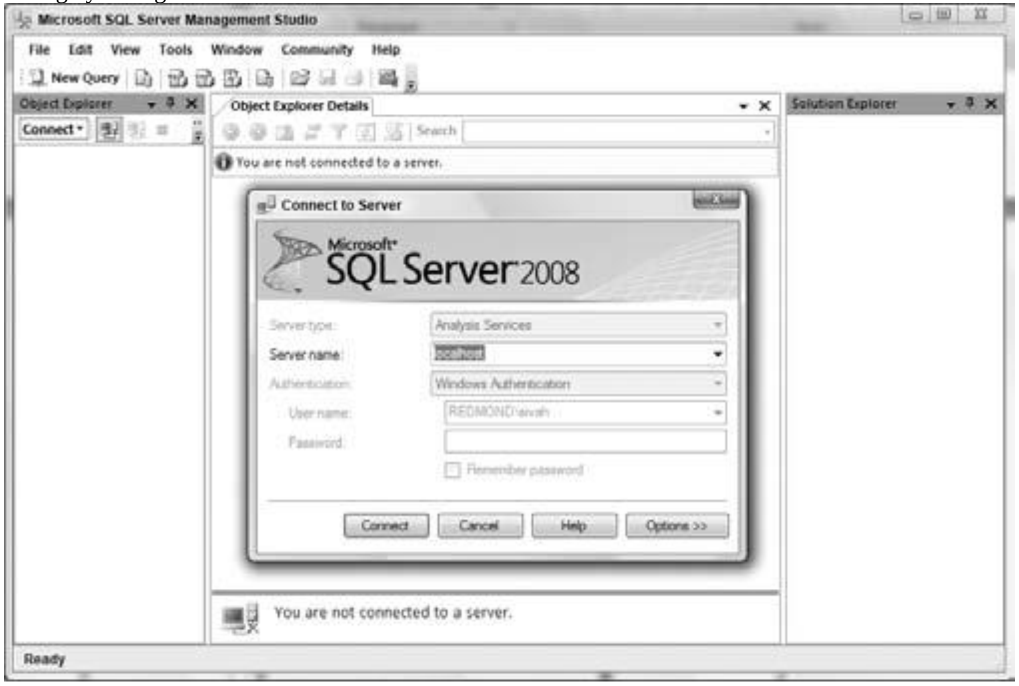
Лекция 4

Тема « Using SQL Server Management Studio »

Using SQL Server Management Studio

SQL Server Management Studio (SSMS) is ground zero for administering the Analysis Services servers resident on your network. Not only that, you can also administer instances of SQL Server, Reporting Services, Integration Services, and SQL Server Compact Edition from within SSMS. In this book you learn how to administer and manage Analysis Servers. This chapter specifically discusses the Analysis Services objects shown in the Object Explorer. Administering Analysis Services is discussed in more detail in Chapters 7 and 13 .

The first step in the process of working with objects in the Object Explorer is connecting to the servers you wish to manage. In fact, as soon as you start Management Studio, you get a dialog prompting you to connect to one of the server types as shown in Figure 2 - 41 . Create a connection to the Analysis Services through your login.



You are not connected to a server.

Server type: Analysis Services
Server name: localhost
Authentication: Windows Authentication
User name: REDMOND\sivah
Password:
Remember password

You are not connected to a server.

Figure 2-41

SSMS provides you with a way to register your servers so that you do not have to specify the login credentials each time you need to connect. Click the View menu and select Registered Servers. You will see a window called Registered Servers in SSMS as shown in Figure 2 - 42 . In the Registered Servers pane, click the toolbar icon second from left; this enables you to register an Analysis Services instance. Now, right - click the Local Server Groups folder and select New Server Registration. In the resulting New Server Registration dialog (see Figure 2 - 43) you need to specify the name of the Analysis Services instances you wish to connect to and optionally specify connection parameters such as time out and enabling encryption. If the server instance you wish to connect to is a named instance, enter its name in the Server name field, otherwise, type in **localhost** , which means you want to register the default instance of Analysis Services on your machine. Once you have filled in the Server name field, you can test the connection by clicking the Test button at the bottom of the dialog. If the connection does not succeed, you need to make sure Analysis Services is running and that your authentication scheme is correct. Once the connection is working, click Save.

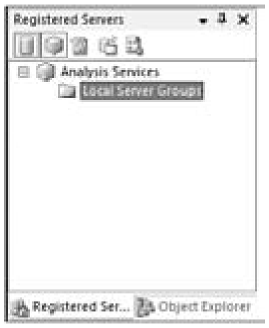
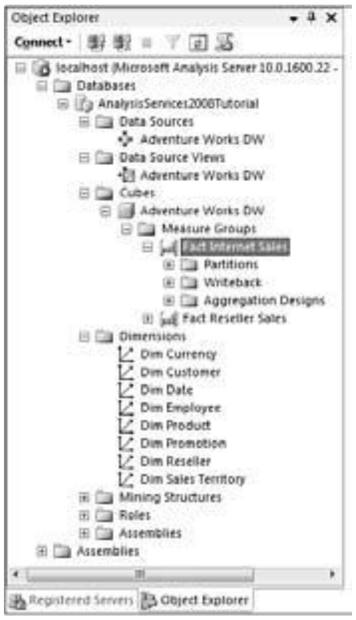


Figure 2-43

The Object Explorer Pane

When you connect to an Analysis Services instance, you see it in the Object Explorer pane (see Figure 2 - 44). This section reviews the various objects in Analysis Services. Open the Databases folder to see the AnalysisServices2008Tutorial database and expand each object type folder. You should be looking at a list of the seven major object types (Data Sources, Data Source Views, Cubes, Dimensions, Mining Structures, Roles, and Assemblies) as shown in Figure 2 - 44 .



The following list describes each of the objects:

Databases: Database objects are where your deployed Analysis Services projects are listed; note that these objects could have been created in On - line mode or Project mode.

Data Sources: The Data Sources folder will, at minimum, contain a single object pointing to a data source like SQL Server 2008 if you have cubes or dimensions or mining models. Behind the scenes, these objects store connection information to a relational data source, which can be for a .NET provider or an OLE DB provider. In either case, you can establish a connection to a data source. In Figure 2 - 44 , you can see the data source called “ Adventure Works DW. ” Most databases will have multiple data sources.

