Microsoft

Microsoft® SQL Server® 2012 Analysis Services

The BISM Tabular Model

Marco Russo
Alberto Ferrari
Chris Webb

Microsoft® SQL Server® 2012 Analysis Services The BISM Tabular Model

Build agile and responsive Business Intelligence solutions

Analyze tabular data using the BI Semantic Model (BISM) in SQL Server 2012 Analysis Services—and discover a simpler method for creating corporate-level BI solutions. Led by three BI experts, you'll learn how to build, deploy, and query a BISM tabular model with step-by-step guides, examples, and best practices. This hands-on book shows you how the tabular model's in-memory database enables you to perform rapid analytics—whether you're a professional BI developer new to Analysis Services or already familiar with its multidimensional model.

Discover how to:

- Determine when a tabular or multidimensional model is right for your project
- Build a tabular model using SQL Server Data Tools in Microsoft Visual Studio® 2010
- Integrate data from multiple sources into a single, coherent view of company information
- Use the Data Analysis eXpressions (DAX) language to create calculated columns, measures, and queries
- Choose a data modeling technique that meets your organization's performance and usability requirements
- Optimize your data model for better performance with xVelocity storage engine
- Manage complex data relationships, such as multicolumn, banding, and many-to-many
- Implement security by establishing administrative and data user roles

Get code and project samples on the web
Ready to download at
http://go.microsoft.com/FWLink/?Linkid=254183

*For **system requirements**, see the Introduction.*

microsoft.com/mspress

ISBN: 978-0-7356-5818-9

9 780735 658189 90000

U.S.A. $59.99
Canada $62.99
[*Recommended*]

Databases/Microsoft SQL Server

About the Authors

Marco Russo, MCT, is a BI consultant and mentor who advises clients on the BI lifecycle and relational and multidimensional design of data warehouses.

Alberto Ferrari is a BI consultant who focuses on a methodological approach to the BI development lifecycle, as well as performance tuning of SQL and ETL code.

Chris Webb, MVP for SQL Server, is an independent consultant and trainer specializing in Microsoft SQL Server Analysis Services, MDX, and DAX.

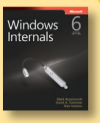
DEVELOPER ROADMAP

Start Here!
- Beginner-level instruction
- Easy to follow explanations and examples
- Exercises to build your first projects

Step by Step
- For experienced developers learning a new topic
- Focus on fundamental techniques and tools
- Hands-on tutorial with practice files plus eBook

Developer Reference
- Professional developers; intermediate to advanced
- Expertly covers essential topics and techniques
- Features extensive, adaptable code examples

Focused Topics
- For programmers who develop complex or advanced solutions
- Specialized topics; narrow focus; deep coverage
- Features extensive, adaptable code examples



Microsoft®

Microsoft®

Microsoft® SQL Server® 2012 Analysis Services: The BISM Tabular Model

Marco Russo
Alberto Ferrari
Chris Webb

ISBN: 978-0-7356-5818-9

2 3 4 5 6 7 8 9 10 LSI 7 6 5 4 3 2

Printed and bound in the United States of America.

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at *mspinput@microsoft.com*. Please tell us what you think of this book at *http://www.microsoft.com/learning/booksurvey*.

Acquisitions and Developmental Editor: Russell Jones

Production Editor: Holly Bauer

Editorial Production: nSight, Inc.

Technical Reviewers: Darren Gosbell and John Mueller

Copyeditor: Kerin Forsyth / Ann Weaver

Indexer: Nancy Guenther

Cover Design: Twist Creative • Seattle

Cover Composition: Karen Montgomery

Illustrator: nSight, Inc.

[2012-11-02]

To the many BI communities that have supported me in the last years.

—Marco Russo

I dedicate this book to Caterina, Lorenzo, and Arianna: my family.

—Alberto Ferrari

I dedicate this book to my wife, Helen, and my two daughters, Natasha and Mimi. Thank you for your love, understanding, and patience.

—Chris Webb

Contents at a Glance

	Foreword	*xix*
	Introduction	*xxi*
CHAPTER 1	**Introducing the Tabular Model**	**1**
CHAPTER 2	**Getting Started with the Tabular Model**	**19**
CHAPTER 3	**Loading Data Inside Tabular**	**75**
CHAPTER 4	**DAX Basics**	**121**
CHAPTER 5	**Understanding Evaluation Context**	**147**
CHAPTER 6	**Querying Tabular**	**185**
CHAPTER 7	**DAX Advanced**	**237**
CHAPTER 8	**Understanding Time Intelligence in DAX**	**291**
CHAPTER 9	**Understanding xVelocity and DirectQuery**	**329**
CHAPTER 10	**Building Hierarchies**	**361**
CHAPTER 11	**Data Modeling in Tabular**	**381**
CHAPTER 12	**Using Advanced Tabular Relationships**	**407**
CHAPTER 13	**The Tabular Presentation Layer**	**429**
CHAPTER 14	**Tabular and PowerPivot**	**449**
CHAPTER 15	**Security**	**463**
CHAPTER 16	**Interfacing with Tabular**	**487**
CHAPTER 17	**Tabular Deployment**	**513**
CHAPTER 18	**Optimizations and Monitoring**	**559**
APPENDIX A	**DAX Functions Reference**	**589**
	Index	*601*

Contents

Foreword . . . . . *xix*
Introduction . . . . . *xxi*

Chapter 1 Introducing the Tabular Model 1

The Microsoft BI Ecosystem 1
What Is Analysis Services and Why Should I Use It? 1
A Short History of Analysis Services 2
The Microsoft BI Stack Today 3
Self-Service BI and Corporate BI 4
Analysis Services 2012 Architecture: One Product, Two Models 6
The Tabular Model 6
The Multidimensional Model 8
Why Have Two Models? 9
The Future of Analysis Services 10
Choosing the Right Model for Your Project 11
Licensing 11
Upgrading from Previous Versions of Analysis Services 12
Ease of Use 12
Compatibility with PowerPivot 12
Query Performance Characteristics 13
Processing Performance Characteristics 13
Hardware Considerations 13
Real-Time BI 14
Client Tools 15
Feature Comparison 15
Summary 17

Chapter 2 Getting Started with the Tabular Model 19

Setting Up a Development Environment 19
Components of a Development Environment 19
Licensing 21
Installation Process 21
Working with SQL Server Data Tools 31
Creating a New Project 31
Configuring a New Project 33
Importing from PowerPivot 37
Importing a Deployed Project from Analysis Services 38
Contents of a Tabular Project 38
Building a Simple Tabular Model 40
Loading Data into Tables 41
Working in the Diagram View 49
Deployment 52
Querying a Tabular Model in Excel 53
Connecting to a Tabular Model 54
Querying a Tabular Model in Power View 65
Creating a Connection to a Tabular Model 65
Building a Basic Power View Report 66
Adding Charts and Slicers 68
Interacting with a Report 69
Working with SQL Server Management Studio 71
Summary 74

Chapter 3 Loading Data Inside Tabular 75

Understanding Data Sources 75
Understanding Impersonation 77
Understanding Server-Side and Client-Side Credentials 78
Working with Big Tables 79
Loading from SQL Server 80
Loading from a List of Tables 83
Loading Relationships 86

Loading from a SQL Query 87
Loading from Views 87
Opening Existing Connections 88
Loading from Access 89
Loading from Analysis Services 90
Using the MDX Editor 92
Loading from a Tabular Database 92
Loading from an Excel File 95
Loading from a Text File 98
Loading from the Clipboard 100
Loading from a Reporting Services Report 103
Loading Reports by Using Data Feeds 108
Loading from a Data Feed 110
Loading from SharePoint 112
Loading from the Windows Azure DataMarket 113
Choosing the Right Data-Loading Method 116
Understanding Why Sorting Data Is Important 118
Summary 119

Chapter 4 DAX Basics 121

Understanding Calculation in DAX 121
DAX Syntax 121
DAX Data Types 123
DAX Operators 124
DAX Values 125
Understanding Calculated Columns and Measures 125
Calculated Columns 126
Measures 126
Editing Measures by Using DAX Editor 129
Choosing Between Calculated Columns and Measures 130
Handling Errors in DAX Expressions 131
Conversion Errors 131

Arithmetical Operation Errors 132
Empty or Missing Values 133
Intercepting Errors 134
Common DAX Functions 135
Aggregate Functions 135
Logical Functions 137
Information Functions 138
Mathematical Functions 139
Text Functions 140
Conversion Functions 140
Date and Time Functions 140
Relational Functions 141
Using Basic DAX Functions 142
Summary 146

Chapter 5 Understanding Evaluation Context 147

Evaluation Context in a Single Table 147
Filter Context in a Single Table 148
Row Context in a Single Table 151
Working with Evaluation Context for a Single Table 157
Understanding the *EARLIER* Function 161
Understanding Evaluation Context in Multiple Tables 164
Row Context with Multiple Tables 164
Understanding Row Context and Chained Relationships 167
Using Filter Context with Multiple Tables 168
Understanding Row and Filter Context Interactions 173
Modifying Filter Context for Multiple Tables 177
Final Considerations for Evaluation Context 183
Summary 183

Chapter 6 Querying Tabular 185

Tools for Querying Tabular 185
DAX Query Syntax 187

Using *CALCULATETABLE* and *FILTER* *189*
Using *ADDCOLUMNS* *192*
Using *SUMMARIZE* *194*
Using *CROSSJOIN*, *GENERATE*, and *GENERATEALL* *203*
Using *ROW* *208*
Using *CONTAINS* *209*
Using *LOOKUPVALUE* *211*
Defining Measures Inside a Query 213
Test Your Measures with a Query 216
Parameters in DAX Query 217
Using DAX Query in SQL Server Reporting Services 219
Querying by Using MDX 223
Using DAX Local Measures in MDX Queries 229
Drillthrough in MDX Queries 230
Choosing Between DAX and MDX 233
Summary 235

Chapter 7 DAX Advanced 237

Understanding *CALCULATE* and *CALCULATETABLE* Functions 237
Evaluation Context in DAX Queries 238
Modifying Filter Context by Using *CALCULATETABLE* *240*
Using *FILTER* in *CALCULATE* and *CALCULATETABLE* Arguments . . 244
Recap of *CALCULATE* and *CALCULATETABLE* Behavior 252
Control Filters and Selections 252
Using *ALLSELECTED* for Visual Totals 253
Filters and Cross Filters 257
Maintaining Complex Filters by Using *KEEPFILTERS* *267*
Sorting Functions 272
Using *TOPN* *272*
Using *RANKX* *276*
Using *RANK.EQ* *284*

Statistical Functions 285
Standard Deviation and Variance by Using *STDEV* and *VAR* *285*
Sampling by Using the *SAMPLE* Function 287
Summary. 290

Chapter 8 Understanding Time Intelligence in DAX 291

Tabular Modeling with Date Table. 291
Creating a Date Table 292
Defining Relationship with Date Tables 296
Duplicating the Date Table. 302
Setting Metadata for a Date Table 306
Time Intelligence Functions in DAX. 307
Aggregating and Comparing over Time 307
Semiadditive Measures 321
Summary. 328

Chapter 9 Understanding xVelocity and DirectQuery 329

Tabular Model Architecture in Analysis Services 2012. 329
In-Memory Mode and xVelocity 331
Query Execution in In-Memory Mode. 331
Row-Oriented vs. Column-Oriented Databases. 334
xVelocity (VertiPaq) Storage 337
Memory Usage in xVelocity (VertiPaq) 339
Optimizing Performance by Reducing Memory Usage 342
Understanding Processing Options 348
Using DirectQuery and Hybrid Modes 351
DirectQuery Mode. 352
Analyzing DirectQuery Mode Events by Using SQL Profiler 354
DirectQuery Settings. 355
Development by Using DirectQuery 359
Summary. 360

Chapter 10 Building Hierarchies 361

Basic Hierarchies 361
What Are Hierarchies? 361
When to Build Hierarchies 363
Building Hierarchies 363
Hierarchy Design Best Practices 364
Hierarchies Spanning Multiple Tables 365

Parent/Child Hierarchies 367
What Are Parent/Child Hierarchies? 367
Configuring Parent/Child Hierarchies 368
Unary Operators 373

Summary 380

Chapter 11 Data Modeling in Tabular 381

Understanding Different Data Modeling Techniques 381
Using the OLTP Database 383

Working with Dimensional Models 384
Working with Slowly Changing Dimensions 386
Working with Degenerate Dimensions 389
Using Snapshot Fact Tables 390

Computing Weighted Aggregations 393

Understanding Circular Dependencies 396

Understanding the Power of Calculated Columns: ABC Analysis 399

Modeling with DirectQuery Enabled 403

Using Views to Decouple from the Database 405

Summary 406

Chapter 12 Using Advanced Tabular Relationships 407

Using Multicolumn Relationships 407

Banding in Tabular 410

Using Many-to-Many Relationships 412

Implementing Basket Analysis 417
Querying Data Models with Advanced Relationships 421
Implementing Currency Conversion 425
Summary 428

Chapter 13 The Tabular Presentation Layer 429

Naming, Sorting, and Formatting 429
Naming Objects 429
Hiding Columns 431
Organizing Measures 432
Sorting Column Data 432
Formatting 436
Perspectives 438
Power View–Related Properties 440
Default Field Set 441
Table Behavior Properties 442
Drillthrough 444
KPIs 445
Summary 448

Chapter 14 Tabular and PowerPivot 449

PowerPivot for Microsoft Excel 2010 449
Using the PowerPivot Field List 452
Understanding Linked Tables 455
PowerPivot for Microsoft SharePoint 455
Using the Right Tool for the Job 458
Prototyping in PowerPivot, Deploying with Tabular 460
Summary 461

Chapter 15 Security 463

Roles 463
Creating Database Roles 464
Membership of Multiple Roles 466

Administrative Security 466
The Server Administrator Role 466
Database Roles and Administrative Permissions 468
Data Security 469
Basic Data Security 469
Testing Data Security 471
Advanced Row Filter Expressions 474
Dynamic Security 479
DAX Functions for Dynamic Security 479
Implementing Dynamic Security by Using *CUSTOMDATA* *480*
Implementing Dynamic Security by Using *USERNAME* *481*
Advanced Authentication Scenarios 482
Connecting to Analysis Services from Outside a Domain 482
Kerberos and the Double-Hop Problem 483
Monitoring Security 484
Summary 486

Chapter 16 Interfacing with Tabular 487

Understanding Different Tabular Interfaces 488
Understanding Tabular vs. Multidimensional Conversion 488
Using AMO from .NET 491
Writing a Complete AMO Application 494
Creating Data Source Views 494
Creating a Cube 495
Loading a SQL Server Table 495
Creating a Measure 498
Creating a Calculated Column 500
Creating Relationships 501
Drawing Some Conclusions 506
Performing Common Operations in AMO with .NET 507
Processing an Object 507
Working with Partitions 508

Using AMO with PowerShell 509
Using XMLA Commands 510
CSDL Extensions 512
Summary 512

Chapter 17 Tabular Deployment 513

Sizing the Server Correctly 513
xVelocity Requirements 513
DirectQuery Requirements 517
Automating Deployment to a Production Server 517
Table Partitioning 518
Defining a Partitioning Strategy 518
Defining Partitions for a Table in a Tabular Model 520
Managing Partitions for a Table 524
Processing Options 527
Available Processing Options 528
Defining a Processing Strategy 532
Executing Processing 535
Processing Automation 539
Using XMLA 539
Using AMO 545
Using PowerShell 546
Using SSIS 547
DirectQuery Deployment 551
Define a DirectQuery Partitioning Strategy 551
Implementing Partitions for DirectQuery and Hybrid Modes 552
Security and Impersonation with DirectQuery 557
Summary 558

Chapter 18 Optimizations and Monitoring 559

Finding the Analysis Services Process 559
Understanding Memory Configuration 561
Using Memory-Related Performance Counters 564

Understanding Query Plans 569
Understanding *SUMX* *575*
Gathering Time Information from the Profiler 577
Common Optimization Techniques 578
Currency Conversion 578
Applying Filters in the Right Place 580
Using Relationships Whenever Possible 582
Monitoring MDX Queries 584
Monitoring DirectQuery 585
Gathering Information by Using Dynamic Management Views 585
Summary 587

Appendix A DAX Functions Reference 589

Statistical Functions 589
Table Transformation Functions 591
Logical Functions 591
Information Functions 592
Mathematical Functions 593
Text Functions 594
Date and Time Functions 595
Filter and Value Functions 597
Time Intelligence Functions 598

Index *601*

About the Authors *627*

What do you think of this book? We want to hear from you!

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

microsoft.com/learning/booksurvey

Foreword

I have known Marco Russo, Alberto Ferrari, and Chris Webb for many years through my work on the Analysis Services product team. Early on, these authors were among the first to embrace multidimensional modeling and offered their insights and suggestions as valued partners to help us make the product even better. When we introduced tabular modeling in SQL Server 2012, the authors were on board from the start, participating in early reviews and applying their substantial skills to this new technology. Marco, Alberto, and Chris have been instrumental in helping to shape the product design and direction, and we are deeply grateful for their contributions.

The authors are truly among the best and brightest in the industry. Individually and collectively, they have authored many books. *Expert Cube Development with Microsoft SQL Server 2008 Analysis Services* notably stands out as a must-have book for understanding multidimensional modeling in Analysis Services. In addition to writing amazing books, you can often find Marco, Alberto, and Chris speaking at key conferences, running training courses, and consulting for companies who are applying business intelligence to improve organizational performance. These authors are at the top of their field; their blogs come up first in the search list for almost any query you might have related to building business intelligence applications.

The book you have in your hands describes ways to build business intelligence applications in detail, using DAX and tabular models. But what truly sets this book apart is its practical advice. This is a book that only seasoned BI practitioners could write. It is a great blend of the information you need the most: an all-up guide to tabular modeling, balanced with sensible advice to guide you through common modeling decisions. I hope you enjoy this book as much as I do. I'm sure it will become an essential resource that you keep close at hand whenever you work on tabular models.

Edward Melomed
Program Manager
SQL Server Analysis Services

Introduction

When we, the authors of this book, first learned what Microsoft's plans were for Analysis Services in the SQL Server 2012 release, we were not happy. Analysis Services hadn't acquired much in the way of new features since 2005, even though in the meantime it had grown to become the biggest-selling OLAP tool. It seemed as if Microsoft had lost interest in the product. The release of PowerPivot and all the hype surrounding self-service Business Intelligence (BI) suggested that Microsoft was no longer interested in traditional corporate BI, or even that Microsoft thought professional BI developers were irrelevant in a world where end users could build their own BI applications directly in Excel. Then, when Microsoft announced that the technology underpinning PowerPivot was to be rolled into Analysis Services, it seemed as if all our worst fears had come true: the richness of the multidimensional model was being abandoned in favor of a dumbed-down, table-based approach; a mature product was being replaced with a version 1.0 that was missing a lot of useful functionality. Fortunately, we were proven wrong and as we started using the first CTPs of the new release, a much more positive—if complex—picture emerged.

SQL Server 2012 is undoubtedly a milestone release for Analysis Services. Despite all the rumors to the contrary, we can say emphatically that Analysis Services is neither dead nor dying; instead, it's metamorphosing into something new and even more powerful. As this change takes place, Analysis Services will be a two-headed beast—almost two separate products (albeit ones that share a lot of the same code). The Analysis Services of cubes and dimensions familiar to many people from previous releases will become known as the "Multidimensional Model," while the new, PowerPivot-like flavor of Analysis Services will be known as the "Tabular Model." These two models have different strengths and weaknesses and are appropriate for different projects. The Tabular Model (which, from here onward, we'll refer to as simply Tabular) does not replace the Multidimensional Model. Tabular is not "better" or "worse" than Multidimensional. Instead, the Tabular and Multidimensional models complement each other well. Despite our deep and long-standing attachment to Multidimensional, Tabular has impressed us because not only is it blindingly fast, but because its simplicity will bring BI to a whole new audience.

In this book we'll be focusing exclusively on Tabular for two reasons. First, there's not much that's new in the Multidimensional Model, so books written for previous versions of Analysis Services will still be relevant. Second, if you're using Analysis Services on a project, you'll have to make a decision early on about which of the two models to use—and it's very unlikely you'll use both. That means anyone who decides to use Tabular is

unlikely to be interested in reading about the Multidimensional Model anyway. One of the first things we'll do in this book is to give you all the information you need to make the decision about which model to use.

We have enjoyed learning about and writing about Tabular and we hope you enjoy reading this book.

Who Should Read This Book

This book is aimed at professional Business Intelligence developers: consultants or members of in-house BI development teams who are about to embark on a project using the Tabular Model.

Assumptions

Although we're going to start with the basics of Tabular—so in a sense this is an introductory book—we're going to assume that you already know certain core BI concepts such as dimensional modeling and data warehouse design. Some previous knowledge of relational databases, and especially SQL Server, will be important when it comes to understanding how Tabular is structured and how to load data into it and for topics such as DirectQuery.

Previous experience with Analysis Services Multidimensional isn't necessary, but because we know most readers of this book will have some we will occasionally refer to its features and compare them with equivalent features in Tabular.

