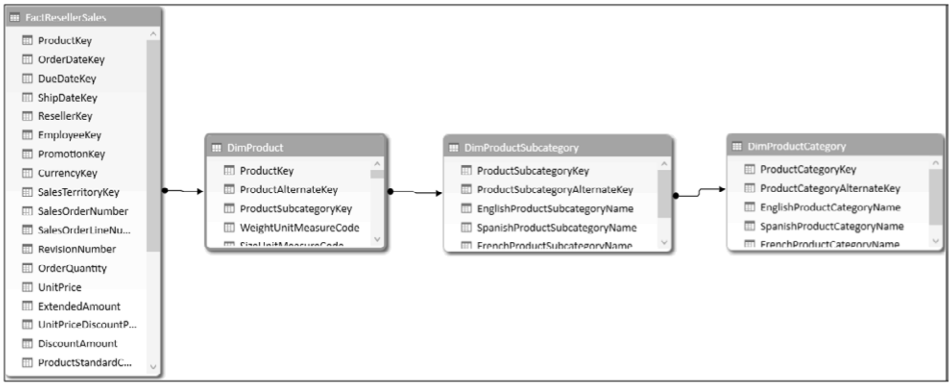


FIGURE 3-17 FactResellerSales has a chained relationship with DimProductCategory.

It does not matter how many steps are necessary to travel from the original table to the related one, DAX will follow the complete chain of relationships and return the related column value. Thus, the formula for the AdjustedCost column can be expressed as follows:

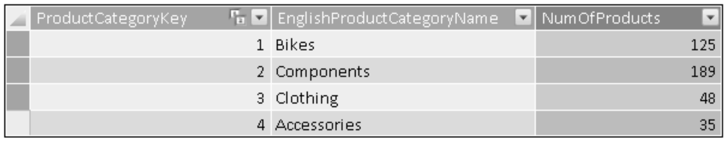
```
=IF (
    RELATED ( DimProductCategory[EnglishProductCategoryName] ) = "Bikes",
    [ProductStandardCost] * 0.95,
    [ProductStandardCost]
 )
```

In a one-to-many relationship, *RELATED* can access the "one" side from the "many" side because, in that case, only one row in the related table exists, if any. If no row is related with the current one, *RELATED* simply returns *BLANK*. This is different than Excel *VLOOKUP* function, which returns an error if there is no match.

If you are on the "one" side of the relationship and you want to access the "many" side, *RELATED* is not helpful because many rows from the other side might be available for a single row in the current table. In that case, *RELATEDTABLE* will return a table containing all the related rows. For example, if you want to know how many products are in each category, you can create a column in DimProductCategory with this formula:

```
= COUNTROWS ( RELATEDTABLE ( DimProduct ) )
```

This calculated column will show the number of products that are related for each product category, as shown in Figure 3-18.



ProductCategoryKey	EnglishProductCategoryName	NumOfProducts
1	Bikes	125
2	Components	189
3	Clothing	48
4	Accessories	35

FIGURE 3-18 Count the number of products by using *RELATEDTABLE*.

Like *RELATED, RELATEDTABLE* can follow a chain of relationships, always starting from the "one" side and going in the direction of the "many" side.

Using basic DAX functions

Now that you have seen the basics of DAX, it is useful to check your knowledge of developing a sample reporting system. With the limited knowledge you have so far, you cannot develop a very complex solution. Nevertheless, even with a basic set of functions, you can already build something interesting.

Start loading the following tables from *AdventureWorksDW* into a new Excel workbook: DimDate, DimProduct, DimProductCategory, DimProductSubcategory, and FactResellerSales. The resulting data model is shown in Figure 3-19. You can find this workbook under the name "CH03-05-Final Exercise. xlsx."

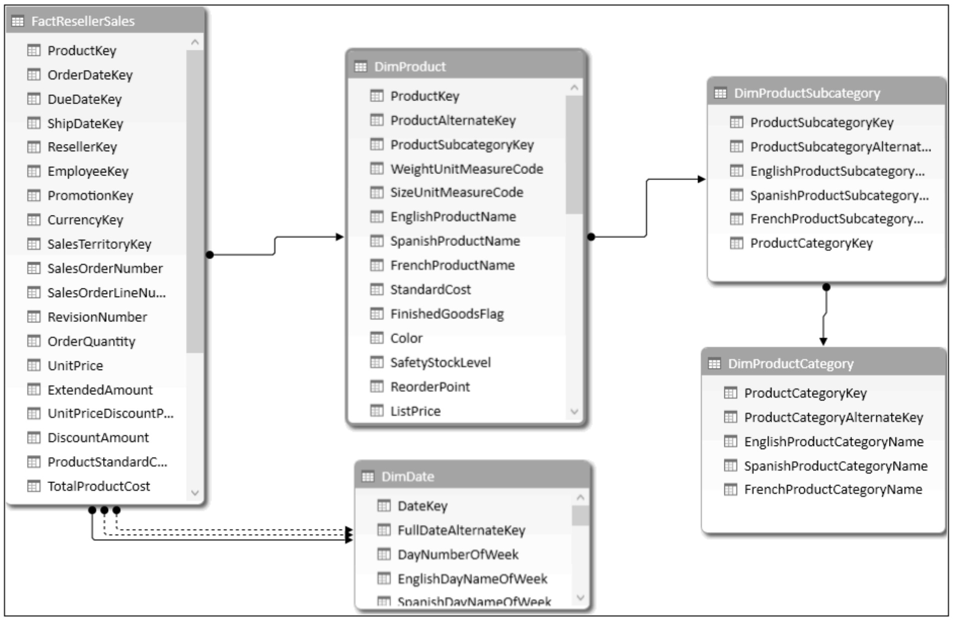


FIGURE 3-19 The Diagram view shows the structure of the demo data model.

To test your new knowledge of the DAX language, let's start solving some reporting problems with this data model.

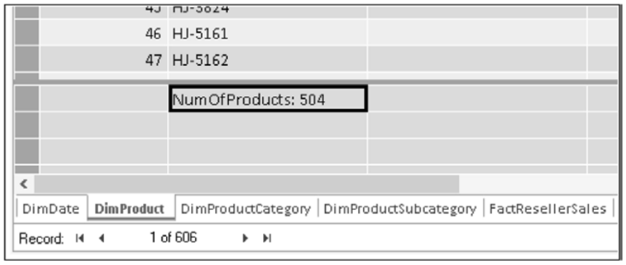
First, count the number of products and enable the user to slice them by category and subcategory, so long as it is with any of the DimProduct columns. It is clear that you cannot rely on calculated columns to perform this task; you need a measure that simply counts the number of products, called NumOfProducts. The code is as follows:

```
NumOfProducts := COUNTROWS ( DimProduct )
```

Although this measure seems very easy to write, there is one problem. Because DimProduct is a slowly changing dimension of type 2 (that is, it can store different versions of the same product to track changes), the same product might appear several times in the table, and you want to count it only once. This common scenario can be solved easily by counting the number of distinct values in the natural key of the table. The natural key of DimProduct is the ProductAlternateKey column. Thus, the correct formula to count the number of products is

```
NumOfProducts := DISTINCTCOUNT ( DimProduct[ProductAlternateKey] )
```

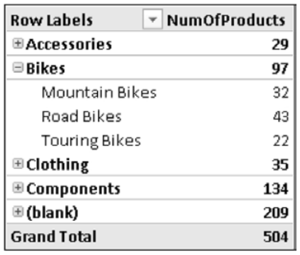
You can see in Figure 3-20 that although the number of rows in the table is 606, the number of products is 504. This number correctly takes into account different versions of the same product, counting them only once.



```
45  HJ-5824
46  HJ-5161
47  HJ-5162
    NumOfProducts: 504
```

FIGURE 3-20 *DISTINCTCOUNT* is a useful and common function for counting.

