Microsoft®

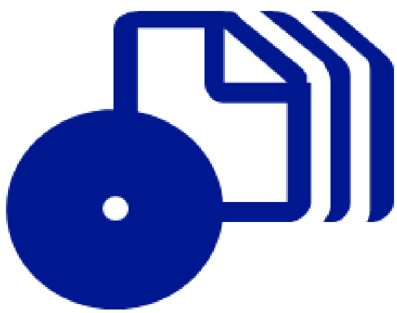
SQL Server® 2008 MDX

Bryan C. Smith
C. Ryan Clay
Hitachi Consulting

eBook + exercises

Step by Step

How to access your CD files



The print edition of this book includes a CD. To access the CD files, go to http://aka.ms/626188/files, and look for the Downloads tab.

Note: Use a desktop web browser, as files may not be accessible from all ereader devices.

Questions? Please contact: mspinput@microsoft.com

Microsoft Press

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2008940528

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 QWT 4 3 2 1 0 9

Distributed in Canada by H.B. Fenn and Company Ltd.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at www.microsoft.com/mspress. Send comments to mspinput@microsoft.com.

Acquisitions Editor: Ken Jones
Developmental Editor: Sally Stickney
Project Editor: Maureen Zimmerman
Editorial Production: S4Carlisle Publishing Services
Technical Reviewer: Todd Meister; Technical Review services provided by Content Master, a member of CM Group, Ltd.
Cover: Tom Draper Design

Body Part No. X14-72187

To my wife, Haruka, for her love, support,
and—above all else—patience

—Bryan C. Smith

To the three most important women in my life,
who have shaped who I am today:
my mother, Phyllis; my wife, Donna;
and my daughter, Emma Kay

—C. Ryan Clay

Contents at a Glance

Part I **MDX Fundamentals**

1 Welcome to MDX 3
2 Using the MDX Query Editor. 15
3 Understanding Tuples 37
4 Working with Sets. 61
5 Working with Expressions 91

Part II **MDX Functions**

6 Building Complex Sets 123
7 Performing Aggregation 157
8 Navigating Hierarchies. 181
9 Working with Time 211

Part III **MDX Applications**

10 Enhancing the Cube 239
11 Implementing Dynamic Security 273
12 Building Reports 311

List of Figures

FIGURE 1-1 A conceptual illustration of a business intelligence environment 4
FIGURE 1-2 The Reseller Sales fact 5
FIGURE 1-3 The Product and Date dimensions 6
FIGURE 1-4 The Reseller Sales fact and associated dimensions 6
FIGURE 1-5 The Reseller Sales and Sales Quota facts and associated dimensions 7
FIGURE 1-6 The members of the Category attribute-hierarchy 9
FIGURE 1-7 The members of the Product Categories user-hierarchy 10
FIGURE 2-1 SQL Server Management Studio and its various sections 17
FIGURE 2-2 The MDX Query Editor and its various sections 21
FIGURE 2-3 Key features of the Query Editor toolbar 21
FIGURE 3-1 A number line with a point at (3) 37
FIGURE 3-2 Two perpendicular number lines with a point at (3, 4) 38
FIGURE 3-3 Three perpendicular number lines with a point at (3, 4, 2) 38
FIGURE 3-4 The representation of the Category attribute-hierarchy as a cube space axis 40
FIGURE 3-5 The structure of the Chapter 3 Cube cube 41
FIGURE 3-6 The process for completing the tuple with a missing Measures member . . 48
FIGURE 3-7 The process for completing the tuple with missing Measures, Calendar Year, and Fiscal Year members 49
FIGURE 3-8 The relationship between the FY 2003 members and CY 2002 and CY 2003 members 50
FIGURE 3-9 The process for completing the tuple specifying the FY 2003 member associated with CY 2003 in the Calendar-To-Fiscal Year user-hierarchy 52
FIGURE 3-10 The process for completing the partial tuple specifying the member CY 2002 within the Calendar-To-Fiscal Year user-hierarchy 55
FIGURE 3-11 The process for completing the partial tuple specifying the member CY 2002 within the Calendar-To-Fiscal Year user-hierarchy and FY 2003 within the Fiscal Year attribute-hierarchy 56
FIGURE 3-12 The process for completing the partial tuple specifying overlapping references to the FY 2003 member 57
FIGURE 3-13 The process for completing the partial tuple specifying conflicting overlapping members from the Calendar-To-Fiscal Year user-hierarchy and Fiscal Year attribute-hierarchy 58
FIGURE 4-1 The first *SELECT* statement in this chapter and the cube space it defines . . . 68
FIGURE 4-2 The *SELECT* statement employing a *WHERE* clause 69
FIGURE 5-1 The process by which the partial tuple *([Date].[Calendar Year].[CY 2003], [Product].[Category].[Bikes & Accessories])* is completed 101
FIGURE 5-2 The process by which the partial tuple *([Product].[Category].[Bikes])* in the expression for the calculated member Bikes & Accessories is resolved 102

FIGURE 6-1 Venn diagram representations of the union, intersection, and exception operations. . . . 143
FIGURE 8-1 The immediate relatives of a given member in a hierarchy (shaded) . . . 181
FIGURE 8-2 The extended relatives of a given member within a hierarchy (shaded) . . . 189
FIGURE 8-3 The members accessed with the basic flags of the *Descendants* function given a specified member and level (shaded). . . . 191
FIGURE 9-1 A user-hierarchy based on the standard calendar . . . 212
FIGURE 10-1 The Solution Explorer window for the MDX Step-by-Step project . . . 242
FIGURE 10-2 The Cube Designer for the Step-by-Step cube. . . . 243
FIGURE 10-3 The Calculations tab's form view for the Step-by-Step cube . . . 243
FIGURE 10-4 The Calculations tab's toolbar . . . 244
FIGURE 12-1 The report assembled in this chapter. . . . 312
FIGURE 12-2 The BIDS interface following the creation of the MdxReports Report Server project . . . 314
FIGURE 12-3 The Report Designer for the MdxReport report item. . . . 315
FIGURE 12-4 The Design mode interface of the Query Designer. . . . 322
FIGURE 12-5 The Query Designer toolbar . . . 322

List of Tables

TABLE 2-1 Additional Hierarchies in the Step-by-Step Cube to Explore . . . 28
TABLE 3-1 Available cell properties . . . 46
TABLE 4-1 Formal, short, and alias names for the first five axes of the *SELECT* statement . . . 72
TABLE 5-1 Operators Supported by Analysis Services . . . 91
TABLE 5-2 VBA Functions Available Through Built-in Assemblies . . . 93
TABLE 5-3 Standard Numeric Formats . . . 109
TABLE 5-4 Standard Date Formats. . . . 109
TABLE 5-5 Member Property Functions . . . 110
TABLE 5-6 Intrinsic Member Properties Frequently Accessed Through the *Properties* Function . . . 110
TABLE 7-1 Records of a Fact Table Over Which an Average Is to Be Calculated . . . 165
TABLE 8-1 Navigation Functions Accessing Immediate Relatives. . . . 182
TABLE 8-2 Navigation Functions for Accessing Extended Relatives. . . . 190
TABLE 8-3 Flags Available for Use with the *Descendants* Function . . . 191
TABLE 8-4 Functions for Evaluating a Member's Position within a Hierarchy . . . 202
TABLE 8-5 Navigation Functions for Accessing Members within a Level . . . 204
TABLE 10-1 The Basic Scripting Statements . . . 246
TABLE 10-2 Some Standard Properties of Cube-Scoped Calculated Members . . . 256
TABLE 12-1 Filter Operators Supported by the Query Designer . . . 338

Table of Contents

Acknowledgements xiii
Introduction xv

Part I MDX Fundamentals

1 Welcome to MDX 3

The Business Intelligence Landscape 3
The Dimensional Model 5
Implementing the Dimensional Model 7
The Relational Data Warehouse 8
The Multidimensional Data Warehouse. 8
The MDX Language 11
Chapter 1 Quick Reference. 14

2 Using the MDX Query Editor 15

SQL Server Management Studio 15
The MDX Query Editor 19
Building a Simple MDX Query 23
Exploring the Step-by-Step Cube 25
Building a More Complex Query. 29
Chapter 2 Quick Reference. 35

3 Understanding Tuples 37

N-dimensional Space. 37
Cube Space 39
Accessing Data with Tuples 41
Understanding Cells. 43
Working with Partial Tuples 47
Building Tuples with User-Hierarchies 51

Understanding User-Hierarchy Translation 51
Avoiding Reference Conflicts 55
Member Reference Shortcuts 59
Chapter 3 Quick Reference 60

4 Working with Sets 61

Set Basics 61
Understanding the *SELECT* Statement 68
Building Sets with Functions 72
The *Members* Function 73
The *Crossjoin* Function 77
Limiting Sets 79
Working with Auto-Exists 79
The *Exists* Function 83
Chapter 4 Quick Reference 88

5 Working with Expressions 91

Expression Basics 91
Calculated Members 94
Building Dynamic Expressions 98
Resolving Contextual Conflicts 103
Avoiding Infinite Recursion 103
Controlling Solve Order 105
Building Complex Expressions 109
Working with the Current Member 109
Working with Sets in Expressions 115
Chapter 5 Quick Reference 117

Part II MDX Functions

6 Building Complex Sets 123

Assembling Ordered Sets 123
Retrieving the First or Last Tuples of a Set 131
Filtering Sets 137
Combining Sets 142
Performing Advanced Set Construction 147
Assembling Sets with the Generate Function 147
Assembling Sets with the *Extract* Function 151
Chapter 6 Quick Reference 153

7 Performing Aggregation 157
Performing Summation. 157
Calculating Averages 161
Calculating Averages with the *Avg* Function. 162
Calculating Averages with Expressions. 165
Identifying Minimum and Maximum Values 170
Counting Tuples in Sets. 172
Chapter 7 Quick Reference. 178

8 Navigating Hierarchies. 181
Accessing Immediate Relatives 181
Accessing Extended Relatives 189
Navigating within a Level 203
Chapter 8 Quick Reference. 208

9 Working with Time. 211
Understanding the Time Dimension. 211
Calculating an Accumulating Total 213
Calculating Rolling Averages 220
Performing Period-over-Period Analysis 222
Combining Time-Based Metrics 229
Chapter 9 Quick Reference. 233

Part III MDX Applications

10 Enhancing the Cube 239
Understanding the MDX Script 239
Constructing Calculated Members 247
Assembling a Basic Calculated Member 247
Setting Calculated Member Properties 256
Assembling Named Sets 266
Chapter 10 Quick Reference. 272

11 Implementing Dynamic Security 273
Understanding Dynamic Security 273
Implementing Attribute-Hierarchy Restrictions 285
Restricting Standard Attribute-Hierarchies 286
Restricting Parent-Child Hierarchies. 297
Implementing Cell-Level Restrictions 302
Chapter 11 Quick Reference. 309

12 Building Reports 311
Getting Started 311
Connecting to Analysis Services 316
Designing the Dataset 320
Adding Parameters to the Dataset 329
Presenting the Data in the Report 340
Chapter 12 Quick Reference 351

Index 353

Acknowledgements

The book you hold in your hands represents the thought, time, and energy of so many more people than those listed on the front cover. We owe all these folks our gratitude for their support without which this book would not be possible.

To identify just a few of these people, we'd like to thank the folks at Microsoft Press for the opportunity to work with them to address what we both perceive as an important need in the Microsoft Business Intelligence community. To Maureen Zimmerman and her team, thank you for your help in crafting this book and keeping things on track. To Ken Jones, thank you for championing us at all the right times.

Thanks also goes out to the Microsoft SQL Server Analysis Services product team for serving as a sounding board early on in the development process. Your encouragement helped us find our voice.

At Hitachi Consulting, we're grateful for the support of Lance Baldwin and Paul Turley who helped us get the ball rolling on this effort and provided continued support throughout the development process. To Eric Winton, Ryan Trout, and Drew Naukam, thank you for your patience while we focused our energy in this direction. To Hilary Feier and Scott Cameron, thank you for your support at critical junctures in this process. To Eric Noack, Reggie Nitcher, Jon Moore, Andrew Alexander, and Bryan Martin, thank you for your feedback and encouragement on the critical chapters of this book. Finally, a big thank you goes out to Reed Jacobson, whose MDX course provided the inspiration for this book and whose feedback was critical in shaping our content.

Last, but by no means least, we'd like to thank our families who have quietly sacrificed alongside us as this book was brought into being. We could not have done this without you and promise we will not do this again (for a little while, at least).

Bryan C. Smith

C. Ryan Clay

Introduction

Microsoft SQL Server Analysis Services is a powerful tool for Business Intelligence. Many organizations, both large and small, have adopted it to provide secure, high-performance access to complex analytics.

MDX is the language used by Analysis Services for data access. Proficiency with this language is essential to the realization of your Analysis Services databases' full potential. The innovative and elegant model underlying the MDX language makes it a very powerful but at the same time challenging tool for data analysis. In this book, we address this model head-on and then guide you through various functions and applications of the MDX language.

Who This Book Is For

This book has been written based on our own experiences as well as those of numerous clients and students. From these, we believe there are a few prerequisites to effectively learning the MDX language.

First, you must have basic familiarity with the concepts of dimensional modeling and data warehousing. If you do not have this knowledge, the overall purpose of Analysis Services and the MDX language will be lost.

Second, you must have basic familiarity with Analysis Services. You do not necessarily have to be a cube designer, but it does help to have worked with Analysis Services enough to be comfortable with its objects and terminology. If you are relatively new to Analysis Services, we recommend that you review *Microsoft SQL Server 2008 Analysis Services Step by Step* by Scott Cameron (Microsoft Press, 2009) before proceeding with this book.

Finally, you must be able put aside the traditional notions of data access you may have become familiar with. Some of the folks whom we've seen struggle the most with MDX have been some of the most talented users of more traditional languages such as SQL. MDX requires you to think about data very differently.

What This Book Is About

This book is about the core concepts and basic applications of MDX; it is not an exhaustive text. Instead, it is intended as a primer for those relatively new to the language. Through the discussions and exercises presented in each chapter you will be introduced to core concepts and applications. This will provide you with a solid foundation for continued learning in real-world scenarios.

This book is divided into three sections, each building on the one before it. We strongly encourage you to read these sections in sequence to ensure that you fully grasp later concepts and techniques.

Part I, "MDX Fundamentals," teaches you the fundamentals of the MDX language and the primary query development tool you use throughout this book.

Chapter 1, "Welcome to MDX," presents MDX as a means to deliver business value. This chapter is critical to establishing the concepts and vocabulary we employ throughout this book.

Chapter 2, "Using the MDX Query Editor," introduces you to the practical aspects of constructing and executing an MDX query using the MDX Query Editor.

Chapter 3, "Understanding Tuples," presents the concept of tuples. Understanding tuples is key to the successful use of the MDX language.

Chapter 4, "Working with Sets," expands the concept of tuples to include sets. With knowledge of tuples and sets, the MDX *SELECT* statement is explored.

Chapter 5, "Working with Expressions," introduces MDX expressions. Using calculated members, you explore expressions as a means for deriving values through Analysis Services.

Part II, "MDX Functions," builds upon the foundation established in Part I to explore the more frequently used MDX functions.

Chapter 6, "Building Complex Sets," guides you through the assembly of complex sets using a variety of MDX functions. Building just the right set is critical to retrieving the data you need from your cubes.

Chapter 7, "Performing Aggregation," explains the appropriate use of the MDX aggregation functions. Thoughtful application of these functions provides access to insightful metrics.

Chapter 8, "Navigating Hierarchies," explores the positioning of members in hierarchies and how this can be exploited using the navigation functions.

Chapter 9, "Working with Time," introduces you to the time-based MDX functions, through which critical business metrics can be derived.

Part III, "MDX Applications," uses concepts and functions explored in Parts I and II to implement three basic applications of the MDX language.

Chapter 10, "Enhancing the Cube," explores the enhancement of the MDX script through which calculated members and named sets can be incorporated into the definition of a cube.

Chapter 11, "Implementing Dynamic Security," presents a few approaches to implementing identity-driven, dynamic dimension data and cell-level security in your cube.

Chapter 12, "Building Reports," guides you through the process of developing MDX-driven reports in Reporting Services, Microsoft's enterprise reporting solution.

Conventions and Features in This Book

This book uses conventions designed to make information easily accessible. Before you start, read the following list, which explains conventions and helpful features within the book.

Conventions

- Each chapter contains multiple exercises demonstrating concepts and functionality. Each is presented as a series of numbered steps (1, 2, and so on) which you should follow in sequence to complete the exercise.
- Notes labeled "Note" provide additional information or alternative methods for completing a step successfully.
- Notes labeled "Important" alert you to information you need to be aware of before continuing.
- Most exercises demonstrate concepts of the MDX language through the use of an MDX *SELECT* statement. As steps progress, the *SELECT* statement introduced in previous steps may be altered. These changes appear in **bold**.

Other Features

- Sidebars are used throughout the book to provide important information related to an exercise or a topic. Sidebars might contain background information, supplemental content, or design tips or alternatives. Sidebars are also used to introduce topics supporting exercises.
- Each chapter ends with a Quick Reference section. The Quick Reference section contains quick reminders of how to perform the tasks you learned in the chapter.

System Requirements

You'll need a computer with the following hardware and software to complete the exercises in this book:

- Microsoft Windows Vista Home Premium edition, Windows Vista Business edition, Windows Vista Enterprise edition, or Windows Vista Ultimate edition
- Microsoft SQL Server 2008 Developer edition or Microsoft SQL Server 2008 Evaluation edition with Analysis Services, Database Engine Services (including Full-Text Search), Business Intelligence Development Studio, Client Tools Connectivity, and Management Tools installed
- CD-ROM or DVD-ROM drive to read the companion CD
- 150 MB free space for sample databases and companion content

In addition to these requirements, you should be able to log on directly to this computer with administrative rights. In addition to operation-level administrative rights, you should have full administrative rights in the SQL Server Database Engine and Analysis Services instances. Without these rights, you will not be able to install the sample databases or complete exercises in some chapters.

Samples

This book's companion CD contains database samples against which you will perform the chapters' exercises. MDX, SQL, and project code samples are also provided for you to verify your work. Instructions provided in the following sections will guide you through the installation of the companion CD's content to a local drive on your computer. This content is placed under the following path:

<Drive>:\Microsoft Press\MDX SBS

The MDX, SQL, and project code samples are provided under the Samples subfolder whereas database samples are provided under the Setup subfolder. Additional instructions are provided to make the sample databases operational.

Before attempting to complete the provided instructions, please verify your computer meets the hardware and software requirements and you have the required access described in the preceding section, "System Requirements."

Digital Content for Digital Book Readers: If you bought a digital-only edition of this book, you can enjoy select content from the print edition's companion CD.
Visit **http://www.microsoftpressstore.com/title/9780735626188** to get your downloadable content. This content is always up-to-date and available to all readers.

Installing the Samples

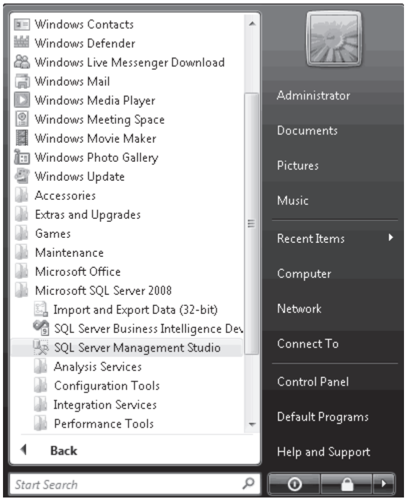
Install the companion CD content

1. Insert the book's companion CD in your computer's CD-ROM drive. A menu screen will appear. If AutoPlay is not enabled, run StartCD.exe at the root of the CD to display a start menu.
2. From the start menu, click Install Samples.
3. Follow the instructions that appear, selecting the drive to which the samples will be installed. These are installed to the following location on that drive:

   <Drive>:\Microsoft Press\MDX SBS

Attach the SQL Server database

1. On the Microsoft Windows task bar, click the Start button.
2. From the Start Menu, select All Programs and then Microsoft SQL Server 2008 to expose the SQL Server Management Studio shortcut.



3. Click the SQL Server Management Studio shortcut to launch the application.

   If this is the first time you have run Management Studio, you may see a dialog box indicating the application is being configured for its first use. This process may take a few minutes to complete before the application is then fully launched.