Who Should Not Read This Book

No book is suitable for every possible audience, and this book is no exception. Those without any existing business intelligence experience will find themselves out of their depth very quickly, as will managers who do not have a technical background.

Organization of This Book

This book is organized as follows: In the first chapter we will introduce the Tabular Model, what it is and when it should (and shouldn't) be used. In Chapters 2 and 3 we will cover the basics of building a Tabular Model. In Chapters 4 through 8 we'll

introduce DAX, its concepts, syntax and functions, and how to use it to create calculated columns, measures, and queries. Chapters 9 through 16 will deal with numerous Tabular design topics such as hierarchies, relationships, many-to-many, and security. Finally, Chapters 17 and 18 will deal with operational issues such as hardware sizing and configuration, optimization, and monitoring.

Conventions and Features in This Book

This book presents information using conventions designed to make the information readable and easy to follow:

- Boxed elements with labels such as "Note" provide additional information or alternative methods for completing a step successfully.
- Text that you type (apart from code blocks) appears in bold.
- A plus sign (+) between two key names means that you must press those keys at the same time. For example, Press Alt+Tab means that you hold down the Alt key while you press the Tab key.
- A vertical bar between two or more menu items (for example, File | Close), means that you should select the first menu or menu item, then the next, and so on.

System Requirements

You will need the following hardware and software to install the code samples and sample database used in this book:

- Windows Vista SP2, Windows 7, Windows Server 2008 SP2, or greater. Either 32-bit or 64-bit editions will be suitable.
- At least 4 GB of free space on disk.
- At least 4 GB of RAM.
- A 2.0GHz x86 or x64 processor or better.
- An instance of SQL Server Analysis Services 2012 Tabular plus client components. Full instructions on how to install this are given in Chapter 2, "Getting Started with the Tabular Model."

Code Samples

The database used for examples in this book is based on Microsoft's Adventure Works 2012 DW sample database. Because there are several different versions of this database in existence, all of which are slightly different, we recommend that you download the database from the link below rather than use your own copy of Adventure Works if you want to follow the examples.

All sample projects and the sample database can be downloaded from the following page:

http://www.microsoftpressstore.com/title/9780735658189

Follow the instructions to download the BismTabularSample.zip file and the sample database.

Installing the Code Samples

Follow these steps to install the code samples on your computer so that you can follow the examples in this book:

1. Unzip the samples file onto your hard drive.
2. Restore the two SQL Server databases from the .bak files that can be found in the Databases directory. Full instructions on how to do this can be found here: *http://msdn.microsoft.com/en-us/library/ms177429.aspx*.
3. Restore the Adventure Works Tabular database to Analysis Services from the .abf file that can also be found in the Databases directory. Full instructions on how to do this can be found here: *http://technet.microsoft.com/en-us/library/ms174874.aspx*.
4. Each chapter has its own directory containing code samples. In many cases this takes the form of a project, which that must be opened in SQL Server Data Tools. Full instructions on how to install SQL Server Data Tools are given in Chapter 2, "Getting Started With the Tabular Model."

Acknowledgments

We'd like to thank the following people for their help and advice: Akshai Mirchandani, Amir Netz, Ashvini Sharma, Brad Daniels, Cristian Petculescu, Dan English, Darren Gosbell, Dave Wickert, Denny Lee, Edward Melomed, Greg Galloway, Howie Dickerman,

Hrvoje Piasevoli, Jeffrey Wang, Jen Stirrup, John Sirmon, John Welch, Kasper de Jonge, Marius Dumitru, Max Uritsky, Paul Sanders, Paul Turley, Rob Collie, Rob Kerr, TK Anand, Teo Lachev, Thierry D'Hers, Thomas Ivarsson, Thomas Kejser, Tomislav Piasevoli, Vidas Matelis, Wayne Robertson, Paul te Braak, Stacia Misner, Javier Guillen, Bobby Henningsen, Toufiq Abrahams, Christo Olivier, Eric Mamet, Cathy Dumas, and Julie Strauss.

Errata & Book Support

We've 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://www.microsoftpressstore.com/title/ 9780735658189

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*.

Please note that product support for Microsoft software is not offered through the addresses above.

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 read every one of your comments and ideas. Thanks in advance for your input!

Stay in Touch

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

CHAPTER 1

Introducing the Tabular Model

The purpose of this chapter is to introduce Analysis Services 2012, provide a brief overview of what the Tabular model is, and explore its relationship to the Multidimensional model, to Analysis Services 2012 as a whole, and to the wider Microsoft business intelligence (BI) stack. This chapter will also help you make what is probably the most important decision in your project's life cycle: whether you should use the Tabular model.

The Microsoft BI Ecosystem

In the Microsoft ecosystem, BI is not a single product; it's a set of features distributed across several products, as explained in the following sections.

What Is Analysis Services and Why Should I Use It?

Analysis Services is an online analytical processing (OLAP) database, a type of database that is highly optimized for the kinds of queries and calculations that are common in a business intelligence environment. It does many of the same things that a relational database can do, but it differs from a relational database in many respects. In most cases, it will be easier to develop your BI solution by using Analysis Services in combination with a relational database such as Microsoft SQL Server than by using SQL Server alone. Analysis Services certainly does not replace the need for a relational database or a properly designed data warehouse.

One way of thinking about Analysis Services is as an extra layer of metadata, or a semantic model, that sits on top of a data warehouse in a relational database. This extra layer contains information about how fact tables and dimension tables should be joined, how measures should aggregate up, how users should be able to explore the data through hierarchies, the definitions of common calculations, and so on. This layer includes one or more models containing the business logic of your data warehouse—and end users query these models rather than the underlying relational database. With all this information stored in a central place and shared by all users, the queries that users need to write become much simpler: All a query needs to do in most cases is describe which columns and rows are required, and the model applies the appropriate business logic to ensure that the numbers that are returned make sense. Most important, it becomes impossible to write a query that returns "incorrect" results due to a mistake by end users, such as joining two tables incorrectly or summing a

column that cannot be summed. This, in turn, means that end-user reporting and analysis tools must do much less work and can provide a clearer visual interface for end users to build queries. It also means that different tools can connect to the same model and return consistent results.

Another way of thinking about Analysis Services is as a kind of cache that you can use to speed up reporting. In most scenarios in which Analysis Services is used, it is loaded with a copy of the data in the data warehouse. Subsequently, all reporting and analytic queries are run against Analysis Services rather than against the relational database. Even though modern relational databases are highly optimized and contain many features specifically aimed at BI reporting, Analysis Services is a database specifically designed for this type of workload and can, in most cases, achieve much better query performance. For end users, optimized query performance is extremely important because it allows them to browse through data without waiting a long time for reports to run and without any breaks in their chain of thought.

For the IT department, the biggest benefit of all this is that it becomes possible to transfer the burden of authoring reports to the end users. A common problem with BI projects that do not use OLAP is that the IT department must build not only a data warehouse but also a set of reports to go with it. This increases the amount of time and effort involved, and can be a cause of frustration for the business when it finds that IT is unable to understand its reporting requirements or to respond to them as quickly as is desirable. When an OLAP database such as Analysis Services is used, the IT department can expose the models it contains to the end users and enable them to build reports themselves by using whatever tool with which they feel comfortable. By far the most popular client tool is Microsoft Excel. Ever since Office 2000, Excel PivotTables have been able to connect directly to Analysis Services cubes and Excel 2010 has some extremely powerful capabilities as a client for Analysis Services.

All in all, Analysis Services not only reduces the IT department's workload but also increases end user satisfaction because users now find they can build the reports they want and explore the data at their own pace without having to go through an intermediary.

A Short History of Analysis Services

SQL Server Analysis Services—or OLAP Services, as it was originally called when it was released with SQL Server 7.0—was the first foray by Microsoft into the BI market. When it was released, many people commented that this showed that BI software was ready to break out of its niche and reach a mass market, and the success of Analysis Services and the rest of the Microsoft BI stack over the past decade has proved them correct. SQL Server Analysis Services 2000 was the first version of Analysis Services to gain significant traction in the marketplace; Analysis Services 2005 quickly became the biggest-selling OLAP tool not long after its release, and, as Analysis Services 2008 and 2008 R2 improved scalability and performance still further, more and more companies started to adopt it as a cornerstone of their BI strategy. Terabyte-sized cubes are now not uncommon, and the famous example of the 24-TB cube Yahoo! built shows just what can be achieved. Analysis Services today is an extremely successful, mature product that is used and trusted in thousands of enterprise-level deployments.

The Microsoft BI Stack Today

The successes of Analysis Services would not have been possible if it had not been part of an equally successful wider suite of BI tools that Microsoft has released over the years. Because there are so many of these tools, it is useful to list them and provide a brief description of what each does.

The Microsoft BI stack can be broken up into two main groups: products that are part of the SQL Server suite of tools and products that are part of the Office group. As of SQL Server 2012, the SQL Server BI-related tools include:

- **SQL Server relational database** The flagship product of the SQL Server suite and the platform for the relational data warehouse. *http://www.microsoft.com/sqlserver/en/us/default.aspx*
- **SQL Azure** The Microsoft cloud-based version of SQL Server, not commonly used for BI purposes at the moment, but, as other cloud-based data sources become more common in the future, it will be used more and more. *https://www.windowsazure.com/en-us/home/features/sql-azure*
- **Parallel Data Warehouse** A highly specialized version of SQL Server, aimed at companies with multiterabyte data warehouses, which can scale out its workload over many physical servers. *http://www.microsoft.com/sqlserver/en/us/solutions-technologies/data-warehousing/pdw.aspx*
- **SQL Server Integration Services** An extract, transform, and load (ETL) tool for moving data from one place to another. Commonly used to load data into data warehouses. *http://www.microsoft.com/sqlserver/en/us/solutions-technologies/business-intelligence/integration-services.aspx*
- **Apache Hadoop** The most widely used open-source tool for aggregating and analyzing large amounts of data. Microsoft has decided to support it explicitly in Windows and provide tools to help integrate it with the rest of the Microsoft BI stack. *http://www.microsoft.com/bigdata*
- **SQL Server Reporting Services** A tool for creating static and semistatic, highly formatted reports and probably the most widely used SQL Server BI tool of them all. *http://www.microsoft.com/sqlserver/en/us/solutions-technologies/business-intelligence/reporting-services.aspx*
- **SQL Azure Reporting** The cloud-based version of SQL Server Reporting Services, in beta at the time of writing. *http://msdn.microsoft.com/en-us/library/windowsazure/gg430130.aspx*
- **Power View** A powerful new data visualization and analysis tool, available through Microsoft SharePoint, which acts as a front end to Analysis Services. *http://www.microsoft.com/sqlserver/en/us/future-editions/SQL-Server-2012-breakthrough-insight.aspx*
- **StreamInsight** A complex event-processing platform for analyzing data that arrives too quickly and in too large a volume to persist in a relational database. *http://www.microsoft.com/sqlserver/en/us/solutions-technologies/business-intelligence/complex-event-processing.aspx*

- **Master Data Services** A tool for managing a consistent set of master data for BI systems. *http://www.microsoft.com/sqlserver/en/us/solutions-technologies/business-intelligence/master-data-services.aspx*
- **Data Quality Services** A data quality and cleansing tool. *http://msdn.microsoft.com/en-us/library/ff877917(v=sql.110).aspx*
- **PowerPivot** A self-service BI tool that enables users to construct their own reporting solutions in Excel and publish them in SharePoint. It is very closely related to Analysis Services and will be discussed in greater detail in the following section, "Self-Service BI and Corporate BI."

BI tools developed by the Office group include:

- **SharePoint 2010** The Microsoft flagship portal and collaboration product. In the view of Microsoft, SharePoint is where all your BI reporting should be surfaced, through Excel and Excel Services, Reporting Services, Power View, or PerformancePoint. It also serves as the hub for sharing PowerPivot models by using PowerPivot for SharePoint.
- **PerformancePoint Services** A tool for creating BI dashboards inside SharePoint.
- **Excel 2010** The venerable spreadsheet program and probably the most widely used BI tool in the world, Excel has long been able to connect directly to Analysis Services through pivot tables and cube formulas. Now, with the release of PowerPivot (which is an Excel add-in), it is at the center of the Microsoft self-service BI strategy.

It is also worth mentioning that Microsoft makes various experimental BI tools available on its SQL Azure Labs site (*http://www.microsoft.com/en-us/sqlazurelabs/default.aspx*), which include the projects code-named "Social Analytics" and "Data Explorer." In addition, a large number of third-party software vendors make valuable contributions to the Microsoft BI ecosystem; for example, by building client tools for Analysis Services.

Self-Service BI and Corporate BI

One of the most significant trends in the BI industry over the last few years has been the appearance of so-called self-service BI tools such as QlikView and Tableau. These tools aim to give power users the ability to create small-scale BI solutions with little or no help from IT departments. In a sense, Analysis Services has always been a kind of self-service BI tool in that it enables end users to build their own queries and reports, but it still requires an IT professional to design and build the Analysis Services database and the underlying data warehouse. This means that it is usually grouped with other, more traditional corporate BI tools, where the design of databases and reporting of and access to data is strictly controlled by the IT department. In many organizations, however, especially smaller ones, the resources simply do not exist to undertake a large-scale BI project; even when they do, the failure rate for this type of project is often very high, hence the appeal to a certain class of users of self-service BI tools that enable them to do everything themselves.

The quickest way to start an argument between two BI professionals is to ask them what they think of self-service BI. On one hand, self-service BI makes BI development extremely business-focused, responsive, and agile. On the other hand, it can amplify the problems associated with the persistence of out-of-date data, poor data quality, lack of integration between multiple source systems, and different interpretations of how data should be modeled, especially because self-service BI proponents often claim that the time-consuming step of building a data warehouse is unnecessary. Whatever the advantages and disadvantages of self-service BI, it is a fast-growing market and one that Microsoft, as a software company, could not ignore, so in 2010 it released its own self-service BI tool called PowerPivot.

PowerPivot is essentially a desktop-based version of Analysis Services, but it takes the form of a free-to-download add-in for Excel 2010. (See *www.powerpivot.com* for more details.) It makes it very easy for Excel power users to import data from a number of sources, build their own models, and then query them using pivot tables. The PowerPivot database runs in-process inside Excel; all the imported data is stored there and all queries from Excel go against it. Excel users can work with vastly greater data volumes than they ever could before if they were storing the data directly inside an Excel worksheet, and they can still get lightning-fast query response times. When the Excel workbook is saved, the PowerPivot database and all the data in it is saved inside the workbook; the workbook can then be copied and shared like any regular Excel workbook, although any other user wishing to query the data held in PowerPivot must also have PowerPivot installed on his or her PC. To share models and reports between groups of users more efficiently, PowerPivot for SharePoint, a service that integrates with Microsoft SharePoint 2010 Enterprise edition, is required. With PowerPivot for SharePoint, it becomes possible to upload a workbook containing a PowerPivot database into SharePoint, enabling other users to view the reports in the workbook over the web by using Excel Service or to query the data held in PowerPivot on the server by using Excel or any other Analysis Services client tool on the desktop.

The release of PowerPivot does not mean that the Microsoft commitment to corporate BI tools has diminished. No single type of tool is appropriate in every situation, and it is to the credit of Microsoft that it not only sells both self-service and corporate BI tools but also has a coherent story for how both types of tools should coexist inside the same organization. Microsoft foresees a world in which IT departments and power users live in harmony, where IT-led projects use corporate BI tools and push data down from a central data warehouse out to the masses through reports and Analysis Services cubes, but where power users are also free to build their own self-service models in PowerPivot, share them with other people, and, if their models are popular, see them handed over to the IT department for further development, support, and eventual incorporation into the corporate model. PowerPivot for SharePoint provides a number of dashboards that enable the IT department to monitor usage of PowerPivot models that have been uploaded to SharePoint and, in Analysis Services 2012, it is possible to import a model created in PowerPivot into Analysis Services. It is likely that future releases will include features that help bridge the gap between the worlds of self-service and corporate BI.

Analysis Services 2012 Architecture: One Product, Two Models

This section explains a little about the architecture of Analysis Services, which in SQL Server 2012 is split into two models.

The first and most important point to make about Analysis Services 2012 is that it is really two products in one. Analysis Services in the SQL Server 2008 R2 release and before is still present, but it is now called the Multidimensional model. It has had a few improvements relating to performance, scalability, and manageability, but there is no new major functionality. Meanwhile, there is a new version of Analysis Services that closely resembles PowerPivot—this is called the Tabular model. The Tabular model is the subject of this book.

When installing Analysis Services, you must choose between installing an instance that runs in Tabular mode and one that runs in Multidimensional mode; more details on the installation process will be given in Chapter 2, "Getting Started with the Tabular Model." A Tabular instance can support only databases containing Tabular models, and a Multidimensional instance can support only databases containing Multidimensional models. Although these two parts of Analysis Services share much of the same code underneath, in most respects they can be treated as separate products. The concepts involved in designing the two types of model are very different, and you cannot convert a Tabular database into a Multidimensional database, or vice versa, without rebuilding everything from the beginning. That said, it is important to emphasize the fact that, from an end user's point of view, the two models do almost the same things and appear almost identical when used through a client tool such as Excel.

The following sections compare the functionality available in the Tabular and Multidimensional models and define some important terms that are used throughout the rest of this book.

The Tabular Model

A *database* is the highest-level object in the Tabular model and is very similar to the concept of a database in the SQL Server relational database. An instance of Analysis Services can contain many databases, and each database can be thought of as a self-contained collection of objects and data relating to a single business solution. If you are writing reports or analyzing data and find that you need to run queries on multiple databases, you have probably made a design mistake somewhere because everything you need should be contained in a single database.

Tabular models are designed by using *SQL Server Data Tools* (*SSDT*), and a project in SSDT maps onto a database in Analysis Services. After you have finished designing a project in SSDT, it must be *deployed* to an instance of Analysis Services, which means SSDT executes a number of commands to create a new database in Analysis Services or alters the structure of an existing database. *SQL Server Management Studio (SSMS)*, a tool that can be used to manage databases that have already been deployed, can also be used to write queries against databases.

Databases are made up of one or more *tables* of data. Again, a table in the Tabular model is very similar to a table in the relational database world. A table in Tabular is usually loaded from a single table in a relational database or from the results of a SQL SELECT statement. A table has a fixed

number of *columns* that are defined at design time and can have a variable number of rows, depending on the amount of data that is loaded. Each column has a fixed type, so for example, a single column could contain only integers, only text, or only decimal values. Loading data into a table is referred to as *processing* that table.

It is also possible to define *relationships* between tables at design time. Unlike in SQL, it is not possible to define relationships at query time; all queries must use these preexisting relationships. However, relationships between tables can be marked as *active* or *inactive*, and at query time it is possible to choose which relationships between tables are actually used. It is also possible to simulate the effect of relationships that do not exist inside queries and calculations. All relationships are one-to-many relationships and must involve just one column from each of two tables. It is not possible to define relationships that are explicitly one to one or many to many, although it is certainly possible to achieve the same effect by writing queries and calculations in a particular way. It is also not possible to design relationships that are based on more than one column from a table or recursive relationships that join a table to itself.

The Tabular model uses a purely memory-based engine and stores only a copy of its data on disk so that no data is lost if the service is restarted. Whereas the Multidimensional model, like most relational database engines, stores its data in a row-based format, the Tabular model uses a column-oriented database called the *xVelocity in-memory analytics engine*, which in most cases offers significant query performance improvements. (For more details on the column-based type of database, see *http://en.wikipedia.org/wiki/Column-oriented_DBMS*.)

Note The xVelocity analytics in-memory engine was known as the Vertipaq engine before the release of Analysis Services 2012. Many references to the Vertipaq name remain in documentation, blog posts, and other material online, and it even persists inside the product itself in property names and Profiler events. The name xVelocity is also used to refer to the wider family of related technologies, including the new column store index feature in the SQL Server 2012 relational database engine. For a more detailed explanation of this terminology, see *http://blogs.msdn.com/b/analysisservices/archive/2012/03/09/xvelocity-and-analysis-services.aspx*.

Queries and calculations in Tabular are defined in *Data Analysis eXpressions* (*DAX*), the native language of the Tabular model, and in PowerPivot. Client tools such as Power View can generate DAX queries to retrieve data from a Tabular model, or you can write your own DAX queries and use them in reports. It is also possible to write queries by using the *MDX* language that Multidimensional models use. This means that the Tabular model is backward compatible with the large number of existing Analysis Services client tools that are available from Microsoft, such as Excel and SQL Server Reporting Services, and tools from third-party software vendors.

Derived columns, called *calculated columns*, can be added to a table in a Tabular model; they use DAX expressions to return values based on the data already loaded in other columns in the same or other tables in the same Analysis Services database. Calculated columns are populated at processing time and, after processing has taken place, behave in exactly the same way as regular columns.

Measures can also be defined on tables by using DAX expressions; a measure can be thought of as a DAX expression that returns some form of aggregated value based on data from one or more columns. A simple example of a measure is one that returns the sum of all values from a column of data that contains sales volumes. *Key performance indicators (KPIs*) are very similar to measures, but are collections of calculations that enable you to determine how well a measure is doing relative to a target value and whether it is getting closer to reaching that target over time.

