

MDX Sets

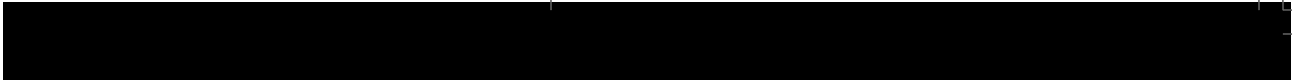
In this chapter, you'll learn MDX query terminology and how to

- Use the MDX Sample application to enter queries.
- Use set functions.
- Create sets by using member constants.
- Create calculated members that aggregate sets of values.

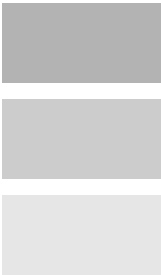
1. Start Analysis Manager. Right-click the server, and click Restore Database.
2. Navigate to the folder containing the sample files for this book, select Chapter 7, and click Open. The archived Chapter 7 OLAP database is identical to the one used in Chapter 6, "MDX Values." Click Restore, and then close the Restore Database Progress dialog box.
3. Open the Chapter 7 HTML file included with the sample files for this book. This HTML file contains a Microsoft Office PivotTable list that retrieves values from the Sales cube of the Chapter 7 OLAP database.

Have you ever noticed that no matter how many times you travel to a location as a passenger, the route seems entirely different the first time you have to navigate—especially if you're alone. The Microsoft Excel PivotTable report and the Office PivotTable list features create multidimensional expressions (MDX) statements to retrieve values from a cube, but creating a report yourself can give you a tremendous appreciation for how Microsoft SQL Server 2000 Analysis Services works.

An MDX query statement is different from an MDX expression. An expression is a formula that calculates a single value. A query is a command that populates a report with many values from a cube. The Office PivotTable list, the Excel PivotTable report, and the browser included in Analysis Manager all generate MDX queries to retrieve values from a cube. You can use tools like these to create reports without writing any MDX statements of your own. Unless you're a developer creating a custom report generator, you'll probably have little occasion to write MDX query statements. So, why should you learn how to create an MDX query statement? Because the most interesting MDX expressions involve creating a single result based on a large set of values from a cube. Those



Chapter 7



Chapter Objectives

Estimated time: 2 hours

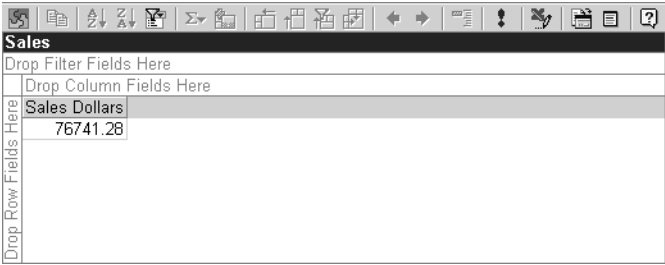
Start the lesson

expressions, in effect, create a subquery behind the scenes. Learning how to create an MDX query will enable you to understand clearly what the subquery is doing when you create complex MDX expressions. In the first part of this chapter, you'll learn how to create explicit queries. In the latter part of the chapter, you'll use the concepts learned in creating the reports to create calculated members based on large sets of values.

Preparing to Create MDX Queries

The purpose of an MDX query is to extract values from an OLAP cube into a report. As explained in Chapter 1, "A Data Analysis Foundation," a cube has dimensions (up to 64, if you count Measures as a dimension). A report does not have dimensions; it has axes (typically, a row axis, a column axis, and a filter axis). An axis can include labels from more than one dimension. A cube contains all possible values for all members of all levels of all dimensions. A report contains only selected values from selected levels of selected dimensions. An MDX query statement consists of the instructions for extracting a report from a cube.

An MDX query statement uses several new terms. Seeing the terms in a familiar context will make them easier to learn and understand. Browsers such as the Office PivotTable list use MDX query statements to populate a report. You can use the Office PivotTable list to learn MDX terminology. The initial report in the Chapter 7 HTML file shows only a single cell showing the total value for the Sales Dollars measure. The grand total is a single value from the cube. To retrieve this value, Analysis Services used the default member for each dimension.



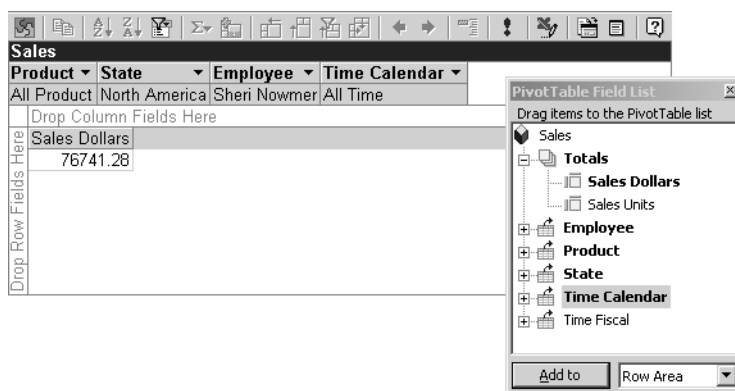
Sales		
Drop Filter Fields Here		
Drop Column Fields Here		
Drop Row Fields Here	Sales Dollars	
		76741.28

1. In the Web browser, click the Field List toolbar button. Drag the Product, State, Employee, and Time Calendar dimensions from the PivotTable Field List window to the filter area of the report. Each dimension displays the default member in the filter area, and the total value never changes as you add the dimensions. (Since the Time Fiscal and Time Calendar dimensions are really hierarchies of the same dimension, you can include only one in the report at a time. The report still uses the default member of the unused dimension.)



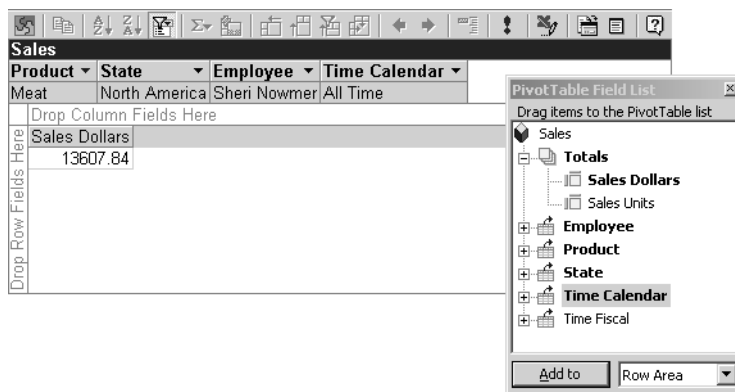
Use a PivotTable list to understand MDX terms





The default member allows a query to ignore some of the dimensions in a cube. If a dimension does not appear on the filter axis, it always uses the default member. If a dimension does appear on the filter axis, it can either use the default member or a different member. Putting a dimension on the filter area of a report overrides the default member for that dimension.

- Click the arrow next to the Product dimension list, select the Meat check box, and click OK to show only the total for meat.



If you use nothing but the filter area, you create a report that displays only a single value, which is rarely useful. The row and column axes of a report allow you to display more than one member from a dimension. More than one member from a single dimension is called a *set*.

- Drag the State dimension to the row axis. The three countries—plus the total—appear. The row axis now displays a set that contains four *positions*. Each position in the set corresponds to a member from the State dimension.



Sales			
Product	Employee	Time Calendar	
Meat	Sheri Nowmer	All Time	
Drop Column Fields Here			
Country	Region	State	Sales Dollars
Canada			1678.57
Mexico			3281.11
USA			8648.16
Grand Total			13607.84

- Click the Country level label, and click the Expand toolbar button.

Sales			
Product	Employee	Time Calendar	
Meat	Sheri Nowmer	All Time	
Drop Column Fields Here			
Country	Region	State	Sales Dollars
Canada	Canada		1678.57
	Total		1678.57
Mexico	Mexico		3281.11
	Total		3281.11
USA	North West		6233.27
	South West		2414.89
	Total		8648.16
Grand Total			13607.84

The regions appear. The row axis now displays a set that contains eight positions. Even though the labels are split into two columns—to show the levels of the dimension—each position in the set still contains only a single member from the State dimension.

- Drag the Sales Units measure to the data area. The column axis now displays a set that contains two positions, each of which corresponds to a single member from the Measures dimension.

Sales				
Product ▾	Employee ▾	Time Calendar ▾		
Meat	Sheri Nowmer	All Time		
Drop Column Fields Here				
Country ▾	Region	State	Sales Dollars	Sales Units
Canada	Canada		1678.57	785
	Total		1678.57	785
Mexico	Mexico		3281.11	1470
	Total		3281.11	1470
USA	North West		6233.27	2795
	South West		2414.89	1106
	Total		8648.16	3901
Grand Total			13607.84	6156

- For each cell value you see in the report, you can determine a complete tuple that includes one member from each dimension in the cube. For the cell in the Sales Dollars column with the value 6233.27, the Product dimension contributes Meat, the Employee Dimension contributes Sheri Nowmer, the Time Calendar dimension contributes All Time, and the State dimension contributes North West.

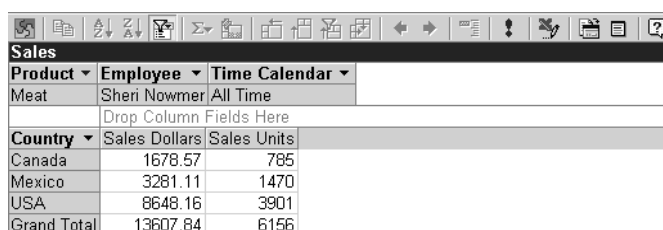
Every value in an MDX report requires a single member from each dimension. The filter axis can include only a single member for any one



dimension. The row and column axes can include only sets, which include multiple values from a single dimension.

