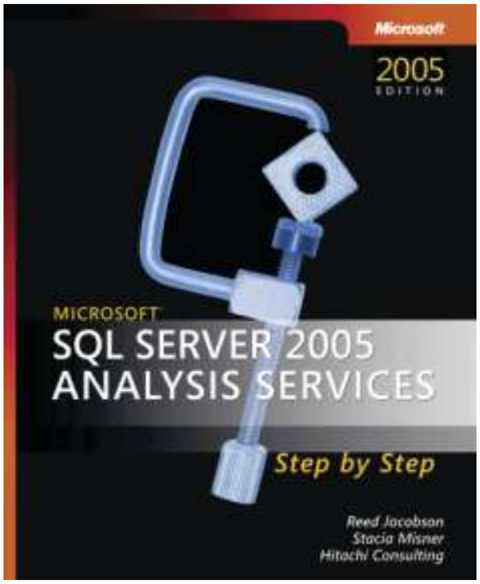

Microsoft® SQL Server™ 2005 Analysis Services Step by Step

Reed Jacobson; Stacia Misner; Hitachi Consulting

To learn more about this book, visit Microsoft Learning at
http://www.microsoft.com/MSPress/books/8592.aspx

9780735621992
Publication Date: March 2006

Microsoft Press

Table of Contents

Chapter 2

Understanding OLAP and Analysis Services

After completing this chapter, you will be able to:

- Understand the definition of OLAP and the benefits an OLAP tool can add to a data warehouse.
- Understand how Microsoft SQL Server Analysis Services 2005 implements OLAP.
- Understand tools for developing and managing an Analysis Services database.

Business intelligence (BI) is a way of thinking. A data warehouse is a general structure for storing the data needed for good BI. But data in a warehouse is of little use until it is converted into the information that decision makers need. The large relational databases typical of data warehouses need additional help to convert the data into information. In this chapter, you will first learn the general benefits of online analytical processing (OLAP)—one of the best technologies for converting data into information—and then you will learn about how Microsoft Analysis Services implements the benefits of OLAP.

Understanding OLAP

The first version of Analysis Services was named OLAP Services. Even though the name now reflects the purpose of the product, rather than the technology, the technology is still important. Understanding the history of the term OLAP can help you understand its meaning.

In 1985, E. F. Codd coined the term *online transaction processing* (OLTP) and proposed 12 criteria that define an OLTP database. His terminology and criteria became widely accepted as the standard for databases used to manage the day-to-day operations (transactions) of a company. In 1993, Codd came up with the term *online analytical processing* (OLAP) and again proposed 12 criteria to define an OLAP database. This time, his criteria did not gain wide acceptance, but the term OLAP did, seeming perfect to many for describing databases designed to facilitate decision making (analysis) in an organization.

Some people use OLAP simply as a synonym for dimensional data warehousing. Usually, however, the term OLAP describes specialized tools that make warehouse data easily accessible. One term that is almost always associated with OLAP—but never associated with relational databases—is the word *cube*. As you learned in the previous chapter, the term *dimension* was appropriated from geometry for use in a relational warehouse. In a similar way, OLAP borrowed the word *cube* to describe what in the relational world would be the integration of the fact table with dimension tables. In geometry, a cube has three dimensions. In OLAP, a cube can have anywhere from one to however many dimensions you need. The word does make some sense because, in geometry, you calculate the size of the cube by multiplying the size of each of the three dimensions. Likewise, in OLAP, you calculate the theoretical maximum size of a cube by multiplying the size of each of the dimensions. Different OLAP tools define, store, and manage cubes differently, but when you hear the word "cube," you're in the OLAP world.

So what is the benefit of an OLAP cube over a relational database? Typically, OLAP tools add the following three benefits to a relational database:

- Consistently fast response
- Metadata-based queries
- Spreadsheet-style formulas

Before looking specifically at Analysis Services, consider how OLAP in general provides these benefits.

Consistently Fast Response

One of the ways that OLAP obtains a consistently fast response is by prestoring calculated values. Basically, the idea is that you either pay for the time of the calculation at query time or you pay for it in advance. OLAP allows you to pay for the calculation time in advance. In terms of how data is physically stored, OLAP tools fall into two basic types: a spreadsheet model and a database model. Analysis Services storage is basically the database model, but it will be useful for you to understand some of the issues and benefits of a spreadsheet model OLAP.