Most front-end tools such as Excel use a PivotTable-like experience for querying Tabular models: Columns from different tables can be dragged onto the rows axis and columns axis of a pivot table so that the distinct values from these columns become the individual rows and columns of the pivot table, and measures display aggregated numeric values inside the table. The overall effect is something like a Group By query in SQL, but the definition of how the data aggregates up is predefined inside the measures and is not necessarily specified inside the query itself. To improve the user experience, it is also possible to define *hierarchies* on tables inside the Tabular model, which create multilevel, predefined drill paths. *Perspectives* can hide certain parts of a complex model, which can aid usability, and security *roles* can be used to deny access to specific rows of data from tables to specific users. Perspectives should not be confused with security, however; even if an object is hidden in a perspective it can still be queried, and perspectives themselves cannot be secured.

The Multidimensional Model

At the highest level, the Multidimensional model is very similar to the Tabular model: Data is organized in databases, and databases are designed in SSDT (formerly BI Development Studio, or BIDS) and managed by using SQL Server Management Studio.

The differences become apparent below the database level, where multidimensional rather than relational concepts are prevalent. In the Multidimensional model, data is modeled as a series of *cubes* and *dimensions*, not tables. Each cube is made up of one or more *measure groups*, and each measure group in a cube is usually mapped onto a single fact table in the data warehouse. A measure group contains one or more *measures*, which are very similar to measures in the Tabular model. A cube also has two or more dimensions: one special dimension, the *Measures dimension*, which contains all the measures from each of the measure groups, and various other dimensions such as Time, Product, Geography, Customer, and so on, which map onto the logical dimensions present in a dimensional model. Each of these non-Measures dimensions consists of one or more *attributes* (for example, on a Date dimension, there might be attributes such as *Date*, *Month*, and *Year*), and these attributes can themselves be used as single-level hierarchies or to construct multilevel *user hierarchies*. Hierarchies can then be used to build queries. Users start by analyzing data at a highly aggregated level, such as a Year level on a Time dimension, and can then navigate to lower levels such as Quarter, Month, and Date to look for trends and interesting anomalies.

As you would expect, because the Multidimensional model is the direct successor to previous versions of Analysis Services, it has a very rich and mature set of features representing the fruit of more than a decade of development, even if some of them are not used very often. Most of the features available in the Tabular model are present in the Multidimensional model, but the Multidimensional

model also has many features that have not yet been implemented in Tabular. A detailed feature comparison between the two models appears later in this chapter.

In terms of data storage, the Multidimensional model can store its data in three ways:

- *Multidimensional OLAP* (*MOLAP*), where all data is stored inside Analysis Services' own disk-based storage format.
- *Relational OLAP* (*ROLAP*), where Analysis Services acts purely as a metadata layer and where no data is stored in Analysis Services itself; SQL queries are run against the relational source database when a cube is queried.
- *Hybrid OLAP* (*HOLAP*), which is the same as ROLAP but where some pre-aggregated values are stored in MOLAP.

MOLAP storage is used in the vast majority of implementations, although ROLAP is sometimes used when a requirement for so-called real-time BI HOLAP is almost never used.

One particular area in which the Multidimensional and Tabular models differ is in the query and calculation languages they support. The native language of the Multidimensional model is MDX, and that is the only language used for defining queries and calculations. The MDX language has been successful and is supported by a large number of third-party client tools for Analysis Services. It was also promoted as a semiopen standard by a cross-vendor industry body called the XMLA Council (now effectively defunct) and, as a result, has also been adopted by many other OLAP tools that are direct competitors to Analysis Services. However, the problem with MDX is the same problem that many people have with the Multidimensional model in general: although it is extremely powerful, many BI professionals have struggled to learn it because the concepts it uses, such as dimensions and hierarchies, are very different from the ones they are accustomed to using in SQL.

In addition, Microsoft has publicly committed (in this post on the Analysis Services team blog and other public announcements at *http://blogs.msdn.com/b/analysisservices/archive/2011/05/16/analysis-services-vision-amp-roadmap-update.aspx*) to support DAX queries on the Multidimensional model at some point after Analysis Services 2012 has been released, possibly as part of a service pack. This will allow Power View to query Multidimensional models and Tabular models, although it is likely that some compromises will have to be made and some Multidimensional features might not work as expected when DAX queries are used.

Why Have Two Models?

Why has this split happened? Although Microsoft does not want to make any public comments on this topic, there are a number of likely reasons.

- Analysis Services Multidimensional is getting old. It was designed in an age of 32-bit servers with one or two processors and less than a gigabyte of RAM, when disk-based storage was the only option for databases. Times have changed, and modern hardware is radically different; now a new generation of memory-based, columnar databases has set the standard for query performance with analytic workloads, and Analysis Services must adopt this new

technology to keep up. Retrofitting the new xVelocity in-memory engine into the existing Multidimensional model was not, however, a straightforward job, so it was necessary to introduce the new Tabular model to take full advantage of xVelocity.

- Despite the success of Analysis Services Multidimensional, there has always been a perception that it is difficult to learn. Some database professionals, accustomed to relational data modeling, struggle to learn multidimensional concepts, and those that do find the learning curve is steep. Therefore, if Microsoft wants to bring BI to an ever-wider audience, it must simplify the development process—hence the move from the complex world of the Multidimensional model to the relatively simple and familiar concepts of the Tabular model.
- Microsoft sees self-service BI as a huge potential source of growth, and PowerPivot is its entry into this market. It is also important to have consistency between the Microsoft self-service and corporate BI tools. Therefore, if Analysis Services must be overhauled, it makes sense to make it compatible with PowerPivot, with a similar design experience so self-service models can easily be upgraded to full-fledged corporate solutions.
- Some types of data are more appropriately, or more easily, modeled by using the Tabular approach, and some types of data are more appropriate for a Multidimensional approach. Having different models gives developers the choice to use whichever approach suits their circumstances.

What Is the BI Semantic Model?

One term that has been mentioned a lot in the discussions about Analysis Services 2012 is the *BI Semantic Model* or *BISM*. This term does not refer to either the Multidimensional or Tabular models specifically but, instead, describes the function of Analysis Services in the Microsoft BI stack: the fact that it acts as a semantic layer on top of a relational data warehouse, adding a rich layer of metadata that includes hierarchies, measures, and calculations. In that respect, it is very similar to the term Unified Dimensional Model that was used around the time of the SQL Server 2005 launch. In some cases, the term BI Semantic Model has referred to the Tabular model only, but this is not correct. Because this book is specifically concerned with the Tabular model, we will not be using this term very often; nevertheless, we believe it is important to understand exactly what it means and how it should be used.

The Future of Analysis Services

Having two models inside Analysis Services, plus two query and calculation languages, is clearly not an ideal state of affairs. First and foremost, it means you have to choose which model to use at the start of your project, when you might not know enough about your requirements to know which one is appropriate—and this is the question we will address in the next section. It also means that anyone who decides to specialize in Analysis Services has to learn two technologies. Presumably, this state of affairs will not continue in the long term.

Microsoft has been very clear in saying that the Multidimensional model is not deprecated and that the Tabular model is not its replacement. It is likely that new features for Multidimensional will be released in future versions of Analysis Services. The fact that the Tabular and Multidimensional models share some of the same code suggests that some new features could easily be developed for both models simultaneously. The post on the Analysis Services blog previously referenced suggests that in time the two models will converge and offer much the same functionality, so the decision about which model to use is based on whether the developer prefers to use a multidimensional or relational way of modeling data. Support for DAX queries in the Multidimensional model, when it arrives, will represent one step in this direction.

One other thing is clear about the future of Analysis Services: It will be moving to the cloud. Although no details are publicly available at the time of writing, Microsoft has confirmed it is working on a cloud-based version of Analysis Services and this, plus SQL Azure, SQL Azure Reporting Services, and Office 365, will form the core of the Microsoft cloud BI strategy.

Choosing the Right Model for Your Project

It might seem strange to be addressing the question of whether the Tabular model is appropriate for your project at this point in the book, before you have learned anything about the Tabular model, but you must answer this question at an equally early stage of your BI project. At a rough guess, either model will work equally well for about 60 percent to 70 percent of projects, but for the remaining 30 percent to 40 percent, the correct choice of model will be vital.

As has already been stated, after you have started developing with one model in Analysis Services, there is no way of switching over to use the other; you have to start all over again from the beginning, possibly wasting much precious development time, so it is very important to make the correct decision as soon as possible. Many factors must be taken into account when making this decision. In this section we discuss all of them in a reasonable amount of detail. You can then bear these factors in mind as you read the rest of this book, and when you have finished it, you will be in a position to know whether to use the Tabular model or the Multidimensional model.

Licensing

Analysis Services 2012 is available in the following editions: SQL Server Standard, SQL Server Business Intelligence, and SQL Server Enterprise. In SQL Server Standard edition, however, only the Multidimensional model is available, and has the same features that were available in SQL Server Standard edition of previous versions of Analysis Services. This means that several important features needed for scaling up the Multidimensional model, such as partitioning, are not available in SQL Server Standard edition. SQL Server Business Intelligence edition contains both the Multidimensional and Tabular models, as does SQL Server Enterprise edition. In terms of Analysis Services functionality, these two editions are the same; the only difference between them is that SQL Server Business Intelligence edition licensing is based on buying a server license plus Client Access Licenses (CALs), whereas SQL Server Enterprise edition is licensed on a per-CPU core basis. (You can no longer license

SQL Server Enterprise edition on a server-plus-CALs basis as was possible in the past.) In SQL Server Business Intelligence and SQL Server Enterprise editions, both Tabular and Multidimensional models contain all available features and can use as many cores as the operating system makes available.

The upshot of this is that it could be more expensive in some situations to use Tabular than Multidimensional because Multidimensional is available in SQL Server Standard edition and Tabular is not. If you have a limited budget, already have existing Multidimensional skills, or are willing to learn them, and your data volumes mean that you do not need to use Multidimensional features such as partitioning, it might make sense to use Multidimensional and SQL Server Standard edition to save money. If you are willing to pay slightly more for SQL Server Business Intelligence edition or SQL Server Enterprise edition, however, then licensing costs should not be a consideration in your choice of model.

Upgrading from Previous Versions of Analysis Services

As has already been mentioned, there is no easy way of turning a Multidimensional model into a Tabular model. Tools undoubtedly will appear on the market that claim to make this transition with a few mouse clicks, but such tools could only ever work for very simple Multidimensional models and would not save much development time. Therefore, if you already have a mature Multidimensional implementation and the skills in house to develop and maintain it, it probably makes no sense to abandon it and move over to Tabular unless you have specific problems with Multidimensional that Tabular is likely to solve.

Ease of Use

In contrast, if you are starting an Analysis Services 2012 project with no previous Multidimensional or OLAP experience, it is very likely that you will find Tabular much easier to learn than Multidimensional. Not only are the concepts much easier to understand, especially if you are used to working with relational databases, but the development process is also much more straightforward and there are far fewer features to learn. Building your first Tabular model is much quicker and easier than building your first Multidimensional model. It can also be argued that DAX is easier to learn than MDX, at least when it comes to writing basic calculations, but the truth is that both MDX and DAX can be equally confusing for anyone used to SQL.

Compatibility with PowerPivot

The Tabular model and PowerPivot are almost identical in the way their models are designed; the user interfaces for doing so are practically the same and both use DAX. PowerPivot models can also be imported into SQL Server data tools to generate a Tabular model, although the process does not work the other way, and a Tabular model cannot be converted to a PowerPivot model. Therefore, if you have a strong commitment to self-service BI by using PowerPivot, it makes sense to use Tabular for your corporate BI projects because development skills and code are transferable between the two.

Query Performance Characteristics

Although it would be dangerous to make sweeping generalizations about query performance, it's fair to say that Tabular will perform at least as well as Multidimensional in most cases and will outperform it in some specific scenarios. Distinct count measures, which are a particular weakness of the Multidimensional model, perform extremely well in Tabular, for instance. Anecdotal evidence also suggests that queries for detail-level reports (for example, queries that return a large number of rows and return data at a granularity close to that of the fact table) will perform much better on Tabular as long as they are written in DAX and not MDX. When more complex calculations or modeling techniques such as many-to-many relationships are involved, it is much more difficult to say whether Multidimensional or Tabular will perform better, unfortunately, and a proper proof of concept will be the only way to tell whether the performance of either model will meet requirements.

Processing Performance Characteristics

Comparing the processing performance of Multidimensional and Tabular is also difficult. It might be a lot slower to process a large table in Tabular than the equivalent measure group in Multidimensional because Tabular cannot process partitions in the same table in parallel, whereas Multidimensional (assuming you are using SQL Server Business Intelligence or SQL Server Enterprise edition and are partitioning your measure groups) can process partitions in the same measure group in parallel. Disregarding the different, noncomparable operations that each model performs when it performs processing, such as building aggregations and indexes in the Multidimensional model, the number of rows of raw data that can be processed per second for a single partition is likely to be similar.

However, Tabular has some significant advantages over Multidimensional when it comes to processing. First, there are no aggregations in the Tabular model, and this means that there is one less time-consuming task to be performed at processing time. Second, processing one table in a Tabular model has no direct impact on any of the other tables in the model, whereas in the Multidimensional model, processing a dimension has consequential effects. Doing a full process on a dimension in the Multidimensional model means that you must do a full process on any cubes that dimension is used in, and even doing a process update on a dimension requires a process index on a cube to rebuild aggregations. Both of these can cause major headaches on large Multidimensional deployments, especially when the window available for processing is small.

Hardware Considerations

The Multidimensional and Tabular models also have very different hardware specification requirements. Multidimensional's disk-based storage means that high-performance disks plus plenty of space on those disks is important; it will cache data in memory as well, so having sufficient RAM for this is very useful but not essential. For Tabular, the performance of disk storage is much less of a priority because it is an in-memory database. For that very reason, though, it is much more important to have enough RAM to hold the database and to accommodate any spikes in memory usage that occur when queries are running or when processing is taking place.

Multidimensional's disk requirements will probably be easier to accommodate than Tabular's memory requirements. Buying a large amount of disk storage for a server is relatively cheap and straightforward for an IT department; many organizations have storage area networks (SANs) that, though they might not perform as well as they should, make providing enough storage space (or increasing that provision) very simple. However, buying large amounts of RAM for a server can be more difficult—you might find that asking for half a terabyte of RAM on a server raises some eyebrows—and if you find you need more RAM than you originally thought, increasing the amount that is available can also be awkward. Based on experience, it is easy to start with what seems like a reasonable amount of RAM and then find that, as fact tables grow, new data is added to the model, and queries become more complex, you start to encounter out-of-memory errors. Furthermore, for some extremely large Analysis Services implementations with several terabytes of data, it might not be possible to buy a server with sufficient RAM to store the model, so Multidimensional might be the only feasible option.

Real-Time BI

Although not quite the industry buzzword that it was a few years ago, the requirement for real-time or near-real-time data in BI projects is becoming more common. Real-time BI usually refers to the need for end users to be able to query and analyze data as soon as it has been loaded into the data warehouse, with no lengthy waits for the data to be loaded into Analysis Services.

The Multidimensional model can handle this in one of two ways: Either use MOLAP storage and partition your data so that all the new data in your data warehouse goes to one relatively small partition that can be processed quickly, or use ROLAP storage and turn off all caching so that Multidimensional issues SQL queries every time it is queried. The first of these options is usually preferred, although it can be difficult to implement, especially if dimension tables and fact tables change. Updating the data in a dimension can be slow and can also require aggregations to be rebuilt. ROLAP storage in Multidimensional can often result in very poor query performance if data volumes are large, so the time taken to run a query in ROLAP mode might be greater than the time taken to reprocess the MOLAP partition in the first option.

The Tabular model offers what are essentially the same two options but with fewer shortcomings than their Multidimensional equivalents. If data is being stored in the xVelocity in-memory engine, updating data in one table has no impact on the data in any other table, so processing times are likely to be faster and implementation much easier. If data is to remain in the relational engine, then the major difference is that the equivalent of ROLAP mode, called DirectQuery, will, it's hoped, perform much better than ROLAP. This is because in DirectQuery mode, Analysis Services tries to push all its query processing back to the relational database by translating the whole query it receives into SQL queries. (Multidimensional ROLAP mode does not do this; it translates some internal operations into SQL queries but will still do some work, such as evaluating calculations, by itself.) DirectQuery, however, also comes with a number of significant limitations: It can accept only DAX queries and not MDX when in DirectQuery mode, which means, for instance, that Excel users cannot see real-time data because Excel can generate only MDX queries; only SQL Server is supported as a data source; data security must be implemented in SQL Server and cannot be implemented in Analysis Services; and, finally, neither calculated columns nor many common DAX functions are supported, so only models

with very simple DAX calculations can be used. A full description of how to configure DirectQuery mode is given in Chapter 9, "Understanding xVelocity and DirectQuery."

Client Tools

In many cases, the success or failure of a BI project depends on the quality of the tools that end users use to analyze the data being provided. Therefore, the question of which client tools are supported by which model is an important one.

Both the Tabular model and the Multidimensional model support MDX queries, so, in theory, most Analysis Services client tools should support both models. However, in practice, although some client tools such as Excel and SQL Server Reporting Services do work equally well on both, some third-party client tools might need to be updated to their latest versions to work, and some older tools that are still in use but are no longer supported might not work properly or at all.

At the time of writing, only the Tabular model supports DAX queries, although support for DAX queries in the Multidimensional model is promised at some point in the future. This means that, at least initially, Power View—the new, highly regarded Microsoft data visualization tool—will work only on Tabular models. Even when DAX support in Multidimensional models is released, it is likely that not all Power View functionality will work on it and, similarly, that not all Multidimensional functionality will work as expected when queried by using DAX.

Feature Comparison

One more thing to consider when choosing a model is the functionality present in the Multidimensional model that either has no equivalent or is only partially implemented in the Tabular model. Not all of this functionality is important for all projects, however, and it must be said that in many scenarios it is possible to approximate some of this Multidimensional functionality in Tabular by using some clever DAX in calculated columns and measures. In any case, if you do not have any previous experience using Multidimensional, you will not miss functionality you have never had.

Here is a list of the most important functionality missing in Tabular:

- **Writeback**, the ability for an end user to write values back to a Multidimensional database. This can be very important for financial applications in which users enter budget figures, for example.
- **Translations**, in which the metadata of a Multidimensional model can appear in different languages for users with different locales on their desktops. There is no way of implementing this in Tabular.
- **Dimension security on measures**, in which access to a single measure can be granted or denied.
- **Cell security**, by which access to individual cells can be granted or denied. Again, there is no way of implementing this in Tabular, but it is only very rarely used in Multidimensional.

- **Ragged hierarchies**, a commonly used technique for avoiding the use of a parent/child hierarchy. In a Multidimensional model, a user hierarchy can be made to look something like a parent/child hierarchy by hiding members if certain conditions are met; for example, if a member has the same name as its parent. This is known as creating a ragged hierarchy. Nothing equivalent is available in the Tabular model.

- **Role-playing dimensions**, designed and processed once, then appear many times in the same model with different names and different relationships to measure groups; in the Multidimensional model, this is known as using role-playing dimensions. Something similar is possible in the Tabular model, by which multiple relationships can be created between two tables (see Chapter 3, "Loading Data Inside Tabular," for more details on this), and although this is extremely useful functionality, it does not do exactly the same thing as a role-playing dimension. In Tabular, if you want to see the same table in two places in the model simultaneously, you must load it twice, and this can increase processing times and make maintenance more difficult.

- **Scoped assignments** and **unary operators**, advanced calculation functionality, is present in MDX in the Multidimensional model but is not possible or at least not easy to re-create in DAX in the Tabular model. These types of calculation are often used in financial applications, so this and the lack of writeback and true parent/child hierarchy support mean that the Tabular model is not suited for this class of application.

The following functionality can be said to be only partially supported in Tabular:

- **Parent/child hierarchy support** in Multidimensional is a special type of hierarchy built from a dimension table with a self-join on it by which each row in the table represents one member in the hierarchy and has a link to another row that represents the member's parent in the hierarchy. Parent/child hierarchies have many limitations in Multidimensional and can cause query performance problems. Nevertheless, they are very useful for modeling hierarchies such as company organization structures because the developer does not need to know the maximum depth of the hierarchy at design time. The Tabular model implements similar functionality by using DAX functions such as *PATH* (see Chapter 9 for details), but, crucially, the developer must decide what the maximum depth of the hierarchy will be at design time.

- **Support for many-to-many relationships** in the Multidimensional model is one of its most important features, and it is frequently used. (For some applications, see the white paper at *http://www.sqlbi.com/articles/many2many/*.) It is possible to re-create this functionality in Tabular by using DAX, as described in Chapter 12, "Using Advanced Tabular Relationships," but even though query performance is likely to be just as good if not better than Multidimensional when using this approach, it adds a lot of complexity to the DAX expressions used in measures. If a model contains a large number of many-to-many relationships or chained many-to-many relationships, this added complexity can mean that maintenance of the DAX used in measures is extremely difficult.

