Microsoft SQL Server Analysis Services Multidimensional Performance and Operations Guide

Thomas Kejser and Denny Lee

Guide

Microsoft

Microsoft SQL Server Analysis Services Multidimensional Performance and Operations Guide

Thomas Kejser and Denny Lee

Contributors and Technical Reviewers: Peter Adshead (UBS), T.K. Anand, KaganArca, Andrew Calvett (UBS), Brad Daniels, John Desch, Marius Dumitru, WillfriedFärber (Trivadis), Alberto Ferrari (SQLBI), Marcel Franke (pmOne), Greg Galloway (Artis Consulting), Darren Gosbell (James & Monroe), DaeSeong Han, Siva Harinath, Thomas Ivarsson (Sigma AB), Alejandro Leguizamo (SolidQ), Alexei Khalyako, Edward Melomed, AkshaiMirchandani, Sanjay Nayyar (IM Group), TomislavPiasevoli, Carl Rabeler (SolidQ), Marco Russo (SQLBI), Ashvini Sharma, Didier Simon, John Sirmon, Richard Tkachuk, Andrea Uggetti, Elizabeth Vitt, Mike Vovchik, Christopher Webb (Crossjoin Consulting), SedatYogurtcuoglu, Anne Zorner

Summary: Download this book to learn about Analysis Services Multidimensional performance tuning from an operational and development perspective. This book consolidates the previously published SQL Server 2008 R2 Analysis Services Operations Guide and SQL Server 2008 R2 Analysis Services Performance Guide into a single publication that you can view on portable devices.

Category: Guide
Applies to: SQL Server 2005, SQL Server 2008, SQL Server 2008 R2, SQL Server 2012
Source: White paper (link to source content, link to source content)
E-book publication date: May 2012
200 pages

Microsoft

This page intentionally left blank

Contents

1 Introduction

This book consolidates two previously published guides into one essential resource for Analysis Services developers and operations personnel. Although the titles of the original publications indicate SQL Server 2008 R2, most of the knowledge that you gain from this book is easily transferred to other versions of Analysis Services, including multidimensional models built using SQL Server 2012.

Part 1 is from the "SQL Server 2008 R2 Analysis Services Performance Guide". Published in October 2011, this guide was created for developers and cube designers who want to build high-performance cubes using best practices and insights learned from real-world development projects. In Part 1, you'll learn proven techniques for building solutions that are faster to process and query, minimizing the need for further tuning down the road.

Part 2 is from the "SQL Server 2008 R2 Analysis Services Operations Guide". This guide, published in June 2011, is intended for developers and operations specialists who manage solutions that are already in production. Part 2 shows you how to extract performance gains from a production cube, including changing server and system properties, and performing system maintenance that help you avoid problems before they start.

While each guide targets a different part of a solution lifecycle, having both in a single portable format gives you an intellectual toolkit that you can access on mobile devices wherever you may be. We hope you find this book helpful and easy to use, but it is only one of several formats available for this content. You can also get printable versions of both guides by downloading them from the Microsoft web site.

2 Part 1: Building a High-Performance Cube

This section provides information about building and tuning Analysis Services cubes for the best possible performance. It is primarily aimed at business intelligence (BI) developers who are building a new cube from scratch or optimizing an existing cube for better performance.

The goal of this section is to provide you with the necessary background to understand design tradeoffs and with techniques and design patterns that will help you achieve the best possible performance of even large cubes.

Cube performance can be divided into two types of workload: query performance and processing performance. Because these workloads are very different, this sectionis organized into four main groups.

Design Patterns for Scalable Cubes – No amount of query tuning and optimization can beat the benefits of a well-designed data model. This section contains guidance to help you get the design right the first time. In general, good cube design follows Kimball modeling techniques, and if you avoid some typical design mistakes, you are in very good shape.

Testing Analysis Services Cubes – In every IT project, preproduction testing is a crucial part of the development and deployment cycle. Even with the most careful design, testing will still be able to shake out errors and avoid production issues. Designing and running a test run of an enterprise cube is time well invested. Hence, this section includes a description of the test methods available to you.

Tuning Query Performance - Query performance directly impacts the quality of the end-user experience. As such, it is the primary benchmark used to evaluate the success of an online analytical processing (OLAP) implementation. Analysis Services provides a variety of mechanisms to accelerate query performance, including aggregations, caching, and indexed data retrieval. This section also provides guidance on writing efficient Multidimensional Expressions (MDX) calculation scripts.

Tuning Processing Performance - Processing is the operation that refreshes data in an Analysis Services database. The faster the processing performance, the sooner users can access refreshed data. Analysis Services provides a variety of mechanisms that you can use to influence processing performance, including parallelized processing designs, relational tuning, and an economical processing strategy (for example, incremental versus full refresh versus proactive caching).

Special Considerations – Some features of Analysis Services such as distinct count measures and many-to-many dimensions require more careful attention to the cube design than others. At the end of Part 1, you will find a section that describes the special techniques you should apply when using these features.

2.1 Design Patterns for Scalable Cubes

Cubes present a unique challenge to the BI developer: they are ad-hoc databases that are expected to respond to most queries in short time. The freedom of the end user is limited only by the data model you implement. Achieving a balance between user freedom and scalable design will determine the

success of a cube. Each industry has specific design patterns that lend themselves well to value adding reporting – and a detailed treatment of optimal, industry specific data model is outside the scope of this book. However, there are a lot of common design patterns you can apply across all industries - this section deals with these patterns and how you can leverage them for increased scalability in your cube design.

2.1.1 Building Optimal Dimensions

A well-tuned dimension design is one of the most critical success factors of a high-performing Analysis Services solution. The dimensions of the cube are the first stop for data analysis and their design has a deep impact on the performance of all measures in the cube.

Dimensions are composed of attributes, which are related to each other through hierarchies. Efficient use of attributes is a key design skill to master, and studying and implementing the attribute relationships available in the business model can help improve cube performance.

In this section, you will find guidance on building optimized dimensions and properly using both attributes and hierarchies.

2.1.1.1 Using the KeyColumns, ValueColumn, and NameColumn Properties Effectively

When you add a new attribute to a dimension, three properties are used to define the attribute. The **KeyColumns** property specifies one or more source fields that uniquely identify each instance of the attribute.

The **NameColumn** property specifies the source field that will be displayed to end users. If you do not specify a value for the **NameColumn** property, it is automatically set to the value of the **KeyColumns** property.

ValueColumn allows you to carry further information about the attribute – typically used for calculations. Unlike member properties, this property of an attribute is strongly typed – providing increased performance when it is used in calculations. The contents of this property can be accessed through the **MemberValue** MDX function.

Using both **ValueColumn** and **NameColumn** to carry information eliminates the need for extraneous attributes. This reduces the total number of attributes in your design, making it more efficient.

It is a best practice to assign a numeric source field, if available, to the **KeyColumns** property rather than a string property. Furthermore, use a single column key instead of a composite, multi-column key. Not only do these practices this reduce processing time, they also reduce the size of the dimension and the likelihood of user errors. This is especially true for attributes that have a large number of members, that is, greater than one million members.

2.1.1.2 Hiding Attribute Hierarchies

For many dimensions, you will want the user to navigate hierarchies created for ease of access. For example, a customer dimension could be navigated by drilling into country and city before reaching the customer name, or by drilling through age groups or income levels. Such hierarchies, covered in more detail later, make navigation of the cube easier – and make queries more efficient.

In addition to user hierarchies, Analysis Services by default creates a flat hierarchy for every attribute in a dimension – these are attribute hierarchies. Hiding attribute hierarchies is often a good idea, because a lot of hierarchies in a single dimension will typically confuse users and make client queries less efficient. Consider setting **AttributeHierarchyVisible** = **false** for most attribute hierarchies and use user hierarchies instead.

2.1.1.2.1 Hiding the Surrogate Key

It is often a good idea to hide the surrogate key attribute in the dimension. If you expose the surrogate key to the client tools as a **ValueColumn**, those tools may refer to the key values in reports. The surrogate key in a Kimball star schema design holds no business information, and may even change if you remodel type2 history. After you create a dependency to the key in the client tools, you cannot change the key without breaking reports. Because of this, you don't want end-user reports referring to the surrogate key directly – and this is why we recommend hiding it.

The best design for a surrogate key is to hide it from users in the dimension design by setting the **AttributeHierarchyVisible** = **false** and by not including the attribute in any user hierarchies. This prevents end-user tools from referencing the surrogate key, leaving you free to change the key value if requirements change.

2.1.1.3 Setting or Disabling Ordering of Attributes

In most cases, you want an attribute to have an explicit ordering. For example, you will want a City attribute to be sorted alphabetically. You should explicitly set the **OrderBy** or **OrderByAttribute** property of the attribute to explicitly control this ordering. Typically, this ordering is by attribute name or key, but it may also be another attribute. If you include an attribute only for the purpose of ordering another attribute, make sure you set **AttributeHierarchyEnabled** = **false** and **AttributeHierarchyOptimizedState** = **NotOptimized** to save on processing operations.

There are few cases where you don't care about the ordering of an attribute, yet the surrogate key is one such case. For such hidden attribute that you used only for implementation purposes, you can set **AttributeHierarchyOrdered** = **false** to save time during processing of the dimension.

2.1.1.4 Setting Default Attribute Members

Any query that does not explicitly reference a hierarchy will use the current member of that hierarchy.The default behavior of Analysis Services is to assign the All member of a dimension as the default member, which is normally the desired behavior. But for some attributes, such as the current

day in a date dimension, it sometimes makes sense to explicitly assign a default member. For example, you may set a default date in the Adventure Works cube like this.

```
ALTERCUBE [Adventure Works]UPDATE
DIMENSION [Date], DEFAULT_MEMBER='[Date].[Date].&[2000]'
```

However, default members may cause issues in the client tool. For example, Microsoft Excel 2010 will not provide a visual indication that a default member is currently selected and hence implicitly influence the query result. This may confuse users who expect the **All** level to be the current member when no other members are implied by the query. Also, if you set a default member in a dimension with multiple hierarchies, you will typically get results that are hard for users to interpret.

In general, prefer explicitly default members only on dimensions with single hierarchies or in hierarchies that do not have an **All** level.

2.1.1.5 Removing the All Level

Most dimensions roll up to a common **All** level, which is the aggregation of all descendants. But there are some exceptions where is does not make sense to query at the **All** level. For example, you may have a currency dimension in the cube – and asking for "the sum of all currencies" is a meaningless question. It can even be expensive to ask for the **All** level of dimension if there is not good aggregate to respond to the query. For example, if you have a cube partitioned by currency, asking for the **All** level of currency will cause a scan of all partitions, which could be expensive and lead to a useless result.

In order to prevent users from querying meaningless **All** levels, you can disable the All member in a hierarchy. You do this by setting the **IsAggregateable** = **false** on the attribute at the top of the hierarchy. Note that if you disable the **All** level, you should also set a default member as described in the previous section– if you don't, Analysis Services will choose one for you.

2.1.1.6 Identifying Attribute Relationships

Attribute relationships define hierarchical dependencies between attributes. In other words, if A has a related attribute B, written A ➔ B, there is one member in B for every member in A, and many members in A for a given member in B. For example, given an attribute relationship City ➔ State, if the current city is Seattle, we know the State must be Washington.

Often, there are relationships between attributes that might or might not be manifested in the original dimension table that can be used by the Analysis Services engine to optimize performance. By default, all attributes are related to the key, and the attribute relationship diagram represents a "bush" where relationships all stem from the key attribute and end at each other's attribute.

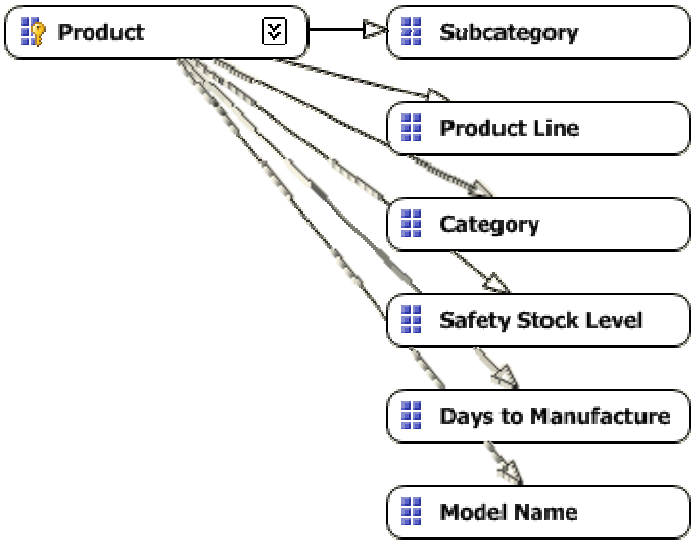


Figure 11: Bushy attribute relationships

You can optimize performance by defining hierarchical relationships supported by the data. In this case, a model name identifies the product line and subcategory, and the subcategory identifies a category. In other words, a single subcategory is not found in more than one category. If you redefine the relationships in the attribute relationship editor, the relationships are clearer.

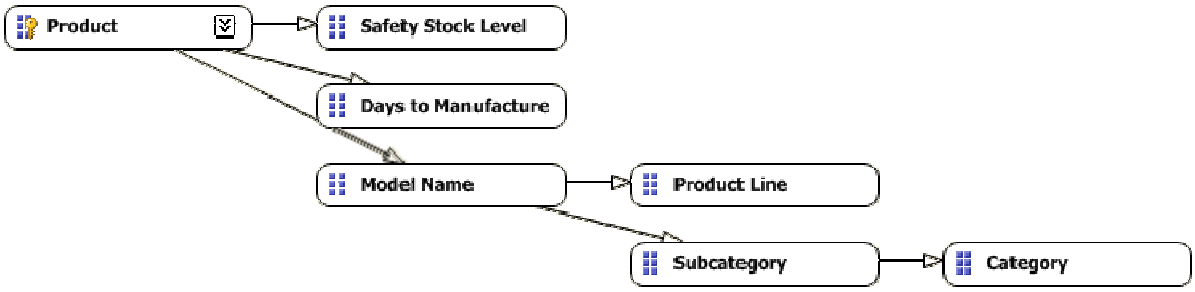


Figure 22: Redefined attribute relationships

Attribute relationships help performance in three significant ways:

- Cross products between levels in the hierarchy do not need to go through the key attribute. This saves CPU time during queries.
- Aggregations built on attributes can be reused for queries on related attributes. This saves resources during processing and for queries.
- Auto-Exist can more efficiently eliminate attribute combinations that do not exist in the data.

Consider the cross-product between **Subcategory** and **Category** in the two figures. In the first, where no attribute relationships have been explicitly defined, the engine must first find which products are in

each subcategory and then determine which categories each of these products belongs to. For large dimensions, this can take a long time. If the attribute relationship is defined, the Analysis Services engine knows beforehand which category each subcategory belongs to via indexes built at process time.

2.1.1.6.1 Flexible vs. Rigid Relationships

When an attribute relationship is defined, the relation can either be flexible or rigid. A flexible attribute relationship is one where members can move around during dimension updates, and a rigid attribute relationship is one where the member relationships are guaranteed to be fixed. For example, the relationship between month and year is fixed because a particular month isn't going to change its year when the dimension is reprocessed. However, the relationship between customer and city may be flexible as customers move.

When a change is detected during process in a flexible relationship, all indexes for partitions referencing the affected dimension (including the indexes for attribute that are not affected) must be invalidated. This is an expensive operation and may cause **Process Update** operations to take a very long time. Indexes invalidated by changes in flexible relationships must be rebuilt after a **Process Update** operation with a **Process Index** on the affected partitions; this adds even more time to cube processing.

Flexible relationships are the default setting. Carefully consider the advantages of rigid relationships and change the default where the design allows it.

2.1.1.7 Using Hierarchies Effectively

Analysis Services enables you to build two types of user hierarchies: natural and unnatural hierarchies. Each type has different design and performance characteristics.

In a natural hierarchy, all attributes participating as levels in the hierarchy have direct or indirect attribute relationships from the bottom of the hierarchy to the top of the hierarchy.

In an unnaturalhierarchy, the hierarchy consists of at least two consecutive levels that have no attribute relationships. Typically these hierarchies are used to create drill-down paths of commonly viewed attributes that do not follow any natural hierarchy. For example, users may want to view a hierarchy of Gender and Education.

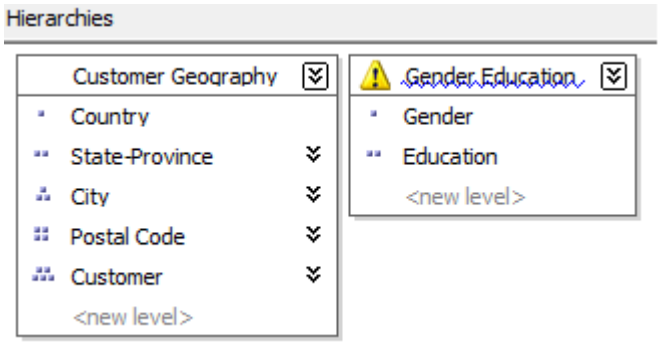


Figure 33: Natural and unnatural hierarchies

From a performance perspective, natural hierarchies behave very differently than unnatural hierarchies do. In natural hierarchies, the hierarchy tree is materialized on disk in hierarchy stores. In addition, all attributes participating in natural hierarchies are automatically considered to be aggregation candidates.

Unnatural hierarchies are not materialized on disk, and the attributes participating in unnatural hierarchies are not automatically considered as aggregation candidates. Rather, they simply provide users with easy-to-use drill-down paths for commonly viewed attributes that do not have natural relationships. By assembling these attributes into hierarchies, you can also use a variety of MDX navigation functions to easily perform calculations like percent of parent.

To take advantage of natural hierarchies, define cascading attribute relationships for all attributes that participate in the hierarchy.

2.1.1.8 Turning Off the Attribute Hierarchy

Member properties provide a different mechanism to expose dimension information. For a given attribute, member properties are automatically created for every direct attribute relationship. For the primary key attribute, this means that every attribute that is directly related to the primary key is available as a member property of the primary key attribute.

If you only want to access an attribute as member property, after you verify that the correct relationship is in place, you can disable the attribute's hierarchy by setting the **AttributeHierarchyEnabled** property to **False**. From a processing perspective, disabling the attribute hierarchy can improve performance and decrease cube size because the attribute will no longer be indexed or aggregated. This can be especially useful for high-cardinality attributes that have a one-to-one relationship with the primary key. High-cardinality attributes such as phone numbers and addresses typically do not require slice-and-dice analysis. By disabling the hierarchies for these attributes and accessing them via member properties, you can save processing time and reduce cube size.

Deciding whether to disable the attribute's hierarchy requires that you consider both the querying and processing impacts of using member properties. Member properties cannot be placed on a query axis in an MDX query in the same manner as attribute hierarchies and user hierarchies. To query a member property, you must query the attribute that contains that member property.

For example, if you require the work phone number for a customer, you must query the properties of customer and then request the phone number property. As a convenience, most front-end tools easily display member properties in their user interfaces.

In general, filtering measures using member properties is slower than filtering using attribute hierarchies, because member properties are not indexed and do not participate in aggregations. The actual impact to query performance depends on how you use the attribute.

For example, if your users want to slice and dice data by both account number and account description, from a querying perspective you may be better off having the attribute hierarchies in place and removing the bitmap indexes if processing performance is an issue.

2.1.1.9 Reference Dimensions

Reference dimensions allow you to build a dimensional model on top of a snow flake relational design. While this is a powerful feature, you should understand the implications of using it.

By default, a reference dimension is **non-materialized**. This means that queries have to perform the join between the reference and the outer dimension table at query time. Also, filters defined on attributes in the outer dimension table are not driven into the measure group when the bitmaps there are scanned. This may result in reading too much data from disk to answer user queries. Leaving a dimension as non-materialized prioritizes modeling flexibility over query performance. Consider carefully whether you can afford this tradeoff: cubes are typically intended to be fast ad-hoc structures, and putting the performance burden on the end user is rarely a good idea.

Analysis Services has the ability to materialize the references dimension. When you enable this option, memory and disk structures are created that make the dimension behave just like a denormalized star schema. This means that you will retain all the performance benefits of a regular, non-reference dimension. However, be careful with materialized reference dimension – if you run a process update on the intermediate dimension, any changes in the relationships between the outer dimension and the reference will *not* be reflected in the cube. Instead, the original relationship between the outer dimension and the measure group is retained – which is most likely not the desired result. In a way, you can consider the reference table to be a rigid relationship to attributes in the outer attributes. The only way to reflect changes in the reference table is to fully process the dimension.

2.1.1.10 Fast-Changing Attributes

Some data models contain attributes that change very fast. Depending on which type of history tracking you need, you may face different challenges.

Type2 Fast-Changing Attributes - If you track every change to a fast-changing attribute, this may cause the dimension containing the attribute to grow very large. Type 2 attributes are typically added to a dimension with a **ProcessAdd** command. At some point, running **ProcessAdd** on a large dimension and running all the consistency checks will take a long time. Also, having a huge dimension is unwieldy because users will have trouble querying it and the server will have trouble keeping it in memory. A good example of such a modeling challenge is the age of a customer – this will change every year and cause the customer dimension to grow dramatically.

Type 1 Fast-Changing Attributes – Even if you do not track every change to the attribute, you may still run into issues with fast-changing attributes. To reflect a change in the data source to the cube, you have to run **Process Update** on the changed dimension. As the cube and dimension grows larger, running Process Update becomes expensive. An example of such a modeling challenge is to track the status attribute of a server in a hosting environment (“Running”, “Shut down”, “Overloaded” and so on). A status attribute like this may change several times per day or even per hour. Running frequent **ProcessUpdates** on such a dimension to reflect changes can be an expensive operation, and it may not be feasible with the locking implementation of Analysis Servicesin a production environment.

In the following sections, we will look at some modeling options you can use to address these problems.

2.1.1.10.1 Type 2 Fast-Changing Attributes

If history tracking is a requirement of a fast-changing attribute, the best option is often to use the fact table to track history. This is best illustrated with an example. Consider again the customer dimension with the age attribute. Modeling the **Age** attribute directly in the customer dimension produces a design like this.

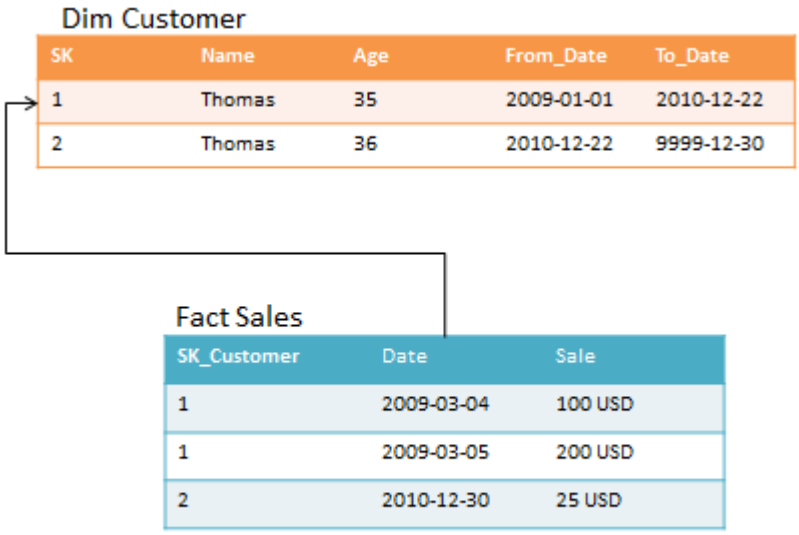


Figure 44: Age in customer dimension

Notice that every time Thomas has a birthday, a new row is added in the dimension table. The alternative design approach splits the customer dimension into two dimensions like this.

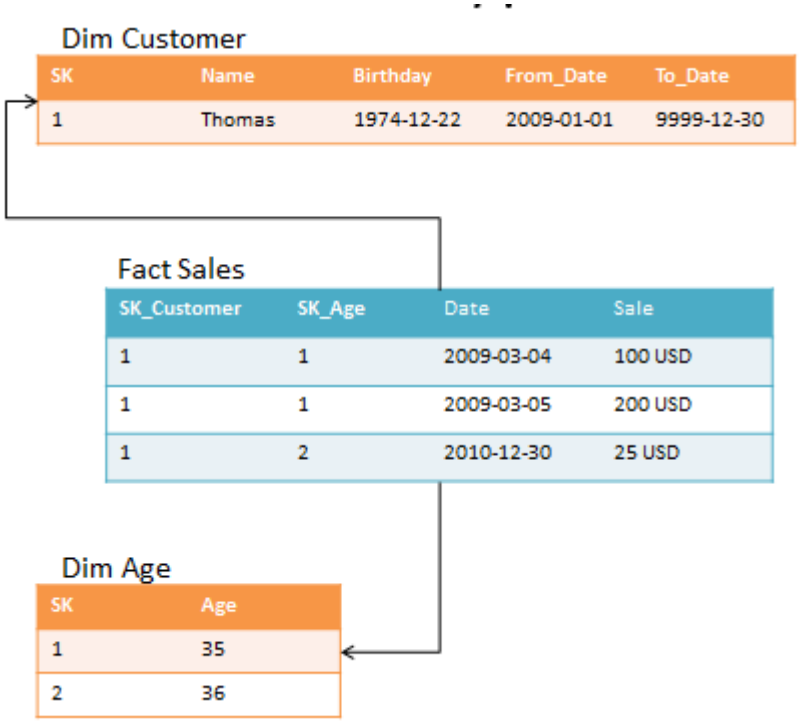


Figure 55: Age in its own dimension

Note that there are some restrictions on the situation where this design can be applied. It works best when the changing attribute takes on a small, distinct set of values. It also adds complexity to the design; by adding more dimensions to the model, it creates more work for the ETL developers when the fact table is loaded. Also, consider the storage impact on the fact table: With the alternative design, the fact table becomes wider, and more bytes have to be stored per row.

2.1.1.10.2 Type 1 Fast-Changing Attributes

Your business requirement may be updating an attribute of a dimension at high frequency, daily, or even hourly. For a small cube, running **Process Update** will help you address this issue. But as the cube grows larger, the run time of **Process Update** can become too long for the batch window or the real-time requirements of the cube (you can read more about tuning process update in the processing section).

Consider again the server hosting example: You may want to track the status, which changes frequently, of all servers. For the example, let us say that the server dimension is used by a fact table tracking performance counters. Assume you have modeled like this.

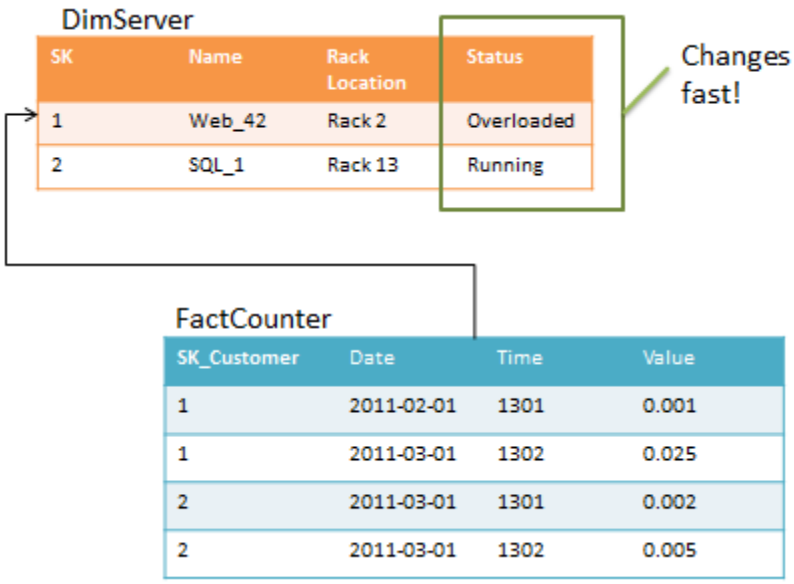


Figure 66: Status column in server dimension

The problem with this model is the **Status** column. If the **Fact Counter** is large and status changes a lot, **Process Update** will take a very long time to run. To optimize, consider this design instead.