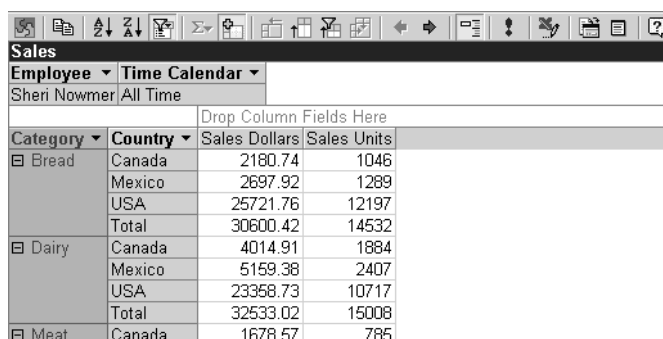
It's not unusual for a cube to contain at least six or seven dimensions. Not counting the filter axis, a report based on an OLAP cube typically contains only two axes—columns and rows. Often, it's useful to combine multiple dimensions from the cube onto a single axis of the report.

7. In the PivotTable List document, drag the Region and State level headings away from the row area, leaving only the countries and Grand Total on the axis.



Country	Sales Dollars	Sales Units
Canada	1678.57	785
Mexico	3281.11	1470
USA	8648.16	3901
Grand Total	13607.84	6156

8. Drag the Product level from the filter area to the row area, to the left of the State level. Drag the Subcategory and Product Name labels away from the report. Select the Category label, and click the Expand toolbar button. The report now shows both categories and countries on the row axis. The row axis still contains a set—a set with 13 positions. But each position within the set now contains a two-member tuple—a coordinate from more than one dimension.



Category	Country	Sales Dollars	Sales Units
Bread	Canada	2180.74	1046
	Mexico	2697.92	1289
	USA	25721.76	12197
	Total	30600.42	14532
Dairy	Canada	4014.91	1884
	Mexico	5159.38	2407
	USA	23358.73	10717
	Total	32533.02	15008
Meat	Canada	1678.57	785

Note Technically, a set always contains tuples. Even when there is only one dimension on an axis, the set consists of single-member tuples.

9. Drag the Time Calendar dimension from the filter area to the row area, to the left of the Category label. Drag the Calendar Quarter and Calendar Month labels away from the report. Then select the Calendar Year label, and click Expand. The report now shows years, categories, and countries on the row axis. The row axis still contains a set—now a set with 17 positions. But each position in the set now contains a three-member tuple.



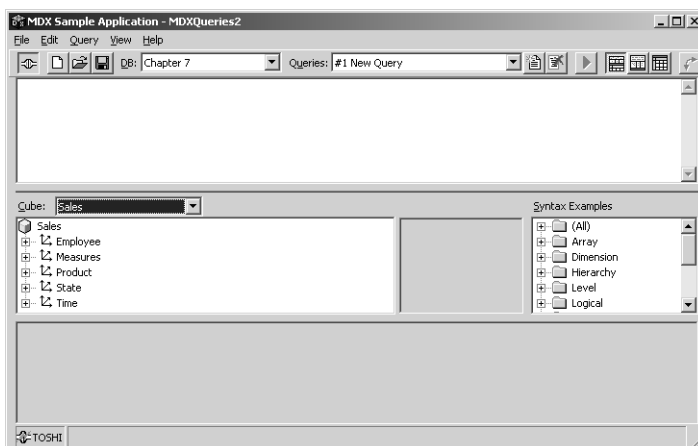
Sales				
Employee ▾				
Sheri Nowmer				
			Drop Column Fields Here	
Calendar Year ▾	Category ▾	Country ▾	Sales Dollars	Sales Units
1997	Bread	USA	11523.31	5559
		Total	11523.31	5559
	Total		11523.31	5559
1998	Bread	Canada	2180.74	1046
		Mexico	2697.92	1289
		USA	14198.45	6638
		Total	19077.11	8973
	Dairy	Canada	4014.91	1884
		Mexico	5159.38	2407
		USA	23358.73	10717

A tuple is a way to specify coordinates from more than one dimension on a single axis. The filter axis can contain only a single tuple. Row and column axes can each contain a set of tuples. The bottom line remains: for each value in the report, there's a single member from each dimension that contributed to the complete coordinate for that cell in the cube. The member for a dimension can come from any of the axes—and more than one dimension can be represented on a single axis. Any dimension not found on the column, row, or filter axis uses the default member.

Analysis Services comes with the MDX Sample application. The application allows you to type an MDX statement, run it, and see the results in a grid. The MDX Sample application includes a *metadata pane*, which allows you to browse the hierarchies and members of a cube, inserting dimension, level, and member names into the MDX expression. The application also comes with complete source files for the Microsoft Visual Basic project used to create it. If you're a programmer of Visual Basic, you can customize or add enhancements to the application. Some of the controls in the MDX Sample application are similar to those in the Calculated Member Builder. In this section, you'll take a few minutes to explore the interface.

1. Launch the MDX Sample application, which is on the Analysis Services menu along with Analysis Manager. Click OK to connect to the server. Click the New Query File toolbar button to create a new query file. A query file can contain multiple queries.
2. In the toolbar, select Chapter 7 as the database name. Then, in the Cube drop-down list box, select Sales as the cube name.

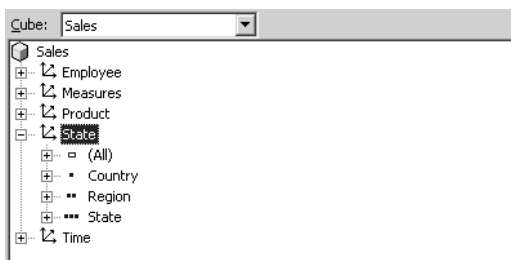
Use the MDX Sample application



The MDX Sample application window is divided into three vertical panes. The top pane is the *query pane*, where you build an MDX query statement. The middle pane is the *metadata pane*, where you inspect information in a selected cube. The bottom pane is the *results pane*, which shows the output of an MDX query. Across the top of the application are a menu bar and a toolbar.

Once you select a cube, the Metadata tree—the leftmost section of the metadata pane—displays the cube name as the top-level parent, plus the names of all the dimensions in the cube. The Metadata tree is equivalent to the Data tree in the Calculated Member Builder. The cube name has a cube icon to its left, and each dimension name has a triple-arrow icon to its left. Unlike the icons in Analysis Manager, the icons in the metadata view do not vary between different types of dimensions. The Metadata tree also displays members, but members often appear multiple times in the tree. Taking a close look at the metadata window can help you avoid confusion and get the most benefit from the MDX Sample application.

3. Expand the State dimension. You see one entry for each level in the dimension, including the All level. The All level has a small rectangular icon to the left of the level name. Every other level has a dot-cluster icon. The number of dots represents the level's depth in the hierarchy.

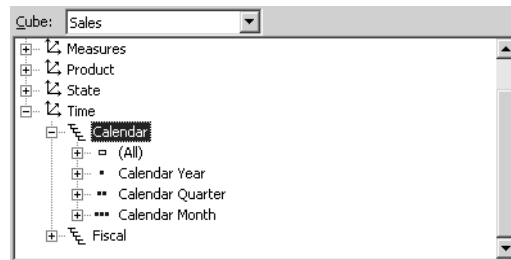




4. Expand the State level. You see one entry for each member in the level. Each member name has a lollipop icon to its left. (It's probably supposed to be a magnifying glass, but thinking of it as a lollipop is more vivid.)



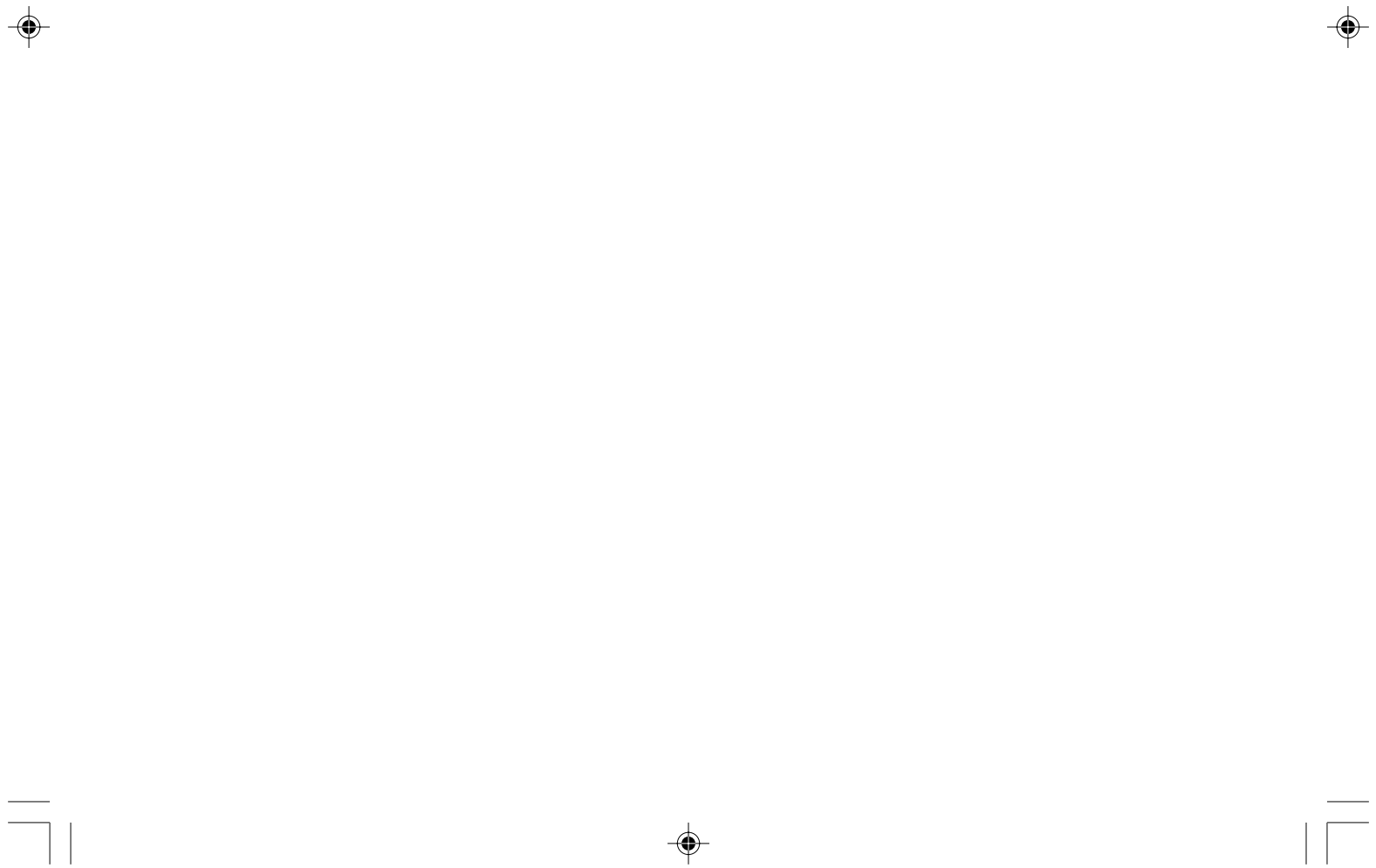
5. Collapse the State dimension, and then expand the Time dimension. You see two stair-step icons, one for the Calendar hierarchy and one for the Fiscal hierarchy. Then expand the Calendar hierarchy to see the level names.