Data Source Views: A Data Source View object refers to a subset of the data identified by its parent data source object. The reason this object type exists is because, in the enterprise environment, a data source might contain thousands of tables, though here you ’ re interested in working with only a small subset of those tables. Using a DSV object, you can restrict the number of tables shown in a given view. This makes working on even the largest database a manageable task. On the other hand, you might want to create a DSV to contain not only all tables in one database, but a portion of tables from a second database. Indeed, you can have a DSV that uses more than one data source for an optimal development environment.

Cubes: You have already looked at the details of cubes in BIDS; they are the lingua franca of Business Intelligence. Well, cubes can also be viewed in the Object Explorer pane. Further, four

sections under the Cubes object provide information about how the cube is physically stored
and whether or not it will allow you to write data back to the cube:

❑**Measure Groups:** Measure groups are comprised of one or more columns of a fact table
which, in turn, are comprised of the data to be aggregated and analyzed. Measure groups
combine multiple logical measures under a single entity.

❑**Partitions:** Partitioning is a way of distributing data to achieve efficient management as
well as improved query performance. You typically partition fact data if you have a large
fact table. In this way you can make the queries run faster. This works because scanning
partitions in parallel is faster than scanning serially. There is a maintenance benefit as well;
when you do incremental updates (process only data changed since the last update) it is
more efficient for the system to update only those partitions that have changed. A typical
partitioning strategy adopted is partitioning the data based on a time dimension. A variation
of the partitioning strategy is to also have different storage modes for some partitions.
In this way, a single fact table might have only up to five years of the most recent data in a
few MOLAP partitions and is therefore subject to queries, whereas the older, less often accessed
data can lie fallow in a ROLAP partition. If you right - click the Partitions folder in
the FactInternetSales measure group, you will see a number of administrative tasks associated
with partitions that can be dealt with directly in SSMS.

❑**Writeback:** Writeback provides the flexibility to perform a " what if " analysis of data or to
perform a specific update to a measure such as budget when your budget for next year
gets changed. The Writeback folder is empty in AnalysisServices2008Tutorial because it
has not been enabled. By default writeback is not turned on. To see what options are available,
right - click the Writeback object.

❑**Aggregation Designs:** Aggregation designs help in pre - aggregating fact data for various
dimension members and storing them on disk. Aggregation designs are created using the
Aggregation Designer or Usage Based Optimization wizards. You learn about the benefits
of aggregations and how to design them in Chapters 9 , 14 , and 15 . Once aggregations are designed for a cube, you can see the aggregation designs of a partition in this folder. You
can assign aggregation designs to a partition or edit existing aggregation designs using
SSMS. Right - click the Aggregation Designs folder or specific aggregation designs to see the
various options.

Dimensions: Dimensions are what cubes are made of, and you can see what dimensions are
available for use in a given project by looking at the contents of this folder. Note that you can
browse, process, and delete dimensions from here with a right - click of the mouse.

Mining Structures: Data mining requires a certain amount of infrastructure to make the
algorithms work. Mining structures are objects that contain one or more mining models. The
mining models themselves contain properties like column content type, your data mining
algorithm of choice, and predictable columns. You create mining models based on a mining
structure. You learn about data mining in Chapter 16 .

Roles: Roles are objects that define a database - specific set of permissions. These objects can be
for individual users or groups of users. Three types of permissions can be set for a role:
Administrator level or Full control, Process Database level, and Read Database Metadata level.
Roles are discussed with the help of a scenario in Chapter 22 .

Assemblies: You learned earlier in this chapter that assemblies are actually stored procedures
(created with .NET or COM - based programming languages) used on the server side for custom
operations. The assembly support in Analysis Services 2005 is continued in Analysis Services
2008. If you are familiar with Analysis Services 2000 and UDFs (user - defined functions), note
that assemblies can do anything UDFs can do and more. Also note that COM UDFs in Analysis
Services 2000 are also supported in Analysis Services 2008 for backward compatibility. The
scope of these assemblies is database - specific; that is, an assembly can only operate on the
Analysis Services database for which it is run.

Server Assemblies: If you want to operate on multiple databases in Analysis Services, you have
to create this type of object, the server assembly. Server assemblies are virtually the same as
assemblies, except their scope is increased; they work across databases in Analysis Services.

Querying Using the MDX Query Editor

Just to recap, MDX is a language that allows you to query multi dimensional databases similar to the way
SQL is used to query relational databases. MDX is used to extract information from Analysis Services
cubes or dimensions. Whereas SQL returns results along two axes — rows and columns — MDX returns
data along multiple axes. You learn about MDX in depth in Chapters 3 and 10 . For now, let ' s look at a

simple MDX query to learn how to execute it and view its results.
The syntax of a typical MDX query is as follows:

```
SELECT [ < axis_specification >
[, < axis_specification > ...]]
FROM [ < cube_specification > ]
[WHERE [ < slicer_specification > ]]
```

The MDX SELECT clause is where you specify the data you need to retrieve across each axis. The FROM clause is used to specify the cube from which you retrieve the data. The optional WHERE clause is used to slice a small section of data from which you need results.
In Analysis Services 2000, an MDX Sample application was included that could be used to send queries to cubes and retrieve results. In Analysis Services 2005 and 2008, query editors are integrated right into SSMS for sending queries to SQL Server and Analysis Services instances. These query editors have IntelliSense (dynamic function name completion) capabilities built in. When MDX queries are saved from SSMS they are saved with the extension .mdx. You can open the MDX query editor in SSMS by selecting File New Analysis Services MDX Query as shown in Figure 2 - 45 or by clicking the MDX query button as shown in Figure 2 - 46 .

