



Professional

Microsoft®

SQL Server®

**Analysis Services 2008
with MDX**

Sivakumar Harinath, Matt Carroll, Sethu Meenakshisundaram, Robert Zare, Denny Guang-Yeu Lee



Analyzing and Optimizing Query Performance

The power of Analysis Services lies in its ability to provide fast query response time for decision makers who need to analyze data, draw conclusions, and make appropriate changes in their business. The OLAP Report defines OLAP as Fast Analysis of Shared Multidimensional Information (<http://www.olapreport.com/fasmi.htm>). The word “fast” in this context means that the system is able to deliver results to users in less than 5 seconds (with a few highly complex queries taking more than 20 seconds). We also expect that most business decision makers will use client tools that graphically represent the data from Analysis Services for easy interpretation and understanding. As an end user, you expect to see the data quickly in order to analyze and make decisions. Some common operations that OLAP client tools offer are drill down, drill up, and compare data year over year. Users do not have time to wait for hours to get a response. Hence, queries sent to Analysis Services need to return data within seconds, at most in minutes. Query performance is pivotal to a successful Business Intelligence project deployment. A system that has very good performance will bring great business value to your company. However, you should be aware there can be queries to Analysis Services that can take more than a few minutes. Typically such queries are issued via overnight reporting systems.

Analysis Services supports three storage modes: MOLAP, ROLAP, and HOLAP. You will usually obtain the best performance when your UDM storage mode is MOLAP. When you choose MOLAP storage, Analysis Services 2008 will store the dimension data and fact data in its own efficient, compact multidimensional structure format. Fact data is compressed and its size is approximately 10 to 30 percent of the size as when stored in a relational database. In addition to its own efficient and compact data store, Analysis Services builds specialized dimension attribute indices for efficient data retrieval. The data is stored specifically in a multidimensional structure to best serve MDX query needs. If you use the ROLAP or HOLAP storage modes, queries to Analysis Services might have to fetch data from the relational data source at query time. Retrieving data from the relational data sources will significantly slow your query performance because you incur relational query processing time, the time needed to fetch the data over the network, and finally the time it takes to aggregate the data within Analysis Services.

Analysis Services 2008 tries to achieve the best of the OLAP and relational worlds. OLAP queries typically request aggregated data. For example, you may store daily sales information for products. Typical OLAP queries will request aggregated sales by month, quarter, or year. In such circumstances, every day’s sales data needs to be aggregated for the period requested by the query, for example the entire year. If users are requesting aggregated data on a single dimension, you will be

Part III: Advanced Administration and Performance Optimization

able to do a simple sum in the relational database. However, OLAP queries are typically multidimensional and need aggregated data across multiple dimensions with complex business logic calculations applied to each dimension. To improve query performance, Analysis Services allows you to specify aggregations, which you learned about in Chapter 14. In addition to aggregations, you learned several design techniques in that chapter to optimize your UDM to get the best performance from your Analysis Services database. Having done your best UDM design to satisfy your business needs, you might still encounter performance issues at query time. In this chapter you learn about the various components of Analysis Services that work together to execute MDX queries. You also learn how to analyze Analysis Services query performance issues as well as techniques and best practices for improving query performance.

The Calculation Model

Before we start looking at the overall Analysis Services query execution architecture, let's recap what you learned about the calculation model of Analysis Services in previous chapters of this book. When using the MOLAP storage mode, the data that comprises the cube is retrieved from a relational database and stored in SSAS's proprietary format. The data will be aggregated by the SSAS engine based on the MDX query. SSAS provides a way to pre-calculate aggregated data. This helps speed the retrieval of query results for MDX queries that can be satisfied with these pre-calculated aggregations. Most of the calculations that apply specific business logic in the UDM are written in MDX scripts, objects within your Analysis Services 2008 database that provide a procedural way to define calculations. SSAS features such as unary operators and custom rollups also help in defining MDX calculations needed within your UDM. The cube editor in Business Intelligence Development Studio provides a way to debug the calculations defined in your MDX scripts. However, there is complex calculation logic within the SSAS engine that defines how the calculations are applied to a specific cell. Each cell within the cube is either a value from your relational database or a calculation, as illustrated in Figure 15-1.

	All	WA	Seattle	Redmond	CA	Los Angeles	San Francisco	San Diego
All	1237	475	176	299	762	148	149	465
Q1	367	110	44	66	257	53	32	172
Jan	148	55	12	43	93	10	-	83
Feb	164	32	32	-	132	25	32	75
Mar	55	23	-	23	32	18	-	14
Q2	360	17	65	113	182	28	65	89
Apr	55	23	23	-	32	9	23	-
May	135	73	19	54	62	19	19	24
Jun	170	82	23	59	88	-	23	65
Q3	235	122	11	111	113	11	17	85
Jul	24	12	-	12	12	-	-	12
Aug	42	34	-	34	8	-	-	8
Sep	169	76	11	65	93	11	17	65
Q4	275	65	56	9	210	56	35	119
Oct	133	24	21	3	109	21	-	88
Nov	100	23	23	-	77	23	23	31
Dec	42	18	12	6	24	12	12	-

Figure 15-1

Figure 15-1 shows cells with sales corresponding to various months in the year and cities in the states of Washington and California. The members of the axes and the cell values that are calculated from the relational backend are shown in one color. You can see that some cells have a dash (–), indicating that no value was available for that specific cell from the relational backend. The remaining cells contain aggregated data and are shown in a darker color. For example, cells corresponding to Seattle and the months April, May, and June were all retrieved from the relational backend table. However, the cell value for the Q2 quarter and Seattle is aggregated from the sales for Seattle for the months of April, May, and June.

You can have several MDX calculations defined for a specific cell. The value for a cell that contains multiple MDX calculations is the value of the last calculation that gets applied to the cell. Several types of calculations can be defined in your MDX scripts: calculated members, named sets, unary operators, custom rollups, assignments, and calculations in sessions or queries. You have learned about all these types of calculations in Chapters 3 through 10. The remainder of this section offers a quick review of calculations in Analysis Services before we look at the details of the MDX query execution architecture.

MDX Script

There are multiple ways calculations can be defined in Analysis Services 2008. Most are defined using the MDX Script, which is a centralized calculation store for the cube. Dimension calculations such as unary operators and custom rollups are a part of the dimension and can be defined using attribute properties. You can define these calculations via MDX script but we highly recommend using the support for defining them via dimension attribute properties to achieve better performance. Each cube in Analysis Services 2008 contains a single MDX Script. Business Intelligence Development Studio (BIDS) exposes the MDX Script object to editing and debugging via the Calculations tab (shown in Figure 15-2), as you learned in Chapters 6 and 9. MDX Script provides a procedural execution model and easier debugging of calculations, as seen in Chapter 6. The commands in the script are executed in the order they have been defined. You learned about the `Pass Value` (also called `Pass Order`) in Chapter 10, which refers to stages of calculations applied to the cube when there are multiple calculations such as custom rollup, unary operators, and assignment statements that are applied to the cells of a cube. In Analysis Services 2008, a new `Pass Value` is created for each MDX calculation defined in the MDX Script to avoid infinite recursion. The creation of a new `PASS Value` for each cell calculation also eliminates the need for `Solve Order`. (`Solve Order` is used to help in determining the order of calculations within a single `Pass` in SQL Server Analysis Services 2000, which is deprecated from SSAS 2005.)

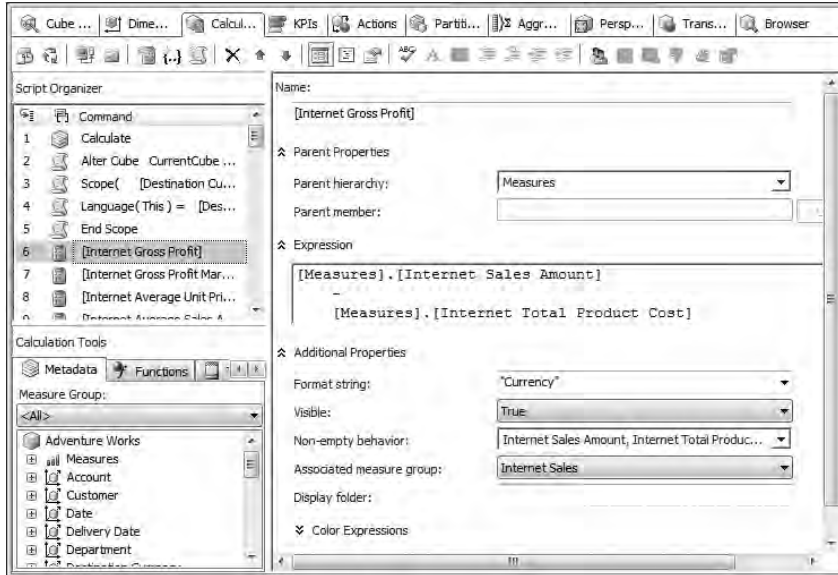


Figure 15-2

The single view of the calculations via the MDX Script simplifies the maintenance of your MDX calculations as well as debugging. As part of your UDM development, you can use source code control and check in various versions of your Analysis Services project. This helps you track the history of changes to your project and also aids in maintenance. Because the calculations are part of the Analysis Services project, you automatically get version control of the calculation changes.

We recommend you periodically check in the changes made to your Analysis Services project similar to what you would do for a C# or a C++ project.

The first and foremost command in an MDX script is the `CALCULATE` command. The `CALCULATE` command populates each cell in the cube along with aggregated data from the fact level data (also called leaf level data). Without the `CALCULATE` command the cube will only contain the fact level data. The syntax of the `CALCULATE` command is

```
CALCULATE [ <subcube> ] ;
```

If the `<subcube>` argument is not specified, the current cube is used. The `CALCULATE` command is automatically added by the Cube Wizard when you create a cube in Business Intelligence Development Studio. BIDS typically adds the `CALCULATE` statement at the beginning of the MDX Script, resulting in the default aggregation behavior for the measures, which you see in previous versions of SQL Server Analysis Services. When Analysis Services evaluates the cells, it first loads the fact data into the cube's cell values. Then it does the default aggregation of the non-leaf cell values. Finally, the MDX calculations as defined by the Analysis Services rules are applied to determine the final values of the cells in the cube. The assignment calculations in the MDX Script are evaluated using the `Pass Value`, which gets incremented for each MDX Script assignment. Note that the `CALCULATE` statement does not have any effect on calculated members defined in the MDX Script.

Scope and Assignments

Analysis Services 2008 supports multiple ways to define cell calculations. Each cell can have one or more calculations defined for it. Unary operators, custom rollups, and Assignments are three ways you can define cell calculations while designing a cube. In addition, you can define calculations as part of sessions (session calculations) or queries (query calculations). Unary operators and custom rollups are defined as part of dimension creation using the dimensions' attribute properties, and Assignments are statements that define cell calculations and are defined in the MDX Script.

Assignments are typically enclosed within a Scope statement, which helps define calculations on a subcube. Following is the syntax for the Scope and Assignment statement (=) that you learned about in Chapter 10. You can have one or more assignments within each Scope statement. In addition, you can have nested scopes. Scopes by default inherit the parent scope, however you can override this. For example, you can have a parent scope of `Customers.USA`, which will scope to all customers in the country USA. You can have a nested scope of `Customers.Canada`, which will override the parent and change the scope to customers in Canada.

```
Scope(<subcube>);
    <subcube1 definition> = expression; [Example: this = 1000;]
    ...
End Scope;
```

Analysis Services restricts the cube space as defined by the Scope statement. Then the assignment statement is evaluated for all the cells within the specified subcube1 definition. The term `this` is a special keyword that denotes the assignment to be evaluated on the default measure of the subcube defined within the Scope statement. You can have multiple assignment statements that overwrite a specific cell within the same Scope statement.