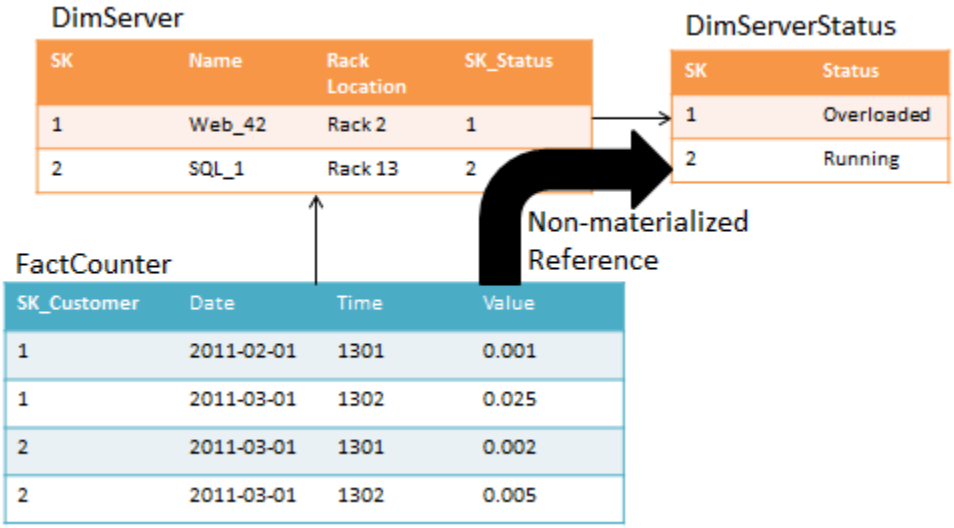


Figure 77: Status column in its own dimension

If you implement **DimServer** as the intermediate reference table to **DimServerStatus**, Analysis Services no longer has to keep track of the metadata in the **FactCounter** when you run **Process Update** on **DimServerStatus**. But as described earlier, this means that the join to **DimServerStatus** will happen at run time, increasing CPU cost and query times. It also means that you cannot index attributes in **DimServer**becausethe intermediate dimension is not materialized. You have to carefully balance the tradeoff between processing time and query speeds.

2.1.1.11 Large Dimensions

In SQL Server 2005, SQL Server 2008, and SQL Server 2008 R2, Analysis Services has some built-in limitations that limit the size of the dimensions you can create. First of all, it takes time to update a dimension – this is expensive because all indexes on fact tables have to be considered for invalidation when an attribute changes. Second, string values in dimension attributes are stored on a disk structure called the string store. This structure has a size limitation of 4 GB. If a dimension contains attributes where the total size of the string values (this includes translations) exceeds 4 GB, you will get an error during processing. The next version of SQL Server Analysis Services, code-named "Denali", is expected to remove this limitation.

Consider for a moment a dimension with tens or even hundreds of millions of members. Such a dimension can be built and added to a cube, even on SQL Server 2005, SQL Server 2008, and SQL Server 2008 R2. But what does such a dimension mean to an ad-hoc user? How will the user navigate it? Which hierarchies will group the members of this dimension into reasonable sizes that can be rendered on a screen? While it may make sense for some reporting purposes to search for individual members in such a dimension, it may not be the right problem to solve with a cube.

When you build cubes, ask yourself: is this a cube problem? For example, think of this typical telco model of call detail records.

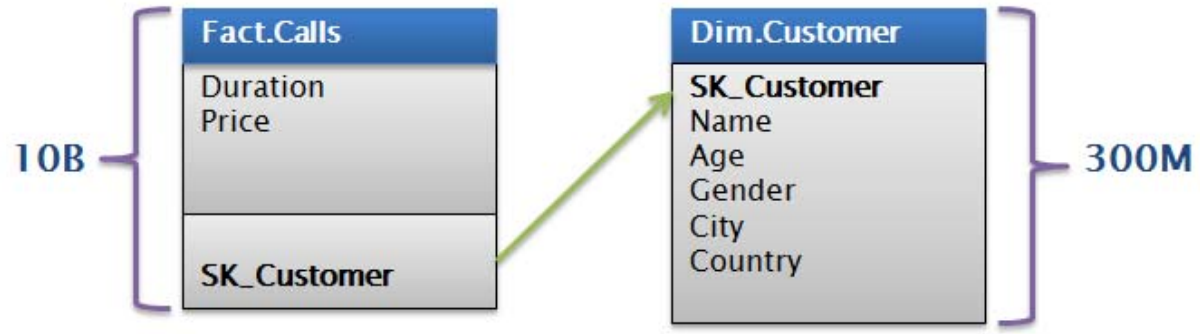


Figure 88: Call detail records (CDRs)

In this particular example, there are 300 million customers in the data model. There is no good way to group these customers and allow ad-hoc access to the cube at reasonable speeds. Even if you manage to optimize the space used to fit in the 4-GB string store, how would users browse a customer dimension like this?

If you find yourself in a situation where a dimension becomes too large and unwieldy, consider building the cube on top of an aggregate. For the telco example, imagine a transformation like the following.

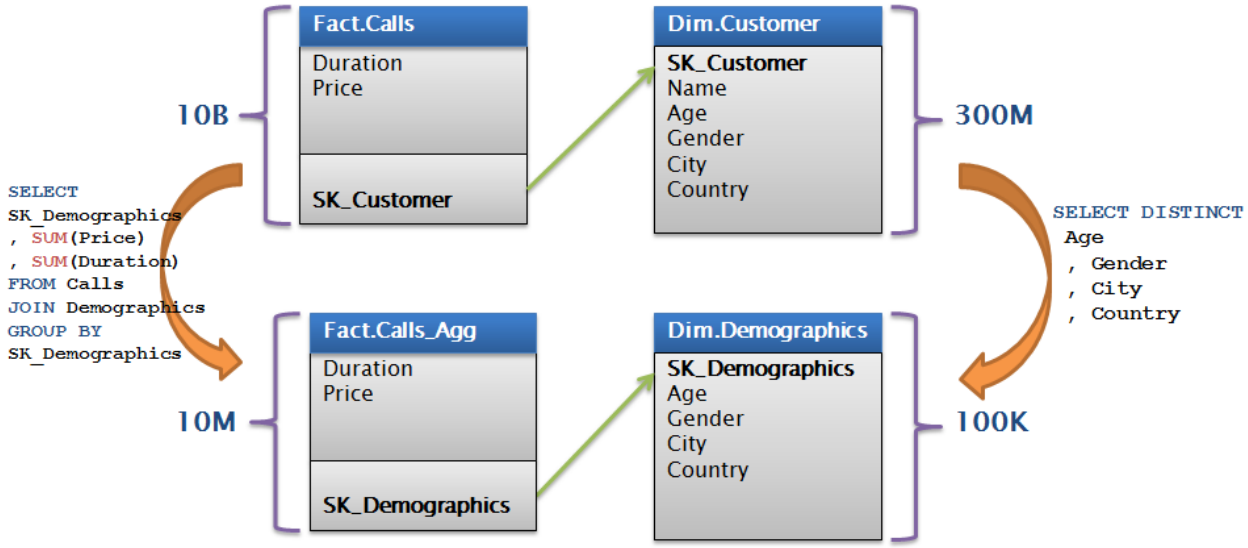


Figure 99: Cube built on aggregate

Using an aggregated fact table, this turns a 300-million-row dimension problem into 100,000-row dimension problem. You can consider aggregating the facts to save storage too – alternatively, you can add a demographics key directly to the original fact table, process on top of this data source, and rely on MOLAP compression to reduce data sizes.

2.1.2 Partitioning a Cube

Partitions separate measure group data into physical storage units. Effective use of partitions can enhance query performance, improve processing performance, and facilitate data management. This section specifically addresses how you can use partitions to improve query performance. You must often make a tradeoff between query and processing performance in your partitioning strategy.

You can use multiple partitions to break up your measure group into separate physical components. The advantages of partitioning for improving query performance are partition elimination and aggregation design.

Partition elimination - Partitions that do not contain data in the subcube are not queried at all, thus avoiding the cost of reading the index (or scanning a table if the server is in ROLAP mode). While reading a partition index and finding no available rows is a cheap operation, as the number of concurrent users grows, these reads begin to put a strain in the threadpool. Also, for queries that do not have indexes to support them, Analysis Services will have to scan all potentially matching partitions for data.

Aggregation design - Each partition can have its own or shared aggregation design. Therefore, partitions queried more often or differently can have their own designs.

```
select
[Reseller].[Business Type].members on rows,
[Measures].[Reseller Sales Amount] on columns
from [Adventure Works]
where [Date].[Calendar Year].&[2003]
```

EventClass	EventSubclass	TextData
Query Begin	0 - MDXQuery	select [Reseller].[Business Type].members on rows, [Measures].[Reseller Sales Amount]...
Progress Report Begin	14 - Query	Started reading data from the 'Reseller_Sales_2003' partition.
Progress Report End	14 - Query	Finished reading data from the 'Reseller_Sales_2003' partition.
Query Subcube	2 - Non-cache data	0000000000000000000000000,00000000,000,0000100000000000,00000,0000000000100000,000...
Query Subcube	1 - Cache data	0000000000000000000000000,00000000,000,0000000000000000,00000,0000000000100000,000...
Query Subcube	1 - Cache data	0000000000000000000000000,00000000,000,0000000000000000,00000,0000000000100000,000...
Query Subcube	1 - Cache data	0000000000000000000000000,00000000,000,0000100000000000,00000,0000000000100000,000...
Query End	0 - MDXQuery	select [Reseller].[Business Type].members on rows, [Measures].[Reseller Sales Amount]...

Figure 1010: Intelligent querying by partitions

Figure 10 displays the profiler trace of query requesting Reseller Sales Amount by Business Type from Adventure Works. The Reseller Sales measure group of the Adventure Works cube contains four partitions: one for each year. Because the query slices on 2003, the storage engine can go directly to the 2003 Reseller Sales partition and ignore other partitions.

2.1.2.1 Partition Slicing

Partitions are bound to a source table, view, or source query. When the formula engine requests a subcube, the storage engine looks at the metadata of partition for the relevant measure group. Each partition may contain a slice definition, a high level description of the minimum and maximum attribute DataIDs that exist in that dimension. If it can be determined from the slice definition that the requested subcube data is not present in the partition, that partition is ignored. If the slice definition is missing or if the information in the slice indicates that required data is present, the partition is accessed by first looking at the indexes (if any) and then scanning the partition segments.

The slice of a partition can be set in two ways:

- **Auto slice** – when Analysis Services reads the data during processing, it keeps track of the minimum and maximum attribute DataID reads. These values are used to set the slice when the indexes are built on the partition.
- **Manual slicer** – There are cases where auto slice will not work – these are described in the next section. For those situations, you can manually set the slice. Manual slices are the only available slice option for ROLAP partitions and proactive caching partitions.

2.1.2.1.1 Auto Slice

During processing of MOLAP partitions, Analysis Services internally identifies the range of data that is contained in each partition by using the Min and Max DataIDs of each attribute to calculate the range of data that is contained in the partition. The data range for each attribute is then combined to create the slice definition for the partition.

The Min and Max DataIDs can specify a either a single member or a range of members. For example, partitioning by year results in the same Min and Max DataID slice for the year attribute, and queries to a specific moment in time only result in partition queries to that year's partition.

It is important to remember that the partition slice is maintained as a range of DataIDs that you have no explicit control over. DataIDs are assigned during dimension processing as new members are encountered. Because Analysis Services just looks at the minimum and maximum value of the DataID, you can end up reading partitions that don't contain relevant data.

For example: if you have a partition, **P2003_4**, that contains both 2003 and 2004 data, you are not guaranteed that the minimum and maximum DataID in the slide contain values next to each other (even though the years are adjacent). In our example, let us say the DataID for 2003 is 42 and the DataID for 2004 is 45. Because you cannot control which DataID gets assigned to which members, you could be in a situation where the DataID for 2005 is 44. When a user requests data for 2005, Analysis Services looks at the slice for **P2003_4**, sees that it contains data in the interval 42 to 45 and therefore concludes that this partition has to be scanned to make sure it does not contain the values for DataID 44 (because 44 is between 42 and 45).

Because of this behavior, auto slice typically works best if the data contained in the partition maps to a single attribute value. When that is the case, the maximum and minimum DataID contained in the slice will be equal and the slice will work efficiently.

Note that the auto slice is not defined and indexes are not built for partitions with fewer rows than **IndexBuildThreshold** (which has a default value of 4096).

2.1.2.1.2 Manually Setting Slices

No metadata is available to Analysis Services about the content of ROLAP and proactive caching partitions. Because of this, you must manually identify the slice in the properties of the partition. It is a best practice to manually set slices in ROLAP and proactive caching partitions.

However, as shown in the previous section, there are cases where auto slice will not give you the desired partition elimination behavior. In these cases you can benefit from defining the slice yourself for MOLAP partitions. For example, if you partition by year with some partitions containing a range of years, defining the slice explicitly avoids the problem of overlapping DataIDs. This can only be done with knowledge of the data – which is where you can add some optimization as a BI developer.

It is generally not a best practice to create partitions before you are ready to fill them with data. But for real-time cubes, it is sometimes a good idea to create partitions in advance to avoid locking issues. When you take this approach, it is also a good idea to set a manual slice on MOLAP partitions to make sure the storage engine does not spend time scanning empty partitions.

2.1.2.2 Partition Sizing

For nondistinct count measure groups, tests with partition sizes in the range of 200 MB to up to 3 GB indicate that partition size alone does not have a substantial impact on query speeds. In fact, we have successfully deployed good query performance on partitions larger than 3 GB.

The following graph shows four different query runs with different partition sizes (the vertical axis is total run time in hours). Performance is comparable between partition sizes and is only affected by the design of the security features in this particular customer cube.

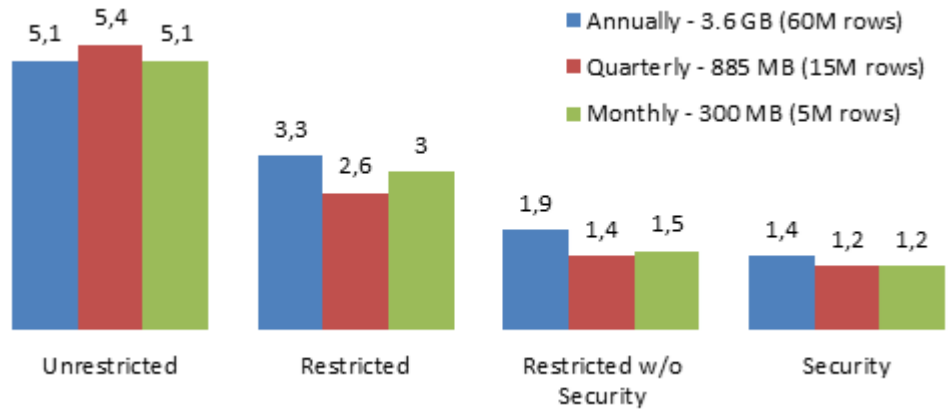


Figure 1111: Throughput by partition size (higher is better)

The partitioning strategy should be based on these factors:

- Increasing processing speed and flexibility
- Increasing manageability of bringing in new data
- Increasing query performance from partition elimination as described earlier
- Support for different aggregation designs

As you add more partitions, the metadata overhead of managing the cube grows exponentially. This affects **ProcessUpdate** and **ProcessAdd** operations on dimensions, which have to traverse the metadata dependencies to update the cube when dimensions change. As a rule of thumb, you should therefore seek to keep the number of partitions in the cube in the low thousands – while at the same time balancing the requirements discussed here.

For large cubes, prefer larger partitions over creating too many partitions. This also means that you can safely ignore the Analysis Management Objects (AMO) warning in Microsoft Visual Studio that partition sizes should not exceed 20 million rows.

2.1.2.3 Partition Strategy

From guidance on partition sizing, we can develop some common design patterns for partition strategies.

2.1.2.3.1 Partition by Date

Most cubes are built on at least one column containing a date. Because data often arrives in monthly, weekly, daily, or even hourly slices, it makes sense to partition the cube on date. Partitioning on date allows you to replace a full day in case you load faulty data. It allows you to selectively archive old data by moving the partition to cheap storage. And finally, it allows you to easily get rid of data, by removing an entire partition. Typically, a date partitioning scheme looks somewhat like this.

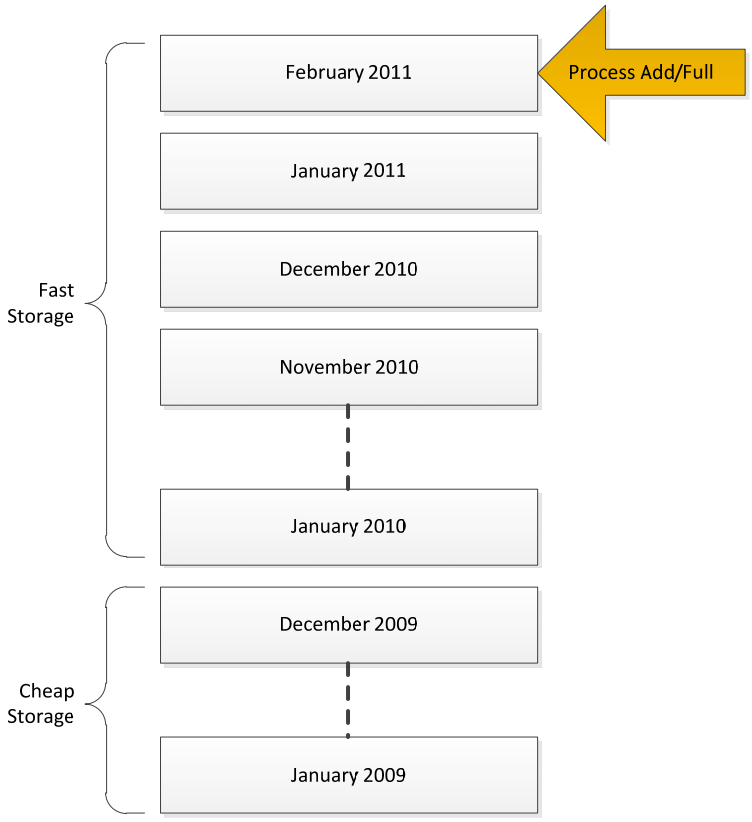


Figure 1212: Partitioning by Date

Note that in order to move the partition to cheaper storage, you will have to change the data location and reprocesses the partition. This design works very well for small to medium-sized cubes. It is reasonably simple to implement and the number of partitions is kept low. However, it does suffer from a few drawbacks:

1. If the granularity of the partitioning is small enough (for example, hourly), the number of partitions can quickly become unmanageable.
2. Assuming data is added only to the latest partition, partition processing is limited to one TCP/IP connection reading from the data source. If you have a lot of data, this can be a scalability limit.

Ad 1) If you have a lot of date-based partitions, it is often a good idea to merge the older ones into large partitions. You can do this either by using the Analysis Services merge functionality or by dropping the old partitions, creating a new, larger partition, and then reprocessing it. Reprocessing will typically take

longer than merging, but we have found that compression of the partition can often increase if you reprocess. A modified, date partitioning scheme may look like this.

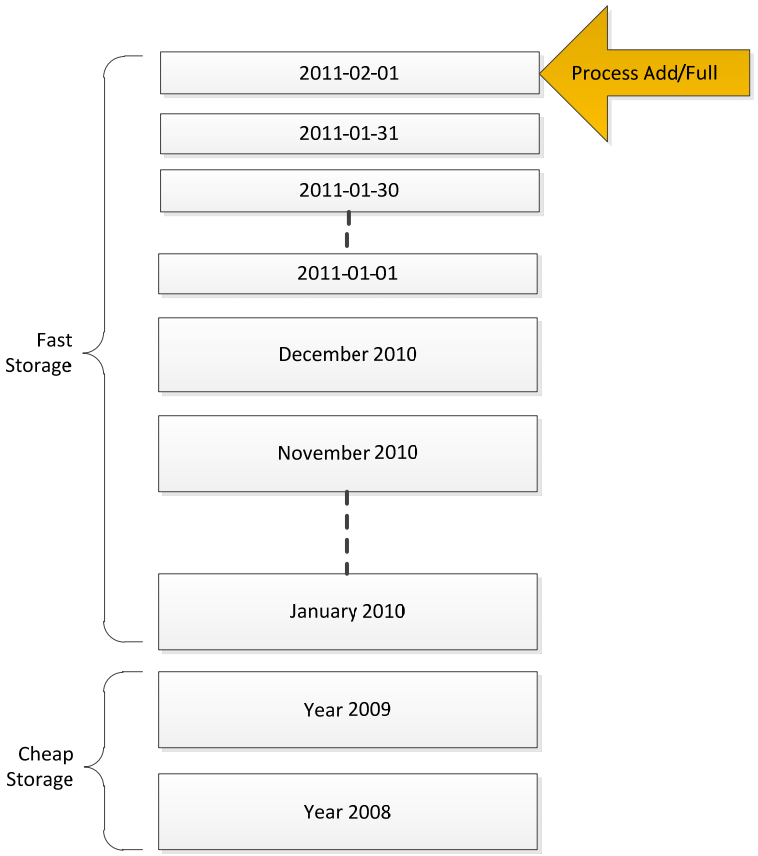


Figure 1313: Modified Date Partitioning

This design addresses the metadata overhead of having too many partitions. But it is still bottlenecked by the maximum speed of the **Process Add** or **Process Full** for the latest partition. If your data source is SQL Server, the speed of a single database connection can be hundreds of thousands of rows every second – which works well for most scenarios. But if the cube requires even faster processing speeds, consider matrix partitioning.

2.1.2.3.2 Matrix Partitioning

For large cubes, it is often a good idea to implement a matrix partitioning scheme: partition on both date **and** some other key. The date partitioning is used to selectively delete or merge old partitions as described earlier. The other key can be used to achieve parallelism during partition processing and to restrict certain users to a subset of the partitions. For example, consider a retailer that operates in US, Europe, and Asia. You might decide to partition like this.

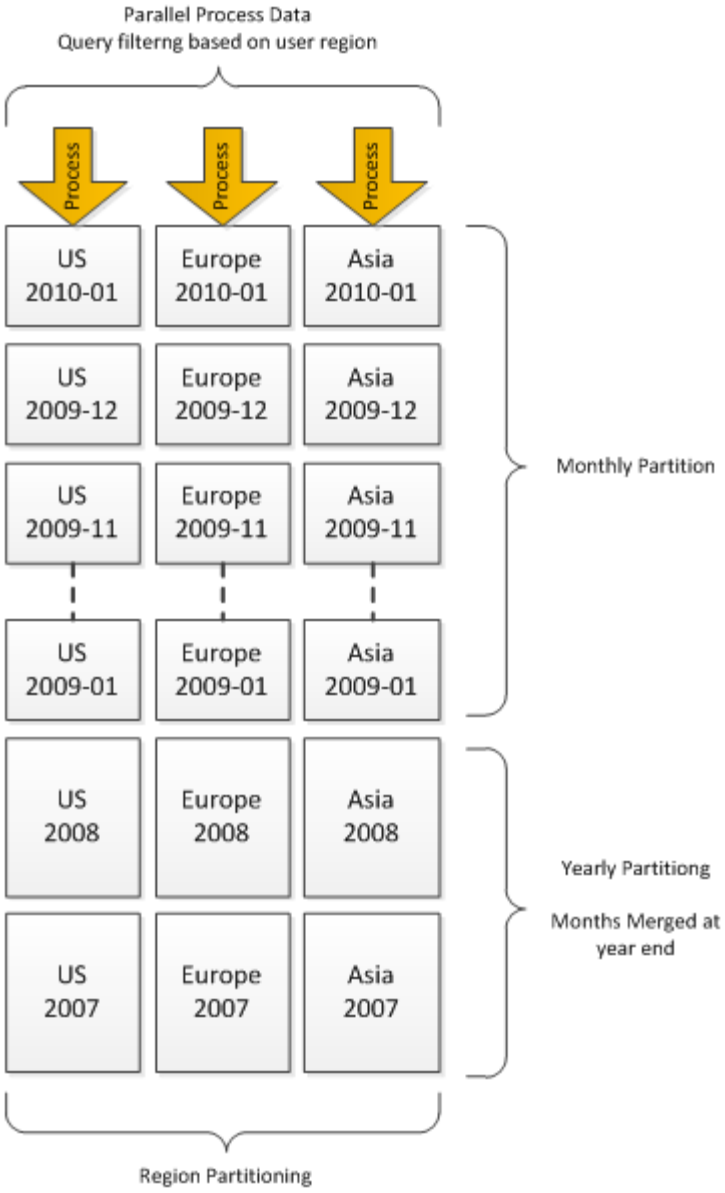


Figure 1414: Example of matrix partitioning

If the retailer grows, they may choose to split the region partitions into smaller partitions to increase parallelism of load further and to limit the worst-case scans that a user can perform. For cubes that are expected to grow dramatically, it is a good idea to choose a partition key that grows with the business and gives you options for extending the matrix partitioning strategy appropriately. The following table contains examples of such partitioning keys.

Industry	Example partition key	Source of data proliferation
Web retail	Customer key	Adding customers and transactions
Store retail	Store key	Adding new stores
Data hosting	Host ID or rack location	Adding a new server

Telecommunications	Switch ID, country code, or area code	Expanding into new geographical regions or adding new services
Computerized manufacturing	Production line ID or machine ID	Adding production lines or (for machines) sensors
Investment banking	Stock exchange or financial instrument	Adding new financial instruments, products, or markets
Retail banking	Credit card number or customer key	Increasing customer transactions
Online gaming	Game key or player key	Adding new games or players

If you implement a matrix partitioning scheme, you should pay special attention to user queries. Queries touching several partitions for every subcube request, such as a query that asks for a high-level aggregate of the partition business key, result in a high thread usage in the storage engine. Because of this, we recommend that you partition the business key so that single queries touch no more than the number of cores available on the target server. For example, if you partition by Store Key and you have 1,000 stores, queries touching the aggregation of all stores will have to touch 1,000 partitions. In such a design, it is a good idea to group the stores into a number of buckets (that is, group the stores on each partition, rather than having individual partitions for each store). For example, if you run on a 16-core server, you can group the store into buckets of around 62 stores for each partition (1,000 stores divided into 16 buckets).

2.1.2.3.3 Hash Partitioning

Sometimes it is not possible to come up with a good distribution of business keys for partitioning the cube. Perhaps you just don't have a good key candidate that fits the description in the previous section, or perhaps the distribution of the key is unknown at design time. In such cases, a brute-force approach can be used: Partition on the hash value of a key that has a high enough cardinality and where there is little skew. If you expect every query to touch many partitions, it is important that you pay special attention to the **CoordinatorQueryBalancingFactor** and the **CoordinatorQueryMaxThread** settings, which are described in Part 2.

2.1.3 Relational Data Source Design

Cubes are typically built on top of relational data sources to serve as data marts. Through the design surface, Analysis Services allows you to create powerful abstractions on top of the relational source. Computed columns and named queries are examples of this. This allows fast prototyping and also enabled you to correct poor relational design when you are not in control of the underlying data source. But the Analysis Services design surface is no panacea – a well-designed relational data source can make queries and processing of a cube faster. In this section, we explore some of the options that you should consider when designing a relational data source. A full treatment of relational data warehousing is out of scope for this document, but we will provide references where appropriate.