- **Drillthrough**, by which the user can click a cell to see all the detail-level data that is aggregated to return that value. Drillthrough is supported in both models but, in the Multidimensional model, it is possible to specify which columns from dimensions and measure groups are returned from a drillthrough. In the Tabular model, no interface exists in SQL Server data tools for doing this and, by default, a drillthrough returns every column from the underlying table. It is possible, though, to edit the XMLA definition of your model manually to do this, as described in the blog post at *http://sqlblog.com/blogs/marco_russo/archive/2011/08/18/drillthrough-for-bism-tabular-and-attribute-keys-in-ssas-denali.aspx*. A user interface to automate this editing process is also available in the BIDS Helper add-in (*http://bidshelper.codeplex.com/*).

Summary

In this chapter, you have seen what the Tabular and Multidimensional models in Analysis Services 2012 are, what their strengths and weaknesses are, and when they should be used. The key point to remember is that the two models are very different—practically separate products—and that you should not make the decision to use the Tabular model on a project without considering whether it is a good fit for your requirements. In the next chapter, you will take a first look at how you can actually build Tabular models.

CHAPTER 4

DAX Basics

Now that you have seen the basics of SQL Server Analysis Services (SSAS) Tabular, it is time to learn the fundamentals of Data Analysis Expressions (DAX) expressions. DAX has its own syntax for defining calculation expressions; it is somewhat similar to a Microsoft Excel expression, but it has specific functions that enable you to create more advanced calculations on data stored in multiple tables.

Understanding Calculation in DAX

Any calculation in DAX begins with the equal sign, which resembles the Excel syntax. Nevertheless, the DAX language is very different from Excel because DAX does not support the concept of cells and ranges as Excel does; to use DAX efficiently, you must learn to work with columns and tables, which are the fundamental objects in the Tabular world.

Before you learn how to express complex formulas, you must master the basics of DAX, which include 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 introduce these concepts.

DAX Syntax

A relatively simple way to understand how DAX syntax works is to start with an example. Suppose you have loaded the FactInternetSales table in a Tabular project. In Figure 4-1, you can see some of its columns.

OrderQuantity	UnitPrice	ExtendedAmount	ProductStandardCost	TotalProductCost	SalesAmount
1	$4.99	$4.99	$1.87	$1.87	$4.99
1	$4.99	$4.99	$1.87	$1.87	$4.99
1	$4.99	$4.99	$1.87	$1.87	$4.99
1	$4.99	$4.99	$1.87	$1.87	$4.99
1	$4.99	$4.99	$1.87	$1.87	$4.99
1	$4.99	$4.99	$1.87	$1.87	$4.99
1	$4.99	$4.99	$1.87	$1.87	$4.99
1	$4.99	$4.99	$1.87	$1.87	$4.99
1	$4.99	$4.99	$1.87	$1.87	$4.99
1	$4.99	$4.99	$1.87	$1.87	$4.99
1	$4.99	$4.99	$1.87	$1.87	$4.99
1	$4.99	$4.99	$1.87	$1.87	$4.99
1	$4.99	$4.99	$1.87	$1.87	$4.99
1	$4.99	$4.99	$1.87	$1.87	$4.99
1	$4.99	$4.99	$1.87	$1.87	$4.99
1	$4.99	$4.99	$1.87	$1.87	$4.99

FIGURE 4-1 Here you can see the FactInternetSales table in a Tabular project.

You now use this data to calculate the margin, subtracting the TotalProductCost from the SalesAmount, and you use the technique already learned in Chapter 2, "Getting Started with the Tabular Model," to create calculated columns. To do that, you must write the following DAX formula in a new calculated column, which you can call *GrossMargin*.

```
= FactInternetSales[SalesAmount] - FactInternetSales[TotalProductCost]
```

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. You can see the resulting column in Figure 4-2. (Later, you see that DAX is used also to define measures.)

OrderQuantity	UnitPrice	ExtendedAmount	ProductStandardCost	TotalProductCost	SalesAmount	GrossMargin
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12

FIGURE 4-2 The *GrossMargin* calculated column has been added to the table.

This DAX expression handles numeric values and returns a numeric value. DAX can work with data types other than numbers. In the next section, you learn the different data types available in DAX and how to work with them.

DAX Data Types

DAX can compute values for seven data types:

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

DAX has a powerful type-handling system so that 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 and on the operator used. Type conversion happens automatically during the expression evaluation.

Be aware of this behavior in case the type returned from a DAX expression is not the expected one; in such a case, you must investigate the data type of the terms used in the expression. For example, if one of the terms of a sum is a date, the result is a date, too. However, if the data type is an integer, the result is an integer. This is known as operator overloading, and you can see an example of its behavior in Figure 4-3, in which the *OrderDatePlusOne* column is calculated by adding 1 to the value in the *OrderDate* column, by using the following formula.

```
= FactInternetSales[OrderDate] + 1
```

The result is a date because the *OrderDate* column is of the *date* data type.

TotalProductCost	SalesAmount	GrossMargin	TaxAmt	Freight	OrderDate	OrderDatePlusOne
$1.87	$4.99	$3.12	$0.40	$0.12	7/1/2003 12:00:00 AM	7/2/2003 12:00:00 AM
$1.87	$4.99	$3.12	$0.40	$0.12	7/1/2003 12:00:00 AM	7/2/2003 12:00:00 AM
$1.87	$4.99	$3.12	$0.40	$0.12	7/2/2003 12:00:00 AM	7/3/2003 12:00:00 AM
$1.87	$4.99	$3.12	$0.40	$0.12	7/2/2003 12:00:00 AM	7/3/2003 12:00:00 AM
$1.87	$4.99	$3.12	$0.40	$0.12	7/2/2003 12:00:00 AM	7/3/2003 12:00:00 AM
$1.87	$4.99	$3.12	$0.40	$0.12	7/2/2003 12:00:00 AM	7/3/2003 12:00:00 AM
$1.87	$4.99	$3.12	$0.40	$0.12	7/2/2003 12:00:00 AM	7/3/2003 12:00:00 AM
$1.87	$4.99	$3.12	$0.40	$0.12	7/3/2003 12:00:00 AM	7/4/2003 12:00:00 AM
$1.87	$4.99	$3.12	$0.40	$0.12	7/3/2003 12:00:00 AM	7/4/2003 12:00:00 AM
$1.87	$4.99	$3.12	$0.40	$0.12	7/3/2003 12:00:00 AM	7/4/2003 12:00:00 AM
$1.87	$4.99	$3.12	$0.40	$0.12	7/3/2003 12:00:00 AM	7/4/2003 12:00:00 AM

FIGURE 4-3 Adding an integer to a date results in a date increased by the corresponding number of days.

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

```
= 5 & 4
```

it returns a "54" string result. However, 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. Even if this behavior is convenient, later in this chapter you see the types of errors that might occur during these automatic conversions.

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. If you must take only the date part of a *DATETIME*, always remember to use *TRUNC* to get rid of the decimal part.

DAX Operators

You have seen the importance of operators in determining the type of an expression; you can now see, in Table 4-1, a list of the operators available in DAX.

TABLE 4-1 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")

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

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

that correspond, respectively, to

```
[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, as previously mentioned. When such values are used directly in formulas, they are called literals and, although using literals is straightforward, the syntax for referencing a column needs some attention. Here is the basic syntax.

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

The table name can be enclosed in single quote characters. Most of the time, 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, however, must always be enclosed in square brackets. Note that the table name is optional. If the table name is omitted, the column name is 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) when you reference a column to avoid any confusion.

Understanding Calculated Columns and Measures

Now that you know the basics of DAX syntax, you must learn one of the most important concepts in DAX: the difference between calculated columns and measures. Even though they might appear similar at first sight because you can make some calculations both ways, you must use measures to implement the most flexible calculations. This is a key to unlock the true power of DAX.

Calculated Columns

If you want to create a calculated column, you can move to the last column of the table, which is named *Add Column*, and start writing the formula. The DAX expression must be inserted into the formula bar, and Microsoft IntelliSense helps you during the writing of the expression.

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

Note As you see later, 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 aggregate column values for a range of rows and possibly operating on a different row by filtering a table made of only one row. More on this topic is in Chapter 5, "Understanding Evaluation Context."

One important concept that must be well understood about calculated columns is that they are computed during the Tabular database processing and then stored in the database, just as any other column. This might seem strange if you are accustomed to SQL-computed columns, which are computed at query time and do not waste space. In Tabular, however, all calculated columns occupy space in memory and are computed once during table processing.

This behavior is handy whenever you create very complex calculated columns. The time required to compute them is always process time and not query time, resulting in a better user experience. Nevertheless, you must always remember that a calculated column uses precious RAM. If, for example, you have a complex formula for a calculated column, you might be tempted to separate the steps of computation into different intermediate columns. Although this technique is useful during project development, it is a bad habit in production because each intermediate calculation is stored in RAM and wastes space.

Measures

You have already seen in Chapter 2 how to create a measure by using the measure grid; now you learn the difference between a calculated column and a measure to understand when to use which one.

Calculated columns are easy to create and use. You have already seen in Figure 4-2 how to define the *GrossMargin* column to compute the amount of the gross margin.

```
[GrossMargin] = FactInternetSales[SalesAmount] - FactInternetSales[TotalProductCost]
```

But what happens if you want to show the gross margin as a percentage of the sales amount? You could create a calculated column with the following formula.

```
[GrossMarginPerc] = FactInternetSales[GrossMargin] / FactInternetSales[SalesAmount]
```

This formula computes the right value at the row level, as you can see in Figure 4-4.

OrderQuantity	UnitPrice	ExtendedAmount	ProductStandardCost	TotalProductCost	SalesAmount	GrossMargin	GrossMarginPerc
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12	62.60 %
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12	62.60 %
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12	62.60 %
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12	62.60 %
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12	62.60 %
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12	62.60 %
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12	62.60 %
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12	62.60 %
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12	62.60 %
1	$4.99	$4.99	$1.87	$1.87	$4.99	$3.12	62.60 %

FIGURE 4-4 The *GrossMarginPerc* column shows the Gross Margin as a percentage, calculated row by row.

Nevertheless, when you compute the aggregate value, you cannot rely on calculated columns. In fact, the aggregate value is computed as the sum of gross margin divided by the sum of sales amount. Thus, the ratio must be computed on the aggregates; you cannot use an aggregation of calculated columns. In other words, you compute the ratio of the sum, not the sum of the ratio.

The correct formula for the *GrossMarginPerc* is as follows.

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

But, as already stated, you cannot enter it into a calculated column. If you need to operate on aggregate values instead of on a row-by-row basis, you must create measures, which is the topic of the current section.

Measures and calculated columns both use DAX expressions; the difference is the context of evaluation. A measure is evaluated in the context of the cell of the pivot table or DAX query, whereas a calculated column is evaluated at the row level of the table to which it belongs. The context of the cell (later in the book, you learn that this is a filter context) depends on the user selections on the pivot table or on the shape of the DAX query. When you use *SUM([SalesAmount])* in a measure, you mean the sum of all the cells that are aggregated under this cell, whereas when you use *[SalesAmount]* in a calculated column, you mean the value of the *SalesAmount* column in this row.

When you create a measure, you can define a value that changes according to the filter that the user applies on a pivot table. In this way, you can solve the problem of calculating the gross margin percentage. To define a measure, you can click anywhere inside the measure grid and write the following measure formula by using the assignment operator *:=*.

```
GrossMarginPct := SUM( FactInternetSales[Gross Margin] ) / SUM( FactInternetSales[SalesAmount] )
```

You can see the formula bar in Figure 4-5.

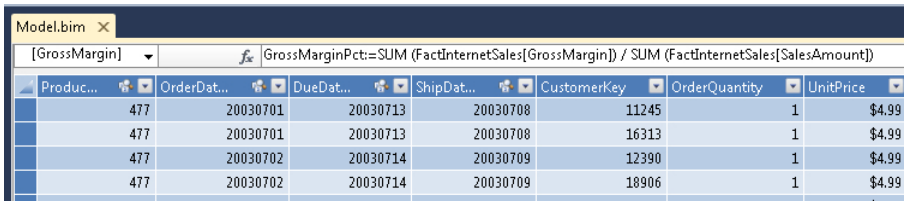


FIGURE 4-5 You can create measures in the formula bar.

After the measure is created, it is visible in the measure grid, as you can see in Figure 4-6.

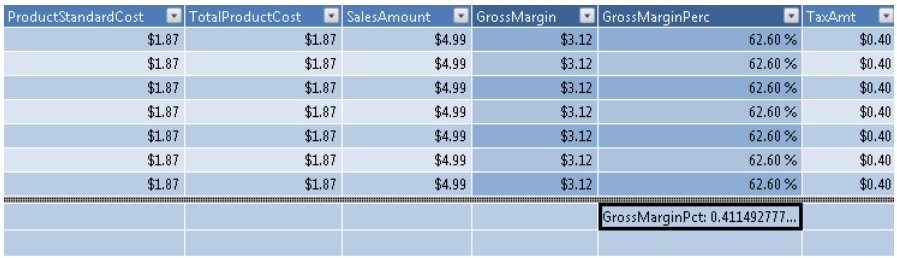


FIGURE 4-6 Measures are shown in the measure grid.

A few interesting things about measures are shown in the measure grid. First, the value shown is dynamically computed and takes filters into account. Thus, the value 0.41149... is the gross margin in percentage for all *AdventureWorks* sales. If you apply a filter to some columns, the value will be updated accordingly.

You can move the measure anywhere in the measure grid by using the technique of cut and paste. To move the measure, cut it and paste it somewhere else. Copy and paste also works if you want to make a copy of a formula and reuse the code.

Measures have more properties that cannot be set in the formula. They must be set in the Properties window. In Figure 4-7, you can see the Properties window for the example measure.

FIGURE 4-7 Measures properties are set in the Properties window.

The Properties window is dynamically updated based on the format of the measure. In Figure 4-7, you can see that the default format for a measure is General. The General format does not have any formatting property. Because you want to format the measure as a percentage (0.41149 really means

41.15%), change the format to Percentage. The updated Properties window (see Figure 4-8) now shows the number of decimal places among the properties of the measure.

FIGURE 4-8 The properties of a measure are updated dynamically based on the format.

Editing Measures by Using DAX Editor

Simple measures can be easily authored by using the formula bar, but, as soon as the measures start to become more complex, using the formula bar is no longer a viable option. Unfortunately, SQL Server Data Tools (SSDT) does not have any advanced editor in its default configuration.

As luck would have it, a team of experts has developed DAX Editor, a Microsoft Visual Studio add-in that greatly helps in measure authoring. You can download the project from CodePlex at *http://daxeditor.codeplex.com*.

DAX Editor supports IntelliSense and automatic measure formatting and enables you to author all the measures in a project by using a single script view, which is convenient for developers. In addition, DAX Editor enables you to add comments to all your measures, resulting in a self-documented script that will make your life easier when maintaining the code.

In Figure 4-9, you can see the DAX Editor window with a couple of measures and some comments.

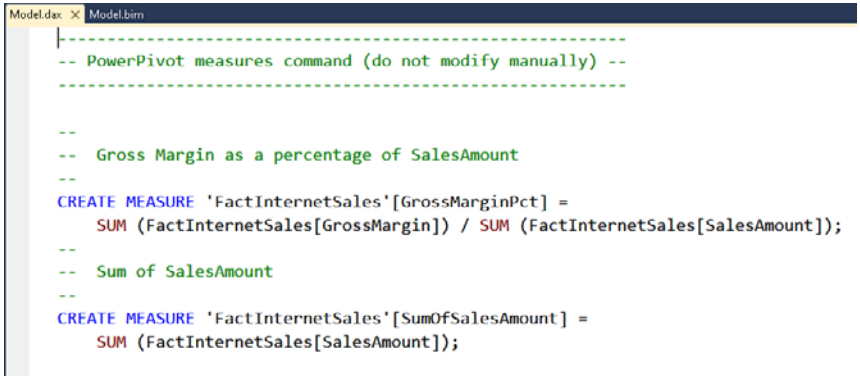


FIGURE 4-9 DAX Editor has syntax highlighting and many useful functions to author DAX code.

We do not want to provide here a detailed description of this add-in, which, being on CodePlex, will be changed and maintained by independent coders, but we strongly suggest that you download and install the add-in. Regardless of whether your measures are simple or complex, your authoring experience will be a much better one.

Choosing Between Calculated Columns and Measures

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

You must define a calculated column whenever you intend to do the following:

- Place the calculated results in an Excel slicer or see results in rows or columns in a pivot table (as opposed to the Values area).
- Define an expression that is strictly bound to the current row. (For example, *Price * Quantity* must be computed before other aggregations take place.)
- Categorize text or numbers (for example, a range of values for a measure, a range of ages of customers, such as 0–18, 18–25, and so on).

However, you must define a measure whenever you intend to display resulting calculation values that reflect pivot table selections made by the user and see them in the Values area of pivot tables, for example:

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

Some calculations can be achieved by using calculated columns or measures, even if different DAX expressions must be used in these cases. For example, you can define *GrossMargin* as a calculated column.

```
= FactInternetSales[SalesAmount] - FactInternetSales[TotalProductCost]
```

It can also be defined as a measure.

```
= SUM( FactInternetSales[SalesAmount] ) - SUM( FactInternetSales[TotalProductCost] )
```

The final result is the same. We suggest you favor the measure in this case because it does not consume memory and disk space, but this is important only in large datasets. When the size of the database is not an issue, you can use the method with which you are more comfortable.

Cross References

It is obvious that a measure 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 a measure; in this way, it forces the calculation of a measure for the context defined by the current row. This might yield strange results, which you will fully understand and master only after having read and digested Chapter 5. This operation transforms and consolidates the result of a measure into a column, which will not be influenced by user actions. Only certain operations can produce meaningful results, because usually a measure makes calculations that strongly depend on the selection made by the user in the pivot table.

Handling Errors in DAX Expressions

Now that you have seen some basic formulas, you learn how to handle invalid calculations gracefully if they happen. A DAX expression might contain invalid calculations because the data 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 but is 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 some special handling.

Before you learn how to handle errors, the following list describes the different kinds of errors that might appear during a DAX formula evaluation. They are:

- Conversion errors
- Arithmetical operations errors
- Empty or missing values

The following sections explain them in more detail.

Conversion Errors

The first kind of error 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 with examples, all 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. What about the following expression?

```
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 determine whether they will be converted and the expression will be evaluated correctly. If some of the content cannot be converted to suit the operator 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 must write more complex DAX expressions that contain error detection logic to intercept error conditions and always return a meaningful result.

Arithmetical Operation Errors

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.

Division by zero, in DAX, requires special handling because it behaves in a way that is not very intuitive (except for mathematicians). When you divide a number by zero, DAX usually returns the special value Infinity. Moreover, in the very special cases of 0 divided by 0 or Infinity divided by Infinity, DAX returns the special *NaN* (not a number) value. These results are summarized in Table 4-2.

TABLE 4-2 Special Result Values for Division by Zero

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

Note that *Infinity* and *NaN* are not errors but special values in DAX. In fact, if you divide a number by Infinity, the expression does not generate an error but returns 0.

```
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 DAX detects errors like this, it blocks any further computation of the expression and raises an error. You can use the special *ISERROR* function to check whether an expression leads to an error, something that you use later in this chapter. Finally, even if special values such as *NaN* are displayed correctly in the SSDT window, they show as errors in an Excel PivotTable, and they will be detected as errors by the error detection functions.

Empty or Missing Values

The third category of errors is not a specific error condition but the presence of empty values, which might result in unexpected results or calculation errors.

DAX handles missing values, blank values, or empty cells by a special value called *BLANK*. *BLANK* is not a real value but a special way to identify these conditions. It is the equivalent of *NULL* in SSAS Multidimensional. 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.

```
= BLANK()
```

On its own, this expression is useless, but the *BLANK* function itself becomes useful every time you want to return or check for an empty value. For example, you might want to display an empty cell instead of 0, as in the following expression, which calculates the total discount for a sale transaction, leaving 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—it is considered 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 you can see 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 does not happen for 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
BLANK() = 0               = TRUE
```

Understanding the behavior of empty or missing values in a DAX expression and using *BLANK()* to return an empty cell in a calculated column or in a measure are important skills to 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 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 with 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 for 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 *IFERROR* 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 *BLANK*; otherwise the product of *Quantity* and *Price* is returned.