Note Each item in the Metadata tree is a cube, a dimension, a hierarchy, a level, or a member. These are all entities that exist within an OLAP database. That contrasts with sets and tuples, which exist only in an MDX expression. In other words, there are no sets or tuples in an OLAP database, just cubes, dimensions, hierarchies, levels, and members.

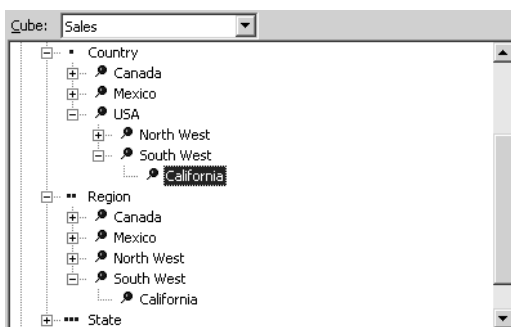
For your convenience, the Metadata tree usually adds each member node to multiple locations in the tree. Specifically, in addition to adding a member node below the level at which it belongs—as with the State level—the Metadata tree also adds the same member as a child of its parent member. Some people find this redundancy confusing at first, but once you understand how the Metadata tree works, it really is convenient.

6. In the State dimension, expand the Region level and the South West member. The California member appears below South West. California is the child—the only child—of the South West member. The California member appears both in the State level and also as a child of the South West member.





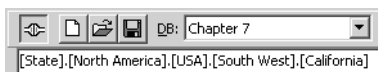
7. Expand the Country level, the USA member, and the South West member. You see the California member once again—this time as a “grandchild” of USA. The California member is still on the State level.



The highest member in an ancestry tree belongs to the level name just above it. For example, USA belongs to the Country level. For each degree of indentation, the members belong to a level one lower in the cube. For example, South West belongs to the Region level—even though the member node can appear under the Country level. Likewise, California belongs to the State level—even though the member node can appear under the Country or Region levels as well as the State level.

You can double-click any node to have the fully qualified name of the item appear in the query pane. You can also drag a node to the query pane, in case you didn’t put the insertion point in the proper place.

8. Double-click the California member. The full name of the member appears in the query pane.



In the Metadata tree, when you select either a level or a member that has children, the members appear in the list to the right of the metadata pane. You can double-click or drag members from the member list to add them to the query pane.



Double-clicking a leaf-level member in the tree view does not move the member into the query pane. You can, however, drag a leaf-level member into the query pane or select the level name and double-click a member from the member list.

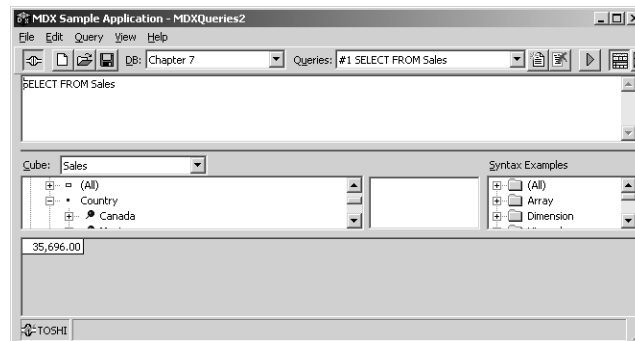


Creating Basic MDX Queries

You can create MDX queries to replicate the reports you create by using a browser such as the Office PivotTable list control.

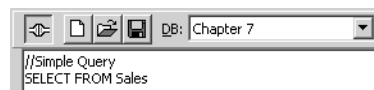
Start by creating the simplest possible query from the Sales cube.

1. Replace the contents of the query pane with **SELECT FROM Sales**. This is the simplest possible MDX query. You can now run this query.
2. Click the Run Query button on the toolbar. (If you don't see the results pane, click the Split View button to the right of the Run Query button; the Split View button displays the results pane below the metadata pane.) A single number appears in the report. The number in the report—35,696.00—is the total Sales Dollars for the entire cube. Because the query does not specify a member from any dimension, the query uses the default member from each dimension.



To add a comment to an MDX statement, you simply enter two adjacent slash characters (//), and MDX will ignore everything from that point to the end of the line.

3. Move the insertion point before the word **SELECT**. Type **//Simple Query**, and press Enter. Then click the Run Query button to run the query again. You don't see any difference—or an error message.



4. Click the Save Query File button on the toolbar, type **Queries** as the name of the file, and click Save. You can add multiple queries to this single file, but you can't save a file until you've entered something in the query pane.

Create a minimal MDX query

*You can also
press F5 to run
a query.*



This simple query retrieves a single value from the cube. You can create more interesting reports by adding sets of members to the column or row axis.

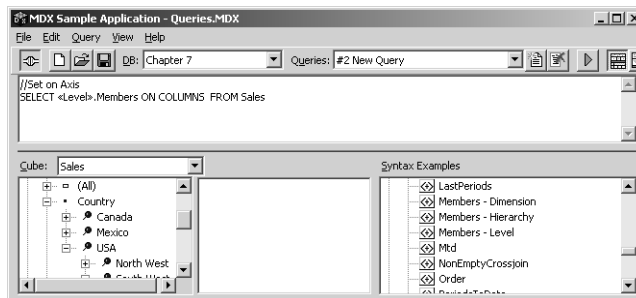
Note In addition to using two sequential slash characters (//) to comment the remainder of the line, you can also use two sequential hyphen characters (--) for the same purpose. If you want to create a comment that either is a partial line or spans multiple lines, you begin the comment with a slash followed by an asterisk (/*) and end it with an asterisk followed by a slash (*//).

To display more than one cell in the result grid, you must put a set of members on an axis. Before creating the new query, add a new query to the query file so that you can keep both queries intact.

1. Select the contents of the query pane, and press Ctrl+C to copy it. Click the New Query button, and press Ctrl+V. Change the comment to **//Set on Axis**.

To display more than one member on an axis, you use a *set* of members, precisely as in the browser. MDX has a number of functions that return sets. To the right of the member list, in the metadata pane, there's a tree labeled Syntax Examples. This list is the same as the Functions tree in the Calculated Member Builder. To get a set of members that you can put on an axis, select a function from the Set group.

2. Click immediately after the word *SELECT*, and type a space. In the Syntax Examples list, expand the Set group and double-click the *Members - Level* function. Then click after the *Members* function, and type a space and **ON COLUMNS**.

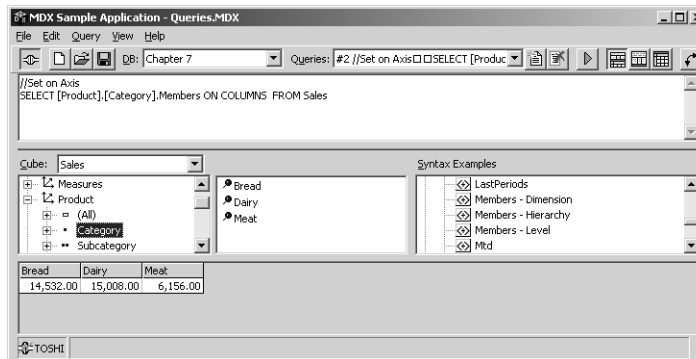


3. Click the word «*Level*». In the Metadata tree, double-click the Category level node in the Product dimension. The level name appears in the query.
4. Click the Run Query button to see the results of the query. The labels from the Category level appear as column headings in the report. The client application, however, can choose to put the “column” axis on either the columns or the rows.



Add a set to the column axis





5. Click the Pivot Results button on the toolbar. The Category labels now appear as row headings, even though the query still says ON COLUMNS.

Bread	14,532.00
Dairy	15,008.00
Meat	6,156.00

6. Click the Pivot Results button again to return the labels to the column headings, and save the query file.

Note The terms *COLUMNS* and *ROWS* are simply aliases for the true names of the axes, *Axis(0)*, and *Axis(1)*, respectively. The underlying names make it clearer to understand why a single-axis report must include a *COLUMNS* axis but not a *ROWS* axis. Technically, an MDX query can have up to 63 axes, with alias names for the first few. There are, however, essentially no situations in which it's necessary to use more than two heading axes for a report.

The tokens in the function templates help you know what kind of item to add to a specific location of the query. You must replace the «*Level*» token with the name of a level. In various contexts, you might also see tokens «*Member*», «*Set*», «*Tuple*», «*Dimension*», and «*Hierarchy*». «*Set*» and «*Tuple*» are MDX constructs that do not exist in the OLAP database, so you typically replace them with functions from the appropriate groups. «*Cube*», «*Dimension*», «*Hierarchy*», and «*Level*» all are OLAP objects, so you typically replace these tokens with items from the Metadata tree. A «*Member*» is the place where MDX and the cube meet. You can replace the «*Member*» token either with a constant member from the Metadata tree or with a function from the Member group.