   Once fully launched, Management Studio presents the Connect To Server dialog box. If you are launching Management Studio for the first time on your machine, the dialog appears as shown below. If this is not the first time, selections and entries may differ.

4. In the Server Type field, verify Database Engine is selected.
5. In the Server Name field, type the name of your SQL Server instance. If you are connecting to a local default instance, you can simply enter **LOCALHOST** for the instance name.
6. Click Connect to establish a connection to SQL Server.
7. Once connected, use the File menu to select Open and then File, launching the Open File dialog box.
8. Using the Open File dialog box, navigate to the following folder installed in previous steps:

   <Drive>:\Microsoft Press\MDX SBS\Setup\SQL Server
9. Select the attach_db.sql file and click OK to open it.
10. If needed, modify the drive letter assigned to the sample database's .mdf file in the script. By default, the script assumes this file is on the C: drive in the following location:

    C:\Microsoft Press\MDX SBS\Setup\SQL Server\MdxStepByStep.mdf
11. With the drive letter modified as needed, select Execute from the Query menu to execute the script.
12. Review the messages provided to confirm the database was successfully attached to SQL Server.
13. From the File menu, select Close to close Management Studio. Select either Yes or No if prompted to save changes to the attach_db.sql file.

Restore the Analysis Services database

1. Launch SQL Server Management Studio as you did in the previous steps.
2. In the Connect To Server dialog box, select Analysis Services for the Server Type field and enter the name of your Analysis Services instance in the Server Name field. If you are connecting to a local default instance, you can simply enter **LOCALHOST** for the instance name.
3. Click Connect to establish a connection to Analysis Services.
4. Once connected, use the File menu to select Open and then File, launching the Open File dialog box.
5. Using the Open File dialog box, navigate to the following folder installed in previous steps:

   <Drive>:\Microsoft Press\MDX SBS\Setup\Analysis Services
6. Select the restore_db.xmla file and click OK to open it.
7. If needed, modify the drive letter assigned to the sample database's .abf file in the script. By default, the script assumes this file is on the C: drive in the following location:

   C:\Microsoft Press\MDX SBS\Setup\Analysis Services\MdxStepByStep.abf

8. With the drive letter modified as needed, select Execute from the Query menu to execute the script.
9. Review the messages provided to confirm the database was successfully attached to Analysis Services.
10. From the File menu, select Close to close Management Studio. Select either Yes or No if prompted to save changes to the restore_db.xmla.

Uninstalling the Samples

Drop the Analysis Services database

1. Launch SQL Server Management Studio and connect to Analysis Services as described in the steps for restoring the Analysis Services database.
2. Once connected, select Open and then File from the File menu.
3. Using the Open File dialog box, navigate to the following folder installed in previous steps:

   <Drive>:\Microsoft Press\MDX SBS\Setup\Analysis Services
4. Select the drop_db.xmla file and click OK to open it.
5. Select Execute from the Query menu to execute the script.
6. Review the messages provided to confirm the database was successfully dropped from Analysis Services.
7. From the File menu, select Close to close Management Studio.

Detach the SQL Server database

1. Launch SQL Server Management Studio and connect to SQL Server as described in the steps for attaching the SQL Server database.
2. Once connected, select Open and then File from the File menu.
3. Using the Open File dialog box, navigate to the following folder installed in previous steps:

   <Drive>:\Microsoft Press\MDX SBS\Setup\SQL Server
4. Select the detach_db.sql file and click OK to open it.
5. Select Execute from the Query menu to execute the script.
6. Review the messages provided to confirm the database was successfully detached from SQL Server.
7. From the File menu, select Close to close Management Studio.

Remove the companion CD content

1. From your computer's Control Panel, open Add or Remove Programs.
2. From the list of Currently Installed Programs, select Microsoft SQL Server 2008 MDX Step by Step.
3. Click Remove.

> **Important** If you have not detached or dropped the sample SQL Server database, you may be prevented from completing these steps.

4. Follow the instructions that appear to remove the samples.

Find Additional Content Online

As new or updated material becomes available that complements your book, it will be posted online on the Microsoft Press Online Developer Tools Web site. The type of material you might find includes updates to book content, articles, links to companion content, errata, sample chapters, and more. This Web site is available at *http://www.microsoft.com/learning/books/online/developer*, and is updated periodically.

Support for This Book

Every effort has been made to ensure the accuracy of this book and the contents of the companion CD. As corrections or changes are collected, they will be added to a Microsoft Knowledge Base article.

Microsoft Press provides support for books and companion CDs at the following Web site:

http://www.microsoft.com/learning/support/books/

Questions and Comments

If you have comments, questions, or ideas regarding the book or the companion CD, or questions that are not answered by visiting the preceding site, please send them to Microsoft Press via e-mail to:

mspinput@microsoft.com

Or via postal mail to:

Microsoft Press
Attn: *Microsoft SQL Server 2008 MDX Step by Step* Editor
One Microsoft Way
Redmond, WA 98052-6399

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

Chapter 3
Understanding Tuples

After completing this chapter, you will be able to:

- Explain the concept of cube space
- Retrieve data from a cube using tuples
- Reference hierarchy members using a variety of syntax

For the purpose of data access, Analysis Services presents cubes as n-dimensional spaces referred to as cube spaces. Within a *cube space*, data are made accessible through *cells*, each uniquely identified by a *tuple*.

In this chapter, you learn how to assemble tuples to access individual cells. This is foundational to your success with MDX.

N-dimensional Space

To understand the concept of cube space, picture a simple number line. As you may remember from your school days, a number line is a line marked at regular intervals by integer (whole-number) values. Figure 3-1 provides an illustration of such a line.

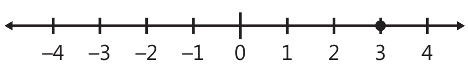


FIGURE 3-1 A number line with a point at (3)

In this illustration, a point resides along the line at the position indicated by the number 3. This number, 3, is the point's *coordinate*. When you wrap the coordinate in parentheses like so

(3)

you have a simple system for expressing the point's position along the line.

Now consider the introduction of another number line perpendicular to the one above. These two lines define a two-dimensional space, as illustrated in Figure 3-2.

Traditionally, the horizontal line in this two-dimensional space is referred to as the x-axis and the vertical line is referred to as the y-axis. Points within this space are identified by their position relative to these two axes. (*Axes* is the plural of axis.)

To express the position of a point, the x-coordinate and y-coordinate of the point is presented in a comma-delimited list. In this list, the x-coordinate precedes the y-coordinate, and the entire list is wrapped in parentheses. This double coordinate system is generically described using the form (x, y).

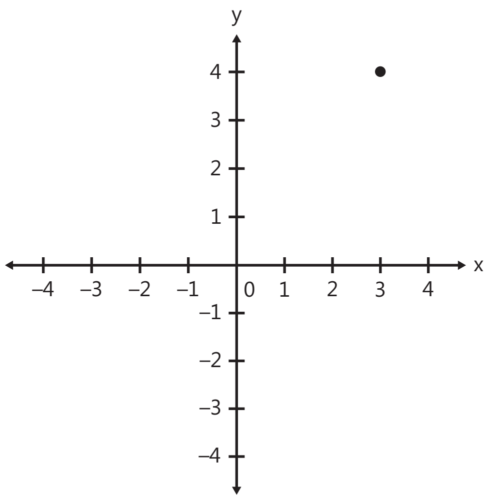


FIGURE 3-2 Two perpendicular number lines with a point at (3, 4)

To illustrate this, consider the point in Figure 3-2. It resides at the intersection of the value 3 along the x-axis and 4 along the y-axis. It is therefore identified using the double coordinate (3, 4).

Taking this one step further, consider the addition of a third line perpendicular to both the x and y axes. Keeping with tradition, the newly introduced third axis is referred to as the z-axis. The space formed by these three axes is illustrated in Figure 3-3. Together with the x and y axes, the z-axis forms a three-dimensional space. Points within this space are represented using a triple-coordinate system, (x, y, z). While challenging to see on paper, a point is presented in Figure 3-3 at the position identified by the triple coordinate (3, 4, 2).

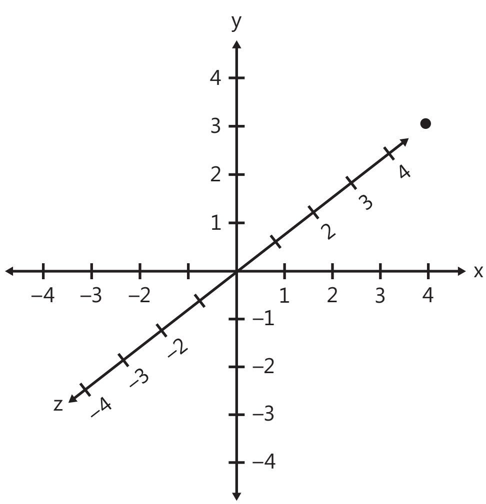


FIGURE 3-3 Three perpendicular number lines with a point at (3, 4, 2)

Now add a fourth axis. The four-dimensional space created can no longer be easily visualized. Still, points within this space can be located using a quadruple-coordinate system.

To describe the form of the quadruple-coordinate system, it's helpful to re-label the axes with the letter *a* and a numerical subscript. Using this approach, the x-axis becomes axis a_1, the y-axis becomes axis a_2, the z-axis becomes axis a_3, and the newly introduced fourth axis becomes axis a_4. Points within this space are then located using a quadruple-coordinate system of the form (a_1, a_2, a_3, a_4).

Adding a fifth axis makes the space even more complex, but points within this space are easily addressed using a quintuple-coordinate system of the form $(a_1, a_2, a_3, a_4, a_5)$. A sixth axis leads to a sextuple-coordinate system $(a_1, a_2, a_3, a_4, a_5, a_6)$; a seventh axis leads to a septuple-coordinate system $(a_1, a_2, a_3, a_4, a_5, a_6, a_7)$; and an eighth axis leads to an octuple-coordinate system $(a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8)$.

You could go on like this forever, and while imagining spaces such as these is a bit mind-blowing, locating a point within any of them is a simple matter of employing an appropriately sized coordinate system.

Generically, these spaces are referred to as *n-dimensional spaces*. These spaces have *n* number of axes, and points within them are located using coordinate systems of the form $(a_1, a_2, \ldots, a_n)$. These coordinate systems are generically referred to as *tuples*.

> **Note** The question of how to properly pronounce the word *tuple* always seems to come up. Some folks pronounce it with a *u* like the one in *cup*. Others pronounce it like with a *u* like the one in *dude*. We aren't really sure which way is right and use both forms ourselves.

Cube Space

In Analysis Services, a cube is presented as an n-dimensional space referred to as a cube space. Each attribute-hierarchy within the dimensions of the cube forms an axis. Along each axis, each member of the associated attribute-hierarchy, including the (All) member, occupies a position. This translation of an attribute-hierarchy to a cube space axis is illustrated in Figure 3-4 for the Product dimension's Category attribute-hierarchy, first described in Chapter 1, "Welcome to MDX."

Measures are also assigned an axis. Although handled differently during cube design, for the purposes of defining a cube space, a cube's measures are simply members of an attribute-hierarchy called Measures, which belongs to the Measures dimension. One thing that differentiates the Measures attribute-hierarchy from other attribute-hierarchies is that it does not (and cannot) have an (All) member.

With each traditional attribute-hierarchy and the measures of a cube translated into axes, the cube space is defined. Points within the cube space can then be referenced using a tuple. Unlike tuples in the n-dimensional spaces formed by number lines, tuples in cube spaces use member references for coordinate values.

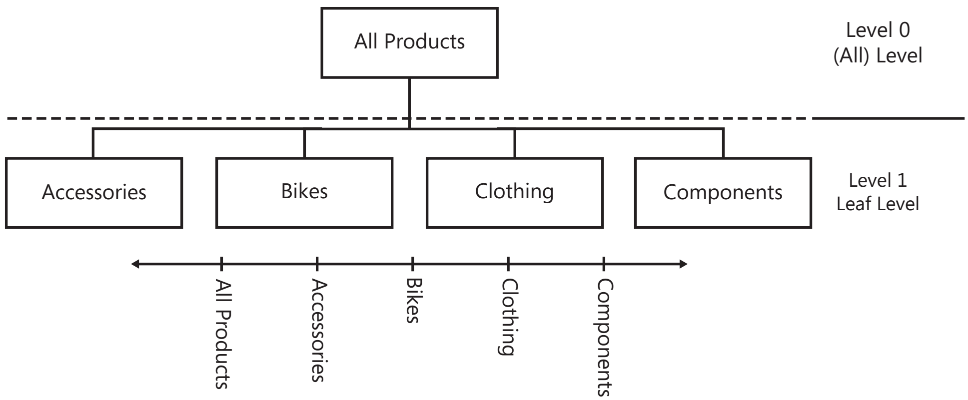


FIGURE 3-4 The representation of the Category attribute-hierarchy as a cube space axis

Basic Member References

You can reference a member within an attribute-hierarchy in a number of ways. The basic member reference identifies the member along with its associated attribute-hierarchy and dimension using the following form:

```
[Dimension].[Hierarchy].[Member].
```

Each of the dimension, attribute-hierarchy, and member object identifiers within the member reference are encapsulated in square brackets. These are separated from each other by periods.

The square brackets around a particular object identifier are optional as long as the object identifier:

1. Is not one of 200+ reserved words identified in SQL Server Books Online
2. Does not start with a character other than a letter or underscore
3. Does not otherwise contain any characters other than letters, numbers, or underscores

Instead of keeping up with all this, you might find it easier to just consistently wrap each identifier in square brackets. This is a standard used throughout this book.

Object names are used as the identifiers for dimensions and attribute-hierarchies. Members are a bit more complex in that they can be identified by either name or key.

A member's name is its user-friendly label. This is what is usually presented in result sets and browsers such as the MDX Query Editor. The following example demonstrates a name-based reference to the member Bikes of the Product dimension's Category attribute-hierarchy:

```
[Product].[Category].[Bikes]
```

Member names suffer one key drawback: They are not guaranteed to be unique within an attribute-hierarchy. This is problematic if more than one member within a hierarchy shares the same name (which is quite common in some dimensional models). Using key-based references resolves this problem.

A member's key is its unique identifier within its associated attribute-hierarchy. Because of its guaranteed uniqueness, a key is the most precise means of identifying a member within an attribute-hierarchy. When identifying a member by key, the identifier is preceded by the ampersand character (&). The previous Bikes reference is demonstrated using its key-based reference:

```
[Product].[Category].&[1]
```

This example illustrates a common issue with key-based references. If you are not aware that the member named Bikes employs a key-value of 1, the key-based reference may be difficult to interpret. This leaves you in the position of using name-based references that may be ambiguous or key-based references that may be difficult to interpret. In this book, we make use of named-based references for interpretability unless a particular concept or ambiguity dictates we use keys. The right choice in your applications depends on the structure of your data.

Accessing Data with Tuples

The MDX Step-by-Step sample database accompanying this book contains a highly simplified cube named Chapter 3 Cube. The cube consists of two dimensions—Product and Date—and a single measure, Reseller Sales Amount. Figure 3-5 presents the structure of this cube.

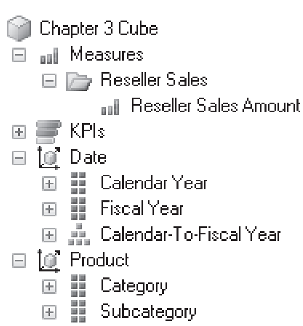


FIGURE 3-5 The structure of the Chapter 3 Cube cube

Within the Product dimension are two attribute-hierarchies, Subcategory and Category. The Date dimension also contains two attribute-hierarchies, Fiscal Year and Calendar Year, which together form the levels of the user-hierarchy Calendar-To-Fiscal Year.

> **Note** The Calendar-To-Fiscal Year user-hierarchy is provided in this cube for no other purpose than to illustrate a few critical concepts while sidestepping a few issues addressed later on. The Calendar-To-Fiscal Year user-hierarchy is not found in the Step-by-Step cube, and such a user-hierarchy combining fiscal year and calendar year attributes is rarely found in the real world. Please consider this hierarchy nothing more than an educational construct.

With four traditional attribute-hierarchies plus the Measures attribute-hierarchy discussed earlier in this chapter, the cube space formed by this cube contains a total of five axes. Points within this cube space are therefore located using a five-part tuple.

For example, the point located at the intersection of the Category member Bikes, the Subcategory member Mountain Bikes, the Calendar Year and Fiscal Year members All Periods, and the Measures member Reseller Sales Amount is identified with the following five-part tuple:

```
(
    [Date].[Calendar Year].[All Periods],
    [Date].[Fiscal Year].[All Periods],
    [Product].[Category].[Bikes],
    [Product].[Subcategory].[Mountain Bikes],
    [Measures].[Measures].[Reseller Sales Amount]
)
```

The use of this tuple to retrieve data is demonstrated in the following exercise.

Use a tuple to access a point in a cube space

1. Open the MDX Query Editor to the MDX Step-by-Step database. If you need assistance with this task, refer to Chapter 2, "Using the MDX Query Editor."
2. In the code pane, enter the following query:

```
SELECT
FROM [Chapter 3 Cube]
WHERE (
    [Date].[Calendar Year].[All Periods],
    [Date].[Fiscal Year].[All Periods],
    [Product].[Category].[Bikes],
    [Product].[Subcategory].[Mountain Bikes],
    [Measures].[Measures].[Reseller Sales Amount]
    )
```

> **Note** The line breaks and indentions used with this tuple are purely for readability.

3. Execute the query.

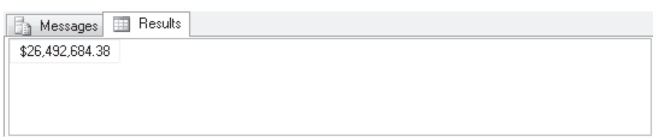
The tuple is employed in the *SELECT* statement to retrieve data from a single point within the cube space formed by the Chapter 3 Cube. Like tuples associated with number lines, this tuple used here consists of a parentheses-enclosed, comma-delimited list of coordinate values. Each of these values consists of a basic member reference identifying a member (by name) and its associated attribute-hierarchy and dimension.

Since an attribute-hierarchy represents an axis in the cube space and a member reference identifies the attribute-hierarchy, the member reference identifies the axis with which it is associated. In other words, member references are self-describing. Therefore, you don't need to rely on the position of a member reference (coordinate value) in the tuple to determine which axis it is associated with. This allows member references to be placed in any order within a tuple without impacting the point identified.

4. Move the *[Product].[Subcategory].[Mountain Bikes]* member reference to the top of the tuple:

```
SELECT
FROM [Chapter 3 Cube]
WHERE (
    [Product].[Subcategory].[Mountain Bikes],
    [Date].[Calendar Year].[All Periods],
    [Date].[Fiscal Year].[All Periods],
    [Product].[Category].[Bikes],
    [Measures].[Measures].[Reseller Sales Amount]
    )
```

5. Execute the query and verify the same value as before is returned.



Try moving around other member references within the tuple. Notice that so long as the tuple is properly formed, the same point within the cube space is identified.

Understanding Cells

In the previous exercise, you used a tuple to locate a point within a cube space. On the surface, it appeared that a simple value is recorded at this point, which is what is returned by the *SELECT* statement. The reality is a bit more complex.

Points within cube spaces are occupied by cells. Cells are objects and as such have a number of properties. When cells are accessed, various properties are returned. The default properties returned are *VALUE* and *FORMATTED_VALUE*.

The *VALUE* property contains an aggregated measure value. That value is based on the measure aggregated against all the other attribute-hierarchy members associated with the cell. For example, the *VALUE* property of the cell associated with the previously employed tuple, repeated here for clarity, contains the aggregated value for the Reseller Sales Amount measure limited to the Calendar Year and Fiscal Year attribute-hierarchies' All Periods members, the Category attribute-hierarchy's Bikes member, and the Subcategory attribute-hierarchy's Mountain Bikes member:

```
(
    [Date].[Calendar Year].[All Periods],
    [Date].[Fiscal Year].[All Periods],
    [Product].[Category].[Bikes],
    [Product].[Subcategory].[Mountain Bikes],
    [Measures].[Measures].[Reseller Sales Amount]
)
```

The *FORMATTED_VALUE* property contains the string representation of the *VALUE* property, formatted per instructions associated with the cell at design time. The *FORMATED_VALUE* is what is displayed in the results pane of the MDX Query Editor. A bit more information on assigning formats is provided in Chapter 5, "Working with Expressions."