This measure is very useful and, when browsed through a PivotTable, slicing by category and subcategory produces a report like the one shown in Figure 3-21.



Row Labels	NumOfProducts
⊞ **Accessories**	**29**
⊟ **Bikes**	**97**
Mountain Bikes	32
Road Bikes	43
Touring Bikes	22
⊞ **Clothing**	**35**
⊞ **Components**	**134**
⊞ **(blank)**	**209**
Grand Total	**504**

FIGURE 3-21 A sample report using NumOfProducts.

In this report, the last row is blank because there are products without a category and subcategory. After investigating the data, you discover that many of these uncategorized products are nuts, whereas other products are of no interest. Thus, you decide to override the Category and Subcategory columns with two new columns following this pattern:

- If the category is not empty, then display the category.
- If the category is empty and the product name contains the word “nut,” show “Nuts” for the category and “Nuts” for the subcategory.
- Otherwise, show “Other” as both category and subcategory.

Because you must use these values to slice data, this time you cannot use calculated fields; you must create some calculated columns. You will put these two calculated columns in the DimProduct table and call them ProductCategory and ProductSubcategory. The code is as follows:

```
ProductSubcategory =
    IF (
        ISBLANK ( DimProduct[ProductSubcategoryKey] ),
        IF (
            ISERROR ( FIND ( "Nut", DimProduct[EnglishProductName] ) ),
            "Other",
            "Nut"
        ),
        RELATED ( DimProductSubcategory[EnglishProductSubcategoryName] )
    )
```

This formula is interesting because it uses several of the functions you have just learned. The first *IF* checks whether the ProductSubcategoryKey is empty and, if so, it searches for the word "Nut" inside the product name. *FIND* returns an error if there is no match, which is why you must surround it with the *ISERROR* function, which intercepts the error and enables you to take care of it. If *FIND* returns an error, the result is "Other"; otherwise, the formula computes the subcategory name from the DimProductSubcategory by using the *RELATED* function.

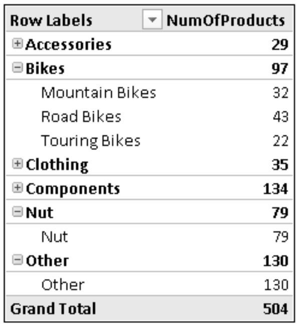
Tip The *ISERROR* function can be slow in such a scenario because it raises errors if it does not find a value. Raising thousands if not millions of errors is a time-consuming operation. In such a case, it is often better to use the fourth parameter of the *FIND* function (which is the default return value if there is no match) to always get a value back, avoiding the error handling. In this formula, we are using *ISERROR* for educational purposes. In a production data model, it is always best to take care of performance speed and use the fourth parameter.

With this calculated column, you have solved the issue with ProductSubcategory. The very same code, by replacing ProductSubcategory with ProductCategory, yields to the second calculated column, which makes the same operation with the category (the difference between the formulas is indicated in bold):

```
ProductCategory =
    IF (
        ISBLANK ( DimProduct[ProductSubcategoryKey] ),
        IF (
            ISERROR ( FIND ( "Nut", DimProduct[EnglishProductName] ) ),
            "Other",
            "Nut"
        ),
        RELATED ( DimProductCategory[EnglishProductCategoryName] )
    )
```

Note that you still must check whether ProductSubcategoryKey is empty because this is the only available column in DimProduct to test if the product has a category. In fact, because of the way data is shaped in *AdventureWorks,* all subcategories have a category, and a product does not have a category if it does not have a subcategory. In different data models, you might face different scenarios, so that, for example, you might need to check `ISBLANK ( RELATED ( DimProductCategory[ProductCategoryKey] ) )`.

If you now browse this new data model in a PivotTable and use the newly created calculated column on the rows, you get the result shown in Figure 3-22.



Row Labels	NumOfProducts
⊞ **Accessories**	**29**
⊟ **Bikes**	**97**
Mountain Bikes	32
Road Bikes	43
Touring Bikes	22
⊞ **Clothing**	**35**
⊞ **Components**	**134**
⊟ **Nut**	**79**
Nut	79
⊟ **Other**	**130**
Other	130
Grand Total	**504**

FIGURE 3-22 You can build a report with the new product category and subcategory.

As you have seen, creating a report and shaping it to fit your analytical needs is easy with a basic knowledge of the DAX language. DAX is the key to many complex reports that you will be able to build with PowerPivot for Excel. You have a long way to go before you become a real DAX master, but the ability to create this report alone, without help from anybody else, is already very rewarding.

Index

Symbols

& (ampersand)
&& (AND) operator, 53, 395
string concatenation operator, 51, 53
* (asterisk), multiplication operator, 53
. (commas), in DAX code, 66
= (equals sign)
equal to operator, 53
in calculated column and calculated field definitions, 67
! (exclamation mark), NOT operator, 53
> (greater than) operator, 53
>= (greater than or equal to) operator, 53
< (less than) operator, 53
>= (less than or equal to) operator, 53
- (minus sign), subtraction/negation operator, 53
<> (not equal to) operator, 53
( ) (parentheses), precedence or grouping operator, 53
+ (plus sign), addition operator, 53
' ' (quotation marks, single) in DAX literals, 53
/ (slash), division operator, 53
∑ symbol before column names, 267
∑ Values field, 38
| (vertical bar), OR operator, 53