You can also add a set to the row axis. In MDX, if you create a query that has only one axis, it must be the column axis. If you create two axes, one must be the column axis and one must be the row axis, although it doesn't matter in which order they appear within the query.



Add a set to a second axis



1. Copy the current contents of the query pane, click the New Query button to add a new query to the current file, and paste the previous query. Change the comment to **//Two axes**.
2. Click after the word *COLUMNS*, and type a comma and a space. You can now add another set.
3. In the Syntax Examples list, look in the Set group and double-click the *Children* function. Then click after the *Children* function, and type **ON ROWS**.



The token that precedes the *Children* function name is *«Member»*. A member has children.

4. Select the *«Member»* token, and in the Metadata tree, find and double-click the 1998 member. The formatted statement will look like this:

```
//Two axes
SELECT
    [Product].[Category].Members ON COLUMNS,
    [Time].[Calendar].[All Time].[1998].Children ON ROWS
FROM Sales
```

This function will now add a set of all the quarters of the year 1998 to the row axis.

5. Click the Run Query button to run the query.

	Bread	Dairy	Meat
Qtr1	2,246.00	1,732.00	
Qtr2	2,522.00	5,240.00	
Qtr3	3,150.00	6,056.00	4,646.00
Qtr4	1,055.00	1,980.00	1,510.00

On the column axis, the *Members* function displays only the members from the Category level. There are actually three versions of the *Members* function—one that returns the members of a level, one that returns the members from a dimension, and one that returns the members from a hierarchy. You can change the query to retrieve all the members of the Product dimension.

6. Delete the Category level name and the period following it, but leave the dimension name. The function will now return all the members of the Product dimension.
7. Click the Run Query button to see the results of the query. Click the Pivot Results button to move the products to the row axis.





	Qtr1	Qtr2	Qtr3	Qtr4
All Product	3,978.00	7,762.00	13,852.00	4,545.00
Bread	2,246.00	2,522.00	3,150.00	1,055.00
Bagels	272.00	275.00	327.00	110.00
Colony Bage	32.00	50.00	36.00	20.00
Fantastic Ba	73.00	53.00	60.00	13.00
Great Bagel	69.00	53.00	60.00	51.00
Modell Bagel	28.00	67.00	76.00	13.00
Sphinx Bage	70.00	52.00	95.00	13.00
Muffins	960.00	1,131.00	1,378.00	497.00
Colony Blue	20.00	51.00	65.00	16.00
Colony Cran	45.00	52.00	76.00	32.00
Colony Engli	70.00	30.00	67.00	27.00

8. Save the query file, and click the Pivot Results button to return the orientation to normal.

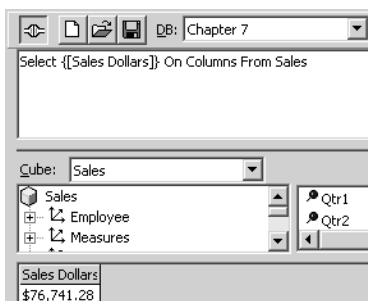
Both axes contain sets. Even though the products are not indented, all the members of the Product hierarchy are in the set.

Note One important rule for working with axes in an MDX query is that a dimension can be represented on only one of the three axes—column, row, and filter—because each axis contributes a member towards the complete tuple required to retrieve a value from the cube. If a single dimension is represented on more than one axis, the tuple will contain two members for a single dimension, which is invalid.

The Syntax Examples list shows three functions named *members*—one for levels, one for hierarchies, and one for dimensions. When a dimension has a dual hierarchy—as with the Time.Fiscal dimension—you must use the hierarchy version, not the dimension version, even if only one of the hierarchies exists in the current cube (as in the Basic Sales virtual cube created in Chapter 4, “Advanced Dimensions and Cubes”).

The only thing you can put on an axis is a set. A set function is one way to get a set to put on the axis. But sometimes, particularly with the Measures dimension, you want to be able to pick one or more individual members to put on an axis. You must be able to put the members into a set. To include members in a set, you must enclose the set in braces ({}). The braces aren’t necessary when you use set functions but are necessary when entering specific member names.

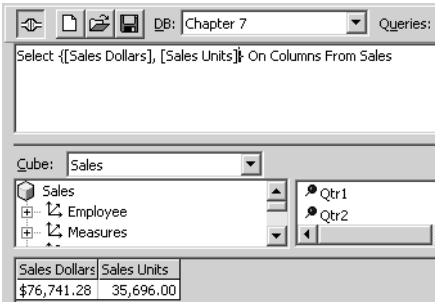
1. Click the New Query button. Type **Select {[Sales Dollars]} On Columns From Sales**. Then run the query.



Create a set by using explicit member names

When you create a set without using a set function, you need to enclose the set in braces, even if it contains only a single member.

- 2. After the *[Sales Dollars]* measure, type a comma, a space to enhance legibility, and then **[Sales Units]**. Then run the query.



The set on the column axis consists of two positions. Each position corresponds to a tuple, and each tuple contains a single member. If you want, you can add parentheses around each tuple, writing the set as *{([Sales Dollars]), ([Sales Units])}*, to show that each member is a separate tuple within the set. When you create a set by using a set function, you don't need to enclose the set in braces (but braces around a set are always acceptable). When you create a set by using explicit members, you do need to enclose the set in braces. If the first tuple in a set consists of a single member, each subsequent tuple in that set must be a member of the same dimension.

Note You can also manually create a set that includes tuples with multiple members. For example, the following is a legitimate set and could be placed on an axis in an MDX query: *{([Bread],[California]),([Meat],[Oregon])}*. It's unusual, however, to create a set with constant member names in multimember tuples. You're much more likely to use the *CrossJoin* function described in the following section to create a set with multiple members in each tuple.

- 3. Add a comment to the query, and save the query file.

The terminology of an MDX query centers on sets, tuples, and members. A set includes one or more tuples, and a tuple includes one or more members. Sets appear on the axes. Tuples appear either in a set (on a column or row axis), or in the *WHERE* clause to create the filter axis. Aside from the cube name in the *FROM* clause, a member is the only object from a cube that appears directly in an MDX query.

In "Use a PivotTable list to understand MDX terms" earlier in this chapter, you saw how the Office PivotTable list can put two dimensions onto the same axis, creating a set where each position in the set includes a tuple with more than one member. In an MDX query, to combine two sets from two dimensions into a single set that you can put onto a single axis, use the *CrossJoin* function.



Put two dimensions on one axis



1. Click the New Query button. Type **Select Country.Members on Columns, Category.Members On Rows From Sales**. Click the Run Query button.

	Canada	Mexico	USA
Bread	1,046.00	1,289.00	12,197.00
Dairy	1,884.00	2,407.00	10,717.00
Meat	785.00	1,470.00	3,901.00

This query puts the members of the country level (of the State dimension) on one axis, and the members of the category level (of the Product dimension) on the second axis. Each axis contains a set, and each set has a single member in each tuple. You can use the *CrossJoin* function to combine the two sets into one set, with two members in each tuple.

2. Change the query definition to **Select CrossJoin(Country.Members, Category.Members) On Columns From Sales**. Click Run Query to see the results.

Canada			Mexico			USA		
Bread	Dairy	Meat	Bread	Dairy	Meat	Bread	Dairy	Meat
1,046.00	1,884.00	785.00	1,289.00	2,407.00	1,470.00	12,197.00	10,717.00	3,901.00

The column axis contains a set with nine positions. Each position is a tuple that includes one member from each of two dimensions. The nine positions correspond to all the possible combinations of the three categories and the three countries.

3. After the Columns keyword, type a comma, and press Enter. Then type **CrossJoin([Calendar Year].Members, [Measures].Members) ON ROWS**, and click Run Query to run the query.

		Canada			Mexico			USA		
		Bread	Dairy	Meat	Bread	Dairy	Meat	Bread	Dairy	Meat
1997	Sales Units							5,559.00		
	Sales Dollars							\$11,523.31		
1998	Sales Units	1,046.00	1,884.00	785.00	1,289.00	2,407.00	1,470.00	6,638.00	10,717.00	3,901.00
	Sales Dollars	\$2,180.74	\$4,014.91	\$1,678.57	\$2,697.92	\$5,159.38	\$3,281.11	\$14,198.45	\$23,358.73	\$8,648.16

This query contains four dimensions from the Sales cube—two on each axis. Remember that you can put a dimension onto only one axis in a single query.

4. Add a comment, and save the query file.

The *CrossJoin* function is a powerful way to create reports that compare multiple dimensions. Because the function is so powerful, it's easy to create sets that are extremely large, particularly when combined with the *Members* function of multiple dimensions.



Creating Calculations by Using Aggregation Functions

In Chapter 6, you created calculated members in a cube by using arithmetic operators and discrete values. Now that you know how to use sets, you can create calculated members that use aggregation functions. First, however, create a simple calculated member within an MDX query statement. The calculated member created as part of an MDX statement works the same as one created in a cube, except that when stored in the cube the member is permanent and is available to all users of a cube. Sometimes it can be effective to create calculated members in a query statement as you develop them and then copy the expressions to a cube when you're satisfied with them.

To see how to create a dynamic calculated member in an MDX query, create an Average Price member.

1. In the MDX Sample application, add a new query that looks like the following:

```
SELECT
    [Measures].Members ON COLUMNS,
    [Product].[Subcategory].Members ON ROWS
FROM [Sales]
```

2. Click the Run Query button to see the results of the query, and save the query file. The query shows the two measures from the cube—Sales Units and Sales Dollars—as column headings.