2.1.3.1 Use a Star Schema for Best Performance

It is widely debated what the most efficient ad-report modeling technique is: star schema, snowflake schema, or even a third to fifth normal form or data vault models (in order of the increased normalization). All are considered by warehouse designers as candidates for reporting.

Note that the Analysis Services Unified Dimensional Model (UDM) is a dimensional model, with some additional features (reference dimensions) that support snowflakes and many-to-many dimensions. No matter which model you choose as the end-user reporting model, performance of the relational model boils down to one simple fact: joins are expensive! This is also partially true for the Analysis Services engine itself. For example: If a snowflake is implemented as a non-materialized reference dimension, users will wait longer for queries, because the join is done at run time inside the Analysis Services engine.

The largest impact of snowflakes occurs during processing of the partition data. For example: If you implement a fact table as a join of two big tables (for example, separating order lines and order headers instead of storing them as pre-joined values), processing of facts will take longer, because the relational engine has to compute the join.

It is possible to build an Analysis Services cube on top of a highly normalized model, but be prepared to pay the price of joins when accessing the relational model. In most cases, that price is paid at processing time. In MOLAP data models, materialized reference dimensions help you store the result of the joined tables on disk and give you high speed queries even on normalized data. However, if you are running ROLAP partitions, queries will pay the price of the join at query time, and your user response times or your hardware budget will suffer if you are unable to resist normalization.

2.1.3.2 Consider Moving Calculations to the Relational Engine

Sometimes calculations can be moved to the Relational Engine and be processed as simple aggregates with much better performance. There is no single solution here; but if you're encountering performance issues, consider whether the calculation can be resolved in the source database or data source view (DSV) and prepopulated, rather than evaluated at query time.

For example, instead of writing expressions like Sum(Customer.City.Members, cint(Customer.City.Currentmember.properties("Population"))), consider defining a separate measure group on the **City** table, with a sum measure on the **Population** column.

As a second example, you can compute the product of revenue * Products Sold at the leaves in the cube and aggregate with calculations. But computing this result in the source database instead can provide superior performance.

2.1.3.3 Use Views

It is generally a good idea to build your UDM on top of database views. A major advantage of views is that they provide an abstraction layer on top of the physical, relational model. If the cube is built on top of views, the relational database can, to some degree, be remodeled without breaking the cube.

Consider a relational source that has chosen to normalize two tables you need to join to obtain a fact table – for example, a data model that splits a sales fact into order lines and orders. If you implement the fact table using query binding, your UDM will contain the following.

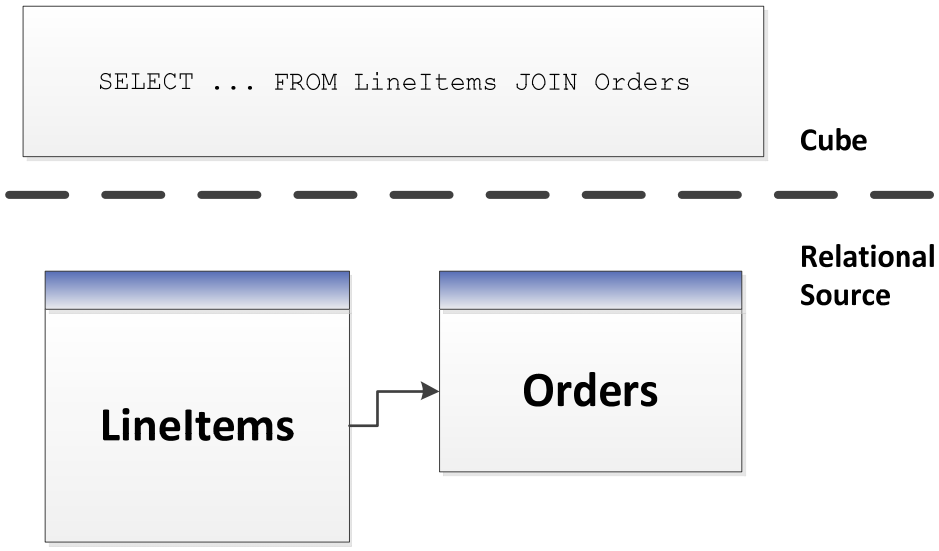


Figure 1515: Using named queries in UDM

In this model, the UDM now has a dependency on the structure of the **LineItems** and **Orders** tables – along with the join between them. If you instead implement a **Sales** view in the database, you can model like this.

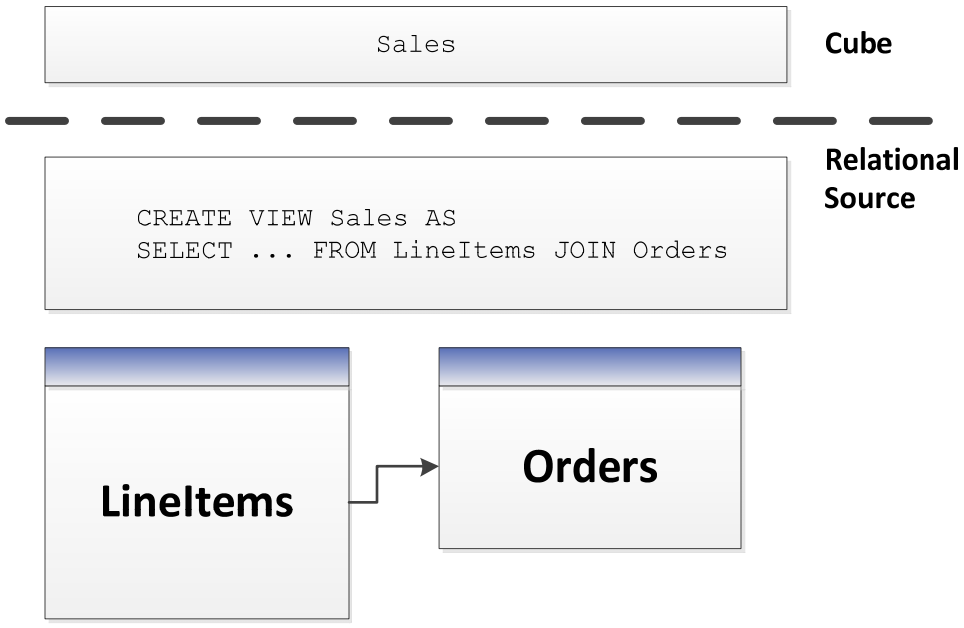


Figure 1616: Implementing UDM on top of views

This model gives the relational database the freedom to optimize the joined results of **LineItems** and **Order**(for example by storing it denormalized), without any impact on the cube. It would be transparent for the cube developer if the DBA of the relational database implemented this change.

Figure 1717: Implementing UDM on top of pre-joined tables

Views provide encapsulation, and it is good practice to use them. If the relational data modelers insist on normalization, give them a chance to change their minds and denormalize without breaking the cube model.

Views also provide easy of debugging. You can issue SQL queries directly on views to compare the relational data with the cube. Hence, views are good way to implement business logic that could you could mimic with query binding in the UDM. While the UDM syntax is similar to the SQL view syntax, you cannot issue SQL statements against the UDM.

2.1.3.3.1 Query Binding Dimensions

Query binding for dimensions does not exist in SQL Server 2008 Analysis Services, but you can implement it by using a view (instead of tables) for your underlying dimension data source. That way, you can use hints, indexed views, or other relational database tuning techniques to optimize the SQL statement that accesses the dimension tables through your view. This also allows you to turn a snowflake design in the relational source into a UDM that is a pure star schema.

2.1.3.3.2 Processing Through Views

Depending on the relational source, views can often provide means to optimize the behavior of the relational database. For example, in SQL Server you can use the NOLOCK hint in the view definition to remove the overhead of locking rows as the view is scanned, balancing this with the possibility of getting dirty reads. Views can also be used to preaggregate large fact tables using a GROUP BY statement; the relational database modeler can even choose to materialize views that use a lot of hardware resources.

2.1.4 Calculation Scripts

The calculation script in the cube allows you to express complex functionality of the cube, conferring the ability to directly manipulate the multidimensional space. In a few lines of code, you can elegantly build highly valuable business logic. But conversely, it takes only a few lines of poorly written calculation code to create a big performance impact on users. If you plan to design a cube with a large calculation script, we highly recommend that you learn the basics of writing good MDX code – the language used for calculations. The references section contains resources that will get you off to a good start.

The query tuning section in this book provides high-level guidance on tuning individual queries. But even at design time, there are some best practices you should apply to the cube that avoid common performance mistakes. This section provides you with some basic rules; these are the bare minimum you should apply when building the cube script.

References:
MDX has a rich community of contributors on the web. Here are some links to get you started:

- Pearson, Bill: "Stairway to MDX"
  - http://www.sqlservercentral.com/stairway/72404/
- Piasevoli, Tomislav: *MDX with Microsoft SQL Server 2008 R2 Analysis Services Cookbook*
  - http://www.packtpub.com/mdx-with-microsoft-sql-server-2008-r2-analysis-services/book
- Russo, Marco: MDX Blog:
  - http://sqlblog.com/blogs/marco_russo/archive/tags/MDX/default.aspx
- Pasumansky, Mosha: Blog
  - http://sqlblog.com/blogs/mosha/
- Piasevoli, Tomislav: Blog
  - http://tomislav.piasevoli.com
- Webb, Christopher: Blog
  - http://cwebbbi.wordpress.com/category/mdx/
- Spofford, George, Sivakumar Harinath, Christopher Webb, Dylan Hai Huang, and Francesco Civardi,: *MDX Solutions: With Microsoft SQL Server Analysis Services 2005 and Hyperion Essbase*, ISBN: 978-0471748083

2.1.4.1 Use Attributes Instead of Sets

When you need to refer to a fixed subset of dimension members in a calculation, use an attribute instead of a set. Attributes enable you to target aggregations to the subset. Attributes are also evaluated faster than sets by the formula engine. Using an attribute for this purpose also allows you to change the set by updating the dimension instead of deploying a new calculation scripts.

Example: Instead of this:

```
CREATE SET[Current Day] AS TAIL([Date].[Calendar].members, 1)

CREATE SET [Previous Day] AS HEAD(TAIL(Date].[Calendar].members),2),1)
```

Do this (assuming today is 2011-06-16):

Calendar Key Attribute	Day Type Attribute (Flexible relationship to key)
2011-06-13	Old Dates
2011-06-14	Old Dates
2011-06-15	Previous Day
2011-06-16	Current Day

Process Update the dimension when the day changes. Users can now refer to the current day by addressing the **Day Type** attribute instead of the set.

2.1.4.2 Use SCOPE Instead of IIF When Addressing Cube Space

Sometimes, you want a calculation to only apply for a specific subset of cube space. SCOPE is a better choice than IIF in this case. Here is an example of what *not* to do.

```
CREATE MEMBER CurrentCube.[Measures].[SixMonthRollingAverage] AS
IIF([Date].[Calendar].CurrentMember.Level
Is [Date].[Calendar].[Month]

,Sum ([Date].[Calendar].CurrentMember.Lag(5)
:[Date].[Calendar].CurrentMember
,[Measures].[Internet Sales Amount])/ 6

,NULL)
```

Instead, use the Analysis Services SCOPE function for this.

```
CREATE MEMBER CurrentCube.[Measures].[SixMonthRollingAverage]
AS NULL ,FORMAT_STRING = "Currency", VISIBLE = 1;

SCOPE([Measures].[SixMonthRollingAverage],[Date].[Calendar].[Month].Members);

      THIS = Sum([Date].[Calendar].CurrentMember.Lag(5)
:[Date].[Calendar].CurrentMember
,[Measures].[Internet Sales Amount])/ 6;
```

```
    END SCOPE;
```

2.1.4.3 Avoid Mimicking Engine Features with Expressions

Several native features can be mimicked with MDX:

- Unary operators
- Calculated columns in the data source view (DSV)
- Measure expressions
- Semiadditive measures

You can reproduce each these features in MDX script (in fact, sometimes you must, because some are only supported in the Enterprise SKU), but doing so often hurts performance.

For example, using distributive unary operators (that is, those whose member order does not matter, such as +, -, and ~) is generally twice as fast as trying to mimic their capabilities with assignments.

There are rare exceptions. For example, you might be able to improve performance of nondistributive unary operators (those involving *, /, or numeric values) with MDX. Furthermore, you may know some special characteristic of your data that allows you to take a shortcut that improves performance. Such optimizations require expert-level tuning – and in general, you can rely on the Analysis Services engine features to do the best job.

Measure expressions also provide a unique challenge, because they disable the use of aggregates (data has to be rolled up from the leaf level). One way to work around this is to use a hidden measure that contains preaggregated values in the relational source. You can then target the hidden measure to the aggregate values with a SCOPE statement in the calculation script.

2.1.4.4 Comparing Objects and Values

When determining whether the current member or tuple is a specific object, use IS. For example, the following query is not only nonperformant, but incorrect. It forces unnecessary cell evaluation and compares values instead of members.

```
[Customer].[Customer Geography].[Country].&[Australia] = [Customer].[Customer
Geography].currentmember
```

Furthermore, don't perform extra steps when deducing whether **CurrentMember** is a particular member by involving **Intersect** and **Counting**.

```
intersect({[Customer].[Customer Geography].[Country].&[Australia]},
[Customer].[Customer Geography].currentmember).count > 0
```

Use IS instead.

```
[Customer].[Customer Geography].[Country].&[Australia] is [Customer].[Customer
Geography].currentmember
```

2.1.4.5 Evaluating Set Membership

Determining whether a member or tuple is in a set is best accomplished with **Intersect**. The **Rank** function does the additional operation of determining where in the set that object lies. If you don't need it, don't use it. For example, the following statement may do more work than you need it to do.

```
rank( [Customer].[Customer Geography].[Country].&[Australia],
<set expression> )>0
```

This statement uses **Intersect** to determine whether the specified information is in the set.

```
intersect({[Customer].[Customer Geography].[Country].&[Australia]}, <set> ).count > 0
```

2.2 Testing Analysis Services Cubes

As you prepare for user acceptance and preproduction testing of a cube, you should first consider what a cube is and what that means for user queries. Depending on your background and role in the development and deployment cycle, there are different ways to look at this.

As a database administrator, you can think of a cube as a database that can accept *any* query from users, and where the response time from any such query is expected to be "reasonable" – a term that is often vaguely defined. In many cases, you can optimize response time for specific queries using aggregates (which for relational DBAs is similar to index tuning), and testing should give you an early idea of good aggregation candidates. But even with aggregates, you must also consider the worst case: You should expect to see leaf-level scan queries. Such queries , which can be easily expressed by tools like Excel, may end up touching every piece of data the cube. Depending on your design, this can be a

significant amount of data. You should consider what you want to do with such queries. There are multiple options: For example, you may choose to scale your hardware to handle them in a decent response time, or you may simply choose to cancel them. In either case, as you prepare for testing, make sure such queries are part the test suite and that you observe what happens to the Analysis Services instance when they run. You should also understand what a “reasonable” response time is for your end user and make that part of the test suite.

As a BI developer, you can look at the cube as your description of the multidimensional space in which queries can be expressed. A part of this space will be instantiated with data structures on disk supporting it: dimensions, measure groups, and their partitions. However, some of this multidimensional space will be served by calculations or ad-hoc memory structures, for example: MDX calculations, many-to-many dimensions, and custom rollups. Whenever queries are not served directly by instantiated data, there is a potential, query-time calculation price to be paid, this may show up as bad response time. As the cube developer, you should make sure that the testing covers these cases. Hence, as the BI-developer, you should make sure the test queries also stress noninstantiated data. This is a valuable exercise, because you can use it to measure the impact on users of complex calculations and then adjust the data model accordingly.

Your approach to testing will depend on which situation you find yourself in. If you are developing a new system, you can work directly towards the test goals driven by business requirements. However, if you are trying to improve an existing system, it is an advantage to acquire a test baseline first.

2.2.1 Testing Goals

Before you design a test harness, you should decide what your testing goals will be. Testing not only allows you to find functional bugs in the system, it also helps quantify the scalability and potential bottlenecks that may be hard to diagnose and fix in a busy production environment. If you do not know what your scale-barrier is or where they system might break, it becomes hard to act with confidence when you tune the final production system.

Consider what characteristics are reasonable to expect from the BI systems and document these expectations as part of your test plan. The definition of *reasonable* depends to a large extent on your familiarity with similar systems, the skills of your cube designers, and the hardware you run on. It is often a good idea to get a second opinion on what test values are reachable – for example from a neutral third party that has experience with similar systems. This helps you set expectations properly with both developers and business users.

BI systems vary in the characteristics organizations require of them – not everyone needs scalability to thousands of users, tens of terabytes, near-zero downtime, and guaranteed subsecond response time. While all these goals can be achieved for most cases, it is not always cheap to acquire the skills required to design a system to support them. Consider what your system needs to do for your organization, and avoid overdesigning in the areas where you don’t need the highest requirements. For example: you may decide that you need very fast response times, but that you also want a very low-cost server that can run in a shared storage environment. For such a scenario, you may want to reduce the data in the cube

to a size that will fit in memory, eliminating the need for the majority of I/O operations and providing fast scan times even for poorly filtered queries.

Here is a table of potential test goals you should consider. If they are relevant for your organization's requirements, you should tailor them to reflect those requirements.

Test Goal	Description	Example goal
Scalability	How many concurrent users should be supported by the system?	"Must support 10,000 concurrently connected users, of which 1,000 run queries simultaneously."
Performance/ throughput	How fast should queries return to the client? This may require you to classify queries into different complexities. Not all queries can be answered quickly and it will often be wise to consult an expert cube designer to liaise with users to understand what query patterns can be expected and what the complexity of answering these queries will be. Another way to look at this test goal is to measure the throughput in queries answered per second in a mixed workload.	"Simple queries returning a single product group for a given year should return in less than 1 second – even at full user concurrency." "Queries that touch no more than 20% of the fact rows should run in less than 30 seconds. Most other queries touching a small part of the cube should return in around 10 seconds. With our workload, we expect throughput to be around 50 queries returned per second." "User queries requesting the end-of-month currency rate conversion should return in no more than 20 seconds. Queries that do not require currency conversion should return in less than 5 seconds."
Data Sizes	What is the granularity of each dimension attribute? How much data will each measure group contain? Note that the cube designers will often have been considering this and may already know the answer.	"The largest customer dimension will contain 30 million rows and have 10 attributes and two user hierarchies. The largest non-key attribute will have 1 million members." "The largest measure group is sales, with 1 billion rows. The second largest is purchases, with 100 million rows. All other measure groups are trivial in size."

Target Server Platforms	Which server model do you want to run on? It is often a good idea to test on both that server and an even bigger server class. This enables you to quantify the benefits of upgrading.	"Must run on 2-socket 6-core Nehalem machine with 32 GB of RAM." "Must be able to scale to 4-socket Nehalem 8-core machine with 256 GB of RAM.
Target I/O system	Which I/O system do you want to use? What characteristics will that system have?	"Must run on corporate SAN and use no more than 1,000 random IOPS at 32,000 block sizes at 6ms latency." "Will run on dedicated NAND devices that support 80,000 IOPS at 100 μs latency."
Target network infrastructure	Which network connectivity will be available between users and Analysis Services, and between Analysis Services and the data sources? Note that you may have to simulate these network conditions in a lab.	"In the worst case scenario, users will connect over a 100ms latency WAN link with a maximum bandwidth of 10Mbit/sec." "There will be a 10Gbit dedicated network available between the data source and the cube."
Processing Speeds	How fast should rows be brought into the cube and how often?	"Dimensions should be fully processed every night within 30 minutes." "Two times during the day, 100,000,000 rows should be added to the sales measure group. This should take no longer than 15 minutes."

2.2.2 Test Scenarios

Based on the considerations from the previous section you should be able to create a user workload that represents typical user behavior and that enables you to measure whether you are meeting your testing goals.

Typical user behavior and well-written queries are unfortunately not the only queries you will receive in most systems. As part of the test phase, you should also try to flush out potential production issues before they arise. We recommend that you make sure your test workload contains the following types of queries and tests them thoroughly :

- ✓ Queries that touch several dimensions at the same time
- ✓ Enough queries to test every MDX expression in the calculation script
- ✓ Queries that exercise many-to-many dimensions
- ✓ Queries that exercise custom rollups
- ✓ Queries that exercise parent/child dimensions
- ✓ Queries that exercise distinct count measure groups
- ✓ Queries that crossjoin attributes from dimensions that contain more than 100,000 members
- ✓ Queries that touch every single partition in the database
- ✓ Queries that touch a large subset of partitions in the database (for example, current year)
- ✓ Queries that return a lot of data to the client (for example, more than 100,000 rows)
- ✓ Queries that use cube security versus queries that do not use it
- ✓ Queries executing concurrently with processing operations – if this is part of your design

You should test on the full dataset for the production cube. If you don't, the results will not be representative of real-life operations. This is especially true for cubes that are larger than the memory on the machine they will eventually run on.

Of course, you should still make sure that you have plenty of queries in the test scenarios that represent typical user behaviors – running on a workload that only showcases the slowest-performing parts of the cube will not represent a real production environment (unless of course, the entire cube is poorly designed).

As you run the tests, you will discover that certain queries are more disruptive than others. One goal of testing is to discover what such queries look like, so that you can either scale the system to deal with them or provide guidance for users so that they can avoid exercising the cube in this way if possible.

Part of your test scenarios should also aim to observe the cubes behavior as user concurrency grows. You should work with BI developers and business users to understand what the worst-case scenario for user concurrency is. Testing at that concurrency will shake out poorly scalable designs and help you configure the cube and hardware for best performance and stability.

2.2.3 Load Generation

It is hard to create a load that actually looks like the expected production load – it requires significant experience and communication with end users to come up with a fully representative set of queries. But after you have a set of queries that match user behavior, you can feed them into a test harness. Analysis Services does not ship with a test harness out of the box, but there are several solutions available that help you get started:

ascmd – You can use this command-line tool to run a set of queries against Analysis Services. It ships with the Analysis Services samples and is maintained on CodePlex.

Visual Studio – You can configure Microsoft Visual Studio to generate load against Analysis Services, and you can also use Visual Studio to visually analyze that load.

Third-party tools – You can use tools such as HP LoadRunner to generate high-concurrency load. Note that Analysis Services also supports an HTTP-based interface, which means it may be possible to use web stress tools to generate load.

Roll your own: We have seen customers write their own test harnesses using .NET and the ADOMD.NET interface to Analysis Services. Using the .NET threading libraries, it is possible to generate a lot of user load from a single load client.

No matter which load tool you use, you should make sure you collect the runtime of all queries and the Performance Monitor counters for all runs. This data enables you to measure the effect of any changes you make during your test runs. When you generate user workload there are also some other factors to consider.

First of all, you should test both a sequential run and a parallel run of queries. The sequential run gives you the best possible run time of the query while no other users are on the system. The parallel run enables you to shake out issues with the cube that are the result of many users running concurrently.

Second, you should make sure the test scenarios contain a sufficient number of queries so that you will be able to run the test scenario for some time. To stress the server properly, and to avoid querying the same hotspot values over and over again, queries should touch a variety of data in the cube. If all your queries touch a small set of dimension values, it will hardly represent a real production run. One way to spread queries over a wider set of cells in the cube is to use query templates. Each template can be used to generate a set of queries that are all variants of the same general user behavior.

Third, your test harness should be able to create reproducible tests. If you are using code that generates many queries from a small set of templates, – make sure that it generates the same queries on every test run. If not, you introduce an element of randomness in the test that makes it hard to compare different runs.

References:

- Ascmd.exe on MSDN - http://msdn.microsoft.com/en-us/library/ms365187%28v=sql.100%29.aspx
- Analysis Services Community samples - http://sqlsrvanalysissrvcs.codeplex.com/
  - Describes how to use **ascmd** for load generation
  - Contains Visual Studio sample code that achieves a similar effect
- HP LoadRunner - https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-126-17^8_4000_100

2.2.4 Clearing Caches

To make test runs reproducible, it is important that each run start with the server in the same state as the previous run. To do this, you must clear out any caches created by earlier runs.

There are three caches in Analysis Services that you should be aware of:

- The formula engine cache
- The storage engine cache
- The file system cache

Clearing formula engine and storage engine caches: The first two caches can be cleared with the XMLA **ClearCache** command. This command can be executed using the **ascmd** command-line utility:

```
<ClearCache
       xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">
<Object>
<DatabaseID><database name></DatabaseID>
</Object>
</ClearCache>
```

Clearing file system caches: The file system cache is a bit harder to get rid of because it resides inside Windows itself.

If you have created a separate Windows volume for the cube database, you can dismount the volume itself using the following command:

fsutil.exe volume dismount < Drive Letter | Mount Point >

This clears the file system cache for this drive letter or mount point. If the cube database resides only on this location, running this command results in a clean file system cache.

Alternatively, you can use the utility **RAMMap** from sysinternals. This utility not only allows you to read the file system cache content, it also allows you to purge it. On the **empty** menu, click **Empty System Working Set**, and then click **Empty Standby List**. This clears the file system cache for the entire system. Note that when **RAMMap** starts up, it temporarily freezes the system while it reads the memory content – this can take some time on a large machine. Hence, **RAMMap** should be used with care.

There is currently a CodePlex project called ASStoredProcedures found at: http://asstoredprocedures.codeplex.com/wikipage?title=FileSystemCache. This project contains code for a utility that enables you to clear the file system cache using a stored procedure that you can run directly on Analysis Services.

Note that neither **FSUTIL** nor **RAMMap** should be used in production cubes –both cause disruption to users connected to the cube. Also note that neither **RAMMap** or ASStoredProcedures is supported by Microsoft.

2.2.5 Actions During Testing for Operations Management

While your test team is creating and running the test harness, your operations team can also take steps to prepare for deployment.

Test your data collection setup: Test runs give you a unique chance to try out your data collection procedures before you go into production. The data collection you perform can also be used to drive early feedback to the development team.

Understand server utilization: While you test, you can get an early insight into server utilization. You will be able to measure the memory usage of the cube and the way the number of users maps to I/O load and CPU utilization. If the cube is larger than memory, you can also measure the effect of concurrency and leaf level scan on the I/O subsystem. Remember to measure the worst-case examples described earlier to understand what the impact on the system is.

Early thread tuning: During testing, you can discover threading bottlenecks, as described in Part 1. This enables you to go into production with pretuned settings that improve user experience, scalability, and hardware utilization of the solution.

2.2.6 Actions After Testing

When testing is complete, you have reports that describe the run time of each query, and you also have a greater understanding of the server utilization. This is a good time to review the design of the cube with the BI developers using the numbers you collected during testing. Often, some easy wins can be harvested at this point.

When you put a cube into production, it is important to understand the long-term effects of users building spreadsheets and reports referencing it. Consider the dependencies that are generated as the cube is successfully deployed in ad-hoc data structures across the organization. The data model created and exposed by the cube will be linked into spreadsheets and reports, and it becomes hard to make data model changes without disturbing users. From an operational perspective, preproduction testing is typically your last chance to request cheap data model changes from your BI developers before business users inevitably lock themselves into the data structures that unlock their data. Notice that this is different from typical, static reporting and development cycles. With static reports, the BI developers are in control of the dependencies ;if you give users Excel or other ad-hoc access to cubes, that control is lost. Explorative data power comes at a price.

2.3 Tuning Query Performance

To improve query performance, you should understand the current situation, diagnose the bottleneck, and then apply one of several techniques including optimizing dimension design, designing and building aggregations, partitioning, and applying best practices. These should be the first stops for optimization, before digging into queries in general.