In the second expression, the result is *BLANK* 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*. The following expressions are functionally equivalent to the previous ones, but the usage of *IFERROR* in the previous ones makes them shorter and easier to understand.

```
= 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 as that tested for an error; you do not have to duplicate the expression, and the resulting formula is more readable and safer in case of future changes. You should use *IF*, however, when you want to return the result of a different expression when there is an error.

For example, the *ISNUMBER* function can detect whether a string (the price in the first line) can be converted to a number and, if it can, calculate the total amount; otherwise, a *BLANK* can be returned.

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

The second example detects whether the argument for *SQRT* is valid, calculating the square root only for positive numbers and returning *BLANK* for negative ones.

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

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

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

Common DAX Functions

Now that you have seen the fundamentals of DAX and how to handle error conditions, take a brief tour through the most commonly used functions and expressions of DAX. In this section, we show the syntax and the meaning of various functions. In the next section, we show how to create a useful report by using these basic functions.

Aggregate Functions

Almost every Tabular data model must operate 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, rows, columns, and filter conditions in a pivot table whenever it is used in a measure.

In Table A-1 of the Appendix, you can see the complete list of aggregated functions available in DAX. The four main aggregation functions (*SUM*, *AVERAGE*, *MIN*, and *MAX*) operate on only numeric values. These functions work only if the column passed as argument is of numeric or date type.

DAX offers an alternative syntax to these functions to make the calculation on columns that can contain both numeric and nonnumeric values such as a text column. That syntax adds the suffix A to the name of the function, just to get the same name and behavior as Excel. However, these functions are useful for only columns containing *TRUE/FALSE* values because *TRUE* is evaluated as 1 and *FALSE* as 0. Any value for a text column is always considered 0. Empty cells are never considered in the calculation, so even if these functions can be used on nonnumeric columns without returning an error, there is no automatic conversion to numbers for text columns. These functions are named *AVERAGEA*, *COUNTA*, *MINA*, and *MAXA*.

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.

You have four functions by which 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.

Finally, the last set of aggregation functions performs calculations at the row level before they are aggregated. This is essentially the same as creating a column calculation and a measure calculation in one formula. This set of functions is quite useful, especially when you want to make calculations by 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 measure with this expression.

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

This function calculates the product of *Quantity* (from the Sales table) and *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 speaking, all the aggregation functions ending with an X suffix behave this 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 aggregation function (*SUM*, *MIN*, *MAX*, or *COUNT*)

applied to the result of those calculations. We explain this behavior further in Chapter 5. Evaluation context is important for understanding how this calculation works. The X-suffixed functions available are *SUMX*, *AVERAGEX*, *COUNTX*, *COUNTAX*, *MINX*, and *MAXX*.

Among the counting functions, one of the most used is *DISTINCTCOUNT*, which does exactly what its name suggests: counts the distinct values of a column, which it takes as its only parameter.

DISTINCTCOUNT deserves a special mention among the various counting functions because of its speed. If you have some knowledge of counting distinct values in previous versions of SSAS, which implemented Multidimensional only, you already know that counting the number of distinct values of a column was problematic. If your database was not small, you had to be very careful whenever you wanted to add distinct counts to the solution and, for medium and big databases, a careful and complex handling of partitioning was necessary to implement distinct counts efficiently. However, in Tabular, *DISTINCTCOUNT* is amazingly fast due to the nature of the columnar database and the way it stores data in memory. In addition, you can use *DISTINCTCOUNT* on any column in your data model without worrying about creating new structures, as in Multidimensional.

Note *DISTINCTOUNT* is a function introduced in the 2012 version of both Microsoft SQL Server and PowerPivot. The earlier version of PowerPivot did not include the *DISTINCTCOUNT* function and, to compute the number of distinct values of a column, you had to use *COUNTROWS(DISTINCT(ColName))*. The two patterns return the same result even if *DISTINCTCOUNT* is somewhat easier to read, requiring only a single function call.

Following what you have already learned in Chapter 1, "Introducing the Tabular Model," if you have a previous SSAS cube that has many problematic *DISTINCTCOUNT* results, measuring performance of the same solution rewritten in Tabular is definitely worth a try; you might have very pleasant surprises and decide to perform the transition of the cube for the sole presence of *DISTINCTCOUNT*.

Logical Functions

Sometimes you might need 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. You have already seen in the previous section, "Handling Errors in DAX Expressions," the two most important functions of this group, which are *IF* and *IFERROR*. In Table A-3 of the Appendix, you can see the list of all these functions (which are *AND*, *FALSE*, *IF*, *IFERROR*, *NOT*, *TRUE*, and *OR*) and their syntax. 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 the *IFERROR* and the *Price* column contains an invalid number, the result for the calculated column would be an error because if a single row generates a calculation error, the error is

propagated to the whole column. The usage of *IFERROR*, however, intercepts the error and replaces it with a blank value.

Another function you might put inside 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 column *Size* 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"))))
```

The following is a more convenient way to express the same formula, by using *SWITCH*.

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

The code in this latter expression is more readable, even if it is not faster, because, internally, switch statements are translated into nested *IF* calls.

Information Functions

Whenever you must analyze the data type of an expression, you can use one of the information functions that are listed in Table A-4 of the Appendix. All these functions return a TRUE/FALSE value and can be used in any logical expression. They are: *ISBLANK*, *ISERROR*, *ISLOGICAL*, *ISNONTEXT*, *ISNUMBER*, and *ISTEXT*.

Note that when a table column is passed as a parameter, the *ISNUMBER*, *ISTEXT*, and *ISNONTEXT* functions always return *TRUE* or *FALSE*, depending on the data type of the column and on the empty condition of each cell.

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 the following.

```
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*.

If, however, you try to use *ISNUMBER*, as in the following expression

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

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 those in Excel, with the same syntax and behavior. You can see the complete list of these functions and their syntax in Table A-5 of the Appendix. The mathematical functions commonly used are *ABS*, *EXP*, *FACT*, *LN*, *LOG*, *LOG10*, *MOD*, *PI*, *POWER*, *QUOTIENT*, *SIGN*, and *SQRT*. Random functions are *RAND* and *RANDBETWEEN*.

There are many rounding functions, summarized here.

```
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 )
ROUNDUP = ROUNDUP( Tests[Value], 2 )
INT = INT( Tests[Value] )
FIXED = FIXED(Tests[Value],2,TRUE)
ISO = ISO.CEILING( Tests[Value], 0.01 )
```

In Figure 4-10, you can see the different results when applied to some test values.

Value	FLOOR	TRUNC	ROUNDDOWN	MROUND	ROUND	CEILING	ROUNDUP	INT	FIXED	ISO
1.12345	1.12	1.12	1.12	1.12	1.12	1.13	1.13	1	1.12	1.13
1.265	1.26	1.26	1.26	1.26	1.27	1.27	1.27	1	1.27	1.27
1.265001	1.26	1.26	1.26	1.27	1.27	1.27	1.27	1	1.27	1.27
1.499999	1.49	1.49	1.49	1.5	1.5	1.5	1.5	1	1.50	1.5
1.51111	1.51	1.51	1.51	1.51	1.51	1.52	1.52	1	1.51	1.52
1.000001	1	1	1	1	1	1.01	1.01	1	1.00	1.01
1.999999	1.99	1.99	1.99	2	2	2	2	1	2.00	2

FIGURE 4-10 Different rounding functions lead to different values.

As you can see, *FLOOR*, *TRUNC*, and *ROUNDDOWN* are very similar, except in the way you can specify the number of digits on which to round. In the opposite direction, *CEILING* and *ROUNDUP* are very similar in their results. You can see a few differences in the way the rounding is done (see row B, in which the 1.265 number is rounded in two ways on the second decimal digit) between the

MROUND and *ROUND* functions. Finally, note that *FLOOR* and *MROUND* functions do not operate on negative numbers, whereas other functions do.

Text Functions

Table A-6 of the Appendix contains a complete description of the text functions available in DAX: they are *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, and are often used in calculated columns to format strings or find specific patterns.

Conversion Functions

You learned that DAX performs automatic conversion of data types to adjust them to the need of the operators. Even if it happens automatically, a set of functions can still 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 parameter, 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 part of the job. DAX has a large number of functions that operate on date and time. Some of them make simple transformations to and from a *datetime* data type, such as the ones described in Table A-7 of the Appendix. These are *DATE*, *DATEVALUE*, *DAY*, *EDATE*, *EOMONTH*, *HOUR*, *MINUTE*, *MONTH*, *NOW*, *SECOND*, *TIME*, *TIMEVALUE*, *TODAY*, *WEEKDAY*, *WEEKNUM*, *YEAR*, and *YEARFRAC*. To make more complex operations 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 is described in Chapter 8, "Understanding Time Intelligence in DAX."

As mentioned before in this chapter, a *datetime* data type internally uses a floating-point number by which the integer part corresponds to the number of days 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 fractions of the day.) Thus, 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.

Relational Functions

Two useful functions that enable you to navigate through relationships inside a DAX formula are *RELATED* and *RELATEDTABLE*. In Chapter 5, you learn all the details of how these functions work; because they are so useful, it is worth describing them 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 other tables unless a relationship is defined in the model between the two tables. However, if the two tables are in 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 it is, apply a reduction factor to the standard cost. To compute such a column, you must write an *IF* that checks the value of the product category, which is not in the FactResellerSales table. Nevertheless, a chain of relationships starts from FactResellerSales, reaching DimProductCategory through DimProduct and DimProductSubcategory, as you can see in Figure 4-11.

FIGURE 4-11 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 relationship and return the related column value. Thus, the formula for the *AdjustedCost* column can be

```
=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, if any, exists in the related table. If no row is related with the current one, *RELATED* returns *BLANK*.

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 are 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 this category, you can create a column in DimProductCategory with this formula.

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

This calculated column will show, for each product category, the number of products related, as you can see in Figure 4-12.

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

FIGURE 4-12 Count the number of products by using *RELATEDTABLE*.

As is the case for *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 your basic set of functions, you can already build something interesting.

Start loading some tables from *AdventureWorksDW* into a new Tabular project. You are interested in DimDate, DimProduct, DimProductCategory, DimProductSubcategory, and FactResellerSales. The resulting data model is shown in Figure 4-13.

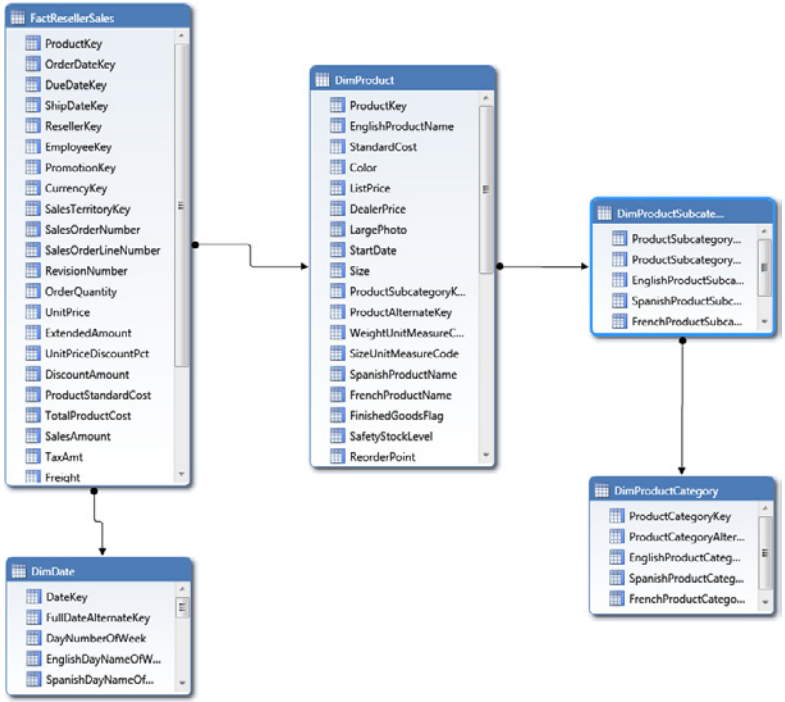


FIGURE 4-13 The Diagram View shows the structure of the demo data model.

To test your new knowledge of the DAX language, use this data model to solve some reporting problems.

First, count the number of products and enable the user to slice them with category and subcategory as 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 just counts the number of products, which we call *NumOfProducts*. The code is the following.

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

Although this measure seems very easy to author, it has an issue. 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 should count it only once. This is a common scenario and can be easily solved by counting the number of distinct values of 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 as follows.

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

You can see in Figure 4-14 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.

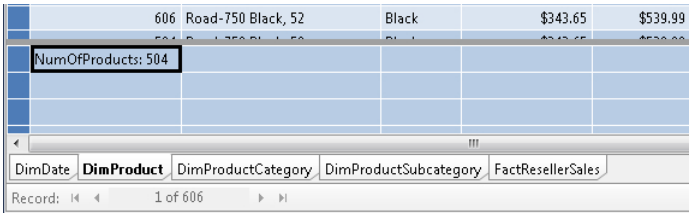


FIGURE 4-14 *DISTINCTCOUNT* is a useful and common function for counting.

This measure is already very useful and, when browsed through Excel, slicing by category and subcategory produces a report like the one shown in Figure 4-15.

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

FIGURE 4-15 This is a sample report using *NumOfProducts*.

In this report, the last two rows are 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 by 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" in both category and subcategory.