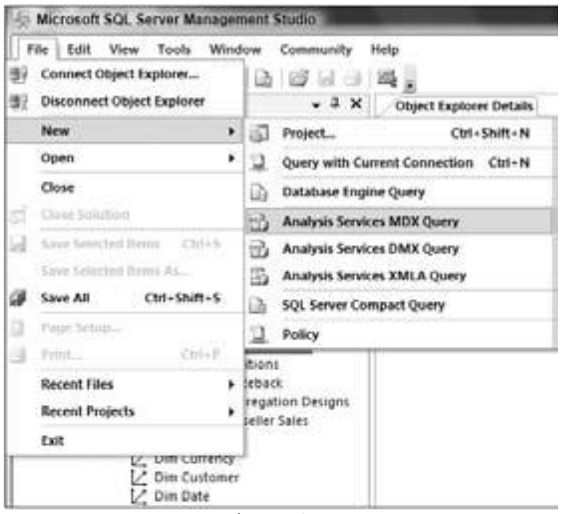
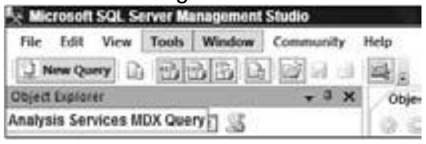
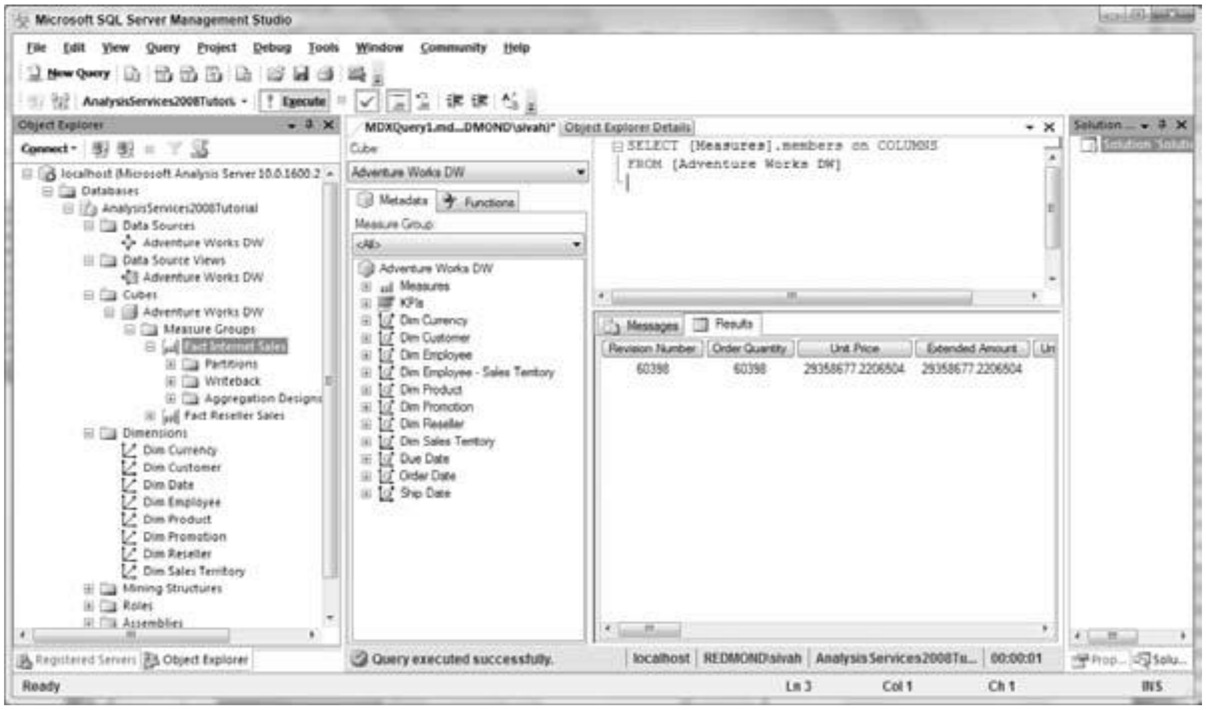


Figure 2-45


Microsoft SQL Server Management Studio
File
Edit
View
Tools
Window
Community
Help
New Query
Object Explorer
Analysis Services MDX Query
Figure 2-46

You will be prompted to connect to your Analysis Services instance. After you establish a connection, you can select the name of the database you wish to use from the Database Selection drop - down box shown in Figure 2 - 47 . Select the AnalysisServices2008Tutorial database that you created in this chapter. In this database you created a single cube called Adventure Works DW, which is shown in the Cube drop - down box. The Query Editor is composed of two window panes, the Metadata pane on the left and the Query pane on the right. In the Query pane, you can make use of the IntelliSense feature by pressing Ctrl and Spacebar after typing in a few characters of an MDX keyword.
Now you can type the following query in the Query pane:

```
SELECT [Measures].members on COLUMNS
FROM [Adventure Works DW]
```



```
SELECT [Measures].members on COLUMNS
FROM [Adventure Works DW]
```

Revision Number	Order Quantity	Unit Price	Extended Amount
60398	60398	29358677.2206504	29358677.2206504

Figure 2-47

You can now execute the query by pressing the Ctrl+E key combination or clicking the Execute button. On execution, the query construction pane splits in two, and the results from the server are shown in the bottom half. All MDX queries cited in this book can be executed using this method. Congratulations, you just ran your first MDX query! You can see the results of the MDX query in the Results pane where you can see the members on axes and the corresponding cell values as shown in Figure 2 - 47 .

Summary

In this chapter you were introduced to Analysis Services 2008 and learned how it overcomes the limitations of its predecessors, Analysis Services 2000 and Analysis Services 2005. In addition to trumping these limitations, Analysis Services 2008 provides a rich suite of tools for development and management of Analysis Services databases, which were first introduced as part of Analysis Services 2005.
You were also introduced to Business Intelligence Development Studio, which is core to designing Analysis Services cubes and dimensions. You successfully created a cube using the Cube Wizard. In the course of building that cube, you learned about data sources, Data Source Views, dimensions, and the wizards used to create these objects. You successfully deployed the cube to Analysis Services and then browsed it within Business Intelligence Development Studio.

In the second part of this chapter you learned about the integrated management environment of SQL Server 2008, SQL Server Management Studio, which is used to manage SQL Server and Analysis Services. You were familiarized with the various objects within an Analysis Services database by browsing them in the Object Explorer.
Finally, you learned that MDX does not require a Ph.D. in nuclear physics to use. The MDX Query Editor can be used easily to execute an MDX query, in this case, against the cube you built. Finally, you were able to view query results. In the next chapter you learn the basics of MDX, which will form the foundation of your deeper understanding of Analysis Services 2008.

Introduction to MDX

In Chapter 2 you ran a simple MDX query to retrieve data from Analysis Services 2008. Building on that, in this chapter you learn the fundamental concepts underlying MDX and how you can manipulate and query multidimensional objects within Analysis Services. This chapter forms the basis for many of the subsequent chapters in this book. In fact, in several places in this chapter and throughout the book you see how each interaction between the client tools and the Analysis

Services instance results in the generation of MDX. You not only see the MDX that is generated, but you also glean some insight as to what the MDX does.
SQL Server 2008 provides a sample Analysis Services project that demonstrates the majority of the features provided by Analysis Services 2008. In this chapter you use the sample Analysis Services project available from www.codeplex.com to learn MDX. The illustrations are limited to three dimensions to help you understand the concepts. You can extend these concepts, if you want, to view data across additional dimensions. In this chapter you learn the basic concepts regarding cells, members, tuples, and sets. In addition, you learn how to create MDX expressions and MDX queries for data analysis.