Much time can be expended pursuing dead ends – it is important to first understand the nature of the problem before applying specific techniques. To gain this understanding, it is often useful to have a mental model of how the query engine works. We will therefore start with a brief introduction to the Analysis Services query processor.

2.3.1 Query Processor Architecture

To make the querying experience as fast as possible for end users, the Analysis Services querying architecture provides several components that work together to efficiently retrieve and evaluate data. The following figure identifies the three major operations that occur during querying—session management, MDX query execution, and data retrieval—as well as the server components that participate in each operation.

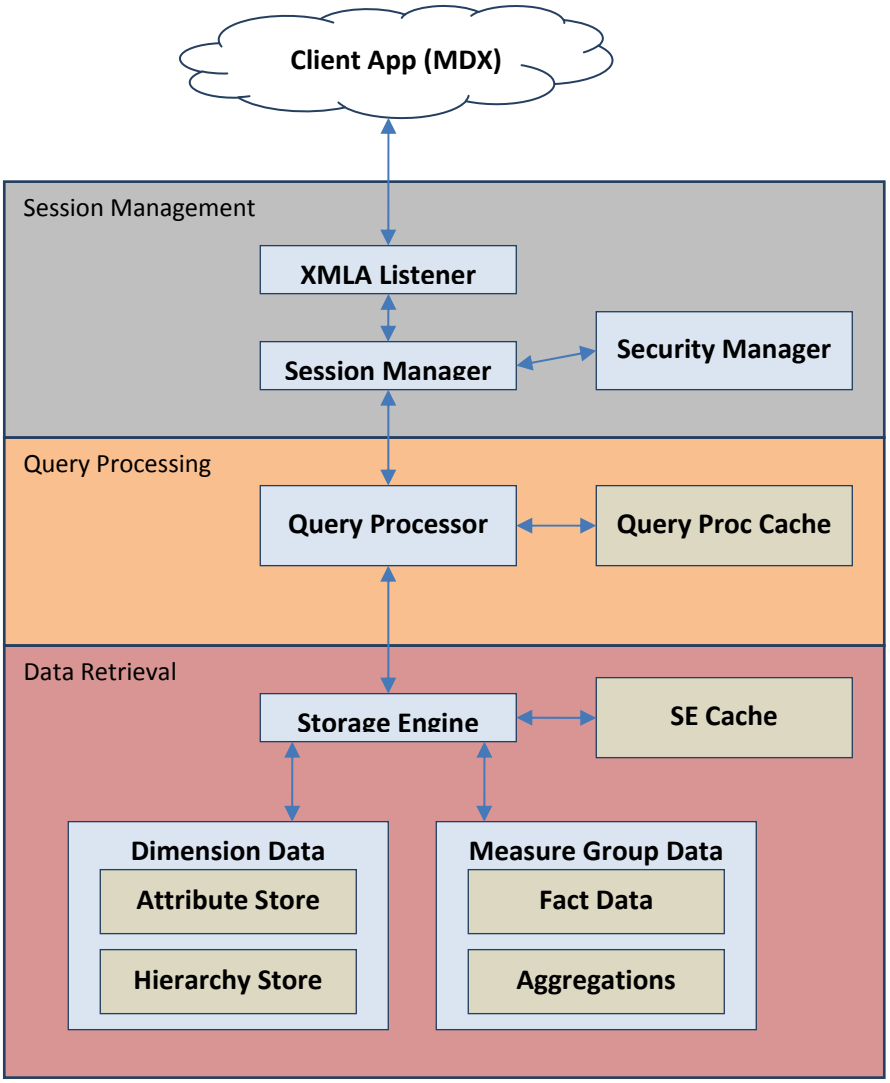


Figure 1818: Analysis Services query processor architecture

2.3.1.1 Session Management

Client applications communicate with Analysis Services using XML for Analysis (XMLA) over TCP/IP or HTTP. Analysis Services provides an XMLA listener component that handles all XMLA communications between Analysis Services and its clients. The Analysis Services Session Manager controls how clients connect to an Analysis Services instance. Users authenticated by the Windows operating system and who have access to at least one database can connect to Analysis Services. After a user connects to

Analysis Services, the Security Manager determines user permissions based on the combination of Analysis Services roles that apply to the user. Depending on the client application architecture and the security privileges of the connection, the client creates a session when the application starts, and then it reuses the session for all of the user's requests. The session provides the context under which client queries are executed by the query processor. A session exists until it is closed by the client application or the server.

2.3.1.2 Query Processing

The query processor executes MDX queries and generates a cellset or rowset in return. This section provides an overview of how the query processor executes queries. For more information about optimizing MDX, see Optimizing MDX.

To retrieve the data requested by a query, the query processor builds an execution plan to generate the requested results from the cube data and calculations. There are two major different types of query execution plans: cell-by-cell (naïve) evaluation or block mode (subspace) computation. Which one is chosen by the engine can have a significant impact on performance. For more information, see Subspace Computation.

To communicate with the storage engine, the query processor uses the execution plan to translate the data request into one or more subcube requests that the storage engine can understand. A subcube is a logical unit of querying, caching, and data retrieval—it is a subset of cube data defined by the crossjoin of one or more members from a single level of each attribute hierarchy. An MDX query can be resolved into multiple subcube requests, depending the attribute granularities involved and calculation complexity; for example, a query involving every member of the Country attribute hierarchy (assuming it's not a parent-child hierarchy) would be split into two subcube requests: one for the All member and another for the countries.

As the query processor evaluates cells, it uses the query processor cache to store calculation results. The primary benefits of the cache are to optimize the evaluation of calculations and to support the reuse of calculation results across users (with the same security roles). To optimize cache reuse, the query processor manages three cache layers that determine the level of cache reusability: global, session, and query.

2.3.1.2.1 Query Processor Cache

During the execution of an MDX query, the query processor stores calculation results in the query processor cache. The primary benefits of the cache are to optimize the evaluation of calculations and to support reuse of calculation results across users. To understand how the query processor uses caching during query execution, consider the following example: You have a calculated member called Profit Margin. When an MDX query requests Profit Margin by Sales Territory, the query processor caches the nonnull Profit Margin values for each Sales Territory. To manage the reuse of the cached results across users, the query processor distinguishes different contexts in the cache:

- **Query Context**—contains the result of calculations created by using the WITH keyword within a query. The query context is created on demand and terminates when the query is over. Therefore, the cache of the query context is not shared across queries in a session.
- **Session Context** —contains the result of calculations created by using the CREATE statement within a given session. The cache of the session context is reused from request to request in the same session, but it is not shared across sessions.
- **Global Context** —contains the result of calculations that are shared among users. The cache of the global context can be shared across sessions if the sessions share the same security roles.

The contexts are tiered in terms of their level of reuse. At the top, the query context is can be reused only within the query. At the bottom, the global context has the greatest potential for reuse across multiple sessions and users because the session context will derive from the global context and the query context will derive itself from the session context.

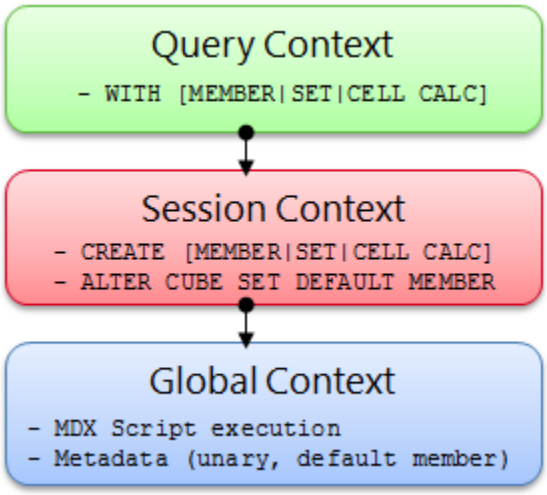


Figure 1919: Cache context layers

During execution, every MDX query must reference all three contexts to identify all of the potential calculations and security conditions that can impact the evaluation of the query. For example, to resolve a query that contains a query calculated member, the query processor creates a query context to resolve the query calculated member, creates a session context to evaluate session calculations, and creates a global context to evaluate the MDX script and retrieve the security permissions of the user who submitted the query. Note that these contexts are created only if they aren't already built. After they are built, they are reused where possible.

Even though a query references all three contexts, it will typically use the cache of a single context. This means that on a per-query basis, the query processor must select which cache to use. The query processor always attempts to use the broadly applicable cache depending on whether or not it detects the presence of calculations at a narrower context.

If the query processor encounters calculations created at query time, it always uses the query context, even if a query also references calculations from the global context (there is an exception to this –

queries with query calculated members of the form Aggregate(<set>) do share the session cache). If there are no query calculations, but there are session calculations, the query processor uses the session cache. The query processor selects the cache based on the presence of any calculation in the scope. This behavior is especially relevant to users with MDX-generating front-end tools. If the front-end tool creates any session calculations or query calculations, the global cache is not used, even if you do not specifically use the session or query calculations.

There are other calculation scenarios that impact how the query processor caches calculations. When you call a stored procedure from an MDX calculation, the engine always uses the query cache. This is because stored procedures are nondeterministic (meaning that there is no guarantee what the stored procedure will return). As a result, after a nondeterministic calculation is encountered during the query, nothing is cached globally or in the session cache. Instead, the remaining calculations are stored in the query cache. In addition, the following scenarios determine how the query processor caches calculation results:

- The use of MDX functions that are locale-dependent (such as Caption or .Properties) prevents the use of the global cache, because different sessions may be connected with different locales and cached results for one locale may not be correct for another locale.
- The use of cell security; functions such as **UserName**, **StrToSet, StrToMember,** and **StrToTuple**; or **LookupCube** functions in the MDX script or in the dimension or cell security definition disable the global cache. That is, just one expression that uses any of these functions or features disables global caching for the entire cube.
- If visual totals are enabled for the session by setting the default MDX Visual Mode property in the Analysis Services connection string to 1, the query processor uses the query cache for all queries issued in that session.
- If you enable visual totals for a query by using the MDX **VisualTotals** function, the query processor uses the query cache.
- Queries that use the subselect syntax (SELECT FROM SELECT) or are based on a session subcube (CREATE SUBCUBE) result in the query or, respectively, session cache to be used.
- Arbitrary shapes can only use the query cache if they are used in a subselect, in the WHERE clause, or in a calculated member. An arbitrary shape is any set that cannot be expressed as a crossjoin of members from the same level of an attribute hierarchy. For example, {(Food, USA), (Drink, Canada)} is an arbitrary set, as is {customer.geography.USA, customer.geography.[British Columbia]}. Note that an arbitrary shape on the query axis does not limit the use of any cache.

Based on this behavior, when your querying workload can benefit from reusing data across users, it is a good practice to define calculations in the global scope. An example of this scenario is a structured reporting workload where you have few security roles. By contrast, if you have a workload that requires individual data sets for each user, such as in an HR cube where you have many security roles or you are

using dynamic security, the opportunity to reuse calculation results across users is lessened or eliminated. As a result, the performance benefits associated with reusing the query processor cache are not as high.

2.3.1.3 Data Retrieval

When you query a cube, the query processor breaks the query into subcube requests for the storage engine. For each subcube request, the storage engine first attempts to retrieve data from the storage engine cache. If no data is available in the cache, it attempts to retrieve data from an aggregation. If no aggregation is present, it must retrieve the data from the fact data from a measure group's partition data.

Retrieving data from a partition requires I/O activity. This I/O can either be served from the file system cache or from disk. Additional details of the I/O subsystem of Analysis Services can be found in Part 2.

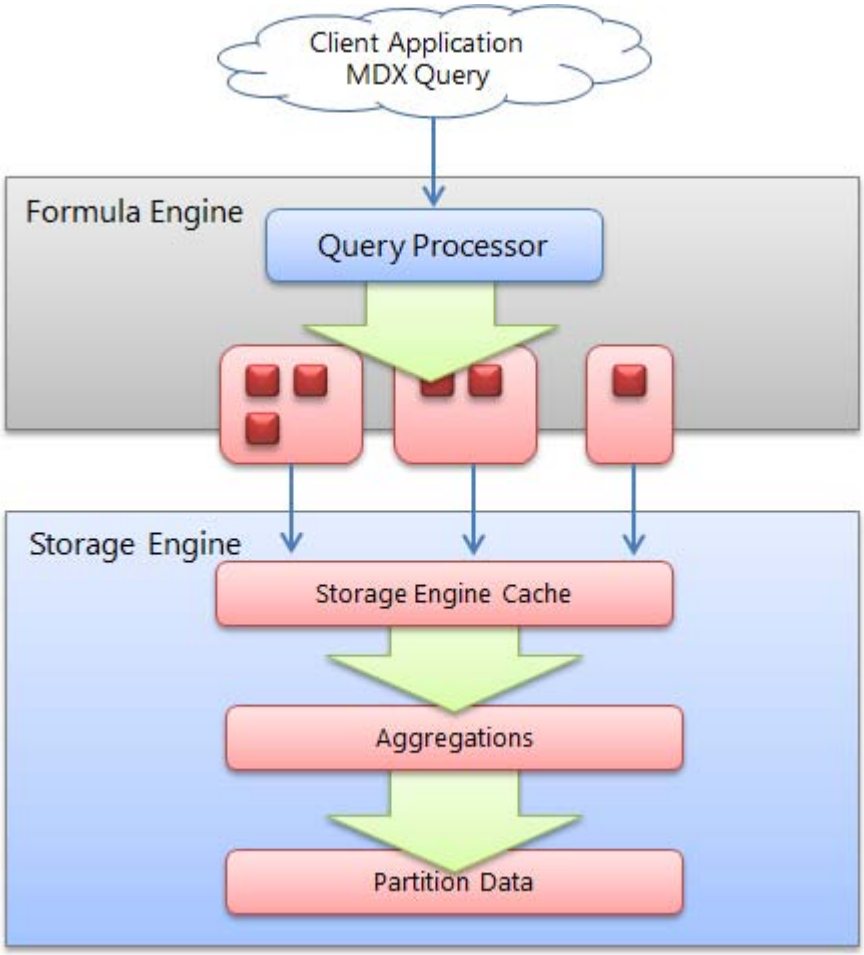


Figure 2020: High-level overview of the data retrieval process

2.3.1.3.1 Storage Engine Cache

The storage engine cache is also known as the data cache registry because it is composed of the dimension and measure group caches that are the same structurally. When a request is made from the

Analysis Services formula engine to the storage engine, it sends a request in the form of a subcube describing the structure of the data request and a data cache structure that will contain the results of the request. Using the data cache registry indexes, it attempts to find a corresponding subcube:

- If there is a matching subcube, the corresponding data cache is returned.
- If a subcube superset is found, a new data cache is generated and the results are filtered to fit the subcube request.
- If lower-grain data exists, the data cache registry can aggregate this data and make it available as well – and the new subcube and data cache are also registered in the cache registry.
- If data does not exist, the request goes to the storage engine and the results are cached in the cache registry for future queries.

Analysis Services allocates memory via memory holders that contain statistical information about the amount of memory being used. Memory holders are in the form of nonshrinkable and shrinkable memory; each combination of a subcube and data cache forms a single *shrinkable* memory holder. When Analysis Services is under heavy memory pressure, cleaner threads remove shrinkable memory. Therefore, ensure your system has enough memory; if it does not, your data cache registry will be cleared out (resulting in slower query performance) when it is placed under memory pressure.

2.3.1.3.2 Aggressive Data Scanning

Sometimes, in the evaluation of an expression, more data is requested than required to determine the result.

If you suspect more data is being retrieved than is required, you can use SQL Server Profiler to diagnose how a query into subcube query events and partition scans. For subcube scans, check the verbose subcube event and whether more members than required are retrieved from the storage engine. For small cubes, this likely isn't a problem. For larger cubes with multiple partitions, it can greatly reduce query performance. The following figure demonstrates how a single query subcube event results in partition scans.

Query Subcube	2 - Non-cache data	0000000000000000000000,00000000,000,00000,00,0000000000000000011,00000000000000...
Progress Report Begin	14 - Query	Started reading data from the 'Reseller_Sales_2001' partition.
Progress Report Begin	14 - Query	Started reading data from the 'Reseller_Sales_2002' partition.
Progress Report Begin	14 - Query	Started reading data from the 'Reseller_Sales_2003' partition.
Progress Report Begin	14 - Query	Started reading data from the 'Reseller_Sales_2004' partition.
Progress Report End	14 - Query	Finished reading data from the 'Reseller_Sales_2001' partition.
Progress Report End	14 - Query	Finished reading data from the 'Reseller_Sales_2004' partition.
Progress Report End	14 - Query	Finished reading data from the 'Reseller_Sales_2002' partition.
Progress Report End	14 - Query	Finished reading data from the 'Reseller_Sales_2003' partition.

Figure 2121: Aggressive partition scanning

There are two potential solutions to this. If a calculation expression contains an arbitrary shape (this is defined in the section on the query processor cache), the query processor may not be able to determine that the data is limited to a single partition and request data from all partitions. Try to eliminate the arbitrary shape.

Other times, the query processor is simply overly aggressive in asking for data. For small cubes, this doesn't matter, but for very large cubes, it does. If you observe this behavior, potential solutions include the following:

- Contact Microsoft Customer Service and Support for further advice.
- Disable Prefetch = 1 (this is done in the connection string): Sometimes Analysis Services requests additional data from the source to prepopulate the cache; it may help to turn it off so that Analysis Services does not request too much data.

2.3.2 Query Processor Internals

There are several changes to query processor internals in SQL Server 2008 Analysis Services that are applicable today (compared to SQL Server 2005 Analysis Services). In this section, these changes are discussed before specific optimization techniques are introduced.

2.3.2.1 Subspace Computation

The key idea behind subspace computation is best introduced by contrasting it with a cell-by-cell evaluation of a calculation. (This is also known as a naïve calculation.) Consider a trivial calculation RollingSum that sums the sales for the previous year and the current year, and a query that requests the RollingSum for 2005 for all Products.

```
RollingSum = (Year.PrevMember, Sales) + Sales

SELECT 2005 on columns, Product.Members on rows WHERE RollingSum
```

A cell-by-cell evaluation of this calculation proceeds as represented in the following figure.

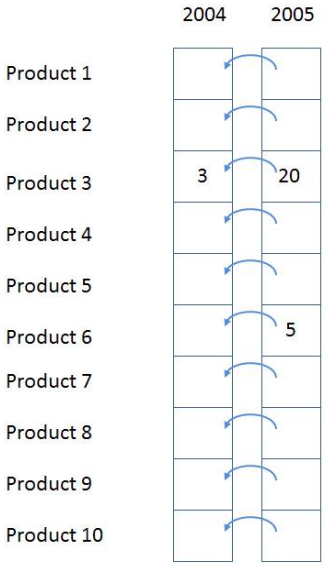


Figure 2222: Cell-by-cell evaluation

The 10 cells for **[2005, All Products]**are each evaluated in turn. For each, the previous year is located, and then the sales value is obtained and then added to the sales for the current year. There are two significant performance issues with this approach.

Firstly, if the data is *sparse* (that is, thinly populated), cells are calculated even though they are bound to return a null value. In the previous example, calculating the cells for anything but Product 3 and Product 6 is a waste of effort. The impact of this can be extreme—in a sparsely populated cube, the difference can be several orders of magnitude in the numbers of cells evaluated.

Secondly, even if the data is totally *dense*, meaning that every cell has a value and there is no *wasted*effort visiting empty cells, there is much repeated effort. The same work (for example, getting the previous Year member, setting up the new context for the previous Year cell, checking for recursion) is redone for each Product. It would be much more efficient to move this work out of the inner loop of evaluating each cell.

Now consider the same example performed using subspace computation. In subspace computation, the engine works its way down an execution tree determining what spaces need to be filled. Given the query, the following space needs to be computed, where * means every member of the attribute hierarchy.

[Product.*, 2005, RollingSum]

Given the calculation, this means that the following space needs to be computed first.

[Product.*, 2004, Sales]

Next, the following space must be computed.

[Product.*, 2005, Sales]

Finally, the + operator needs to be added to those two spaces.

If Sales were itself covered by calculations, the spaces necessary to calculate Sales would be determined and the tree would be expanded. In this case Sales is a base measure, so the storage engine data is used to fill the two spaces at the leaves, and then, working up the tree, the operator is applied to fill the space at the root. Hence the one row (Product3, 2004, 3) and the two rows { (Product3, 2005, 20), (Product6, 2005, 5)} are retrieved, and the + operator applied to them to yields the following result.

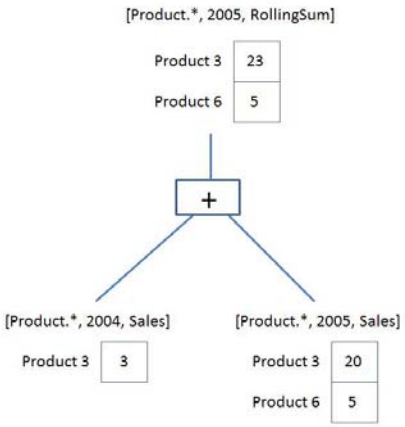


Figure 2323: Execution plan

The + operator operates on *spaces*, not simply *scalarvalues.* It is responsible for combining the two given spaces to produce a space that contains each product that appears in either space with the summed value. This is the *query execution plan*. Note that it operates only on data that could contribute to the result. There is no notion of the theoretical space over which the calculation must be performed.

A query execution plan is not one or the other but can contain both subspace and cell-by-cell nodes. Some functions are not supported in subspace mode, causing the engine to fall back to cell-by-cell mode. But even when evaluating an expression in cell-by-cell mode, the engine can return to subspace mode.

2.3.2.2 Expensive vs. Inexpensive Query Plans

It can be costly to build a query plan. In fact, the cost of building an execution plan can exceed the cost of query execution. The Analysis Services engine has a coarse classification scheme—expensive versus inexpensive. A plan is deemed *expensive* if cell-by-cell mode is used or if cube data must be read to build the plan. Otherwise the execution plan is deemed *inexpensive*.

Cube data is used in query plans in several scenarios. Some query plans result in the mapping of one member to another because of MDX functions such as **PrevMember** and **Parent**. The mappings are built from cube data and materialized during the construction of the query plans. The **IIf**, CASE, and IF functions can generate expensive query plans as well, should it be necessary to read cube data in order to partition cube space for evaluation of one of the branches. For more information, see IIf Function in SQL Server 2008 Analysis Services.

2.3.2.3 Expression Sparsity

An expression's *sparsity* refers to the number of cells with nonnull values compared to the total number of cells in the result of the evaluation of the expression. If there are relatively few nonnull values, the expression is termed sparse. If there are many, the expression is dense. As we shall see later, whether an expression is sparse or dense can influence the query plan.

But how can you tell whether an expression is dense or sparse? Consider a simple noncalculated measure – is it dense or sparse? In OLAP, base fact measures are considered sparse by the Analysis Services engine. This means that the typical measure does not have values for every attribute member. For example, a customer does not purchase most products on most days from most stores. In fact it's the quite the opposite. A typical customer purchases a small percentage of all products from a small number of stores on a few days. The following table lists some other simple rules for popular expressions.

Expression	Sparse/dense
Regular measure	Sparse
Constant Value	Dense (excluding constant null values, true/false values)
Scalar expression; for example, count, .properties	Dense
<exp1>+<exp2> **<exp1>-<exp2>**	Sparse if both exp1 and exp2 are sparse; otherwise dense
<exp1>*<exp2>	Sparse if either exp1 or exp2 is sparse; otherwise dense
<exp1> / <exp2>	Sparse if <exp1> is sparse; otherwise dense
Sum(<set>, <exp>) **Aggregate(<set>, <exp>)**	Inherited from <exp>
IIf(<cond>, <exp1>, <exp2>)	Determined by sparsity of default branch (refer to **IIf function**)

For more information about sparsity and density, see Gross margin - dense vs. sparse block evaluation mode in MDX (http://sqlblog.com/blogs/mosha/archive/2008/11/01/gross-margin-dense-vs-sparse-block-evaluation-mode-in-mdx.aspx).

2.3.2.4 Default Values

Every expression has a default value—the value the expression assumes most of the time. The query processor calculates an expression's default value and reuses across most of its space. Most of the time this is null because oftentimes (but not always) the result of an expression with null input values is null. The engine can then compute the null result once, and then it needs to compute only values for the much reduced nonnull space.

Another important use of the default values is in the condition in the **IIf** function. Knowing which branch is evaluated more often drives the execution plan. The default values of some popular expressions are listed in the following table.

Expression	Default value	Comment
Regular measure	Null	None.
IsEmpty(<regular measure>)	True	The majority of theoretical space is occupied by null values. Therefore, **IsEmpty** will return True most often.
<regular measure A> = <regular measure B>	True	Values for both measures are principally null, so this evaluates to True most of the time.
<member A> IS <member B>	False	This is different than comparing values – the engine assumes that different members are compared most of the time.

2.3.2.5 Varying Attributes

Cell values mostly depend on attribute coordinates. But some calculations do not depend on every attribute. For example, the following expressiondepends only on the Customer attribute in the customer dimension.

[Customer].[Customer Geography].properties("Postal Code")

When this expression is evaluated over a subspace involving other attributes, any attributes the expression doesn't depend on can be eliminated, and then the expression can be resolved and projected back over the original subspace. The attributes an expression depends on are termed its varying attributes. For example, consider the following query.

```
with member measures.Zip as

[Customer].[Customer Geography].currentmember.properties("Postal Code")

select measures.zip on 0,

[Product].[Category].members on 1

from [Adventure Works]
```

```
where [Customer].[Customer Geography].[Customer].&[25818]
```

The expression depends on the customer attribute and not the category attribute; therefore, customer is a varying attribute and category is not. In this case the expression is evaluated only once for the customer and not as many times as there are product categories.

2.3.3 Optimizing MDX

Debugging calculation performance issues across a cube can be difficult if there are many calculations. The first step is to try to narrow down where the problem expression is and then apply best practices to the MDX. In order to narrow down a problem, you will first need a baseline.

2.3.3.1 Baselining Query Speeds

Before beginning optimization, you need reproducible cold-cache baseline measurements.

To do this, you should be aware of the following three Analysis Services caches:

- The formula engine cache
- The storage engine cache
- The file system cache

Both the Analysis Services and the operating system caches need to be cleared before you start taking measurements.

2.3.3.1.1 Clearing the Analysis Services Caches

The Analysis Services formula engine and storage engine caches can be cleared with the XMLA **ClearCache** command. You can use SQL Server Management Studio to run **ClearCache**.

```
<ClearCache
       xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">
<Object>
<DatabaseID><database name></DatabaseID>
</Object>
</ClearCache>
```

2.3.3.1.2 Clearing the Operating System Caches

The file system cache is a bit harder to get rid of because it resides inside Windows itself. You can use any of the following tools to perform this task:

- **Fsutil.exe: Windows File System Utility**
  If you have created a separate Windows volume for the cube database, you can dismount the volume itself using the following command:

  fsutil.exe volume dismount< Drive Letter | Mount Point >

  This clears the file system cache for this drive letter or mount point. If the cube database resides only on this location, running this command results in a clean file system cache.

- **RAMMap: Sysinternals tool**
  Alternatively, you can use **RAMMap** from Sysinternals (as of this writing, RAMMap v1.11 is available at: http://technet.microsoft.com/en-us/sysinternals/ff700229.aspx). RAMMap can help you understand how Windows manages memory. This tool not only allows you to read the file system cache content, it also allows you to purge it. On the **empty** menu, click **Empty System Working Set**, and then click **Empty Standby List**. This clears the file system cache for the entire system. Note that when **RAMMap** starts up, it temporarily freezes the system while it reads the memory content – this can take some time on a large machine. Hence, **RAMMap** should be used with care.