Dimension Attribute Calculations

You learned about the Custom Rollup and Unary Operators features in Analysis Services 2008 in Chapter 8. These features help define how to aggregate data to parent members or other members in the hierarchy. Analysis Services uses special rules while aggregating data when performing cell calculations in MDX scripts. In general, you can assume that the last cell calculation is the one that will be the final cell value. This behavior is referred to as "Latest Wins." In addition, there are instances where a calculation called as the closest calculation for the cell being aggregated will be the final value; this is called "Closest Wins." Richard Tkachuk, Program Manager from Microsoft, has written a white paper, "Introduction to MDX Scripting in Microsoft SQL Server 2005," that demonstrates examples of Latest Wins and Closest Wins (<http://msdn.microsoft.com/en-us/library/ms345116.asp>).

Session and Query Calculations

As you learned in Chapter 10, Analysis Services allows you to specify cell calculations in session, query, or global scopes. Following are the examples from Chapter 10 that show how a cell calculation is defined at query, session, or global scopes, respectively:

```
WITH CELL CALCULATION [SalesQuota2005]
FOR '( [Date].[Fiscal Year].&[2005],
    [Date].[Fiscal].[Month].MEMBERS,
    [Measures].[Sales Amount Quota] )'
AS '( PARALLELPERIOD( [Date].[Fiscal].[Fiscal Year], 1,
    [Date].[Fiscal].CurrentMember), [Measures].[Sales Amount] ) * 2'
SELECT { [Measures].[Sales Amount Quota],
```

(continued)

(continued)

```
[Measures].[Sales Amount] } ON COLUMNS,
DESCENDANTS( { [Date].[Fiscal].[Fiscal Year].&[2004],
[Date].[Fiscal].[Fiscal Year].&[2005] }, 3, SELF ) ON ROWS
FROM [Adventure Works]
```

```
CREATE CELL CALCULATION [Adventure Works].[SalesQuota2005]
FOR '( [Date].[Fiscal].[Month].MEMBERS, [Measures].[Sales Amount Quota]
)'
AS '(PARALLELPERIOD ( [Date].[Fiscal].[Fiscal Year],
1, [Date].[Fiscal].CurrentMember), [Measures].[Sales Amount])*2 ',
CONDITION = ' [Date].[Fiscal Year].CurrentMember IS
[Date].[Fiscal Year].&[2005] '
```

```
SCOPE([Date].[Fiscal Year].&[2005],
[Date].[Fiscal].[Month].MEMBERS,
[Measures].[Sales Amount Quota]);

THIS = (ParallelPeriod( [Date].[Fiscal].[Fiscal Year],
1, [Date].[Fiscal].CurrentMember), [Measures].[Sales Amount])*2;

END SCOPE;
```

Having calculations at appropriate scopes is based on the requirements of your cube and the client tools used to interact with the cube. Analysis Services 2008 has specific optimizations that cache the results of calculations at each scope. When a query is being evaluated, Analysis Services 2008 first tries to retrieve the results from query scope. If this is not possible, it looks at session scope and finally at global scope. This is a specific optimization implemented in Analysis Services to help improve query performance, however some calculations may not be cached (such as calculations that include locale-related information).

Having reviewed the calculation model of Analysis Services, let's now look at the architecture and the steps involved when executing an MDX query.

Query Execution Architecture

Microsoft SQL Server Analysis Services 2008 consists of server and tools components that enable you to create databases and manage them. The server components are a set of binaries that comprise the Analysis Services service. BIDS, SQL Server Management Studio (SSMS), Profiler, and a few additional binaries constitute the tools components. The multidimensional databases are stored on the server, which is also referred to as the SSAS engine. SSAS clients communicate to the SSAS engine via XML for Analysis, a standardized application programming interface for online analytical processing (OLAP). The XMLA API has two main methods, Discover and Execute. Discover allows callers to request metadata and data from the databases. Execute lets callers send commands such as Create, Alter, and Process, which are used for creating/updating the multidimensional database or MultiDimensional Expressions (MDX) queries. MDX query results can be retrieved in multidimensional or tabular format by the client. The Create, Alter, Delete, and Process statements are part of the Data Definition Language (DDL). SSAS provides a set of object models that abstract XMLA and make it easy for developers to build applications that can communicate with the SSAS engine.

Analysis Services Engine Components

Figure 15-3 shows the Analysis Services query execution architecture. Five major components constitute the Analysis Services server: Infrastructure, Data Mining, Metadata Manager, Storage Engine, and Formula Engine. These are detailed in the following list.

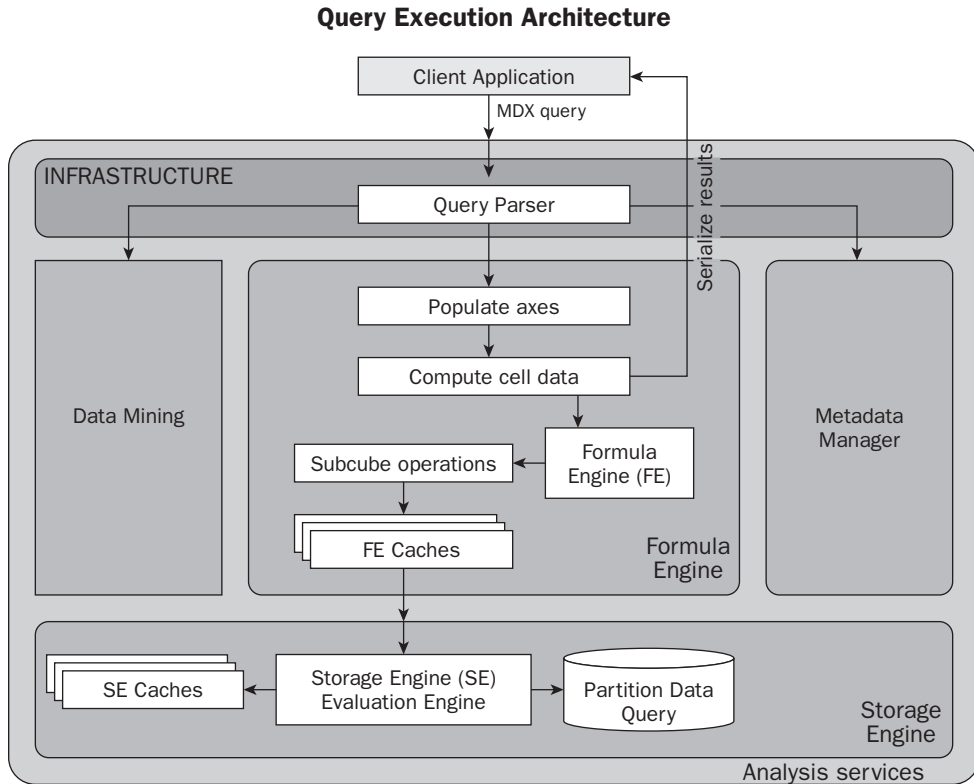


Figure 15-3

- ❑ **Infrastructure:** The Infrastructure handles operations such as accepting requests from clients, distributing the requests to the appropriate components, scheduling the jobs, and memory management. Parsing and validating the XMLA requests are also part of this component, as well as providing the support for retrieving data from external data sources. Consider this component as being the main interface for the client and also providing appropriate infrastructure to support the operation of the remaining components.
- ❑ **Metadata Manager:** The Metadata Manager handles the DDL statements that operate on the multidimensional database objects. DDL statements such as Create, Alter, Delete, and Process are directed from the infrastructure component to the Metadata Manager. This component also implements the transaction handling for all Analysis Services objects. When processing statements are issued, it coordinates with the storage engine or data mining component and the infrastructure to retrieve data from the relational data sources and store them in an optimized storage format within Analysis Services.

- ❑ **Data Mining:** The Data Mining component (you learn about data mining in Chapter 16) serves all Data Mining requests. It coordinates with the infrastructure and metadata manager at the time of processing data mining models. If there are OLAP mining models, the data mining component sends queries to the storage engine and formula engine components to retrieve appropriate data from the cube. This component handles Discover and DMX queries sent to the data mining models.
- ❑ **Storage Engine:** The Storage Engine is one of the core components of an OLAP database. It populates the multidimensional database with data from relational databases and optimally stores them on disk. It also optimizes the storage for dimension and cube data and builds relevant indices to aid in fast and efficient retrieval of the data from the disk. Typically you will see around a 10:1 compression ratio between the relational data and the OLAP data. The storage engine component provides internal interfaces to the formula engine component so that subcubes of data can be retrieved; these can then be used by the formula engine for efficient retrieval and aggregation of the data to satisfy MDX query requests.
- ❑ **Formula Engine:** The MDX Query Processor, also referred to as Formula Engine, determines the execution strategy for each MDX query. The Formula Engine can be considered the most important component with respect to MDX queries and calculations because the query evaluation and computation is done by this component. It translates each query into a sequence of requests to the Storage Engine to access the data, and computes the results of the query based on any calculations defined in the multidimensional database. It also implements caching for optimal query performance.

Stages of Query Execution

A query is sent from a client to the Analysis Services engine, as shown in Figure 15-3. The Analysis Services engine first parses the client request and routes it to the Data Mining Engine, the Formula Engine, or the Metadata Manager. Figure 15-3 shows the query execution architecture for serving Discover and MDX queries. There are several key steps in query evaluation: parsing the query, populating and serializing the axes, computing the cell data, and serializing the results back to the client. The following list provides more detail of each of the steps:

- ❑ **Parsing the query:** The MDX query is first analyzed by the query parser and then passed on to the Formula Engine. If there is a syntactical error in the query, the parser returns an appropriate error message to the client.
- ❑ **Populating the axes:** The Formula Engine evaluates the members of the axes of the MDX query. After this has been done, the details of the axes are populated.
- ❑ **Serializing the axes:** After the axes are evaluated and populated, Analysis Services sends details of the cube being queried back to the client, including the hierarchies and levels of the cube dimensions. Then the axes information, which includes the tuples and members that form the axes, are serialized. Some dimension properties of the members such as caption, unique name, and level name are sent to the client by default. If additional properties are requested in the MDX query, they will be included as well.
- ❑ **Evaluating the cell data:** After the axes data has been populated, the Analysis Services engine understands which cell coordinates need to be evaluated. The Formula Engine (FE) first tries to retrieve the results from the FE cache. If the query cannot be retrieved from the FE cache, appropriate internal queries are sent to the Storage Engine. The Storage Engine (SE) has its own cache. The SE determines if the query can be satisfied from the SE cache. If the query results are

not available in the SE cache, results are retrieved from partition data on the disk, stored in the SE cache, and sent to the FE. The FE then performs the calculations needed to satisfy the query and is then ready to send the results back to the client.

- ❑ **Serializing the cells:** After the results are available, they are sent back to the client. The results are sent in the XMLA format.

Query Evaluation Modes

Now that you understand the various stages of MDX query evaluation, it's time to look at the two query evaluation modes in Analysis Services 2008: cell by cell mode and subspace computation.

Cell by Cell Mode

When an MDX query has been parsed, it is evaluated to see if the query can use the subspace computation mode. (The factors that determine whether the query can use the subspace computation mode are addressed in the next section.) If the query cannot be evaluated in the subspace computation mode, it is evaluated in the cell by cell mode.

Query evaluation can include several thousand or even millions of cells, and thus evaluating every cell, which happens in the cell by cell mode, is typically slower than the subspace computation mode. The following example, an MDX query against the sample Adventure Works DW 2008, will help you see how cell by cell mode works:

```
WITH MEMBER Measures.ContributionToParent AS
    ([Measures].[Internet Sales Amount]/
        ([Measures].[Internet Sales Amount],
            [Customer].[Customer Geography].CurrentMember.Parent)),
    FORMAT_STRING="Percent"
SELECT {[Product].[Product Categories].[Category].MEMBERS} ON 1,
    [Customer].[Customer Geography].[Country].MEMBERS ON 0
FROM [Adventure Works]
WHERE (Measures.ContributionToParent)
```

The preceding MDX query contains a calculated member that calculates the contribution of [Internet Sales Amount] from each country for each product. If you execute this query in the SSMS MDX query editor, you will see the results as shown in Figure 15-4.