	Sales Units	Sales Dollars
Bagels	1,541.00	\$3,552.28
Muffins	6,445.00	\$13,081.28
Sliced Bread	6,546.00	\$13,966.86
Cheese	7,493.00	\$17,709.11
Milk	4,151.00	\$7,259.42
Sour Cream	1,607.00	\$2,746.58
Yogurt	1,757.00	\$4,817.91
Deli Meats	1,977.00	\$4,337.12
Fresh Chicken	485.00	\$1,143.56
Frozen Chicken	1,338.00	\$3,214.98
Hamburger	1,015.00	\$2,175.56
Hot Dogs	1,341.00	\$2,736.62

For a calculated member, you must specify a parent. For the Measures dimension—which has only one level—use the dimension name as the parent. You can enter almost anything as the name of the member, but it must be enclosed in brackets following the same guidelines as any member name. The expression for a calculated member consists of instructions to generate a new value.

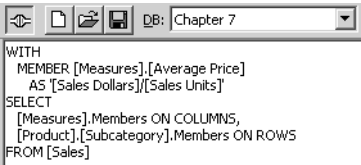


Create a calculated member as part of a query



3. Before the Select keyword, insert the following:

```
WITH MEMBER [Measures].[Average Price] AS '[Sales Dollars]/[Sales Units]'
```



```
WITH
  MEMBER [Measures].[Average Price]
  AS '[Sales Dollars]/[Sales Units]'
SELECT
  [Measures].Members ON COLUMNS,
  [Product].[Subcategory].Members ON ROWS
FROM [Sales]
```

In the main query, the set for the column axis is *[Measures].Members*, which would seem to include all measures, but see what happens when you run the query after defining a new calculated measure.

4. Click the Run Query button. Nothing changes. Calculated members are not retrieved by the *Members* function of a dimension or level.
5. Change the expression *[Measures].Members* to **[Measures].AllMembers**. The *AllMembers* function is like *Members*, except that it also includes calculated members. Click the Run Query button to see the calculated member.

	Sales Units	Sales Dollars	Average Price
Bagels	1,541.00	\$3,552.28	2.3051
Muffins	6,445.00	\$13,081.28	2.0296
Sliced Bread	6,546.00	\$13,966.86	2.1336
Cheese	7,493.00	\$17,709.11	2.3634
Milk	4,151.00	\$7,259.42	1.7488
Sour Cream	1,607.00	\$2,746.58	1.7091
Yogurt	1,757.00	\$4,817.91	2.7421
Deli Meats	1,977.00	\$4,337.12	2.1937
Fresh Chicken	485.00	\$1,143.56	2.3578
Frozen Chicken	1,338.00	\$3,214.98	2.4028
Hamburger	1,015.00	\$2,175.56	2.1434
Hot Dogs	1,341.00	\$2,736.62	2.0407

6. Add a comment to the query, and save the query file.

To dynamically create a calculated member, add a *MEMBER* clause at the beginning of the MDX query. You can add more than one *MEMBER* clause to a query. The word *WITH* must precede the first *MEMBER* clause. You do *not* put commas between successive *MEMBER* clauses.

The most frequent calculated member you'll create is a calculated measure—a member of the Measures dimension—but you can also create a calculated member of any dimension. When you create a calculated member of a nonmeasure dimension, you must specify where in the dimension hierarchy you want the new member to go. Suppose, for example, that you want to create a new member that shows the total Sales Units and Sales Dollars for all countries other than the United States—that is, for the total of Canada and Mexico.

Create a calculated member of a nonmeasure dimension

1. Create a new query, insert the following MDX query statement, and then run the query:

```
SELECT
    [Measures].Members ON COLUMNS,
    [State].[Country].Members ON ROWS
FROM Sales
```

	Sales Units	Sales Dollars
Canada	3,715.00	\$7,874.22
Mexico	5,166.00	\$11,138.41
USA	26,815.00	\$57,728.65

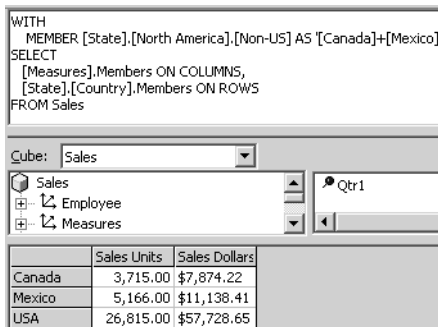
The query shows three countries on the row axis.

2. Before the SELECT keyword, insert the following clause:

```
WITH
    MEMBER [State].[North America].[Non-US] AS
        '[Canada]+[Mexico]'
```

Specifying *[State].[North America]* as the parent of the new member puts the new member at the Country level—the same level as Canada, Mexico, and USA. This is the level already specified in the query.

3. Run the query to see the results. The new member does not appear. As with the Measures dimension, the *Members* function does not retrieve calculated members of nonmeasure dimensions.



```
WITH
    MEMBER [State].[North America].[Non-US] AS '[Canada]+[Mexico]'
```

```
SELECT
    [Measures].Members ON COLUMNS,
    [State].[Country].Members ON ROWS
FROM Sales
```

Cube: Sales

Sales

- Employee
- Measures

Qtr1

	Sales Units	Sales Dollars
Canada	3,715.00	\$7,874.22
Mexico	5,166.00	\$11,138.41
USA	26,815.00	\$57,728.65

4. On the row axis, change the expression *[State].[Country].Members* to **[State].[Country].AllMembers**. Then run the query again.

The new member is now included on the row axis.

	Sales Units	Sales Dollars
Canada	3,715.00	\$7,874.22
Mexico	5,166.00	\$11,138.41
USA	26,815.00	\$57,728.65
Non-US	8,881.00	\$19,012.63

Think through the tuple that generates the 8,881.00 value in the result grid. The Product, Employee, and Time dimensions do not appear



in the MDX statement and thus use the default members. The column axis supplies the member from the Measures dimension (Sales Units), and the row axis supplies the member from the State dimension (Non-US). The Non-US member is a calculated member. That calculated member's expression requires two values. The expression overrides the State dimension member for each of those two values, inheriting the member for each of the other three dimensions from the cell in the result grid.

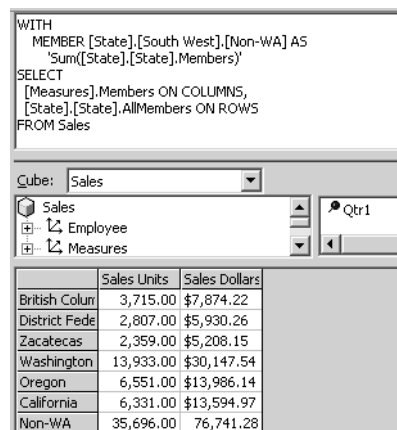
Creating an aggregate member by using a plus sign to add values works fine when you have only two or three values to aggregate, but it's cumbersome if you have multiple values. For example, suppose that you want to create a query that calculates the total for all states other than Washington.

5. Save the query file, copy the entire contents of the query pane, create a new query, and paste the copied query. Change the query to look like this (changed portions are in bold):

```
WITH
    MEMBER [State].[South West].[Non-WA] AS
        'Sum([State].[State].Members)'
SELECT
    [Measures].Members ON COLUMNS,
    [State].[State].AllMembers ON ROWS
FROM Sales
```

The parent is now *[South West]* rather than *[All Product]*. Making the South West region the parent puts the new member at the State level. Since South West is the last region, this puts the new member at the end of the list of states. For the time being, the expression includes all the states and doesn't exclude Washington.

6. Run the query. The states appear, with Non-WA at the bottom of the list. The values of the two Non-WA cells include the total of all the states.



The screenshot shows a query tool interface. At the top, the query text is displayed in a text area. Below it, there's a 'Cube:' dropdown set to 'Sales'. Underneath, there are expandable sections for 'Sales', 'Employee', and 'Measures'. To the right, there's a 'Qtr1' button. At the bottom, a results grid is shown with two columns: 'Sales Units' and 'Sales Dollars'. The grid lists several states and a 'Non-WA' member at the bottom.

	Sales Units	Sales Dollars
British Colum	3,715.00	\$7,874.22
District Fede	2,807.00	\$5,930.26
Zacatecas	2,359.00	\$5,208.15
Washington	13,933.00	\$30,147.54
Oregon	6,551.00	\$13,986.14
California	6,331.00	\$13,594.97
Non-WA	35,696.00	76,741.28

You still need to exclude Washington from the set of all the members of the State level. MDX has a set function that creates a new set by taking



all the members of one set *except* those that belong to a second set. The function is named, appropriately enough, *Except*.

7. In the member definition, change the expression 'Sum([State].[State].Members)' to '**Sum(Except([State].[State].Members, {[Washington]}))**', and run the query. While Washington still appears on the row axis—the row axis definition does not exclude the member—the totals for the Non-WA member no longer include Washington.

WITH

MEMBER [State].[South West].[Non-WA] AS

'Sum(Except([State].[State].Members, {[Washington]}))'

SELECT

[Measures].Members ON COLUMNS,

[State].[State].AllMembers ON ROWS

FROM Sales

Cube: Sales

Sales

Employee

Measures

Qtr1

	Sales Units	Sales Dollars
British Colum	3,715.00	\$7,874.22
District Fede	2,807.00	\$5,930.26
Zacatecas	2,359.00	\$5,208.15
Washington	13,933.00	\$30,147.54
Oregon	6,551.00	\$13,986.14
California	6,331.00	\$13,594.97
Non-WA	21,763.00	46,593.74

The Non-WA calculated member uses the same expression for each measure. Since both Sales Units and Sales Dollars have *Sum* as the member's aggregation function, using the *Sum* function in the Non-WA member works fine with either measure. Suppose, however, that the cube contained a measure such as Minimum Price, defined with the *Min* aggregation function. Using *Sum* to aggregate that measure would create an inappropriate value. MDX does have a *Min* function, but it would be extremely unwieldy to create a different Non-WA member for each measure. To resolve this problem, MDX has an *Aggregate* function. The *Aggregate* function simply aggregates each measure using the aggregation function defined for the measure. In general, when you create a calculated member on a nonmeasure dimension, you should use the *Aggregate* function rather than the *Sum* function.