- **Analysis Services Stored Procedure Project (CodePlex): FileSystemCache class**
  There is currently a CodePlex project called the Analysis Services Stored Procedure Project found at: http://asstoredprocedures.codeplex.com/wikipage?title=FileSystemCache. This project contains code for a utility that enables you to clear the file system cache using a stored procedure that you can run directly on Analysis Services.

Note that neither **FSUTIL** nor **RAMMap** should be used in production cubes –both cause disruption to service. Also note that neither **RAMMap** nor the Analysis Services Stored Procedures Project is supported by Microsoft.

2.3.3.1.3 Measure Query Speeds

When all caches are clear, you should initialize the calculation script by executing a query that returns and caches nothing. Here is an example.

```
select {} on 0 from [Adventure Works]
```

Execute the query you want to optimize and then use SQL Server Profiler with the **Standard (default)** trace and these additionalevents enabled:

- Query Processing\Query Subcube Verbose
- Query Processing\Get Data From Aggregation

Save the profiler trace, because it contains important information that you can use to diagnose slow query times.

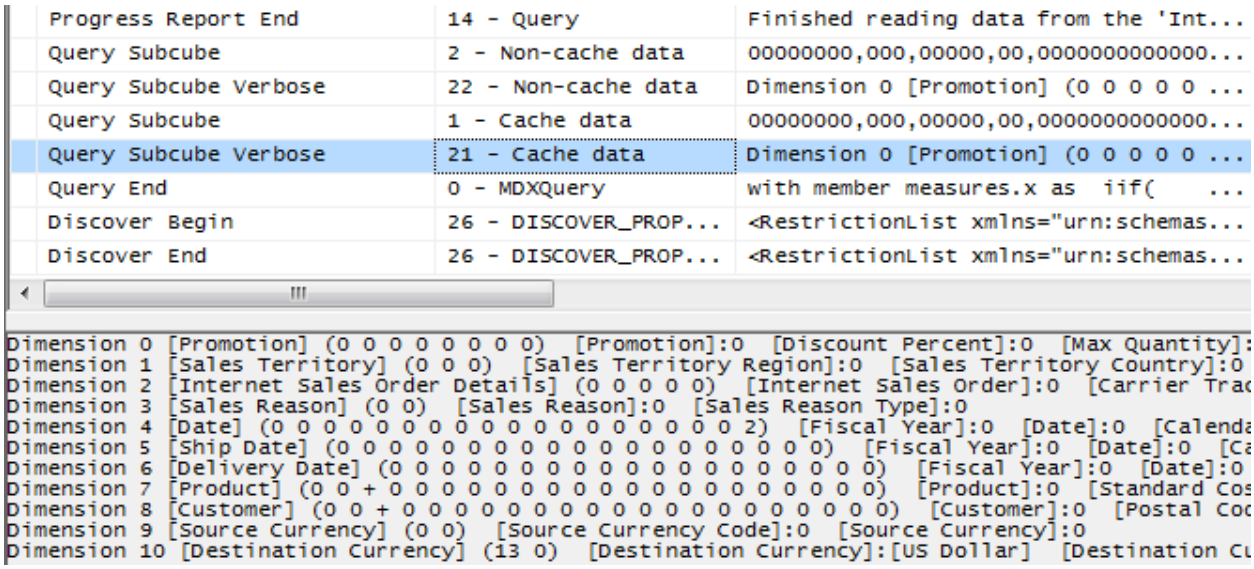


Figure 2424: Sample trace

The text for the query subcube verbose event deserves some explanation. It contains information for each attribute in every dimension:

- 0: Indicates that the attribute is not included in query (the **All** member is hit).
- * : Indicates that every member of the attribute was requested.
- + : Indicates that two or more members of the attribute were requested.
- - : Indicates that a slice below granularity is requested.
- <integer value> : Indicates that a single member of the attribute was hit. The integer represents the member's data ID (an internal identifier generated by the engine).

For more information about the query subcube verbose event textdata, see the following:

- Identifying and Resolving MDX Query Performance Bottlenecks in SQL Server 2005 Analysis Services (http://sqlcat.com/sqlcat/b/whitepapers/archive/2007/12/16/identifying-and-resolving-mdx-query-performance-bottlenecks-in-sql-server-2005-analysis-services.aspx)
- Configuring the Analysis Services Query Log (http://msdn.microsoft.com/en-us/library/cc917676.aspx): Refer to the *The Dataset Column in the Query Log Table* section

SQL Server Management Studio displays the total query time. But be careful: This time is the amount of time taken to retrieve and display the cellset. For large results, the time to render the cellset on the client can rival the time it took the server to generate it. Instead of using SQL Server Management Studio, use the SQL Server Profiler Query End event to measure how long the query takes from the server's perspective and get the Analysis Services engine duration.

2.3.3.2 Isolating the Problem

Diagnosing the problem may be straightforward if a simple query calls out a specific calculation (in which case you should continue to the next section), but if there are chains of expressions or a complex

query, it can be time-consuming to locate the problem. Try to reduce the query to the simplest expression possible that continues to reproduce the performance issue. If possible, remove expressions such as MDX scripts, unary operators, measure expressions, custom member formulas, semi-additive measures, and custom rollup properties. With some client applications, the query generated by the client itself, not the cube, can be the problem. For example, problems can arise when client applications generate queries that demand large data volumes, push down to unnecessarily low granularities, unnecessarily bypass aggregations, or contain query calculations that bypass the global and session query processor caches. If you can confirm that the issue is in the cube itself, comment out calculated members in the cube or query until you have narrowed down the offending calculation. Using a binary chop method is useful to quickly reduce the query to the simplest form that reproduces the issue. Experienced tuners will be able to quickly narrow in on typical calculation issues.

When you have removed calculations until the performance issue reproduces, the first step is to determine whether the problem lies in the query processor (the formula engine) or the storage engine. To determine the amount of time the engine spends scanning data, use the SQL Server Profiler trace created earlier. Limit the events to noncached storage engine retrievals by selecting only the query subcube verbose event and filtering on `event subclass = 22`. The result will be similar to the following.

EventClass	Clear Trace Window ectionID	Databa...	Duration	EndTime	EventSubclass
Query Subcube Verbose	50	RBA...	8	2008-08-07 13:47:24.000	22 - Non-cache data
Query Subcube Verbose	50	RBA...	9	2008-08-07 13:47:24.000	22 - Non-cache data
Query Subcube Verbose	50	RBA...	8	2008-08-07 13:47:24.000	22 - Non-cache data
Query Subcube Verbose	50	RBA...	10	2008-08-07 13:47:24.000	22 - Non-cache data
Query Subcube Verbose	50	RBA...	29	2008-08-07 13:47:24.000	22 - Non-cache data
Query Subcube Verbose	50	RBA...	11	2008-08-07 13:47:24.000	22 - Non-cache data

Figure 2525: Trace of query subcube events

If the majority of time is spent in the storage engine with long-running **query subcube** events, the problem is likely with the storage engine. In this case, consider optimizing dimension design, designing aggregations, or using partitions to improve query performance. In addition, you may want to consider optimizing the disk subsystem.

If the majority of time is not spent in the storage engine but in the query processor, focus on optimizing the MDX script or the query itself. Note, the problem can involve *both* the formula and storage engines.

A “fragmented query space” can be diagnosed with SQL Server Profiler if you see many query subcube events generated by a single query. Each request may not take long, but the sum of them may. If this is the case, consider warming the cache to make sure subcubes and calculations are already cached. Also, consider rewriting the query to remove arbitrary shapes, because arbitrary subcubes cannot be cached. For more information, see Cache Warmingin a later section.

If the cube and MDX query are already fully optimized, you may consider doing thread, memory, and configuration tuning of the cube. You may even want to look at larger hardware. Server-level tuning techniques are described in Part 2.

References:

- The SQL Server 2008 R2 Analysis Services Operations Guide(http://sqlcat.com/sqlcat/b/whitepapers/archive/2011/06/01/sql-server-2008r2-analysis-services-operations-guide.aspx)
- Predeployment I/O Best Practices (http://sqlcat.com/sqlcat/b/whitepapers/archive/2007/11/21/predeployment-i-o-best-practices.aspx): The concepts in this document provide an overview of disk I/O and its impact query performance; focus on the random I/O context.
- Scalable Shared Databases Part 5 (http://sqlcat.com/sqlcat/b/whitepapers/archive/2011/06/01/sql-server-2008r2-analysis-services-operations-guide.aspx): Review to better understand on query performance in context of random I/O vs. sequential I/O.

2.3.3.3 Cell-by-Cell Mode vs. Subspace Mode

Almost always, performance obtained by using subspace (or block computation) mode is superior to that obtained by using cell-by-cell (nor naïve) mode. For more information, including the list of functions supported in subspace mode, see "Performance Improvements for MDX in SQL Server 2008 Analysis Services (http://msdn.microsoft.com/en-us/library/bb934106(v=SQL.105).aspx) in SQL Server Books Online.

The following table lists the most common reasons for leaving subspace mode.

Feature or function	Comment

Set aliases	Replace with a set expression rather than an alias. For example, this query operates in subspace mode.
	
	`with`
	`member measures.SubspaceMode as`
	`  sum(`
	`  [Product].[Category].[Category].members,`
	`  [Measures].[Internet Sales Amount]`
	`  )`
	`select`
	`{measures.SubspaceMode,[Measures].[Internet Sales Amount]} on 0 ,`
	`[Customer].[Customer Geography].[Country].memberson 1`
	`from [Adventure Works]`
	`cellpropertiesvalue`
	
	However, almost the same query ,where the set is replaced with an alias, operates in cell-by-cell mode:

	with set y as [Product].[Category].[Category].members member measures.Naive as sum(y, [Measures].[Internet Sales Amount]) select {measures.Naive,[Measures].[Internet Sales Amount]} on 0 , [Customer].[Customer Geography].[Country].memberson 1 from [Adventure Works] cell properties value Note: This functionality has been fixed with the latest service pack of SQL Server 2008 R2 Analysis Services.
Late binding in functions: **LinkMember**, **StrToSet**, **StrToMember**, **StrToValue**	Late-binding functions are functions that depend on query context and cannot be statically evaluated. For example, the following code is statically bound. withmember measures.x as (strtomember("[Customer].[Customer Geography].[Country].&[Australia]"),[Measures].[Internet Sales Amount]) select measures.x on 0, [Customer].[Customer Geography].[Country].memberson 1 from [Adventure Works] cell properties value A query is late-bound if an argument can be evaluated only in context. withmember measures.x as (strtomember([Customer].[Customer Geography].currentmember.uniquename),[Measures].[Internet Sales Amount]) select measures.x on 0, [Customer].[Customer Geography].[Country].memberson 1 from [Adventure Works] cell properties value
User-defined stored procedures	User-defined stored procedures are evaluated in cell-by-cell mode. Some popular Microsoft Visual Basic for Applications (VBA) functions are natively supported in MDX, but they are still not optimized to work in subspace mode.
LookupCube	Linked measure groups are often a viable alternative.
Application of cell level security	By definition, cell level security requires cell-by-cell evaluation to ensure the correct security context is applied; therefore performance improvements of block computation cannot be applied.

2.3.3.4 Avoid Assigning Nonnull Values to Otherwise Empty Cells

The Analysis Services engine is very efficient at using sparsity of the data to improve performance. Adding calculations with nonempty values replacing empty values does not allow Analysis Services to eliminate these rows. For example, the following query replaces empty values with the dash; therefore the **non empty** keyword does not eliminate them.

```
with member measures.x as

iif( not isempty([Measures].[Internet Sales Amount]),[Measures].[Internet Sales
Amount],"-")

select descendants([Date].[Calendar].[Calendar Year].&[2004] ) on 0,

non empty [Customer].[Customer Geography].[Customer].members on 1

from [Adventure Works]

where measures.x
```

Note, **non empty** operates on cell values but not on formatted values. In rare cases you can instead use the format string to replace null values with the same character while still eliminating empty rows and columns in roughly half the execution time.

```
with member measures.x as

[Measures].[Internet Sales Amount], FORMAT_STRING = "#.00;(#.00);#.00;-"

select descendants([Date].[Calendar].[Calendar Year].&[2004] ) on 0,

non empty [Customer].[Customer Geography].[Customer].members on 1

from [Adventure Works]

where measures.x
```

The reason this can only be used in rare cases is that the query is not equivalent – the second query eliminates completely empty rows. More importantly, neither Excel nor SQL Server Reporting Services supports the fourth argument in the format_string.

References:

- For more information about using the format_string calculation property, see FORMAT_STRING Contents (MDX) (http://msdn.microsoft.com/en-us/library/ms146084.aspx)in SQL Server Books Online.
- For more information about how Excel uses the format_string property, see Create or delete a custom number format(http://office.microsoft.com/en-us/excel-help/create-or-delete-a-custom-number-format-HP010342372.aspx).

2.3.3.5 Sparse/Dense Considerations with "expr1 * expr2" Expressions

When you write expressions as products of two other expressions, place the *sparser* one on the left-hand side. Recall, an expression is sparse if there are few non-null values compared to the total number of cells; for more information, see Expression Sparsity earlier in this section.

Consider the following two queries, which have the signature of a currency conversion calculation of applying the exchange rate at leaves of the date dimension in Adventure Works. The only difference is that the order of the expressions in the product of the cell calculation changes. The results are the same, but using the sparser internet sales amount first results in about a 10% savings. (That's not much in this case, but it could be substantially more in others. Savings depends on relative sparsity between the two expressions, and performance benefits may vary).

Sparse First

```
with cell CALCULATION x for '({[Measures].[Internet Sales Amount]},leaves([Date]))'

as [Measures].[Internet Sales Amount] *

([Measures].[Average Rate],[Destination Currency].[Destination Currency].&[EURO])

select

non empty [Date].[Calendar].members on 0,

non empty [Product].[Product Categories].members on 1

from [Adventure Works]

where ([Measures].[Internet Sales Amount], [Customer].[Customer Geography].[State-Province].&[BC]&[CA])
```

Dense First

```
with cell CALCULATION x for '({[Measures].[Internet Sales Amount]},leaves([Date]))'

as

([Measures].[Average Rate],[Destination Currency].[Destination Currency].&[EURO])*

[Measures].[Internet Sales Amount]

select

non empty [Date].[Calendar].members on 0,

non empty [Product].[Product Categories].members on 1

from [Adventure Works]

where ([Measures].[Internet Sales Amount], [Customer].[Customer Geography].[State-Province].&[BC]&[CA])
```

2.3.3.6 IIf Function in SQL Server 2008 Analysis Services

The **IIf** MDX function is a commonly used expression that can be costly to evaluate. The engine optimizes performance based on a few simple criteria. The **IIf** function takes three arguments:

```
iif(<condition>, <then branch>, <else branch>)
```

Where the condition evaluates to true, the value from the then branch is used; otherwise the else branch expression is used. Note the term *used* – one or both branches may be evaluated even if the value is not used. It may be cheaper for the engine to evaluate the expression over the entire space and use it when needed - termed an *eager* plan – than it would be to chop up the space into a potentially enormous number of fragments and evaluate only where needed - a *strict* plan.

> **Note:** One of the most common errors in MDX scripting is using **IIf** when the condition depends on cell coordinates instead of values. If the condition depends on cell coordinates, use scopes and assignments as described in section 2. When this is done, the condition is not evaluated over the space and the engine does not evaluate one or both branches over the entire space. Admittedly, in some cases, using assignments forces some unwieldy scoping and repetition of assignments, but it is always worthwhile comparing the two approaches.

IIf considerations:

1) The first consideration is whether the *query plan isexpensive or inexpensive*.
   Most **IIf** condition query plans are inexpensive, but complex nested conditions with more **IIf** functions can go to cell by cell.
2) The next consideration the engine makes is *what value the condition takes most*. This is driven by the condition's default value. If the condition's default value is true, the then branch is the default branch – the branch that is evaluated over most of the subspace.

Knowing a few simple rules on how the condition is evaluated helps to determine the default branch:

- In sparse expressions, most cells are empty. The default value of the **IsEmpty** function on a sparse expression is true.
- Comparison to zero of a sparse expression is true.
- The default value of the IS operator is false.
- If the condition cannot be evaluated in subspace mode, there is no default branch.

For example, one of the most common uses of the **IIf** function is to check whether the denominator is nonzero:

```
iif([Measures].[Internet Sales Amount]=0
  , null
  , [Measures].[Internet Order Quantity]/[Measures].[Internet Sales Amount])
```

There is no calculation on Internet Sales Amount; therefore it is a *regular measure expression* and it is sparse. Therefore the default value of the condition is true. Thus the default branch is the then branch with the null expression.

The following table shows how each branch of an **IIf** function is evaluated.

Branch query plan	Branch is default branch	Branch expression sparsity	Evaluation
Expensive	Not applicable	Not applicable	Strict
Inexpensive	True	Not applicable	Eager
Inexpensive	False	Dense	Strict
Inexpensive	False	Sparse	Eager

In SQL Server 2008 Analysis Services, you can overrule the default behavior with query hints.

```
iif(
      [<condition>
      , <then branch> [hint [Eager | Strict]]
      , <else branch> [hint [Eager | Strict]]
)
```

Here are the most common scenarios where you might want to change the default behavior:

- The engine determines the query plan for the condition is expensive and evaluates each branch in strict mode.
- The condition is evaluated in cell-by-cell mode, and each branch is evaluated in eager mode.
- The branch expression is dense but easily evaluated.

For example, consider the following simple expression, which takes the inverse of a measure.

```
with member

measures.x as

iif(

   [Measures].[Internet Sales Amount]=0

   , null

   , (1/[Measures].[Internet Sales Amount]) )

select {[Measures].x} on 0,
```

```
[Customer].[Customer Geography].[Country].members *

[Product].[Product Categories].[Category].members on 1

from [Adventure Works]

cell properties value
```

The query plan is not expensive, the else branch is not the default branch, and the expression is dense, so it is evaluated in strict mode. This forces the engine to materialize the space over which it is evaluated. This can be seen in SQL Server Profiler with query subcube verbose events selected as displayed in Figure 26.

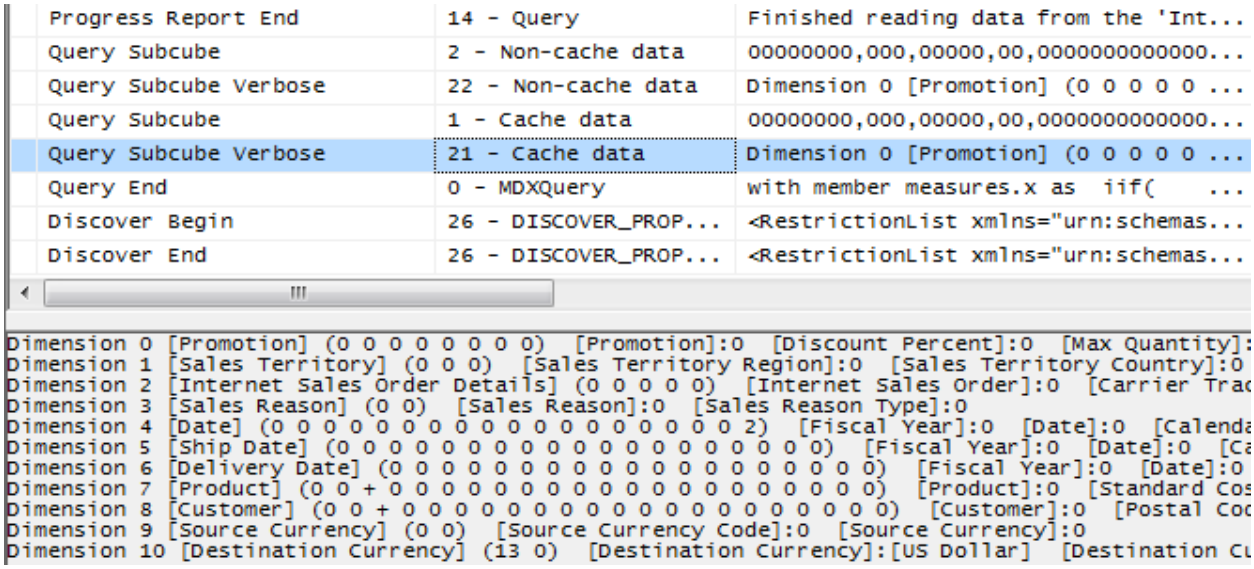


Figure 2626: Default IIf query trace

Note the subcube definition for the Product and Customer dimensions (dimensions 7 and 8 respectively) with the '+' indicator on the Country and Category attributes. This means that more than one but not all members are included – the query processor has determined which tuples meet the condition and partitioned the space, and it is evaluating the fraction over that space.

To prevent the query plan from partitioning the space, the query can be modified as follows (in bold).

```
with member

measures.x as

iif(
```

```
   [Measures].[Internet Sales Amount]=0

   , null

   , (1/[Measures].[Internet Sales Amount]) hint eager)

select {[Measures].x} on 0,

[Customer].[Customer Geography].[Country].members *

[Product].[Product Categories].[Category].members on 1

from [Adventure Works]

cell properties value
```

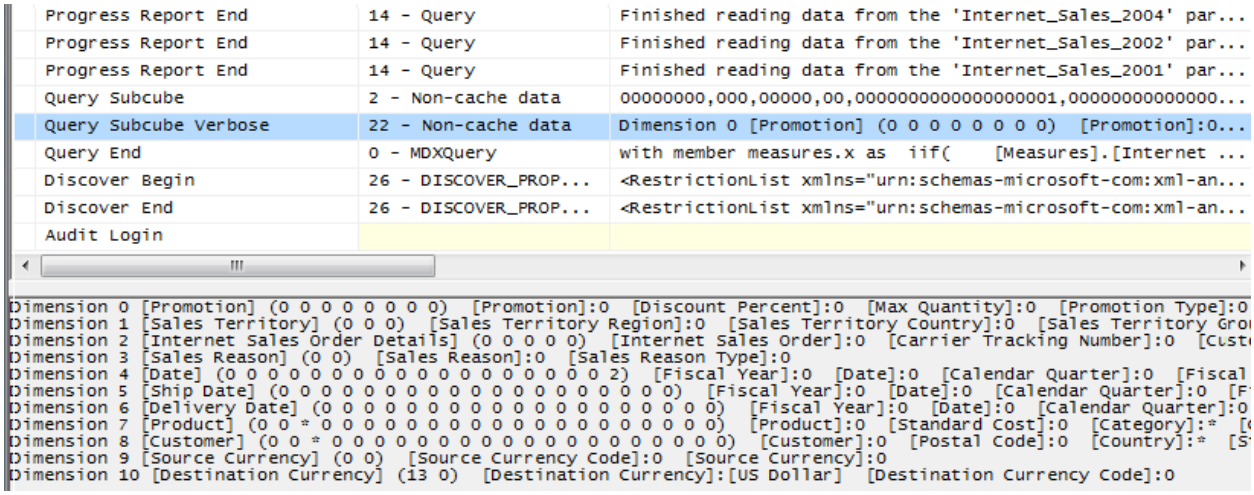


Figure 2727: IIf trace with MDX query hints

Now the same attributes are marked with a '*' indicator, meaning that the expression is evaluated over the entire space instead of a partitioned space.

2.3.3.7 Cache Partial Expressions and Cell Properties

Partial expressions (those that are part of a calculated member or assignment) are not cached. So if an expensive subexpression is used more than once, consider creating a separate calculated member to allow the query processor to cache and reuse. For example, consider the following.

```
this = iif(<expensive expression >= 0, 1/<expensive expression>, null);
```

The repeated partial expressions can be extracted and replaced with a hidden calculated member as follows.

```
create member currentcube.measures.MyPartialExpression as <expensive expression> ,
visible=0;

this = iif(measures.MyPartialExpression >= 0, 1/ measures.MyPartialExpression, null);
```

Only the value cell property is cached. If you have complex cell properties to support such things as bubble-up exception coloring, consider creating a separate calculated measure. For example, this expression includes color in the definition, which creates extra work every time the expression is used.

```
create member currentcube.measures.[Value] as <exp> , backgroundColor=<complex
expression>;
```

The following is more efficient because it creates a calculated measure to handle the color effect.

```
create member currentcube.measures.MyCellProperty as <complex expression> ,
visible=0;

create member currentcube.measures.[Value] as <exp> ,
backgroundColor=<MyCellProperty>;
```

2.3.3.8 Eliminate Varying Attributes in Set Expressions

Set expressions do not support *varying attributes*. This impacts all set functions including **Filter**, **Aggregate**, **Avg**, and others. You can work around this problem by explicitly overwriting invariant attributes to a single member.

For example, in this calculation, the average of sales only including those exceeding $100 is computed.

```
with member measures.AvgSales as
avg(
      filter(
            descendants([Customer].[Customer Geography].[All Customers],,leaves)
            , [Measures].[Internet Sales Amount]>100
      )
      ,[Measures].[Internet Sales Amount]
)
select measures.AvgSales on 0,
[Customer].[Customer Geography].[City].members on 1
from [Adventure Works]
```

On a desktop box, this calculation takes approximately 2:29. However, the average of sales for all customers everywhere does not depend on the current city (this is just another way of saying that city is not a varying attribute). You can explicitly eliminate city as a varying attribute by overwriting it to the all member as follows.

```
with member measures.AvgSales as
avg(
      filter(
            descendants([Customer].[Customer Geography].[All Customers],,leaves)
            , [Measures].[Internet Sales Amount]>100
      )
      ,[Measures].[Internet Sales Amount]
)
member measures.AvgSalesWithOverWrite as (measures.AvgSales, [All Customers])
select measures.AvgSalesWithOverWrite on 0,
[Customer].[Customer Geography].[City].members on 1
from [Adventure Works]
```

With the modification, this query takes less than two seconds to complete. The following is a partial view aggregating the SQL Server Profiler traces of the two queries in the example by **EventClass** and **EventSubClass**.

EventClass > EventSubClass	AvgSalesWithOverwrite		AvgSales	
	Events	Duration	Events	Duration
Query Cube End	1	515	1	161526
Serial Results End	1	499	1	161526
Query Dimension	586			
Get Data From Cache > Get Data from Flat Cache	586			
Query Subcube > Non-Cache Data	5	64	5	218

The **Query Subcube**>**Non-Cache Data** durations are relatively small, denoting that most of the query calculation is done by the Analysis Services formula engine. This is apparent with the *AvgSales* calculation because most of the query durations correspond to the *Serial Results* event, which reports the status of serializing axes and cells. The use of `[All Customers]` ensures that the expression is evaluated only once for each Customer, improving performance.

2.3.3.9 Eliminate Cost of Computing Formatted Values

In some circumstances, the cost of determining the format string for an expression outweighs the cost of the value itself. To determine whether this applies to a slow-running query, compare execution times with and without the formatted value cell property, as in the following query.

```
select [Measures].[Internet Average Sales Amount] on 0 from [Adventure Works] cell
properties value
```

If the result is noticeable faster without the formatting, apply the formatting directly in the script as follows.

```
scope([Measures].[Internet Average Sales Amount]);
    FORMAT_STRING(this) = "currency";
end scope;
```

Execute the query (with formatting applied) to determine the extent of any performance benefit.

2.3.3.10 NON_EMPTY_BEHAVIOR

In some situations, it is expensive to compute the result of an expression, even if you know it will be null beforehand based on the value of some indicator tuple. In earlier versions of SQL Server Analysis Services, the **NON_EMPTY_BEHAVIOR** property is sometimes helpful for these kinds of calculations. When this property evaluates to null, the expression is guaranteed to be null and (most of the time) vice versa.