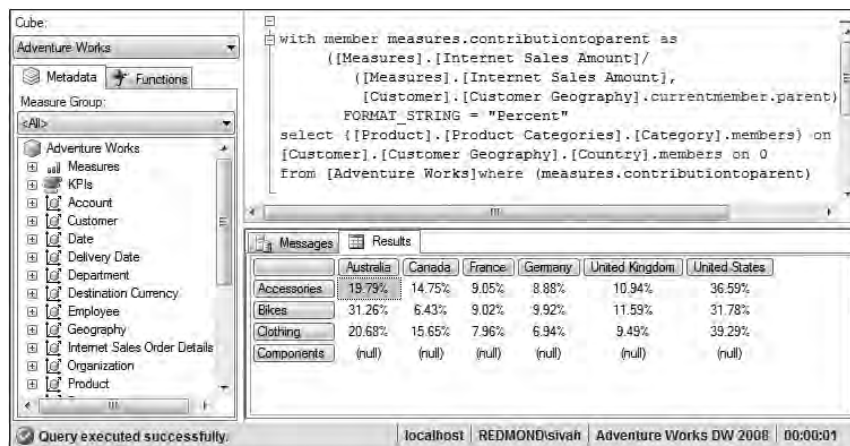


Figure 15-4

Part III: Advanced Administration and Performance Optimization

You can see that there are six countries and four products in the results. If you aggregate the percentage for each product across all countries you will get 100 percent. You can easily see that the United State's contribution for the company's [Internet Sales Amount] is the maximum for all the products. Once the axes information is populated, Analysis Services needs to calculate the values for 24 cells. The cell by cell mode in Analysis Services does this using the following steps:

1. Evaluate the measure [Internet Sales Amount] for a cell.
2. Evaluate the [Internet Sales Amount] for the member [Customer].[Customer Geography].[All Customers] for that cell.
3. Evaluate the measure ContributionToParent, which is the calculated member in the MDX query for that cell.
4. Repeat steps 1, 2, and 3 for each cell including cells that have null values.

Results of steps 1 and 2 for all the cells are shown in Figure 15-5.

```
select {[Product].[Product Categories].[Category].members} on 1,
{{[Customer].[Customer Geography].[Country].members} on 0
from [Adventure Works]where (measures.[Internet Sales Amount]):
go

select {[Product].[Product Categories].[Category].members} on 1,
{[Customer].[Customer Geography].[All Customers]} on 0
from [Adventure Works]where (measures.[Internet Sales Amount]):
|
go
```

	Australia	Canada	France	Germany	United Kingdom	United States
Accessories	\$138,690.63	\$103,377.85	\$63,406.78	\$62,232.59	\$76,630.04	\$256,422.07
Bikes	\$8,852,050.00	\$1,821,302.39	\$2,553,575.71	\$2,808,514.35	\$3,282,842.66	\$8,999,859.53
Clothing	\$70,259.95	\$53,164.62	\$27,035.22	\$23,565.40	\$32,239.51	\$133,507.91
Components	(null)	(null)	(null)	(null)	(null)	(null)

	All Customers
Accessories	\$700,759.96
Bikes	\$28,318,144.65
Clothing	\$339,772.61
Components	(null)

Figure 15-5

In this example you can see that the evaluation of step 2 needs to be done only once. In addition, the cells for which [Internet Sales Amount] is null don't have to be calculated because the calculated measure ContributionToParent for null values will be null. When there are millions of cells, evaluation of each and every cell can take a considerable amount of time. The next evaluation mode, subspace computation, helps optimize query evaluation.

Subspace Computation

The Analysis Services 2008 cube space is typically sparse. This means that only some of the cells in the dimensional space have values. The remaining cell values are null. The goal of the subspace computation query evaluation mode is to evaluate MDX expressions only when they need to be evaluated. For example, if a cell value is null, an MDX expression using that value will result in a null value and

therefore doesn't have to be evaluated. Subspace computation can reduce cell evaluation time by orders of magnitude, depending on the sparseness of the cube. Some queries that run in minutes using cell by cell mode are evaluated within seconds using subspace computation mode. Subspace computation was first introduced in Analysis Services 2005 Service Pack 2 for a limited number of scenarios. In Analysis Services 2008, subspace computation mode has been enhanced to cover a wider scope of MDX evaluations and automatic query optimizations.

The subspace computation mode can be taken by Analysis Services only under specific conditions. Some of the important conditions where Analysis Services will use the subspace computation mode are given here:

- ❑ Basic operations that involve arithmetic operators (*, /, +, -), and relational operators (<, >, <=, >=, =).
- ❑ Static references to members and tuples as well as constant scalars such as NULL.
- ❑ Scalar operations using functions IS, MemberValue, Properties, Name, IIF, IsNonEmpty, Case, IsLeaf, IsSiblings, CalculationPassValue, and member functions such as PrevMember, NextMember, Lag, Lead, FirstChild, LastChild, Ancestor, and so on.
- ❑ Basic Aggregate functions such as Sum, Min, Max, and Aggregate on static sets; as well as sets built using functions PeriodsToDate, YTD, QTD, MTD, Crossjoin, Cousin, Descendants, Children, Hierarchize, and Members.
- ❑ The CurrentMember function (only on the Measures dimension) and basic unary operators and semi-additive measures.

Some examples where subspace computation mode will not be chosen include named sets when used with Aggregate functions, dynamic operations (for example: [Date].[Fiscal].Lag([Measures].[Count])), and when encountering recursion.

As an example, consider the simple MDX query from the previous section. The MDX query is first analyzed to determine if it can be evaluated using subspace computation mode. Because the answer is yes, Analysis Services uses the following steps for evaluating the query:

1. Retrieve non-null values of the [Internet Sales Amount] measure for the query results space.
2. Retrieve the [Internet Sales Amount] for member [Customer].[Customer Geography].[All Customers] once.
3. Evaluate the ContributionToParent measure for the non-null values retrieved.

Figure 15-6 provides a graphical illustration comparing the cell by cell and subspace computation modes. Assume the machine in the diagram is the Analysis Services engine. The figure on the left shows the cell by cell mode, where all the cells are evaluated. The figure on the right shows that the cells that have non-null values (highlighted by darker color) are first identified via storage engine requests and then evaluation is only done for those cells. Note that when Analysis Services serializes the results back to the client, it only includes the cell values that contain data. The remaining cell values are assumed to be null. This is shown in Figure 15-7 with an MDX query. This MDX query should return 24 cells with cell ordinals 0 to 23. However, this only returns 18 cell values because the cell values corresponding to the product member Components ([Product].[Product Categories].[Category].&[2]) are null. The client object models provided by Analysis Services 2008 interpret the results returned from the Analysis Services engine and populate the missing cells with null values for the client accessing the data.