What Is MDX ?

Just as SQL (Structured Query Language) is a query language used to retrieve data from relational databases, MDX (Multi - Dimensional eXpressions) is a query language used to retrieve data from multidimensional databases. More specifically, MDX is used for querying multidimensional data from Analysis Services and supports two distinct modes. When used in an expression, MDX can define and manipulate multidimensional objects and data to calculate values. As a query language, it is used to retrieve data from Analysis Services databases. MDX was originally designed by Microsoft and introduced in SQL Server Analysis Services 7.0 in 1998.
MDX is not a proprietary language; it is a standards - based query language used to retrieve data from OLAP databases. MDX is part of the OLE DB for OLAP specification sponsored by Microsoft. Many other OLAP providers support MDX, including Microstrategy ' s Intelligence Server, Hyperion ' s Essbase Server, and SAS ' s Enterprise BI Server. There are those who wish to extend the standard for additional functionality, and MDX extensions have indeed been developed by individual vendors.
MDX extensions provide functionality not specified by the standard, but the constituent parts of any extension are expected to be consistent with the MDX standard. Analysis Services 2008 does provide several extensions to the standard MDX defined by the OLE DB for OLAP specification. In this book you learn about the form of MDX supported by Analysis Services 2008.

When one refers to MDX they might be referring either to the MDX query language or to MDX expressions. Even though the MDX query language has similar syntax as that of SQL, it is significantly different. Nonetheless, we will use SQL to teach you some MDX. Before you get into the details of MDX query language and MDX expressions, you need to learn some fundamental concepts.

Fundamental Concepts

A multidimensional database is typically referred to as a cube. The cube is the foundation of a multidimensional database, and each cube typically contains more than two dimensions. The Adventure Works cube in the sample database contains 21 dimensions. The SQL Server 2008 product samples need to be downloaded from http://www.codeplex.com/MSFTDBProdSamples . Find and download the SQL2008.AdventureWorks_DW_BI_v2008< architecture > .msi and install it on your machine where the architecture is x86, x64, or ia64. This package contains the sample relational database AdventureWorksDW2008 and the Analysis Services project AdventureWorks. You need to specify the SQL Server 2008 relational database instance to install the sample. Using Business Intelligence Development Studio (BIDS), open the sample Adventure Works project from Program Files\Microsoft SQL Server\100\Tools\Samples\AdventureWorks 2008 Analysis Services Project\Enterprise. Deploy the project to your Analysis Services instance. If your SQL Server 2008 relational server and/or Analysis Services instance are named instances, you need to make changes to the data source and the Analysis Services target server as mentioned in Chapter 2 . If you open the Adventure Works cube in BIDS you can see the measures and dimensions that make up the cube in the Cube Structure tab as shown in Figure 3- 1.