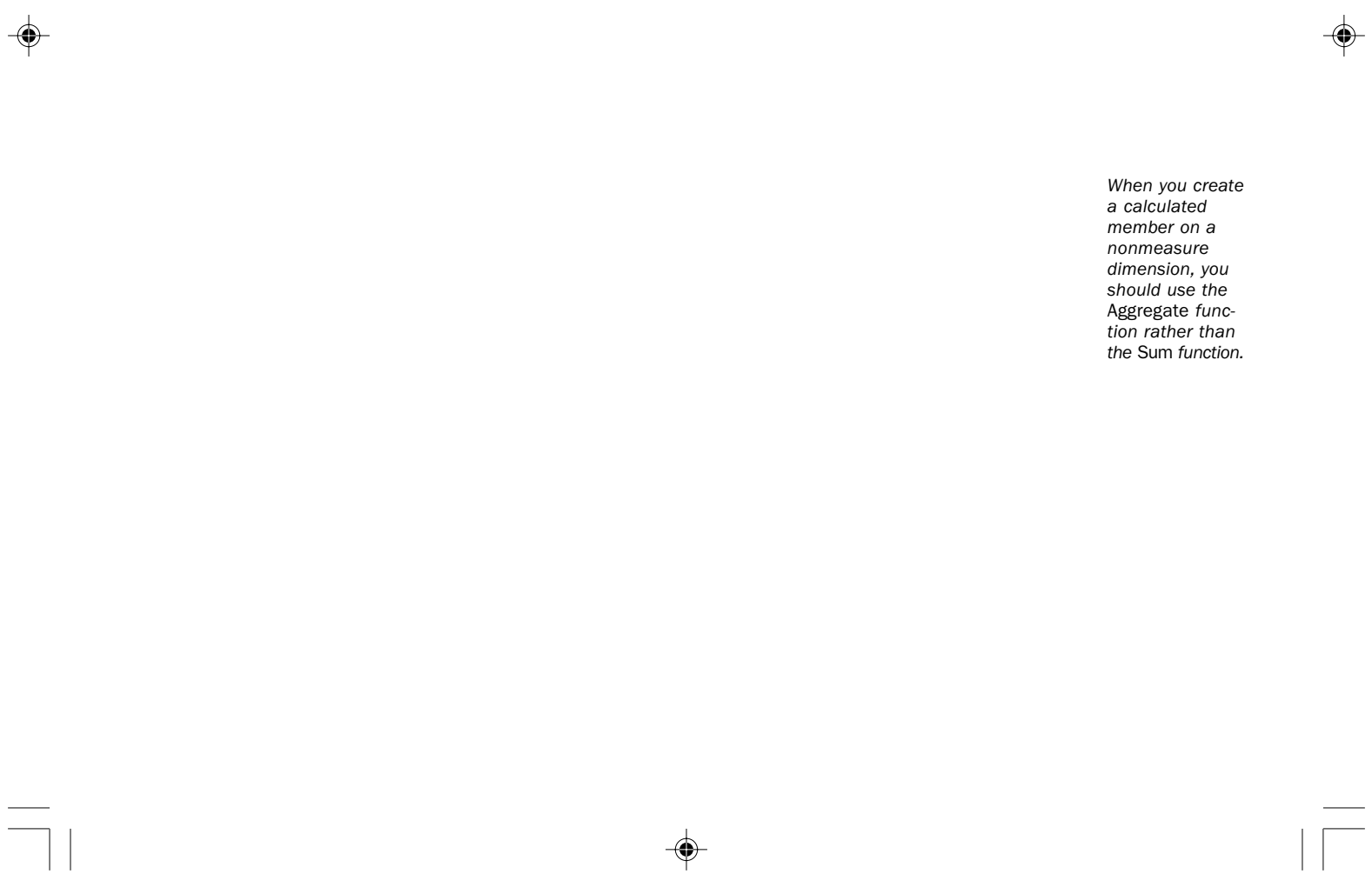
Note MDX has three related functions that manipulate two sets (Set1 and Set2) to produce a third set (Set3). The *Union* function puts into Set3 all the members in either Set1 or Set2. The *Intersect* function puts into Set3 only those members in both Set1 and Set2. The *Except* function puts into Set3 only the members from Set1 not in Set2. Each of the three functions strips duplicate values from Set3, unless you include the keyword *ALL* as an optional third argument.

8. In the Non-WA member definition, change the function name *Sum* to **Aggregate** and run the revised query. The values in the result grid should not change. Save the query file.



See “Specify the aggregation function for a measure” in Chapter 3, “Dimension and Cube Editors,” for details about assigning an aggregation function for a measure.

When you create a calculated member on a nonmeasure dimension, you should use the Aggregate function rather than the Sum function.



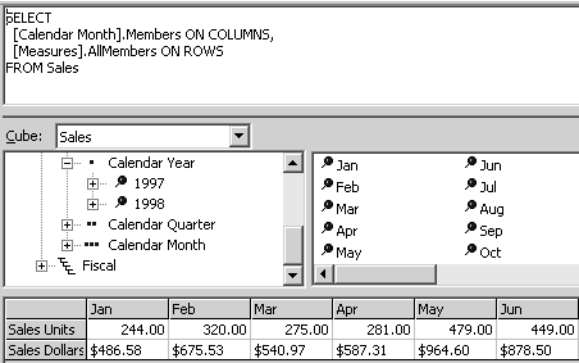
When you create a calculated member on a nonmeasure dimension, that member will, by definition, intersect with all the members of the Measures dimension. Each of those measures already has an aggregation function defined. Using the *Aggregate* function takes advantage of that previously defined aggregation function.

The *Sum* (or *Aggregate*) function is often used to create a calculated member on a nonmeasure dimension. You can also use the *Sum* function to create a new calculated measure. This is often done to create a cumulative total.

1. Create a new query, and insert the following MDX query statement:

```
SELECT
    [Calendar Month].Members ON COLUMNS,
    [Measures].AllMembers ON ROWS
FROM Sales
```

2. Run the query. This displays all the months as column headings and the two measures—Sales Dollars and Sales Units—as row headings. The *AllMembers* function allows you to see new calculated measures as you create them.



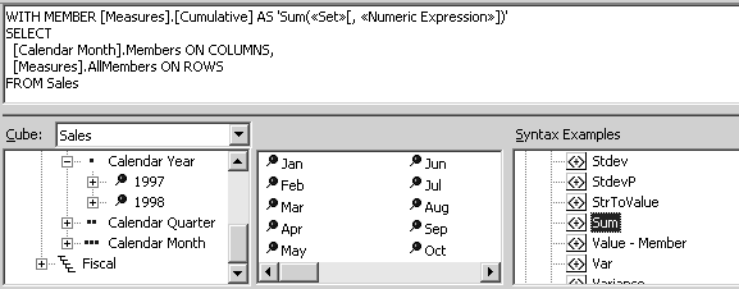
The screenshot shows the MDX query editor with the following query:

```
SELECT
    [Calendar Month].Members ON COLUMNS,
    [Measures].AllMembers ON ROWS
FROM Sales
```

Below the query, the 'Cube: Sales' is selected. The 'Calendar Month' dimension is expanded, showing months from Jan to Jun. The 'Measures' dimension is also expanded, showing 'Sales Units' and 'Sales Dollars'.

	Jan	Feb	Mar	Apr	May	Jun
Sales Units	244.00	320.00	275.00	281.00	479.00	449.00
Sales Dollars	\$486.58	\$675.53	\$540.97	\$587.31	\$964.60	\$878.50

3. Before the SELECT keyword, enter the clause **WITH MEMBER** **[Measures].[Cumulative] AS ''** and put the insertion point between the single quotes. Then expand the Numeric group of the Syntax Examples tree, and double-click the *Sum* function.



The screenshot shows the MDX query editor with the following query:

```
WITH MEMBER [Measures].[Cumulative] AS 'Sum(«Set»[, «Numeric Expression»])'
SELECT
    [Calendar Month].Members ON COLUMNS,
    [Measures].AllMembers ON ROWS
FROM Sales
```

Below the query, the 'Cube: Sales' is selected. The 'Calendar Month' dimension is expanded, showing months from Jan to Jun. The 'Measures' dimension is also expanded, showing 'Sales Units' and 'Sales Dollars'. The 'Syntax Examples' tree is expanded, showing the 'Sum' function selected.

	Jan	Feb	Mar	Apr	May	Jun
Sales Units	244.00	320.00	275.00	281.00	479.00	449.00
Sales Dollars	\$486.58	\$675.53	\$540.97	\$587.31	\$964.60	\$878.50



Create a cumulative total



The *Sum* function syntax shows two arguments. The first argument, «Set», is required; this is the argument you used when you created the Non-WA measure in the preceding section. The second argument, «*Numeric Expression*», is surrounded by square brackets, indicating that it is optional. If you create a member on a nonmeasure dimension (such as Non-WA), you'll almost never use the «*Numeric Expression*» argument. If you create a member on the Measures dimension (such as Cumulative), you'll almost always use the «*Numeric Expression*» argument. The «*Numeric Expression*» argument tells you which measure to use as the base for the new measure.