- **Spreadsheet model OLAP** In a spreadsheet, you can insert a value or a formula into any cell. Spreadsheets are very useful for complex formulas because they give you a great deal of control. One problem with spreadsheets is that they are limited in size, and a spreadsheet is essentially a two-dimensional structure. An OLAP cube built using a spreadsheet storage model expands the model into multiple dimensions, and can be much larger than a regular spreadsheet. With OLAP based on a spreadsheet model, any cell in the entire cube space has the potential to be physically stored. That is both a good thing and a bad thing. It's a good thing because you can enter constant values at any point in the cube space, and you can also store the results of a calculation at any point in the cube space. It's a bad thing because it limits the size of the OLAP cube due to a little problem called data explosion.

You have perhaps heard the story of the man who invented chess. He lived in India, and according to legend, his name was Sessa. The king of India was very impressed with the game of chess and asked Sessa to name his reward. Sessa's request was so modest that it offended the king: He asked simply for one grain of rice for the first square of his chess board, two grains for the second square, four grains for the third, and so forth, doubling for each of the 64 squares of the board. Of course, by the time the king's magicians calculated the total amount of rice needed to pay the reward, they realized that—had they known the metric system and the distance to the sun—it would require a warehouse 3 meters by 5 meters by twice the distance to the sun to pay the reward. In one version of the legend, the king simply solved the problem by cutting off Sessa's head. In another version, the king was more noble and also more clever. He gave Sessa a sack, pointed him to the warehouse and told him to go count out his reward—no rush.

The problem Sessa gave the king was the result of a geometric progression: When numbers increase geometrically, they get very large very quickly, and the size of a cube increases geometrically with the number of dimensions. That is the problem with OLAP stored using a spreadsheet model. Because any cell in cube space has the potential for being stored physically, data explosion becomes a very real problem that must be managed. The more dimensions you include in the cube, and the more members in each dimension, the greater the data explosion potential. Spreadsheet-based OLAP tools typically have elaborate—and complicated—techniques for managing data explosion, but even so, they are still very limited in size.

Spreadsheet-based OLAP tools are typically associated with financial applications. Most financial applications involve relatively small databases coupled with complex, nonadditive calculations.

- **Database model OLAP** OLAP tools that store cube data by using a database model behave very differently. They take advantage of the fact that most reporting requires addition, and that addition is an associative operation. For example, when adding the numbers 3, 5, and 7, it doesn't matter whether you add 3 and 5 to get 8 before adding the 7, or whether you add 5 and 7 to get 12 before adding 3. In either case, the final answer is 15. In a purely relational database, you can get fast query results by creating aggregate tables. In an aggregate table, you presummarize values that will be needed in a report. For example, in a fact table that includes thousands of products, five years of daily data, and perhaps several other dimensions, you may have millions of rows in the fact table, requiring many minutes to generate a report by product subcategory and by quarter, even if there are only 50 subcategories and 20 quarters. But if you presummarize the data into an aggregate table that includes only subcategories and quarters, the aggregate table will have at most one thousand rows, and a report requesting totals by subcategory and by quarter will be extremely fast. In fact, because of the associative nature of addition, a report requesting totals by category and by year can use the same aggregate table, again producing the results very quickly.

Perhaps the biggest benefit of OLAP stored using the database model is the ability to avoid data explosion. Because you need relatively few aggregate tables to provide fast results, you can have much larger cubes with many more dimensions and attributes than by using a spreadsheet model. Perhaps the biggest disadvantage of OLAP stored by using a database model is that there is no inherent way to physically store values that are calculated using nonassociative operators. An extreme example of a difficult financial calculation is Retained Earnings Since Inception. To calculate this value, you must first calculate Net Income—itself a hodgepodge of various additions, subtractions, and multiplications. And you must calculate Net Income for every period back to the beginning of time so that you can sum them together. This is not an associative calculation, so calculating for all of the business units does not make it any easier to calculate the value for the total company.