This property oftentimes resulted in substantial performance improvements in past releases. However, starting with SQL Server 2008, the property is oftentimes ignored (because the engine automatically deals with nonempty cells in many cases) and can sometimes result in degraded performance. Eliminate it from the MDX script and add it back after performance testing demonstrates improvement.

For assignments, the property is used as follows.

```
this = <e1>;
Non_Empty_Behavior(this) = <e2>;
```

For calculated members in the MDX script, the property is used this way.

```
create member currentcube.measures.x as <e1>, non_empty_behavior = <e2>
```

In SQL Server 2005 Analysis Services, there were complex rules on how the property could be defined, when the engine used it or ignored it, and how the engine would use it. In SQL Server 2008 Analysis Services, the behavior of this property has changed:

- It remains a guarantee that when NON_EMPTY_BEHAVIOR is null that the expression must also be null. (If this is not true, incorrect query results can still be returned.)
- However, the reverse is not necessarily true; that is, the NON_EMPTY_BEHAVIOR expression can return non null when the original expression is null.
- The engine more often than not ignores this property and deduces the nonempty behavior of the expression on its own.

If the property is defined and is applied by the engine, it is semantically equivalent (not performance equivalent, however) to the following expression.

```
this = <e1> * iif(isempty(<e2>), null, 1)
```

The NON_EMPTY_BEHAVIOR property is used if <e2> is sparse and <e1> is dense or <e1> is evaluated in the naïve *cell-by-cell* mode. If these conditions are not met and both <e1> and <e2> are sparse (that is, if <e2> is much sparser than <e1>), you may be able to achieve improved performance by forcing the behavior as follows.

```
this = iif(isempty(<e2>), null, <e1>);
```

The NON_EMPTY_BEHAVIOR property can be expressed as a simple tuple expression including simple member navigation functions such as .prevmember or .parent or an enumerated set. An enumerated set is equivalent to NON_EMPTY_BEHAVIOR of the resultant sum.

References

Below are links to some handy MDX optimization articles, books, and blog posts:

- Query calculated members invalidate formula engine cache (http://cwebbbi.wordpress.com/2009/01/30/formula-caching-and-query-scope/) by Chris Webb
- Subselect preventing caching (http://cwebbbi.wordpress.com/2008/10/28/reporting-services-generated-mdx-subselects-and-formula-caching/) by Chris Webb
- Measure datatypes (http://bidshelper.codeplex.com/wikipage?title=Measure%20Group%20Health%20Check&ProjectName=bidshelper)
- Currency datatype (http://sqlcat.com/sqlcat/b/technicalnotes/archive/2008/09/25/the-many-benefits-of-money-data-type.aspx)

2.3.4 Aggregations

An *aggregation* is a data structure that stores precalculated data that Analysis Services uses to enhance query performance. You can define the aggregation design for each partition independently. Each partition can be thought of as being an aggregation at the lowest granularity of the measure group. Aggregations that are defined for a partition are processed out of the leaf level partition data by aggregating it to a higher granularity.

When a query requests data at higher levels, the aggregation structure can deliver the data more quickly because the data is already aggregated in fewer rows. As you design aggregations, you must consider the querying benefits that aggregations provide compared with the time it takes to create and refresh the aggregations. In fact, adding unnecessary aggregations can worsen query performance because the rare hits move the aggregation into the file cache at the cost of moving something else out.

While aggregations are physically designed per measure group partition, the optimization techniques for maximizing aggregation design apply whether you have one or many partitions. In this section, unless otherwise stated, aggregations are discussed in the fundamental context of a cube with a single measure group and single partition. For more information about how you can improve query performance using multiple partitions, see Partition Strategy.

2.3.4.1 Detecting Aggregation Hits

Use SQL Server Profiler to view how and when aggregations are used to satisfy queries. Within SQL Server Profiler, there are several events that describe how a query is fulfilled. The event that specifically pertains to aggregation hits is the **Get Data From Aggregation** event.

EventClass	EventSubclass	TextData
Query Begin	0 - MDXQuery	select category.members on rows, [Measures].[Ord...
Query Cube Begin		
Get Data From Aggregation		Aggregation c 0000,0001,0000
Progress Report Begin	14 - Query	Started reading data from the 'Aggregation c' aggregation.
Progress Report End	14 - Query	Finished reading data from the 'Aggregation c' aggregat...
Query Subcube	2 - Non-cache data	0000,0001,0000
Get Data From Cache	1 - Get data from measure group cache	Dimension 0 [Customer] (0 0 0 0) [Customer Key]:0 [Ge...
Query Subcube	1 - Cache data	0000,0000,0000
Get Data From Cache	1 - Get data from measure group cache	Dimension 0 [Customer] (0 0 0 0) [Customer Key]:0 [Ge...
Query Subcube	1 - Cache data	0000,0000,0000
Get Data From Cache	1 - Get data from measure group cache	Dimension 0 [Customer] (0 0 0 0) [Customer Key]:0 [Ge...
Query Subcube	1 - Cache data	0000,0001,0000
Query Cube End		
Query End	0 - MDXQuery	select category.members on rows, [Measures].[Ord...

Figure 2828: Scenario 1: SQL Server Profiler trace for cube with an aggregation hit

This figure displays a SQL Server Profiler trace of the query's resolution against a cube with aggregations. In the SQL Server Profiler trace, the operations that the storage engine performs to produce the result set are revealed.

The storage engine gets data from Aggregation C 0000, 0001, 0000 as indicated by the **Get Data From Aggregation** event. In addition to the aggregation name, Aggregation C, Figure 29 displays a vector, **000, 0001, 0000**, that describes the content of the aggregation. More information on what this vector actually means is described in the next section, How to Interpret Aggregations. The aggregation data is loaded into the storage engine measure group cache from where the query processor retrieves it and returns the result set to the client.

When no aggregations can satisfy the query request, notice the missing **Get Data From Aggregation** event from the same cube with no aggregations as noted in the following figure.

EventClass	EventSubclass	TextData
Query Begin	0 - MDXQuery	select category.members on rows, [Measures].[Order Qu...
Query Cube Begin		
Progress Report Begin	14 - Query	Started reading data from the 'Factintsalesnonulls' partition.
Progress Report End	14 - Query	Finished reading data from the 'Factintsalesnonulls' partition.
Query Subcube	2 - Non-cache data	0000,0001,0000
Get Data From Cache	1 - Get data from measure group cache	Dimension 0 [Customer] (0 0 0 0) [Customer Key]:0 [Gender]...
Query Subcube	1 - Cache data	0000,0000,0000
Get Data From Cache	1 - Get data from measure group cache	Dimension 0 [Customer] (0 0 0 0) [Customer Key]:0 [Gender]...
Query Subcube	1 - Cache data	0000,0000,0000
Get Data From Cache	1 - Get data from measure group cache	Dimension 0 [Customer] (0 0 0 0) [Customer Key]:0 [Gender]...
Query Subcube	1 - Cache data	0000,0001,0000
Query Cube End		
Query End	0 - MDXQuery	select category.members on rows, [Measures].[Order Qu...

Figure 2929: Scenario 2: SQL Server Profiler trace for cube with no aggregation hit

After the query is submitted, rather than retrieving data from an aggregation, the storage engine goes to the detail data in the partition. From this point, the process is the same. The data is loaded into the storage engine measure group cache.

2.3.4.2 How to Interpret Aggregations

When Analysis Services creates an aggregation, each dimension is named by a vector, indicating whether the attribute points to the attribute or to the **All** level. The Attribute level is represented by 1 and the All level is represented by 0. For example, consider the following examples of aggregation vectors for the product dimension:

- **Aggregation By ProductKey Attribute**= [Product Key]:1 [Color]:0 [Subcategory]:0 [Category]:0 or **1000**
- **Aggregation By Category Attribute**= [Product Key]:0 [Color]:0 [Subcategory]:0 [Category]:1 or **0001**
- **Aggregation By ProductKey.All** and **Color.All** and **Subcategory.All** and **Category.All** = [Product Key]:0 [Color]:0 [Subcategory]:0 [Category]:0 or **0000**

To identify each aggregation, Analysis Services combines the dimension vectors into one long vector path, also called a *subcube*, with each dimension vector separated by commas.

The order of the dimensions in the vector is determined by the order of the dimensions in the measure group. To find the order of dimensions in the measure group, use one of the following two techniques:

1. With the cube opened in SQL Server Business Intelligence Development Studio, review the order of dimensions in a measure group on the **Cube Structure** tab. The order of dimensions in the cube is displayed in the **Dimensions** pane.
2. As an alternative, review the order of dimensions listed in the cube's XMLA definition.

The order of attributes in the vector for each dimension is determined by the order of attributes in the dimension. You can identify the order of attributes in each dimension by reviewing the dimension XML file.

For example, the subcube definition (0000, 0001, 0001) describes an aggregation for the following:

- Product – All, All, All, All
- Customer – All, All, All, State/Province
- Order Date – All, All, All, Year

Understanding how to read these vectors is helpful when you review aggregation hits in SQL Server Profiler. In SQL Server Profiler, you can view how the vector maps to specific dimension attributes by enabling the **Query Subcube Verbose** event. In some cases (such as when attributes are disabled), it may be easier to view the **Aggregation Design** tab and use the Advanced View of the aggregations.

2.3.4.3 Aggregation Tradeoffs

Aggregations can improve query response time but they can increase processing time and disk storage space, use up memory that could be allocated to cache, and potentially slow the speed of other queries. The latter may occur because there is a direct correlation between the number of aggregations and the duration for the Analysis Services storage engine to parse them. As well, aggregations may cause thrashing due to their potential impact to the file system cache. A general rule of thumb is that aggregations should be less than 1/3 the size of the fact table.

2.3.4.4 BuildingAggregations

Individual aggregations are organized into collections of aggregations called AggregationDesigns. You can apply an AggregationDesign to many partitions. As well, one measure group can have multiple AggregationDesigns so that you can choose different sets of aggregations for different partitions. To help Analysis Services successfully apply the AggregationDesign algorithm, you can perform the following optimization techniques to influence and enhance the AggregationDesign. In this section we will discuss the following:

- The importance of attribute hierarchies
- Aggregation design and partitions
- Specifying statistics about cube data
- Suggesting aggregation candidates
- Usage-based optimization
- Large cube aggregations
- Distinct count partition aggregation considerations

2.3.4.4.1 Importance of Attribute Hierarchies

Aggregations work better when the cube is based on a multidimensional data model that includes natural hierarchies. While it is common in relational databases to have attributes independent of each other, multidimensional star schemas have attributes related to each other to create natural hierarchies. This is important because it allows aggregations built at a lower level of a natural hierarchy to be used when querying at a higher level.

Note that attributes that are exposed only in attribute hierarchies are not automatically considered for aggregation by the Aggregation Design Wizard. Therefore, queries involving these attributes are satisfied by summarizing data from the primary key. Without the benefit of aggregations, query

performance against these attributes hierarchies can be slow. To enhance performance, it is possible to flag an attribute as an aggregation candidate by using the **Aggregation Usage** property. For more information about this technique, see Suggesting Aggregation Candidates. However, before you modify the **Aggregation Usage** property, you should consider whether you can take advantage of user hierarchies.

2.3.4.4.2 Aggregation Design and Partitions

When you define your partitions, they do not necessarily have to contain uniform datasets or aggregation designs. For example, for a given measure group, you may have 3 yearly partitions, 11 monthly partitions, 3 weekly partitions, and 1–7 daily partitions. Heterogeneous partitions with different levels of detail allows you to more easily manage the loading of new data without disturbing existing, larger, and stale partitions (more on this in the processing section) and you can design aggregations for groups of partitions that share the same access pattern. For each partition, you can use a different aggregation design. By taking advantage of this flexibility, you can identify those data sets that require higher aggregation design.

Consider the following example. In a cube with multiple monthly partitions, new data may flow into the single partition corresponding to the latest month. Generally that is also the partition most frequently queried. A common aggregation strategy in this case is to perform usage-based optimization to the most recent partition, leaving older, less frequently queried partitions as they are.

If you automate partition creation, it is easy to simply set the AggregationDesignID for the new partition at creation time and specify the slice for the partition; now it is ready to be processed. At a later stage, you may choose to update the aggregation design for a partition when its usage pattern changes – again, you can just update the AggregationDesignID, but you will also need to invoke **ProcessIndexes** so that the new aggregation design takes effect for the processed partition.

2.3.4.4.3 Specifying Statistics About Cube Data

To make intelligent assessments of aggregation costs, the design algorithm analyzes statistics about the cube for each aggregation candidate. Examples of this metadata include member counts and fact table counts. Ensuring that your metadata is up-to-date can improve the effectiveness of your aggregation design.

Whenever you use multiple partitions for a given measure group, ensure that you update the data statistics for each partition. More specifically, it is important to ensure that the partition data and member counts (such as **EstimatedRows** and **EstimatedCount** properties) accurately reflect the specific data in the partition and not the data across the entire measure group.

2.3.4.4.4 Suggesting Aggregation Candidates

When Analysis Services designs aggregations, the aggregation design algorithm does not automatically consider every attribute for aggregation. Consequently, in your cube design, verify the attributes that are considered for aggregation and determine whether you need to suggest additional aggregation candidates. To streamline this process, Analysis Services uses the **Aggregation Usage** property to determine which attributes it should consider. For every measure group, verify the attributes that are

automatically considered for aggregation and then determine whether you need to suggest additional aggregation candidates.

Aggregation Usage Rules

An *aggregation candidate* is an attribute that Analysis Services considers for potential aggregation. To determine whether or not a specific attribute is an aggregation candidate, the storage engine relies on the value of the **Aggregation Usage** property. The **Aggregation Usage** property is assigned a per-cube attribute, so it globally applies across all measure groups and partitions in the cube. For each attribute in a cube, the **Aggregation Usage** property can have one of four potential values: **Full**, **None**, **Unrestricted**, and **Default**.

- **Full**— Every aggregation for the cube must include this attribute or a related attribute that is lower in the attribute chain. For example, you have a product dimension with the following chain of related attributes: Product, Product Subcategory, and Product Category. If you specify the **Aggregation Usage** for Product Category to be **Full**, Analysis Services may create an aggregation that includes Product Subcategory as opposed to Product Category, given that Product Subcategory is related to Category and can be used to derive Category totals.
- **None**—No aggregation for the cube can include this attribute.
- **Unrestricted**—No restrictions are placed on the aggregation designer; however, the attribute must still be evaluated to determine whether it is a valuable aggregation candidate.
- **Default**—The designer applies a *default rule* based on the type of attribute and dimension. This is the default value of the **Aggregation Usage** property.

The default rule is highly conservative about which attributes are considered for aggregation. The default rule is broken down into four constraints.

- **Default Constraint 1—Unrestricted** - For a dimension's measure group granularity attribute, default means **Unrestricted**. The granularity attribute is the same as the dimension's key attribute as long as the measure group joins to a dimension using the primary key attribute.
- **Default Constraint 2—None for Special Dimension Types -** For all attributes (except All) in many-to-many, nonmaterialized reference dimensions, and data mining dimensions, default means **None**. This means you can sometimes benefit from creating leaf level projections for many-to-many dimensions. Note, these defaults do not apply for parent-child dimensions; for more information, see the Special Considerations > Parent-Child dimensions section.
- **Default Constraint 3—Unrestricted for Natural Hierarchies -** A naturalhierarchy is a user hierarchy where all attributes participating in the hierarchy contain attribute relationships to the attribute sourcing the next level. For such attributes, default means **Unrestricted,** except for nonaggregatable attributes, which are set to **Full** (even if they are not in a user hierarchy).
- **Default Constraint 4—None For Everything Else**. For all other dimension attributes, default means **None**.

Aggregation Usage Guidelines

In light of the behavior of the **Aggregation Usage** property, use the following guidelines:

- **Attributes exposed solely as attribute hierarchies**- If a given attribute is only exposed as an attribute hierarchy such as Color, you may want to change its **Aggregation Usage** property as follows.
    - First, change the value of the **Aggregation Usage** property from **Default** to **Unrestricted** if the attribute is a commonly used attribute or if there are special considerations for improving the performance in a particular pivot or drilldown. For example, if you have highly summarized scorecard style reports, you want to ensure that the users experience good initial query response time before drilling around into more detail.
    - While setting the **Aggregation Usage** property of a particular attribute hierarchy to **Unrestricted** is appropriate is some scenarios, do not set all attribute hierarchies to **Unrestricted**. Increasing the number of attributes to be considered increases the problem space the aggregation algorithm must consider. The wizard can take at least an hour to complete the design and considerably much more time to process. Set the property to **Unrestricted** only for the commonly queried attribute hierarchies. The general rule is five to ten **Unrestricted** attributes per dimension.
    - Next, change the value of the **Aggregation Usage** property from **Default** to **Full** in the unusual case that it is used in virtually every query you want to optimize. This is a rare case, and this change should be made only for attributes that have a relatively small number of members.
- **Infrequently used attributes**—For attributes participating in natural hierarchies, you may want to change the **Aggregation Usage** property from **Default** to **None** if users would only infrequently use it. Using this approach can help you reduce the aggregation space and get to the five to ten **Unrestricted** attributes per dimension. For example, you may have certain attributes that are only used by a few advanced users who are willing to accept slightly slower performance. In this scenario, you are essentially forcing the aggregation design algorithm to spend time building only the aggregations that provide the most benefit to the majority of users.

2.3.4.4.5 Usage-Based Optimization

The Usage-Based Optimization Wizard reviews the queries in the query log (which you must set up beforehand) and designs aggregations that cover *up to the top 100* slowest queries. Use the Usage-Based Optimization Wizard with a 100% performance gain - this will design aggregations to avoid hitting the partition directly.

After the aggregations are designed, you can add them to the existing design or completely replace the design. Be careful adding them to the existing design – the two designs may contain aggregations that serve almost identical purposes that when combined are redundant with one another. As well, aggregation designs have a costly metadata impact – don't overdesign but try to keep the number of aggregation designs per measure group to a minimum. Inspect the new aggregations compared to the

old and ensure there are no near-duplicates. The aggregation design can be copied to other partitions in SQL Server Management Studio or Business Intelligence Design Studio.

References:

- Reintroducing Usage-Based Optimization in SQL Server 2008 Analysis Services (http://sqlcat.com/sqlcat/b/technicalnotes/archive/2008/11/18/reintroducing-usage-based-optimization-in-sql-server-2008-analysis-services.aspx)
- Analysis Services 2005 Aggregation Design Strategy (http://sqlcat.com/sqlcat/b/technicalnotes/archive/2007/09/11/analysis-services-2005-aggregation-design-strategy.aspx)
- Microsoft SQL Server Community Samples: Analysis Services (http://sqlsrvanalysissrvcs.codeplex.com/): This CodePlex project contains many useful Analysis Services CodePlex samples, including the Aggregation Manager

2.3.4.4.6 Large Cube Aggregation Considerations

It is important to note that small cubes may not need aggregations, because aggregations are not even built for partitions with fewer records than the **IndexBuildThreshold** (which has a default value of 4096). Even if the cube partitions exceed the **IndexBuildThreshold**, aggregations that are correctly designed for smaller cubes may not be the correct ones for large cubes.

However, as cubes become larger, it becomes more important to design aggregations and to do so correctly. As a general rule of thumb, MOLAP performance is approximately between 10 and 40 million rows per second per core, plus the I/O for aggregating data.

It is important to note that larger cubes have more constraints such as small processing windows and/or not enough disk space. Therefore it may be difficult to create all of your desired aggregations. The result is a tradeoff in designing aggregations to be considered more carefully.

2.3.5 Cache Warming

Cache warming can be a last-ditch effort for improving the performance of a query. The following sections describe guidelines and implementation strategies for cache warming.

2.3.5.1 Cache Warming Guidelines

During querying, memory is primarily used to store cached results in the storage engine and query processor caches. To optimize the benefits of caching, you can often increase query responsiveness by preloading data into one or both of these caches. This can be done by either pre-executing one or more queries or using the CREATE CACHE statement (which returns no cellsets and has the advantage of executing faster because it bypasses the query processor). This process is called *cachewarming*.

When possible, Analysis Services returns results from the Analysis Services data cache without using aggregations (because it is the fastest way to get data). With smaller cubes there may be enough memory to keep a large portion of the data in the cache. In this case, aggregations are not needed and

existing aggregations may never be used. In this scenario, cache warming can be used so that users will always have excellent performance.

But with larger cubes, there may be insufficient memory to keep enough of the data in cache. For that matter, cached results can be pushed out by other query results.Hence, cache warming will only help a portion of the queries—it is important to create well-designed aggregations to provide solid query performance. But because of the memory bottlenecks, it is important to note that too many aggregations may thrash the cache as different data resultsets and aggregations are requested and swapped from the cache.

2.3.5.2 Implementing a Cache Warming Strategy

While cache warming can improve the performance of a query, you should note that there is a significant difference between the performance of the query on a cold cache and a warm cache. As well, it is important to ensure there is enough memory available so that the cache is not being thrashed.

To warm the cache, it is important to remember that the Analysis Services formula engine can only be warmed by MDX queries. To warm the storage engine caches, you can use the WITH CACHE or CREATE CACHE statements:

- To discover what needs to be cached (which can be difficult at times), use SQL Server Profiler to trace the query execution and examine the subcube events.
- Finding many subcube requests to the same grain may indicate that the query processor is making many requests for slightly different data, resulting in the storage engine making many small but time-consuming I/O requests where it could more efficiently retrieve the data *en masse* and then return results from cache.
- To pre-execute queries, create an application (or use something like **ascmd**) that executes a set of generalized queries to simulate typical user activity in order to expedite the process of populating the cache. Execute these queries post-Analysis Services startup or post-processing to preload the cache prior to user queries.
  To determine how to generalize your queries, you can potentially refer to the Analysis Services query log to determine the dimension attributes typically queried. Be careful when you generalize because you may include attributes or subcubes that are not beneficial and unnecessarily take up cache.
- When testing the effectiveness of different cache-warming queries, you should empty the query results cache between each test to ensure the validity of your testing.
- Because cached results can be pushed out by other query results, it may be necessary to schedule refreshes of the cache results. Also, limit cache warming to what can fit in memory, leaving enough for other queries to be cached.

References:

- How to warm up the Analysis Services data cache using Create Cache statement? (http://sqlcat.com/sqlcat/b/technicalnotes/archive/2007/09/11/how-to-warm-up-the-analysis-services-data-cache-using-create-cache-statement.aspx)

2.3.6 Scale-Out

If you have many concurrent users querying your Analysis Services cubes, a potential query performance solution is to scale out your Analysis Services query servers. There are different forms of scale-out, which are discussed in Part 2, but the basic principle is that you have multiple query servers aimed at the same database (or the database is replicated) so there are multiple servers to address user queries. This can be beneficial in the cases like the following:

- In cases where your server is under memory pressure due to concurrency, scaling out allows you to distribute the query load to multiple servers, thus alleviating memory bottlenecks on a single server. Memory pressure can be caused by many issues, including (but not limited to):
    - Users executing many different unique queries thus filling up and thrashing available cache.
    - Complex or large queries requiring large subcubes thus requiring a large memory space.
    - Too many concurrent users accessing the same server.
- You have many long running queries against your Analysis Services cube, which will:
    - Block other queries.
    - Block processing commits.

  In this case, scaling out the long-running queries to separate servers can help alleviate contention problems.

References:

- SQL Server 2008 R2 Analysis Services Operations Guide (http://sqlcat.com/sqlcat/b/whitepapers/archive/2011/06/01/sql-server-2008r2-analysis-services-operations-guide.aspx)
- Scale-Out Querying for Analysis Services with Read-Only Databases (http://sqlcat.com/sqlcat/b/whitepapers/archive/2010/06/08/scale-out-querying-for-analysis-services-with-read-only-databases.aspx)
- Scale-Out Querying with Analysis Services (http://sqlcat.com/sqlcat/b/whitepapers/archive/2007/12/16/scale-out-querying-with-analysis-services.aspx)
- Scale-Out Querying with Analysis Services Using SAN Snapshots (http://sqlcat.com/sqlcat/b/whitepapers/archive/2007/11/19/scale-out-querying-with-analysis-services-using-san-snapshots.aspx)

2.4 Tuning Processing Performance

In the following sections we will provide guidance on tuning cube processing. Processing is the operation that loads data from one or more data sources into one or more Analysis Services objects. Although OLAP systems are not generally judged by how fast they process data, processing performance impacts how quickly new data is available for querying. Every application has different data refresh requirements, ranging from monthly updates to near real-time data refreshes; however, in all cases, the faster the processing performance, the sooner users can query refreshed data.

Analysis Services provides several processing commands, allowing granular control over the data loading and refresh frequency of cubes.

To manage processing operations, Analysis Services uses centrally controlled jobs. A processing job is a generic unit of work generated by a processing request.

From an architectural perspective, a job can be broken down into parent jobs and child jobs. For a given object, you can have multiple levels of nested jobs depending on where the object is located in the OLAP database hierarchy. The number and type of parent and child jobs depend on 1) the object that you are processing, such as a dimension, cube, measure group, or partition, and 2) the processing operation that you are requesting, such as **ProcessFull**, **ProcessUpdate**, or **ProcessIndexes**.

For example, when you issue a **ProcessFull** operation for a measure group, a parent job is created for the measure group with child jobs created for each partition. For each partition, a series of child jobs are spawned to carry out the **ProcessFull** operation of the fact data and aggregations. In addition, Analysis Services implements dependencies between jobs. For example, cube jobs are dependent on dimension jobs.

The most significant opportunities to tune performance involve the processing jobs for the core processing objects: dimensions and partitions. Each of these is covered in the sections that follow.

References:

- Additional background information on processing can be found in the technical note Analysis Services 2005 Processing Architecture (http://msdn.microsoft.com/en-us/library/ms345142(SQL.90).aspx).

2.4.1 Baselining Processing

To quantify the effects of your tuning and diagnose problems, you should first create a baseline. The baseline allows you to analyze root causes and to target optimization effort.

This section describes how to set up the baseline.

2.4.1.1 Performance Monitor Trace

Windows performance counters are the bread and butter of performance tuning Analysis Services. Use the tool **perfmon** to set up a trace with these counters:

- **MSOLAP: Processing**
  - **Rows read/sec**
- **MSOLAP: Proc Aggregations**
  - **Temp File Bytes Writes/sec**
  - **Rows created/Sec**
  - **Current Partitions**
- **MSOLAP: Threads**
  - **Processing pool idle threads**
  - **Processing pool job queue length**
  - **Processing pool busy threads**
- **MSSQL: Memory Manager**
  - **Total Server Memory**
  - **Target Server Memory**
- **Process**
  - **Virtual Bytes – msmdsrv.exe**
  - **Working Set – msmdsrv.exe**
  - **Private Bytes – msmdsrv.exe**
  - **% Processor Time – msmdsrv.exe and sqlservr.exe**
- **MSOLAP: Memory**
  - **Quote Blocked**
- **Logical Disk:**
  - **Avg. Disk sec/Transfer – All Instances**
- **Processor:**
  - **% Processor Time – Total**
- **System:**
  - **Context Switches / sec**

Configure the trace to save data to a file. Measuring every 15 seconds will be sufficient for tuning processing.

As you tune processing, you should measure these counters again after each change to see whether you are getting closer to your performance goal. Also note the total time used by processing. The following sections explain how to use and interpret the individual counters.