Because you must use these values to slice data, this time you cannot use measures; you must author some calculated columns. Put these two calculated columns in the DimProduct table and call them *ProductCategory* and *ProductSubcategory*.

```
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 newly learned functions. The first *IF* checks whether the *ProductSubcategoryKey* is empty and, if so, it searches for the word "nut" inside the product name. *FIND*, in the case of no match, returns an error, and this is why you must surround it with the *ISERROR* function, which intercepts the error and enables you to take care of it as if it is a correct situation (which, in this specific scenario, is correct). If *FIND* returns an error, the result is "Other"; otherwise, the formula computes the subcategory name from the DimProductSubcategory by using the *RELATED* function.

Note Note that 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 can be 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 in case of 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 performances.

With this calculated column, you have solved the issue with the *ProductSubcategory*. The same code, by replacing *ProductSubcategory* with *ProductCategory*, yields to the second calculated column, which makes the same operation with the category.

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

Note that you still must check for the emptiness of *ProductSubcategoryKey* because this is the only available column in DimProduct to test whether the product has a category.

If you now browse this new data model with Excel and use the newly created calculated column on the rows, you get the result shown in Figure 4-16.

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 4-16 You can build a report with the new product category and subcategory, taking care of nuts.

Summary

In this chapter, you explored the syntax of DAX, its data types, and the available operators and functions. The most important concept you have learned is the difference between a calculated column and a measure. Although both are authored in DAX, the difference between them is huge, and you will always have to choose whether a value should be computed by using a calculated column or a measure.

You also learned the following:

- How to handle errors and empty values in DAX expressions by using common patterns.
- The groups of functions available in DAX. These functions can be learned only by using them; we provided the syntax and a brief explanation. During the demo of the next chapters, you learn how to use them in practice.
- With the last exercise, you put into practice some of the key concepts you learned, creating a complete reporting system that, although simple, already shows some of the power of Tabular.

Index

Symbols

64-bit Access Database Engine (ACE) driver, 90
:= (assignment operator), 127
* (asterisk) operator, in hierarchy, 374
/ (forward slash) operator, in hierarchy, 374
- (minus sign) operator, in hierarchy, 374
& operator, for strings, 123, 124
() (parentheses), in DAX, 124
+ (plus sign) operator, and hierarchy aggregation, 373
' (quote characters), for table name, 125
[ ] (square brackets), for column names in DAX, 125
@ symbol, for named parameters, 217
~ (tilde) operator, in hierarchy, 374
Σ (sum) button in toolbar, 45

A

ABC analysis, 399–403
.abf file, 40
ABS function, 139
Access Database Engine (ACE) driver (64-bit), 90
Access, loading data from, 89
account, for Analysis Services service, 30
active relationships, 7, 419
  for Date table, 298
active row context, 151
active rows, defining set of, 148
Add Analysis Services Object window, 550
ADDCOLUMNS function, 192–194, 214
  and filter display, 261
  applying filter context, 259
  FILTER and, 581
  row context from, 240
  row context transformed to filter context, 271
  SQL equivalent, 240
  vs. SUMMARIZE, 195
Add Counters dialog box (Performance Monitor), 565
additive measures, 321
administrative security, 466–469
Administrator setting, for administrative permissions, 469
ADOMD.NET library, 187
  connection, 218, 234
ADO.NET Entity Framework, 512
AdventureWorks DW
  loading tables from, 142
  product-related tables in, 167
  sample database, 40
aggregated sales figures, 230
aggregate values, 127. *See also* measures
  absence in Tabular model, 13
  weighted, 393–395
aggregation functions, 135
  in CALCULATE, 242
  parameters, 158
  for time, 307–321
ALLEXCEPT function, 161, 180, 280
  avoiding circular dependencies with, 399
ALL function, 157, 159, 180, 254, 326
  vs. VALUES, 245
ALLNOBLANKROW function, 165, 597
Allow Partition To Be Processed setting, 553
ALLSELECTED function, 254–255
  FILTERS function and, 262–264
  for visual totals, 253–257
  without parameters, and bug, 257
alter command (XMLA), 40, 445
AMO (Analysis Management Object), 487, 488, 491–493, 545–546
  application for Tabular database creation, 494–507
    calculated columns, 500–501
    cubes, 495
    data source views, 494–495
    measure creation, 498–499

relationships creation, 501–506
SQL Server table loading, 495–498
operations with .NET, 507–509
PowerShell with, 509–510
reference to library, 492
AMO2Tabular project, 487, 507
Analysis Service Properties dialog box, 467
Analysis Services
2012 architecture, 6–11
licensing, 11–12
Multidimensional model, 8–9
SWITCH implementation in RTM version, 317
Tabular model, 6–8
basics, 1–2
connecting
configuring for SQL Server, 41
to instance with SSMS, 71
from outside domain, 482–483
creating report with SSDT, 460
database backup and restore operations, 518
data sources, 75–76
decision to use, 458
Dynamic Management Views (DMVs), 585–587
Excel to browse memory used by, 339
finding process, 559–561
future, 10–11
history, 2
importing deployed project from, 38
loading data from, 90–94, 513
managing instance, 487
metadata from, 234
paging use, 349
security, 77
starting, 513
stored procedures, 480
Tabular models architecture in, 329–330
upgrading from previous versions, 12
xVelocity and, 338
Analysis Services Configuration page, 29
Analysis Services Execute DDL Task control, 549–551
Analysis Services Processing Task, 547
Analysis Services Processing Task Editor, 547–548
Analysis Services Properties window, 561–563
Analysis Services server property, 466
Analysis Services Tabular Project, as new project option, 32
Analyze In Excel dialog box, 54, 439, 471–472
AND condition
for cross filters, 258
in DAX, 124
animated scatter chart, 68
Apache Hadoop, 3
arithmetic operation errors, 132–133
arithmetic operators in DAX, 124
ascending order
for Sort By Column property, 433
for RANKX results, 283
ASC keyword, for sort order, 188
ASCMD utility, 512
executing XML script from, 543
.asdatabase file, 40
assignment operator (:=), 127
asterisk (*) operator in hierarchy, 374
A-suffixed functions, 136
.atomsvc file, 109
Attribute Key, in Multidimensional, 225
attribute relationships, 225, 500, 502
attributes
of dimensions, 8, 384
on slicer, 59
authentication, 482–483
for SQL Server, 81
Windows, 56, 482
automating processing, 539–551
setting for, 34
AVERAGE function, 136, 321
AVERAGEX function, 156
AVGX function, 394

B

backup
Analysis Services database, 518
setting for, 35
workspace database, 30
balances, updating using transactions, 326–328
banding, 410–412
basket analysis, 417–421
Batch element (XMLA), 540
batch execution of process commands, 545
best practices
date tables, 292, 324
hiding Date ID column from user, 299
hierarchy design, 364–365
for multiple Date tables, 305
BI Development Studio, 19. *See also* SQL Server Data Tools (SSDT)
BIDS Helper, 17
installing on development workstation, 27

big tables, working with, 79–80
BIM file, 355
Binary data type, in xVelocity, 347
binary large objects (BLOBs), 347
BI Semantic Model (BISM), 10
BI Semantic Model Connection, 65
  troubleshooting access, 66
BISM Normalizer, 459
  installing on development workstation, 27
bitmap index, for xVelocity dictionary, 337
BLANK function, 133–134
BLANK value, 133
  from LOOKUPVALUE function, 212
  SUMMARIZE and, 198–200
BLOBs (binary large objects), 347
Books Online, 373
Boolean expressions
  in DAX, 124
  as filter parameter in CALCULATE, 244
bridge table, for many-to-many relationship, 414
browsing
  data without filter, 169
  data with period table, 315–318
  workspace database, 54–55
Build menu, Deploy, 52
BuiltinAdminsAreServerAdmins property, 466
business intelligence (BI)
  real-time, 14–15
  self-service and corporate, 4–17
business intelligence (BI) stack, 1
business logic of data warehouse, 1
bus technique of Kimball, 382

C

cache, 516
  clearing
    for query performance monitoring, 578
    by SSAS, 563
  system in DAX, 215
CalcAmount measure, 155
calculated columns, 7, 14
  AMO for creating, 500–501
  for banding, 410
  creating, 48–49, 126
  DirectQuery and, 353, 404
  example, 152
  expression evaluation in row context, 152
  for concatenating employee first and last names, 368
  for hierarchies, 365
  LOOKUPVALUE function for, 408
  vs. measures, 125, 130–131
  memory for, 395
  power in data modeling, 399–403
  RAM use vs. complexity, 403
  reference to measure, 131
  refreshing in table processing, 349
  refresh processing in xVelocity, 348
  row filtering and, 162, 477
CALCULATE function, 157, 176, 183, 237–252
  ALLEXCEPT filter, 387
  behavior recap, 252
  and DatesYTD function, 309
  dependency list for column, 397
  effect of combining FILTER, ALL, and VALUES, 246–251
  filter conditions, 181
  vs. FILTER function, 178
  in FILTER function argument, 244–251
  filter parameters, 177
  in lookup operation, 213
  measure formula in, 239
  nested, 420
  as parameter of CALCULATE, 389
  passing multiple table filter parameters in, 180
  and removing part of filter, 160–161
  result of, 242
  row context and, 173–175
  VALUES function in, 282
  vs. RELATEDTABLE, 167
CALCULATETABLE function, 174, 176, 183, 237–252, 574
  behavior recap, 252
  evaluation order in nested, 191–192
  in FILTER function argument, 244–251
  modifying filter context with, 240–244
  query with FILTER, 189–192
  vs. CONTAINS, 210
calculations
  in DAX, 121–125
  unrelated table for parameters, 318
calendar table, time period selection, 419
CallbackDataID function, 576
CaptureXml property, 545
cardinality of column, in xVelocity, 342
Cartesian product between two tables, 205

cascading many-to-many relationship, 416, 421–424
C# console application, AMO library use in, 491–493
CEILING function, 139
cells
  aggregate functions and empty, 136
  context of, 127
cell security, 477
  missing in Tabular, 15
cellset for MDX, 187, 234
chained relationships
  Power View and, 424
  row context and, 167–168
  USERELATIONSHIP function for inactive, 177
charts
  applying filters to, 70
  in reports, 68–69
circular dependencies, 396–399
ClearCache command (XMLA), 332
clearing
  cache for query performance monitoring, 578
  filters in PivotTables, 59
client side credentials, 78–79
client tools
  for data analysis, 15
  DirectQuery and, 404
Clipboard, loading data from, 100–103
ClosingBalance, 324–326
CLOSINGBALANCEYEAR function, 311, 325
ClosingEOM measure, 325
ClosingEOQ measure, 325
ClosingEOY measure, 325
ClosingMonth, formula for, 325
ClosingQuarter, formula for, 325
ClosingYear, formula for, 325
cloud, Analysis Services and, 11
C# (Microsoft), 218
CodePlex, 130, 487
coherent data, in data source, 117
Collation property, for Tabular model, 438
column in Grid View, filtering, 45
column-oriented databases, row-oriented vs., 334–336
columns in PivotTable
  selecting, 57
  sorting and filtering, 60–62
columns in tables, 7. *See also* calculated columns
  best practices for referencing, 125
  choosing for partition, 521
  detecting type in loaded CSV files, 99
  filtering, 85, 257–261
    constraint, 252
  formatting, 436–440
  grouping data by, 194. *See also* SUMMARIZE function
  hiding, 364, 431–432
  memory cost in xVelocity, 347
  merging for multiple column relationships, 409
  names, 94, 429
    changing inside view, 304
    in copied tables, 303
    in Excel, 96
    vs. measure name, 217
  as primary key, 443
  propagating filter context between, 207
  reducing number of values of, 343
  reference in DAX expression, 126
  reference to, in another table, 141
  returning last date of, 313
  sorting data, 432–436
  SSDT setting to ignore headers, 102
  table identifier use to reference column, 214
  working with, 44
command line, executing XMLA script from, 543
Command Type DMX mode, Query Designer in, 220
comma-separated values (CSV) files, 75
  loading data from, 98–99
comments for measures, 129
comparison
  DAX functions for time, 307–321
  DAX operators, 124
composite key, 408–409
compressing table, Vertipaq process, 118
CONCATENATE function, 140
concatenating text in DAX, 124
Conceptual Schema Definition Language (CSDL), 512
configuration table, for banding, 410
connecting
  to Analysis Services
    from outside domain, 482–483
    with SSMS, 71
  components for, 79
  to data source, opening existing, 88–89
  to deployed database, 55–57
  Power View to Tabular model, 65–66
  slicer to multiple PivotTables, 59
  to Tabular model, 54–64
    driver for, 187
Connection Properties dialog box, 473
connection string properties, testing roles with, 472–474

constrained delegation, 483
CONTAINS function, 209–211
context of cell, 127
control filters
  ALLSELECTED for visual totals, 253–257
  and selections, 252–272
controls for slicer, resizing and moving, 68
conversion DAX functions, 140
conversion errors in DAX formula, 131
converting
  PivotTable to formulas, 63
  Tabular model to Multidimensional model, 223
copy and paste Excel table into Tabular model, 295
Copy command (SSMS), for partition, 526
corporate business intelligence (BI), 4–17
COUNTA function, 136
COUNTAX function, 156
COUNTBLANK function, 136
COUNT function, 136, 321
COUNTROWS function, 136, 175, 176, 264
COUNTX function, 156
CPU
  SSAS Tabular use of, 561
  xVelocity requirements, 515–516
CPUTime, 577
Create Role dialog box (SSMS), 464, 465
credentials, server and client side, 78–79
cross-filters, 257–266
CROSSJOIN function, 203–204, 207, 427, 579, 580
  and filter loss, 268
  GENERATE vs., 206
CROSS JOIN statement (SQL), 207
cross references, measures and calculated columns, 131
cross table filtering, 173
CSDL Extension for Tabular Models, 512
cubes, 8, 338
  adding calculations to, 500
  AMO to create, 495
  Excel viewing model as, 56
  formulas in Excel, 63–64
  online analytical processing (OLAP), 91
  setting for name, 35
CubeValue() function, 64
currency
  conversion, 425–428
    optimization, 578–579
  data type, in xVelocity, 347
CURRENCY function, 140
Currency Symbol, Data Format property values for, 437
CurrentListPrice, computing, 388
current row, 151
Current Windows User, as default Analyze in Excel dialog box option, 54
custom application, query results integrated in, 234
CUSTOMDATA function, 479, 480–481
cut/copy and paste, to move measures, 46, 128

D

data
  grouping by year, 196
  preparation for hierarchy, 363
  refreshed by server, 76
Data Analysis eXpressions (DAX), 7. *See also* DAX (Data Analysis eXpressions)
Data Backup model property, 35
Database name, for SQL Server connection, 81
Database Properties window in SSMS, 357
database roles, 463, 468–469
  creating, 464–465
databases. *See also* workspace database
  production, processing, 79
  row-oriented vs. column-oriented, 334–336
  in Tabular model, 6
  views for decoupling from, 405–406
Data Connection Wizard, 56
DataDir property, of Analysis Services instance, 338
data feeds, 75
  loading data from, 110–112
  for loading reports, 108–110
data feed URL, for Table Import Wizard, 110
data file, 339
data filtering, 85. *See also* filtering
Data Format property, 436–437
DataID, 338
DataMarket. *See* Windows Azure Marketplace
data marts, design, 384
data memory usage, in xVelocity, 340–341
Data Mining Extension (DMX), 219
data modeling in Tabular, 381
  calculated columns power, 399–403
  circular dependencies, 396–399
  data stored, 388
  dimensional models, 384–393
    degenerate dimensions, 389–390
    slowly changing (SCD), 386–389
    snapshot fact tables, 390–393

techniques, 381–384
OLTP database, 383–384
views to decouple from database, 405–406
weighted aggregations, 393–395
with DirectQuery enabled, 403–405
data models
for currency conversion, 426
for computing ABC classes, 400
for many-to-many relationships, 413
with advanced relationships, querying, 421–424
data processing. *See also* processing
for table, 350
in xVelocity, 348
Data Quality Services, 4
data security, 469–478. *See also* security
testing, 471–474
Data Source Properties window, 221
data sources, 75–76
C# code to add to database, 493
OLE DB connection type for defining, 220
Windows account credentials for loading information from, 77
data source views, 488
AMO for, 494–495
loading table metadata in, 495
data storage, PowerPivot, 451
Data tab on ribbon (Excel)
Connections button, 473
From Other Sources button, 55, 56
data types
choosing in xVelocity, 346–348
in DAX, 123–125
in data sources, 117
data warehouses, 2
Analysis Services and, 1
design approaches, 381
DATEADD function, 311
and error, 319–321
date and time functions, 140
Date, as DAX reserved word, 176
date column
as LASTDATE parameter, 323
in xVelocity, 343
for TOTALYTD function, 309
Date data type
Data Format property values for, 437
for DAX functions, 306
in xVelocity, 347
date filters, for importing data, 85
DATE function, 140
date hierarchy in Excel, 362
dates. *See also* time intelligence
handling wrong or missing, 296
managing granularity, 300–301
measures for calculations related to, 302
DATESBETWEEN function, 313
DateStream on Windows Azure Marketplace,
importing Date table from, 296
DATESYTD function, 308, 311, 312
CALCULATE function and, 309
Date tables
best practices, 324
creating, 292–296
defining relationships with, 296–301
duplicating, 302–305
generating dates in SQL query, 293–295
importing from DateStream on DataMarket, 296
importing from Excel, 295
marking table as, 443
metadata for, 306–307
separating time from date, 298–300
sort order for month and day names, 306
Table Import Wizard for, 303
Tabular modeling with, 291–307
using duplicate, 305
DATETIME column
including milliseconds, 346
transforming into columns for date and time, 344
datetime data type, 124, 140
DAX (Data Analysis eXpressions), 7
automatic detection of relationships, 175
cache system, 215
for calculated columns, 126
editing expression for, 48
calculations, 121–125
data types, 123–125
drillthrough in, 231
error handling, 131–135
formulas for many-to-many relationships, 412
functions
aggregate functions, 135
basic use, 142–145
conversion functions, 140
date and time functions, 140
information functions, 138–139
logical functions, 137–138
mathematical functions, 139–140
relational functions, 141–142
text functions, 140
implementing unary operators with, 374–379
MDX vs., 185, 233–234
for measures, 46, 127

operators, 124–125
performance, vs. MDX, 225
syntax, 121–122
values, 125
DAX Editor
for measure editing, 129–130
installing on development workstation, 27
DAX filter expressions, for data security, 469
DAX optimizer, SQL Server optimizer vs., 569, 572
DAX queries
column names in, 94
converting in SQL statement, 557
evaluation context in, 238–240
executing, 73
on Multidimensional model, Microsoft support of, 9
ORDER BY clause, 188–189
parameters, 217–223
SQL Server Management Studio to run, 73, 157
in SQL Server Reporting Services, 219–224
steps to optimize and execute, 570
syntax, 187–189
syntax for measures, 214
in Table Import Wizard, 93
testing in Query Designer, 222
DAX query plans, 569
generation, 332
MDX query translated into, 584
DAX Storage Engine, caching by, 332
day names
Date table sort order for, 306
sorting by, 433
decimal numbers, in xVelocity, 347
Decimal Number type, Data Format property values for, 437
decoupling from database, views for, 405–406
decoupling layer, views as, 304
dedicated development server, 20
default development server, 33
Default Environment Settings page, 31
Default Field Set dialog box, 432, 441
Default Field Set for table, 441–442
default format for measure, 128
Default Image property, for Table Behavior, 443
default instance, for development server install, 28
Default Label property, for Table Behavior, 443
Default Properties Wizard, 33
DEFINE MEASURE function, 214, 239
degenerate dimensions, 385, 389–390
DegenerateMeasureGroupDimension measure group, 497
de Jonge, Kasper, 310
delegation, constrained, 483
Delete command (SSMS), for partition, 526
deleting
columns, 45
partitions with AMO, 509
relationship, 51
denormalizing columns, 408
DENSE argument, for RANKX function, 276
dependencies
between database table structure and Tabular solution, 406
circular, 396–399
deployed database, connecting to, 55–57
deployed project, importing from Analysis Services, 38
deploying Tabular model
after prototyping in PowerPivot, 460–461
automating to production server, 517–518
in Diagram view, 52–53
processing options, 527–539
sizing server, 513–517
table partitioning, 518–527
.deploymentoptions file, 40
Deployment Options\Processing Option property, 34
deployment properties of project, DirectQuery, 355
Deployment Server\Cube Name property, 35
Deployment Server\Database property, 35
Deployment Server\Edition property, 34
Deployment Server\Server property, 34
.deploymenttargets file, 40
Deployment Wizard, 517
derived columns, 7. *See also* calculated columns
descending order for RANKX results, 283
DESC keyword, for query sort order, 188
design mode, for Query Designer, 219
developer environment
components, 19–21
installation process, 21–30
setup, 19–30
SQL Server Developer Ediiton for, 21
development database, vs. workspace database, 21
development server, 20
database metadata deployment to, 52
installation, 27–30
development time, model selection and, 11
development workstation, installation, 21–27

Diagram view, 49–52
demo data model, 143
hiearchy design in, 364
hierarchy creation in, 52
model deployment, 52–53
dictionary, 338
creating and updating, 349
size reduction impact on performance, 342–346
for xVelocity values, 337
dimensional models, 384–393
degenerate dimensions, 389–390
Kimball's methodology and, 382
slowly changing (SCD), 386–389
snapshot fact tables, 390–393
dimension folder, 338
DimensionID element (XMLA), 541
dimensions, 8, 384
defining properties, 489
junk, 343
in Multidimensional data model, 489
processing before measure groups, 348
security on measures, missing in Tabular, 15
shared relationship, 503
dimension table, self-join for parent/child hierarchy, 367
dimmed item, on slicer, 172
directories, for development server install, 29
DirectQuery, 14, 351–360
deployment, 551–558
development with, 359–360
implementing partitions for, 552–556
limitations, 353, 404
modeling with enabled, 403–405
monitoring, 585
partitioning strategy, 551–552
query execution in, 352
reasons for using, 353
security and impersonation with, 557–558
server requirements, 517
settings, 355–359
SQL Profiler for event analysis, 354–355
SQL Server for, 118
for Tabular model definition, 329
DirectQuery Mode property, 355
DirectQuery option for DirectQuery Mode setting, 358
DirectQuery with In-Memory mode, 552
DirectQuery with In-Memory option, 358
dirty data, in hierarchy, 376
DISCOVER_OBJECT_MEMORY_USAGE, 586
Disk Space Requirements page
for development server install, 29
disk storage
Tabular vs. Multidimensional models, 13
xVelocity requirements, 513–514
DISTINCTCOUNT function, 137, 143
DISTINCT function, 174, 415
vs. SUMMARIZE, 195
vs. VALUES function, 182
distinct values, VALUES function to return from specified column, 182
divide-by-zero error, 132
IF statement to avoid, 314
DMX (Data Mining Extension), 219
documentation for SQL Server, installing, 24–25
domain user accounts, 463
double-hop problem, Kerberos and, 483