4. Replace the «*Numeric Expression*» token with **[Sales Units]**, and delete the optional brackets. For the «Set» argument, you need a set that begins with the time period of the first column and ends with the time period of the current column. You can use a colon (:) to indicate a range of members.
5. Select the «Set» token. In the Metadata tree, select the Calendar Month level of the Time.Calendar dimension. Then double-click the first Jan member in the member name list. Click after the *[Jan]*, type a colon, press enter, and then type **[Time].[Calendar].CurrentMember**. The final definition of the member before the SELECT word should look like this:

WITH

```
MEMBER [Measure].[Cumulative] AS
    'Sum([Time].[Calendar].[All Time].[1997].[Qtr1].[Jan]:
        [Time].[Calendar].CurrentMember, [Sales Units])'
```

6. Run the query. The result grid shows the monthly values along with the cumulative values.

WITH

```
MEMBER [Measures].[Cumulative] AS
    'Sum([Time].[Calendar].[All Time].[1997].[Qtr1].[Jan]:
        [Time].[Calendar].CurrentMember, [Sales Units])'
```

SELECT

[Calendar Month].Members ON COLUMNS,

[Measures].AllMembers ON ROWS

FROM Sales

Cube: Sales

Syntax Examples

- Stdev
- StdevP
- StrToValue
- Sum

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug
Sales Units	244.00	320.00	275.00	281.00	479.00	449.00	564.00	446.00
Sales Dollars	\$486.58	\$675.53	\$540.97	\$587.31	\$964.60	\$878.50	\$1,228.89	\$905.87
Cumulative	244.00	564.00	839.00	1,120.00	1,599.00	2,048.00	2,612.00	3,058.00

Calculating the cumulative values from the beginning of a particular time period is an extremely common MDX task, so MDX provides functions to simplify the process. The most important is the *PeriodsToDate* function. This function creates a set much like the range you created in the *Cumulative* calculated member: the ending point of the set is the current member of the dimension flagged as a Time dimension (the first



See “Working with Time Dimensions” in Chapter 3 for details about flagging a dimension as a Time dimension.



one, if a cube contains more than one dimension flagged as Time). The default beginning member of the set is the first member that shares the same parent as the ending member. For example, in a normal calendar-year hierarchy with Calendar Year, Calendar Quarter, and Calendar Month levels, a current member of August would produce the set *July:August*. (*Qtr3* is the common parent.) A current member of *[1998].[Qtr3]* would produce the set *[1998].[Qtr1]:[1998].[Qtr3]*. (*[1998]* is the common parent.)

If you want to go back to a different common ancestor, simply specify the level of the ancestor you want. For example, with a current member of *[August]*, specifying Year as the level produces the range *[January]:[August]*. You can use the (All) level to go back to the beginning of the entire level.

Note The *PeriodsToDate* function behaves very similarly to the *ParallelPeriod* function described in Chapter 6. The *ParallelPeriod* function returns a single member, and the *PeriodsToDate* function returns a set of members that can be used in an aggregation function such as *Sum*.

7. Change the set portion of the Cumulative member expression to **PeriodsToDate([Time].[Calendar].[Calendar Year])**, run the query, and scroll to see the Cumulative value drop back to match the Sales Units value in January.

WITH
 MEMBER [Measures].[Cumulative] AS
 'Sum(PeriodsToDate([Time].[Calendar].[Calendar Year]), [Sales Units])'
 SELECT
 [Calendar Month].Members ON COLUMNS,
 [Measures].AllMembers ON ROWS
 FROM Sales

Cube: Sales

	Nov	Dec	Jan	Feb	Mar	Apr
Sales Units	786.00	837.00	691.00	737.00	2,550.00	2,349.00
Sales Dollars	\$1,660.16	\$1,805.03	\$1,463.49	\$1,555.51	\$5,555.38	\$5,062.12
Cumulative	4,722.00	5,559.00	691.00	1,428.00	3,978.00	6,327.00

8. Remove the argument from the *PeriodsToDate* function altogether—leaving only the empty parentheses. Run the query, and see the Cumulative value resynchronize with the Sales Units value every quarter.



WITH						
MEMBER [Measures].[Cumulative] AS						
'Sum(PeriodsToDate(), [Sales Units])'						
SELECT						
[Calendar Month].Members ON COLUMNS,						
[Measures].AllMembers ON ROWS						
FROM Sales						
Cube: Sales						
Calendar Year 1997 1998 Calendar Quarter Jan Feb						
	Jan	Feb	Mar	Apr	May	
Sales Units	244.00	320.00	275.00	281.00	479.00	
Sales Dollars	\$486.58	\$675.53	\$540.97	\$587.31	\$964.60	
Cumulative	244.00	564.00	839.00	281.00	760.00	

9. Give the query a comment, and save the query file.

You can use functions that aggregate multiple values when you create a new member for either the Measures dimension or for a nonmeasure dimension. On the Measures dimension, specify a Measure member to aggregate; you must use an explicit aggregation function—*Sum*, *Min*, *Max*, or *Count*. On a nonmeasure dimension, do not specify a Measure member (since the measure will intersect all the possible measures); you generally use the *Aggregate* function to apply the appropriate aggregation for whichever measure is currently being calculated.

The Sum Function Creates a Subquery

Consider how MDX calculates the value for a member using the *Sum* function. For each value, the *Sum* function creates a subquery behind the scenes. The set you provide as the first argument to the *Sum* function serves as the set for the row axis.

When you use the *Sum* function to create a nonmeasure member, the Measure member comes from the main query—from the row or column axis, or from an actual or implied filter. For example, you could use the following MDX statement to replicate the subquery used to calculate the Non-WA value for the Sales Units column in “Create a calculated member of a nonmeasure dimension”:

```
SELECT
    {[Sales Units]} ON COLUMNS,
    Except([State].[State].Members,{[Washington]}) ON ROWS
FROM Sales
```

The result grid of that subquery would look like this:

	Sales Units
British Colum	3,715.00
District Fede	2,807.00
Zacatecas	2,359.00
Oregon	6,551.00
California	6,331.00

(continued)



The *Sum* Function Creates a Subquery (continued)

The *Sum* function then simply sums all the values in the subquery result grid and returns that single value to the main query. The *Sum* function for the Sales Dollars column creates an analogous subquery to produce its single value.

When the *Sum* function is used to create a member of the Measures dimension, it still creates a subquery, but this time the Measures member cannot come from the main query since the Measures member of the main query is, by definition, the measure currently being calculated. That's why the *Sum* function requires a «*Numeric Expression*» argument when used with a Measures member. The «*Set*» argument to the *Sum* function still serves as the set for the row axis of the subquery. The «*Numeric Expression*» argument serves as the lone member of the column axis of the subquery, and all other dimensions are inherited from the main query. For example, you could use the following statement to simulate the subquery needed to calculate the cumulative May value from the query in "Create a cumulative total":

```
SELECT
  {[Sales Units]} ON COLUMNS,
  {[Time].[Calendar].[All Time].[1997].[Qtr1].[Jan]:
   [Time].[Calendar].[All Time].[1997].[Qtr2].[May]} ON ROWS
FROM Sales
```

The result grid of that subquery looks like this:

	Sales Units
Jan	244.00
Feb	320.00
Mar	275.00
Apr	281.00
May	479.00

Once again, the *Sum* function simply summarizes all the numbers and returns that single value. An analogous subquery is created for each cell of the Cumulative row in the main query. Fortunately, most of the values used in each subquery are stored in the client cache, so each subquery executes quickly.

The Sales Dollars values in the Chapter 7 OLAP database are net of discounts. In other words, if you multiply the Sales Units for a product by the official list price of the product as stored in the Product table of the data warehouse, you'll get a number larger than the Sales Dollars amount. The difference is in various discounts that can be applied. To calculate the Discount Dollars at every level of summarization on all dimensions involves several steps. In this section, you'll learn those steps.

Calculate discount dollars by using a member function

The first step is to retrieve the price for a product. The Product dimension includes Price as a member property. You can create a calculated member that retrieves the value of that member property.

1. Create a new query, and insert the following MDX query statement:

```
SELECT
    [Measures].AllMembers ON COLUMNS,
    Product.Members ON ROWS
FROM Sales
```

2. Run the query. This extracts the Sales Units and Sales Dollars for all levels of the product hierarchy. You can now create a calculated measure that retrieves the Price member property.

	Sales Units	Sales Dollars
All Product	35,696.00	\$76,741.28
Bread	14,532.00	\$30,600.42
Bagels	1,541.00	\$3,552.28
Colony Bagel	264.00	\$308.88
Fantastic Ba	299.00	\$1,022.58
Great Bagels	334.00	\$604.54
Modell Bagel	304.00	\$994.08
Sphinx Bagel	340.00	\$622.20
Muffins	6,445.00	\$13,081.28

3. Insert the following calculated member:

```
WITH
    MEMBER [Measures].[Base Price] AS
        '[Product].Properties("Price")'
```

4. Run the query. When you retrieve a member property for a dimension, you get a value only for the level at which the member property was defined. At any other level, the member property returns an error value.

	Sales Units	Sales Dollars	Base Price
All Product	35,696.00	\$76,741.28	#ERR
Bread	14,532.00	\$30,600.42	#ERR
Bagels	1,541.00	\$3,552.28	#ERR
Colony Bagel	264.00	\$308.88	1.22
Fantastic Ba	299.00	\$1,022.58	3.69
Great Bagels	334.00	\$604.54	1.86
Modell Bagel	304.00	\$994.08	3.4
Sphinx Bagel	340.00	\$622.20	1.85
Muffins	6,445.00	\$13,081.28	#ERR
Colony Bluet	298.00	\$739.04	2.65