Even OLAP cubes that are stored by using the database model can calculate some nonassociative values very quickly. For example, an Average Selling Price is not an additive value—you can't simply add prices together. But to calculate the Average Selling Price for an entire product line, you simply sum the Sales Amount and Sales Quantity across the product line, and then, at the product line level, you divide the total Sales Amount by the total Sales Quantity. Because you are calculating a simple ratio of two additive values, the result is essentially just as fast as retrieving a simple additive value.

Database-style OLAP tools are usually associated with sales or similar databases. Sales cubes are often huge—both with hundreds of millions of fact-table rows, and with multiple dimensions with many attributes. Sales cubes also often involve additive measures (dollars and units are generally additive) or formulas that can be calculated quickly based on additive values.

One of the major benefits of OLAP is the ability to precalculate values so that reports can be rendered very quickly. Different OLAP technologies may have different strengths and weaknesses, but a good OLAP implementation will be much faster than the equivalent relational query whenever highly summarized values are involved.

Metadata-Based Queries

When you write queries against a relational data source, you use Structured Query Language (SQL). SQL is an excellent language, but it was developed primarily for transaction systems, not for reporting applications. One of the problems with SQL is not the language itself, but the fact that the database provides relatively little information about itself. Information about how the data is stored and structured, and perhaps more importantly, what the data means, is called *metadata*. Relational databases contain a small amount of metadata, but most of the information about the database has to come from you—the person writing the SQL query.

An OLAP cube, on the other hand, contains a great deal of metadata. For example, when you create an OLAP cube, you define not only what the measures are, but also how they should be aggregated, what the caption should be, and even how the number should best be formatted. Likewise, in an OLAP cube, when you create a dimension with many attributes, you define which attributes are groupable, and whether any of the groupable attributes should be linked together into a hierarchy. Unfortunately, SQL is not able to take advantage of this metadata as you create queries.

Consequently, when you use an OLAP data source, you use a different query language, most likely multidimensional expressions, or MDX. MDX was originally developed by Microsoft, and many OLAP vendors have their own proprietary query languages. But in 2001, Microsoft, Hyperion, and SAS formed the XML for Analysis (XMLA) council to formulate a common specification for working with OLAP data sources. The query language chosen for the XMLA specification is MDX. Most major OLAP vendors have joined the XMLA council and now have XMLA providers. (For more information about XMLA, check out the council's Web site at *www.xmla.org*.)

In this section, you will be introduced to some of the benefits of MDX as a metadata-based query language. You don't need to try to learn the details of how to write MDX; you'll learn more about MDX specifics in a later chapter. Everything you learn about MDX queries in this book definitely applies to Microsoft Analysis Services. Most of it will also apply to most other OLAP providers, but some of the details may be different.

One of the key benefits of a query language that can work with the metadata of an OLAP source is that you can use a general-purpose browser to query a specific data source. For example, with a Microsoft Analysis Services cube, you can choose to use Microsoft client tools such as those included in Microsoft Office, or you can choose tools from any of dozens of other vendors. Any client tool that uses MDX or XMLA can understand your cube and generate meaningful reports without the need for you to create custom queries. In other words, because MDX query statements are based on metadata stored in the OLAP cube, you can probably use a tool that will generate the query for you, and you won't have to write any MDX query statements at all.

If you do have a reason for writing custom MDX queries, the metadata makes it much easier than writing SQL queries. As a simple example, in SQL, if you create a query that calculates the total Sales Units for each customer's City, you still need to add a clause to make sure that the cities are sorted properly; but in an MDX query, you simply state that you want the members of the City attribute and you automatically get the default sort order as defined in the metadata. As another example, in a SQL table that contains both Country and City columns, there is nothing to suggest that Cities belong to specific countries, so if you want to show all the cities from Germany, you have to explicitly include the fact the you want to filter by Germany but show cities; in an OLAP cube, where Country is defined as the parent of City, you can specify the query using the expression *[Germany].Children*. In fact, if you later inserted a Region attribute between Country and City, the MDX query would automatically return the regions in Germany, based on the hierarchical relationships defined in the metadata.