2.4.1.2 Profiler Trace

To optimize the SQL queries that form part of processing, you should trace the relational database too. If the relational database is SQL Server, you use SQL Server Profiler for this. If you are not using SQL Server, consult your database vendor or DBA for help on tuning the database. In the following we will assume that you use SQL Server as the relational foundation for Analysis Services. For users of other databases, the knowledge here will most likely transfer cleanly to your platform.

In your SQL Server Profiler trace you should also capture the events:

- **Performance/Showplan XML Statistics Profile**
- **TSQL/SQL:BatchCompleted**

Include these event columns:

- **TextData**
- **Reads**
- **DatabaseName**
- **SPID**
- **Duration**

You can use the **Tuning** template and just add the **Reads** column and **Showplan XML Statistics Profiles**. Like the **perfmon** trace, configure the trace to save to a file for later analysis.

Configure your SQL Server Profiler trace to log to a table instead of a file. This makes it easier to correlate the traces later.

The performance data gathered by these traces will be used in the following section to help you tune processing.

2.4.1.3 Determining Where You Spend Processing Time

To properly target the tuning of processing, you should first determine where you are spending your time: partition processing or dimension processing.

To assist with tuning and future monitoring, it is useful to split the dimension processing and partition processing into two different commands in the processing, to tune each individually.

For partition processing, you should distinguish between **ProcessData** and **ProcessIndex**—the tuning techniques for each are very different. If you follow our recommended best practice of doing **ProcessData** followed by **ProcessIndex** instead of **ProcessFull**, the time spent in each should be easy to read.

If you use **ProcessFull** instead of splitting into **ProcessData** and **ProcessIndex**, you can get an idea of when each phase ends by observing the following **perfmon** counters:

- During **ProcessData** the counter **MSOLAP:Processing – Rows read/Sec** is greater than zero.
- During **ProcessIndex** the counter **MSOLAP:Proc Aggregations – Row created/Sec** is greater than zero.

ProcessData can be further split into the time spent by the SQL Server process and the time spent by the Analysis Services process. You can use the **Process** counters collected to see where most of the CPU time is spent. The following diagram provides an overview of the operations included in a full cube processing.

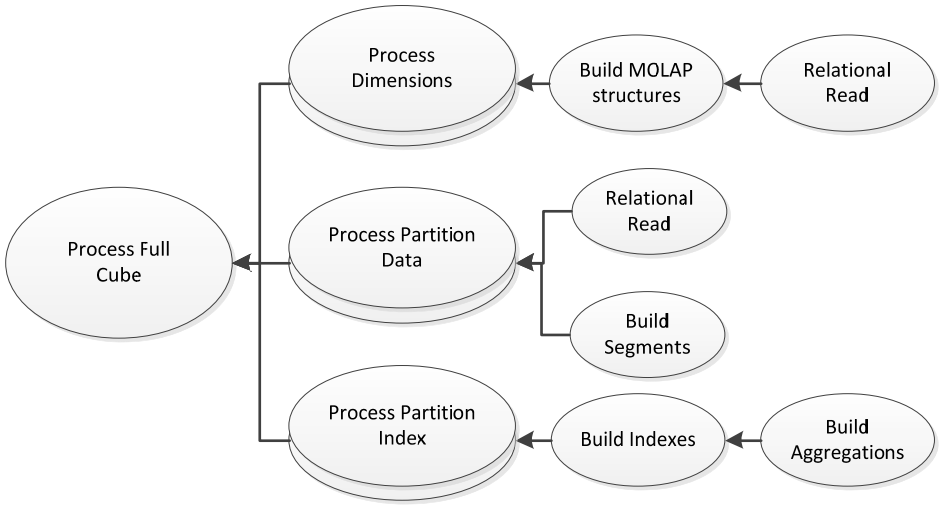


Figure 3030: Full cube processing overview

2.4.2 Tuning Dimension Processing

The performance goal of dimension processing is to refresh dimension data in an efficient manner that does not negatively impact the query performance of dependent partitions. The following techniques for accomplishing this goal are discussed in this section:

- Reducing attribute overhead.
- Optimizing SQL source queries.

To provide a mental model of the workload, we will first introduce the dimension processing architecture.

2.4.2.1 Dimension Processing Architecture

During the processing of MOLAP dimensions, jobs are used to extract, index, and persist data in a series of dimension stores.

To create these dimension stores, the storage engine uses the series of jobs displayed in the following diagram.

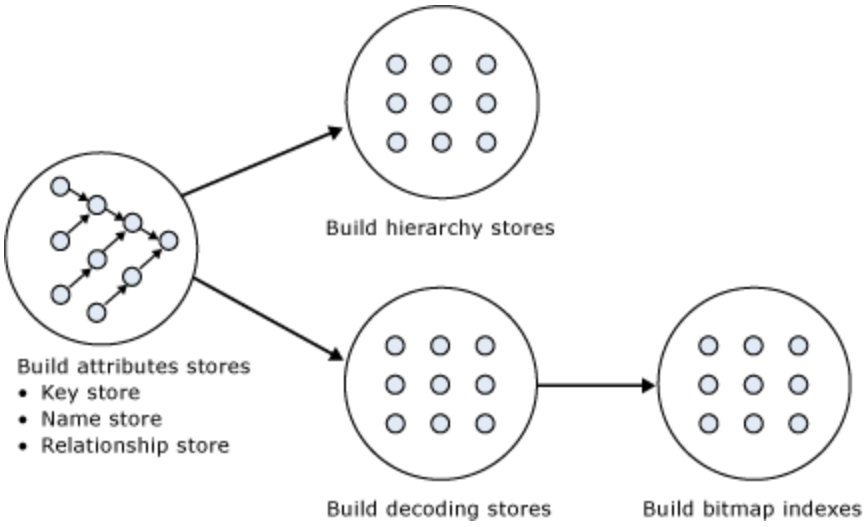


Figure 3131: Dimension processing jobs

Build Attribute Stores - For each attribute in a dimension, a job is instantiated to extract and persist the attribute members into an attribute store. The attribute store consists of the key store, name store, and relationship store. The data structures build during dimension processing are saved to disk with the following extensions:

- Hierarchy stores: ***.ostore**, ***.sstore** and ***.lstore**
- Key store: ***.kstore**, ***.khstore** and ***.ksstore**
- Name Store: ***.asstore**, ***.ahstore** and ***.hstore**
- Relationship store: ***.data** and ***.data.hdr**
- Decoding Stores: ***.dstore**
- Bitmap indexes: ***.map** and ***.map.hdr**

Because the relationship stores contain information about dependent attributes, an ordering of the processing jobs is required. To provide the correct workflow, the storage engine analyzes the dependencies between attributes, and then it creates an execution tree with the correct ordering. The execution tree is then used to determine the best parallel execution of the dimension processing.

Figure 32 displays an example execution tree for a Time dimension. The solid arrows represent the attribute relationships in the dimension. The dashed arrows represent the implicit relationship of each attribute to the All attribute.

Note: The dimension has been configured using cascading attribute relationships, which is a best practice for all dimension designs.

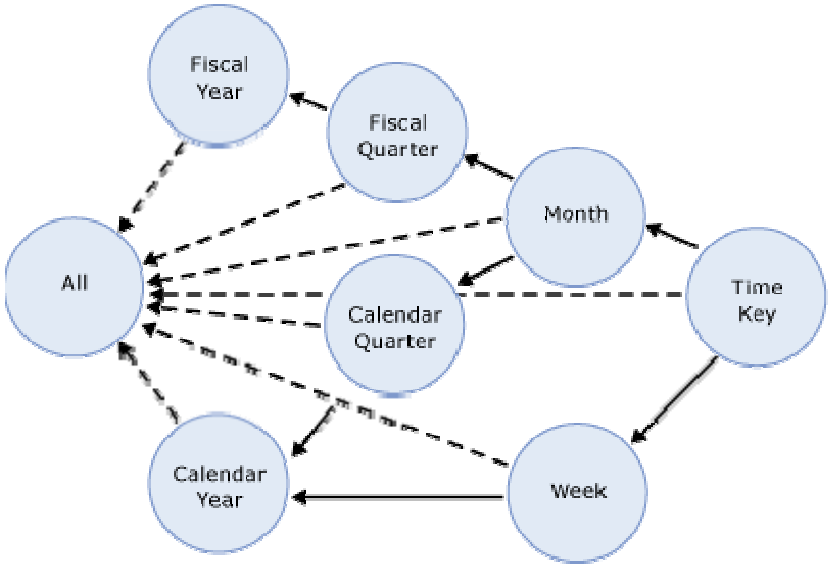


Figure 3232: Execution tree example

In this example, the **All** attribute proceeds first, given that it has no dependencies to another attribute, followed by the **Fiscal Year** and **Calendar Year** attributes, which can be processed in parallel. The other attributes proceed according to the dependencies in the execution tree, with the key attribute always being processed last, because it always has at least one attribute relationship, except when it is the only attribute in the dimension.

The time taken to process an attribute is generally dependent on 1) the number of members and 2) the number of attribute relationships. While you cannot control the number of members for a given attribute, you can improve processing performance by using cascading attribute relationships. This is especially critical for the key attribute, because it has the most members and all other jobs (hierarchy, decoding, bitmap indexes) are waiting for it to complete. Attribute relationships lower the memory requirement during processing. When an attribute is processed, all dependent attributes must be kept in memory. If you have no attribute relationships, all attributes must kept in memory while the key attribute is processed. This may cause out-of-memory conditions.

Build Decoding Stores - Decoding stores are used extensively by the storage engine. During querying, they are used to retrieve data from the dimension. During processing, they are used to build the dimension's bitmap indexes.

Build Hierarchy Stores - A *hierarchy store* is a persistent representation of the tree structure. For each natural hierarchy in the dimension, a job is instantiated to create the hierarchy stores.

Build Bitmap Indexes - To efficiently locate attribute data in the relationship store at querying time, the storage engine creates bitmap indexes at processing time. For attributes with a very large number of members, the bitmap indexes can take some time to process. In most scenarios, the bitmap indexes provide significant querying benefits; however, when you have high-cardinality attributes, the querying

benefit that the bitmap index provides may not outweigh the processing cost of creating the bitmap index.

2.4.2.2 Dimension-Processing Commands

When you need to perform a process operation on a dimension, you issue dimension processing commands. Each processing command creates one or more jobs to perform the necessary operations.

From a performance perspective, the following dimension processing commands are the most important:

- **ProcessData**
- **ProcessFull**
- **ProcessUpdate**
- **ProcessAdd**

The **ProcessFull** and **ProcessData** commands discard all storage contents of the dimension and rebuild them. Behind the scenes, **ProcessFull** executes all dimension processing jobs and performs an implicit **ProcessClear** on all dependent partitions. This means that whenever you perform a **ProcessFull** operation of a dimension, you need to perform a **ProcessFull** operation on dependent partitions to bring the cube back online. **ProcessFull** also builds indexes on the dimension data itself (note that indexes on the partitions are built separately). If you do **ProcessData** on a dimension, you should do **ProcessIndexes** subsequently so that dimension queries are able to use these indexes.

Unlike **ProcessFull**, **ProcessUpdate** does not discard the dimension storage contents. Instead, it applies updates intelligently in order to preserve dependent partitions. More specifically, **ProcessUpdate** sends SQL queries to read the entire dimension table and then applies changes to the dimension stores.

ProcessAdd optimizes **ProcessUpdate** in scenarios where you only need to insert new members. **ProcessAdd** does not delete or update existing members. The performance benefit of **ProcessAdd** is that you can use a different source table or data source view named query that restrict the rows of the source dimension table to only return the new rows. This eliminates the need to read all of the source data. In addition, **ProcessAdd** also retains all indexes and aggregations (flexible and rigid).

ProcessUpdate and **ProcessAdd** have some special behaviors that you should be aware of. These behaviors are discussed in the following sections.

2.4.2.2.1 ProcessUpdate

A **ProcessUpdate** can handle inserts, updates, and deletions, depending on the type of attribute relationships (rigid versus flexible) in the dimension. Note that **ProcessUpdate** drops invalid aggregations and indexes, requiring you to take action to rebuild the aggregations in order to maintain query performance. However, flexible aggregations are dropped only if a change is detected.

When **ProcessUpdate** runs, it must walk through the partitions that depend on the dimension. For each partition, all indexes and aggregation must be checked to see whether they require updating. On a cube

with many partitions, indexes, and aggregates, this can take a very long time. Because this dependency walk is expensive, **ProcessUpdate** is often the most expensive of all processing operations on a well-tuned system, dwarfing even large partition processing commands.

2.4.2.2.2 ProcessAdd

Note that **ProcessAdd** is only available as an XMLA command and not from SQL Server Management Studio. **ProcessAdd** is the preferred way of managing Type 2 changing dimensions. Because Analysis Services knows that existing indexes do not need to be checked for invalidation, **ProcessAdd** typically runs much faster than **ProcessUpdate**.

In the default configuration of Analysis Services, **ProcessAdd** typically triggers a processing error when run, reporting duplicate key values. This is caused by the "addition" of non-key properties that already exist in the dimension. For example, consider the addition of a new customer to a dimension. If the customer lives in a country that is already present in the dimension, this country cannot be added (it is already there) and Analysis Services throws an error. The solution in this case is to set the <**KeyDuplicate>** to **IgnoreError** on the dimension processing command.

Note that you cannot run a **ProcessAdd** on an empty dimension. The dimension must first be fully processed.

References:

- For detailed information about automating **ProcessAdd,** see Greg Galloway's blog entry: http://www.artisconsulting.com/blogs/greggalloway/Lists/Posts/Post.aspx?ID=4
- For information about how to avoid set the KeyDuplicate, see this forum thread: http://social.msdn.microsoft.com/Forums/en-US/sqlanalysisservices/thread/8e7f1304-56a1-467e-9cc6-68428bd92aa6?prof=required

2.4.3 Tuning Cube Dimension Processing

In section 2, we described how to create a good and high-performance dimension design. In SQL Server 2008 and SQL Server 2008 R2 Analysis Services, the Analysis Management Objects (AMO) warnings are provided by Business Intelligence Development Studio to assist you with following these best practices.

When it comes to dimension processing, you must pay a price for having many attributes. If the processing time for the dimension is restrictive, you most likely have to change the attribute to design in order to improve performance.

2.4.3.1 Reduce Attribute Overhead

Every attribute that you include in a dimension impacts the cube size, the dimension size, the aggregation design, and processing performance. Whenever you identify an attribute that will not be used by end users, delete the attribute entirely from your dimension. After you have removed extraneous attributes, you can apply a series of techniques to optimize the processing of remaining attributes.

2.4.3.1.1 Remove Bitmap Indexes

During processing of the primary key attribute, bitmap indexes are created for every related attribute. Building the bitmap indexes for the primary key can take time if it has one or more related attributes with high cardinality. At query time, the bitmap indexes for these attributes are not useful in speeding up retrieval, because the storage engine still must sift through a large number of distinct values. This may have a negative impact on query response times.

For example, the primary key of the customer dimension uniquely identifies each customer by account number; however, users also want to slice and dice data by the customer's social security number. Each customer account number has a one-to-one relationship with a customer social security number. You can consider removing the creation of bitmaps for the social security number.

You can also consider removing bitmap indexes from attributes that are always queried together with other attributes that already have bitmap indexes that are highly selective. If the other attributes have sufficient selectivity, adding another bitmap index to filter the segments will not yield a great benefit.

For example, you are creating a sales fact and users always query both date and store dimensions. Sometimes a filter is also applied by the store clerk dimension, but because you have already filtered down to stores, adding a bitmap on the store clerk may only yield a trivial benefit. In this case, you can consider disabling bitmap indexes on store clerk attributes.

You can disable the creation of bitmap indexes for an attribute by setting the **AttributeHierarchyOptimizedState** property to **Not Optimized**.

2.4.3.1.2 Optimize Attribute Processing Across Multiple Data Sources

When a dimension comes from multiple data sources, using cascading attribute relationships allows the system to segment attributes during processing according to data source. If an attribute's key, name, and attribute relationships come from the same database, the system can optimize the SQL query for that attribute by querying only one database. Without cascading attribute relationships, the SQL Server OPENROWSET function, which provides a method for accessing data from multiple sources, is used to merge the data streams. In this situation, the processing for the attribute is extremely slow, because it must access multiple OPENROWSET derived tables.

If you have the option, consider performing ETL to bring all data needed for the dimension into the same SQL Server database. This allows you to utilize the Relational Engine to tune the query.

2.4.3.2 Tuning the Relational Dimension Processing Queries

Unlike fact partitions, which only send one query to the server per partition, dimension process operations send multiple queries. Dimensions tend to be small, complex tables with very few changes, compared to facts that are typically simpler tables, but with many changes. Tables that have the characteristics of dimensions can often be heavily indexed with little insert/update performance overhead to the system. You can use this to your advantage during processing and be wasteful with the relational indexes.

To quickly tune the relational queries used for dimension processing you can use the Database Engine Tuning Advisor on a profiler trace of the dimension processing. For the small dimension tables, chances are that you can get away with adding every suggested index. For the larger tables, target the indexes towards the longest-running queries. For detailed tuning advice on large dimension tables, see Part 2.

2.4.3.2.1 Using ByTable Processing

By setting the **ProcessingGroup** property of the dimension to be **ByTable** you will change howAnalysis Services behaves during dimension processing. Instead of sending multiple SELECT DISTINCT queries, the processing task instead requests the entire table with one query. If you have enough memory to hold all the new dimension data while processing is happening, this option can provide a fast way to optimize processing. However, you should be careful about this setting – if Analysis Services runs out of memory during processing, this will have a large impact on both query and processing performance. Experiment with this setting carefully before putting it into production.

Note also that **ByTable** processing will cause duplicate key (KeyDuplicate) errors because SELECT DISTINCT is not executed for each attribute, and the same members will be encountered repeatedly during processing. Therefore, you will need to specify a custom error configuration and disable the KeyDuplicate errors.

2.4.4 Tuning Partition Processing

The performance goal of partition processing is to refresh fact data and aggregations in an efficient manner that satisfies your overall data refresh requirements.

The following techniques for accomplishing this goal are discussed in this section: optimizing SQL source queries and using a partitioning strategy (both in the cube and the relational database) to optimize processing. For detailed guidance on server tuning, hardware optimization and relational indexing, see Part 2.

2.4.4.1 Partition Processing Architecture

During partition processing, source data is extracted and stored on disk using the series of jobs displayed In Figure 33.

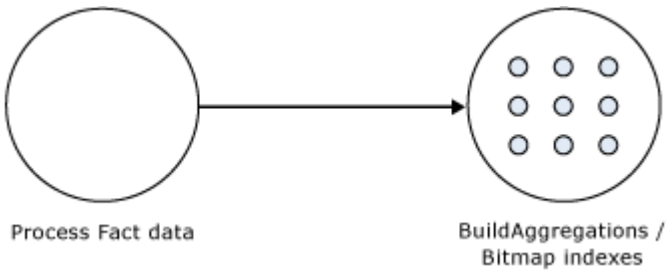


Figure 3333: Partition processing jobs

Process Fact Data - Fact data is processed using three concurrent threads that perform the following tasks:

- Send SQL statements to extract data from data sources.
- Look up dimension keys in dimension stores and populate the processing buffer.
- When the processing buffer is full, write out the buffer to disk.

Build Aggregations and Bitmap Indexes - Aggregations are built in memory during processing. Although too few aggregations may have little impact on query performance, excessive aggregations can increase processing time without much added value on query performance.

If aggregations do not fit in memory, chunks are written to temp files and merged at the end of the process. Bitmap indexes are also built during this phase and written to disk on a segment-by-segment basis.

2.4.4.2 Partition-Processing Commands

When you need to perform a process operation on a partition, you issue partition processing commands. Each processing command creates one or more jobs to perform the necessary operations.

The following partition processing commands are available:

- **ProcessFull**
- **ProcessData**
- **ProcessIndexes**
- **ProcessAdd**
- **ProcessClear**
- **ProcessClearIndexes**

ProcessFull discards the storage contents of the partition and then rebuilds them. Behind the scenes, **ProcessFull** executes **ProcessData** and **ProcessIndexes** jobs.

ProcessData discards the storage contents of the object and rebuilds only the fact data.

ProcessIndexes requires a partition to have built its data already. **ProcessIndexes** preserves the data created during **ProcessData** and creates new aggregations and bitmap indexes based on it.

ProcessAdd internally creates a temporary partition, processes it with the target fact data, and then merges it with the existing partition. Note that **ProcessAdd** is the name of the XMLA command, in Business Intelligence Development Studio and SQL Server Management Studio this is exposed as **ProcessIncremental.**

ProcessClear removes all data from the partition.Note the **ProcessClear** is the name of the XMLA command. In Business Intelligence Development Studio and SQL Server Management Studio, it is exposed as **UnProcess.**

ProcessClearIndexes removes all indexes and aggregates from the partition. This brings the partitions in the same state as if **ProcessClear** followed by **ProcessData** had just been run.Note that

ProcessClearIndexes is the name of the XMLA command. This command is not available in Business Intelligence Development Studio and SQL Server Management Studio.

2.4.4.3 Partition Processing Performance Best Practices

When designing your fact tables, use the guidance in the following technical notes:

- Top 10 Best Practices for Building a Large Scale Relational Data Warehouse (http://sqlcat.com/sqlcat/b/top10lists/archive/2008/02/06/top-10-best-practices-for-building-a-large-scale-relational-data-warehouse.aspx)
- Analysis Services Processing Best Practices (http://sqlcat.com/sqlcat/b/whitepapers/archive/2007/11/15/analysis-services-processing-best-practices.aspx)

2.4.4.4 Optimizing Data Inserts, Updates, and Deletes

This section provides guidance on how to efficiently refresh partition data to handle inserts, updates, and deletes.

2.4.4.4.1 Inserts

If you have a browsable, processed cube and you need to add new data to an existing measure group partition, you can apply one of the following techniques:

- **ProcessFull**—Perform a **ProcessFull** operation for the existing partition. During the **ProcessFull** operation, the cube remains available for browsing with the existing data while a separate set of data files are created to contain the new data. When the processing is complete, the new partition data is available for browsing. Note that **ProcessFull** is technically not necessary, given that you are only doing inserts. To optimize processing for insert operations, you can use **ProcessAdd**.
- **ProcessAdd**—Use this operation to append data to the existing partition files. If you frequently perform **ProcessAdd**, we recommend that you periodically perform **ProcessFull** in order to rebuild and recompress the partition data files. **ProcessAdd** internally creates a temporary partition and merges it. This results in data fragmentation over time and the need to periodically perform **ProcessFull**.

If your measure group contains multiple partitions, as described in the previous section, a more effective approach is to create a new partition that contains the new data and then perform **ProcessFull** on that partition. This technique allows you to add new data without impacting the existing partitions. When the new partition has completed processing, it is available for querying.

2.4.4.4.2 Updates

When you need to perform data updates, you can perform a **ProcessFull**. Of course it is useful if you can target the updates to a specific partition so you only have to process a single partition. Rather than directly updating fact data, a better practice is to use a *journaling* mechanism to implement data changes. In this scenario, you turn an update into an insertion that corrects that existing data. With this

approach, you can simply continue to add new data to the partition by using a **ProcessAdd**. By using journaling, you also have an audit trail of the changes that have been made to the fact table.

2.4.4.4.3 Deletes

For deletions, multiple partitions provide a great mechanism for you to roll out expired data. Consider the following example. You currently have 13 months of data in a measure group, 1 month per partition. You want to roll out the oldest month from the cube. To do this, you can simply delete the partition without affecting any of the other partitions.

If there are any old dimension members that only appeared in the expired month, you can remove these using a **ProcessUpdate** operation on the dimension (but only if it contains flexible relationships). In order to delete members from the key/granularity attribute of a dimension, you must set the dimension's **UnknownMember** property to **Hidden**. This is because the server does not know if there is a fact record assigned to the deleted member. With this property set appropriately, the member will be hidden at query time. Another option is to remove the data from the underlying table and perform a **ProcessFull** operation. However, this may take longer than **ProcessUpdate**.

As your dimension grows larger, you may want to perform a **ProcessFull** operation on the dimension to completely remove deleted keys. However, if you do this, all related partitions must also be reprocessed. This may require a large batch window and is not viable for all scenarios.

2.4.4.5 Picking Efficient Data Types in Fact Tables

During processing, data has to be moved out of SQL Server and into Analysis Services. The wider your rows are, the more bandwidth must be spent moving the rows.

Some data types are, by the nature of their design, faster to use than others. For fastest performance, consider using only these data types in fact tables.

Fact column type	Fastest SQL Server data types
Surrogate keys	**tinyint**, **smallint**, **int**, **bigint**
Date key	**int** in the format yyyyMMdd
Integer measures	**tinyint**, **smallint**, **int**, **bigint**
Numeric measures	**smallmoney**, **money**, **real**, **float** *(*Note that **decimal** and **vardecimal** require more CPU power to process than **money** and **float** types)
Distinct count columns	**tinyint**, **smallint**, **int**, **bigint** *(*If your count column is **char**, consider either hashing or replacing with surrogate key*)*

2.4.4.6 Tuning the Relational Partition Processing Query

During the **ProcessData** phase, rows are read from a relational source and into Analysis Services. Analysis Services can consume rows at a very high rate during this phase. To achieve these high speeds, you need to tune the relational database to provide a proper throughput.

In the following subsection, we assume that your relational source is SQL Server. If you are using another relational source, some of the advice still applies – consult your database specialist for platform specific guidance.

Analysis Services uses the partition information to generate the query. Unless you have done any query binding in the UDM, the SELECT statement issues to the relational source is very simple. It consists of:

- A SELECT of the columns required to process. This will be the dimension columns and the measures.
- Optionally, a WHERE criterion if you use partitions. You can control this WHERE criterion by changing the query binding of the partition.

2.4.4.6.1 Getting Rid of Joins

If you are using a database view or a UDM named query as the basis of partitions, you should seek to eliminate joins in the query send to the database. You can achieve this by denormalizing the joined columns to the fact table. If you are using a star schema design, you should already have done this.

References:

- For background on relational star schemas and how to design and denormalize for optimal performance, refer to: Ralph Kimball, *The Data Warehouse Toolkit.*

2.4.4.6.2 Getting Relational Partitioning Right

If you use partitioning on the relational side, you should ensure that each cube partition touches at most one relational partition. To check this, use the **XML Showplan** event from your SQL Server Profiler trace.

If you got rid of all joins, your query plan should look something like the following figure.

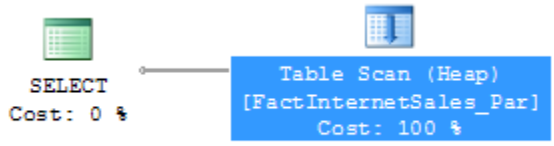


Figure 3434: An optimal partition processing query

Click on the table scan (it may also be a range scan or index seek in your case) and bring up the properties pane.

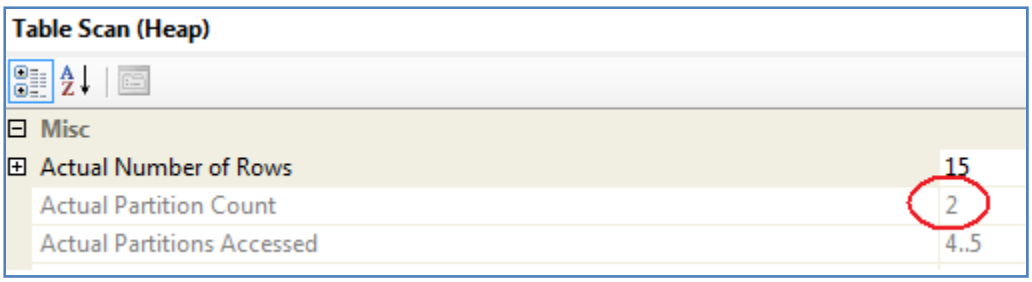


Figure 3535: Too many partitions accessed

Both partition 4 and partition 5 are accessed. The value for **Actual Partition Count** should be 1. If this is not the case (as in the figure), you should consider repartitioning the relational source data so that each cube partition touches at most one relational partition.