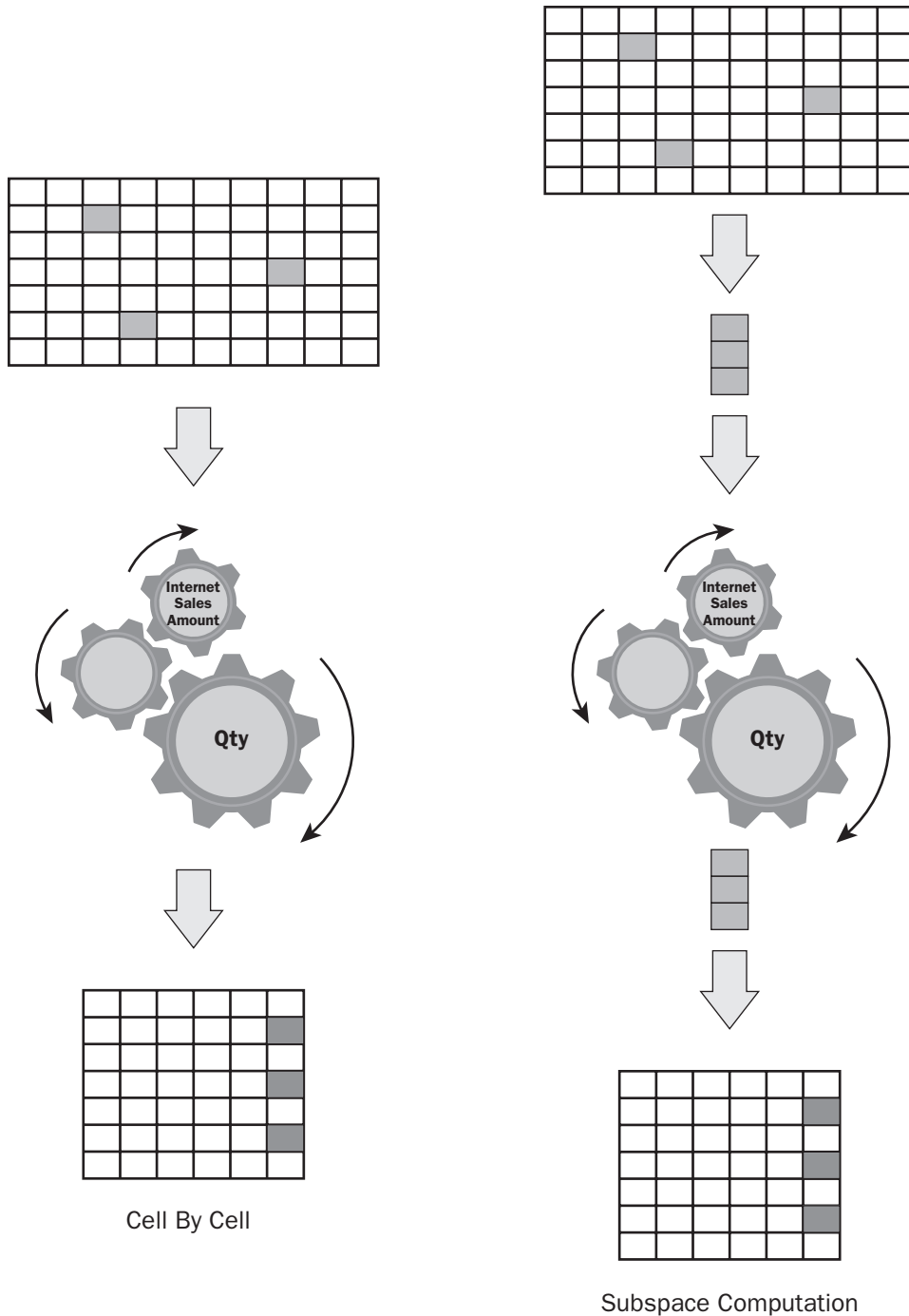


Figure 15-6

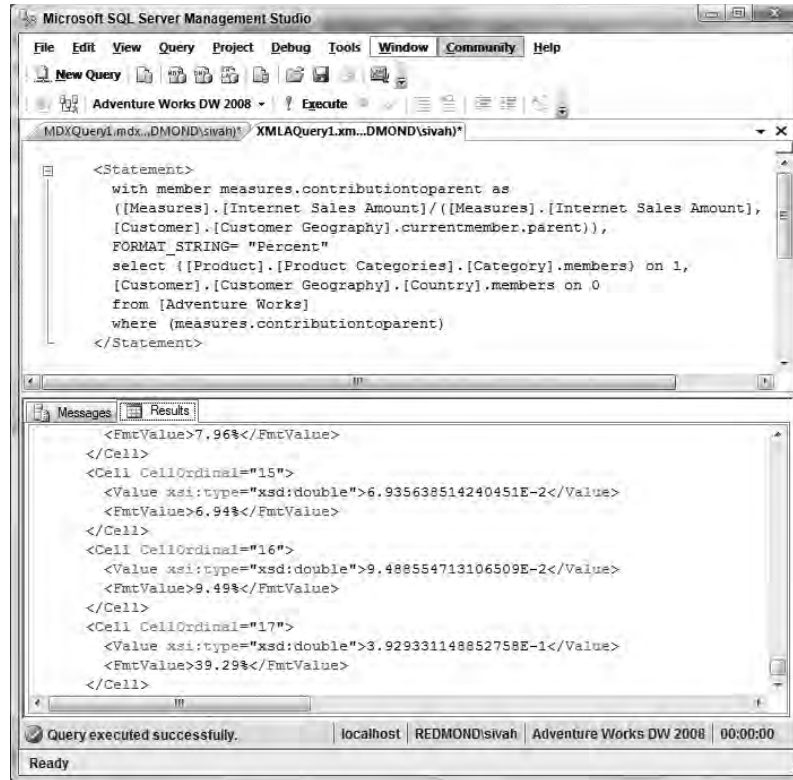


Figure 15-7

Analysis Services has two NON EMPTY code paths that would eliminate null cell values: Naïve NON EMPTY and Express NON EMPTY. The Naïve NON EMPTY code path was used in the cell by cell mode and the Express NON EMPTY path was used to identify the tuples that contained data. However, in Analysis Services 2005 Express NON EMPTY was restricted to measures that did not have calculations or where NON_EMPTY_BEHAVIOR (discussed later in this chapter) was specified. Analysis Services 2008 Express NON EMPTY has been enhanced to support measures with calculations (except for recursive or complex overlapping calculations). Now that you've learned more about the Analysis Services query execution architecture and query evaluation modes, let's look into analyzing performance bottlenecks and fine-tuning them.

Performance Analysis and Tuning Tools

Analysis Services 2008 includes significant enhancements targeted at getting the best query performance. Improvements include tools to help in designing cubes, subspace computation optimization, caching enhancements, and improved writeback query performance (you learn about this later in the chapter). You might still have queries that are not performing as expected, however, due to cube design or the way MDX has been written. To analyze and improve your query performance, you can use tools that will help you analyze the performance of your queries and then tune them to get the best performance from Analysis Services 2008.

SQL Server Profiler

SQL Server Profiler is a tool used to trace operations on the SQL Server and Analysis Services database engines. SQL Server Profiler is the primary performance analysis tool used to debug performance bottlenecks in SQL Server (including Analysis Services). The ability to trace Analysis Services operations through SQL Server Profiler was first introduced in SQL Server 2005.

Analysis Services exposes the commands sent to it as well as internal operations that occur within the server through what are called *events*. Some examples of these events are Command Begin, Command End, Query Begin, and Query End. Each event has properties associated with it such as start time, end time, and the user sending the query. These properties are shown as columns in the tool. SQL Server Profiler requests these events and their properties through trace commands to the server. Analysis Services periodically sends the events to the clients who have subscribed to a trace. SQL Server Profiler shows the events and event column values in a grid. Only Analysis Services administrators can trace Analysis Services events. To learn more about how to use the Profiler, follow these steps:

1. Make sure you are an administrator on the Analysis Services server you want to profile. You can connect to Analysis Services through SSMS and use the Analysis Services Server Properties dialog to add users as administrators of Analysis Services, as you learned in Chapter 7.
2. Launch SQL Server Profiler from the Start menu: All Programs ⇨ Microsoft SQL Server 2008 ⇨ Performance Tools ⇨ SQL Server Profiler.
3. The SQL Server Profiler application appears. Create a new trace by selecting File ⇨ New Trace.
4. In the Connect to Server dialog, shown in Figure 15-8, select Analysis Services as the Server type and enter the name of your Analysis Services instance. Click Connect.



Figure 15-8

5. In the Trace Properties dialog, enter the Trace name, for example “FirstTrace.” SQL Server Profiler provides three trace templates with pre-selected events to trace. Select the Standard template as shown in Figure 15-9.

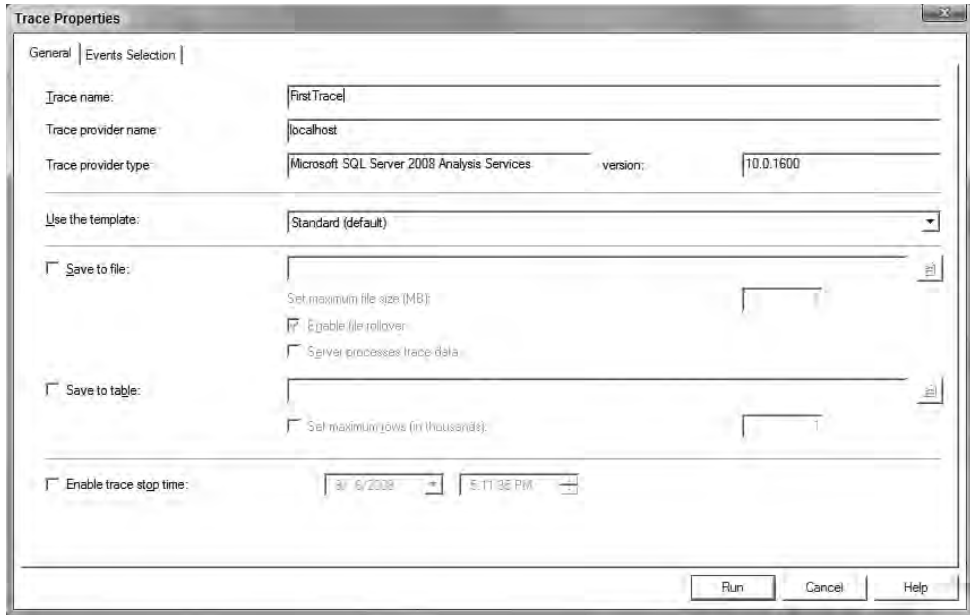


Figure 15-9

6. To see the events selected in the standard template, click the Events Selection tab. You will see the event columns that have been selected, as in Figure 15-10. This page only shows the events that have properties that have been selected. To see all the events and event properties supported by Analysis Services, check the Show All Events and Show All Columns checkboxes, respectively. Familiarize yourself with the various events and click Run.

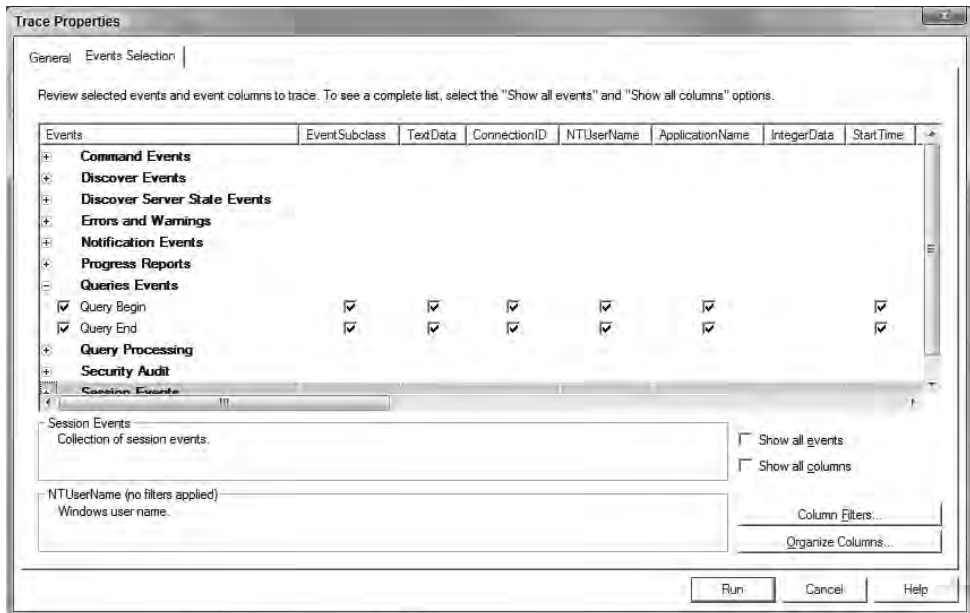
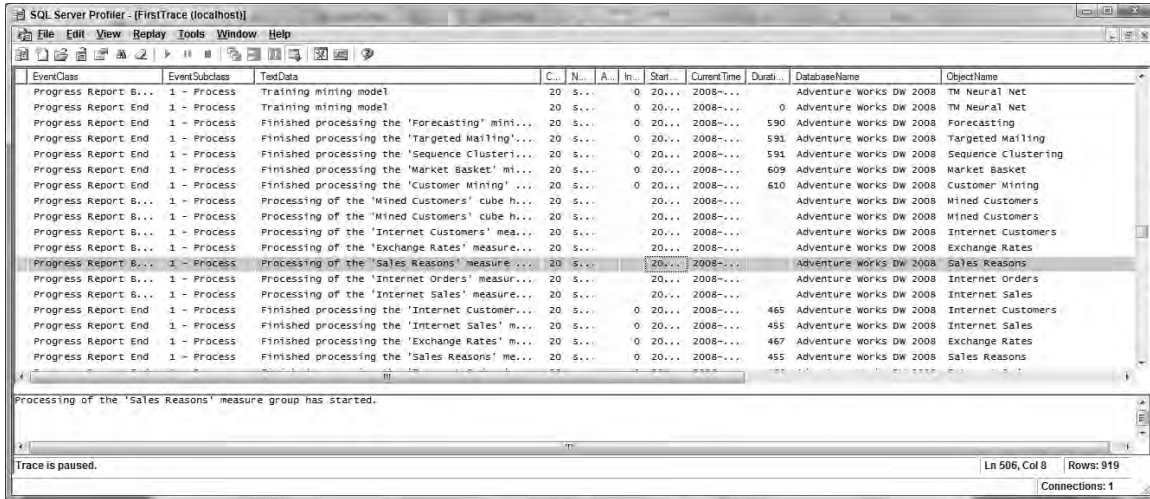


Figure 15-10

Part III: Advanced Administration and Performance Optimization

7. You will see the various event property columns within Profiler. To see processing operations events, open the Adventure Works DW sample project and deploy it to the Analysis Services instance. You will see the events that happen during processing, including the processing duration of each object, as shown in Figure 15-11. The SQL Server Profiler gives you useful information such as the time it takes to process each dimension, the partition processing time, and the overall processing time of the entire database.