These are just a taste of the kind of benefits MDX brings to the area of reporting queries. Many other kinds of reporting queries that are difficult in SQL—such as a cross-tabulation that shows the best-selling products as column headings and the best-selling regions as row headings—are very simple by using MDX queries. Some reports that are simply impossible in SQL—such as nesting multiple layers of attributes as column headings—are also very simple by using MDX queries.

Spreadsheet-Style Formulas

Arguably half the world's businesses are managed by using spreadsheets. Spreadsheets are notoriously decentralized, error-prone, difficult to consolidate, and impossible to manage. So why are they such a key component of business management? Because spreadsheet formulas are intuitive to create. To calculate the percentage of the total for a given product, you point at the product cell, add a division sign (/), point at the total cell, and you're done. With a little fiddling with the formula, you can copy it to calculate the percentage for any product. When you're creating the percentage formula, you don't need to worry about how the total got calculated; you solved that with a different formula, so now you can simply use the result. The same is true for other formulas such as month-to-month growth, or growth from the same month of the previous year and many other useful analytical formulas. Many very useful formulas that would be very difficult to create using pure relational SQL queries are easy to create in a spreadsheet.

But even from a spreadsheet user's perspective, formulas have inherent problems. A spreadsheet formula is inherently two-dimensional: You have numbers for rows and letters for columns. If you need to replicate the same spreadsheet for a different time period—particularly one in which there are different products or different dates—it is cumbersome to modify the formulas. And it is easy to make mistakes: There is nothing about the reference C12 that reassures you that you are indeed getting the value for March and not for April. As formulas become long and complex, it can be difficult even for the original creator to figure out what the formula really means. In addition, you can easily replace a formula in the middle of a range with an "adjusted" formula, or a constant value, and then forget that you made the change.

From a management perspective, spreadsheet formulas have even bigger problems: The formulas in a spreadsheet are key "business logic," and yet they are spread out all over the organization. The growth calculation created by Rajif may have some subtle differences from the one created by Sayoko, even though they ostensibly (and apparently) use the same logic.

Formulas in OLAP cubes have many of the same benefits as a spreadsheet formula: While creating a formula, you can reference any cell in the entire cube without concern for how that value was calculated.

Most OLAP providers have their own proprietary formula languages. Even providers who support MDX queries as part of the XMLA specification may not support the full potential of MDX formulas. Microsoft Analysis Services has a very rich implementation of MDX formulas. Here are a few examples of ways that MDX formulas are even easier than spreadsheet formulas:

- References in a spreadsheet formula are cryptic. In MDX, formulas can have meaningful names in references. Thus, instead of =C14/D14, the formula might be [Actual]/[Budget].
- In a spreadsheet, a formula must be explicitly copied to each cell that needs it. In MDX, a formula is defined generically, so that switching a report to show 500 products instead of just 50 requires you to make sure that the formulas apply properly to the new rows.

Likewise, if you create a new worksheet—say, for a new region—you must make sure that the formulas on the new worksheet point to the proper cells. In MDX, switching to a new region automatically uses the same generic formula.

- The nature of a spreadsheet reference is two-dimensional, with a letter for the column and a number for the row. This inherently limits the number of dimensions you can easily incorporate into a formula. MDX references use a structure (similar to that used for geometric coordinates) that is not tied to a two-dimensional physical location, and can explicitly include dozens of dimensions, if necessary. In addition, an MDX reference simplifies the use of multiple dimensions by taking advantage of the concept of a "current" member. For example, in the same way that copying the formula =C14/D14 to multiple sheets in a single workbook automatically uses the values from cells on the current sheet, using the MDX formula [Actual]/[Budget] automatically uses the current time period, or the current department, or the current product.
- A spreadsheet formula has no knowledge of the logical relationships between other cells; it has no knowledge of metadata. MDX formulas, on the other hand, can take advantage of a cube metadata to calculate relationships that would be difficult in a spreadsheet. For example, in a spreadsheet, it is easy to calculate the percentage each product contributes to the grand total, but it is very difficult to calculate the percentage each product contributes to its product group. In MDX, because the metadata can include information about hierarchical relationships, calculating the Percent of Parent within a product hierarchy is very easy.
- A spreadsheet formula can only refer to values that are on the same worksheet (or perhaps another worksheet in the same workbook). An MDX formula has access to any value anywhere in the cube space. This allows you to create *bubble-up* or *exception* formulas. An example of a bubble-up exception formula would be a report that shows the total sales at the region level, but displays the value in red if any of the districts within the region is significantly lower than its target. It does this even though the districts don't appear on the report.