drillthrough, 444–445
in MDX queries, 230–233
operation on measure, 322
partial support in Tabular, 17
Dumas, Cathy, 432
blog, 30
duplicating Date table, 302–305
Duration, SQL Profiler information on, 577
dynamic feeds, loading data from, 75
Dynamic Management Views (DMVs), 585–587
dynamic security, 479–482
with CUSTOMDATA function, 480–482
DAX functions for, 479–480
with USERNAME function, 481

E

EARLIER function, 161–164
EARLIEST function, 163
Edit command (SSMS), for partition, 525
editing projects online, 32
Edit Relationship dialog box, 51
EffectiveUserName connection string property, 472, 483
employee names, calculated columns for concatenating first and last, 368
empty cells, aggregate functions and, 136
empty items in parent/child hierarchies
handling, 372–373
empty values, 133–134
ENDOFYEAR function, 311
end-user reporting, Analysis Services benefits for, 2

errors
arithmetical operation errors, 132–133
circular dependency causing, 396
from drillthrough on MDX calculated measure, 231
empty or missing values, 133–134
from duplicate measure and column names, 217
from MDX vs. DAX local measures, 230
from measure definition referencing row value, 153
from removing column from group, 281
from sorting by column, 434–436
in DAX, 131–135
intercepting errors, 134
from LOOKUPVALUE function, 212
and year-to-date calculation, 310
estimating database size, for xVelocity, 340
EVALUATE function, 187
ORDER BY clause, 200
evaluation context
adding filters to, 173
in DAX queries, 238–240
EARLIER function, 161–164
final considerations, 183
in multiple tables, 164–183
in single table, 147–161
measure in, 239
testing in SSDT and SSMS, 156–157
types, 147
evaluation order, in nested CALCULATETABLE, 191–192
EXACT function, 140
Excel, 4, 27. *See also* PowerPivot
as client for complex relationship models, 424
connecting to deployed database, 55–57
copying table content to clipboard, 100
cube formulas, 63–64
date hierarchy, 362
DirectQuery and, 14, 404
editing connection string in, 473
importing data saved in SharePoint, 112
importing Date table from, 295
limitation as data source, 117
loading data from, 95–97
PowerPivot database process inside, 5
querying Tabular model in, 53–64
sample table for loading in Tabular, 95
slicers, 58–60, 148–149
testing roles with, 471–472
to browse memory used by Analysis Services, 339
workbook containing PowerPivot model, 37
exchange rate for currency, table for, 426
Exclude First Row Of Copied Data setting, 102
Execute button (SSMS toolbar), 73
ExecuteCaptureLog command, 545
Execute method (XMLA), 218
Existing Connection dialog box, 88
Existing Session event, trace for, 485
EXISTS keyword (SQL), 211
EXP function, 139
Export To Data Feed icon, 108
Expression editor, 222

F

FACT function, 139
fact relationship, in Tabular data model, 504
facts
in dimensional model, 385
dimension relationships with, 384
fact tables
attributes stored in, 389
currency for, 425–426
measures represented in, 345
names for, 430
snapshot, 390–393
fault tolerance, 516
Feature Selection page
for development workstation install, 23, 24
for development server install, 27, 28
Field List, in PowerPivot, 452–454
File menu (SSDT), New\Project, 31
File menu (SSMS), New\Analysis Services MDX Query, 185
file names of .bim file, 36
filter context, 147
applying to ADDCOLUMNS function, 259
CALCULATE to transform row context, 174
changing only part, 160
checks for multiple active values, 316
in single table, 148–151
interaction with row context, 173–177
many-to-many relationship and, 414
modifying with CALCULATETABLE, 240–244
with multiple tables, 168–173
modifying, 177–182
order of evaluation, 192
propagating between columns, 207
removing filter from, 158
row context transformed to, 260, 271, 397

FILTER function, 157, 183, 414
ADDCOLUMNS and, 581
as aggregation function parameter, 158
CALCULATE function vs., 178
CALCULATETABLE function with, 189–192
in CALCULATE and CALCULATETABLE arguments, 244–251
filtering
adding to evaluation context, 173
applying to charts, 70
browsing data without, 169
CallbackDataID and, 576
column in Grid View, 45
constraint in CALCULATE function, 252
CONTAINS as condition, 211
and cross-filters, 257–266
data display in table, 44
hierarchies for, 362
for importing to table, 85
location for optimizing, 580–581
maintaining complex, 267–272
MDX query WHERE condition for, 227
in measure, 216
on lookup table, propagating to related tables, 173
order for conditions in CALCULATE function, 181
overriding, 180
for partitions, 522
PivotTable rows and columns, 60–62
removing in PivotTables, 59
removing in SQL, 242
slicers for, 58–60
FILTERS function
ALLSELECTED function and, 262–264
VALUES and, 261–262
FILTER statement, 173
FIND function, 140, 144
firewalls, 43
FIRSTDATE function, 326
FIRSTNONBLANK function, 324
fiscal year, calcuating year-to-date measure over, 310
Fit-to-Screen button (Diagram View), 50
FIXED function, 139, 140
FLOOR function, 139
foreign key
of existing values only, 199
Tabular and, 408
FORMAT function, 140
formatting
columns in tables, 436–440
measures, 437
formula bar
measure definitions in, 47, 128
resizing, 47
Formula Engine (FE), 574
vs. Storage Engine, 334
formulas
converting PivotTable to, 63
inserting line break in, 47
forward slash (/) operator in hierarchy, 374
fragmentation, dictionary update and, 349
free-form reports in Excel, 64
freezing columns, 45
Friendly connection name, for SQL Server, 81
from table, 502

G

gallery in SharePoint, for PowerPivot workbooks, 456
General format
for data types, 437
for measures, 128
GENERATEALL function, 205, 207
GENERATE function, 205, 207, 422
vs. CROSSJOIN, 206
for applying TOPN, 275, 280
globally unique identifier (GUID), 339
Grand Total row, 239
granularity
of dates, managing, 300–301
of process request, 527
of snapshot fact table, 391
Grid view, 43, 44
calculated column creation, 48–49
creating measures, 45–48
filtering column, 45
gross margin, 126
GrossMargin calculated column, formula, 122
grouping data
by banding, 410–412
by year, 196
MDX assumption about, 226
removing in SQL, 242

H

HardMemoryLimit performance counter, 567
HardMemoryLimit setting for SSAS, 563
hardware, Multidimensional vs. Tabular needs, 13–14
HASONEFILTER function, 264–266
  when to use, 265–266
HASONEVALUE function, 264–266
help, installing for SQL Server, 24
Help Library application, 25
Hide From Client tool command, 317
hiding measures, 317
.HIDX file, 339
hierarchies, 179, 339
  basics, 361–362
  browsing time with, 300
  building, 52, 363–364
    guidelines on when, 363
  DAX and, 228
  defining on tables, 8
  design best practices, 364–365
  design in Diagram View, 364
  MDX for navigating into, 227
  names of, 429
  parent/child, 367–379
    support in Multidimensional, 16
  in PivotTables, 57
  rebuilding for xVelocity table, 349
  spanning multiple tables, 365–367
  unary operators, 373–379
HOLAP (Hybrid OLAP), 9
Home tab in ribbon (Sharepoint), Full Screen button, 69
HTTP connectivity, 482
HyperThreading, 516

I

.IDF extension, 339
IFERROR function, 134, 137, 320, 411
IF function, 134, 144, 372
  to avoid divide-by-zero error, 314
ImpersonateCurrentUser setting, for Analysis Services, 557
impersonation, 77–78
  DirectQuery for, 557–558
  in Table Import Wizard, 42
  PowerPivot and, 451
  settings, 358
  testing roles by, 474
implicit measures, in PowerPivot, 453–454
imported tables, names for, 83
Import from PowerPivot, as new project option, 32
Import from Server (Tabular), as new project option, 32
importing
  Date table from DateStream on Windows Azure Marketplace, 296
  from PowerPivot, 460
  from PowerPivot, 37–38
inactive relationships, 7, 419
  in SUMMARIZE, 201–202
  row context and, 175–177
indexes, 335, 339
  cost of refreshing, 520
  rebuilding in xVelocity processing, 349
  xVelocity memory-optimized columnstore, 336
indirect filters (cross-filters), 257–266
Infinity value, 132
information DAX functions, 138–139
in-memory database, xVelocity as, 338
In-Memory mode
  and xVelocity, 331–351
  for Tabular model definition, 329–330
  query execution, 331–334
In-Memory option, for DirectQuery Mode settings, 357
In-Memory with DirectQuery mode, 358, 552
Inmon, William, 381
INNER JOIN CROSS APPLY statement (SQL), 207
INNER JOIN statement (SQL), 205
inserting line break in formulas, 47
Insert Slicers dialog box, 58, 149
Insert tab on ribbon (Excel), Slicer button, 58
installation
  development server, 27–30
  development workstation, 21–27
  workspace database server, 30
Install Content From Online page, 26
Instance Configuration page, for development server install, 28
instance node in Object Explorer pane, 72
Integration Services package, for loading Excel workbook, 95
IntelliSense, 47, 48
  DAX Editor support for, 129
intercepting errors, 134
Internet Total Sales measure, 308
INT function, 139, 140
invalid calculations, troubleshooting, 131

I/O, xVelocity requirements, 513–514
ISBLANK condition, testing, 581
ISBLANK function, 138
ISCROSSFILTERED function, 257–261
ISERROR function, 133, 134, 138, 139, 144, 145
ISFILTERED function, 257–261, 372
ISLOGICAL function, 138
ISNONTEXT function, 138
ISNUMBER function, 135, 138, 139
ISSUBTOTAL function, 199
ISTEXT function, 138
IT departments, and power users, 5

J

junk dimensions, 343, 385

K

KEEPFILTERS function, 267–272
Keep In Memory setting, for workspace retention, 36
Keep Unique Rows setting, for table, 443
Kerberos, double-hop problem, 483
Key Performance Indicator dialog box, 446
key performance indicators (KPIs), 8, 445–447
Kimball, Ralph, 381, 382

L

L2 cache of processor, 516. *See also* cache
Language property, for Tabular model, 438
LASTDATE function, 313, 322, 323, 325, 327
LASTNONBLANK function, 323, 324
latency, in publishing measure, 216
.layout file, 40
Layout tab on ribbon (Sharepoint), Chart Title\
None, 68
LEFT function, 140
LEFT JOIN OUTER APPLY statement (SQL), 207
LEFT JOIN statement (SQL), 207
LEN function, 140
licensing, 11–12, 21
linear dependencies, 396
line break, inserting in formulas, 47
linked tables
  in PowerPivot, 455
  in SSDT, 101
literals, 125
LN function, 139
load balancing, 516
loading data
  from Access, 89
  from Analysis Services, 90–94
  choosing right method, 116–118
  from Clipboard, 100–103
  from data feed, 110–112
  from data source
    Windows account credentials for, 77
  from Excel, 95–97
  from list of tables, 83–85
  recommendations, 87
  from Reporting Services report, 103–110
  from SharePoint, 112–113
  from SQL query, 87
  into tables, 41–49
  from Tabular database, 92–94
  from text files, 98–99
  from views, 87–88
  from Windows Azure Marketplace, 113
loading relationships, 86
loading reports, data feeds for, 108–110
loading tables from AdventureWorksDW, 142
local measures, 229–230
  defining, 215
  overriding measure in Tabular model, 216–217
local user accounts, 463
LOG10 function, 139
LOG function, 139
logical functions, 137–138
logical operators in DAX, 124
logical query plan, 569
  graphical representation, 573
logon credentials, Windows Authentication for, 56
Log on to the server option, for SQL Server
authentication, 81
lookup, RELATED function for, 213
lookup table, 165, 343
  applying filter, 173
  filter on, 169
  foreign key of existing values only, 199
LOOKUPVALUE function, 211–213
  for employee key values, 370
  for calculated column, 408
  performance impact, 427
  RELATED function vs., 334
LOWER function, 140
LowMemoryLimit setting for SSAS, 563

M

Manage Relationships dialog box, 51
Manage Sets command (Excel), 318
many-to-many relationships, 412
  in basket analysis, 418
  cascading, 416, 421–424
  in Tabular vs. Multidimensional, 416
  partial support in Tabular, 16
  whitepaper, 417
margin percentage column, query creating, 193
Mark As Date Table dialog box, 306
Master Data Services, 4
materialization, 568
mathematical functions, 139–140
MAT (moving annual total), 313
MAX function, 136, 321
MAXX function, 156
MDX editor, 91, 92, 93, 185, 186
MDX Formula Engine, 331
  caching by, 332
MDX language, 7, 9
  DAX vs., 185, 225, 233–234
  DirectQuery model and, 14
  limitations for measure definition, 498–499
  queries with, 223–233
  WHERE condition, 227
MDX queries
  cellset for, 187
  DAX local measures in, 229–230
  designer, 91
  DirectQuery and, 353
  drillthrough in, 230–233
  execution in Tabular model, vs. DAX, 331
  monitoring, 584–585
  in SQL Server Management Studio, 73
  WHERE condition, filter in, 232
MDX script, Tabular and, 225
MDX set expression, New Set dialog box for writing, 62
MEASURE function, 229
measure grid, 46, 127
  key performance indicator in, 447
measure groups
  creating, 497
  in Multidimensional, 489
measures, 8, 126–131
  AMO to create, 498–499
  calculated columns vs., 125, 130–131
  calculations related to dates, 302
  comments for, 129
  counting number of products, 143
  creating, 45–48
    in PowerPivot, 453–454
  data security and, 470
  DAX definition, 46
  DAX Editor for editing, 129–130
  definition
    and CALCULATE behavior, 242
    inside query, 213–217
  definition location, 215
    before EVALUATE, 239
    standard location, 46
  drillthrough operation on, 322
  error message from referencing row value, 153
  formatting, 437
  hiding, 317
  MDX query WHERE condition for filter applied to, 227
  in Multidimensional, 224
  names, 429
    vs. column name, 217
  organizing, 432
  overriding with local measure, 216–217
  in PowerPivot
    field list creation, 452
    implicit, 453–454
  selecting for PivotTables, 57
  semiadditive, 321–328
  testing with query, 216–217
  time intelligence-based, 319–321
Measures dimension, 8
Measure Settings dialog box, 453–454
memory
  calculated columns need for, 126, 395
  configuration for Tabular, 561–564
  cost of xVelocity table, 339
  DirectQuery and, 353
  DMVs for memory occupation by object, 586
  optimizing performance by reducing, 342–348
  performance counters for, 564–568
  for PowerPivot, 451
  Process Full operation and, 531
  query use of, 568
  SSAS Tabular use of, 561
  system bandwidth, 515
  Tabular vs. Multidimensional models, 13
  xVelocity requirements, 339–342, 514–515
Memory\VertiPaqPagingPolicy server setting, 531
Merge command, for partition, 509, 526
Merge Partition window, 526, 527

merging partitions, 520
metadata
from Analysis Services, 234
for database, deployment, 52
for Date tables, 306–307
for project, 40
Microsoft
ADO.NET Entity Framework, 512
Amo2Tabular project, 487, 507
C#, 218
Office 2010, need to install, 27
support of DAX queries on Multidimensional model, 9
Microsoft business intelligence (BI) ecosystem, 1–5
current status of stack, 3–4
Microsoft Visual Basic .NET (VB.NET), 218
Microsoft Visual Studio, DAX Editor as add-in, 129
MID function, 140
milliseconds, in DATETIME column, 346
MIN function, 136, 321
minimap, 50
minus sign (-) operator, in hierarchies, 374
MINX function, 156
missing values, 133–134
Model.bim file, 35, 38, 40, 355
Model menu (SSDT)
Analyze In Excel, 54, 471
Existing Connections, 88, 302
Import From Data Source, 41, 76, 80
Model View\Diagram View, 49
Perspectives, 438
Process\Process All, 52
Roles, 464
Model Properties dialog box, 35
models. *See also* Multidimensional model; Tabular model
choosing for project, 11–17
deployment in Diagram view, 52–53
properties, 35–36
reasons for two, 9–10
MOD function, 139
MOLAP (Multidimensional OLAP), 9, 14
vs. ROLAP, 330
monitoring
DirectQuery, 585
MDX queries, 584–585
security, 484–485
Month In Year column, 434, 435
month names, Date table sort order for, 306
month-to-date (MTD) calculation, 308–311
moving
measures, 128
slicer controls, 68
moving annual total (MAT), 313, 314
MROUND function, 139
MSDN website, 25
msmdpump.dll file, 482
MSMDSRV.EXE (Microsoft Multidimensional Server), 559
msmdsrv.ini file, memory settings, 561
multicolumn relationships, 407–409
Multidimensional database, Tabular data model stored as, 507
Multidimensional model, 6, 8–9, 9
cell security in, 477
converting Tabular model to, 223
data sources, 488
data storage, 8–9
degenerate dimensions in, 389
dynamic security in, 480
features missing in Tabular, 15–17, 225
hardware requirements, 13–14
measure group partitioning, 519
Microsoft support of DAX queries on, 9
MOLAP vs. ROLAP, 330
processing dimensions before measure groups, 348
processing performance compared to Tabular, 13
ragged hierarchies and, 367
reference relationship handling, 503
role-playing dimension in, 302
security, 470
storing measures to be aggregated, 345
Tabular project translation into, 488–491
Tool dimension in, 315
unary operators in, 373
Multidimensional OLAP (MOLAP), 9, 14
vs. ROLAP, 330
Multidimensional sources, 75
multithread-enabled operation, xVelocity Storage Engine as, 332

N

Named Set, 60
creating, 61
names
of columns
changing inside view, 304
in DAX queries, 94

in Excel file, 96
in table copies, 303
for development server instance, 28
in hierarchies, 365
for imported table, 83
measure vs. column, 217
of objects, 429
of Date table, 293
@ symbol for parameters, 217
NaN (not a number) value, 132
natural keys, for dimensions, 384
negative number, square root of, 132
nested CALCULATETABLE, evaluation order in, 191–192
nested FILTER statements, 191
.NET, AMO operations with, 507–509
network-attached storage (NAS), 516
Never Process This Partition in the Processing Option, 553
New BI Semantic Model Connection page, 66
New command (SSMS), for partition, 525
New MDX Query button (SSMS toolbar), 73
New Project dialog box, 32
New Set dialog box, 62
NEXTDAY function, 313
NEXTYEAR function, 311
nonadditive measure, 321
None, default setting for administrative permissions, 468
Non-Uniform Memory Access (NUMA) architecture, 515
NOT operator in DAX, 124
null value, in xVelocity, 348
numbers
automatic conversion of strings to, 123
data types in xVelocity, 346
precision, optimization and, 346
testing text value conversion, 138
transforming to strings, 140

O

Object element (XMLA), 541
Object Explorer pane (SSMS), 72
objects
AMO processing with .NET, 507–508
names of, 429
OData, limitation as data source, 117
Office 2010, need to install, 27
offline help, 25
OLAP database, Analysis Services as, 1
OLAP PivotTable Extensions, 27
OLAP Services, 2. *See also* Analysis Services
OLE DB ADO.NET, 234
OLE DB connection
for defining data source, 220
in Report Builder, 219
OLE DB for OLAP provider, 187
OLTP databases, 381, 383–384
one-to-many relationship, 7
hierarchies and, 363
row filtering and, 476
between tables, 164
online analytical processing (OLAP) cubes, 91
online analytical processing (OLAP) database, 1
online mode, for Multidimensional models, 32
OpenBalance, 324–326
Open Data Protocol, 75, 110
OPENINGBALANCEMONTH function, 326
OPENINGBALANCEQUARTER function, 326
OPENINGBALANCEYEAR function, 311, 326
opening existing connections to data source, 88–89
operators
in DAX, 124–125
overloading, 123
unary, 373–379
optimization, 578–583
by memory usage reduction, 342–348
currency conversion, 578–579
filters location, 580–581
query performance, 2
relationships for, 582–583
Options dialog box, 36–37
OR condition in DAX, 124
OrderBy Attribute setting, in Multidimensional, 225
ORDER BY clause, 238, 273
in DAX queries, 188–189
in EVALUATE, 200
ORDER BY condition, in DAX, 225
Order ID, 344
Original Size button (Diagram View), 50
overriding filters, 180

P

paging
in Analysis Services, 349
disabling, 338
for SQL Server storage space, 335

Parallel Data Warehouse, 3
PARALLELPERIOD function, 311
parameterizing DAX query, Reporting Services
expression for, 223
parameters in DAX queries, 217–223
  defining, 220
parent/child hierarchies, 367–379
  configuring, 368–373
  example, 371
  handling empty items, 372–373
  support in Multidimensional, 16
parentheses [( )], in DAX, 124
Partition Manager dialog box, 520
Partition query, 522
partitions, 359, 518–527
  AMO with .NET for, 508–509
  defining, 520–524
  deleting using AMO, 509
  filters for, 522
  for hybrid mode configuration, 556
  implementing for DirectQuery and hybrid modes, 552–556
  managing, 524–527
  PowerPivot and, 451
  Process Add operation, 534
  Process Data operation, 533–534
  Process Full operation, 533
  processing, 538–539
  reasons for, 519
  strategy for DirectQuery, 551–552
  tables, 518–527
  in xVelocity, 349, 555
Partitions window, editing table partitions in, 525
password, for SQL Server account, 41
Paste Append, 101
pasted tables, 432
Paste Preview window, 100, 102
Paste Replace, 101
PATH function, 368
  DirectQuery and, 405
  output, 369
PATHLENGTH function, 368
pathnames, for reports, 108
percentage
  formatting data as, 437
  Gross Margin as, 127
  year-over-year difference as, 314
performance
  DAX vs. MDX, 225
  ISERROR function and, 145
  LOOKUPVALUE function and, 427
  optimizing by memory usage reduction, 342–348
  snapshot tables and, 393
performance counters, for memory, 564–568
Performance Monitor, 564–568
PerformancePoint Services, 4
period of the prior year (PY), 311–314
period table, browsing data with, 315–318
permissions
  defining in SQL Server, 358
  role membership and, 466
  row filtering using table, 478
perspectives, 8, 438–440
  hierarchy visibility to, 363
Perspectives dialog box, 439
physical query plan, 569
PI function, 139
PivotTable
  converting to formulas, 63
  creating, 56
  drillthrough in, 230, 232
  filter context definition, 148
  key performance indicator in, 447
  queries in MDX, 233
  relationships between cells and table rows, 151
  removing filters, 59
  sample, 58
  sorting and filtering rows and columns, 60–62
  using, 57–58
  with CurrentListPrice and historical ListPrice, 387
  year-to-date calculation for prior year and fiscal year in, 313
PivotTable field list, 454
  Sets folder in, 62
PivotTable\Options tab on ribbon (Excel)
  Calculations button, 60
  Insert Slicer button, 148
  OLAP Tools button, 63
plus sign (+) operator, and hierarchy aggregation, 373
POWER function, 139
PowerPivot, 4, 5, 10, 449–455
  advanced mode, 450
  data storage, 451
  datetime data type, 124
  decision to use, 458
  Field List, 452–454
  implicit measures, 453–454
  importing from, 37–38

linked tables, 455
measures in, 453–454
model embedded in Excel workbook, 37, 112
performance penalty in first version, 346
PivotTable field list, 454
prototyping in, 460–461
sample rows imported from report, 107
Tabular model compatibility with, 12
PowerPivot for SharePoint, 5, 455–458
decision to use, 458
PowerPivot import warning dialog box, 37
PowerPivot tab on ribbon (Excel), 449
PowerPivot window, 450
PowerShell, 546–547
AMO with, 509–510
power users, IT departments and, 5
Power View, 3, 7, 15
blank view in report, 67
building basic report, 66–68
complex relationships and, 424
connecting to Tabular model, 65–66
DAX queries, 233
Default Field Set in, 441–442
manufacturing (RTM) version, 362
querying Tabular model, 65–71
reports interaction, 69–71
report using Default Field Set, 442
Tabular model properties as metadata, 440–444
tile report with images, 444
precision of number, optimization and, 346
previous year, difference over, 314–315
PREVIOUSYEAR function, 311
prices, banding, 410
primary key, specifying column as, 443
Process Add operation, 528
of partitions, 534
Process Clear operation, 528, 532, 533
Process Database window, 535
Process Data operation, 529
in xVelocity, 350
of selected partitions and tables, 533–534
Process Default operation, 529, 530
Process Defrag operation, 530
in xVelocity, 350
Process Full operation, 531
of database, 532
in xVelocity, 350
of selected partitions and tables, 533
processing
automating, 539–551
with SSIS, 547–551
with PowerShell, 546
database, 514
executing, 535–539
options for, 527–539
SSAS memory use during, 561
strategy for, 532–535
tables, 7
processing memory usage, in xVelocity, 341–342
processing time
predictability of source, 117
table partitioning and, 519
Process method, 507
Process Partition option, 538–539
Process Recalc operation, 530, 531