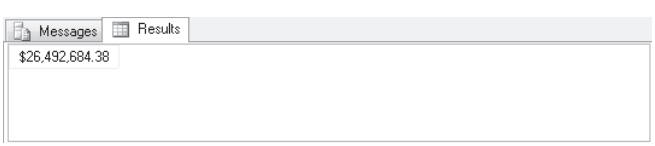
A number of other properties can be returned with a cell. Within a *SELECT* statement, these are accessed using the *CELL PROPERTIES* keyword as demonstrated in the following exercise.

Access cell properties

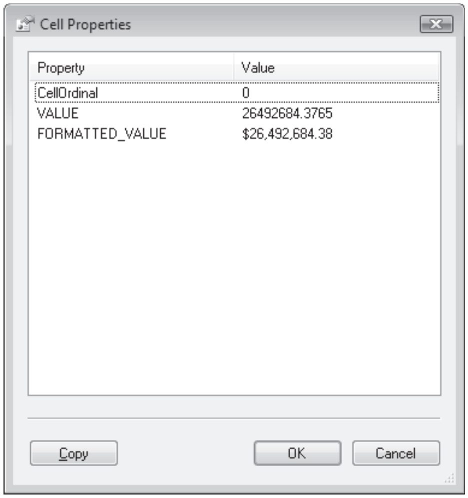
1. If you have not already done so, open the MDX Query Editor to the MDX Step-by-Step database.
2. In the code pane, re-enter the last query from the previous exercise:

```
SELECT
FROM [Chapter 3 Cube]
WHERE (
    [Product].[Subcategory].[Mountain Bikes],
    [Date].[Calendar Year].[All Periods],
    [Date].[Fiscal Year].[All Periods],
    [Product].[Category].[Bikes],
    [Measures].[Measures].[Reseller Sales Amount]
    )
```

3. Execute the query to retrieve the results.



4. Double-click the cell returned in the Results pane to open the Cell Properties dialog box.



The default properties *VALUE* and *FORMATTED_VALUE* are returned with the cell. The *CELL_ORDINAL* property, displayed as CellOrdinal, is also returned to indicate the position of the returned cell in the query's cell set. Cell sets are discussed in Chapter 4, "Working with Sets."

You can retrieve additional properties by including the *CELL PROPERTIES* keyword in your query. If you use the *CELL PROPERTIES* keyword, the *VALUE* and *FORMATTED_VALUE* properties are not returned unless explicitly requested. (The *CELL_ORDINAL* property is always returned as it is a property of the retrieved data.)

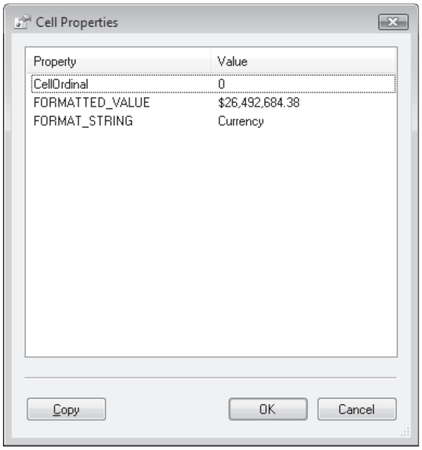
5. Click the OK button in the Cell Properties dialog box to close it.

6. Modify the query to request the *FORMATTED_VALUE* and *FORMAT_STRING* cell properties, purposely omitting the *VALUE* and *CELL_ORDINAL* properties:

```
SELECT
FROM [Chapter 3 Cube]
WHERE (
    [Product].[Subcategory].[Mountain Bikes],
    [Date].[Calendar Year].[All Periods],
    [Date].[Fiscal Year].[All Periods],
    [Product].[Category].[Bikes],
    [Measures].[Measures].[Reseller Sales Amount]
    )
CELL PROPERTIES FORMATTED_VALUE, FORMAT_STRING
```

7. Execute the query.

8. Double-click the returned cell to open the Cell Properties dialog box.



   Notice that *VALUE* is omitted from the list of cell properties, but the *CELL_ORDINAL* property is returned with the cell.

9. Review the property values and then click OK to close the dialog box.

The complete list of available cell properties and their descriptions is provided in Table 3-1. Additional information on each property is available through SQL Server Books Online.

TABLE 3-1 Available cell properties

Cell Property	Description
ACTION_TYPE	A bitmask indicating the type of action(s) associated with the cell.
BACK_COLOR	A bitmask indicating the background color to use when displaying the *VALUE* or *FORMATTED_VALUE* property of the cell.
CELL_ORDINAL	The ordinal number of the cell in the cell set.
FONT_FLAGS	A bitmask indicating whether the cell's font should be presented using italic, bold, underline, or strikeout detailing.
FONT_NAME	The name of the font to use when displaying the *VALUE* or *FORMATTED_VALUE* property of the cell.
FONT_SIZE	The font size to use when displaying the *VALUE* or *FORMATTED_VALUE* property of the cell.
FORE_COLOR	A bitmask indicating the foreground color to use when displaying the *VALUE* or *FORMATTED_VALUE* property of the cell.
FORMAT	This is the same as the *FORMAT_STRING* property.
FORMAT_STRING	The format string used to create the value of *FORMATTED_VALUE* property of the cell.
FORMATTED_VALUE	The character string representation of the *VALUE* property formatted per the *FORMAT_STRING* value.
LANGUAGE	The locale against which the *FORMAT_STRING* will be applied.
UPDATEABLE	A value indicating whether the cell can be updated.
VALUE	The unformatted value of the cell.

Working with Partial Tuples

The cube used in this chapter has a very simple structure. With only five attribute-hierarchies (including Measures), points within this cube are identifiable using a five-part tuple. Imagine a more typical cube with tens or even hundreds of attributes. Having to specify a member reference for each attribute-hierarchy within the cube to complete a tuple would simply be overwhelming.

Thankfully, Analysis Services allows you to submit partial tuples. Within a partial tuple one or more member references are omitted. Because a complete tuple is required to locate a point in the cube space, Analysis Services takes responsibility for filling in the missing references. This is done by applying the following rules for each missing attribute-hierarchy member reference:

1. If the member reference is omitted, use the attribute's default member.
2. If the member reference is omitted and no default member is specified, use the attribute's (All) member.
3. If the member reference is omitted, no default member is specified, and the (All) member does not exist, use the attribute's first member.

In the following exercise, you put these rules to work.

Access cells in a cube using partial tuples

1. If you have not already done so, open the MDX Query Editor to the MDX Step-by-Step database.
2. In the code pane, enter the following query specifying a complete tuple:

```
SELECT
FROM [Chapter 3 Cube]
WHERE (
    [Date].[Calendar Year].[All Periods],
    [Date].[Fiscal Year].[All Periods],
    [Product].[Category].[Bikes],
    [Product].[Subcategory].[Mountain Bikes],
    [Measures].[Measures].[Reseller Sales Amount]
    )
```

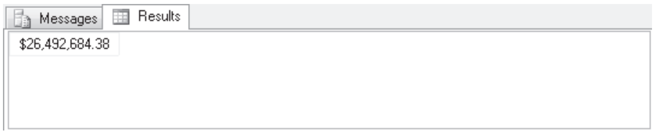
3. Execute the query and note the result.

4. Now, specify a partial tuple by removing the Measures member reference:

```
SELECT
FROM [Chapter 3 Cube]
WHERE (
    [Date].[Calendar Year].[All Periods],
    [Date].[Fiscal Year].[All Periods],
    [Product].[Category].[Bikes],
    [Product].[Subcategory].[Mountain Bikes]
    )
```

Note Be certain to remove the comma following the Mountain Bikes member reference.

5. Execute the query and compare the result to that of the previous query.



With the Measures member removed, a partial tuple is submitted to Analysis Services. Analysis Services supplies the missing Measures reference by first checking for a default member. The default member of the Measures attribute-hierarchy is Reseller Sales Amount. That member is applied and the tuple is complete. The process by which the tuple is completed is illustrated in Figure 3-6. Because the completed tuple is the same tuple specified in the first query of this exercise, the same cell is accessed.

Position	Partial Tuple	Rule 1: Default Member	Rule 2: (All) Member	Rule 3: First Member	Completed Tuple
Date. Calendar Year	All Periods	→	→	→	All Periods
Date. Fiscal Year	All Periods	→	→	→	All Periods
Product. Category	Bikes	→	→	→	Bikes
Product. Subcategory	Mountain Bikes	→	→	→	Mountain Bikes
Measures. Measures	*(omitted)*	Reseller Sales Amount	→	→	Reseller Sales Amount

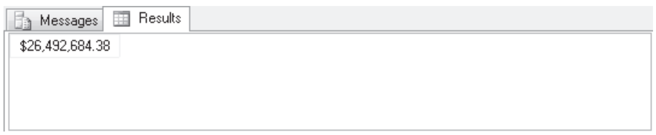
FIGURE 3-6 The process for completing the tuple with a missing Measures member

Note The default member of the Measures attribute-hierarchy is defined at design time when a default measure is assigned to the cube. In this cube, a default measure of Reseller Sales Amount has been assigned. Had this not been explicitly assigned, the third rule would have completed the tuple with Reseller Sales Amount, the first (and only) measure in the cube.

6. Alter the query by removing the two member references associated with the Date dimension:

```
SELECT
FROM [Chapter 3 Cube]
WHERE (
    [Product].[Category].[Bikes],
    [Product].[Subcategory].[Mountain Bikes]
    )
```

7. Execute the query and compare the result to that of the previous query.



With this query, Analysis Services supplies the Measures member reference by applying the first rule. For the Date dimension's Calendar Year and Fiscal Year attribute-hierarchies, a default member is not defined so the first rule does not address these omitted references. However, an (All) member, All Periods, is defined for these attribute-hierarchies, so the second rule fills in the blanks. The process by which this partial tuple is completed is illustrated in Figure 3-7. As before, the completed tuple is the same as the tuple in the first query of this exercise so that the same cell as before is accessed.

Position	Partial Tuple	Rule 1: Default Member	Rule 2: (All) Member	Rule 3: First Member	Completed Tuple
Date. Calendar Year	*(omitted)*	*(not available)*	All Periods →		All Periods
Date. Fiscal Year	*(omitted)*	*(not available)*	All Periods →		All Periods
Product. Category	Bikes →				Bikes
Product. Subcategory	Mountain Bikes →				Mountain Bikes
Measures. Measures	*(omitted)*	Reseller Sales Amount →			Reseller Sales Amount

FIGURE 3-7 The process for completing the tuple with missing Measures, Calendar Year, and Fiscal Year members

Now that you understand partial tuples, it should be clear what the basic query introduced in Chapter 2 returns. This query, *SELECT FROM [Step-by-Step]*, returns the cell associated with the partial tuple within which no member references are supplied. Analysis Services completes each member reference using the three preceding rules and accesses the identified cell.

More Member References

Members in user-hierarchies may also be referenced using the form, *[Dimension].[Hierarchy].[Member],* introduced earlier in this chapter. For example, the calendar year 2003 member of the Calendar-To-Fiscal Year user-hierarchy can be identified as follows:

```
[Date].[Calendar-To-Fiscal Year].[CY 2003]
```

However, because user-hierarchies are assembled from multiple attribute-hierarchies, the member identifier has greater opportunity to be non-unique. This is true not only when member names are employed but also with member keys. To illustrate this, consider the following member reference. Does it reference calendar year 2003 or fiscal year 2003?

```
[Date].[Calendar-To-Fiscal Year].&[2003]
```

This reference is ambiguous. Both the calendar year 2003 and fiscal year 2003 members use the number 2003 as their key. Referencing the member using the form *[Dimension].[Hierarchy].[Level].[Member]* resolves this ambiguity:

```
[Date].[Calendar-To-Fiscal Year].[Calendar Year].&[2003]
```

This new form works with both member keys and member names and is ideal when the member identifier is unique within a specified level but not necessarily unique across the levels of the hierarchy.

Unfortunately, in some situations this new form of member reference is still ambiguous. Consider the Fiscal Year members in Figure 3-8. In particular, pay attention to the two FY 2003 members.

FIGURE 3-8 The relationship between the FY 2003 members and CY 2002 and CY 2003 members

There is one FY 2003 member in the Fiscal Year attribute-hierarchy representing the period July 1, 2002, to June 30, 2003. Since the fiscal year 2003 straddles calendar years 2002 and 2003, two FY 2003 members (one under CY 2002 and the other under CY 2003) are found in the user-hierarchy. Within the user-hierarchy, the FY 2003 member is presented as two distinct members.

In this situation, the only way to differentiate between the two is to identify the Fiscal Year member in relation to its Calendar Year parent. Here are member references identifying these two distinct user-hierarchy members:

```
[Date].[Calendar-To-Fiscal Year].[Calendar Year].[CY 2002].[FY 2003]
[Date].[Calendar-To-Fiscal Year].[Calendar Year].[CY 2003].[FY 2003]
```

Building Tuples with User-Hierarchies

The exercises presented thus far have built tuples exclusively using references to members in attribute-hierarchies. You can also use user-hierarchies to assemble tuples. When a user-hierarchy member reference is employed, Analysis Services translates that reference into one or more attribute-hierarchy member references to assemble a resolvable tuple.

Understanding User-Hierarchy Translation

To translate a user-hierarchy member reference into one or more attribute-hierarchy references, Analysis Services first locates the specified member within the user-hierarchy. With this member located, that member and each member in the levels above it forming the member's lineage in the user-hierarchy is then known. As each level in a user-hierarchy is derived from an attribute-hierarchy, an attribute-hierarchy reference for the specified member and each member in its lineage is then generated. The lone exception to this is the user-hierarchy's (All) member, which does not map to any member in an attribute-hierarchy and is therefore simply ignored in the translation process.

The following exercise demonstrates the process of translating user-hierarchy member references to attribute-hierarchy references.

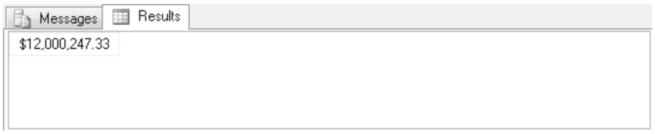
Access cells with tuples containing user-hierarchies

1. If you have not already done so, open the MDX Query Editor to the MDX Step-by-Step database.
2. In the code pane, enter the following query:

```
SELECT
FROM [Chapter 3 Cube]
WHERE (
    [Date].[Calendar-To-Fiscal Year].[Calendar Year].[CY 2003].[FY 2003]
    )
```

> **Note** When a tuple is specified using a single member reference, the tuple's parentheses can be omitted. Parentheses are applied to the tuple in the preceding query for the purpose of consistency.

3. Execute the query and note the result.



To resolve this tuple, Analysis Services first locates the FY 2003 member in the Fiscal Year level associated with the CY 2003 member of the Calendar Year level of the Calendar-To-Fiscal Year user-hierarchy. Analysis Services then determines the lineage of this member, which you already know given the explicit structure of the member reference. Each member in the lineage is then translated into an attribute-hierarchy reference and the tuple is completed as illustrated in Figure 3-9.

Position	User-Hierarchy Translation	Partial Tuple	Rule 1: Default Member	Rule 2: (All) Member	Rule 3: First Member	Completed Tuple
Date. Calendar-To-Fiscal Year	Calendar Year. CY 2003. FY 2003					
Date. Calendar Year	CY 2003 →				→	CY 2003
Date. Fiscal Year	FY 2003 →				→	FY 2003
Product. Category		*(omitted)*	*(not available)*	All Products	→	All Products
Product. Subcategory		*(omitted)*	*(not available)*	All Products	→	All Products
Measures. Measures		*(omitted)*	Reseller Sales Amount	→	→	Reseller Sales Amount

FIGURE 3-9 The process for completing the tuple specifying the FY 2003 member associated with CY 2003 in the Calendar-To-Fiscal Year user-hierarchy

To verify this, you can submit the translated (partial) tuple to see that the same cell is returned.

4. Modify the query to reflect the translated (partial) tuple:

```
SELECT
FROM [Chapter 3 Cube]
WHERE (
   [Date].[Calendar Year].[CY 2003],
   [Date].[Fiscal Year].[FY 2003]
   )
```

5. Execute the query and compare the result to that in step 3.

When the lineage for FY 2003 is not specified in the user-hierarchy member reference, the reference becomes ambiguous, as described in the previous sidebar "More Member References". Analysis Services retrieves the first FY 2003 member within the Fiscal Year level of the user-hierarchy it encounters. It then proceeds with the translation process, as previously described.

6. Modify the query to use an ambiguous reference to the FY 2003 member of the Calendar-To-Fiscal Year user-hierarchy:

```
SELECT
FROM [Chapter 3 Cube]
WHERE (
    [Date].[Calendar-To-Fiscal Year].[Fiscal Year].[FY 2003]
    )
```

7. Execute the query and note the result.

By simply removing the parent member identifier, a different cell is accessed. Analysis Services searches the Fiscal Year level for a member named FY 2003 and the first FY 2003 member encountered just so happens to be the member associated with the CY 2002 member of the Calendar Year level. You can verify this by explicitly requesting this cell and comparing its value to that of the previous query.

8. Modify the query to reflect the translated tuple:

```
SELECT
FROM [Chapter 3 Cube]
WHERE (
    [Date].[Calendar Year].[CY 2002],
    [Date].[Fiscal Year].[FY 2003]
    )
```

9. Execute the query and compare its results to those of the previous query.

These steps demonstrate the process by which a reference to a leaf-level member in a user-hierarchy is translated into attribute-hierarchy references. You would expect this process to work the same for references to non-leaf members, and it does. When a reference to a non-leaf member in a user-hierarchy is made, the member is identified along with its ancestors, just as before. Descendant members, those related to the specified member in lower levels of the hierarchy are simply ignored for the purposes of translation.

10. Modify the query, specifying a member from the Calendar Year level of the user-hierarchy:

```
SELECT
FROM [Chapter 3 Cube]
WHERE (
    [Date].[Calendar-To-Fiscal Year].[Calendar Year].[CY 2002]
    )
```

11. Execute the query.

The CY 2002 member is located within the Calendar Year level of the Calendar-To-Fiscal Year user-hierarchy. This is a non-leaf level. As before, the specified member, CY 2002, is located. That member and the members in its lineage, of which there are none (of any relevance), are translated into attribute-hierarchy references. No Fiscal Year attribute-hierarchy member reference is created, as illustrated in Figure 3-10.

You can verify this by submitting the translated tuple and comparing its results to that of the prior query.

12. Modify the query to reflect the translated tuple:

```
SELECT
FROM [Chapter 3 Cube]
WHERE (
    [Date].[Calendar Year].[CY 2002]
    )
```

Position	User-Hierarchy Translation	Partial Tuple	Rule 1: Default Member	Rule 2: (All) Member	Rule 3: First Member	Completed Tuple
Date. Calendar-To-Fiscal Year	Calendar Year. CY 2002					
Date. Calendar Year	CY 2002					CY 2002
Date. Fiscal Year		*(omitted)*	*(not available)*	All Periods		All Periods
Product. Category		*(omitted)*	*(not available)*	All Products		All Products
Product. Subcategory		*(omitted)*	*(not available)*	All Products		All Products
Measures. Measures		*(omitted)*	Reseller Sales Amount			Reseller Sales Amount

FIGURE 3-10 The process for completing the partial tuple specifying the member CY 2002 within the Calendar-To-Fiscal Year user-hierarchy

13. Execute the query and compare the result to those in step 11.

Avoiding Reference Conflicts

As has been mentioned, user-hierarchies are assembled from attribute-hierarchies. The translation process described in this chapter deconstructs a user-hierarchy member reference into its associated attribute-hierarchy member references. But, what if a tuple already contains a reference to one of the attribute-hierarchies from which the user-hierarchy is derived? This creates an opportunity for the translation to generate conflicting attribute-hierarchy references.