EventClass	EventSubclass	TextData	C...	N...	A...	In...	Start	CurrentTime	Durati...	DatabaseName	ObjectName
Progress Report B...	1 - Process	Training mining model	20	S...			0 20...	2008-...		Adventure works DW 2008	TM Neural Net
Progress Report End	1 - Process	Training mining model	20	S...			0 20...	2008-...	0	Adventure works DW 2008	TM Neural Net
Progress Report End	1 - Process	Finished processing the 'Forecasting' mini...	20	S...			0 20...	2008-...	590	Adventure works DW 2008	Forecasting
Progress Report End	1 - Process	Finished processing the 'Targeted Mailing'...	20	S...			0 20...	2008-...	591	Adventure works DW 2008	Targeted Mailing
Progress Report End	1 - Process	Finished processing the 'Sequence Clusteri...	20	S...			0 20...	2008-...	591	Adventure works DW 2008	Sequence Clustering
Progress Report End	1 - Process	Finished processing the 'Market Basket' mi...	20	S...			0 20...	2008-...	609	Adventure works DW 2008	Market Basket
Progress Report End	1 - Process	Finished processing the 'Customer Mining'...	20	S...			0 20...	2008-...	610	Adventure works DW 2008	Customer Mining
Progress Report B...	1 - Process	Processing of the 'Mined Customers' cube h...	20	S...			20...	2008-...		Adventure works DW 2008	Mined Customers
Progress Report B...	1 - Process	Processing of the 'Mined Customers' cube h...	20	S...			20...	2008-...		Adventure works DW 2008	Mined Customers
Progress Report B...	1 - Process	Processing of the 'Internet Customers' mea...	20	S...			20...	2008-...		Adventure works DW 2008	Internet Customers
Progress Report B...	1 - Process	Processing of the 'Exchange Rates' mea...	20	S...			20...	2008-...		Adventure works DW 2008	Exchange Rates
Progress Report B...	1 - Process	Processing of the 'Sales Reasons' measure ...	20	S...			20...	2008-...		Adventure works DW 2008	Sales Reasons
Progress Report B...	1 - Process	Processing of the 'Internet Orders' measur...	20	S...			20...	2008-...		Adventure works DW 2008	Internet Orders
Progress Report B...	1 - Process	Processing of the 'Internet Sales' measure...	20	S...			20...	2008-...		Adventure works DW 2008	Internet Sales
Progress Report End	1 - Process	Finished processing the 'Internet Customer...	20	S...			0 20...	2008-...	465	Adventure works DW 2008	Internet Customers
Progress Report End	1 - Process	Finished processing the 'Internet Sales' m...	20	S...			0 20...	2008-...	455	Adventure works DW 2008	Internet Sales
Progress Report End	1 - Process	Finished processing the 'Exchange Rates' m...	20	S...			0 20...	2008-...	467	Adventure works DW 2008	Exchange Rates
Progress Report End	1 - Process	Finished processing the 'Sales Reasons' me...	20	S...			0 20...	2008-...	455	Adventure works DW 2008	Sales Reasons

Figure 15-11

After the processing has completed for the Adventure Works cube, send the following MDX query using SQL Server Management Studio:

```
SELECT {[Measures].[Sales Amount],[Measures].[Gross Profit]} ON 0,
[Customer].[Customer Geography].MEMBERS ON 1
FROM [Adventure Works]
```

You can see the Query events in the SQL Server Profiler as shown in Figure 15-12. You can see the duration of each event in the Profiler trace (not shown in Figure 15-12). One piece of information that is interesting to notice is the subcubes accessed by this query and how long each subcube query took. The subcube events indicate the requests of the storage engine to retrieve data from disk. You can utilize this subcube information to build custom aggregations to optimize query performance.

EventClass	EventSubclass	TextData	ConnectionID	NTUserName	ApplicationName	IntegerData	StartTime	CurrentTime
Notification	0 - Object...		19	sivah		1	2008-09-06 17:23:31...	2008-09-06 17:23
Query Begin	0 - MDXQuery	select {[Measures].[Sales Amount], [Me...	14	sivah	Microsoft SQ...		2008-09-06 17:23:47...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query Subcube	1 - cache ...	00000,00000000000000000000,00000000...	14				2008-09-06 17:23:49...	2008-09-06 17:23
Query End	0 - MDXQuery	select {[Measures].[Sales Amount]}, [Me...	14	sivah	Microsoft SQ...		2008-09-06 17:23:47...	2008-09-06 17:23
Discover Begin	26 - DISCO...	<RestrictionList xmlns="urn:schemas...	14	sivah	Microsoft SQ...		2008-09-06 17:23:55...	2008-09-06 17:23
Discover End	26 - DISCO...	<RestrictionList xmlns="urn:schemas...	14	sivah	Microsoft SQ...		2008-09-06 17:23:55...	2008-09-06 17:23

```
select {[Measures].[Sales Amount]},[Measures].[Gross Profit]} on 0, [Customer],[Customer Geography].members on 1 from [Adventure works]

<PropertyList xmlns="urn:schemas-microsoft-com:xml-analysis">
  <SSPROPInitAppName>Microsoft SQL Server Management Studio - Query/SSPROPInitAppName</SSPROPInitAppName>
  <LocalIdentifier>1033/LocalIdentifier</LocalIdentifier>
  <ClientProcessID>17806/ClientProcessID</ClientProcessID>
  <Catalog>Adventure Works DW 2008/Catalog</Catalog>
  <Format>Native/Format</Format>
  <AxisFormat>TupleFormat</AxisFormat>
  <Content>SchemaData</Content>
  <Timeout>0/Timeout</Timeout>
</PropertyList>
```

Figure 15-12

Assume you built aggregations using a usage-based optimization wizard. You would like to find out if the aggregations are being utilized. Analysis Services provides events that help you identify if the aggregations are hit. Create a new Trace and switch to the Events Selection tab. Check the box next to Show All Events. Expand the events under the Query Processing event group. You can see the events related to query processing that are provided by Analysis Services as shown in Figure 15-13.

If you select the events under Query Processing and monitor the trace events you will be able to obtain information such as if the Non Empty code path is being utilized, if the MDX script is being evaluated, if data is retrieved from Aggregations (Get Data From Aggregation event) or from the existing cache (Get Data From Cache event). These events help you identify more details about the queries sent by the users as well as their duration. You can later analyze the MDX queries, build usage-based optimization aggregations for long-running queries, enhance your aggregations using the new Aggregation Designer (which is discussed in Chapter 9), or try to optimize the long-running MDX queries (which is discussed later in this chapter). You do need to know a little bit about the internals of the server to fine-tune it. We believe the ability to trace Analysis Services activity through SQL Server Profiler will help with that, so try it out.

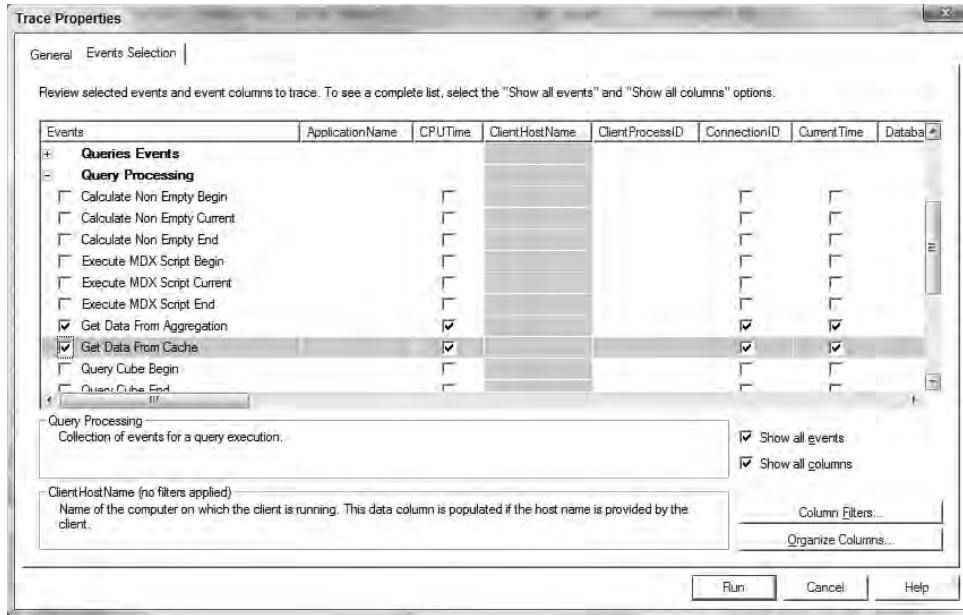


Figure 15-13

Performance Monitor

Analysis Services provides several performance monitoring counters that help you understand internal operations of your Analysis Services server, as well as help in debugging and troubleshooting performance issues. You need to be an administrator on the Analysis Services server to utilize PerfMon, a tool that lets you observe and analyze the Analysis Services 2008 performance counter values. The following steps walk you through working with Analysis Services performance counters:

1. Click Start and type **perfmon**, as shown in Figure 15-14, and select **perfmon.exe** from the Programs list.



Figure 15-14

2. You will see the Reliability and Performance Monitor application. Select the Performance Monitor page as shown in Figure 15-15.

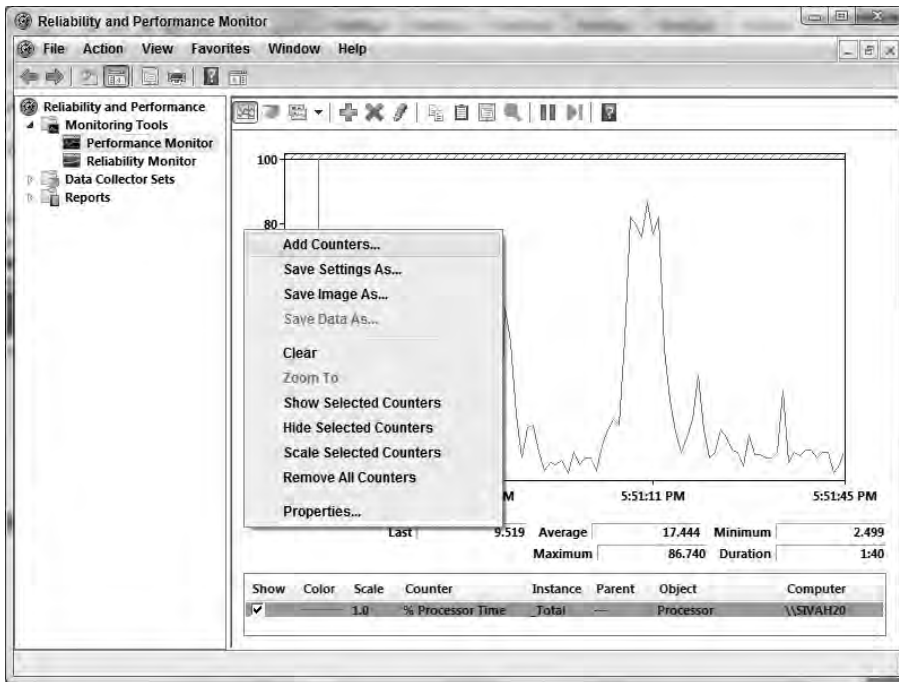


Figure 15-15

3. Right-click the page and select Add Counters. You will see the groups of Analysis Services performance counters as shown in Figure 15-16. You can expand a specific group to see the list of counters in that group.
4. Select the MSAS 2008:MDX category of performance counters and click Add to include these counters.
5. Click OK in the Add Counters dialog.

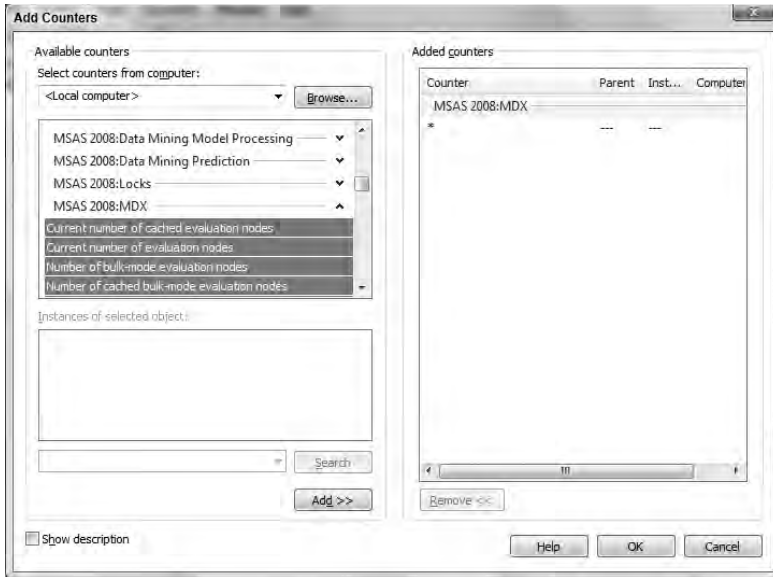


Figure 15-16

The MDX counters are added to the Performance Monitor page as shown in Figure 15-17. If you click a specific counter you can see the line view of that value over time. It is easier to understand and analyze the counters using their raw numbers.