in xVelocity, 350
Process setting, for administrative permissions, 469
Process Table option, 536–538
Process Table window, 537
Script command, 511
product category, computing number of products for each, 193
production database, processing, 79
production projects, prototypes vs., 103
production server, automating Tabular deployment to, 517–518
Project Properties dialog box, 34, 357
projects
building and deploying, 20
choosing model for, 11–17
configuring, 33–37
creating, 31–32
editing online, 32
importing deployed from Analysis Services, 38
metadata for, 40
properties, 33–35
Properties command (SSMS), for partition, 526
Properties window, for measures, 128
prototypes
in PowerPivot, 460–461
vs. production projects, 103
.P suffix, for standard deviation or variance function, 285
PushedDataSource data source, 101

Q

QlikView, 4
QtyGreen measure, 244
quarter-to-date (QTD) calculation, 308–311
queries
  ADDCOLUMNS function, 192–194
  CALCULATETABLE and FILTER, 189–192
  clearing cache for performance monitoring, 578
  CONTAINS for, 209–211
  CROSSJOIN, GENERATE, GENERATEALL for, 203–208
  DAX syntax, 187–189
  in Excel, 8, 53–64
  execution
    in Analysis Services Tabular model, 330
    in In-Memory mode, 331–334
  language options, 185
  LOOKUPVALUE for, 211–213
  MDX for, 223–233
  measure definition in, 213–217
  memory usage, 514
    in xVelocity, 342
  optimizing performance, 2
  performance characteristics, 13
  Performance Monitor for execution, 566–567
  in Power View, 65–71
  ROW function for, 208–209
  SSAS memory use during, 561
  SUMMARIZE, 194–202
  of table data sample, 287–289
  testing measures with, 216–217
  time intelligence-based measures, 319–321
  tools, 185–187
Query Designer
  Command Type DMX mode, 219, 220
  testing DAX query in, 222
Query Editor (SQL Server), 87
Query Mode property, of Tabular project, 356
query plans, 569–578
query projection, 196
quote characters ('), for table name, 125
QUOTIENT function, 139

R

ragged hierarchies, 367
  missing in Tabular, 16
RANDBETWEEN function, 139
RAND function, 139
RANK.EQ function, 284
RANKX function, 276–284
  specifying <value> parameter in, 282
Read and Process setting, for administrative permissions, 469
Read setting, for administrative permissions, 468
real-time reporting, with DirectQuery, 353
reference relationships, in Tabular data model, 504
referencing columns, best practices, 125
refresh interval, for PowerPivot report, 457
RELATED function, 141–142, 166, 174, 183
  for lookup operation, 213
  list of chained relationships for, 167
  LOOKUPVALUE function vs., 334
RELATEDTABLE function, 141, 142, 167, 174, 183
  CROSSJOIN and, 204
  list of chained relationships for, 167
  vs. CALCULATE, 167
related tables, selecting, 86
relational database, 75
  goal in, 342
  vs. OLAP database, 1
relational Data Warehouse, 382
relational DAX functions, 141–142
Relational OLAP (ROLAP), 9, 14
relationships, 7, 224
  active, 419
  AMO for creating, 501–506
  automatic detection by DAX functions, 175
  between measure groups and dimensions, 489
  between table rows and PivotTable cells, 151
  chained, row context and, 167–168
  complex, Power View and, 424
  creating, 50–51
  data models with advanced, querying, 421–424
  defining with Date tables, 296–301
  displaying in Diagram view, 49
  inactive, 419
    row context and, 175–177
    in SUMMARIZE, 201–202
  loading, 86
  many-to-many, 412. *See also* many-to-many relationships
  multicolumn, 407–409
  optimization with, 582–583
  pointing to different tables, 419
  rebuilding for xVelocity table processing, 349
  row filters and, 476–477
  USERELATIONSHIP function for inactive chained, 177
  in xVelocity, 339

remote connections, SQL Server enabled to accept, 43
Remove method, for partition, 509
removing data from table, partitions and, 519
removing filters, in PivotTables, 59
renaming columns, 45
REPLACE function, 140
Report Builder
  data source definition, 221
  OLE DB connection in, 219
Reporting Services. *See* SQL Server Reporting Services
reports. *See also* Power View
  charts in, 68–69
  importing from SharePoint, 112
  pathnames for, 108
  slicers in, 68–69
  SSDT to design, 219
  surfaced by Excel Services, 457
  with tables, 68
reprocessing models, dedicated server for, 20
REPT function, 140
reserved words in DAX, Date, 176, 293
Reset Layout button (Diagram View), 50
resizing
  formula bar, 47
  slicer controls, 68
resources, for Windows process, 560
restore operation, Analysis Services database, 518
ribbon in Excel, PowerPivot tab, 449
RIGHT function, 140
ROLAP (Relational OLAP), 9, 14
  for Multidimensional model, 352
  MOLAP vs., 330
Role Manager dialog box (SSDT), 464, 465, 474
  Permissions drop-down list, 468
  Row Filters configuration, 470
roleplaying dimensions, 418
  in Multidimensional model, 302
  missing in Tabular, 16
roles, 463–466
  database, and administrative permissions, 468–469
  membership in multiple, 466
  Server Administrator, 466–468
  testing
    with connection string properties, 472–474
    with Excel, 471–472
    by impersonating users, 474
Roles connection string property, 472
ROLLUP function, 239
  limitations to use, 422
roll-up rows, in SUMMARIZE results, 198–200
ROUNDDOWN function, 139
ROUND function, 139
rounding functions, 139
  and data storage, 346
ROUNDUP function, 139
row context, 147, 238
  from ADDCOLUMNS function, 240
  and CALCULATE function, 173–175
  and chained relationships, 167–168
  and inactive relationships, 175–177
  in multiple tables, 164–167
  interaction with filter context, 173–177
  nesting, 162
  in single table, 151–157, 164
  transformed to filter context, 239, 260, 271
row filtering
  advanced expressions, 474–478
    for multiple columns, 475
    table relationships and, 476–477
  calculated columns and, 477
  permissions table for, 478
  table relationships and, 476–477
ROW function, 208–209
  evaluating CONTAINS in, 209
row identifier, marking column as, 398
Row Identifier property, 443
RowNumber column, 338, 496, 500, 502
row-oriented databases, column-oriented vs., 334–336
rows in PivotTable, sorting and filtering, 60–62
rows in tables
  checking for existence of condition, 209. *See also* CONTAINS function
  denying access to every, 475
  relationships between PivotTable cells and, 151
  specifying maximum to be returned, 273
Run As Different User option, 474
running totals, 162–163, 401
Russo, Marco, 445

S

SAMEPERIODLASTYEAR function, 311, 313
SAMEPERIODSLASTYEAR function, and error, 319–321
SAMPLE function, 287–289
SANs (storage area networks), 14

scatter chart
animated, 68
example, 70
schema.ini file, 98
schema, star vs. snowflake, 256
Scientific formatting for data, 437
scoped assignments, missing in Tabular, 16
Script Action To New Query Window command (SSMS), 511
SEARCH function, 140
security
administrative permissions, 466–469
authentication, 482–483
data, 469–478
DirectQuery and, 404, 557–558
dynamic, 479–482
monitoring, 484–485
PowerPivot and, 450
roles, 8, 463–466
SSAS, 77
selecting
columns for PivotTables, 57
control filters and, 252–272
measures for PivotTables, 57
related tables, 86
select-object cmdlet, 509
SELECT statement (MDX), 229
SELECT statement (SQL)
GROUP BY condition, 238
GROUP BY, SUMMARIZE compared, 195
WHERE clause, filter context and, 150
self-join on dimension table, for parent/child hierarchy, 367
self-service business intelligence (BI), 4–17
semiadditive measures, 321–328
separating time from date, 298–300
server administrator role, 463, 466–468
Server Configuration page, 29
Server object, C# code creating, 493
servers
data refresh by, 76
development, 20
for default workspace database server, 33
name for SQL Server connection, 81
sizing for Tabular deployment, 513–517
workplace database, 21
server side credentials, 78–79
Service Account, for impersonation, 77
ServiceAccountIsServerAdmin server property, 466
service packs, for SQL Server updates, 22
Sets folder, in PivotTable Field List, 62
.settings file, 40
SETUP.EXE, 21
Setup Support Rules, 22
page, 23
ShareDimensionStorage property, 497
SharePoint, 4. *See also* PowerPivot for SharePoint
creating Power View connection in, 65
loading data from, 112–113
SIGN function, 139
Size Slicer, 241
sizing server for Tabular deployment, 513–517
SKIP argument, for RANKX function, 276
slicers, 58–60, 169
in Excel, 148–149
in reports, 68–69
item dimmed on, 172
making selection on, 169
PowerPivot field list handling of, 452
table of parameters, 318
Slicer Setting dialog box, 172
Slicer Tools\Options tab on ribbon (Excel), PivotTable Connections button, 149
slowly changing dimensional models (SCD), 386–389
slowly changing dimensions, 384, 385
snapshot fact tables, 390–393
snowflaked dimension, 365
snowflake schema, star schema vs., 256
Software as a Service (SaaS) applications, 113
Solution Explorer pane
Show All Files button, 38, 39
Solution Explorer pane, 39
Sort by Column property, 188, 324
of column, 432
of Date table column, 306
in Tabular, 224
in MDX, 238
sort functions, 272–284
sorting, 44
and memory pressure during processing, 341
column data, 432–436
importance of, 118–119
order in Date table for month and day names, 306
PivotTable rows and columns, 60–62
for price banding, 411
sort order, ORDER BY clause, 188–189
source control, for development workstation install, 26
SourceDirect property, 551

source of model, XML file as, 103
source tables, 519
spooling temporary tables, memory for, 568
SQL Azure, 3
  experimental BI tools available, 4
SQL Azure Reporting, 3
SqlDataAdapter, 496
SQL Engine optimizer, vs. DAX Query optimizer, 572
SQL Profiler
  for analyzing In-Memory mode events, 332–334
  for DirectQuery event analysis, 354–355
  running trace, 484
  time information from, 577–578
  to view query plans, 570
SQL queries
  generating dates in, 293–295
  loading data from, 87
SQL Server
  AMO for loading table, 495–498
  configuring Analysis Services connection to, 41
  for DirectQuery, 118
  documentation installation, 24–25
  editions with Analysis Services, 11
  enabled to accept remote connections, 43
  loading from list of tables, 83–85
  relational database engine
    installing, 30
  storage space in pages, 335
SQL Server Agent, 543–544
SQL Server Analysis Services (SSAS), 2. *See also* Analysis Services
SQL Server database, 3
  loading data into workspace database, 80
  semantic layer on top, 353
SQL Server Data Tools (SSDT), 6, 19
  client-side operations, 78
  creating new Analysis Services report with, 460
  creating project, 31–32
  deploying Tabular model from, 517
  editing column names, 303
  linked tables in, 101
  private Tabular database, 21
  Query Mode property setting in, 357
  starting, 31
  testing evaluation context, 156–157
SQL Server designer of Access, 90
SQL Server Installation Center window, 21, 22
SQL Server installer, for development workstation, 21
SQL Server Integration Services (SSIS), 3, 547–551
  process execution from, 544
SQL Server Management Studio (SSMS), 6, 71–73
  browser in, 27
  connection string properties in, 473–474
  database role creation, 464
  DAX query preparation in, 93
  DirectQueryMode property, 357
  opening project in, 101
  Partitions context menu, 525
  for query writing, 185
  scripting to learn XMLA commands, 510
  testing evaluation context, 156–157
SQL Server optimizer, 569
SQL Server Reporting Services, 3
  DAX Query in, 219–224
  exporting data feed, 110
  expression for parameterizing DAX query, 223
  loading data from report, 103–110
  MDX vs. DAX, 234
  web interface, 108
SQL statement, manual changes, Table Preview mode and, 523
SQRT function, 139
square brackets ([ ]), for column names in DAX, 125
square root of negative number, 132
SSDT. *See* SQL Server Data Tools (SSDT)
.S suffix, for standard deviation or variance function, 285
standard deviation, 285
star schema
  basket analysis and, 418
  Date table from, 293
  junk dimensions in, 343
  snowflake schema vs., 256
START AT condition, in DAX query, 189
STARTOFYEAR function, 311
statistical functions, 285–289
STDEV function, 285
STDEVX function, 286
storage area networks (SANs), 14
Storage Engine (SE), 574
  Formula Engine vs., 334
StreamInsight, 3
strings
  automatic conversion to numbers, 123
  concatenation for composite key, 409
  transforming numbers to, 140
  in xVelocity, 347
SubCategories Count measure, 215
SUBSTITUTE function, 140

SUM function, 135, 153, 178, 321, 346, 579
error message from using, 154
SUMX vs., 394
in CALCULATE, 161
SUMMARIZE function, 194–202, 208, 239, 254, 256, 261, 267, 319, 415
inactive relationships in, 201–202
limitations in use, 422
multiple tables in, 196
virtual table for, 270
without adding columns, 197
Sum of Quantity measure, 242
grouped by channel, 238
Sum of SalesAmount measure, 453
SUMX function, 136, 153, 154, 155, 156, 178, 346
SUM vs., 394
in CALCULATE, 161
xVelocity for, 334, 575–576
sum (Σ) button in toolbar, 45
surrogate keys for dimensions, 384
SWITCH function, 138
SWITCH statement, 316
Synchronize Database Wizard, 517

T

Tableau, 4
Table Behavior dialog box, 442
Table Behavior Properties, 442–448
table filter, 177
constraint, 252
table identifier, to reference column, 214
Table Import Wizard
for Access data source, 89
Choose How to Import the Data, 83
as client side operation, 78
connecting SQL Server, 42
Connect to a Microsoft Access Database, 89
Connect to Flat File, 99
Connect to Microsoft Excel File, 95
Connect to Microsoft SQL Server Analysis Services, 90
for Data Feed, 109, 110
data sources list in, 76
for Date table, 303
DAX queries in, 93
for importing from SSDT report, 104–105
impersonation in, 42
Impersonation Information page, 77, 82
loading data from Windows Azure Marketplace, 114
Preview Selected Table, 84
report preview, 106
Select Related Tables button, 86
Select Tables and Views, 43, 84
parameters for SQL Server connection, 81
starting, 41
for text files, 98
Work Item list, 86
Table menu (SSDT)
Create Relationship, 51
Date\Date Table Settings, 306
Date\Mark As Date Table, 306
Manage Relationships, 51
Table Preview mode, 524
table qualifier, in ADDCOLUMNS function, 193
tables, 6. *See also* relationships
big, working with, 79–80
Cartesian product between two, 205
creating hierarchy, 364
Default Field Set for, 441–442
dependencies between Tabular solution and structure of, 406
DMVs for memory usage, 587
evaluation context for single, 157–161
evaluation context in multiple, 164–183
facts and dimensions stored in, 385
filter context for multiple, 168–173
modifying, 177–182
hierarchies spanning multiple, 365–367
loading data from list of, 83–85
loading data into, 41–49
lookup, 165
names of, 83, 429
in PowerPivot, linked, 455
Process Data operation, 533–534
Process Full operation, 533
processing, 536–538
querying data sample from, 287–289
report with, 68
row context in single, 151–157
Row Identifier property, 399
selecting related, 86
selecting to import from data feed, 107
size in xVelocity, 340
viewing data in different, 44
xVelocity process operation steps, 349
table scan, cost of complete, 335

tables in Excel
converting to PowerPivot table, 455
Tabular database
C# code creating, 493
loading data from, 92–94
Tabular interface, understanding different, 488
Tabular models, 6–8
architecture in Analysis Services 2012, 329–330
as ETL tool, 498
building, 40–53
loading data into tables, 41–49
compatibility with PowerPivot, 12
connecting Power View to, 65–66
connecting to, 54–64
DAX vs. MDX queries execution, 331
dependencies between database table structure and, 406
deployment after prototyping in PowerPivot, 460–461
driver for connecting to, 187
ease of use, 12
functionality of Multidimensional model not present in, 15–17
hardware requirements, 13–14
language. *See* DAX (Data Analysis eXpressions)
mapped on Multidimensional data structure, 490
vs. Multidimensional model, 8
period table defined in, 316
private SSDT database, 21
processing performance compared to Multidimensional, 13
querying in Excel, 53–64
querying in Power View, 65–71
query performance characteristics, 13
ragged hierarchy, lack of support, 367
reference dimension relationship, 504–506
relationships between tables of, 224
relationship types, 504
security role types, 463
storage as Multidimensional database, 507
tool selection for creating, 458–459
Tabular presentation layer
drillthrough, 444–445
formatting, 436–438
hiding columns, 431–432
key performance indicators (KPI), 445–447
Language and Collation properties, 438
naming objects, 429
organizing measures, 432
perspectives, 438–440
sorting column data, 432–436
Tabular project
contents, 38–40
loading tables from AdventureWorksDW, 142
translation into Multidimensional, 488–491
temporary tables, memory for spooling, 568
testing
data security, 471–474
DAX query in Query Designer, 222
evaluation context in SSDT and SSMS, 156–157
measures with query, 216–217
relationship performances, 582
text value conversion to number, 138
text
concatenation in DAX, 124
testing conversion to numbers, 138
text files
as data source, 75
limitation, 117
loading data from, 98–99
text functions, 140
Thousand Separator, Data Format property values for, 437
ties in ranking, 282
tilde (~) operator in hierarchy, 374
time
browsing with hierarchies, 300
separating from date, 298–300
SQL Profiler for information on, 577–578
TIME function, 140
time intelligence, 291
DAX functions, 140, 307–328
measures based on, 319–321
Tabular modeling with Date table, 291–307
Tool dimension, in Multidimensional model, 315
TOPN function, 272–276
to table, 502
TotalMemoryLimit performance counter, 567
TotalMemoryLimit setting for SSAS, 563
TOTALMTD function, 310
TOTALQTD function, 310
Total Units Calculated measure, 327–328
Total Units Check measure, 327
Total Units measure, 322
formula for, 323
Total Units Movement measure, 326
TOTALYTD function, 309, 311
trace events, intercepting, 332
Transaction attribute, of Batch element, 541
Transaction ID, 344

transactions
for Process operations, 532
processing tables in separate, 529
updating balances using, 326–328
translations functionality, missing in Tabular, 15
TRIM function, 140
troubleshooting
BISM Connection type, 66
invalid calculations, 131
workspace database connection to SQL Server database, 43
True/False data type, Data Format property values for, 437
TRUNC function, 124, 139, 298
type conversion, 123
type-handling system, in DAX, 123

U

unary operators, 373–379
effective use in hiearchy levels, 378
implementing with DAX, 374–379
missing in Tabular, 16
Units Balance measure, 326
Update method, of AMO object, 493
UPPER function, 140
URL
for report, 108
for Table Import Wizard data feed, 110
usability, 429
Use First Row As Column Headers check box (Text Import Wizard), 99
USERELATIONSHIP function, 176, 233, 298, 419, 582
DAX expression to invoke, 229
for inactive chained relationships, 177
user hierarchies, 8
user interface, Tabular model as, 429
username, for SQL Server account, 41
USERNAME function, 479, 481–482
users, 463
SSAS impersonation, 77

V

VALUE function, 140
Value Not Supported error, 348
values
in DAX, 125
empty or missing, 133–134
VALUES function, 174, 182, 183, 387, 411
ALL expression vs., 245
FILTERS function and, 261–262
in CALCULATE, 282
VAR function, 285
variance of a variable, 285
VARX function, 286
VB.NET (Visual Basic .NET), 218
VertiPaq engine, 7, 118. *See also* xVelocity in-memory analytics engine (VertiPaq)
VertiPaqMemoryLimit setting for SSAS, 562
VertiPaqPagedKB counter, 568
VertiPaqPagingPolicy setting for SSAS, 562, 564
VertiPaq queries, 574
execution, 571–572
views
as decoupling layer, 304, 405–406
loading data from, 87–88
selecting in Table Import Wizard, 43
Visual Basic .NET (VB.NET), 218
Visual Studio, DAX editor for SQL Server extension in, 186
VISUALTOTAL function (MDX), 253
visual totals, ALLSELECTED function for, 253–257
VisualTotals function (MDX), 471
Visual Totals property, 470

W

Wang, Jeffrey, 225
warehouse stocking, 390
warnings, during install, 22
web browser, for SQL Server help, 25
web resources, data sources, 296
weighted aggregations, 393–395
WHERE condition, 574
in MDX query, 227
filter in, 232
Whole Number data type, 437
whole numbers, in xVelocity, 346
Windows account credentials, 77
Windows Authentication, 56, 482
Windows Azure Marketplace, 113
costs, 116
DateStream, 296
DateStream feed, 115
home page, 115
loading data from, 113
Windows integrated security, 463

Windows Task Manager
Processes tab, 560
Services tab, Analysis Services display, 559
Windows user account, as Analysis Services administrator, 466
Workbook Connections dialog box, 473
workbooks, merging into single SSAS Tabular solution, 459
workplace database server, 21
workspace database, 78
browsing, 54–55
connecting to SQL Server database, 43
recommended size, 79
server installation, 30
Workspace Retention model property, 36
Workspace Server, Service Account as user running SSAS, 77
workstation in development environment, 19–20
Writeback functionality, missing in Tabular, 15

X

XMLA Council, 9
XMLA (XML for Analysis), 218, 487, 510–512
Batch element to group commands, 540–542
command reference, 543
executing script from command line, 543
parameterizing command content, 551
processing automation with, 539–544
query window, 539
script for Tabular model deployment, 517
XML file, as source of model, 103
X-suffix aggregate functions, 136
xVelocity in-memory analytics engine (VertiPaq), 7, 326, 329, 330, 574. *See also* PowerPivot
cardinality of column, 342
CPU requirements, 515–516
database size estimation, 340
data types in, 346–348
disk and I/O requirements, 513
In-Memory mode and, 331–351
memory for, 338, 339–342, 514
pageable memory for, 568
partitions and, 520, 555
processing memory usage, 341–342
processing options, 348–351
querying memory usage, 342
server requirements, 513–516
snapshots and, 391
storage, 337–339
ADDCOLUMNS and, 193
internal structures, 338
SUMX function in, 575–576
and updating data, 14
use during development, 359

Y

year-over-year (YOY), 314
year-to-date aggregation, 307
year-to-date of the prior year, calculating, 312
YOY (year-over-year), 314
YOY YTD Sales%, formulas to define, 315
YTD order quantity column, creating for snapshot table, 391
YTD (year-to-date) calculation, 308–311

About the Authors

MARCO RUSSO is a Business Intelligence (BI) consultant and mentor. His main activities are related to data warehouse relational and multidimensional design, but he is also involved in the complete development life cycle of a BI solution. He has particular experience and competence in such sectors as financial services (including complex OLAP designs in the banking area), manufacturing, gambling, and commercial distribution.

Marco is also a book author and, apart from his BI-related publications, has written books on .NET programming. He is also a speaker at international conferences such as PASS Summit, SQLRally, and SQLBits.

He has achieved the unique SSAS Maestro certification and is also a Microsoft Certified Trainer with several Microsoft Certified Professional certifications.

ALBERTO FERRARI is a BI consultant. His main interests are in two areas: the methodological approach to the BI development life cycle and performance tuning of ETL and SQL code.

His activities are related to designing and implementing solutions based on Integration Services and Analysis Services for the financial, manufacturing, and statistical markets.

A certified SSAS Maestro, Alberto is also a book author and a speaker at international conferences such as PASS Summit, SQLRally, and SQLBits.

CHRIS WEBB is a consultant specializing in Analysis Services, MDX, PowerPivot, and DAX. He is a coauthor of *Expert Cube Development with SQL Server 2008 Analysis Services* and *MDX Solutions: With Microsoft SQL Server Analysis Services 2005 and Hyperion Essbase*.

Chris is a certified SSAS Maestro and is a regular speaker at PASS Summit and SQLBits conferences.