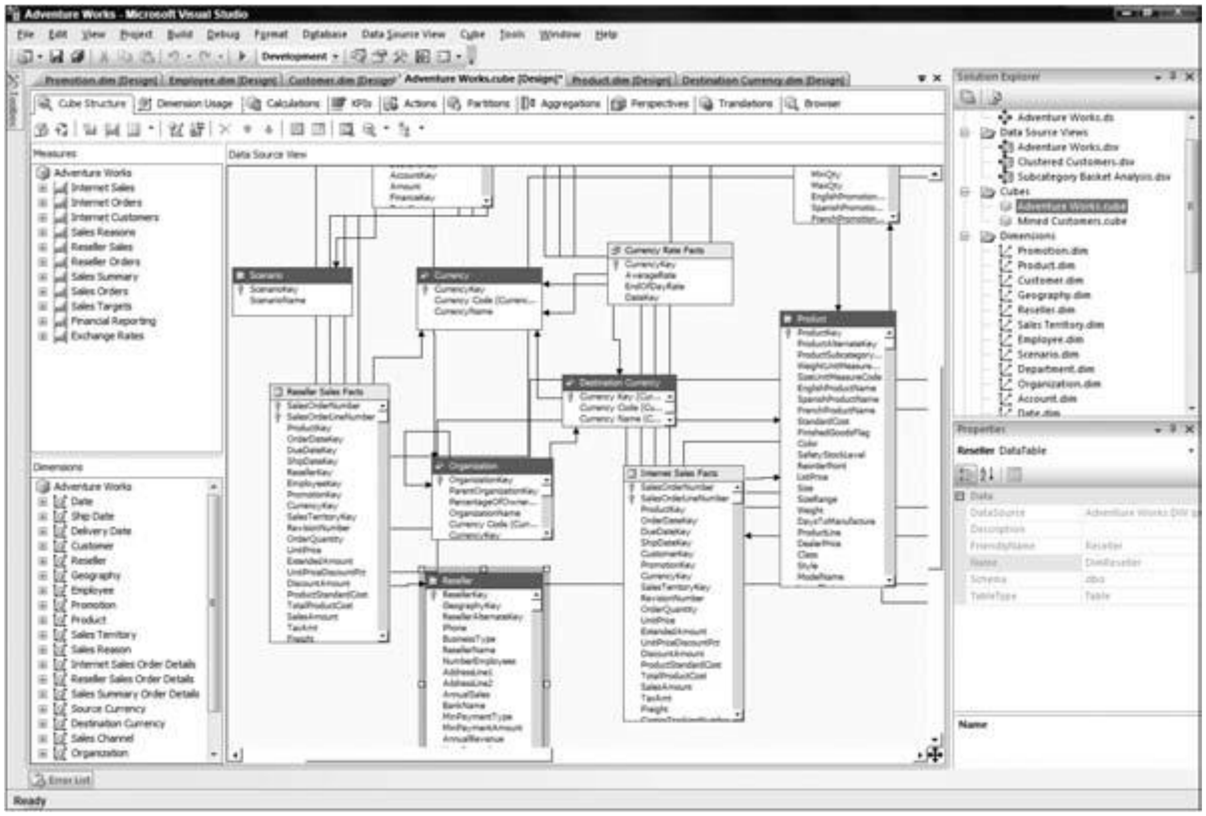


Figure 3 - 1

The *Measures* object within a cube is a special cube dimension that is a collection of measures. Measures are quantitative entities that are used for analysis. You can see the measures in the sample project in Figure 3 - 1 . Each measure is part of an entity called a measure group. *Measure groups* are collections of related measures and each measure can only be part of a single measure group. Often you will want to have one measure group for each fact table in your data warehouse. Measure groups are primarily used for navigational purposes by design tools or client tools in order to have better readability or ease of use for end users. Measure groups are never used in MDX queries when querying measures. However, they can be used in certain MDX functions which, by the way, you see in this chapter and in Chapter 10 . By default, Analysis Services generates a measure group for each fact table, so you don ’ t have to worry about changing the measure group ’ s design. If you want to, of course, you can.
In Figure 3 - 1 you can see the dimensions that are part of the Adventure Works cube. Each dimension has one or more hierarchies and each hierarchy contains one or more levels. You learn more about dimensions, hierarchies, and levels in Chapters 5 and 8 . To demonstrate the fundamental concepts of MDX, we will use three of the dimensions: Product, Customer, and Date. We will use the hierarchies Calendar, Product Line, and Country from the dimensions Date, Product, and Customer, respectively, to illustrate fundamental concepts in MDX. Figure 3 - 2 shows a section of the Adventure Works cube using the three hierarchies: Calendar, Product Line, and Country. The Calendar hierarchy of the Date dimension contains five levels: Calendar Year, Calendar Semester, Calendar Quarter, Month, and Date. The Product Line and Country are attribute hierarchies and have two levels: the All level and the Product Line or Country level, respectively. For illustration purposes, Figure 3 - 2 does not contain all the members or levels of the Calendar, Product Line, and Country hierarchies and hence Figure 3 - 2 does not reflect the actual data in the sample cube.

Members

Each hierarchy of a dimension contains one or more items that are referred to as *members* . Each member corresponds to one or more occurrences of that value in the underlying dimension table. Figure 3 - 3 shows the members of the Calendar hierarchy in the Date dimension. In the Calendar hierarchy, the items CY 2003, H1 CY 2003, H2 CY 2003, Q1 CY 2003, Q2 CY 2003, and Q3 CY 2003 and Q4 CY 2004 are the members. You can see that the items at each level together form the collection of the members of the hierarchy. You can also query the members of a specific level. For example, Q1 CY 2003, Q2 CY 2003, Q3

CY 2003, and Q3 CY 2004 are members of the Calendar Quarter level for the calendar year CY 2003.

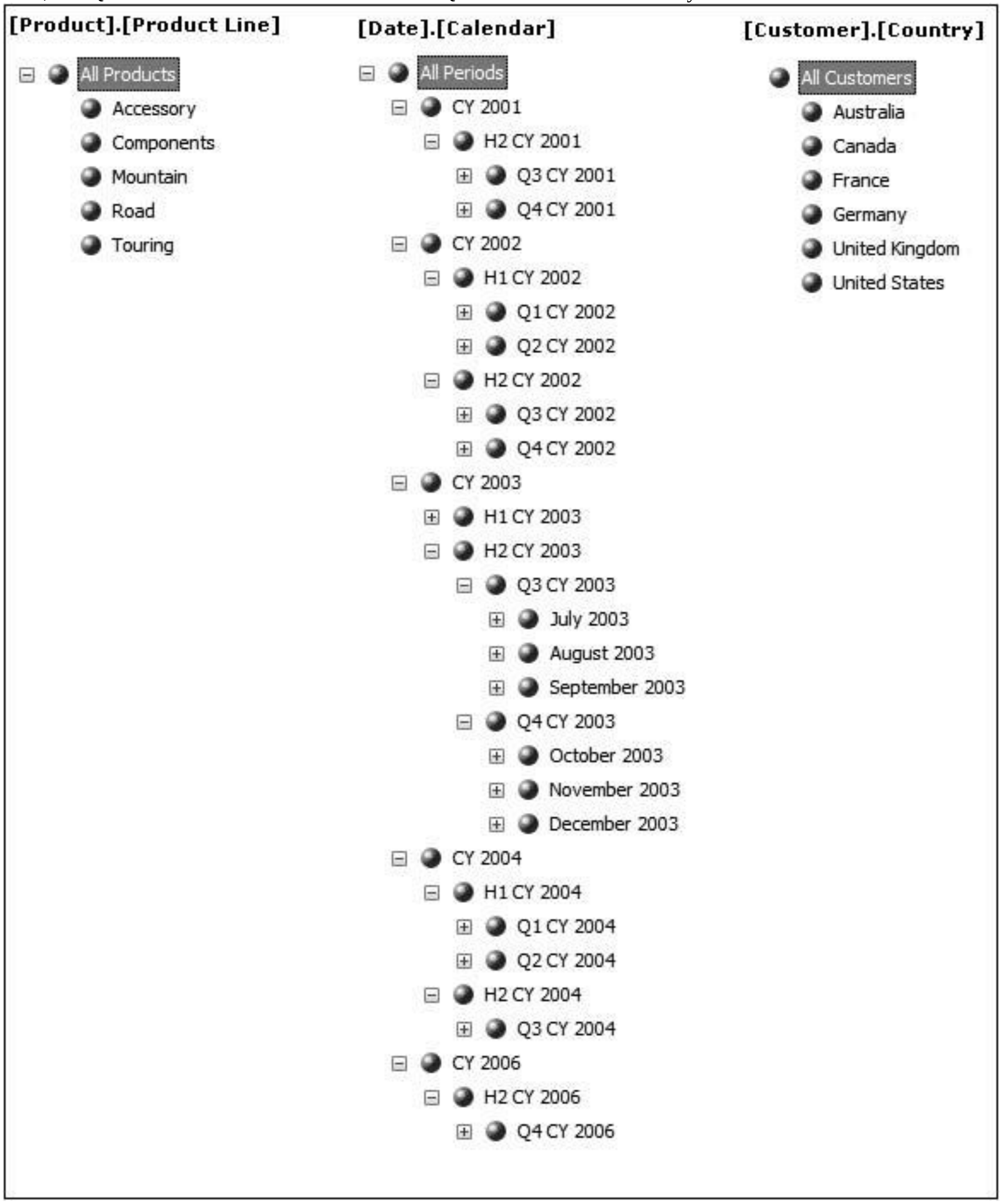


Figure 3-3

In MDX, each member of a hierarchy is represented by a unique name. The unique name is used to identify a specific member. The unique name for a member is dependent upon the dimension properties such as MemberUniqueNameStyle and HierarchyUniqueNameStyle. The algorithm determining the unique name of a member is not discussed in this book. You can access members of a dimension using the name path (using the name of the member) or the key path (using the key of the member). Using the

default properties in BIDS to create your cubes and dimensions, you can access a member in a dimension with its dimension name, hierarchy name, and level name. For example, member Q1 CY 2004 in the Calendar hierarchy is represented as

```
[Date].[Calendar].[Calendar Quarter].[Q1 CY 2004]
```