A

ABC analysis, 407–411
computing with linked-back table, 435–438
data model to compute ABC classes for products, 408
dynamic, using VBA, 451–455
ABS function, 74
Access, loading data from, 143–144
actions defined in SSAS database, showing, 468
ADDCOLUMNS function, 441
using in DAX queries, 425–427
using VALUES with, 426
AddConnection function, 456
additive measures, 353
additive nature of formula, many-to-many relationships and, 385
Add To Data Model button, 159
AdventureWorks sample database, 5
aggregate functions, 68–71
A-suffixed, 69
changing, 46
using for date calculations, 349–351
X-suffixed, 71
aggregating and comparing over time, 339–352
ALL function, 193
filter context and, 209
parameter of CALCULATE, 212
removal of filters with, 194
using in FILTER expression in CALCULATE, 219
ALLSELECTED function, 230–232
detecting if column is filtered in hierarchy, 246
restoring original filter context of table, 297
using with RANKX, 377
Analysis Services. *See* SSAS
AND function, 71
AND operator, 53
combining filters, 395
conditions for filtering with CALCULATE, 218
TRUE/FALSE arguments in CALCULATE, 221
animation (in a bubble chart), 276
arithmetical operations in DAX, 61
arithmetic operators, 53
Assign Macro dialog box, 449
A-suffixed aggregate functions, 69
.atomsvc file extension, 156
technical information about source data feeds, 168
AVERAGE function, 68

averages, computing moving averages, 363–368
AVERAGEX function, 71, 391
  KEEPFILTERS function with, 399
  VALUES function with, 394
Avg SalesGrowth calculated field, 297
Azure Marketplace. *See* Windows Azure Marketplace

B

banding, 369–374, 433
Bar Chart button, DESIGN tab of ribbon, 263
bar charts, 273. *See also* charts
basket analysis, implementing, 400–407
  data model using many-to-many relationships, 400
BI (Business Intelligence) tools, 463
  features of different tools, 464
BLANK value, 62
  DISTINCTCOUNT function and, 70
blank values, removing from P/C hierarchy, 252
BLOB type in DAX, 50
Booleans, TRUE/FALSE type in DAX, 50, 52
Browser View Options dialog box, 121
bubble charts, 275–277. *See also* charts
buttons, adding to a worksheet, 449

C

calculated columns, 54–55
  ABC analysis, 407–411
  choosing between calculated fields and, 58
  creating in DAX, 37–41
  creating using CALCULATE, 222
  creating YearMonth column for Power View, 303
  differences from calculated fields, 58
  row context, 184
  using in maps to concatenate address, 280
  using SUM function in, 185–186
Calculated Field dialog box, 56, 186
calculated fields, 42, 55–58
  as base field for KPI, 295
  choosing between calculated columns and, 58
  creating with DEFINE, 418
  defining a KPI, 295
  differences from calculated columns, 58
  filtering in Power View Filters pane, 267
  Sum of SalesAmount (example), 180
  using CALCULATE function, 212
  using fields in, 186–188
  using instead of default summarization in Power View, 306
CALCULATE function, 209–232
  ALLSELECTED function, 230–232
  altering the filter context, 211–213
  calculating closing balance over time, 353–362
  calculating monthly average of YTD sales, 351
  circular dependencies, 225–229
  computing difference over previous year, 351–353
  computing moving annual total (MAT), 348
  computing prior year (PY) periods, 346–348
  creating price bands, 371
  currency conversions with, 415
  evaluation contexts and, 179
  examples of use, 213–221
    filtering a column, 213–217
    filtering with complex conditions, 218–221
  forcing context transition, 384
  KEEPFILTERS with, 395–400
  reasons it's needed, 209
  rules for, 229
  time intelligence with, 343–346
  using in row context, 222–225
    context transition, 223–225
  using to calculate average price, 350
  using with ADDCOLUMNS, 426
CALCULATETABLE function, 221, 453
  using in queries, 424
  using on top of FILTER, 405
calculations
  automatic, columns representing, 267
  date calculations in DAX, 323–368
    building a calendar table, 323–339
  performing using hierarchies, 238–248
Calendar hierarchy, 234
calendar tables
  aggregating and comparing over time, 339–352
  building using DAX, 323–339
    characteristics of correct calendar tables, 324
    formulas for Dates table, 325
    working with multiple calandar tables, 326–329
cards
  Card view using Power View metadata, 310
  Card view with two slicers, 272
  using card visualization in Power View, 270
Cartesian product, 439
CEILING function, 74
charts

changing Power View tables into, 264
drill-down feature, 283
inside tile section, filtering by tile, 285
multipliers in, 286
using as filters, 265
using in Power View, 272–277
bar and column charts, 273
line charts, 273
pie charts, 274
scatter charts, 275–277
circular dependencies in formulas, 225–229
Clipboard
loading data from, 165–166
avoiding use of Clipboard as data source, 166
CLOSINGBALANCE functions, 356–360
closing balance over time, 353–362
OPENINGBALANCE and CLOSINGBALANCE functions, 356–360
semiadditive measures, 353–356
updating balances using transactions, 360–362
CLOSINGBALANCEYEAR function, 357
column-based databases, normalization and, 94
column charts, 273. *See also* charts
column filtering, tables before importing, 139
column-oriented databases, 24
column references in DAX code, 66
columns
adding to hierarchies, 235
evaluation context defined by, 183
filtering in Power View Filters pane, 267
filtering single column with CALCULATE, 213–217
naming when using multiple tables, 328
reduction of number returned as result with SUMMARIZE, 430
rollup, 430
column summarization metadata, 305
Comma Separated Values. *See* CSV files
commas (.) in DAX code, 66
comparison operators, 53
CONCATENATE function, 74
concatenation operator (&), 53
Connection Properties dialog box, 178, 460
Connections button, DATA tab of Excel ribbon, 457
connections to data sources
Access database, 144
AddConnection function, 456
credentials for SharePoint data, 132
loading data from existing connection, 418
opening existing connection, 143–144
options for SQL Server database connection, 136
refreshing, 176–178
SQL Server Analysis Services, 146
understanding data connections, 459–462
interactions between Excel and PowerPoint connections, 460
using VBA to modify data model, 462
Windows Azure Marketplace, 175
contexts. *See* evaluation contexts
context transition
AVERAGEX and, 394
forcing with CALCULATE function, 384
RANKX and, 378
using CALCULATE, 223–225
conversion errors in DAX, 60
conversion functions, 75
Convert To Formulas option, 21
COUNTA function, 69
COUNTAX function, 71
COUNTBLANK function, 69
COUNT function, 70
using to count products (example), 87
COUNTROWS function, 69, 190
USERELATIONSHIP function with, 405
using, 79
COUNTX function, 71
Create a New PivotTable dialog, 9
Create PivotTable dialog box, 14
CreatePivotTable function, 448
CreatePivotTable macro, running, 450
Create Relationship button, Design tab of PowerPivot ribbon, 90
Create Relationship dialog box, 90
Create Set Based On Row Items... command, 312
credentials
avoiding storage on SharePoint, 132
Credentials section, Data Refresh configuration, 131
cross-filtering, 203, 383
CROSSJOIN function, 439
CSV (Comma Separated Values) files, 163
adjusting content of, 165
handling more complex CSV files, 164
CUBEMEMBER formulas, 22
CUBEVALUE formulas, 22
currency conversion, 411–416
CURRENCY function, 75
Currency type in DAX, 50
customers, new and returning, computing, 386–390