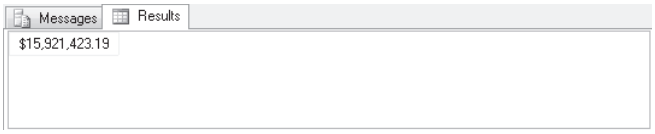
Access cells with tuples containing overlapping references

1. If you have not already done so, open the MDX Query Editor to the MDX Step-by-Step database.
2. In the code pane, enter the following query to employ references to both the Calendar-To-Fiscal Year user-hierarchy and Fiscal Year attribute-hierarchy:

```
SELECT
FROM [Chapter 3 Cube]
```

```
WHERE (
    [Date].[Calendar-To-Fiscal Year].[Calendar Year].[CY 2002],
    [Date].[Fiscal Year].[FY 2003]
    )
```

3. Execute the query.



The process of translation and tuple completion is illustrated in Figure 3-11.

Position	**User-Hierarchy Translation**	**Partial Tuple**	**Rule 1: Default Member**	**Rule 2: (All) Member**	**Rule 3: First Member**	**Completed Tuple**
Date. Calendar-To-Fiscal Year	Calendar Year. CY 2002					
Date. Calendar Year	CY 2002					CY 2002
Date. Fiscal Year		FY 2003				FY 2003
Product. Category		*(omitted)*	*(not available)*	All Products		All Products
Product. Subcategory		*(omitted)*	*(not available)*	All Products		All Products
Measures. Measures		*(omitted)*	Reseller Sales Amount			Reseller Sales Amount

FIGURE 3-11 The process for completing the partial tuple specifying the member CY 2002 within the Calendar-To-Fiscal Year user-hierarchy and FY 2003 within the Fiscal Year attribute-hierarchy

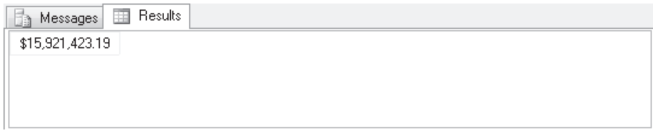
Although the tuple is syntactically valid, the combination of references to an attribute-hierarchy and a user-hierarchy based on that same attribute-hierarchy creates an opportunity for overlapping references following translation. In the previous query, this was avoided. The same is not true in the next query.

4. Modify the query to create an overlapping reference to FY 2003:

```
SELECT
FROM [Chapter 3 Cube]
WHERE (
    [Date].[Calendar-To-Fiscal Year].[Calendar Year].[CY 2002].[FY 2003],
    [Date].[Fiscal Year].[FY 2003]
    )
```

5. Execute the query.

   The translation process is illustrated in Figure 3-12.



Position	User-Hierarchy Translation	Partial Tuple	Rule 1: Default Member	Rule 2: (All) Member	Rule 3: First Member	Completed Tuple
Date. Calendar-To-Fiscal Year	Calendar Year. CY 2002. FY 2003					
Date. Calendar Year	CY 2002 →					CY 2002
Date. Fiscal Year	FY 2003 →	FY 2003 →				FY 2003
Product. Category		*(omitted)*	*(not available)*	All Products →		All Products
Product. Subcategory		*(omitted)*	*(not available)*	All Products →		All Products
Measures. Measures		*(omitted)*	Reseller Sales Amount →			Reseller Sales Amount

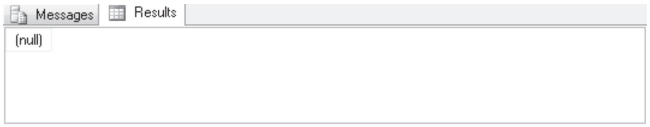
FIGURE 3-12 The process for completing the partial tuple specifying overlapping references to the FY 2003 member

Here, the user-hierarchy member reference is translated to Calendar Year and Fiscal Year attribute-hierarchy references. The tuple already employs a Fiscal Year attribute-hierarchy reference creating overlap. The overlap has a happy ending since the two Fiscal Year attribute-hierarchy member references are identical. Had this not been the case, the overlap would have created a conflict, resulting in an invalid tuple.

6. Modify the query to create an overlapping reference with conflicting Fiscal Year members:

```
SELECT
FROM [Chapter 3 Cube]
WHERE (
    [Date].[Calendar-To-Fiscal Year].[Calendar Year].[CY 2002].[FY 2003],
    [Date].[Fiscal Year].[FY 2002]
    )
```

7. Execute the query.



In this query, the user-hierarchy member reference is translated into Calendar Year and Fiscal Year attribute-hierarchy references. As shown in Figure 3-13, the FY 2003 member reference created through this process conflicts with the FY 2002 attribute-hierarchy member reference. The conflict in member references results in an invalid reference to the Fiscal Year attribute-hierarchy, which results in an empty cell being returned.

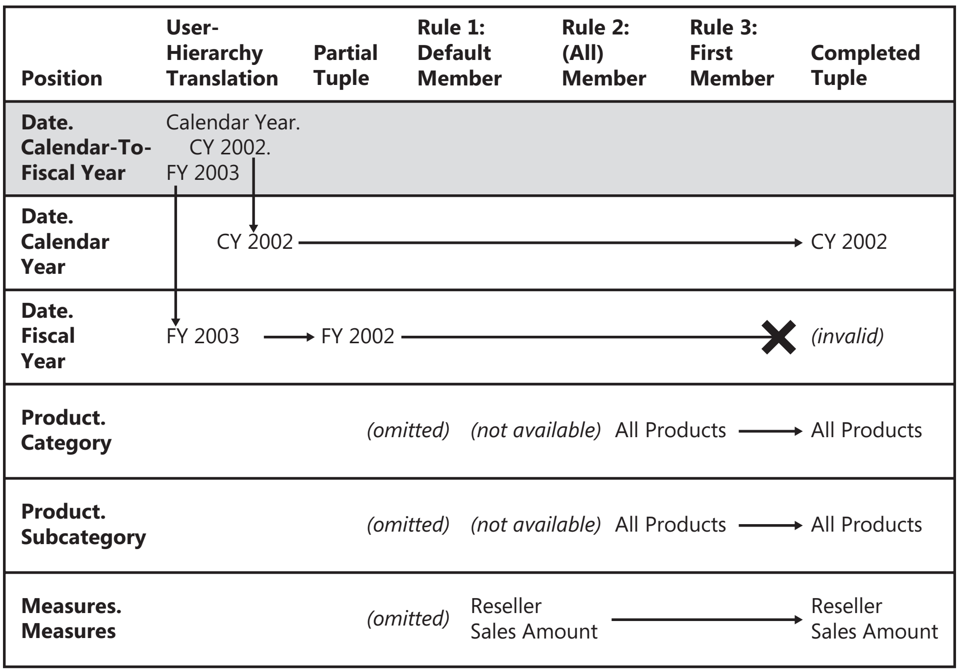


FIGURE 3-13 The process for completing the partial tuple specifying conflicting overlapping members from the Calendar-To-Fiscal Year user-hierarchy and Fiscal Year attribute-hierarchy

For the reason demonstrated here, it is recommended you consider the possibility of overlap when employing references to user-hierarchies in combination with references to the attribute-hierarchies from which they are derived.

> **Note** Analysis Services enforces a rule that a hierarchy can be referenced no more than once in a given tuple. The process of translation as demonstrated in the last two queries can result in redundant (overlapping) member references, which violates this rule without triggering an error. When working with combinations of attribute and user-hierarchies from a given dimension, be certain to understand which attribute-hierarchies are ultimately being referenced, and employ member references in a way that minimizes the potential for overlapping member references.

Member Reference Shortcuts

The last two sidebars introduced you to three forms of member reference. These forms provide greater and greater degrees of precision to address various forms of ambiguity.

However, not all member references are ambiguous. Many members are unique, whether by name or key, across all hierarchies in a dimension. Still others are unique across all hierarchies in all dimensions. In these situations, omitting the dimension or hierarchy identifier in a member reference still allows the specified member to be found without ambiguity.

Although not encouraged, Analysis Services allows you to take these shortcuts in member reference syntax. These shortcuts can include the omission of dimensions and hierarchy identifiers, allowing tuples to be expressed using a more compact format. For example, the first tuple presented in this chapter can be expressed using the shortened form:

```
(
    [Calendar Year].[All Periods],
    [Fiscal Year].[All Periods],
    [Bikes],
    [Subcategory].[Mountain Bikes],
    [Reseller Sales Amount]
)
```

Although this makes the tuple more compact (and therefore reduces the amount of typing you must do), consider some important pitfalls. First, the shortened syntax is less immediately interpretable and may be harder to support in the long run. Second, unless directed to a specific object, Analysis Services searches the various objects within the cube for matches; this results in noticeable performance overhead. Finally, and most important, Analysis Services discontinues its search as soon as a match is found. If you misjudge the ambiguity of the reference, the result of the query may not be what is expected. For this reason, we encourage you to always employ reasonably precise references supplying at a minimum the dimension and hierarchy identifiers along with the member's key or name.

Having said that, there is one shortcut we employ throughout the remainder of this book. Apart from the previous examples in this chapter, you rarely see a measure identified using its fully qualified form. Instead, measures are almost always identified using the simplified form: *[Measures].[Member]*. Although we refer to the Reseller Sales Amount measure as *[Measures].[Measures].[Reseller Sales Amount]* earlier in this chapter to demonstrate a point about measures as members, we now refer to this measure as *[Measures].[Reseller Sales Amount]* (and all other measures with the same form).

Chapter 3 Quick Reference

<table>
<tr><th>To</th><th>Do this</th></tr>
<tr><td>Reference a member by name</td><td>Write the member reference in the form [Dimension].[Hierarchy].[Member Name]. For example:<pre>[Product].[Category].[Bikes]</pre></td></tr>
<tr><td>Reference a member by key</td><td>Write the member reference in the form [Dimension].[Hierarchy].&[Member Key]. For example:<pre>[Product].[Category].&[1]</pre></td></tr>
<tr><td>Reference a member by name within a level of a user-hierarchy</td><td>Write the member reference in the form [Dimension].[Hierarchy].[Level].[Member Name]. For example:<pre>[Date].[Calendar-To-Fiscal Year].[Calendar Year].[CY 2003]</pre>In some instances this member reference is ambiguous. To avoid ambiguity, you may use a member reference that includes lineage information, such as this:<pre>[Date].[Calendar-To-Fiscal Year].[Calendar Year].[CY 2003].
[FY 2003]</pre></td></tr>
<tr><td>Reference a cell using a tuple</td><td>Write a parentheses-enclosed, comma-delimited list of member references. For example:<pre>(
    [Date].[Calendar Year].[All Periods],
    [Product].[Category].[Bikes],
    [Product].[Subcategory].[Mountain Bikes]
)</pre>Keep in mind user-hierarchy member references will be translated into attribute-hierarchy member references and any missing attribute-hierarchy member references will be supplied by Analysis Services.</td></tr>
<tr><td>Retrieve cell properties as part of the query result set</td><td>Include the CELL PROPERTIES keyword in the MDX SELECT statement, indicating the desired cell properties. For example:<pre>SELECT
FROM [Chapter 3 Cube]
WHERE (
    [Product].[Subcategory].[Mountain Bikes],
    [Date].[Calendar Year].[All Periods],
    [Date].[Fiscal Year].[All Periods],
    [Product].[Category].[Bikes],
    [Measures].[Measures].[Reseller Sales Amount]
)
CELL PROPERTIES FORMATTED_VALUE, FORMAT_STRING</pre>Otherwise, do not specify the CELL PROPERTIES keyword to return the default properties VALUE and FORMATTED_VALUE.</td></tr>
</table>

Chapter 9
Working with Time

After completing this chapter, you will be able to:

- Explain the requirements for effective time-based analysis in Analysis Services
- Employ MDX functions to calculate common time-based metrics
- Combine time-based expressions to assemble complex metrics

Time is a critical component of business analysis. Analysts interpret the state of the business now, often in relation to what it was in the past, with the goal of understanding what it might be in the future.

To support this, Analysis Services provides a number of time-based MDX functions. Using these functions, powerful metrics can be assembled. In this chapter, you learn how to employ the time-based MDX functions to calculate some of the more frequently requested of these metrics.

Understanding the Time Dimension

Analysis Services has no inherent awareness of the concept of time. Although at first glance this may seem like a shortcoming of the tool, it actually affords you the flexibility to define your time dimension in a way that reflects how time is managed in your specific organization.

At the heart of the time dimension is one or more user-hierarchies referred to as *calendars*. Calendars allow you to drill down in time from higher levels of granularity, such as years, into lower levels of granularity, such as quarters, months, and days. Figure 9-1 illustrates one such calendar hierarchy based on the standard calendar we employ in everyday life.

When employed against calendar hierarchies, the time-based MDX functions give the appearance of time awareness. However, most time-based functions are simply exploiting the basic structure of the hierarchy to return the set or member required. In fact, SQL Server Books Online goes so far as to provide the navigational equivalents of each of the time-based functions. If you require slightly different functionality, you can use the navigational functions to implement it yourself.

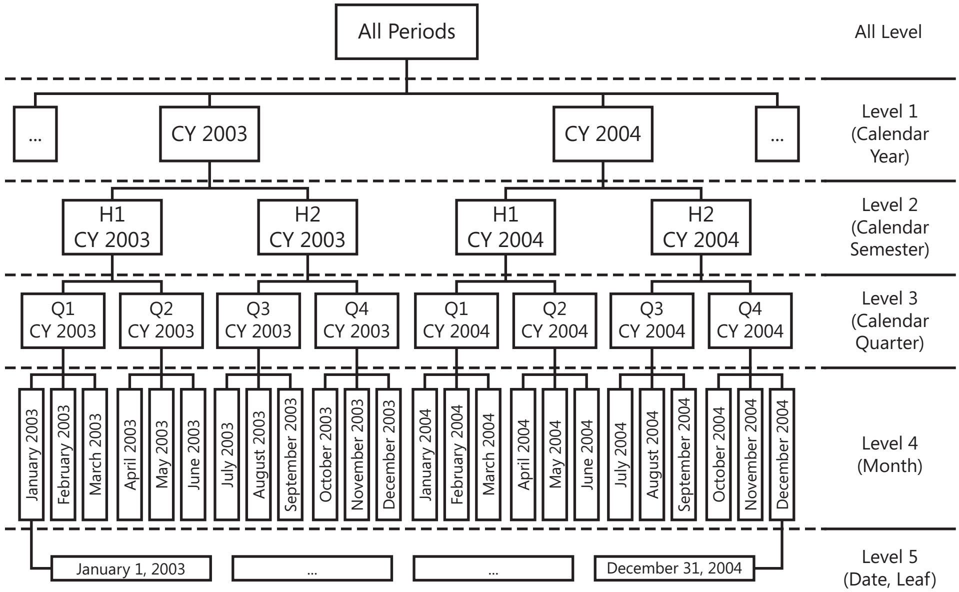


FIGURE 9-1 A user-hierarchy based on the standard calendar

The reliance on the calendar hierarchies for time-based functionality imposes two critical constraints on the attributes of the time dimension. First, the members of the attributes comprising the calendar hierarchies must be ordered in time-based sequence from the past to the present because many time-based functions assume this order. Second, complete sets of members for each attribute should be provided because missing members throw off position-based navigation.

Each of these issues is addressed through cube and ETL-layer design. As an MDX developer, you may not have the responsibility or the access required to ensure that these are addressed in a manner appropriate to your needs. However, if you intend to successfully make use of the time-based functions, you must make sure those responsible for assembling the time dimension are aware of these issues.

Determining the Current Value

A very common request is to return the current value of a metric. Although determining the current value is a seemingly simple request, it can be quite challenging.

First, you need to determine the granularity of the request. We often think of time as continuous, but in Analysis Services time is recorded as discrete members representing ranges of time. Between attributes, these members overlap so that the current date member of one attribute is associated with the current month member of another and the current quarter and year members of still others. Each of these represents quite different ranges of time, but each represents the current time.

Once you know the grain, the next challenge is to determine which member represents the current time. A key characteristic of any data warehouse is latency. The time it takes for changes to data in source systems to be reflected in the data warehouse varies from implementation to implementation, but some degree of latency is always present. Because of this, the data warehouse is only current as of some point in the past. Knowing this simply shifts the challenge from identifying the member associated with the current time to identifying the member associated with the time at which the data is current.

One technique for identifying the time at which the data is current is to employ the VBA time functions *Date*, *Time*, or *Now* to retrieve the current time, and then use the VBA date math functions *DateAdd* or *DateDiff* to adjust the time for latency. You can then use the adjusted value or parts of it extracted by using the VBA *DatePart* function to locate the current time member.

Although effective, this technique requires certainty in the amount of latency in the data. Try as you might, you may not be able to always accurately reflect this in the calculation. Considering the potential complexity of the expression logic as well, other alternatives should be explored.

A preferred alternative is to incorporate a property or attribute within the time dimension identifying a member at an appropriately low level of granularity as current. Relationships between attributes can then be employed to identify current time members at higher levels of granularity. The particulars of this design-time solution to the problem of identifying the current time member vary with the circumstances of your data warehouse, but the approach allows the data warehouse to tell you how up to date it is rather than you telling it how up to date it should be.

Calculating an Accumulating Total

In business, metrics are quite frequently reported as accumulating totals. For example, consider reseller sales in the month of October. Although sales in this month alone are interesting and important, the accumulation of sales over the months of the year up to and including October may be more interesting, especially if you are tracking sales against an annual target.

To calculate accumulating totals, you must determine the set of time members over which a value is to be aggregated. This is done using the *PeriodsToDate* function:

```
PeriodsToDate( [Level , [Member]] )
```

The *PeriodsToDate* function returns the set of members from the start of a given period up to and including a specified member. The *Level* argument identifies the level of the hierarchy representing the period over which the returned set should span, whereas the *Member* argument identifies the set's ending member. You can think of Analysis Services as starting with the specified member, navigating up to its ancestor in the specified level and then back down to the first sibling of the specified member under this shared ancestor. The set returned represents the range of members between and including these two members.

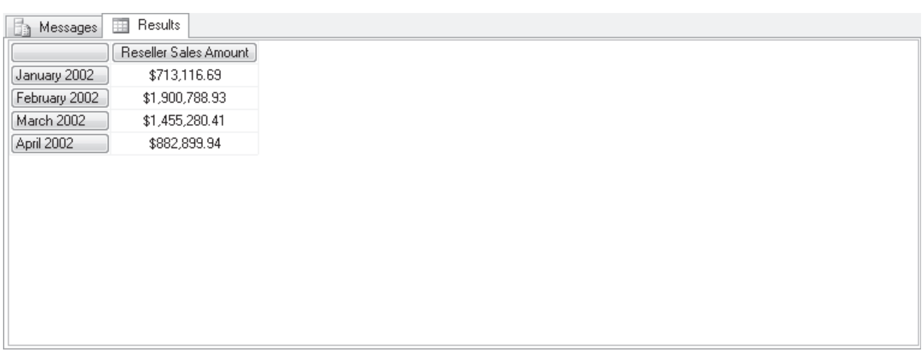
If the *Member* argument is not specified but the *Level* argument is, Analysis Services infers the current member of the hierarchy for the *Member* argument. If neither the *Member* nor the *Level* argument is specified, Analysis Services infers the current member of a hierarchy in a time dimension for the *Member* argument and the parent level of this member for the *Level* argument. For most applications of the *PeriodsToDate* function, you are encouraged to supply both arguments to ensure clarity.

Calculate year-to-date reseller sales

1. Open the MDX Query Editor to the MDX Step-by-Step database.
2. In the code pane, enter the following query to retrieve reseller sales for the periods to date for the month of April 2002:

```
SELECT
    {([Measures].[Reseller Sales Amount])} ON COLUMNS,
    {
        PeriodsToDate(
            [Date].[Calendar].[Calendar Year],
            [Date].[Calendar].[Month].[April 2002]
            )
        } ON ROWS
FROM [Step-by-Step]
```

3. Execute the query and review the results.



	Reseller Sales Amount
January 2002	$713,116.69
February 2002	$1,900,788.93
March 2002	$1,455,280.41
April 2002	$882,899.94

In the preceding query, you use the *PeriodsToDate* function to retrieve all months in the year 2002 prior to and including the month of April. By specifying the Calendar Year level of the Calendar hierarchy, Analysis Services moves from the member April 2002 to its ancestor along this level, CY 2002. It then selects the CY 2002 member's first descendant within the Month level—the level occupied by the specified member April 2002. This first descendant, January 2002, and the specified member, April 2002, then are used to form a range, [Date].[Calendar].[Month].[January 2002]:[Date].[Calendar].[Month].[April 2002], which resolves to the set presented along the *ROWS* axis.