6. Click the down arrow beside the Change Graph Type icon and select Report, as shown in Figure 15-17.

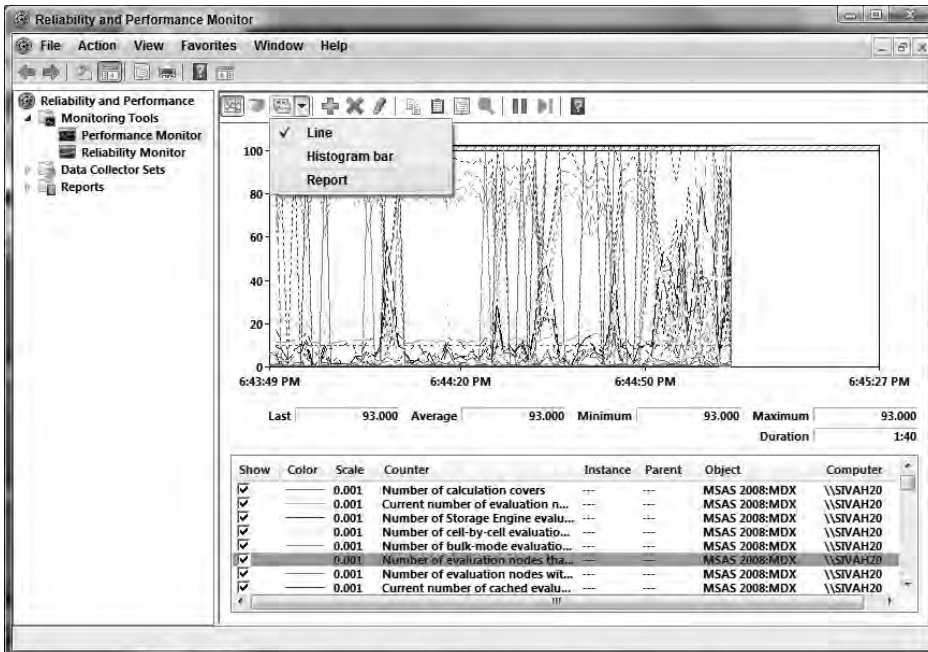


Figure 15-17

You will see the list of MDX counters along with their values in a report format as shown in Figure 15-18. If you execute MDX queries from SSMS or browse the cube using the Cube Browser in SSMS or BIDS, you should see these values getting updated. For example, you can see the “Number of bulk-mode evaluation nodes” (nodes during subspace computation evaluation) and “Number of cell-by-cell evaluation nodes” performance counters for a specific query to understand if the query is using the subspace computation or a cell by cell evaluation mode. This can help you to understand and optimize your MDX query. Similar to MDX performance counters, there are Analysis Services performance counters in other categories such as Processing, Aggregations, Connections, and so on. These counters are very valuable when you are troubleshooting specific problems and are not able to understand or resolve the problem using SQL Server Profiler traces. We recommend that you take a look at the various Analysis Services counter groups. In addition to the performance counters provided by Analysis Services, you can also look at other performance counters such as processors, memory, and disk I/O on your computer system to understand and troubleshoot relevant issues such as long-running queries, which are CPU intensive or memory/disk intensive.

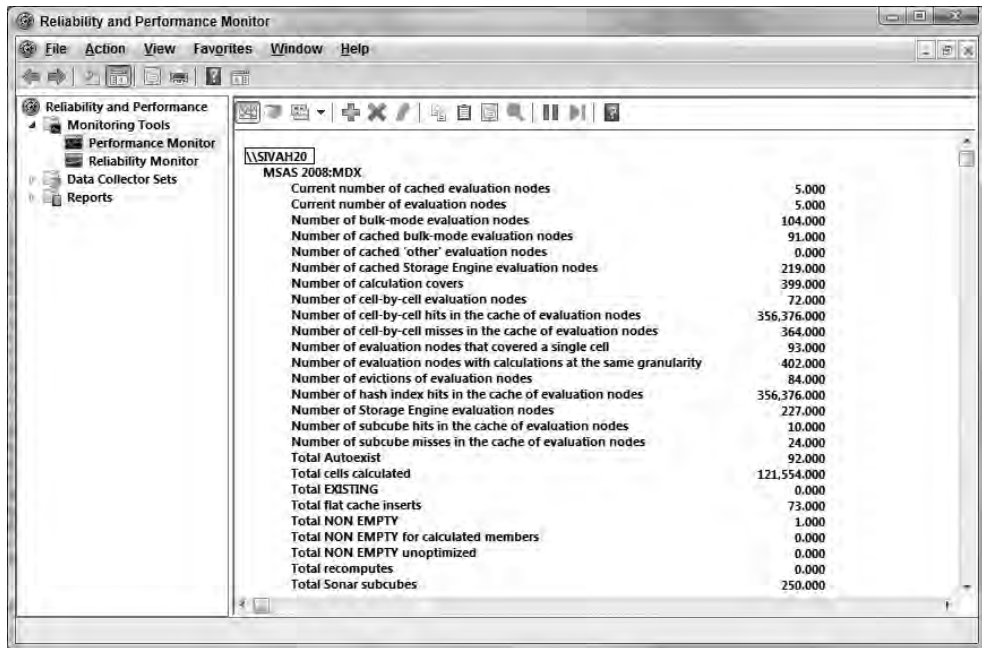


Figure 15-18

Task Manager

Most of you have used the Task Manager on your computer to look at the percentage of CPU time or memory consumed by a process. You can get the same information for Analysis Services using the Task Manager as shown in Figure 15-19. The process `msmdsrv.exe` is the Analysis Services 2008 process. If you have multiple instances of Analysis Services installed you will see multiple instances of `msmdsrv.exe` in Task Manager. You can also see the various instances of Analysis Services on your machine using the Services tab in Task Manager. The Task Manager gives you a quick way to understand if your Analysis Services server is CPU-intensive or its memory usage is growing when you have executed a long-running query.

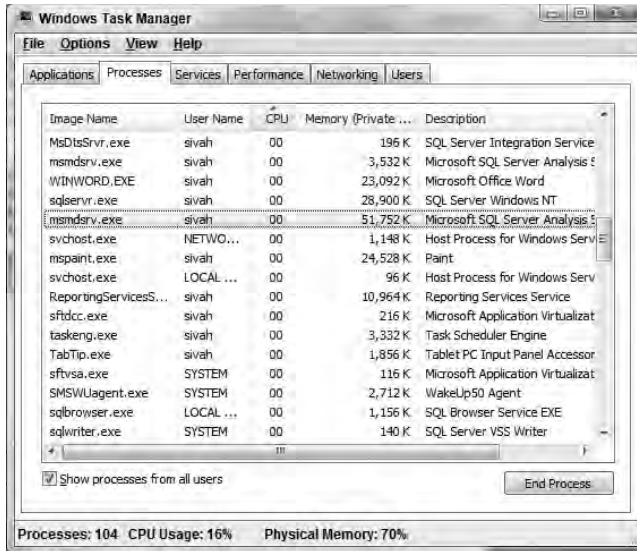


Figure 15-19

SQL Server Profiler, Performance Monitoring counters, and Task Manager help you analyze and troubleshoot issues with your Analysis Services. SQL Server Management Studio and Business Intelligence Development Studio are tools that help you tune your Analysis Services instance.

SQL Server Management Studio

You can use SSMS to execute your MDX queries and get the query execution time or look at the query results. You can also use it along with the Profiler to troubleshoot specific query issues. In addition, you can also use SSMS for debugging processing issues or tuning your Analysis Services server processing options. SSMS also helps you define aggregations to help speed up query performance. Other important uses of SSMS are changing your Analysis Services server properties, fine-tuning engine behavior, and restarting your Analysis Services service if needed.

Business Intelligence Development Studio

You can use BIDS to refine your cube and dimension design based on the troubleshooting you have done using other tools (discussed in Chapters 5, 6, 8, and 9). In addition, BIDS helps you build custom aggregations and make use of usage-based optimization, which helps you improve query performance.

Analyzing Query Performance Issues

Analysis Services 2008 has significant query optimization features. However, there are still factors that can affect query performance such as the complexity of the cube's design, aggregations, server configuration properties, hardware resources, and so on. Before you start analyzing query performance you need to understand where time is being spent during the overall execution. You already learned that there are two major components, Formula Engine (FE) and Storage Engine (SE), where the majority of

the execution time is being spent. The time spent in the infrastructure component is negligible and hence we can arrive at the following equation:

```
MDX Query execution time = Formula Engine time + Storage Engine time
```

In the “SQL Server Profiler” section earlier in the chapter, you learned that query subcube events indicate requests to the SE. Hence the SE time is the duration of time spent for all the query subcube events. The overall query execution time for the query can be obtained from the SQL Server Profiler trace. The time spent by the query in the FE component is equal to the difference of total execution time minus the SE time. These relationships are expressed in the following equations:

```
Storage Engine time = Time needed to evaluate all query subcube events
```

```
Formula Engine time = Total query execution time - Storage Engine time
```

Assuming you want to analyze and optimize your query execution time, we recommend that you focus your efforts on the following recommendations:

- ❑ If the SE time is greater than 30 percent of the total execution time, look at optimizing it.
- ❑ If the FE time is greater than 30 percent of the total execution time, look at optimizing it.
- ❑ If both FE and SE times are greater than 30 percent of the total execution time, then look at optimizing both areas.

Understanding FE and SE Characteristics

The FE performs the evaluation of the results and sends the results back to the client. This component is mostly single-threaded and CPU-intensive because it might have to iterate over millions of cells to perform calculations. If you observe, using Task Manager, that Analysis Services is consuming 100 percent of one of the processors during a query evaluation, then you can assume that the time is being spent in the Formula Engine. The FE has very little disk utilization.

The SE retrieves data for subcubes from the SE cache or from disk when requested by the FE. Partition and dimension data is stored as segments that can be read in parallel. Hence the SE component is heavily multithreaded to maximize the hardware resources and perform I/O operations in parallel. The SE is CPU- and disk-intensive. Hence if you see all the processors of your machine utilized and heavy disk usage (using Task Manager or performance counters), you can be confident that the query is spending time in the SE component.

When analyzing query performance, one important thing you need to be aware of is predictability. Analysis Services caches data in the SE and FE components. In addition, you have caching done by the operating system for disk I/O and the multi-user environment that play a critical factor in query performance. Hence, executing the same MDX query a second time can result in improved performance due to caching in Analysis Services. The recommended approach is to investigate query performance in single-user mode. In addition, Analysis Services has the Clear Cache command that clears all the Analysis Services caches. This improves the predictability of query execution when you are investigating performance issues. The syntax for the Clear Cache statement is shown in the following code. You need to pass the database ID as input to the statement to clear the caches of a specific database.

```
<ClearCache xmlns="http://schemas.microsoft.com/analysiservices/2003/engine">  
  <Object>  
    <DatabaseID>MyDatabaseID</DatabaseID>  
  </Object>  
</ClearCache>
```


Operating system file caching can also impact query performance. To get repeatable results, you can shut down and restart the Analysis Services service or even the entire machine if needed. In most cases you should be able to get repeatable results using the Clear Cache statement.

Common Solutions for Slow Queries

MDX query execution time is the sum of time spent in the FE and SE components. The issues causing queries to be slow can be classified into three main categories: large SE requests, multiple SE requests, and FE-intensive queries.

Large Storage Engine Requests

As you learned earlier, query evaluation plans are decided by the FE. A large SE request translates to a subcube query that takes a really long time. This means the majority of the query execution time is being spent getting results for a single SE request. An SE request can take a very long time due to factors such as having a very large partition, no aggregations, or aggregations getting missed. You need to follow the best practices mentioned in Chapter 14 to design the right cube, including defining effective attribute relationships, adopting an effective partitioning scheme, and designing aggregations using Aggregation Designer or usage-based optimization. These will help resolve the issue of an MDX query being slow due to a large storage engine request.