This is just a taste of the ways that an MDX formula can be more powerful than a simple spreadsheet formula. In addition, MDX formulas are stored on the server, putting business logic into a centralized, manageable location, rather than spreading the business logic across hundreds of independent spreadsheets.

Understanding Analysis Services

You don't need Analysis Services to create a data warehouse; you create a data warehouse in a relational database. Even if you want to add the benefits of OLAP, you can choose any of several OLAP vendors. So why use Analysis Services for OLAP? Some people say that Microsoft products are popular because they have an inexpensive licensing model. But buying a cheap tool can be an expensive mistake. For something as important as BI, you want to be sure that the tools you use are the best you can use. So what makes Microsoft SQL Server 2005 Analysis

Services a good choice? In order to answer that, you need to understand some of the fundamental architecture of Analysis Services. In the first half of this chapter, you learned three major benefits of OLAP technology. Now you will learn how Analysis Services implements those three main benefits.

Analysis Services and Speed

Speed comes from precalculating values. Querying a 100-million-row table for a grand total is going to take much more time than querying a 100-row summary table. Because most very large data warehouse databases use addition for *aggregations*, Analysis Services stores data in a database style, using the equivalent of summary tables for aggregations. Of course, it can store the data in a special format that is particularly efficient for storage and retrieval, but conceptually, creating aggregations in Analysis Services is the same as creating summary tables in a relational database. Because the values are additive (or similar), you don't need to create a space for every possible value. Rather, you create "strategic" aggregations, so that relatively few aggregations can support hundreds or thousands of possible types of queries.

The biggest problem with creating summary tables in a relational data warehouse is that there is an incredible amount of administrative work involved.

- First, you must decide which of the potential millions of possible aggregate tables you will actually create.
- Second, you must create, populate, and update the aggregate tables.
- Finally, you must change reports to use the appropriate aggregate tables.

Each one of these steps is a major undertaking. Analysis Services basically takes care of all of them for you. (You can do some tuning, but the process is essentially automatic.) Analysis Services has sophisticated tools to simplify the process of designing, creating, maintaining, and querying aggregate tables, which it then stores in its extremely efficient proprietary structures. Managing aggregations has always been an extremely strong feature of Analysis Services. Because of its ability to avoid data explosion issues, Analysis Services can handle extremely large—multiterabyte—databases.

Analysis Services and Metadata

Analysis Services in SQL Server 2005 has significantly re-architected the way that metadata is defined—both for dimensions and for cubes.

Dimension Metadata

Consider a Customer dimension. In a relational data warehouse, you would typically have a table with a primary key—one that uniquely identifies each customer. Then you have a number of attributes that relate to that customer. For example, you might have Street Address, City, Country, Region, Age, Age Group, Gender, and potentially many other attributes. In Analysis

Services 2005, you simply define the dimension as a key with attributes. The metadata matches the logic of the data.

Some attributes—such as Street Address—will never be used for grouping or selecting customers, so you flag them in the metadata.

Some attributes—such as Gender—can be used for grouping on a report, and can also be added into a total, which essentially ignores the attribute. This is the automatic, default behavior of an attribute in Analysis Services. A single-level groupable attribute is called an *attribute hierarchy*.

A single dimension can have many attribute hierarchies. Again, the metadata matches the logic of the data.

Some attributes form a natural hierarchy. For example, each customer has an age, and each age belongs to an age group. Analysis Services allows you to create a multilevel hierarchy of attributes that reflects this relationship. A customer might belong to multiple hierarchies. For example, in your organization, you might have each customer belong to a city, which belongs to a country, which then belongs to a region. In Analysis Services, you can define multiple multilevel hierarchies from attributes in a single dimension—again, making the metadata match the logic of the data.