Customize Ribbon page, Excel Options dialog box, 445

D

dashboards, creating, 19–23
databases
- column-oriented, 24
- in-memory, 25
- loading data from into PowerPivot, 135–142
- logical (user) versus physical (technician) view of, 92
- number of, PowerPivot vs. SSAS, 471
- row-oriented, 23
- row-oriented versus column-oriented, 25
- size, PowerPivot vs. SSAS, 469

data category metadata, 305
data feeds
- data source for PowerPivot, 134
- loading data into PowerPivot, 171–173
- SharePoint libraries exported as, 155
- SQL Server Reporting Services reports as, 167

data filtering. *See* filtering; filters
data loading. *See* loading data
data marts, 110. *See also* data warehouses
- and data loaded from OLAP cubes, 152

data models, 83–116
- building, 88
- building in PowerPivot versus building reports, 44
- computing ABC classes for products, 408
- connecting Excel to SharePoint Excel data model, 125
- creating for Power View, 299–304
  - understanding Power View metadata, 304–310
- creating relationships and adding tables to data model, 434
- creating using SharePoint and PowerPivot, 155
- creating with many tables, 8
- data sources in, 134
- defined, 13
- denormalizing within SQL queries, 95–103
- Excel data model, 241
- for basket analysis, 400
- for currency conversion, 412
- importing data using VBA, 456–459
- interaction between contexts and relationships, 195
- linked tables in, 159
- loading data directly into PowerPivot model, 32
- many-to-many relationships in, 382
- merging data from different tables, 83
- modifying using VBA, 462
- multiple calendar tables in, 327
- normalization and denormalization, 91–94
- of complex solutions, 319
- producing a report without, 84–88
- querying, 14
- refreshing in PowerPivot, 48
- relationships and, 89–91
- report data imported into, 169
- sets and, 314
- Tabular, loading data from, 152
- understanding OLTP and data marts, 106–112
- understanding over-denormalization, 104–106
- using advanced relationships, 112–116

Data Refresh, managing from SharePoint, 129–132
data sources
- limitations for refreshing data, 135
- understanding, 133–135
  - complete list for PowerPivot, 134

data types in DAX, 50–52
- conversion errors, 60
- conversion functions, 75
- Date, 52
- TRUE/FALSE, 52

data warehouses, 106
- choosing between OLTP system and, 112
- ignoring Kimball methodology rules, 113
- Kimball methodology, 110

DATEADD function, 346
date columns
- choosing which to use, 342
- in different tables, 327
- specifying for date table, 341

DATE function, 75, 76
dates and time
- date and time functions, 76
- date calculations in DAX, 323–368
  - aggregating and comparing over time, 339–352
  - building calendar table, 323–339
  - closing balance over time, 353–362
- Date data in DAX, 52
- sets, use with dates, 314

DATESBETWEEN function, 348

DateStream, 175
DATESYTD function, 343, 347
date tables, marking calendar tables as, 340
DATEVALUE function, 76
DAX (Data Analysis eXpression) language, 37–48, 49–82
- advanced DAX, 369–416
  - banding, 369–374
  - calculated columns and ABC analysis, 407–411
  - computing new and returning customers, 386–390
  - currency conversion, 411–416
  - implementing basket analysis, 400–407
  - KEEPFILTERS function, 390–400
  - ranking, 374–381
  - using many-to-many relationships, 382–386
- calculated columns, 54–55
- calculated columns versus calculated fields, 58
- calculated fields, 55–58
- choosing between calculated columns and measures, 58
- common DAX functions, 68–78
  - aggregate functions, 68–71
  - conversion functions, 75
  - information functions, 72
  - logical functions, 71–72
  - mathematical functions, 74
  - relational functions, 77
  - text functions, 74
- creating calculated column, 37–41
- date calculations in, 323–368
  - aggregating and comparing over time, 339–352
  - building a calendar table, 323–339
  - closing balance over time, 353–362
  - computing moving averages, 363–368
- evaluation contexts and, 179
- formatting code, 65–67
- handling errors in DAX expressions, 60–65
  - arithmetical operations, 61
  - conversion errors, 60
  - from empty or missing values, 62
  - intercepting errors, 63
- querying Tabular database, 152
- understanding calculations, 49–54
  - DAX data types, 50–52
  - DAX syntax, 49–50
  - DAX values, 53
  - IntelliSense, 54
- using as query language, 417–444
  - ABC analysis with linked-back table, 435–438
  - ADDCOLUMNS function, 425–427
  - creating Excel table with EVALUATE, 418–423
  - CROSSJOIN function, 439
  - GENERATE function, 440–442
  - linking back a DAX query, 431–435
  - making query dynamic with VBA, 451–455
  - querying with DAX Studio, 443
  - SUMMARIZE function, 427–431
  - understanding EVALUATE, 417
  - using common functions in queries, 423–425
- using basic DAX functions, 78–82

DAX Studio, querying with, 443
DAY function, 76
decorating reports in Power View, 267
default field set metadata, 305
- using, 308

DEFINE function, 418
denormalization. *See* normalization and denormalization
dense or nondense ranking, 381
DESIGN tab of the ribbon
- Map button, 278
- Power View format options, 264
- Slicer button, 272
- Switch Visualization button group, 268

DEVELOPER tab of the ribbon, 445–450
- enabling, 445
- Insert button, adding button to a worksheet, 449
- opening VBA window, 447
- Record Macro button, 446
- Stop Recording, 447
- VBA operations that create PivotTable, 448

Diagram view, 18
- designing hierarchies in, 235
- showing tables belonging to a perspective, 321

dimensions, 110
- in star schema, 111

Distinct Count aggregate function, 46
DISTINCTCOUNT function, 70
- using, 80

DISTINCT function, using instead of VALUES, 427
division by zero, 61
Do Not Summarize option, 307
drill-down, 281–283
drill-through, 322
- comparison of SSAS and PowerPivot for Excel, 467