This query demonstrates the basic functionality of the *PeriodsToDate* function, but your goal is to calculate a year-to-date total for reseller sales. Instead of using *PeriodsToDate* to define a set along an axis, you can use the function to define the set over which you aggregate values in a calculated member. As a starting point towards this goal, re-factor the query to return all months along the *ROWS* axis.

4. Modify the query to retrieve reseller sales for each month:

```
SELECT
    {([Measures].[Reseller Sales Amount])} ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```

5. Execute the query and review the results.

Messages | Results

	Reseller Sales Amount
July 2001	$489,328.58
August 2001	$1,538,408.31
September 2001	$1,165,897.08
October 2001	$844,721.00
November 2001	$2,324,135.80
December 2001	$1,702,944.54
January 2002	$713,116.69
February 2002	$1,900,788.93
March 2002	$1,455,280.41
April 2002	$882,899.94
May 2002	$2,269,116.71
June 2002	$1,001,803.77

6. Modify the query to calculate the year-to-date cumulative reseller sales for each member along the *ROWS* axis:

```
WITH
MEMBER [Measures].[Year to Date Reseller Sales] AS
    Aggregate(
        PeriodsToDate(
            [Date].[Calendar].[Calendar Year],
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Year to Date Reseller Sales])
        } ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```

7. Execute the query and review the results.

	Reseller Sales Amount	Year to Date Reseller Sales
July 2001	$489,328.58	$489,328.58
August 2001	$1,538,408.31	$2,027,736.89
September 2001	$1,165,897.08	$3,193,633.97
October 2001	$844,721.00	$4,038,354.97
November 2001	$2,324,135.80	$6,362,490.76
December 2001	$1,702,944.54	$8,065,435.31
January 2002	$713,116.69	$713,116.69
February 2002	$1,900,788.93	$2,613,905.62
March 2002	$1,455,280.41	$4,069,186.04
April 2002	$882,899.94	$4,952,085.98
May 2002	$2,269,116.71	$7,221,202.69
June 2002	$1,001,803.77	$8,223,006.46

For each member along the *ROWS* axis, the *PeriodsToDate* function returns the set of members from the start of its calendar year up to and including this member. Over this set, the current measure, Reseller Sales Amount, is aggregated to calculate year-to-date sales. Comparing the year-to-date totals to the monthly sales values for previous months, you can verify this logic.

Note The preceding calculation employs the *Aggregate* function to calculate a running total. For more information on this and the other MDX aggregation functions, see Chapter 7, "Performing Aggregation."

As you review these results, notice between December 2001 and January 2002 the value of the accumulating total "resets." This is because these two members have differing ancestor members within the Calendar Year level. This pattern of accumulation and reset is observed whenever transitions between ancestors occur, as demonstrated in the following calculations of quarter-to-date totals.

8. Add a quarter-to-date total for reseller sales to the query:

```
WITH
MEMBER [Measures].[Year to Date Reseller Sales] AS
    Aggregate(
        PeriodsToDate(
            [Date].[Calendar].[Calendar Year],
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
MEMBER [Measures].[Quarter to Date Reseller Sales] AS
    Aggregate(
        PeriodsToDate(
            [Date].[Calendar].[Calendar Quarter],
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
```

```
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Year to Date Reseller Sales]),
        ([Measures].[Quarter to Date Reseller Sales])
        } ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```

9. Execute the query and review the new Quarter To Date Reseller Sales values.

	Reseller Sales Amount	Year to Date Reseller Sales	Quarter to Date Reseller Sales
July 2001	$489,328.58	$489,328.58	$489,328.58
August 2001	$1,538,408.31	$2,027,736.89	$2,027,736.89
September 2001	$1,165,897.08	$3,193,633.97	$3,193,633.97
October 2001	$844,721.00	$4,038,354.97	$844,721.00
November 2001	$2,324,135.80	$6,362,490.76	$3,168,856.79
December 2001	$1,702,944.54	$8,065,435.31	$4,871,801.34
January 2002	$713,116.69	$713,116.69	$713,116.69
February 2002	$1,900,788.93	$2,613,905.62	$2,613,905.62
March 2002	$1,455,280.41	$4,069,186.04	$4,069,186.04
April 2002	$882,899.94	$4,952,085.98	$882,899.94
May 2002	$2,269,116.71	$7,221,202.69	$3,152,016.65
June 2002	$1,001,803.77	$8,223,006.46	$4,153,820.42

Reviewing the results, you can see the same pattern of accumulation and reset with the Quarter To Date Reseller Sales calculated measure as you do with the Year To Date Reseller Sales calculated measure. The only difference is that the pattern is based on a quarterly cycle as opposed to an annual one.

Simplifying Periods-to-Date Calculations

Many of the attributes in a time dimension are assigned *Type* property values at design time, identifying the attributes as representing years, quarters, months, or weeks. Analysis Services can return period-to-date sets based on these type assignments without the identification of a level by name. This functionality is provided through the specialized *Ytd*, *Qtd*, *Mtd*, and *Wtd* functions returning year-to-date, quarter-to-date, month-to-date, and week-to-date sets, respectively:

```
Ytd( [Member] )
Qtd( [Member] )
Mtd( [Member] )
Wtd( [Member] )
```

These functions, collectively referred to as the *xTD* functions, are logically equivalent to the *PeriodsToDate* function with hard-coded level arguments. Their reliance on the proper assignment of *Type* property values at design time makes them more succinct but also makes them dependent on settings into which you may have little insight. If you use the *xTD* functions, it is important for you to verify the set returned.

To demonstrate the use of the *xTD* functions, the last query of the previous exercise is rewritten using *Ytd* and *Qtd* to derive the year-to-date and quarter-to-date sets, respectively:

```
WITH
MEMBER [Measures].[Year to Date Reseller Sales] AS
    Aggregate(
        Ytd([Date].[Calendar].CurrentMember),
        ([Measures].[Reseller Sales Amount])
        )
MEMBER [Measures].[Quarter to Date Reseller Sales] AS
    Aggregate(
        Qtd([Date].[Calendar].CurrentMember),
        ([Measures].[Reseller Sales Amount])
        )
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Year to Date Reseller Sales]),
        ([Measures].[Quarter to Date Reseller Sales])
        } ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```

	Reseller Sales Amount	Year to Date Reseller Sales	Quarter to Date Reseller Sales
July 2001	$489,328.58	$489,328.58	$489,328.58
August 2001	$1,538,408.31	$2,027,736.89	$2,027,736.89
September 2001	$1,165,897.08	$3,193,633.97	$3,193,633.97
October 2001	$844,721.00	$4,038,354.97	$844,721.00
November 2001	$2,324,135.80	$6,362,490.76	$3,168,856.79
December 2001	$1,702,944.54	$8,065,435.31	$4,871,801.34
January 2002	$713,116.69	$713,116.69	$713,116.69
February 2002	$1,900,788.93	$2,613,905.62	$2,613,905.62
March 2002	$1,455,280.41	$4,069,186.04	$4,069,186.04
April 2002	$882,899.94	$4,952,085.98	$882,899.94
May 2002	$2,269,116.71	$7,221,202.69	$3,152,016.65
June 2002	$1,001,803.77	$8,223,006.46	$4,153,820.42

Calculating Inception-to-Date

The period-to-date calculations return a value based on a range that is restricted to a particular period, such as a quarter or year. Occasionally, you may wish to calculate an accumulating value across all periods for which data is recorded. This is referred to as an *inception-to-date* value.

You can retrieve the inception-to-date range using the *PeriodsToDate* function with the calendar's (All) member's level as the period identifier, as demonstrated in the following expression:

```
PeriodsToDate(
    [Date].[Calendar].[(All)],
    [Date].[Calendar].CurrentMember
    )
```

Although this expression is perfectly valid, many MDX developers typically calculate inception-to-date sets employing a range-based shortcut:

```
Null: [Date].[Calendar].CurrentMember
```

The Null member reference forces Analysis Services to evaluate the range from a position just prior to the first member of the level on which the current time member resides. The result is the same set returned by the previous expression that employed the *PeriodsToDate* function.

Whichever technique you employ, measures are aggregated over the set just as with other period-to-date calculations, as demonstrated in the following example:

```
WITH
MEMBER [Measures].[Inception to Date Reseller Sales - PTD] AS
    Aggregate(
        PeriodsToDate(
            [Date].[Calendar].[(All)],
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
MEMBER [Measures].[Inception to Date Reseller Sales - Range] AS
    Aggregate(
        NULL:[Date].[Calendar].CurrentMember,
        ([Measures].[Reseller Sales Amount])
        )
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Inception to Date Reseller Sales - PTD]),
        ([Measures].[Inception to Date Reseller Sales - Range])
        } ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```

Messages | Results

	Reseller Sales Amount	Inception to Date Reseller Sales - PTD	Inception to Date Reseller Sales - Range
July 2001	$489,328.58	$489,328.58	$489,328.58
August 2001	$1,538,408.31	$2,027,736.89	$2,027,736.89
September 2001	$1,165,897.08	$3,193,633.97	$3,193,633.97
October 2001	$844,721.00	$4,038,354.97	$4,038,354.97
November 2001	$2,324,135.80	$6,362,490.76	$6,362,490.76
December 2001	$1,702,944.54	$8,065,435.31	$8,065,435.31
January 2002	$713,116.69	$8,778,552.00	$8,778,552.00
February 2002	$1,900,788.93	$10,679,340.93	$10,679,340.93
March 2002	$1,455,280.41	$12,134,621.34	$12,134,621.34
April 2002	$882,899.94	$13,017,521.29	$13,017,521.29
May 2002	$2,269,116.71	$15,286,638.00	$15,286,638.00
June 2002	$1,001,803.77	$16,288,441.77	$16,288,441.77

Calculating Rolling Averages

Analysts often look for changes in values over time. Natural variability in most data can make it difficult to identify meaningful changes. Rolling averages are frequently employed to *smooth out* some of this variation, allowing more significant or longer-term changes to be more readily identified.

A rolling average is calculated as the average of values for some number of periods before or after (and including) the period of interest. For example, the three-month rolling average of sales for the month of February might be determined as the average of sales for February, January, and December. A three-month rolling average calculated in this manner is common in business analysis.

The heart of the rolling average calculation is the determination of the set of periods over which values will be averaged. To support the retrieval of this set, the MDX function *LastPeriods* is provided:

```
LastPeriods( n [, Member] )
```

The *LastPeriods* function returns a set of *n* members before or after (and including) a specified member of a time hierarchy. If a positive *n* value is provided, the set returned includes the members preceding the member of interest. If a negative *n* value is provided, the set returned includes the members following the member of interest.

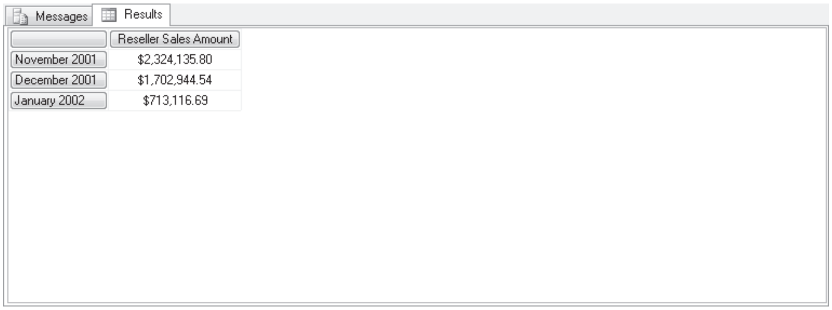
The function's second argument is optional. If the second argument is not supplied, Analysis Services assumes the current member of a hierarchy in a time dimension. For most applications of the *LastPeriods* function, you are encouraged to employ the *Member* argument to ensure clarity.

Calculate the three-month rolling average for reseller sales

1. Open the MDX Query Editor to the MDX Step-by-Step database.
2. In the code pane, enter the following query to retrieve reseller sales for the three periods preceding and including January 2002:

```
SELECT
    {([Measures].[Reseller Sales Amount])} ON COLUMNS,
    {
        LastPeriods(
            3,
            [Date].[Calendar].[Month].[January 2002]
            )
        } ON ROWS
FROM [Step-by-Step]
```

3. Execute the query and review the results.



	Reseller Sales Amount
November 2001	$2,324,135.80
December 2001	$1,702,944.54
January 2002	$713,116.69

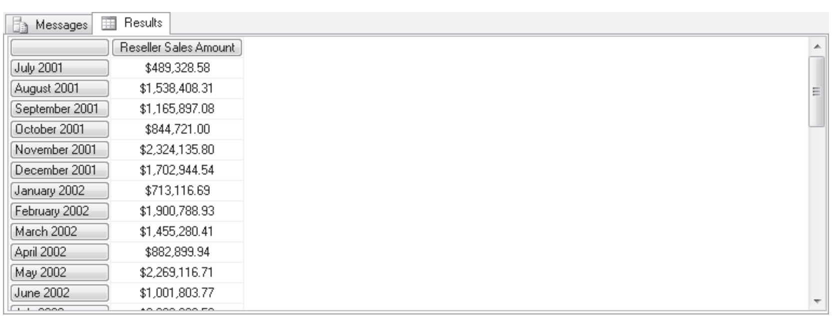
In this query, you use the *LastPeriods* function to retrieve the three-month period preceding and including January 2002. Analysis Services starts with the specified member, January 2002, and treats this as period 1. This leaves *n*-1 or 2 members to return in the set. Because *n* is a positive number, Analysis Services retrieves the January 2002 member's two preceding siblings to complete the set. (Notice that the November and December 2001 siblings were selected without regard for the change in the Calendar Year ancestor between them and the January 2002 member.)

This query demonstrates the basic functionality of the *LastPeriods* function, but your goal is to calculate a rolling average for reseller sales. Instead of using *LastPeriods* to define a set along an axis, you can use the function to define the set over which you will average values in a calculated member. As a starting point towards this goal, re-factor the query to return all months along the *ROWS* axis.

4. Alter the query to retrieve reseller sales for various months:

```
SELECT
    {([Measures].[Reseller Sales Amount])} ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```

5. Execute the query and review the results.



	Reseller Sales Amount
July 2001	$489,328.58
August 2001	$1,538,408.31
September 2001	$1,165,897.08
October 2001	$844,721.00
November 2001	$2,324,135.80
December 2001	$1,702,944.54
January 2002	$713,116.69
February 2002	$1,900,788.93
March 2002	$1,455,280.41
April 2002	$882,899.94
May 2002	$2,269,116.71
June 2002	$1,001,803.77

Reseller sales vary considerably between various months. For example, take a look at the six-month period between October 2001 and March 2002. The wild swings between monthly sales make it difficult to determine any general upward or downward trends during this period. The same is true of the months between June 2002 and December 2002.

6. Alter the query to calculate a three-month rolling average for reseller sales:

```
WITH
MEMBER [Measures].[Three Month Avg Reseller Sales Amount] AS
    Avg(
        LastPeriods(
            3,
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Three Month Avg Reseller Sales Amount])
        } ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```

7. Execute the query and compare the monthly reseller sales values to the three-month rolling average values.

	Reseller Sales Amount	Three Month Avg Reseller Sales Amount
July 2001	$489,328.58	$489,328.58
August 2001	$1,538,408.31	$1,013,868.45
September 2001	$1,165,897.08	$1,064,544.66
October 2001	$844,721.00	$1,183,008.80
November 2001	$2,324,135.80	$1,444,917.96
December 2001	$1,702,944.54	$1,623,933.78
January 2002	$713,116.69	$1,580,065.68
February 2002	$1,900,788.93	$1,438,950.06
March 2002	$1,455,280.41	$1,356,395.35
April 2002	$882,899.94	$1,412,989.76
May 2002	$2,269,116.71	$1,535,765.69
June 2002	$1,001,803.77	$1,384,606.81

The three-month rolling average smoothes out some of the variability in the data, making general trends more easily observed. The period from October 2001 to March 2002 that reflected so much variability based on monthly sales totals now appears to be trending only slightly upward. The period from June 2002 and December 2002 that also displayed considerable variability appears to be trending more significantly upward. Without the smoothing effect of the rolling average, these trends would be harder to observe and differentiate.

Performing Period-over-Period Analysis

Historical values are frequently used in data analysis to provide perspective on current values. When comparing historical to current values, it is important you select values from time periods relatively similar to one another. Although no two time periods are exactly alike, analysts often compare values from what are referred to as *parallel periods* to minimize differences resulting from cyclical, time-dependent variations in the data.

To understand parallel periods, consider the month of April 2003. This month is the fourth month of the calendar year 2003. In a business heavily influenced by annual cycles, you might compare values for this month to those for the month of April in a prior year. In doing so, you might accurately (or inaccurately) assume that differences in current and historical values are due to factors other than the annual cyclical influence.

Should you compare values for April 2003 to those of January 2003 or October 2002? Your first response may be to say no. However, if your business is heavily influenced by quarterly cycles, this might be completely appropriate. April 2003 is the first month of a calendar quarter. January 2003 is the first month of the prior quarter and is therefore a parallel member based on quarter. October 2002 is also a parallel member except that it is from two quarters prior. What constitutes an appropriate parallel period for your analysis is highly dependent upon the time-based cycles influencing your business.

To assist you with the retrieval of parallel period members, Analysis Services provides the *ParallelPeriod* function:

```
ParallelPeriod( [Level [,n [, Member]]] )
```

The function's first argument identifies the level of the time hierarchy across which you wish to identify the parallel period member. If no level is identified, the parent level of the current time member is assumed.

The function's second argument identifies how far back along the identified level you wish to go to retrieve the parallel member. If no value is provided, a value of 1 is assumed, indicating the prior period.

The function's final argument identifies the member for which the parallel period is to be determined. The position of this member relative to its ancestor in the specified level determines the member retrieved from the historical period. If no member is identified, the current time member is assumed.

Calculate growth over prior period

1. Open the MDX Query Editor to the MDX Step-by-Step database.
2. In the code pane, enter the following query to retrieve reseller sales for the months of calendar year 2003:

```
SELECT
    {([Measures].[Reseller Sales Amount])} ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[Calendar Year].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
         } ON ROWS
FROM [Step-by-Step]
```

3. Execute the query and review the results.

	Reseller Sales Amount
January 2003	$1,317,541.83
February 2003	$2,384,846.59
March 2003	$1,563,955.08
April 2003	$1,865,278.43
May 2003	$2,880,752.68
June 2003	$1,987,872.71
July 2003	$2,665,650.54
August 2003	$4,212,971.51
September 2003	$4,047,574.04
October 2003	$2,282,115.88
November 2003	$3,483,161.40
December 2003	$3,510,948.73

The query returns reseller sales for the months of calendar year 2003. To assess the strength of these numbers in a business influenced by annual sales cycles, you might compare them to sales in the prior year. To do this, start by identifying the prior period for each month.

4. Alter the query to identify the parallel period in the prior year for each month:

```
WITH
MEMBER [Measures].[x] AS
    ParallelPeriod(
        [Date].[Calendar].[Calendar Year],
        1,
        [Date].[Calendar].CurrentMember
        ).Name
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[x])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[Calendar Year].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
         } ON ROWS
FROM [Step-by-Step]
```

5. Execute the query and review the results.

	Reseller Sales Amount	x
January 2003	$1,317,541.83	January 2002
February 2003	$2,384,846.59	February 2002
March 2003	$1,563,955.08	March 2002
April 2003	$1,865,278.43	April 2002
May 2003	$2,880,752.68	May 2002
June 2003	$1,987,872.71	June 2002
July 2003	$2,665,650.54	July 2002
August 2003	$4,212,971.51	August 2002
September 2003	$4,047,574.04	September 2002
October 2003	$2,282,115.88	October 2002
November 2003	$3,483,161.40	November 2002
December 2003	$3,510,948.73	December 2002

In the preceding query, the *ParallelPeriod* function is used to identify the parallel period in the prior year for each month in calendar year 2003 along the *ROWS* axis. The *ParallelPeriod* function returns a member and the name of that member is returned with a new calculated member to verify that the appropriate member is being identified. Now that you are comfortable the correct member is being located, you can use the returned member to determine prior period sales.

6. Alter the query to calculate prior period sales:

```
WITH
MEMBER [Measures].[Prior Period Reseller Sales Amount] AS
    (
        ParallelPeriod(
            [Date].[Calendar].[Calendar Year],
            1,
            [Date].[Calendar].CurrentMember
            ),
        [Measures].[Reseller Sales Amount]
        )
    ,FORMAT="Currency"
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Prior Period Reseller Sales Amount])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[Calendar Year].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
         } ON ROWS
FROM [Step-by-Step]
```

7. Execute the query and review the results.

Messages | Results

	Reseller Sales Amount	Prior Period Reseller Sales Amount
January 2003	$1,317,541.83	$713,116.69
February 2003	$2,384,846.59	$1,900,788.93
March 2003	$1,563,955.08	$1,455,280.41
April 2003	$1,865,278.43	$882,899.94
May 2003	$2,880,752.68	$2,269,116.71
June 2003	$1,987,872.71	$1,001,803.77
July 2003	$2,665,650.54	$2,393,689.53
August 2003	$4,212,971.51	$3,601,190.71
September 2003	$4,047,574.04	$2,885,359.20
October 2003	$2,282,115.88	$1,802,154.21
November 2003	$3,483,161.40	$3,053,816.33
December 2003	$3,510,948.73	$2,185,213.21

Using the member returned by the *ParallelPeriod* function to assemble a tuple allows you to retrieve reseller sales for the prior period. This newly calculated measure is returned along the *COLUMNS* axis for comparison against sales in the months displayed across the rows. To facilitate comparison, you might wish to present the percent change in sales from the prior period.

8. Alter the query to calculate the percent change in sales (growth) between the current and prior periods:

```
WITH
MEMBER [Measures].[Prior Period Reseller Sales Amount] AS
    (
        ParallelPeriod(
            [Date].[Calendar].[Calendar Year],
            1,
            [Date].[Calendar].CurrentMember
            ),
        [Measures].[Reseller Sales Amount]
        )
    ,FORMAT="Currency"
MEMBER [Measures].[Prior Period Growth] AS
    (
        ([Measures].[Reseller Sales Amount])-
            ([Measures].[Prior Period Reseller Sales Amount])
        ) /
        ([Measures].[Prior Period Reseller Sales Amount])
    ,FORMAT="Percent"
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Prior Period Reseller Sales Amount]),
        ([Measures].[Prior Period Growth])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[Calendar Year].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
         } ON ROWS
FROM [Step-by-Step]
```

9. Execute the query and review the results.

Messages | Results

	Reseller Sales Amount	Prior Period Reseller Sales Amount	Prior Period Growth
January 2003	$1,317,541.83	$713,116.69	84.76%
February 2003	$2,384,846.59	$1,900,788.93	25.47%
March 2003	$1,563,955.08	$1,455,280.41	7.47%
April 2003	$1,865,278.43	$882,899.94	111.27%
May 2003	$2,880,752.68	$2,269,116.71	26.95%
June 2003	$1,987,872.71	$1,001,803.77	98.43%
July 2003	$2,665,650.54	$2,393,689.53	11.36%
August 2003	$4,212,971.51	$3,601,190.71	16.99%
September 2003	$4,047,574.04	$2,885,359.20	40.28%
October 2003	$2,282,115.88	$1,802,154.21	26.63%
November 2003	$3,483,161.40	$3,053,816.33	14.06%
December 2003	$3,510,948.73	$2,185,213.21	60.67%

The results show each month of calendar year 2003 experienced considerable growth in reseller sales from those of the month in the prior year.

A Word of Caution

As explained at the start of this chapter, the time-based MDX functions are not time-aware and simply employ basic navigation for their functionality. This is illustrated by rewriting the query in Step 4 of the previous exercise with the navigation functions *Cousin*, *Ancestor*, and *Lag*:

```
WITH
MEMBER [Measures].[x] AS
    Cousin(
        [Date].[Calendar].CurrentMember,
        Ancestor(
            [Date].[Calendar].CurrentMember,
            [Date].[Calendar].[Calendar Year]
            ).Lag(1)
        ).Name
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[x])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[Calendar Year].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
         } ON ROWS
FROM [Step-by-Step]
```

Messages | Results

	Reseller Sales Amount	x
January 2003	$1,317,541.83	January 2002
February 2003	$2,384,846.59	February 2002
March 2003	$1,563,955.08	March 2002
April 2003	$1,865,278.43	April 2002
May 2003	$2,880,752.68	May 2002
June 2003	$1,987,872.71	June 2002
July 2003	$2,665,650.54	July 2002
August 2003	$4,212,971.51	August 2002
September 2003	$4,047,574.04	September 2002
October 2003	$2,282,115.88	October 2002
November 2003	$3,483,161.40	November 2002
December 2003	$3,510,948.73	December 2002

As previously mentioned, the use of basic navigation to provide time-based functionality imposes some constraints on your time dimension. One of these is that all members of a time period should be provided in the cube. Again, the query in Step 4 from the previous exercise provides a very clear demonstration of why this is important. Here is that query adjusted to present the months of calendar year 2002 along the *ROWS* axis:

```
WITH
MEMBER [Measures].[x] AS
    ParallelPeriod(
        [Date].[Calendar].[Calendar Year],
```

```
        1,
        [Date].[Calendar].CurrentMember
        ).Name
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[x])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[Calendar Year].[CY 2002],
            [Date].[Calendar].[Month],
            SELF
            )
         } ON ROWS
FROM [Step-by-Step]
```

Messages | Results

	Reseller Sales Amount	x
January 2002	$713,116.69	July 2001
February 2002	$1,900,788.93	August 2001
March 2002	$1,455,280.41	September 2001
April 2002	$882,899.94	October 2001
May 2002	$2,269,116.71	November 2001
June 2002	$1,001,803.77	December 2001
July 2002	$2,393,689.53	(null)
August 2002	$3,601,190.71	(null)
September 2002	$2,885,359.20	(null)
October 2002	$1,802,154.21	(null)
November 2002	$3,053,816.33	(null)
December 2002	$2,185,213.21	(null)

Notice in the results of this query that the month of January 2002 has a parallel period of July 2001. January 2002 is the first month-level descendant of calendar year 2002. Its parallel period in the prior year is the first month-level descendant of calendar year 2001. Because the first month recorded in 2001 is July, July 2001 becomes the parallel period of January 2002 based on simple navigation. Apply this logic to July 2002, the seventh month-level descendant of calendar year 2002, and you see why it has no parallel period in 2001, a year in which only six months were recorded.

If all twelve months for calendar year 2001 had been recorded, this problem could have been avoided. However, this problem would now be deferred to the fiscal calendar whose years start prior to 2001. In other words, there is no way in this dimension to provide complete sets of members under each period.

So what's the solution to this problem? The short answer is there really isn't one. You as the query developer must be aware of boundary issues such as this when developing queries employing time-based functions. You might have data at the head and tail of the time dimension extended to cover periods for which no data is recorded to avoid misalignment as illustrated previously, but you still need to be aware that no data is recorded for those periods so that some forms of analysis, such as period-over-period growth, might not be appropriate.

Combining Time-Based Metrics

Throughout this chapter, you have explored the various time-based functions and how they can be used to enhance business analysis and solve business problems. Although each of these functions is valuable on its own, they are often used in combination to provide even greater insight and clarity into the analysis of business data. These may seem like very challenging metrics to assemble, but in reality they are no more complex than most other metrics calculated throughout this book. The trick is to remember tuple and expression basics.

Calculate year-to-date and prior period year-to-date sales

1. Open the MDX Query Editor to the MDX Step-by-Step database.
2. Enter the following query to retrieve reseller sales for the months of calendar year 2003:

```
SELECT
    {
        ([Measures].[Reseller Sales Amount])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
        } ON ROWS
FROM [Step-by-Step]
```

3. Execute the query and review the results.

	Reseller Sales Amount
January 2003	$1,317,541.83
February 2003	$2,384,846.59
March 2003	$1,563,955.08
April 2003	$1,865,278.43
May 2003	$2,880,752.68
June 2003	$1,987,872.71
July 2003	$2,665,650.54
August 2003	$4,212,971.51
September 2003	$4,047,574.04
October 2003	$2,282,115.88
November 2003	$3,483,161.40
December 2003	$3,510,948.73

The query returns reseller sales by month for calendar year 2003. Using the *PeriodsToDate* function, you can calculate year-to-date sales just like before.

4. Alter the query to calculate a year-to-date sales:

```
WITH
MEMBER [Measures].[Year to Date Reseller Sales] AS
    Aggregate(
        PeriodsToDate(
            [Date].[Calendar].[Calendar Year],
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
    ,FORMAT="Currency"
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Year to Date Reseller Sales])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
        } ON ROWS
FROM [Step-by-Step]
```

5. Execute the query and review the results.

	Reseller Sales Amount	Year to Date Reseller Sales
January 2003	$1,317,541.83	$1,317,541.83
February 2003	$2,384,846.59	$3,702,388.42
March 2003	$1,563,955.08	$5,266,343.51
April 2003	$1,865,278.43	$7,131,621.94
May 2003	$2,880,752.68	$10,012,374.62
June 2003	$1,987,872.71	$12,000,247.33
July 2003	$2,665,650.54	$14,665,897.87
August 2003	$4,212,971.51	$18,878,869.38
September 2003	$4,047,574.04	$22,926,443.41
October 2003	$2,282,115.88	$25,208,559.29
November 2003	$3,483,161.40	$28,691,720.69
December 2003	$3,510,948.73	$32,202,669.43

Using the Year To Date Reseller Sales calculated member in a tuple, you can easily calculate year-to-date sales for the prior period.

6. Alter the query to calculate the prior period year-to-date sales:

```
WITH
MEMBER [Measures].[Prior Period Year to Date Reseller Sales] AS
    (
        ParallelPeriod(
            [Date].[Calendar].[Calendar Year],
            1,
            [Date].[Calendar].CurrentMember
            ),
        [Measures].[Year to Date Reseller Sales]
        )
    ,FORMAT="Currency"
```

```
MEMBER [Measures].[Year to Date Reseller Sales] AS
    Aggregate(
        PeriodsToDate(
            [Date].[Calendar].[Calendar Year],
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
    ,FORMAT="Currency"
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Year to Date Reseller Sales]),
        ([Measures].[Prior Period Year to Date Reseller Sales])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
        } ON ROWS
FROM [Step-by-Step]
```

7. Execute the query and review the results.

Messages | Results

	Reseller Sales Amount	Year to Date Reseller Sales	Prior Period Year to Date Reseller Sales
January 2003	$1,317,541.83	$1,317,541.83	$713,116.69
February 2003	$2,384,846.59	$3,702,388.42	$2,613,905.62
March 2003	$1,563,955.08	$5,266,343.51	$4,069,186.04
April 2003	$1,865,278.43	$7,131,621.94	$4,952,085.98
May 2003	$2,880,752.68	$10,012,374.62	$7,221,202.69
June 2003	$1,987,872.71	$12,000,247.33	$8,223,006.46
July 2003	$2,665,650.54	$14,665,897.87	$10,616,695.99
August 2003	$4,212,971.51	$18,878,869.38	$14,217,886.70
September 2003	$4,047,574.04	$22,926,443.41	$17,103,245.90
October 2003	$2,282,115.88	$25,208,559.29	$18,905,400.11
November 2003	$3,483,161.40	$28,691,720.69	$21,959,216.44
December 2003	$3,510,948.73	$32,202,669.43	$24,144,429.65

This exercise demonstrates a very simple approach to combining calculated members that use time-based functions. When formulating complex metrics, you can easily lose sight of the basic techniques allowing logic in one calculated member to be leveraged for another. As easily as you combined a period-to-date calculation with a prior period calculation, you could extend this query to include the difference, variance, or percent growth of the current year year-to-date values compared to the prior year year-to-date values or any flavors thereof.

The *OpeningPeriod* and *ClosingPeriod* Functions

We would be remiss if we did not mention the *OpeningPeriod* and *ClosingPeriod* functions. The introduction of expanded support for semi-additive measures in the 2005 release of

Analysis Services has diminished the role of these functions, which return the first and last members of a period:

```
OpeningPeriod( [Level [, Member]] )
ClosingPeriod( [Level [, Member]] )
```

The *OpeningPeriod* and *ClosingPeriod* functions return the first or last member, respectively, of the descendants from a given level and a specified member. If no level is specified, Analysis Services assumes the topmost level of the time hierarchy. If no member is specified, Analysis Services assumes the current time member. As with the other time-based functions, you are encouraged to supply both arguments to ensure clarity.

As previously mentioned, both the *OpeningPeriod* and *ClosingPeriod* functions have seen their use diminished with recent releases of Analysis Services. Historically, these functions have been used to calculate values now returned by the FirstChild, FirstNonEmpty, LastChild, and LastNonEmpty aggregate functions. These aggregate functions are frequently employed with finance facts, exchange rates, and other snapshot facts to identify period starting and ending values.

For example, the end-of-day exchange rate employs the LastNonEmpty aggregate function to provide access to the last available value within a given period. But what if you needed to determine the end-of-day exchange rate at the start of a period? The following query illustrates the use of the *OpeningPeriod* function to calculate this value:

```
WITH
MEMBER [Measures].[First Child Rate] AS
    (
        OpeningPeriod(
            [Date].[Calendar].[Date],
            [Date].[Calendar].CurrentMember
            ),
             [Measures].[End of Day Rate]
            )
       ,FORMAT="Standard"
SELECT
    {
        ([Measures].[First Child Rate]),
        ([Measures].[End of Day Rate])
        } ON COLUMNS,
    {[Date].[Calendar].Members} ON ROWS
FROM [Step-by-Step]
WHERE ([Destination Currency].[Destination Currency].[Euro])
```

Messages | Results

	First Child Rate	End of Day Rate
All Periods	1,03	,97
CY 2001	1,03	,91
H2 CY 2001	1,03	,91
Q3 CY 2001	1,03	,99
July 2001	1,03	1,00
July 1, 2001	1,03	1,03
July 2, 2001	1,03	1,03
July 3, 2001	1,04	1,04
July 4, 2001	1,04	1,04
July 5, 2001	1,03	1,03
July 6, 2001	1,03	1,03
July 7, 2001	1,03	1,03

This query provides both the first and last available end-of-day exchange rates for the specified period. The former is provided through the MDX *OpeningPeriod* function; the latter is provided through a cube aggregate function. You could further extend the query to identify the difference or variance in exchange rates across the opening and closing of the period.

Chapter 9 Quick Reference

To	Do this
Retrieve the periods-to-date for any specified period	Use the *PeriodsToDate* function to return a set of sibling members from the same level as a given member, starting with the first sibling and ending with the given member, as constrained by a specified level of a calendar hierarchy. For example, the following query retrieves the periods-to-date over the calendar year for each of the Month members along the *ROWS* axis to calculate a year-to-date total for reseller sales:

```
WITH
MEMBER [Measures].[Year to Date Reseller Sales] AS
    Aggregate(
        PeriodsToDate(
            [Date].[Calendar].[Calendar Year],
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Year to Date Reseller Sales])
        } ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```

<table>
<tr><th>To</th><th>Do this</th></tr>
<tr><td>Retrieve the periods-to-date for a year</td><td>Use the Ytd function to return a set of sibling members from the same level as a given member, starting with the first sibling and ending with the given member, as constrained by the Year level of a calendar hierarchy. For example, the following query retrieves the year-to-date periods for each of the Month members along the ROWS axis to calculate a year-to-date total for reseller sales:
<pre>
WITH
MEMBER [Measures].[Year to Date Reseller Sales] AS
    Aggregate(
        Ytd([Date].[Calendar].CurrentMember),
        ([Measures].[Reseller Sales Amount])
)
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Year to Date Reseller Sales])
        } ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
</pre>
For quarter-to-date, month-to-date, and week-to-date calculations, use the Qtd, Mtd, and Wtd functions, respectively, in a similar manner.</td></tr>
<tr><td>Retrieve a number of prior periods</td><td>Use the LastPeriods function to retrieve a set of members up to and including a specified member. For example, the following query retrieves the last three months for each of the Month members along the ROWS axis to calculate a rolling three-month average for reseller sales:
<pre>
WITH
MEMBER [Measures].[Three Month Avg Reseller Sales Amount] AS
    Avg(
        LastPeriods(
            3,
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
)
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Three Month Avg Reseller Sales Amount])
        } ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
</pre></td></tr>
</table>

To	Do this
Retrieve a parallel member	Use the *ParallelPeriod* function to identify a member from a prior period in the same relative position as a specified member. For example, the following query retrieves prior period reseller sales for each of the Month members along the *ROWS* axis:

```
WITH
MEMBER [Measures].[Prior Period Reseller Sales Amount] AS
    (
        ParallelPeriod(
            [Date].[Calendar].[Calendar Year],
            1,
            [Date].[Calendar].CurrentMember
            ),
        [Measures].[Reseller Sales Amount]
        )
    ,FORMAT="Currency"
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Prior Period Reseller Sales Amount])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[Calendar Year].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
         } ON ROWS
FROM [Step-by-Step]
```

To	Do this
Retrieve the opening period or closing period	Use the *OpeningPeriod* or *ClosingPeriod* functions, respectively. For example, the following query employs the *OpeningPeriod* function to retrieve the exchange rate for the first day in each period:

```
WITH
MEMBER [Measures].[First Child Rate] AS
    (
        OpeningPeriod(
            [Date].[Calendar].[Date],
            [Date].[Calendar].CurrentMember
            ),
             [Measures].[End of Day Rate]
            )
       ,FORMAT="Standard"
SELECT
    {
        ([Measures].[First Child Rate]),
        ([Measures].[End of Day Rate])
        } ON COLUMNS,
    {[Date].[Calendar].Members} ON ROWS
FROM [Step-by-Step]
WHERE ([Destination Currency].[Destination Currency].[Euro])
```

Index

Symbols and Numbers

-- (double dash), inline comments, 93
- (except) operator, 91
- (exception) operator, 143
- (negative) operator, 91
- (subtract) operator, 91
! (exclamation point) character, 93
& (ampersand) character, 41
* (crossjoin) operator, 77, 92
* (multiply) operator, 92
/ (divide) operator, 92
/**/ (paired forward slashes and asterisks), multiline comments, 93
// (double forward slash), inline comments, 93
^ (power) operator, 92
+ (add) operator, 92
+ (positive) operator, 92
+ (string concatenation) operator, 92
+ (union) operator, 92, 143
< (less than) operator, 92
<= (less than or equal to) operator, 92
<> (not equal to) operator, 92
= (equal to) operator, 92
> (greater than) operator, 92
>= (greater than or equal to) operator, 92