Several Storage Engine Requests

If you see several subcube query events in the SQL Server Profiler when you execute an MDX query repeatedly, it means that the SE caches are being missed each time and hence the SE component has to retrieve data from the disk. Retrieving data from disk is an expensive operation compared to getting the data from the SE caches. The EventSubclass property of the Query Subcube Verbose event shows whether the query is retrieved from cache or non-cache data. If the query is retrieved from non-cache data, the data is being retrieved from disk. Analysis Services 2008 provides you a way to forcibly cache the data in SE component using the Create Cache statement. The syntax of the statement is

```
CREATE CACHE FOR <CubeName> AS <MDX Expression>
```

The Create Cache statement applies to a specific cube. This is extremely useful in cases where you are aware of long-running queries due to several storage engine requests. You can “warm up” the Analysis Services SE cache using this statement, which can help improve performance of MDX queries using this cache.

Formula Engine-Intensive Query

The FE component is single threaded. Hence, if an MDX query contains intensive calculations, it could spend a significant amount of its execution time in the FE component. You should be able to identify an FE-intensive query using Task Manager. Look for msmdsrv.exe pegging one CPU on your machine at 100 percent. One of the critical factors in getting the best performance from your MDX query is to make sure your query uses the subspace computation code path. Looking at MDX performance counters and SQL Server Profiler traces should help you identify if your queries are not using subspace computation. In addition, other MDX query optimization techniques can help you reduce the time spent in the FE component, which you learn about in the next section.

Figure 15-20 provides a summary of the three categories of problems that can contribute to slow MDX queries and what techniques to investigate to improve query performance.

Scenario	Large SE Request	Several SE Requests	FE Intensive Query	
Solution	Partitioning, aggregations	CREATE CACHE	Cell-by-cell → Subspace	Optimizations: NEB, Auto-exists, Scope, MemberValue ...

Figure 15-20

Query Optimization Techniques

As you learned earlier in this chapter, MDX query execution time can be impacted by several factors such as cube design, Analysis Services caching, and hardware. One of the important factors in getting the best MDX query execution time is the efficiency of your MDX. Using the right MDX query optimization technique is not simple and involves a deeper understanding of your cube and MDX. In this section you learn some of the important techniques that can help you optimize your MDX queries.

Using *NON EMPTY* on Axes

Most cubes are quite sparse. By sparse we mean that many of the cells in the cube space do not have a value associated with them. For example, in the Adventure Works DW 2008 sample Analysis Services database, if every coordinate of the Internet Sales measure group has data and assuming only the key attribute in each dimension, the total number of cells with data would be (Date) 1189 * Date (Ship Date) 1189 * Date (Delivery Date) 1189 * Customer (18485) * Promotion (17) * Product (398) * Sales Territory (12) * Sales Reason (11) * Source Currency (106) * Destination Currency (15) * Internet Sales Order Details (60,399), which is 2.66×10^{27} cells. This result increases when additional attributes are added from each dimension. Although most of the cells do not have any business meaning associated with them — for example, if delivery date is ahead of order date — they belong to cube space and can be queried by the users. Querying such cells results in a null value, which indicates that data is not available for that cell's coordinates.

The fact table rows represent the leaf-level cells for a cube. The fact table row count is much less than possible cube space. The Analysis Services engine has many optimizations for improving query performance by limiting the search space. The basic rule is that if a cube doesn't have calculations (such as calculated scripts, custom rollup, and custom members), the non-empty space of the cube is defined by fact table cells and their aggregations. Analysis Services allows users to write effective, optimized MDX queries to prevent empty cells from being returned. This is because those empty cells simply do not add value for business analysis. By limiting the search space, Analysis Services can find the results much more quickly.

Analysis Services 2008 supports many ways for users to eliminate cells containing null values in a query. The keyword *NON EMPTY* eliminates members along an axis whose cell values are null. The *NON EMPTY* keyword is used at the beginning of the axis statement in an MDX query as shown here:

```
SELECT Measures.Members on COLUMNS,
NON EMPTY Dimension.Hierarchy.Members on ROWS
From <CubeName>
```

Part III: Advanced Administration and Performance Optimization

The NON EMPTY keyword can be used on rows or columns (or any axis). In most cases, only results with non-empty cells are meaningful for end users. Hence, most Analysis Services 2008 client tools generate MDX queries with the NON EMPTY keyword. We recommend that you use the NON EMPTY keyword in your MDX cell set and row set queries whenever possible. Not only will it limit the size of the returned cell set, but additional optimizations are applied when you do this that will speed up your query execution time.

Following is an MDX query without the NON EMPTY keyword. Execute this query using SQL Server Management Studio against a deployed sample Adventure Works project.

```
SELECT [Customer].[Customer Geography].[Customer].members *
    Descendants([Product].[Product Categories].[Category].&[3],[Product].
[Product Categories].[Product Name]) ON 1,
    {[Measures].[Internet Sales Amount]} ON 0
FROM [Adventure Works]
```

The query returns 18,485 cells. Now change the query to include the NON EMPTY keyword on both axes as shown here and execute the new query in SQL Server Management Studio:

```
SELECT NON EMPTY [Customer].[Customer Geography].[Customer].members *
    Descendants([Product].[Product Categories].[Category].&[3],[Product].
[Product Categories].[Product Name]) ON 1,
    {[Measures].[Internet Sales Amount]} ON 0
FROM [Adventure Works]
```

This query, which includes the NON EMPTY keyword, returns just 6,853 cells, which is a reduced number of cells to evaluate. The execution time for the query with NON EMPTY is lower than that of the query without NON EMPTY. We recommend that you follow these steps to observe the performance:

1. Connect to the sample Adventure Works 2008 Analysis Services database.
2. Start SQL Server Profiler.
3. Create a New Trace with Query Begin and Query End events selected.
4. Send the Clear Cache statement.
5. Send the query without NON EMPTY.
6. Send the Clear Cache statement.
7. Send the MDX query with NON EMPTY.
8. Observe the Duration column to see the performance difference between the two queries.

You can see the performance difference in duration times between the two queries as shown in Figure 15-21. This example highlights the benefit of eliminating empty cells using NON EMPTY.

SessionID	StartTime	TextData	CPUTime	Duration	EndTime
9CA29F...	2008-10-01 07:20:22...	SELECT [Customer].[Customer Geograp...		0	2008-10-01 07:20:22...
9CA29F...	2008-10-01 07:20:29...	SELECT [Customer].[Customer Geograp...	484	1467	2008-10-01 07:20:29...
9CA29F...	2008-10-01 07:21:18...	SELECT NON EMPTY [Customer].[Custom...	172	704	2008-10-01 07:21:18...

SELECT NON EMPTY [Customer].[Customer Geography].[Customer].members * Descendants([Product].[Product Categories].[Category].&[3],[Product].[Product Name]) ON 1, ([Measures].[Internet Sales Amount]) ON 0 FROM [Adventure Works]

Trace is running. Ln 6, Col 1 Rows: 6 Connections: 1

Figure 15-21

Using Non Empty for Filtering and Sorting

Many users apply filter conditions on a set or try to evaluate the top *N* members of a set based on certain conditions using the Filter and TopCount functions, respectively. In most cases, only non-empty members are needed in the results of the Filter and TopCount functions. You can improve the performance dramatically by first using NONEMPTY() to retrieve non-empty sets, followed by the Filter, Sort, or TopCount functions on the smaller set. In the Adventure Works sample, for example, if you want to get the top ten Customer/Product combinations to start a marketing campaign, your query will look like the following:

```
SELECT
TopCount([Customer].[Customer Geography].[Customer].members*
[Product].[Product Categories].[Product].members, 10 ,
[Measures].[Internet Sales Amount]) ON ROWS ,
[Measures].[Internet Sales Amount] ON COLUMNMS
FROM [Adventure Works]
```

Notice this query contains a cross-join of customers and products (shown by the following expression). Whenever a cross-join is applied, the server sorts the result based on the order of the hierarchies.

```
([Customer].[Customer Geography].[Customer].members*[Product].[
Product Categories].[Product].members)
```

The cross-join of the customer and product dimension results in $18484 * 397 = 7,338,148$ cells. Analysis Services now evaluates the top 10 cells out of the seven million cells to return the results for the preceding query. This query took around 48 seconds on the machine we used to run the query and it consumed 1 CPU at 100 percent during the entire execution. Most of the cells of the cross-join were actually empty cells that need not have been part of the result of the cross-join. Not only did the server take the

time in sorting these cells, but it also had to iterate through the seven million cells to determine the top 10. The following query uses the `NonEmptyCrossJoin` function, which eliminates the empty cells:

```
SELECT
TopCount (NONEMPTYCROSSJOIN (
  [Customer].[Customer Geography].[Customer].members*
  [Product].[Product Categories].[Product].members,
  {[Measures].[Internet Sales Amount]},1),10,
  [Measures].[Internet Sales Amount]) ON ROWS ,
  [Measures].[Internet Sales Amount] ON COLUMNS
FROM [Adventure Works]
```

In this query, the `NonEmptyCrossJoin` function first eliminates all empty cells, and hence the `TopCount` function had a smaller set of cells to work on. The query took 3 seconds on the same machine used in the previous example because of the optimization provided by using the `NonEmptyCrossJoin` function. Only cells containing fact data were sorted and the top 10 values were returned. The performance improvement is dramatic (can be observed in SSMS or in the SQL Server Profiler duration column) and both queries returned the exact same results. The rule of thumb is that the fewer tuples or cells involved in calculations, the better the query performance. Because Analysis Services has an efficient algorithm to get non-empty sets, you should use `NonEmpty` whenever it is applicable and appropriate for your business requirements. You can use the `NonEmptyCrossJoin` function whenever you are aware that a real measure will be used by the server for Non-Empty evaluation, but use it with caution when you have calculated measures because certain optimization may not be available for all calculated measures. You can also use the `HAVING` clause, which eliminates cells with null values as shown in Chapter 10.

Using `NON_EMPTY_BEHAVIOR` for Calculations

The `NON EMPTY` keyword checks the fact data to determine empty cells. However, if cell values are the result of calculations, `Non Empty` can be slow. If you query for cells that involve evaluation of complex calculations, then the cells' emptiness (if the cell returns a null value) is not determined by fact data; each cell must be evaluated to return the correct results. Analysis Services provides you with a keyword called `NON_EMPTY_BEHAVIOR` to instruct the server to use an optimized algorithm to determine cells' emptiness. The following query returns the forecast sales by applying different rates:

```
WITH MEMBER [Measures].[ForecastSales] AS
'iif([Measures].[Internet Sales Amount] >500 ,
  [Measures].[Internet Sales Amount]*1.2,
  [Measures].[Internet Sales Amount]*1.2) '
SELECT NON EMPTY [Customer].[Customer Geography].[Customer].members*
  Descendants([Product].[Product Categories].[Category].&[3],[Product].
  [Product Categories].[Product]) ON 1 ,
  NON EMPTY {[Measures].[ForecastSales]} ON 0
FROM [Adventure Works]
```

Even though this query uses `NON EMPTY`, MDX queries with calculations like this one can be slow. This is because the optimized code path is not applied on complex calculated members. In this query you have a calculated member that is multiplied by 1.2 and hence the server needs to evaluate the expression to identify if the corresponding cells are empty. In order to have Analysis Services apply `NON EMPTY` behavior to the calculated member, you can specify the `NON_EMPTY_BEHAVIOR` property, which ties the calculated measure to a real fact measure. The server will then use the optimized

code path for the non-empty determination. Execute the following modified query that specifies NON_EMPTY_BEHAVIOR:

```
WITH MEMBER [Measures].[ForecastSales] AS
  'iif([Measures].[Internet Sales Amount] >500 ,
    [Measures].[Internet Sales Amount]*1.2,
    [Measures].[Internet Sales Amount]*1.2) ',
NON_EMPTY_BEHAVIOR = '[Measures].[Internet Sales Amount] '
SELECT NON EMPTY [Customer].[Customer Geography].[Customer].members*
  Descendants([Product].[Product Categories].[Category].&[3],[Product].
  [Product Categories].[Product]) ON 1 ,
NON EMPTY {[Measures].[ForecastSales]} ON 0
FROM [Adventure Works]
```

Here you have provided a hint to Analysis Services to use the [Internet Sales Amount] measure while evaluating the calculation. Such hints help reduce the query execution time because Analysis Services is able to determine if the calculation returns a null value using the base measure.