E

EARLIER function, 191
  second parameter, number of steps to skip, 192
EARLIEST function, 192
EDATE function, 76
Edit DAX dialog box, 421
E-mail Notifications section, Data Refresh configuration, 131
empty or missing values (in DAX), 62
  COUNTBLANK function, 70
end of the fiscal year, 342
ENDOFYEAR function, 357
Enterprise Client Access License, 118
EOMONTH function, 76
errors
  handling in DAX expressions, 60–65
  in calculated fields, 57
  trying to access column name in calculated field, 188
EVALUATE statement, 417
  creating Excel table with, 418–423
evaluation contexts, 179–208
  context transition using CALCULATE, 223–225
  creating a parameter table, 206–208
  creating row context with iterators, 189–192
  defined, 180
  expression inside CALCULATE, 212
  FILTER, ALL, and context interactions, 192–194
  filter and row contexts, 185
  filter context, 183
  recap of important points, 205
  row context, 184–185
  SUMMARIZE function, 430
  testing understanding of, 185–188
    using fields in calculated fields, 186–188
    using SUM in calculated column, 185–186
  working with many tables, 195–204
    filter context and relationships, 198
    ISFILTERED and ISCROSSFILTERED functions, 201–204
    row contexts and relationships, 196–198
    VALUES function, 199–200
EXACT function, 74
Excel
  aggregate functions in, 69
  data model, 6, 241
  empty values in, 63
  loading data from files into PowerPivot, 160–163
  security and, 465
Excel 2010, PowerPivot for Excel 1.0, 5
Excel 2013
  data model, 13
  PowerPivot add-in, 15
Excel Options dialog box, Customize Ribbon page, 445
Excel Services
  document size limit, 119
  Enterprise Client Access License for, 118
  opening document on SharePoint, 120
Existing Connection dialog box, 459
Existing Connections button, 143, 418
Existing Connections dialog box, 419
EXP function, 74
exporting data, SharePoint libraries as data feeds, 155
external sources, loading data from, 31–37

F

FACT function, 74
facts, 110
  in star schema, 111
FALSE function, 71
Field Settings button, ANALYZE tab of ribbon, 86
Field Settings dialog box, 87
Fields, Items, & Sets button, Options ribbon of Excel, 311
fields, using in calculated field, 186–188
file size, SharePoint and PowerPivot Gallery, 124
filter context, 183. *See also* evaluation contexts
  altering by adding or removing filters, 209
  and row context, 185
  AVERAGEX and context transition, 394
  created by SUMMARIZE, 430
  defining set items, 311
  ignoring existing context with ALL function, 193
  in ranking, 377
  intraction with relationships, 198
  ISFILTERED and ISCROSSFILTERED functions, 201–204
  many-to-many relationships and, 383
  modifying with CALCULATE, 211–213
  RANKX and context transition, 378
  recap of important points, 205
  relationships and, 195
  restoring original with ALLSELECTED function, 297

transforming row context into, using CALCULATE, 223
FILTER function, 190
filtering new customers, 389
filtering price bands, 371
understanding FILTER, ALL, and context interactions, 192–194
use as filter parameter in CALCULATE, 217
using ALL with, 219
using CALCULATETABLE with, 221
using in CALCULATE formula, 216
using in queries, 423
ADDCOLUMNS function and, 426
using on complete table in CALCULATE expression, 215
using with EVALUATE, 418
filtering
tables before importing, 138
testing whether column is filtered, 242
using CALCULATE function
filtering on single column, 213–217
filtering with complex conditions, 218–232
filters
charts acting as, 265
evaluation context defined by, 183
KEEPFILTERS function, 390–400
removal with ALL function, 194
tiles as, 284
using Filters pane in Power View, 265–267
application of filters to entire report or part of it, 266
FIND function, 74, 81
FIRSTDATE function, 358
FIRSTNONBLANK function, 356
fiscal year, 342
FIXED function, 74
flexibility, PowerPivot for Excel vs. SSAS, 467
FLOOR function, 74
FORMAT function, 74
using with dates, 76
formulas. *See also* DAX language
circular dependencies in, 225–229
converting PivotTable to, 18–23
definitions for Date table in Excel, 325
evaluation context, 181
filter and row contexts, 185
Excel versus DAX, 37
fomatting in DAX code, 65
IntelliSense help with, 54
FROM clause, SQL queries, 96
From Database button, PowerPivot ribbon, 32, 135
From Analysis Services Or Power Pivot option, 146
From Other Sources button, PowerPivot ribbon, 134
From Text button, 163
FULL OUTER JOIN relationships, 100
functions
common DAX functions, 68–78
aggregate functions, 68–71
conversion functions, 75
information functions, 72–73
logical functions, 71–72
mathematical functions, 74
relational functions, 77
text functions, 74
displaying in VBA window, 448
in DAX code, 67
using basic DAX functions, 78–82
using common functions in queries, 423–425
VBA Model object, 456

G

GENERATE function, using in DAX queries, 440–442
Get External Data From Data Feeds feature, 168
GROUP BY clause in a SELECT statement (SQL), 429
GroupBy columns, 429
growth, analyzing, 290

H

HASONEVALUE function, 208, 344
using with RANKX, 377
Hide From Client Tools, 93
hierarchies, 233–260
building, 235
creating on multiple tables, 236
overview, 233–234
using parent/child (P/C) hierarchies, 248–259
maximum number of levels, 251
using to perform calculations, 238–248
when to build, 235
holidays table, 334
HOUR function, 76

I

identity relationship, 91
IFERROR function, 64, 71
  using with VALUES for price bands, 373
IF function, 64, 71
  example of use, 81
  order or, testing hierarchy lievels, 244
  using nested IF calls, 72
images
  adding to reports in Power View, 267
  pictures in column in Card view, 271
Import Data dialog box, 419
Infinity (special value), 61
information functions, 72–73
in-memory databases, 25
INNER JOIN relationships, 100
INNER type of relationship, 99
Insert Pivot dialog box, 36
integers
  Integer type in DAX, 50
  INT function, 75
IntelliSense, 54
interactive reports, building using tiles in Power View, 285
ISBLANK function, 65, 72
ISCROSSFILTERED function, 201, 203
ISERROR function, 72
  using, 81
ISFILTERED function, 201–204
  detecting hierarchy level, 244
  detecting whether hierarchy Level column is filtered, 254
  inability to detect filter source, 245
  testing whether column is filtered, 242
ISLOGICAL function, 72
ISNONTEXT function, 72
ISNUMBER function, 64, 72
ISTEXT function, 72
iterators, creating row context with, 189–192

J

JOIN predicate (SQL queries), 96
JOINs, types of, 100
  OUTER JOIN and chains of relationships, 102

K

KEEPFILTERS function, 390–400
Keep Unique Rows, flagging column with, 309
Key Performance Indicator (KPI) dialog box, 294
key performance indicators. *See* KPIs
keywords in DAX code, 66
Kimball methodology, 110
  data warehouses not using, 113
KPIs (key performance indicators), 289–299
  creating new KPI, 294
  defined, 294
  defining month as good, bad, or outstanding, 296