In previous versions of Analysis Services, each hierarchy essentially became a separate dimension, even though they all came from the same underlying relational dimension. In Analysis Services 2005, all the attributes and hierarchies of a logical dimension belong to that dimension in the Analysis Services dimension. In fact, even without creating multilevel hierarchies, if you nest attributes on a report—putting, for example, Gender and then Age Group on the rows of a report—Analysis Services automatically recognizes the combinations that actually exist in the dimension and ignores any that do not. This allows incredible flexibility in reporting without hurting query performance.

Cube Metadata

Suppose you decide to design a cube before you create the data warehouse to support it—which, incidentally, you can do in Analysis Services 2005. First, you select a measure—say, Sales Amount. Next, decide what dimensions you would like for that measure, and at what level of detail—say, Product by Customer, by Date. This defines the *grain* for the measure. Finally, decide if there are any other measures that have the same grain—perhaps Sales Units. You would then create a *measure group* that contains all the measures that have the same dimensions at the same grain.

Suppose you select a new measure requiring a different grain. For example, suppose you want Sales Target to have product categories by calendar quarter by scenario. This measure does not have the same grain as Sales Amount and Sales Quantity, so you create a new measure group. If there are any other measures that require the same grain as Sales Target, you can add them to the same measure group.

A measure group is simply the *group* of *measures* that share the same grain. When you go to build your data warehouse, you would create a separate fact table for each measure group. Conversely, if you already have a data warehouse with several fact tables, you simply create a measure group for each fact table.

A *cube* is then the combination of all the measure groups. This means that a single cube can contain measures with different grains. This pushes the meaning of *cube* even further from its geometrical origins. Perhaps you can visualize a cube as a cluster of crystals of varying sizes and shapes, many of which share common sides. In this new way of thinking, a single cube can contain all the metadata for all the data in your data warehouse. Because of this, a cube is now sometimes called a *Unified Dimensional Model,* or UDM. Sometimes a cube has more information than is manageable by a single person. For example, a procurement manager may not care about how sales discounts are applied. Analysis Services allows you to create a *perspective* that is like a cube that contains only a subset of the measures and dimensions of the whole cube. You can create as many perspectives as you want within a cube.

A cube is a logical structure, not a physical one. The same is true for a measure group. It defines the metadata so that client tools can access the data. You define measures and dimensions, and specify how measures should be aggregated across the dimensions.

Conceptually, each measure group contains all the detail values stored in the fact table, but that doesn't mean that the measure group must physically copy all the detail values from the fact table. If you choose, you can make the measure group dynamically retrieve values as needed from the fact table. In this case, you're using the measure group only to define metadata. This is called relational OLAP, or ROLAP. For faster query performance, you can tell the measure group to copy the detail values into a proprietary structure that allows for extremely fast retrieval. This is called multidimensional OLAP, or MOLAP. Analysis Services allows you, as the cube designer, to decide whether to store the values as MOLAP or ROLAP. Aside from performance differences, where the detail values are physically stored is completely invisible to a user of a cube. Whether you use MOLAP or ROLAP, values are stored in a memory *cache*—on a space-available basis—to make subsequent queries faster. You can think of MOLAP storage as a disk-based cache that allows the Analysis Server to load the memory cache much faster than if it had to go to the relational database.

Analysis Services Formulas

Even without any explicit formulas, an Analysis Services cube contains many calculations—the totals that aggregate up the hierarchies in each dimension are calculations, and they happen automatically. If you create a cube that consists primarily of additive measures—for example, a cube that summarizes sales or other transactions—the basic cube engine does most of the calculation work. When you create MOLAP aggregations, Analysis Services physically stores the values needed to query sum, count, min, and max calculations extremely quickly. In addition, you can create *calculated members* that perform calculations on aggregated values. Calculated members make it easy to create values such as average prices, weighted averages, ratios, growth calculations, and other key performance indicators (KPIs) to analyze your data. In addition to including sophisticated built-in tools for creating calculated members, Analysis Services allows you to access external functions from Microsoft Visual Basic for Applications (VBA) or Microsoft Excel, or even write your own external functions.

Because a cube contains multiple measure groups, it is easy to create calculations that include measures from different fact tables. For example, you could calculate a percentage by dividing Sales Amount by Sales Target even though the two measures are in different measure groups.