The brackets are used to enclose the names of the dimension, hierarchy, levels, and members. It is not necessary that these names be enclosed within the square brackets every time, but whenever you have a name that contains a space, has a number in it, or is an MDX keyword, brackets must be used. In the preceding expression the dimension name Date is an MDX keyword and hence must be enclosed within brackets.
The following three representations are also valid for the member Q1 CY 2004:

```
[Date].[Calendar].[Q1 CY 2004] (1)
[Date].[Calendar].[CY 2004].[H1 CY 2004].[Q1 CY 2004] (2)
[Date].[Calendar].[Calendar Quarter]. & [2004] & [1] (3)
```

In the first representation the member is represented in the format Dimension.Hierarchy.Member name. You can use this format as long as there are no two members with the same name. For example, if quarter 1 in each year is called Q1, you cannot use the preceding format; you would need to qualify using the level name in the MDX expression. If you do use the preceding format it will always retrieve Q1 for the first year in the hierarchy. In the second format, you can see the navigational path for the member clearly because you see all the members in the path. So far, the formats you have seen for accessing members all use the names of the members. The final format uses the key path where the keys of the members in a path are represented as & [membername]. When you use the key path, the members are always preceded with the ampersand (&) symbol.
Another example is the Australia member of the Country hierarchy in the Customer dimension, which would be specified as:

```
[Customer].[Country].Australia
```

Notice that there are no square brackets in the expression for the member Australia. This is because Australia is one word and no numbers are involved. In general, you can use the following format for accessing a member:

```
[DimensionName].[HierarchyName].[LevelName].[MemberName]
```

This format is predominantly used in this chapter as well as the rest of the book. If you are developing client tools we recommend you retrieve the unique name of the members directly from Analysis Services and use that in the MDX queries generated from the client tool instead of hard-coding the unique name in the client tool.

Cells

In Figure 3 - 2 you can see three faces of the cube. The front face has been divided into 16 small squares, and each square holds a number. Assume the number within each square is the measure " Internet Sales Amount " of the AdventureWorksDW cube. If you view the remaining visible faces of the cube you will realize that each square you analyzed in the front face of the cube is actually a small cube itself. The top - right - corner square of the front face contains the value 1134; you will notice that the same number is represented on the other sides as well. This smaller cube is referred to as a *cell* .
A cell is an entity from which you can retrieve data that is pertinent to an intersection of the dimension members. The number of cells within a cube depends on the number of hierarchies within each dimension and the number of members in each hierarchy. As you can imagine, cells hold the data values of all measures in a cube. If the data value for a measure within a cell is not available, the corresponding measure value is Null.

If you are familiar with three - dimensional coordinate geometry, you are aware of the three axes X, Y, and Z. Each point in the three - dimensional coordinate space is represented by an X, Y, and Z coordinate value. Similarly, each cell within a cube is represented by dimension members. In the illustration shown in Figure 3 - 4 , you can see the three dimensions: Product, Customer, and Date. Assume that each of these dimensions has exactly one hierarchy as is shown in Figure 3 - 4 , namely, Product Line, Country, and Calendar. From Figure 3 - 4 you can see that Product Line has four members, Calendar has four members (considering only quarters), and Country has six members. Therefore the number of cells is equal to 4*4*6 = 96 cells.

Now that you have learned what a cell is, you need to understand how to retrieve data from it. Assume you want to retrieve the data shown by the shaded area in the cube. The Sales amount value in this cell is 966. This cell is located at the intersection of Product=Mountain, Date=Quarter2, and Customer=Australia. To retrieve data from the cube you need to send an MDX query to Analysis Services. The query needs to retrieve the " Internet Sales Amount " from the Cube based on the conditions that uniquely identify the cell that contains the value 966. That MDX query is:

```
SELECT Measures.[Internet Sales Amount] ON COLUMNS
FROM [Adventure Works]
WHERE ( [Date].[Calendar].[Calendar Quarter]. & [2003] & [2],
[Product].[Product Line].[Mountain],
[Customer].[Country].[Australia] )
```

You can see from this query that you are selecting the Measures.[Internet Sales Amount] value from the Adventure Works cube based on a specific condition mentioned in the WHERE clause of the MDX query. That condition uniquely identifies the cell. All you have done in the condition is list the members (which you learned about in the previous section) that uniquely identify the cell, separated by commas. An MDX expression like this that uniquely identifies a cell is called a tuple.

Tuples

As you saw in the previous section, a *tuple* uniquely identifies a cell or a section of a cube. A tuple is represented by one member from each dimension, separated by a comma, and is enclosed within parentheses. A tuple does not necessarily have to explicitly contain members from all the dimensions in the cube. Some examples of tuples based on the Adventure Works cube are:

1. ([Customer].[Country].[Australia])
2. ([Date].[Calendar].[2004].[H1 CY 2004].[Q1 CY 2004], [Customer].[Country].[Australia])
3. ([Date].[Calendar].[2004].[H1 CY 2004].[Q1 CY 2004], [Product].[Product Line].[Mountain], [Customer].[Country].[Australia])

In the preceding examples, tuples 1 and 2 do not contain members from all the dimensions in the cube. Therefore they represent sections of the cube. A section of the cube represented by a tuple is called a *slice* because you are slicing the cube to form a section (slice) based on certain dimension members.
When you refer to the tuple ([Customer].[Country].[Australia]) you actually refer to the sixteen cells that correspond to the country Australia in the example shown in Figure 3 - 4 . Therefore when you retrieve the data held by the cells pointed to by this tuple you are actually retrieving the Internet Sales Amount of all the customers in Australia. The Internet Sales Amount value for the tuple [Customer].[Country].[Australia] is an aggregate of the cells encompassed in the front face of the cube. The MDX query to retrieve data represented by this tuple is :

```
SELECT Measures.[Internet Sales Amount] ON COLUMNS
FROM [Adventure Works]
WHERE ([Customer].[Country].[Australia])
```