L

labels, table default label in Card view, 271
LASTDATE function, 348, 354
  using in closing balance calculations, 357
LASTNONBLANK function, 355, 361
  using, 356
last twelve months (LTM), 348
LAYOUT tab of ribbon, changing map charts, 281
LEFT function, 74
LEFT OUTER JOIN relationships, 100
LEN function, 74
licensing and setup (SharePoint), 118
linear dependencies in formulas, 225
line charts, 273. *See also* charts
  creating in Power View report, 301
linked-back tables, computing ABC analysis with, 435–438
linked tables, 432
  difference from Excel file imports, 163
  limitation of, 160
  using to load data into PowerPivot, 157–160
linking back a DAX query, 431–435
literals, 53
LN function, 74
loading data, 133–178
  from Access, 143–144
    Access query designer in PowerPivot, 145
  from a database, 135–142
    choosing loading method, 137
    loading from list of tables, 137–139
    loading from SQL query, 141
    loading from views, 142

loading relationships, 139
selecting related tables, 140
from Clipboard, 165–166
from data feeds, 171–173
from Excel files into PowerPivot, 160–163
from external source, 31–37
from reports, 167–171
from SharePoint, 153–157
from SQL Server Analysis Services, 146–153
handling of keys in OLAP cube, 150
loading from tabular database, 152
using MDX editor, 148–150
from text files, 163–165
from Windows Azure Marketplace, 173–176
opening existing connections, 143–144
refreshing connections, 176–178
understanding data sources, 133–135
using linked tables, 157–160
using VBA, 456–459
LOCATIONS panel in Power View, 278
drill-down path for map chart, 282
LOG10 function, 74
LOG function, 74
logical functions, 71–72
logical operators, 52, 53
LOOKUPVALUE function, 251
finding original currency of sales, 413
performance and, 414
LOWER function, 74
LTM (last twelve months), 348

M

macros
Assign Macro dialog box, 449
displaying in VBA window, 448
Record Macro button, DEVELOPER tab, 446
running CreatePivotTable macro, 450
Manage Data Refresh button, PowerPivot Gallery, 130
Manage Relationship dialog box, 403
many side of relationship, source table as, 90
many-to-many relationships, 108, 382–386
advanced DAX coding with, 115
DAX formula for, 384
RELATED and RELATEDTABLE functions not working with, 197
resulting from over-denormalization, 116
using in basket analysis data model, 400, 407
white paper on, 385
maps
transforming tables into, 27
using in Power View, 277–281
data model for map demo, 278
drill-down path created in LOCATIONS panel, 282
getting more detailed information, 279
modifying appearance of map charts, 281
using multipliers in, 286
Mark As Date Table button, 340
Mark As Date Table dialog box, 341
mathematical functions, 74
MAT (moving annual total), 348
matrix, using in Power View, 269
MAX function, 68
MAXX function, 71
MDX (Multidimensional Expressions), 22
creating dynamic sets with, 314–318
MDX editor, 147
using, 148–150
measure groups, selecting in MDX query designer, 148
measures, 42. *See also* calculated fields
metadata, Power View, 304–310
on Advanced tab of PowerPivot ribbon, 305
using default field set, 308
using Table Behavior dialog box, 309
Microsoft Office 2013, PowerPivot, 6
Microsoft SQL Server 2012 Business Intelligence, license for, 118
Microsoft Word table, loading into PowerPivot, 165
MID function, 74
MIN function, 68
MINUTE function, 76
MINX function, 71
missing values (in DAX), 62
Model object, 455
useful functions, 456
ModelRelationships function, 456
ModelTables function, 456
MOD function, 74
Modify Set dialog box
designing MDX sets, 315
Recalculate Set With Every Update check box, 317
MONTH function, 76
month-to-date (MTD), 339
MORE TABLES... option, 8
moving annual total (MAT), 348

moving averages, computing, 363–368
MROUND function, 74
MTD (month-to-date), 339
multidimensional databases, loading data from, 133
Multidimensional Expressions. *See* MDX
multipliers, 286

N

naming columns when using multiple tables, 328
NaN (not a number), 61
new and returning customers, computing, 386–390
New Calculated Field option (PowerPivot ribbon), 56
New Set dialog box, 312
NEXTDAY function, 348
nonadditive behavior of formulas with many-to-many relationships, 385
normalization and denormalization, 91–94
- denormalized data in star schema, 111
- denormalizing within SQL queries, 95–103
- frequent cause of over-denormalization, 103
- hiding normalized tables from data model, 93
- many-to-many relationships resulting from over-denormalization, 116
- need for normalization, 94
- turning normalized data model into denormalized one, 92
- understanding over-denormalization, 104–106
- when to denormalize tables, 102

NOT function, 71
NOT operator, 53
NOW function, 76
numbers, conversion to strings, 51

O

OData (Open Data Protocol), 171
- DateStream, 175

Office 2013, PowerPivot, 6
OFFSET function, 437
OLAP cubes, 4
- handling of keys in, 150
- hosted in databases, loading data in PowerPivot, 133
- MDX query language for, 147
- querying in PowerPivot database with MDX, 316

OLAP Tools, 18–23
- button on ANALYZE tab of Excel ribbon, 21

OLTP, 107–112
- deciding wheter to load data from OLTP or data warehouse, 112
- querying OLTP database, 107–109

one side of relationship, target table as, 90
one-to-many relationships in PowerPivot data model, 91
Open Data Protocol (OData), 171
- DateStream, 175

OPENINGBALANCE functions, 356–360
OPENINGBALANCEMONTH function, 358
OPENINGBALANCEQUARTER function, 358
OPENINGBALANCEYEAR function, 358
Open New Excel Workbook button, PowerPivot Gallery, 125
operator overloading, 51
operators in DAX, 52
Options ribbon of Excel, Fields, Items, & Sets button, 311
ORDER BY function, 418, 436
OR function, 71
OR operator, 53
OUTER JOINs, 100
- and chains of relationships, 102

over-denormalization. *See* normalization and denormalization

P

PARALLELPERIOD function, 346
Parameters tab, Publish Options dialog box, 121
parameter tables, creating, 206–208
parent/child (P/C) hierarchies, 248–259
- calculated column indicating if a node is a leaf, 258
- canonical representation of, 249
- computing depth of browsing, 255
- final formula for handling, 258
- final result of browsing with a PivotTable, 250
- functions for handling, 250
- hiding level that is too deep, 254
- maximum number of levels, 251
- unique qualities of, 248
- visualization problems with, 252

Pareto principle, 407
Paste Preview dialog box, 166
PATH function, 250
PATHITEM function, 251
PATHLENGTH function, 254
- computing node depth in P/C hierarchies, 254