A

ABS function, 126
access rights, 273–74. *See also* security, dynamic
account name, user, 274
accumulating total, 213–19
ACTION_TYPE property, 46
add (+) operator, 92
additive aggregations, 8
additive measures, 248–49
administrative rights, 274
Adventure Works Cycles, 4
AFTER flag, 191, 200–1
Aggregate function, 158, 178
  calculated members, 158–61
  cube-scoped calculated members, 249–50
  reports, 345–46, 352
  Sum function vs., 161
  year-to-date sales calculation, 214–17
aggregation, 157, 178–80
  accumulating total, 213–14
  averages calculation, 161–69
  minimum and maximum value identification, 170–72
  multidimensional data warehouse, 8
  relational data warehouse, 8
  Reporting Services vs. Analysis Services, 345–46
  reports, 352
  summation, 157–61
  tuples, counting in sets, 172–77
alias
  axis, 72
  named sets, 266
ALL flag, 143
  Except function, 155
  Generate function, 147
  Intersect function, 156
All folder, 33
All member, 9
  Members function, 74–75
  omitted references, 47
  partial tuples, 49–50
  user-hierarchy translation, 51
All members
  dataset parameters, 338
  report aggregation, 346
All Products member
  calculated, 112–15
  division-by-zero error, 185–86, 195–96
  ranking, 189
  security restrictions, 296
AllMembers function, 96–97, 118
  Query Designer, 328
allowed sets, 285–86
  designing, 287–90
  implementation, 290–95, 300–1
ampersand (&) character, 41
an axes, 38–39
Analysis Services
  administrative rights, 274
  aggregation, 157, 352. *See also* aggregation
  aggregation, reports, 345–46
  auto-exists, 79–83
  browsing objects within an instance, 18–19
  connecting to, 15–17, 35, 316–20
  cubes, 3. *See also* cubes
  data storage and retrieval, 11
  dynamic named sets, 271
  expressions, 11, 91–94. *See also* expressions
  functions, 115, 339
  functions, non-native, 92–93
  MDX script, 239–46
  multidimensional data warehouse, 8–9
  Null value, 93–94
  OpeningPeriod and ClosingPeriod functions, 231–33
  operators supported, 91–92
  overlapping references, 58
  partial tuples. *See* partial tuples
  Query Designer. *See* Query Designer
  reports. *See* reports
  security, 273. *See also* security, dynamic
  sets, 61–62. *See also* sets
  special member functions, 115
  string conversion functions, 339
  time dimension, 211–12. *See also* time dimension
  user-hierarchies, 9. *See also* user-hierarchies
Ancestor function, 190, 209
  ParallelPeriod function, 227–28
  percent contribution calculation, 192–96
ancestors, 189–92
  percent contribution to, calculating, 192–96
Ancestors function, 190
AND operator, 92
  Filter function, 137
ASC flag, 124, 128, 153
Ascendants expression, 305
Ascendants function, 190, 209
  product percent contribution calculation, 196–98
ascending sorts, 188
assemblies, functions, 93
ASSOCIATED_MEASURE_GROUP property, 256–57, 272

attribute-hierarchies, 9–10, 27–29, 101
  axis translation, 39
  calculated member creation, 94–96
  calendar, 28
  calendar year. *See* calendar year attribute-hierarchy
  category. *See* category attribute-hierarchy
  color, 29, 82–86
  country, 29, 62–67
  defined, 14
  employee, 87–88
  fact table restrictions, 286
  fiscal year, 41–42, 49–50, 55–58
  foreign key restrictions, 286
  measure, 76
  measures, 39, 42, 48, 76
  month, 86–88
  navigation. *See* navigation
  product, 29, 289–90, 294–97
  product categories, 28, 183–89
  security restrictions, 285–302, 309
  set limiting, auto-exists, 79–83
  set limiting, Exists function, 83–88
  set sorting, 126–30
  size, 29
  standard, restricting, 286–97
  subcategory. *See* subcategory attribute-hierarchy
  user, 287–95
  user-hierarchy translation, 51–55
attributes, 5–7
  defined, 14
  multidimensional data warehouse, 8–10
  relational data warehouses, 8
  security restrictions, 273–74
  time-based, 212–13. *See also* time dimension
authentication, 274
authority, user, 274
  removing, 284
auto-exists
  EXISTING keyword, 115–17
  hierarchy restrictions, 285
  set limiting, 79–83
averages, calculating, 161–69
  Avg function, 162–65, 179
  monthly, 166–68
  order-level, 167–68
  quarterly, 162–65
  rolling, 220–22
  transaction-level, 166–68, 264–65
  with expressions, 165–69
Avg function, 162–65, 179
axes. *See also* COLUMNS axis; ROWS axis
  cube spaces, 37–39
  MDX queries, 320–21
  SELECT statement, 72
  sets, 68

B

BACK_COLOR property, 46
BackgroundColor property, 351
BASC flag, 124, 128, 153
BDESC flag, 124, 128, 153
BEFORE flag, 191
BEFORE_AND_AFTER flag, 191, 201–2
Begins With operator, 339
best performers, extracting, 132–34
Boolean operations, 94
Boolean values, 91–92
BottomCount function, 131–34, 154
braces, 61, 88, 285
brackets, square, 40
break-hierarchy sorts, 126–30
Business Intelligence Development Studio (BIDS), 239–40
  cube-scoped complex calculated member implementation, 259–62
  cube-scoped named set creation, 268–70
  MDX script of Step-by-Step database, 240–46
  NON_EMPTY_BEHAVIOR property, 264–65
  report creation, 312–16, 351
Business Intelligence landscape, 3–4

C

CALCULATE statement, 239–40
calculated members, 94–98, 117–19
  BIDS creation, 244–46
  contextual conflicts, 103–9
  creating, 94–97
  cube-scoped. *See* cube-scoped calculated members
  dataset addition, 351
  declaring, 98
  dynamic, construction of, 98–102
  expressions, 94–98
  formatting, 108–9
  infinite recursion, 103–5
  query addition, 324–26
  query-scoped, 98, 247–48, 257–59
  session-scoped, 98, 266
  solve order, 105–8
  summation, 158–61
Calculations tab, 242–43
calendar attribute-hierarchy, 28
Calendar folder, 27–28
calendar hierarchies, 211–12
calendar user-hierarchy, 62–67
calendar year
  reseller sales average, 162–65
  year-to-date reseller sales calculation, 214–17
calendar year attribute-hierarchy, 41–42
  calculated member creation, 94–96
  overlapping references, 55–58
  partial tuples, 49–50
calendar year user-hierarchy, 52–55
calendar-to-fiscal year user-hierarchy, 42, 52–55
  overlapping references, 55–58
Cameron, Scott, 3, 273
CAPTION member property, 110
category attribute-hierarchy, 29, 41–42, 62–67
  calculated member creation, 94–96
  Members function, 74–76
  set limiting, auto-exists, 81–83
  set limiting, exists function, 83–86
  set sorting, 127–30
category members, set sorting, 127–30
CELL PROPERTIES keyword, 45
  Query Designer, 328
CELL_ORDINAL property, 44, 46
  accessing, 44–46
cells, 37, 43–46
  accessing partial tuples, 47–50
  accessing with overlapping-reference tuples, 55–58
  accessing with user-hierarchy tuples, 51–55
  contextual conflicts, 103–9
  empty, eliminating, 86–89
  permissions, 302–3
  properties, 44–46, 60
  security restrictions, 273–74, 302–8, 310
  security restrictions, implementing, 305–7
  security restrictions, logical expression design, 303–5
  security restrictions, testing, 307–8
Children function, 182, 200, 208
children member relationship, 181–82
CHILDREN_CARDINALITY member property, 110

ClosingPeriods function, 231–33, 235
code pane
  highlighted text, 34
  Query Editor, 22
Codeplex Web site, 4
color attribute-hierarchy, 29
  set limiting, auto exists, 82–83
  set limiting, Exists function, 83–86
COLUMNS axis
  calculated member creation, 94–96
  empty sets, 284–85
  MDX queries, 320–21
  SELECT staement, 42–72
  set limiting, auto-exists, 79–83
combinations, sets, 142–46
Command Prompt window, UserName function evaluation, 281–84
commas, 61, 88
comments, expressions, 93
comparison operators, 91–92
Computer Management Console, local user account creation, 274–77
ComputerName, 281
conforming dimensions, 6–7
connection string, 319–20
CONSTRAINED flag, 339
Contains operator, 339
context
  conflicts, expressions, 103–9
  named sets and, 271
coordinates, 37–39
Count Aggregate function, 158
Count function, 116, 172–76, 180
country attribute-hierarchy, 29
  sets, 62–67
Cousin function, 190, 192
  ParallelPeriod function, 227–28
cousin member relationship, 189–92
CREATE keyword, 271
CREATE MEMBER statement, 239–40
  cube-based calculated members, 250–53
  cube-scoped calculated members, 247–48
CREATE SET statement, 239–40, 266–70
cross-fact analysis, 6–7
crossjoin (*) operator, 77, 92
Crossjoin function, 77–79, 89
crossjoins
  auto-exists set limiting, 79–83
  count aggregation, 176
  Exists function, 83–86
  Other-Form Exists function, 86–88
Cube Designer, 240, 242
CUBE_NAME member property, 110
cubes, 8–10, 239, 272
  access to, 273–74. *See also* security, dynamic
  calculated member construction, 247–65. *See also* calculated members
  defined, 14
  dimensions, 9. *See also* dimensions
  MDX script, 239–46
  named set assembly, 266–71
  space, 15, 39–41
  space, SELECT statement definition, 68–69
  Step-By-Step, 25–29
  summation, 157–58
  tuples, 39–41
cube-scoped calculated members, 98, 247, 272
  basic, constructing, 247–55
  complex, construction, 256–65
  deploying, 253–54
  formatting, 272
  hiding, 272
  implementing, 250–53, 259–62
  key performance indicators, 255–56
  properties, 256–57
  properties, setting, 256–65
  verifying, 255, 262–64
cube-scoped named sets, 268–72
currency number format, 109
current measures, 175
current members, 108–15
current value determination, 212–13
CurrentMember function, 108–15, 119

D

data access rights, 273–74. *See also* security, dynamic
data analysis, 3–4
data flattening, 320–21
data presentation, 340–51. *See also* reports
Data Source parameter, 319–20
data sources, 316. *See also* embedded data sources
  connection string, 319–20
Data Warehouse layer, 3–5
  relational data warehouse, 8
data warehouses
  latency, 213
  multidimensional, 8–10
  relational, 8
database roles, 273–74
  creation, 277–81
DataMember function, 115
dataset design, 320–29, 351–52
  calculated member addition, 324–26
  hidden datasets, 336
  parameter addition, 329–39
  query assembly, 321–24
  query modification, 326–29
Date dimension, 27–28, 41–42
  partial tuples, 49–50
  sets, 62–67
date formats, 109
Date function, 213
date values, 91–92
DateAdd function, 213
DateDiff function, 213
DatePart function, 213
DefaultMember function, 115
denied sets, 285–86
DESC flag, 124, 128, 153
descendants, 189–92
  set assembly, 199–202
Descendants function, 190, 210
  flags, 190–91
  set assembly, 199–202
dicing, 5, 9
  defined, 14
DIMENSION PROPERTIES keyword, 328
dimension tables, 8
DIMENSION_UNIQUE_NAME member property, 111
dimensional model, 5–7
  defined, 14
  implementation, 7–10
dimensionality, shared, 6–7, 61–67
dimensions, 5–7
  conforming, 6–7
  defined, 14
  relational data warehouse, 8
DISPLAY_FOLDER property, 256–57, 267
Distinct Count Aggregate function, 158
Distinct function
  set building, 89
  sets, 67
DistinctCount function, 176–77
divide (/) operator, 92
division-by-zero errors, 185–86, 195–96, 207–8
double coordinate, 37–38
drill-down operation, 9
dynamic expressions, 98–102
DYNAMIC keyword, 271–72
dynamic named sets, 271
dynamic security. *See* security, dynamic

E

embedded data sources, 316
connection string, 319–20
creation, 316–19, 351
employee attribute-hierarchy, 87–88
empty cells, 86–89
empty sets, 284–85
enterprise reporting. *See* reports
Equal operator, 338–39
equal to (=) operator, 92
errors
contextual, expressions, 103–9
division-by-zero, 185–86, 195–96, 207–8
EXCLUDEEMPTY flag, 175
highlighted text, code pane, 34
infinite recursion, 103–5, 118
NON_EMPTY_BEHAVIOR property, 257
overlapping references, 58
shared dimensionality, 65–67
solve order, 105–8, 118
time dimension, 227–28
ETL (Extraction, transformation, and loading) layer, 4
except (-) operator, 91
Except function, 142–43, 155
set construction, 143–46
exception (-) operator, 143
exclamation point character, 93
EXCLUDEEMPTY flag, 172, 174–75, 180
DistinctCount function, 176–77
Execute button, 22, 35
EXISTING keyword, 116–17, 119
averages calculation, 167
Generate function, 149
Item function, 137
named sets, 271
Exists function, 89, 200
attribute-hierarchy restrictions, 286
Other-Form, 86–88
set limiting, 83–88
Expression element
CREATE MEMBER statement, 247–48
CREATE SET statement, 266–67
cube-based calculated members, 250–53
expressions, 11, 91–94
allowed and denied sets, 285–86
averages calculation, 165–69
building, 109–17
calculated members, 94–98
comments, 93
connection string as, 319–20
contextual conflict resolution, 103–9
dynamic, 98–102
KPI objects, 255–56
query-scoped complex calculated member, 257–59
Reporting Services, 316
security, 273
set sorting by, 124–26
sets, 115–17
extended relatives, accessing, 189–203
Extract function, 156
set assembly, 151–53
Extraction, transformation, and loading (ETL) layer, 4

F

fact measures, 5
defined, 14
fact tables, 8
attribute-hierarchy restrictions, 286
averages calculation, 165
facts, 5–7
attributes. *See* attributes
cross-fact analysis, 6
defined, 14
dicing, 5
slicing, 5
Filer function, 137
Filter expression, 305
Filter function, 133, 154
Extract function, 152
set membership limitation, 138–40
filtering
dataset parameters, 329–31
operators, 339
sets, 137–42, 154
FirstChild function, 232–33
FirstNonEmpty function, 232–33
Fiscal folder, 27–28
fiscal year attribute-hierarchy, 41–42
overlapping references, 55–58
partial tuples, 49–50
fiscal year user-hierarchy, 52–55
fixed number format, 109
flags. *See also* specific flags
Descendants function, 190–91
string conversion functions, 339
FONT_FLAGS property, 46
FONT_NAME property, 46
FONT_SIZE property, 46
For Each loop, 147
FORE_COLOR property, 46
foreign keys
attribute-hierarchy restrictions, 286
fact tables, 8
Format function, 348–50
FORMAT property, 46
FORMAT_STRING property, 45–46, 256–57, 272
calculated members, 108–9
FORMATTED_VALUE property, 44, 46, 352
accessing, 44–46
calculated members, 108–9
table formatting, 347–48
formatting, table, 346–50
FREEZE statement, 246
FROM clause, 13
dataset filtering, 331
SELECT statement, 69–72
FROM keyword, 32
functions. *See also* specific functions
adding to queries, 32–33
aggregation, within measures vs. MDX, 157
assemblies, 93
building sets, 72–79
navigation, 182, 190, 204, 227–28
non-native, 92–93
string conversion, 339
VBA. *See* VBA functions
XTD, 217–18

G

general date format, 109
general number format, 109
Generate function, 156
aggregation, 177
set assembly, 147–50
Geography dimension, 29
sets, 62–67
goal, KPI object, 255
greater than (>) operator, 92
greater than or equal to (>=) operator, 92
growth over prior period calculation, 223–26

H

Head function, 134
hidden. *See also* VISIBLE property
cube-scoped calculated members, 272
datasets, 336
measures, 286

- HIDDEN keyword, 267
- Hierarachize function, 129–30
- hierarchical sorts, 124
  - breaking constraints, 126–30
- hierarchies. *See also* attribute-hierarchies; user-hierarchies
  - calculated member declaration, 98
  - calendar, 211–12
  - current member, 108–15
  - extended relatives, 189–92
  - navigating, 181–210. *See also* navigation
  - parent-child, restricting, 297–302
  - ranking members of, 187–89
  - security restrictions, 285–302
  - shared, 61
  - shared hierarchality, 65–67
  - Step-by-Step cube, 28–29
  - time-based, 192
- Hierarchize function, 153, 197
- HIERARCHY_UNIQUE_NAME member property, 111
- horizontal navigation, 203–4
- Hyperion Essbase, 11

I

- IIF function, 109–10, 113
  - dataset filtering, 333
- immediate relatives, accessing, 181–89
- In operator, 339
- inception-to-date calculation, 218–19
- INCLUDEEMPTY flag, 172, 180
  - Distinct Count function, 176–77
- infinite recursion, 103–5, 118
- Initial Catalog parameter, 319–20
- inline comments, 93
- inline-IF function, 110
- instances, browsing objects within, 18–19
- Internet Orders measure group, 27
- Internet Sales measure group, 27
- Intersect function, 142–43, 156
  - set construction, 143–46
- IS operator, 91–92, 94
- IS_DATAMEMBER member property, 111
- IsAncestor function, 202–3
- ISEmpty function, 94
- IsLeaf function, 202–3
- IsSibling function, 202–3
- Item function, 135–37

K

- KEY member property, 111
- key performance indicators (KPIs), 255–56
- key-based references, 41
- keys, member, 50–51
- key-value, 41
- KEYx member property, 111
- KPIs (key performance indicators), 255–56

L

- Lag function, 203–5, 210
  - ParallelPeriod function, 227–28
- LANGUAGE property, 46
- language, MDX, 11–13
- LastChild function, 232–33
- LastNonEmpty function, 232–33
- LastPeriods function, 220, 234
  - rolling average calculation, 220–22
- latency, 213
- Lead function, 203–4
- LEAF flag, 190–91
- leaf level, 9
  - Members function, 74–75
  - set limiting, 81–83
- leaf-level members, 202–3
  - user-hierarchy translation, 54
- LEAVES flag, 191
- less than (<) operator, 92
- less than or equal to (<+) operator, 92
- Level argument, 214
- LEVEL_NUMBER member property, 111
- LEVEL_UNIQUE_NAME member property, 111
- levels, navigation within, 203–8
- local user account creation, 274–77
- LOCALHOST keyword, 16
- logical comparison (IS) operator, 92
- logical conjunction (AND) operator, 92
- logical disjunction (OR) operator, 92
- logical exclusion (XOR) operator, 92
- logical expression design, 303–5
- logical inverse (NOT) operator, 92
- logical operators, 91–92
- long date format, 109

M

- main toolbar
  - Management Studio, 17
  - Query Editor, 21
- Management Studio. *See* SQL Server Management Studio
- Max function, 170, 180
- maximum, subcategory, sales difference determination, 170–72
- MDX (multidimensional expressions), 14
  - aggregation functions, 157. *See also* aggregation
  - basics, 3–14
  - language, 11–13
  - MdxUser. *See* MdxUser account
  - navigation functions, 181–82, 190, 204. *See also* navigation
  - operators, 339
  - queries, 320–21. *See also* queries; queries, reports
  - Query Editor. *See* Query Editor
  - reports. *See* reports
  - script, 239–46
  - scripting statements, 246
  - SELECT statement. *See* SELECT statement
  - statements, 11–13. *See also* statements
  - time-based functions. *See* time dimension
- MdxReport, project creation, 312–16
- MdxUser account
  - attribute-hierarchy restrictions, 287–97
  - cell-level restrictions, 302–8
  - creation, 274–77
  - database role creation, 277–81
  - parent-child hierarchy restrictions, 297–302
  - removing, 308
  - UserName function, 281–84
- measure attribute-hierarchy, 76
- measure groups, 8
  - attribute-hierarchy restriction, 286–87
  - defined, 14
- measures, 8
  - adding to queries, 31
  - additive, 248–49
  - axis translation, 39
  - calculated, 272
  - current, 175
  - defined, 14
  - functions, 157
  - hiding, 286
  - summation, top five, 158–61
- measures attribute-hierarchy, 39, 42
  - default member, 48
- Measures folder, 26–27

MeasuresGroupMeasures function
  set building, 76
Member argument, 214
  LastPeriods function, 220
Member property, 9
  defined, 14
MEMBER substatement, 117
member.Children function, 182
member.FirstChild function, 182
member.FirstSibling function, 182
member.LastChild function, 182
member.LastSibling function, 182
member.Parent function, 182
member.Siblings function, 182
Member_Caption function, 110
MEMBER_CAPTION member property, 111
MEMBER_NAME member property, 111
MEMBER_UNIQUE_NAME member property, 111
MEMBER_VALUE member property, 111
MemberName element, 247–48
  cube-based calculated members, 250–53
members, 9, 28
  adding to queries, 29–31
  calculated. *See* calculated members
  current, 108–15
  defined, 14
  extended relatives, 189–92
  hierarchies. *See* attribute-hierarchies; hierarchies; user-hierarchies
  immediate relationships, 181–82
  investigating, 202–3
  keys, 30, 50–51
  leaf-level, 54, 202–3
  level identifiers, 75–76
  limiting, 79–88
  names, 30, 40–41, 50–51
  percent contribution to ancestors calculation, 192–96
  percent-of-parent calculation, 183–86
  properties, 109–11
  query-scoped. *See* query-scoped calculated members
  references, 40–41, 50–51, 60
  references, conflicts, 55–58
  references, omitted, 47
  references, shortcuts, 59
  sibling comparison, 186
  special, 115
Members folder, 28
Members function
  allowed set creation, 290
  allowed set implementation, 294–95
  set building, 73–76, 89
MemberValue function, 110
menu bar, Management Studio, 17
metadata
  measures, 262–63
  properties, 257
  Reporting Services, 320–21
  reporting, query assembly, 321–24
metadata pane, 22
  adding functions to queries, 32–33
  adding measures to queries, 31
  adding members to queries, 29–31
  Step-by-Step cube exploration, 25–29
Microsoft SQL Server. *See* SQL Server
Microsoft SQL Server 2008 Analysis Services Step by Step (Cameron), 3, 273
Microsoft SQL Server 2008 Reporting Services Step by Step (Misner), 311, 320, 351
Min function, 170, 180
Misner, Stacia, 311, 320, 351
month attribute-hierarchy, 86–88
monthly averages calculation, 166–68
monthly sales calculation, 204–8
month-to-date sales calculations, 217–18
Mtd function, 217–18
multidimensional data warehouse, 8–10
multidimensional expressions (MDX). *See* MDX (multidimensional expressions)
multiline comments, 93
multiplication operations, 94
multiply (*) operator, 92