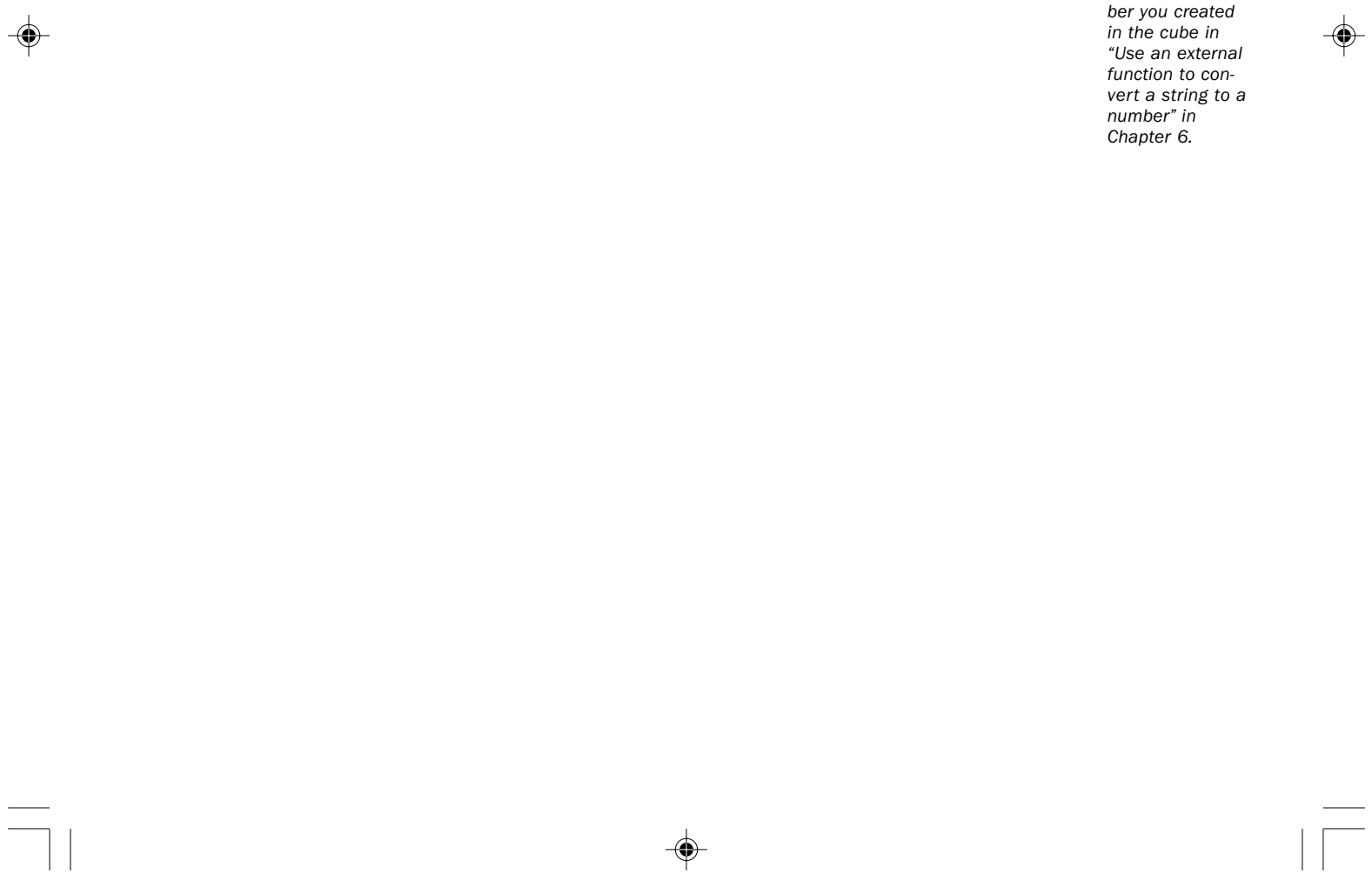
The Price member property was defined at the product level. A member property is always stored as a text string, even if it is a numeric value such as the price. To use the price, you must convert the string to a number. As explained in Chapter 6, MDX doesn't have a built-in function to convert a string to a value, but you can use the Microsoft Visual Basic for Applications (VBA) function *Cdbl* to convert a string to a double (that is, a number that can include a decimal place).

5. Change the expression for the Base Price member to '**Cdbl([Product].Properties("Price"))**'. Run the query. Prices exist only at the Product level. The error values that previously appeared at the higher summary



The Price member property was added to the Product dimension in “Create a detail-level member property” in Chapter 3.

This is the same calculated member you created in the cube in “Use an external function to convert a string to a number” in Chapter 6.



levels are now some strange large numbers. You can ignore them for the time being. The prices at the product level values don't appear to change, but you can now use them in arithmetic calculations.

	Sales Units	Sales Dollars	Base Price
All Product	35,696.00	\$76,741.28	-2147467259
Bread	14,532.00	\$30,600.42	-2147467259
Bagels	1,541.00	\$3,552.28	-2147467259
Colony Bagel	264.00	\$308.88	1.22
Fantastic Ba	299.00	\$1,022.58	3.69
Great Bagel	334.00	\$604.54	1.86
Modell Bagel	304.00	\$994.08	3.4

If you want to calculate the Gross Dollars for a product, you must multiply the Price value by the Sales Units. To calculate the Gross Product for a product category or subcategory, you must sum up the Gross Dollars for all the products in that category or subcategory. For example, suppose that you want to calculate the Gross Dollars for the Bagels product subcategory. You must first find all the products that belong to that subcategory. For each product, multiply the Base Price by the Sales Units to calculate the Gross Dollars for the product. Then sum the Gross Dollars for all the products to get the Gross Dollars for the subcategory.

6. Add the following additional calculated member to the query:

```
MEMBER [Measures].[Gross Dollars] AS
    'Sum(Descendants([Product].CurrentMember,[Product Name]) ,
        [Base Price]*[Sales Units])'
```

The expression for this member uses the *Sum* function. Think through the subquery the *Sum* function generates for the Bagels product. The first argument of the *Sum* function specifies a set for the row axis of the subquery. In this case, you define that set by using the *Descendants* function. The *Descendants* function is like the *Children* function, except that it can retrieve values at any level. The *Descendants* function starts with the Bagels member. It finds all the descendants of Bagels at the Product level—the row axis of the subquery contains all the members of the Product level that are descendants of the Bagels member. The second argument of the *Sum* function specifies the column axis of the subquery. In this case, the subquery includes the Base Price (derived from the Price member property) and the Sales Units (as stored in the cube). The members of the other dimensions are inherited from the main query. The subquery for the Bagels member would look like this:

```
WITH
    MEMBER [Measures].[Base Price] AS
        'CDB1([Product].Properties("Price"))'
    SELECT {[Base Price], [Sales Units]} On Columns,
        Descendants([Bagels],[Product Name]) On Rows
    From [Sales]
```



And the results of the subquery for the Bagels member would look like this:

	Base Price	Sales Units
Colony Bagel	1.22	264.00
Fantastic Ba	3.69	299.00
Great Bagels	1.86	334.00
Modell Bagel	3.4	304.00
Sphinx Bagel	1.85	340.00

The *Sum* function then multiplies the values in the first column by those in the second column and sums the results.

- Now run the query with the Gross Dollars calculated member.

	Sales Units	Sales Dollars	Base Price	Gross Dollar
All Product	35,696.00	\$76,741.28	2147467259	80,535.43
Bread	14,532.00	\$30,600.42	2147467259	31,938.62
Bagels	1,541.00	\$3,552.28	2147467259	3,709.23
Colony Bagel	264.00	\$308.88	1.22	322.08
Fantastic Ba	299.00	\$1,022.58	3.69	1,103.31
Great Bagels	334.00	\$604.54	1.86	621.24
Modell Bagel	304.00	\$994.08	3.4	1,033.60
Sphinx Bagel	340.00	\$622.20	1.85	629.00
Muffins	6,445.00	\$13,081.28	2147467259	13,795.88

As expected, the Gross Dollars are slightly higher than the Sales Dollars at each level of summary. At the Product level, the subquery performed by the *Sum* function is small—only a single row. At the All Product level, the subquery includes all the products in the entire hierarchy. Percolating a member property up the hierarchy can be a slow operation on a large hierarchy.

- To calculate the Discount dollars, add the following calculated member, and then run the query, add a comment, and save the query file. (Then try varying the set on the row axis—use sets from other dimensions.)

MEMBER [Measures].[Discount Dollars] AS
'[Gross Dollars] - [Sales Dollars]'

```
WITH
    MEMBER [Measures].[Base Price] AS
        'Cdbk([Product].Properties("Price"))'
    MEMBER [Measures].[Gross Dollars] AS
        'Sum( Descendants([Product],CurrentMember,[Product Name]) ,
            [Base Price]*[Sales Units])'
    MEMBER [Measures].[Discount Dollars] AS
        '[Gross Dollars] - [Sales Dollars]'
SELECT
    [Measures].AllMembers ON COLUMNS,
    Product.Members ON ROWS
FROM Sales
```

Cube: Sales

Time: All Time

Year: 1997

	Sales Units	Sales Dollars	Base Price	Gross Dollar	Discount Dol
All Product	35,696.00	\$76,741.28	2147467259	80,535.43	3794.15
Bread	14,532.00	\$30,600.42	2147467259	31,938.62	1338.2
Bagels	1,541.00	\$3,552.28	2147467259	3,709.23	156.9499
Colony Bagel	264.00	\$308.88	1.22	322.08	13.1999
Fantastic Ba	299.00	\$1,022.58	3.69	1,103.31	80.7299
Great Bagel	334.00	\$604.54	1.86	621.24	16.7



You can use a member property—particularly one that has a numeric value—within calculated members. You must remember to convert the member property to a numeric value before using it in a calculation, and you must use the member property only at the level at which it was created, using aggregation functions to percolate the values up the rest of the hierarchy.

To	Do this
Add a comment to an MDX query	Enter double slash characters (//) before the comment.
Include all the members of the Product Name level on the column axis	Use the clause <i>[Product Name].Members ON COLUMNS</i> .
Put sets of members on two axes	Use both ON COLUMNS and ON ROWS in a query.
Create a set from explicit members	Enclose the list of members separated by commas, between opening braces ({} and closing braces ({}).
Create a set consisting of all possible combinations of two sets	Use the <i>CrossJoin</i> function with the two sets as arguments.
Calculate the average of all members of the State level of the State dimension	Use the expression <i>Avg([State].[State].Members)</i> .
Calculate an aggregation by using each measure's aggregation function	Use the <i>Aggregate</i> function.
Calculate the cumulative total restarting with each new member of the Calendar Year level	Use the expression <i>Sum(PeriodsToDate([Calendar Year]))</i> .
Calculate a numeric value of the Price member property of the current member of the Product dimension	Use the expression <i>Cdbl([Product].Properties("Price"))</i> .



Chapter Summary