The result of this query is $9,061,000.58.

following tuples:

1. ([Date].[Calendar].[2005].[H1 CY 2004].[Q1 CY 2004], [Product].[Product Line].[Mountain], [Customer].[Country].[Australia])
2. ([Product].[Product Line].[Mountain], [Customer].[Country].[Australia], ([Date].[Calendar].[2005].[H1 CY 2004].[Q1 CY 2004])
3. ([Customer].[Country].[Australia], [Date].[Calendar].[2005].[H1CY 2005].[Q1CY 2005], [Product].[Product Line].[Mountain])

are equivalent and uniquely identify just one cell. Because a tuple uniquely identifies a cell, it cannot contain more than one member from each dimension.
A tuple represented by a single member is called a *simple tuple* and does not have to be enclosed within parentheses. ([Customer].[Country].[Australia]) is a simple tuple and can be referred to as [Customer].[Country].[Australia] or simply Customer.Country.Australia. When there is more than one dimension in a tuple, it needs to be enclosed in parentheses. A collection of tuples forms a new object called a set. Sets are frequently used in MDX queries and expressions.

Sets

An MDX *set* is a collection of tuples that are defined using the exact same set of dimensions, both in type and number. In the context of Analysis Services 2008, a set of dimensions will actually be a set of hierarchies in your MDX expressions or queries. Hence we refer to hierarchies in this section and throughout the book. A set is specified within curly brace characters ({ and }). Set members are separated by commas. The following examples illustrate sets:
Example 1: The tuples (Customer.Country.Australia) and (Customer.Country.Canada) are resolved to the exact same hierarchy Customer.Country. A collection of these two tuples is a valid set and is specified as:

```
{(Customer.Country.Australia), (Customer.Country.Canada)}
```

Example 2: The tuples (Customer.Country.Australia, [Product].[Product Line].[Mountain]) and (Customer.Country.Canada, [Date].[Calendar].[2004].[H1 CY 2004].[Q1 CY 2004]) cannot be combined to form a set. Even though they are formed by two hierarchies, the dimensions used to resolve the tuple are different. Both tuples have the Customer.Country hierarchy dimension but the second hierarchies are different.
Example 3: Each of the following tuples has the three dimensions Date, Product, and Customer:

```
1. ([Date].[Calendar].[2004].[H1 CY 2004].[Q1 CY 2004], [Product].
[Product Line].[Mountain], [Customer].[Country].[Australia]),
2. ([Product].[Product Line].[Mountain], [Customer].[Country].[Australia],
([Date].[Calendar].[2002].[H1 CY 2002].[Q1 CY 2002])
3. ([Customer].[Country].[Australia], [Date].[Calendar].[2003].
[H1 CY 2003].[Q1 CY 2003], [Product].[Product Line].[Mountain] )
```

The members in the Date.Calendar hierarchy of the three preceding tuples are different and therefore these tuples refer to different cells. As per the definition of a set, a collection of these tuples is a valid set and is shown here:

```
{ ([Date].[Calendar].[2004].[H1 CY 2004].[Q1 CY 2004], [Product].[Product
Line].[Mountain], [Customer].[Country].[Australia]), ([Product].[Product
Line].[Mountain], [Customer].[Country].[Australia],

([Date].[Calendar].[2002].[H1 CY 2002].[Q1 CY
2002]),([Customer].[Country].[Australia], [Date].[Calendar].[2003].[H1 CY
2003].[Q1 CY 2003], [Product].[Product Line].[Mountain] )}
```

A set can contain zero, one, or more tuples. A set with zero tuples is referred to as an empty set. An empty set is represented as :

{ }

A set can contain duplicate tuples. An example of such a set is :

```
{Customer.Country.Australia, Customer.Country.Canada,
Customer.Country.Australia}
```

This set contains two instances of the tuple Customer.Country.Australia. Because a member of a dimension by itself forms a tuple, it can be used as such in MDX queries. Similarly, if there is a tuple that is specified by only one hierarchy, you do not need the parentheses to specify it as a set. When there is a single tuple specified in a query you do not need curly braces to indicate it should be treated as a set. When the query is executed, the tuple is implicitly converted to a set.

Now that you have learned the key concepts that will help you understand MDX better, the following section dives right into MDX query syntax and the operators used in an MDX query or an MDX expression.

MDX Queries

Chapter 2 introduced you to the MDX SELECT statement. The syntax for an MDX query is as follows:

```
[WITH < formula_expression > [, < formula_expression > ...]]
SELECT [ < axis_expression > , [ < axis_expression > ...]]
FROM [ < cube_expression > ]
[WHERE [slicer_expression]]
```

You might be wondering whether the SELECT , FROM , and WHERE clauses are the same as those in Structured Query Language (SQL). Even though they look identical to those in SQL, the MDX language is different and supports more complex operations. You learn about some of these operations in this chapter and throughout the book.

The keywords WITH , SELECT , FROM , and WHERE along with the expressions following them are referred to as a *clauses* . In the preceding MDX query template, anything specified within square brackets means it is optional; that is, that section of the query is not mandatory in an MDX query.

You can see that the WITH and WHERE clauses are optional because they are enclosed within square brackets. Therefore, you might be thinking that the simplest possible MDX query should be the following:

```
SELECT
FROM [Adventure Works]
```

Super! You are absolutely correct. This MDX query returns a single value. Which value, you might ask? Recall that fact data is stored in a special dimension called Measures. When you send the preceding query to the Analysis Services instance, you get the value of the default member from the Measures dimension which, for the Adventure Works cube, is Reseller Sales Amount from the Reseller Sales measure group. The result of this query is the aggregated value of all the cells in the cube for this measure for the default values of each cube dimension.

The WITH clause is typically used for custom calculations and operations, and you learn about this later in this chapter. First, though, let ' s take a look at the SELECT , FROM , and WHERE clauses.

The SELECT Statement and Axis Specification

The MDX SELECT statement is used to retrieve a subset of the multidimensional data in an OLAP cube. In SQL, the SELECT statement allows you to specify which columns will be included in the row data you retrieve, which is viewed as two - dimensional data. If you consider a two - dimensional coordinate system, you have the X and Y axes. The Y axis is used for the COLUMNS and the X axis is used for ROWS. In MDX, the SELECT statement is specified in a way that allows retrieving data with more than just two dimensions. Indeed, MDX provides you with the capability of retrieving data on one, two, or many axes.