Using SCOPE versus IIF and CASE

You learned about the SCOPE, IIF, and CASE statements in Chapter 10. Using SCOPE helps improve query performance when evaluating cells compared to the IIF and CASE statements. When using SCOPE, calculations only get applied to the subcube, compared to other calculations, which get evaluated for the entire cube space. In addition, SCOPE statements are evaluated once statically, compared to IIF/CASE, which are evaluated dynamically. These two factors contribute to improving query performance when using SCOPE. The following code is an example of an MDX expression that is translated from IIF to SCOPE:

```
CREATE MEMBER Measures.[Sales Amount] AS
  IIF([Destination Currency].CurrentMember IS Currency.USD,
  Measures.[Internet Sales Amount], Measures.[Internet Sales Amount] *
  Measures.AverageRate);

CREATE MEMBER Measures.[Sales Amount] AS Null;
SCOPE(Measures.[Sales], [Destination Currency].Members);
  THIS = Measures.[Internet Sales Amount] * Measures.AverageRate;
SCOPE(Currency.USA);
  THIS = Measures.[Internet Sales Amount];
END SCOPE;
END SCOPE;
```

Auto Exists versus Properties

When you include attributes and hierarchies within a dimension in a query, Analysis Services only returns the relevant members. For example, take the following MDX query:

```
SELECT [Measures].[Internet Sales Amount] ON 0,
  [Customer].[City].&[Seattle]&[WA] * [Customer].[State-Province].MEMBERS ON 1
FROM [Direct Sales]
```

Part III: Advanced Administration and Performance Optimization

This query only returns results for (Seattle, Washington) and (Seattle, All Customers). It does not return the complete cross product of Seattle and all the States in the Customer.[State-Province] hierarchy. As you learned in Chapter 10, this behavior is called *auto exists* and helps improve performance. Hence we recommend using *Exists* or *CrossJoin* functions instead of using the *Properties* function in your MDX expression. The following code is an example of how you can rewrite your MDX expressions that use the *Properties* function:

```
Filter(Customer.Members,
    Customer.CurrentMember.Properties("Gender") = "Male")

Exists(Customer.Members, Gender.[Male])
```

Member Value versus Properties

Analysis Services 2008 has an attribute member property called *Value Column*. When defined, this is helpful in retrieving the values in a typed format. For example, if you have the yearly income of a customer, you can retrieve its value as integer rather than as a string and then converting it using one of the VBA functions. Here is an example of how to use the *MemberValue* MDX function to retrieve the *Value Column*:

```
Create Set [Adventure Works].RichCustomers As
    Filter(Customer.Customer.Members,
        CInt(Customer.CustomerCurrentMember.Properties("Yearly Income"))
        > 100000);

Create Set RichCustomers As
    Filter(Customer.Customer.Members,
        Customer.Salary.MemberValue > 100000);
```

In this example, the first expression creates a set of customers whose *Yearly Income* is greater than 100000 using the *Properties* MDX function, which retrieves the member property. The return type for the *MemberProperty* function is a string and you need to use the *CInt* VBA function to convert this to an integer value before you compare it with 100000. Please note that the preceding example is provided for illustration purposes only. The actual data in the sample *Adventure Works 2008* in the *Yearly Income* is a range represented as string. We recommend you try the preceding illustration on a large database with appropriate data to see the benefits. When you have a large number of customers, converting strings to integers becomes expensive. The second MDX expression uses the *MemberValue* MDX function to retrieve the value directly as an integer.

Move Simple Calculations to Data Source View

If there are very simple static calculations such as converting based on exchange rates, these calculations can be changed to calculated columns in the *Data Source View (DSV)*. Analysis Services does these calculations at processing time and stores the values in the cube rather than calculating these expressions during query execution time. These simple calculations should be really fast in Analysis Services 2008 and you may not observe the performance hit. However, as a general best practice, we recommend that you move them to DSV.

Features versus MDX Scripts

Analysis Services 2008 provides features such as many-to-many dimensions, measure expressions, unary operators, custom rollup, and semi-additive measures. You have learned about these features and how and when to use them during the course of this book. Almost all of these features can be defined as MDX expressions in MDX script. We highly recommend that you design your cubes using built-in features rather than defining the equivalent functionality as MDX expressions in MDX Script. The built-in features will provide better performance for most scenarios. If you do find a specific feature causing query performance degradation, you can re-visit implementing the functionality in MDX scripts.

We have looked at some of the common problems the Analysis Services team has observed while investigating customer performance issues and how to solve them in this section. There are additional MDX optimizations that you can perform. We recommend that you look at the following resources for additional information on MDX optimizations:

- ❑ *MDX Solutions: With Microsoft SQL Server Analysis Services*, 2nd edition by George Spofford et al., (Wiley, 2006), www.wiley.com
- ❑ The SQL Customer Advisory Team, <http://www.sqlcat.com>
- ❑ Microsoft OLAP by Mosha Pashumansky, <http://sqlblog.com/blogs/mosha>
- ❑ Richard Tkachuk's Analysis Services Page, <http://www.sqlserveranalysisisservices.com>

Scale Out with Read-Only Database

After performing all these query optimization techniques you might still find query performance degradation when multiple users are connected to your Analysis Services instance and actively querying the database. This is one of the problems Analysis Services customers face when the customer load on a specific database increases. You can try to use larger machines with more CPUs if memory or CPU is the bottleneck. You can also move to 64-bit machines if you are currently using 32-bit hardware. Of course, as you scale up, the cost of your machines will become higher. Analysis Services 2008 provides the new read-only database feature, which is discussed in Chapter 7. If your customer needs are only to improve query performance and your customers are only performing read-only queries (no updates, no writeback), you can use the read-only database feature and create a scale out strategy as shown in Figure 15-21 to improve query performance for this type of multi-user scenario.

We recommend that you have multiple Analysis Services 2008 servers configured to read from a single database on a shared SAN (Storage Area Network) to form Scalable Shared Databases that can be queried by multiple users. These servers need to be load balanced using a network load balancer as shown in Figure 15-22. You need a separate isolated machine for processing the database when there are data updates. Once the database has been processed you can detach the database and copy it to your SAN. You can then utilize XMLA scripts to attach the database to the Analysis Services query servers in read-only mode. All the query servers will have an identical copy of the database and be able to serve multiple users. This scale out strategy will help you improve query performance for this type of multiple user scenario.

Scalable Shared Databases

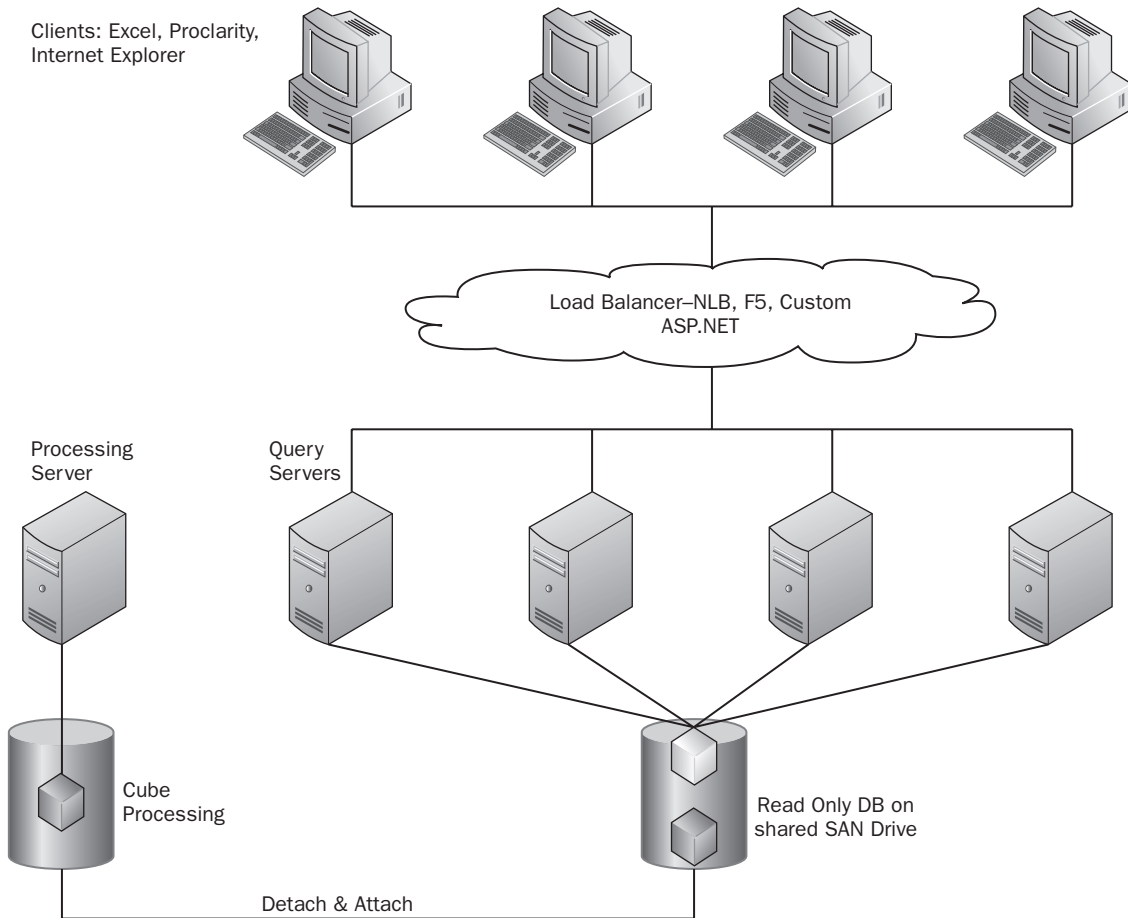


Figure 15-22

Writeback Query Performance

As you learned in Chapter 12, with Analysis Services 2008 you can obtain improved writeback performance by enabling the MOLAP storage option when you enable writeback. When you perform cell writeback, Analysis Services writes data back to the relational table specified. In addition, the MOLAP partition associated with the writeback partition is reprocessed automatically. Because of this, all queries using the writeback partition will retrieve the data from the MOLAP storage rather than fetching the data from the relational table and then aggregating the data. Thus, using MOLAP storage for the writeback partition helps improve query performance. We recommend that you set MOLAP storage mode for the writeback partition when you enable the cube for cell writeback.

Summary

In this chapter you first learned about the calculation model in Analysis Services, followed by the query execution architecture of Analysis Services. You learned about the various tools that can be used to investigate the performance of Analysis Services. The SQL Server Profiler and perfmon counters in particular are very valuable tools that can help you investigate performance bottlenecks. You then learned about various classes of problems that can contribute to slow queries, along with recommendations on how to solve these problems. Finally, you learned about important query optimization techniques and best practices that can help you fine-tune your MDX queries. After reading Chapters 14 and 15, upon hearing the very word *performance*, your head should swell with visions of highly performing, scalable systems, each optimized to fulfill its designated mission!