Finance Formulas

Financial applications typically require much more sophisticated formulas than simple addition. This is one of the reasons spreadsheets are very popular for financial analysis. Analysis Services has special features to support financial analysis:

- **Unary operators** Most financial analysts expect expenses (which are really negative) to show up as positive numbers. Some accounts—such as the number of employees—are called memo accounts and should not be added or subtracted. Analysis Services provides a mechanism for properly managing these types of accounts.
- **Semiadditive calculations** Some measures are actually snapshots at a point in time. Typical examples include inventory quantities and bank account balances. These measures should be added up over all dimensions *except* time. Analysis Services supports semiadditive calculations.
- **By account aggregations** Sometimes a single measure should behave differently depending on what type of account it is. For example, a Revenues account should add up over time, but a Balance account should not. The By Account aggregation type allows you to have different aggregation definitions for different account types within a single measure.
- **Script assignments** For certain complex financial calculations, you need to change a value that would otherwise be calculated in the cube—and then allow that value to be re-aggregated within the normal dimension aggregation rules. You can think of it as changing a specific formula in a spreadsheet, even when other formulas depend on it. This was possible in Analysis Services 2000, but was very obscure and difficult. In Analysis Services 2005, the method for assigning formulas to portions of the cube has become much more simple and straightforward.

MDX formulas have always been very powerful for complex spreadsheet-like calculations. Even with the advent of XMLA for making MDX a standardized query language, Analysis Services has a much stronger implementation of MDX as a formula language than any other OLAP tool.

Analysis Services Tools

When you are responsible for an Analysis Services cube—or UDM—you perform two basic roles. On the one hand, you act as a developer—designing and creating the dimensions and cubes. On the other hand, you act as an administrator—keeping deployed cubes up-to-date and performing properly. In a large-scale implementation, it is common for these roles to be performed by different people, or even for multiple people to be involved in each part. Analysis Services in SQL Server 2005 recognizes that these are completely different roles and gives you two completely different tools for performing them.

For the developer, there is Business Intelligence Development Studio (BIDS). This is actually a copy of Visual Studio 2005, but with business intelligence designers installed instead of designers for C#.NET or VB.NET. If you use Visual Studio to write .NET applications, BIDS integrates smoothly with your existing installation. If you do not use Visual Studio for any other purpose, the Visual Studio shell, along with the business intelligence designers, is included with SQL Server 2005. Within BIDS, you can have multiple developers working on different parts of a single project, using XMLA to deploy the Analysis Services application to the development, test, or production server as appropriate. You can even integrate the project with Microsoft Visual Source Safe (VSS) so that you can safely manage the "source code" for an Analysis Services cube. If you want to automate either development or production tasks, you can use the .NET libraries in Analysis Management Objects (AMO), or you can use XMLA scripts.

Analysis Services 2005 is very effective at implementing the three benefits of OLAP. It uses a database model—with automatic management of aggregations—to handle extremely fast response from huge databases with little or no data explosion. It allows you to create a metadata model that accurately represents the true nature of both dimensions and cubes. And it supports a powerful implementation of the MDX formula language with capabilities that range from simple calculated ratios to complex financial calculations with sophisticated ripple effects. In essence, Analysis Services is simple enough for small, uncomplicated organizations, and powerful enough for large or complex organizations, allowing all types of organizations to add analytical power to their BI solutions.

Chapter 2 Quick Reference

This term	Means this
Aggregation	Summarized values of a measure
Cache	Server-based storage locations both in memory (automatic) or on disk (designed) that enhance query performance
Calculated member	A mechanism for aggregating measures using formulas more complex than those stored in a cube
Cube	A collection of one or more related measure groups and their associated dimensions
Cube metadata	Instructions for creating and querying OLAP structures such as cubes and dimensions
Hierarchy	Levels of aggregation within a single dimension
Measure group	The conceptual container of detail values from a single fact table, along with all possible aggregations for one or more dimension hierarchies
Online analytical processing (OLAP)	A database system optimized to support decision-making processes
Online transaction processing (OLTP)	A database system used to manage transactions such as order processing
Unified Dimensional Model (UDM)	The measure groups and dimensions that define your organization’s BI data; essentially synonymous with a cube