P/C hierarchies. *See* parent/child hierarchies
percentages
  computation with ALLSELECTED, 232
  running percentage, 437
perspectives, 318–321
  choosing from PowerPivot for SharePoint model, 321
  on Advanced tab of Power Pivot ribbon, 319
  selecting, 320
Perspectives dialog box, 319
Picture button, Power View tab of ribbon, 267
pie charts, 274. *See also* charts
  multipliers in, 286
PI function, 74
PivotTable
  comparison to PowerPivot, 2
  converting to formulas, 18–23
  Create PivotTable dialog box, 14
  creating in PowerPivot, 35–37
  limitations of, 4
POWER function, 74
PowerPivot
  creating PivotTable, 35–37
  opening main window, 17
  sample data for testing, 5
  understanding PowerPivot for Excel 2013, 23–25
PowerPivot add-in, activating, 15–18
PowerPivot for Excel 2013, 463
  comparison to SSAS, 463–472
    database size, 469
    different versions of engine, 463
    features, 464
    number of databases, 471
    PowerPivot as prototyping system, 472
    programmability and flexibility, 466–468
    security, 465–466
    translations, 468
PowerPivot for SharePoint, 464
PowerPivot Gallery, 118
  connecting Excel to SharePoint Excel data model, 125
  creating Power View report from SharePoint, 128
  Manage Data Refresh button, 130
  using, 123–124
    versioning and file size, 124
PowerPivot query designer, 95–102
Power View, 261–288
  basics of, creating a report, 262–268
    adding columns to existing table or new table, 263
    changing table into bar chart, 263
    decorating a report, 267
    using Filters pane, 265–267
  creating a report, 25–30
  creating data models for, 299–304
    calculated column for year and month, 303
    difference between PivotTable data models and, 304
    line chart, 301
    table as default visualization, 301
  defined, 29, 261
  herarchies in reports, 234
  inability to use sets, 314
  metadata, 304–310
    default field set, 308
    sets of metadata, 305
    using Summarize By, 305–307
    using Table Behavior dialog box, 309
  report, creating for Excel workbook saved in SharePoint, 127–129
  understanding drill-down, 281–283
  understanding multipliers, 286
  understanding table, matrix, and cards, 268–272
    using card visualization, 270
    using matrix visualization, 269
    using table as a slicer, 272
  using charts, 272–277
    line charts, 273
    pie charts, 274
    scatter charts, 275–277
  using effectively, 287
  using maps, 277–281
  using tiles, 283–285
Power View Fields panel, 262
primary keys, 91
prior year. *See* PY
programmability and flexibility, PowerPivot for Excel vs. SSAS, 466
prototyping system, PowerPivot as, 472
Publish Options dialog box, Parameters tab, 121
PY (prior year)
  computing difference over, 351–353
  computing periods from, 346–348

Q

QTD (quarter-to-date), 339
queries. *See* DAX; query designer
query designer, 95–102

intuitive graphical interface, 98
opening and starting query editor, 96
SQL query editor with query written by designer, 99
types of relationships, 100
QUOTIENT function, 74

R

RANDBETWEEN function, 74
RAND function, 74
ranking, 374–381, 433
RANKX and context transition, 378
RANKX function, 374–381
and context transition, 378
descending or ascending sort and dense or nondense ranking, 381
expression evaluated to find the rank, 379
steps completed by, 375
using ALL on Products, 376
Real type in DAX, 50
Record Macro dialog box, 447
Refresh function, 456
refreshing data
database size and, 469
data source limitations for, 135
Excel tables versus linked tables, 160
from connections, 176–178
imported data in PowerPivot, 163
linked-back tables and, 435
PowerPivot data model, 48
PowerPivot report on SharePoint, 129–132
Windows Azure Marketplace data, 176
RELATED function, 77, 205
multiple paths connecting two tables, 334
row contexts and, 196
traversing chain of relationships, 197
using in creating hierarchy, 237
using SQL editor instead of, 95
RELATEDTABLE function, 77, 205
row context on one side of relationships, 196
traversing chain of relationships, 197
relational dartabases, loading data into PowerPivot, 133
relational functions, 77
Relationship pane, query designer, 98
relationships, 89–91
activating with USERELATIONSHIPS, 405
automatic detection of, 12
between linked tables, 159
calendar tables, 326
corrected relationship set in query designer, 101
creating and adding tables to data model, 434
data loading from external source, 34
diagram of, 18
filter and row contexts' beharior regarding, 223
inactivating, 402
in query designer, 98
in star schema, 111
interaction with evaluation contexts, 195, 205
filter context, 198
row contexts, 196–198
loading for tables imported into PowerPivot, 139
many-to-many relationships, 108
ModelRelationships function, 456
OUTER JOIN and chains of relationships, 102
selecting related tables in Table Import Wizard, 140
self-referencing, 249
types in query designer, 99, 100
understanding, 11
using advanced relationships, 112–116
versus using VLOOKUP, 84
warning about, 10
REPLACE function, 74
Reporting Services in SharePoint mode, 128
reports
data from, loading into PowerPivot, 167–171
report path names, 171
shaping, 289–322
creating data models for Power View, 299–304
creating dynamic sets with MDX, 314–318
defining sets, 310–314
drill-through, 322
KPIs (key performance indicators), 289–299
Power View metadata, 304–310
using perspectives, 318–321
REPT function, 74
returning and new customers, filtering, 386–390
reverse-linked tables, 432
reverse-linking PowerPivot table in Excel, 418–420
RIGHT function, 74
RIGHT OUTER JOIN relationships, 100
ROLLUP option with SUMMARIZE function, 430
ROUNDDOWN function, 74
ROUND function, 74
rounding functions, 74
ROUNDUP function, 74

row context, 184–185, 209. *See also* evaluation contexts
  and temporary columns created by functions, 426
  creating with iterators, 189–192
  interaction with relationships, 196–198
  no row context in calculated fields, 187
  problems with, VALUES function used with RANKX, 380
  recap of important points, 205
  relationships and, 195
  retrieving value of previous context, 191
  transformation by CALCULATE, 211
  using CALCULATE in, 222–225
Row Identifier property, updating in Table Behavior window, 228
Row Identifier setting, Table Behavior dialog box, 309
row-oriented databases, 23
rows
  COUNTROWS function, 69
  creating sets based on row items, 312
  evaluation context defined by, 183
  Keep Unique Rows, column flagged with, 309
  limits for tables in Excel and PowerPivot, 421
running totals, computing, 409

S

SAMEPERIODLASTYEAR function, 291, 346–348, 348
scatter charts, 275–277. *See also* charts
Schedule Details section, Data Refresh configuration, 131
SEARCH function, 74
SECOND function, 76
security
  comparison of PowerPivot for Excel and SSAS, 465
  Security Notice window, .atomsvc connection created by SharePoint, 156
SELECT command, SQL queries, 96
Select Related Tables button, Table Import Wizard, 140
Select Table window, 168
self-referencing relationships, not supported by PowerPivot, 249
semiadditive measures, 353–356
Set Manager dialog box, creating MDX sets, 315
sets
  defining, 310–314
  dynamic, creating with MDX, 314–318
  inability to use in Power View, 314
SharePoint
  connecting Excel to SharePoint Excel data model, 125
  licensing and setup, 118
  loading data from, 153–157
  PowerPivot for SharePoint, 464
    as prototyping system, 472
    database size and refreshes, 469
    features, 464
    number of databases, 471
    security, 465
  PowerPivot for SharePoint model, choosing perspectives, 321
  publising a workbook to, 118–123
    creating Power View report, 127–129
    managaing PowerPivot data refresh, 129–132
    Parameters tab, Publish Options dialog box, 121
    saving a document to SharePoint, 119
  SharePoint 2013 and PowerPivot integration, 117
Show Details option for drill-through, 322
SIGN function, 74
Silverlight, requirement for Power View, 26
slicers
  evaluation context defined by, 183
  numeric columns used as, 307
  using table as slicer in Power View, 272–273
sorting, price bands, 372
SQL Server Analysis Services. *See* SSAS
SQL Server Reporting Services, loading data into PowerPivot, 167
SQL (Structured Query Language)
  Access query designer, 145
  denormalizing within SQL queries, 95–103
    example of simple query, 95
    when to denormalize tables, 102
  GROUP BY clause in SELECT statement, 429
  loading data into PowerPivot from SQL query, 141
  use of SQL queries when starting data model is complex, 108
SQRT function, 74
SSAS (SQL Server Analysis Services), 4
  comparison to PowerPivot in Excel, 463–472
    database size, 469
    different versions of engine, 463
    features, 464