The syntax of the SELECT statement is :

```
SELECT [ < axis_expression > , [ < axis_expression > ...]]
```

The axis_expressions specified after the SELECT refer to the dimension data you are interested in retrieving. These dimensions are referred to as axis dimensions because the data from these dimensions are projected onto the corresponding axes. The syntax for axis_expression is :

```
< axis_expression > := < set > ON (axis | AXIS(axis number) | axis number)
```

Axis dimensions are used to retrieve multidimensional result sets. The set, a collection of tuples, is defined to form an axis dimension. MDX provides you with the capability of specifying up to 128 axes in the SELECT statement. The first five axes have aliases. They are COLUMNS, ROWS, PAGES, SECTIONS, and CHAPTERS. Axes can also be specified as a number, which allows you to specify more than five dimensions in your SELECT statement. Take the following example:

```
SELECT Measures.[Internet Sales Amount] ON COLUMNS,
[Customer].[Country].MEMBERS ON ROWS,
[Product].[Product Line].MEMBERS ON PAGES
FROM [Adventure Works]
```

Three axes are specified in the SELECT statement. Data from dimensions Measures, Customers, and Product are mapped on to the three axes to form the axis dimensions. This statement could equivalently be written as :

```
SELECT Measures.[Internet Sales Amount] ON 0,
[Customer].[Country].MEMBERS ON 1,
[Product].[Product Line].MEMBERS ON 2
FROM [Adventure Works]
```

Axis Dimensions

The axis dimensions are what you build when you define a SELECT statement. A SELECT statement specifies a set for each dimension; COLUMNS, ROWS, and additional axes — if you have them. Unlike the slicer dimension (described later in this chapter), axis dimensions retrieve and retain data for multiple members, not just single members. Please note that when we refer to axis dimension, this actually corresponds to a hierarchy for Analysis Services 2008 because you include hierarchies in the MDX statement.

No Shortcuts! In MDX you cannot create a workable query that omits lower axes.
If you want to specify a PAGES axis, you must also specify COLUMNS and ROWS.

The FROM Clause and Cube Specification

The FROM clause in an MDX query determines the cube from which you retrieve and analyze data. It ' s similar to the FROM clause in a SQL query where you specify a table name. The FROM clause is a necessity for any MDX query. The syntax of the FROM clause is :

```
FROM < cube_expression >
```

The cube_expression denotes the name of a cube or a subsection of a cube from which you want to retrieve data. In SQL ' s FROM clause you can specify more than one table, but in an MDX FROM clause you can define just one cube name. The cube specified in the FROM clause is called the *cube context* and the query is executed within this cube context. That is, every part of axis_expressions are retrieved from the cube context specified in the FROM clause:

```
SELECT [Measures].[Internet Sales Amount] ON COLUMNS
FROM [Adventure Works]
```

This is a valid MDX query that retrieves data from the [Internet Sales Amount] measure on the X - axis. The measure data is retrieved from the cube context [Adventure Works]. Even though the FROM clause restricts you to working with only one cube or section of a cube, you can retrieve data from other cubes using the MDX LookupCube function. When there are two ore more cubes having common dimension members, the LookupCube function retrieves measures outside the current cube ' s context using the common dimension members.

The WHERE Clause and Slicer Specification

In pretty much any relational database work that you do, you issue queries that return only portions of the total data available in a given table, set of joined tables, and/or joined databases. This is accomplished using SQL statements that specify what data you do and do not want returned as a result of running your query. Here is an example of an unrestricted SQL query on a table named Product that contains sales information for products:

```
SELECT *
FROM Product
```

Assume the preceding query results in five columns being retrieved with the following four rows:

The * represents “ all, ” meaning that query will dump the entire contents of the table. If you want to know only the Color and Product Line for each row, you can restrict the query so that it returns just the information you want. The following simple example demonstrates a query constructed to return just two columns from the table:

```
SELECT ProductLine, Color
FROM Product
```

This query returns the following:

```
Product Line Color
Accessories Silver
Mountain Grey
Road Silver
Touring Red
```

The concept of crafting queries to return only the data you need maps directly to MDX from SQL. In fact, they share a conditional statement that adds a whole new level of power to restricting queries to return only desired data. It is called the *WHERE* clause. After taking a look at the SQL WHERE clause you will see how the concept is similar to its use in MDX. Here is a SQL query that uses WHERE to restrict the returned rows to those products whose color is silver:

```
SELECT ProductLine, Sales
FROM Product
WHERE Color = ‘Silver’
```

This query returns the following:

```
Product Line Sales
Accessories 200.00
Road 2500.00
```

The same concept applies to MDX. The MDX SELECT statement is used to identify the dimensions and members a query will return and the WHERE statement limits the result set by some criteria. The preceding SQL example restricts the returned data to records where Color = ‘ Silver ’ . Note that in MDX members are the elements that make up a dimension ’ s hierarchy. The Product table, when modeled as a cube, will contain two measures, Sales and Weight, and a Product dimension with the hierarchies ProductID, ProductLine, and Color. In this example the Product table is used as a fact as well as a dimension table. An MDX query against the cube that produces the same results as that of the SQL query is :

```
SELECT Measures.[Sales] ON COLUMNS,
[Product].[Product Line].MEMBERS on ROWS
FROM [ProductsCube]
WHERE ([Product].[Color].[Silver])
```

The two columns selected in SQL are now on the axes COLUMNS and ROWS. The condition in the SQL WHERE clause, which is a string comparison, is transformed to an MDX WHERE clause, which refers to a slice on the cube that contains products that have silver color. As you can see, even though the SQL and MDX queries look similar, their semantics are quite different.

The Slicer Dimension

The *slicer dimension* is what you build when you define the WHERE statement. It is a filter that removes unwanted dimensions and members. As mentioned earlier in this chapter, in the context of Analysis Services 2008, the dimensions will actually be hierarchies in Analysis Services 2008. What makes things

included in any of the queried axes. The default members of hierarchies not included in the query axes are used in the slicer axis. Regardless of how it gets its data, the slicer dimension will only accept MDX expressions (described later in this chapter) that evaluate to a single set. When there are tuples specified for the slicer axis, MDX will evaluate those tuples as a set and the results of the tuples are aggregated based on the measures included in the query and the aggregation function of that specific measure.