N

Name function, 110, 137
NAME member property, 111
name, user, 284
  parent-child restrictions, 298–300
name-based references, 41
named sets, 266–67, 272
  assembly, 266–71
  context and, 271
  cube-scoped, 268–71
  query-scoped, 267–68
  static vs. dynamic, 271
navigation, 181, 208–10
  extended relative access, 189–203
  functions, 182, 190, 204, 227–28. *See also* specific functions
  horizontal, 203–4
  immediate relative access, 181–89
  position-based, 212
  within levels, 203–8
n-dimensional space, 11, 37–39
n-dimensionality, 68
negative (-) operator, 91
NextMember function, 203–4
NON EMPTY keyword, 86–89
  Query Designer, 327
  set filtering, 141–42
NON_EMPTY_BEHAVIOR property, 256–57, 264–65, 272
non-additive aggregations, 8
NonEmpty function, 133, 141–42, 155
non-native functions, 92–93
Not Equal operator, 338
not equal to (<>) operator, 92
Not In operator, 339
NOT operator, 92
Null values
  division-by-zero errors, 185–86, 207–8
  expressions, 93–94
  inception-to-date calculation, 219
  multiplication operations, 94
  NON_EMPTY_BEHAVIOR property, 264–65
Number of Products, 115–17
numeric formats, 108–9
numeric operators, 91–92
numeric values, 91–92

O

object comparison (IS) operator, 91
Object Explorer, 15, 17
  hiding, 20
object identifiers, 40–41
  member reference shortcuts, 59
objects
  browsing within and instance, 18–19
  KPI, 255–56
octuple coordinate, 39
ON keyword, 72, 92, 130–31
OpeningPeriod function, 231–33, 235
operations
  order of, 92, 105–8, 118
operators. *See also* specific operators

comparison, 91–92
crossjoin, 77
filter, 338–39
logical, 91–92, 137
numeric, 91–92
set, 91–92
string, 91–92
supported, Analysis Services, 91–92
OR operator, 92
Filter function, 137
Order function, 123–24, 153
Head and Tail functions, 134
set sorting by expression, 124–26
set sorting while breaking hierarchical constraints, 126–30
order of operations, 92, 105–8, 118
ordered sets, 123–31
range operator, 130–31
order-level averages, 167–68
Other-Form, Exists function, 86–88
overlapping references, 55–58

P

parallel periods, 222–23
ParallelPeriod function, 223, 235
growth over prior period calculation, 223–26
navigation functions, 227–28
parameters
connection string, 319–20
dataset design, 329–39, 351
modifying, 333–38
multi-valued, 338
parameter-value pairs, 319–20
Parent function, 112, 182, 208
percent-of-parent calculation, 183–86
Parent Member calculated member, 185
Parent Member Name calculated member, 112–15
parent member relationship, 181–82
PARENT_COUNT member property, 111
PARENT_LEVEL member property, 111
PARENT_UNIQUE_NAME member property, 111
parent-child hierarchies, security restrictions, 297–302, 309
parentheses, 52
Parse command, 22, 35
parsing, 274
statements, 35
partial tuples, 47–51
calculated member creation, 96
calculated members, 98–102
infinite recursion, 103–5
percent change in sales calculation, 204–8
percent contribution across lineage calculation, 196–98
percent contribution to ancestors calculation, 192–96
Percent of Parent calculated member, 113–15
percent-of-category calculation, 192–96
percent-of-parent calculation, 183–86
period-over-period analysis, 222–28
periods, growth over calculation, 223–26
periods, parallel, 222–23
periods-to-date calculations
reseller sales, 216–17
simplifying, 217–18
PeriodsToDate function, 213–14, 233
inception-to-date calculation, 218–19
year-to-date and prior period year-to-date sales calculation, 229–31
year-to-date reseller sales calculation, 214–17
permissions, cell-level, 302–3
points, 68
accessing in cube space, 42–43
positive (+) operator, 92
POST flag, 130, 153
power (^) operator, 92
Presentation layer, 4
PrevMember function, 203–4, 210
change in sales calculation, 204–8
prior period year-to-date sales calculation, 229–31
product attribute-hierarchy, 29
allowed set creation, 289–90
allowed set implementation, 294–95
testing restrictions, 295–97
product categories attribute-hierarchy, 28
percent-of-parent calculation, 183–86
Rank function, 187–89
Product Categories calculated member, 112–15
product categories user-hierarchy
parameter conversion, 336
sets, 64–67
Product dimension, 28–29, 41–42
calculated member creation, 96
Members function, 74–76
set limiting, auto-exists, 80–83
sets, 62–67
product percent contribution across lineage calculation, 196–98
products, top five, calculations, 268–71
properties, 9. *See also* specific properties
calculated member, 256–57
cell, 44–46, 60
member, 109–11
Properties function
intrinsic member properties, 110–11

Q

Qtd function, 217–18
quadruple coordinate, 38–39
quarterly average, 162–65
quarter-to-date calculations, 217–18
quarter-to-date reseller sales calculation, 216–17
queries
adding functions to, 32–33
adding measures to, 31
adding members to, 29–31
building, 23–25
complex, 29–35
executing, 33–34
nested, 142
restoring, 34–35
saving, 34–35
queries, reports, 320–21
assembling, 321–24
calculated member addition, 324–26
modification, 326–29
parameter addition, 329–33
Query Designer, 321
calculated member addition, 324–26, 351
filter operators, 338–39
parameter addition, 352
parameter modification, 336–38
query assembly, 321–24
query modification, 326–29
Query Editor, 15
complex query building, 29–35
Functions tab, 22
layout, 21–22
message pane, 22
metadata pane, 22
Metadata tab, 22
opening, 19–22, 35
query building, 23–25
results pane, 22

SQL Server Management Studio, 15–19
Step-by-Step cube, 25–29
target database, changing, 35
query-scoped calculated members, 98, 247–48
assembling, 257–59
query-scoped named sets, 267–68
quintuple coordinate, 39

R

Range (Exclusive) operator, 339
Range (Inclusive) operator, 339
Rank function, 186–89
ranking
All Products member, 189
by sales, 187–89
siblings, 188–89
Read Contingent permission, 302–3
Read permission, 302–3
Read/Write permission, 302–3
references
conflicts, 55–58
key-based, 41
member, 40–41, 50–51, 60
member, conflicts, 55–58
member, omitted, 47
member, shortcuts, 59
name-based
overlapping, 55–58
relational data warehouse, 8
relatives, accessing
extended, 189–203
immediate, 181–89
Report Designer, 333–38
Report Server, project creation, 312–16
Reporting Services, 311, 351–52
aggregation, 345–46
Analysis Services connection, 316–20
connection string, 319–20
data presentation, 340–51
dataset design, 320–29
expressions, 316
finishing touches, 350–51
parameter addition, 329–39
parameter modification, 333–38
report totals, adding, 344–46
Select All option, 338
table assembly, 340–43
table formatting, 346–50
reports, 311–16, 351–52
Analysis Services connection, 316–20
data presentation, 340–51
dataset design, 320–29
finishing touches, 350–51
parameter addition, 329–39
project creation, Report Server, 312–16
table formatting, 352
Reseller Order Count
Aggregate function, 158–61
report, 311. *See also* reports
report, aggregation, 345–46
report, calculated member addition, 324–26
report, query assembly, 324
report, table assembly, 340–43
report, table formatting, 346–50
Reseller Orders measure group, 26
Reseller Sales Amount
accumulating total calculation, 213–14
Aggregate function, 158–61
average calculation, 162–65
calendar year average, 162–65
Count function, 172–76
cube-scoped calculated members, 248–49
cube-scoped complex calculated membes, 259–62
expression calculation, 165–69
growth over prior period calculation, 223–26
maximum and difference calculation, 170–72
NON_EMPTY_BEHAVIOR property, 264–65
partial tuples, 47–50
percent change in between months calculation, 204–8
product percent contribution across lineage calculation, 196–98
products meeting or exceeding, determining, 172–76
quarter-to-date, 216–17
query-scoped calculated member assembly, 258–59
ranking by sales, 187–89
ranking siblings, 188–89
report, 311. *See also* reports
report, aggregation, 345–46
report, calculated member addition, 324
report, query assembly, 324
report, table assembly, 340–43
report, table formatting, 346–50
rolling averages calculation, 220–22
standard deviation calculation, 261
variance, 262
year-to-date and prior period year-to-date sales calculation, 229–31
year-to-date sales calculation, 214–17
Reseller Sales measure group, 26
Reseller Transaction Count, Aggregate function, 158–61
roles, database, 273–74
creation, 277–81
rolling averages calculation, 220–22
ROWS axis
calculated member creation, 94–96
Crossjoin function, 77–79
empty sets, 284–85
MDX queries, 320–21
Other-Form Exists function, 86–88
parameter modification, 337–38
SELECT statement, 68–72
set limiting, auto-exists, 79–83
set limiting, Exists function, 83–86
RunAs command, 282

S

sales. *See also* Reseller Sales Amount
averages. *See* averages, calculating
growth over prior period, 223–26
monthly, 204–8
month-to-date, 217–18
percent change in, 204–8
period-to-date, 216–18
prior period year-to-date, 229–31
quarter-to-date, 216–18
ranking by, 216–17
year-to-date, 214–17, 229–31
Sales Target measure group, 88
SAP Netweaver BI, 11
SAS OLAP Server, 11
SCOPE statement, 246
Script Organizer, 244
security, dynamic, 273–74, 308–10
attribute-hierarchy restrictions, 285–302
cell-level restrictions, 302–8
database role creation, 277–81
empty sets, 284–85
local user account creation, testing, 274–77
UserName function evaluation, 281–84
Select All option, parameters, 338
SELECT clause, 13
SELECT keyword, 31
SELECT statement, 11–13, 43
axes, 72

calculated member
declarartion, 98
dataset filtering, 331
MDX, 12–13
MEMBER substatement, 117
sets, 68–72
SQL, 12–13
sub-, 331
SELF flag, 190–91, 200
SELF_AND_AFTER flag, 191, 201, 210
SELF_AND_BEFORE flag, 191
SELF_BEFORE_AFTER flag, 191
semi-additive aggregations
relational data warehouse, 8
septuple coordinate, 39
session-scoped calculated members,
98, 266
set operators, 91–92
SetName element, 266–67
sets, 61–67
advanced construction, 147–53
allowed. *See also* allowed sets
allowed vs. denied, 285–86
assembly, 62–67
assembly, Generate function,
147–50
building, 72–79, 88–89, 123
building, Members function,
73–76
combining, 142–46
construction, Union, Intersect,
and Except function, 143–46
context and, 271
descendant assembly, 199–202
duplicate removal, 67
duplicate tuple removal, 176–77
empty, 284–85
expressions, 115–17. *See also*
expressions
Extract function assembly,
151–53
filtering, 137–42, 154
Generate function assembly,
147–50
Generate function
exploration, 177
limiting, 62–88, 131–34
limiting, Exists function, 83–88
limiting, Filter function, 138–40
merging, 155. *See also* Union
function
named. *See* named sets
ordered, 123–31
ordered, range operator, 130–31
SELECT statement, 68–72.
See also SELECT statement
single- and multi-member,
WHERE clause, 333
sorting, 153–56
sorting by expression, 124–26
sorting while breaking
hierarchical constraints, 126–30
sorting with Hierarchize function,
129–30
tuple retrieval, 131–37
sextuple coordinate, 39
shared data sources, 316
shared dimensionality, 6–7, 61
set assembly, 62–67
shared hierarchality, 61
set assembly, 65–67
shortcuts
Command Prompt, 281
member references, 59
sibling member relationship,
181–82
Siblings function, 182, 209
ranking siblings, 188–89
siblings, ranking, 187–89
size attribute-hierarchy, 29
slicer axis, 69
slicing, 5, 9
defined, 14
snowflake schema, 8
Solution Explorer, 242
solve order, 105–8, 118
KPI objects, 256
MDX scripts, 240
SOLVE_ORDER property, 105–8, 118
sort order
by expression, sets, 124–26
set tuples, 123–24
sibling ranking, 188–89
Source layer, 4
special member functions, 115
SQL Server
Adventure Works Cycles, 4
Analysis Service. *See* Analysis
Service
Database Engine Services, 8, 11
SELECT statement. *See* SELECT
statement
SQL Server Books Online, 211,
303, 349
SQL Server Management Studio,
15–19
allowed set implementation,
290–95, 300–1
cell-level restrictions, 305–7
database role creation, 277–81
document workspace, 17
launching, 282
layout, 17
Object explorer. *See* Object
Explorer
Query Editor. *See* Query Editor
saving and restoring queries,
34–35
testing cell-level restrictions,
307–8
testing restrictions, 301–2
testing set restrictions,
295–97
square brackets, 40
standard attribute-hierarchies,
restricting, 286–97
standard deviation calculation,
168–69, 257–58
NON_EMPTY_BEHAVIOR
property, 264–65
star schema, 8
stars, 6
statements, 11–13. *See also* specific
statements
executing, 35
parsing, 35
scripting, 246
subcube, 142
static named sets, 271
status, KPI object, 255
Stdev function, 168–69
SELECT FROM, 49–50
Step-by-Step cube, 25–29
attribute hierarchies, 101
Chapter 3 cube, 41–42
hierarchies, 28–29
MDX script, 240–46
metadata pane exploration,
25–29
query building within, 23–25
set limiting, 88. *See also* set
limiting
sets, 62–67
string concatenation (+)
operator, 92
string conversion functions, 339
string operations, 94
string operators, 91–92
string values, 91–92
StrToMember function, 339
StrToSet function, 288, 339
StrToTuple function, 339
subcategory attribute-hierarchy, 29,
41–42
set limiting, auto-exists, 80–83
set-limiting, Exists function,
83–86
subcategory members, set sorting,
127–30
subcategory sales difference
determination, 170–72
sub-SELECT statement, 331
subtract (–) operator, 91
Sum function, 157–58, 178
Aggregate function vs., 161
summation, 157–61, 178
calculated members, 158–61

T

table
  assembly, reports, 340–43
  formatting, reports, 346–50, 352
Tail function, 134
testing
  cell-level restrictions, 307–8
  set restrictions, 295–97, 301–2
  user account creation for, 274–77
text wrapping, 347
THIS statement, 246
time dimension, 211–12, 233–35
  accumulating total, 213–19
  cautions, 227–28
  combining time-based metrics, 229–33
  current value determination, 212–13
  growth over prior period calculation, 223–26
  inception-to-date calculation, 218–19
  OpeningPeriod and ClosingPeriod functions, 231–33
  period-over-period analysis, 222–28
  quarter-to-date reseller sales calculation, 216–17
  rolling averages calculation, 220–22
  year-to-date and prior period year-to-date sales calculation, 229–31
  year-to-date reseller sales calculation, 214–17
Time function, 213
time-based metrics, 229–33
tooltips, 33
TopCount function, 131–34, 154
  named sets, 271
transaction-level averages
  calculating, 166–68
  NON_EMPTY_BEHAVIOR property, 264–65
transaction-level minimums and maximums, 170
transaction-level standard deviation calculation, 169, 257–58
translation, user-hierarchy, 51–55
trend, KPI object, 256
triple coordinate, 38
tuples, 11, 37
  as values, 98
  cells, 43–46
  counting, 172–77, 180
  cube space, 39–41
  data access, 41–43
  DistinctCount function, 176–77
  duplicate removal, 176–77
  empty, excluding, 155
  member reference shortcuts, 59
  n-dimensional space, 37–39
  partial, 47–51, 96, 98–105
  point access in cube space, 42–43
  Rank function, 186
  retrieving, sets, 131–37
  sets. *See* sets
  sort order, 123–24
  user-hierarchies, 51–59
Type property value, 217–18
TYPED flag, 110

U

union (+) operator, 92, 143
Union function, 142–43, 155
  set construction, 143–46
UniqueName function, 110
UnknownMember function, 115
UPDATETABLE property, 46
user account creation, local, 274–77
user attribute-hierarchy
  allowed set implementation, 290–95
  allowed sets creation, 287–90
user authentication, 274
User dimension, 286–87, 309
  allowed set implementation, 290–95
  allowed sets creation, 287–90
User Product Relationship measure group, 290
User property, 309
  parent-child restrictions, 299–300
user rights, 273–74.
  See also security, dynamic
user-hierarchies, 9–10
  calendar, 62–67, 211–12
  calendar year, 52–55
  calendar-to-fiscal year, 42, 55–58
  fiscal year, 52–55
  navigation. *See* navigation
  product categories, 64–67, 336
  Step-By-Step cube, 27–29
  tuples, 51–59
user-hierarchy
  defined, 14
  translation, 51–55
UserName function, 274, 309
  cell-level restrictions, 303
  evaluation, 281–84
  parent-child restrictions, 299–300

V

VALUE property, 44, 46
  accessing, 44–46
  table formatting, 347–48
values
  aggregating. *See* aggregation
  available, parameter, 335–36
  averaging, 179
  Boolean, 91–92
  date, 91–92
  historical, 222–23
  KPI object, 255
  minimum and maximum, identifying, 163–72, 180
  Null. *See* Null values
  numeric, 91–92
  rolling averages calculation, 220–22
  string, 91–92
  tuples as, 98
  types of, 61–92
VBA functions, 93, 126
  date math, 213
  parsing, 274
  time, 213
Venn diagrams, 142–43
verification
  cube-based calculated members, 255
  cube-scoped calculated members, 262–64
vertical navigation, 203
VISIBLE property, 256–57
  cube-scoped calculated members, 272
  measure groups, 286
Visual Studio, BIDS hosting, 241

W

week-to-date calculations, 217–18
WHERE clause, 13
  dataset filtering, 331
  Nonempty function, 142
  SELECT statement, 69–72
  set limiting, 82–83

set limiting, Exists function, 83–86
single- and multi-member sets, 333
WHERE keyword, 30
Windows authentication, 274
WITH clause
calculated member declaration, 98
MEMBER substatement, 117
worst performers, extracting, 132–34
Wtd function, 217–18

X

x-axis, 37–39
x-coordinate, 37–39
XML for Analysis (XMLA), 11
XOR operator, 92
Filter function, 137
XTD functions, 217–18

Y

y-axis, 37–39
y-coordinate, 37–39
year-to-date sales calculation, 214–17, 229–31
year-to-date sales calculations, 217–18
Ytd function, 217–18, 234

Z

z-axis, 38–39

Bryan C. Smith

Bryan is a manager of specialized services with Hitachi Consulting's Microsoft Database Technologies team. As a member of this team, he designs and implements business intelligence solutions for clients in a variety of industries using the products in the Microsoft SQL Server suite. Bryan has degrees from Texas A&M and Duke Universities, holds a number of Microsoft certifications, and has more than 10 years of experience developing solutions supporting data analysis. Bryan lives in the Dallas area with his (amazing) wife, Haruka, and their two (equally amazing) children, Aki and Umi.

C. Ryan Clay

C. Ryan Clay is a senior architect with Hitachi Consulting, specializing in business intelligence, data management, portal and collaboration, and SAP integration/interoperability solutions employing Microsoft technologies. Ryan has implemented Microsoft Business Intelligence solutions using Analysis Services and MDX for a variety of Fortune 500 clients in the retail, construction, finance, and consumer goods industries. Ryan holds degrees in computer science as well as a number of Microsoft certifications and is active in the Microsoft community through speaking engagements and presentations at regional and national events. He lives in the Dallas area with his wife and daughter.

Hitachi Consulting

As the global consulting company of Hitachi Ltd. (NYSE: HIT), Hitachi Consulting is a recognized leader in delivering proven business and IT solutions to Global 2000 companies across many industries. We leverage decades of business process, vertical industry, and leading-edge technology experience to understand each company's unique business needs. From business strategy development through application deployment, our consultants are committed to helping clients quickly realize measurable business value and achieve sustainable return on investment. For more information, visit *www.hitachiconsulting.com*. Hitachi Consulting – Inspiring your next success®.