number of databases, 471
PowerPivot as prototyping system, 472
programmability and flexibility, 466–468
security, 465
translations, 468
engine in Excel, 241
Excel workbook saved to SharePoint, 126
loading data into PowerPivot from, 146–153
SQL Server Analysis Services 2012 Tabular, 464
star schemas, 111
START AT function, 418
STARTOFMONTH function, 358
STARTOFQUARTER function, 358
STARTOFYEAR function, 358
strategic analysis of data, 290
string concatenation operator (&), 51, 53
strings
conversion to numbers in PowerPivot, 51
String type in DAX, 50
SUBSTITUTE function, 74
Suggest Related Data data source, using for Windows Azure Marketplace, 176
SUM function, 68
additive measures, 353
using in CALCULATE, 224
using in calculated column, 185–186
Summarize By feature, 305–307
Summarize By Property option, 307
SUMMARIZE function
temporary columns created by, 426
using in DAX queries, 427–431
ROLLUP option, 430
with GENERATE, 442
SUMX function, 71, 189
calculating sales in original currency, 413
computing running total of sales, 409
SWITCH function, 71, 72
Switch Visualization button group, DESIGN tab of ribbon, 268

T

Table Behavior dialog box, 271, 309
table behavior metadata, 305
Table Behavior window, updating RowIdentifier property, 228
Table Import Wizard, 32, 134
data connections, 462
DAX query in, 152
for Microsoft SQL Server, 136
loading data from Excel files, 161
loading data from flat files, 164
loading data from OData feeds, 172
loading data from Windows Azure Marketplace, 174
loading from Access, 144
loading report data into PowerPivot, 169
Preview & Filter button, 138
Select Related Tables button, 140
SQL Server Analysis Services connection, 147
Table Properties button, 100
tables
creating Excel table with EVALUATE, 418–423
default visualization in Power View, 301
in Power View, changing to matrix or cards, 268
inside tile section, filtering by tile, 285
linked and reverse-linked, 432
ModelTables function, 456
using as slicer in Power View, 272–273
tabular databases, loading data from, 152
Test Connection button, 136
text files
data sources for PowerPivot, 134
loading data from, 163–165
text functions, 74
TILE BY panel in Power View, 283
tiles
inability to mix with multipliers, 286
using in Power View, 283–285
time. *See* dates and time
TIME function, 75, 76
time intelligence functions
Date column, 324
Mark As Date Table and, 340
reflecting fiscal year, 342
using CALCULATE, 343–346
time interval of data refresh operations, 131
TIMEVALUE function, 76
TODAY function, 76
Top 10 Filter dialog box, 397
TopCount function, 316
TOPN function, 432
using in GENERATE query, 441
TOTALMTD function, 342
TOTALQTD function, 342
TOTALYTD function, 340–343
transactions, updating balances with, 360–362
translations, 468
TRIM function, 74

TRUE/FALSE (Boolean) types in DAX, 50, 52
  conversion of value to a number, 254
TRUE/FALSE conditions in CALCULATE, 215, 217
  changing filter into FILTER equivalent, 221
TRUE function, 71
TRUNC function, 74

U

UPPER function, 74
URLs
  for data feeds, 171
  for Reporting Services reports, 167, 171
user credentials, 131
USERELATIONSHIP function, using with COUNTROWS, 405

V

Value Field Settings, 46
Value Field Settings dialog box, 239
VALUE function, 74, 75
VALUES function, 199–200, 205, 208
  computing band name, 371
  computing each day of a range, 366
  restricting filter to current year, 298
  used with RANKX, problems with row context, 380
  using IFERROR with, 373
  using with ADDCOLUMNS, 426
  using with AVERAGEX, 394
values (in DAX), 53
VBA (Visual Basic for Applications), automating operations with, 445–462
  enabling DEVELOPER tab of ribbon, 445–450
    making a query dynamic, 451–455
  importing data into data model, 456–459
  understanding data connections, 459–462
  using Model object, 455
versioning of files, SharePoint and PowerPivot Gallery, 124
VertiPaq database, 463
views, loading data from, 142
Visual Basic for Applications. *See* VBA, automating operations with
VLOOKUP function, 7, 84
  comparison to relationships, 11
  LOOKUPVALUE and, 413

W

WEEKDAY function, 76
week number in a year, calculating, 325
WEEKNUM function, 76
WHERE clause, SQL queries, 96
Windows Azure Marketplace
  data from, loading into PowerPivot, 173–176
    using Suggest Related Data, 176
Windows Live account ID, 174
Word, loading table into PowerPivot, 165
Workbook Connections dialog box, 177, 458, 460
working days, calculating, 329–339
  computing difference in working days, 335–339
  in different countries, 332
worksheets, importation from Excel files, 162

X

X-suffixed aggregate functions, 71
xVelocity engine, 23, 463

Y

YEARFRAC function, 76
YEAR function, 76
year-over-year (YOY) comparisons, 351–353
year-to-date. *See* YTD
YOY (year-over-year) comparisons, 351–353
YTD (year-to-date)
  CALCULATE version of, 343
  calculating for previous year, 346–348
  calculating with TOTALYTD, 339–343

About the Authors

Alberto Ferrari (*alberto.ferrari@sqlbi.com*) and **Marco Russo** (*marco.russo@sqlbi.com*) are the two founders of SQLBI.COM, where they regularly publish articles about Microsoft PowerPivot, DAX, and SQL Server Analysis Services Tabular. They have worked with PowerPivot since the first beta version in 2009.

They both provide consultancy and mentoring on business intelligence (BI), with a particular specialization in the Microsoft technologies related to BI. They have written several books and papers about these topics, with a particular mention of "SQLBI methodology," which is a complete methodology for designing and implementing the back end of a BI solution; and "The Many-to-Many Revolution," which is a paper dedicated to modeling patterns using many-to-many dimension relationships in SQL Server Analysis Services and PowerPivot.

Marco and Alberto are also regular speakers at major international conferences, such as TechEd, PASS Summit, SQLRally, and SQLBits.