2.4.4.7 Splitting Processing Index and Process Data

It is good practice to split partition processing into its components: **ProcessData** and **ProcessIndex**. This has several advantages.

First, it allows you to restart a failed processing at the last valid state. For example, if you fail processing during **ProcessIndex**, you can restart this phase instead of reverting to running **ProcessData** again.

Second, **ProcessData** and **ProcessIndex** have different performance characteristics. Typically, you want to have more parallel commands executing during **ProcessData** than you want during **ProcessIndex**. By splitting them into two different commands, you can override parallelism on the individual commands.

Of course, if you don't want to micromanage partition processing, you may just opt for running a **ProcessFull** on the measure group. For small cubes where performance is not a concern, this will work well.

2.4.4.8 Increasing Concurrency by Adding More Partitions

If your tuning is bound only by the amount of CPU power you have (as opposed to I/O, for example), you should optimize to make the best use of the CPU cores available to you. It is time to have a look at the **Processor:Total** counter from the baseline trace. If this counter is not 100%, you are not taking full advantage of your CPU power. As you continue the tuning, keep comparing the baselines to measure improvement, and watch out for bottlenecks to appear again as you push more data through the system.

Using multiple partitions can enhance processing performance. Partitions allow you to work on many, smaller parts of the fact table in parallel. Because a single connection to SQL Server can only transfer a limited amount of rows per second, adding more partitions, and hence, more connections, can increase throughput. How many partitions you can process in parallel depends on your CPU and machine architecture. As a rule of thumb, keep increasing parallelism until you no longer see an increase in **MSOLAP:Processing – Rows read/Sec**. You can measure the amount of concurrent partitions you are processing by looking at the perfmon counter **MSOLAP: Proc Aggregations - Current Partitions**.

Being able to process multiple partitions in parallel is useful in a variety of scenarios; however, there are a few guidelines that you must follow. Keep in mind that whenever you process a measure group that has no processed partitions, Analysis Services must initialize the cube structure for that measure group. To do this, it takes an exclusive lock that prevents parallel processing of partitions. You should eliminate this lock before you start the full parallel process on the system. To remove the initialization lock, ensure that you have at least one processed partition per measure group before you begin the parallel operation. If you do not have a processed partition, you can perform a **ProcessStructure** on the cube to build its initial structure and then proceed to process measure group partitions in parallel. You will not encounter this limitation if you process partitions in the same client session and use the **MaxParallel** XMLA element to control the level of parallelism.

2.4.4.9 Adjusting Maximum Number of Connections

When you increase parallelism of the processing above 10 concurrent partitions, you will need to adjust the maximum number of connections that Analysis Services keeps open on the database. This number can be changed in the properties of the data source (the **Maximum number of connections** box).

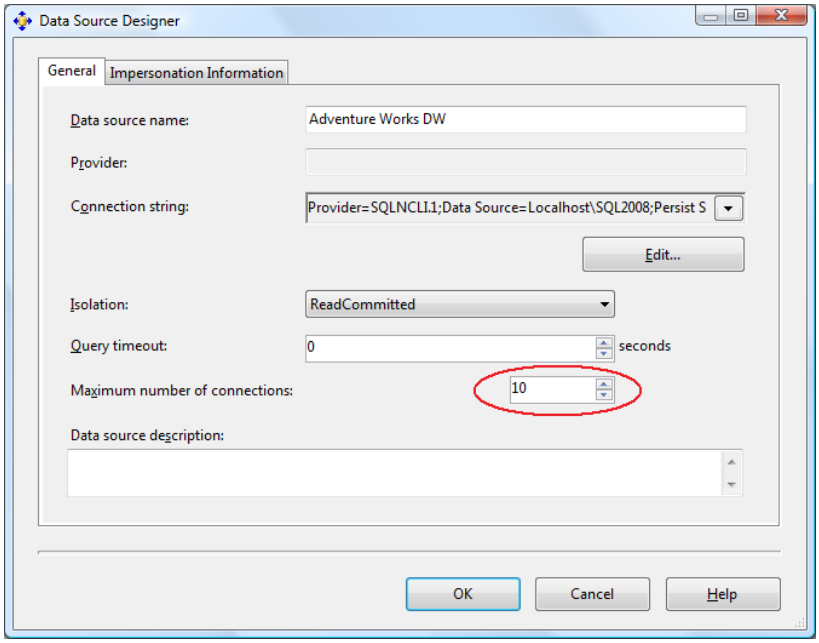


Figure 3636: Adding more database connections

Set this number to at least the number of partitions you want to process in parallel.

2.4.4.10 Tuning the Process Index Phase

During the **ProcessIndex** phase the aggregations in the cube are built. At this point, no more activity happens in the Relational Engine, and if Analysis Services and the Relational Engine are sharing the same box, you can dedicate all your CPU cores to Analysis Services.

The key figure you optimize during **ProcessIndex** is the performance counter **MSOLAP:Proc Aggregations – Row created/Sec.** As the counter increases, the **ProcessIndex** time decreases. You can use this counter to check if your tuning efforts improve the speed.

An additional important counter to look at is the temporary files counter – when an aggregation doesn't fit in memory, the aggregation data is spilled to temporary disk files. Building disk based aggregations is much more expensive, and if you notice this you may be able to find a way to either allow more memory to be available for the index building phase, or drop some of the larger aggregations to avoid this issue.

2.4.4.10.1 Add Partitions to Increase Parallelism

As was the case with **ProcessData**, processing more partitions in parallel can speed up **ProcessIndex**. The same tuning strategy applies: Keep increasing partition count until you no longer see an increase in processing speed.

2.4.4.11 Partitioning the Relational Source

The best partition strategy to implement in the relational source varies by database product capabilities, but some general guidance applies.

It is often a good idea to reflect the cube partition strategy in the relation design. Partitions in the relational source serve as "coarse indexes," and matching relational partitions with the cube allows you to get the best possible table scan speeds by touching only the records you need. Another way to achieve that effect is to use a SQL Server clustered index (or the equivalent in your preferred database engine) to support fast scan queries during partition processing. If you have used a matrix partition schema as described earlier, you may even want to combine the partition and cluster index strategy, using partitioning to support one of the partitioned dimension and cluster indexes to support the other.

The following figures illustrate some examples of partition strategies you should consider.

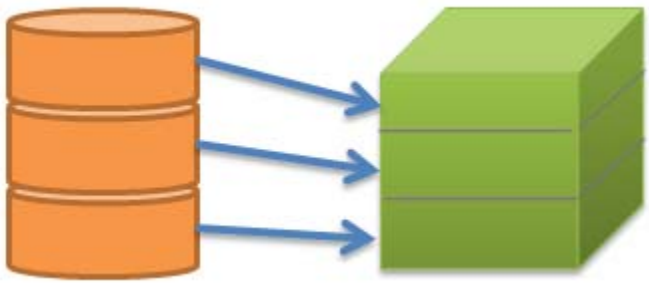


Figure 3737: Matching partition strategies

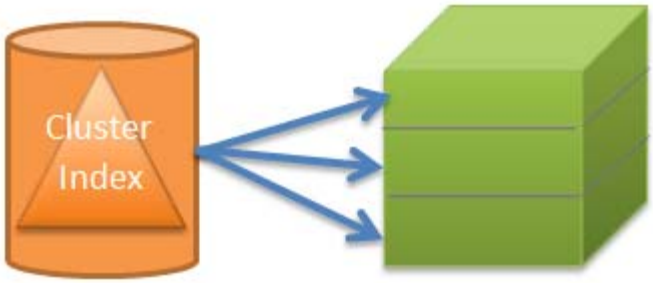


Figure 3838: Clustering the relational table

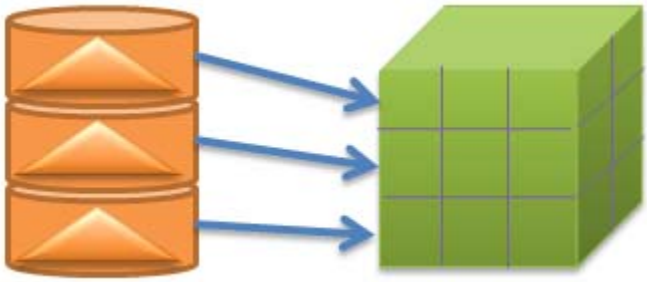


Figure 3939: Supporting matrix partitioning with a combination of relational layouts

2.5 Special Considerations

There are certain features of Analysis Services that provide a lot of business intelligence value, but that require special attention to succeed. This section describes these scenarios and the tuning you can apply when you encounter them.

2.5.1 Distinct Count

Distinct count measures are architecturally very different from other Analysis Services measures because they are not additive in nature. This means that more data must be kept on disk and in general, most distinct count queries have a heavy impact on the storage engine.

2.5.1.1 Partition Design

When distinct count partitions are queried, each partition's segment jobs must coordinate with one another to avoid counting duplicates. For example, if counting distinct customers with customer ID and the same customer ID is in multiple partitions, the partitions' jobs must recognize the match so that they do not count the same customer more than once.

If each partition contains nonoverlapping range of values, this coordination between jobs is avoided and query performance can improve by orders of magnitude, depending on hardware! As well, there are a number of additional optimizations to help improve distinct count performance:

- The key to improving distinct count query performance is to have a partitioning strategy that involves a time period and your *distinct count* value. Start by partitioning by time and x number of distinct value partitions of equal size with non-overlapping ranges, where x is the number of cores. Refine x by testing with different partitioning schemes.
- To distribute your *distinct value* across your partitions with non-overlapping ranges, considering building a *hash of the distinct value*. A modulo function is simple and straightforward but it requires extra processing (for example, convert character key to integer values) and storage (for example, to maintain an IDENTITY table). A hash function such as the SQL **HashBytes** function will avoid the latter issues but may introduce hash key collisions (that is, when the hash value is repeated based on different source values).
- The distinct count measure must be directly contained in the query. If you partition your cube by the *hash of the distinct value*, it is important that your query is against the *hash of the distinct value* (versus the distinct value itself). Even if the *distinct value* and the *hash of the distinct value*have the same distribution of data, and even if you partition data by the latter, the header files contain only the range of values associated with the *hash of the distinct value*. This ultimately means that the Analysis Services storage engine must query all of the partitions to perform the distinct on the *distinct value*.
- The distinct count values need to be *continuous*.

For more information, see Analysis Services Distinct Count Optimization Using Solid State Devices (http://sqlcat.com/sqlcat/b/technicalnotes/archive/2010/09/20/analysis-services-distinct-count-optimization-using-solid-state-devices.aspx).

2.5.1.2 Processing of Distinct Count

Distinct count measure groups have special requirements for partitioning. Normally, you use time and potentially some other dimension as the partitioning column (see the section on Partitioning earlier in this book). However, if you partition a distinct count measure group, you should partition on the value of the distinct count *measure* column instead of a dimension.

Group the distinct count measure column into separate, nonoverlapping intervals. Each interval should contain approximately the same amount of rows from the source. These intervals then form the source of your Analysis Services partitions.

Because the parallelism of the Process Data phase is limited by the amount of partitions you have, for optimal processing performance, split the distinct count measure into as many equal-sized nonoverlapping intervals as you have CPU cores on the Analysis Services computer.

Starting with SQL Server 2005 Analysis Services, it is possible to use noninteger columns for distinct count measure groups. However, for performance reasons (and the potential to hit the 4-GB limit) you should avoid this. The white paper Analysis Services Distinct Count Optimization (http://sqlcat.com/sqlcat/b/technicalnotes/archive/2010/09/20/analysis-services-distinct-count-optimization-using-solid-state-devices.aspx) describes how you can use hash functions to transform

noninteger columns into integers for distinct count. It also provides examples of the nonoverlapping interval-partitioning strategy.

You should also investigate the possibility of optimizing the relational database for the particular SQL queries that are generated during processing of distinct count partitions. The processing query will send an ORDER BY clause in the SQL, and there may be techniques that you can follow to build indexes in the relational database that will produce better performance for this query pattern.

2.5.1.3 Distinct Count Partition Aggregation Considerations

Aggregations created for distinct count partitions are different because distinct count values cannot be naturally aggregated at different levels . Analysis Services creates aggregations at the different granularities by also including the value that needs to be counted. If you think of an aggregation as a GROUP BY on the aggregation granularities, a distinct count aggregation is a GROUP BY on the aggregation granularities and the distinct count column. Having the distinct count column in the aggregation data allows the aggregation to be used when querying a higher granularity—but unfortunately it also makes the aggregations much larger.

To get the most value out of aggregations for distinct count partitions, design aggregations at a commonly viewed higher level attribute related to the distinct count attribute. For example, a report about customers is typically viewed at the Customer Group level; hence, build aggregations at that level. A common approach is run the typical queries against your distinct count partition and use usage-based optimization to build the appropriate aggregations.

2.5.1.4 Optimize the Disk Subsystem for Random I/O

As noted in the beginning of this section, distinct count queries have a heavy impact on the Analysis Services storage engine, which for large cubes means there is a large impact on the disk subsystem. For each query, Analysis Services generates potentially multiple processes—each one parsing the disk subsystem to perform a portion of the distinct count calculation. This results in heavy random I/O on the disk subsystem, which can significantly reduce the query performance of your distinct counts (and all of your Analysis Services queries overall).

The disk optimization techniques described in Part 2 are especially important for distinct count measure groups.

References:

- SQL Server 2008 R2 Analysis Services Operations Guide (http://sqlcat.com/sqlcat/b/whitepapers/archive/2011/06/01/sql-server-2008r2-analysis-services-operations-guide.aspx)
- Analysis Services Distinct Count Optimization (http://sqlcat.com/sqlcat/b/whitepapers/archive/2008/04/17/analysis-services-distinct-count-optimization.aspx)

- Analysis Services Distinct Count Optimization Using Solid State Devices (http://sqlcat.com/sqlcat/b/technicalnotes/archive/2010/09/20/analysis-services-distinct-count-optimization-using-solid-state-devices.aspx)
- SQLBI Many-to-Many Project (http://www.sqlbi.com/Projects/Manytomanydimensionalmodeling/tabid/80/Default.aspx)

2.5.2 Large Many-to-Many Dimensions

Many-to-many relationships are used heavily in many business scenarios ranging from sales to accounting to healthcare. But at times there may be query performance issues when dealing with a large number of many-to-many relationships and perceived accuracy issues. One way to think about a many-to-many dimension is that it is a generalization of the distinct count measure. The use of many-to-many dimensions enables you to apply distinct count logic to other Analysis Services measures such as sum, count, max, min, and so on. To calculate these distinct count or aggregates, the Analysis Services storage engine must parse through the lowest level of granularity of data. This is because when a query includes a many-to-many dimension, the query calculation is performed at query-time between the measure group and intermediate measure group at the attribute level. The result is a processor- and memory-intensive process to return the result.

Performance and accuracy issues concerning many-to-many dimensions include the following:

- The join between the measure group and intermediate measure group is a hash join strategy; hence it is very memory-intensive to perform this operation.
- Because queries involving many-to-many dimensions result in a join between the measure group and an intermediate measure group, reducing the size of your intermediate measure group (a general rule is less than 1 million rows) provides the best performance. For additional techniques, see the Analysis Services Many-to-Many Dimensions: Query Performance Optimization Techniques white paper (http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=137).
- Many-to-many relationships cannot be aggregated (although it generally is not very easy to create general purpose aggregates for distinct counts as well). Therefore, queries involving many-to-many dimensions cannot use aggregations or aggregate caches—only a direct hit will work. There are specific situations where many-to-many relationships can be aggregated; you can find more information in the Analysis Services Many-to-Many Dimensions: Query Performance Optimization Techniques white paper.
    - Because many-to-many cannot be aggregated, there are various MDX calculation issues with VisualTotals, subselects, and CREATE SUBCUBE.
- Similar to distinct count, there may be perceived double counting issues because it is difficult to identify which members of the dimension are involved with the many-to-many relationship.

To help improve the performance of many-to-many dimensions, one can make use of the Many-to-Many matrix compression, which removes repeated many-to-many relationships thus reducing the size of your intermediate measure group. As can be seen in the following figure, a *MatrixKey* is created

based on each set of common dimension member combinations so that repeated combinations are eliminated.

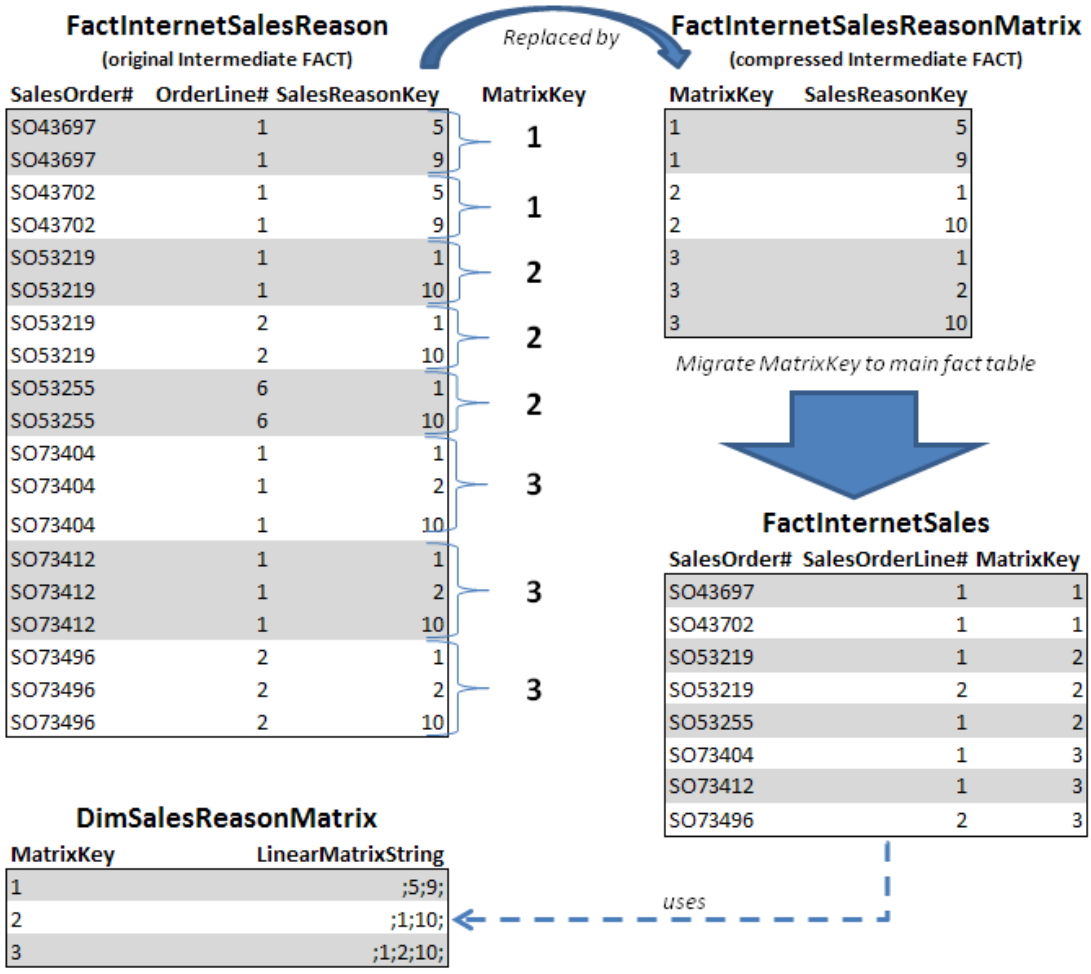


Figure 4040: Compressing the FactInternetSalesReason intermediate fact table (from Analysis Services Many-to-Many Dimensions: Query Performance Optimization Techniques)

References:

- Many-to-Many Matrix Compression (http://bidshelper.codeplex.com/wikipage?title=Many-to-Many%20Matrix%20Compression)
- SQLBI Many-to-Many Project (http://www.sqlbi.com/Projects/Manytomanydimensionalmodeling/tabid/80/Default.aspx)
- Analysis Services: Should you use many-to-many dimensions? (http://sqlcat.com/sqlcat/b/technicalnotes/archive/2008/02/11/analysis-services-should-you-use-many-to-many-dimensions.aspx)

2.5.3 Parent-Child Dimensions

The parent-child dimension is a compact and powerful way to represent hierarchies in a relational database – especially ragged and unbalanced hierarchies. Yet within Analysis Services, the query performance tends to be suboptimal, especially for large parent-child dimensions, because aggregations are created only for the key attribute and the top attribute (that is, the **All** attribute) unless it is disabled. Therefore, a common best practice is to refrain from using parent-child hierarchies that contain a large number of members. (How big is large? There isn't a specific number because query performance at intermediate levels of the parent-child hierarchy degrades linearly with the number of members.) Additionally, limit the number of parent-child hierarchies in your cube.

If you are in a design scenario with a large parent-child hierarchy, consider altering the source schema to reorganize part or all of the hierarchy into a regular hierarchy with a fixed number of levels. For example, say you have a parent-child hierarchy such as the one shown here.

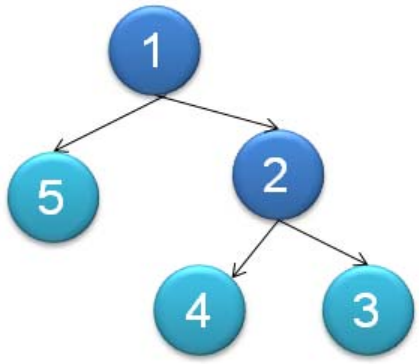


Figure 4141: Sample parent-child hierarchy

The data from this parent-child hierarchy is represented in relational format as per the following table.

SK	Parent_SK
1	NULL
2	1
3	2
4	2
5	1

Converting this table to a regularly hierarchy results in a relational table with the following format.

SK	Level0_SK	Level1_SK	Level2_SK
1	1	NULL	NULL
2	1	2	NULL
3	1	2	3
4	1	2	4
5	1	5	NULL

After the data has been reorganized into the user hierarchy, you can use the **Hide Member If** property of each level to hide the redundant or missing members. To help convert your parent-child hierarchy into a regular hierarchy, refer to the Analysis Services Parent-Child Dimension Naturalizer tool in CodePlex (http://pcdimnaturalize.codeplex.com/wikipage?title=Home&version=12&ProjectName=pcdimnaturalize).

References:

- Analysis Services Parent-Child Dimension Naturalizer (http://pcdimnaturalize.codeplex.com/wikipage?title=Home&version=12&ProjectName=pcdimnaturalize)
- Including Child Members Multiple Places in a Parent-Child Hierarchy (http://sqlcat.com/sqlcat/b/technicalnotes/archive/2008/03/17/including-child-members-multiple-places-in-a-parent-child-hierarchy.aspx)

2.5.4 Near Real Time and ROLAP

As your Analysis Services data becomes more valuable to the business, a common next requirement is to provide near real-time capabilities so users can have immediate access to their business intelligence system. Near real-time data has special requirements:

- Typically the data must reside in memory for low latency access.
- Often, you do not have time to maintain indexes on the data.
- You will typically run into locking and/or concurrency issues that must be dealt with.

It is important to note that due to the locking logic invoked by Analysis Services, long-running queries in Analysis Services can both prevent processing from committing and block other queries.

To provide near real-time results and avoid the Analysis Services query locking, start with using ROLAP so that the queries go directly to the relational database. Yet even relational databases have locking and/or concurrency issues that need to be dealt with. To minimize the impact of blocking queries within your relational database, place the real-time portion of the data into its own separate table but keep historical data within your partitioned table.After you have done this, you can apply other techniques. In this section we discuss the following:

- MOLAP switching
- ROLAP + MOLAP
- ROLAP partitioning

2.5.4.1 MOLAP Switching

The basic principle behind MOLAP switching is to create some partitions for historical data and another set of partitions for the latest data. The latencies associated with frequently processing the current MOLAP partitions are in minutes. This methodology is well suited for something like a time-zone scenario in which you have active partitions throughout the day. For example, say you have active partitions for different regions such as New York, London, Mumbai, and Tokyo. In this scenario, you would create partitions by both time and the specific region. This provides you with the following benefits:

- You can fully process (as often as needed) the active region / time partition (for example, Tokyo / Day 1) without interfering with other partitions (for example, New York / Day 1).
- You can "roll with the daylight" and process New York, London, Mumbai, and Tokyo with minimal overlap.

However, long-running queries for a region can block the processing for that region. A processing commit of current New York data might be blocked by an existing long running query for New York data. To alleviate this problem, use two copies of the same cube, alternating data processing between them (known as cube flipping).

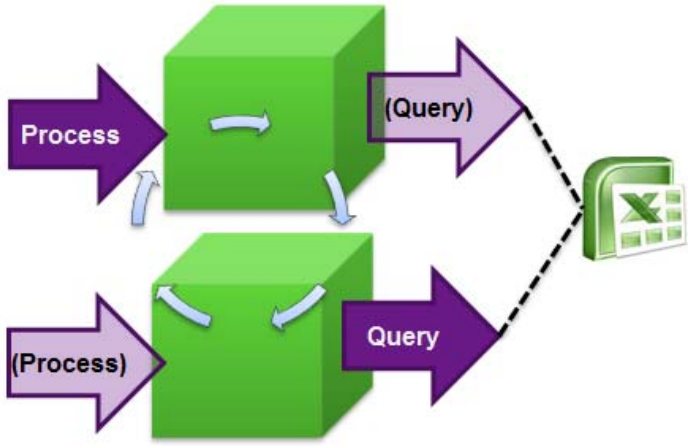


Figure 4242: Cube-flipping concept

While one cube processes data, the other cube is available for querying. To flip between the cubes, you can use the Analysis Services Load Balancing Toolkit (http://sqlcat.com/sqlcat/b/toolbox/archive/2010/02/08/aslb.aspx) or create your own custom plug-in to your UI (you can use Excel to do this, for example) that can detect which cube it should query against. It will be important for the plug-in to hold session state so that user queries use the query cache. Session state should automatically refresh when the connection string is changed.

2.5.4.2 ROLAP + MOLAP

The basic principle behind ROLAP + MOLAP is to create two sets of partitions: a ROLAP partition for frequently updated current data and MOLAP partitions for historical data. In this scenario, you typically can achieve latencies in terms of seconds. If you use this technique, be sure to follow these guidelines:

- Maintain a coherent ROLAP cache. For example, if you query the relational data, the results are placed into the storage engine cache. By default, the next query uses that storage engine cache entry, but the cache entry may not reflect any new changes to the underlying relational database. It is even possible to have aggregate values stored in the data cache that when aggregated up do not add up correctly to the parent.
- Use **Real Time OLAP = true** within the connection string.
- Assume that the MOLAP partitions are write-once / read-sometimes. If you need to make changes to the MOLAP partitions, ensure the changes do not have an impact on users querying the system.
- For the ROLAP partition, ensure that the underlying SQL data source can handle concurrent queries and load. A potential solution is to use Read Committed Snapshot Isolation (RSCI); for more information, see Bulk Loading Data into a Table with Concurrent Queries (http://sqlcat.com/sqlcat/b/technicalnotes/archive/2009/04/06/bulk-loading-data-into-a-table-with-concurrent-queries.aspx).

2.5.4.3 Comparing MOLAP Switching and ROLAP + MOLAP

The following table compares the MOLAP switching and ROLAP + MOLAP methodologies.

Component	MOLAP Switching	ROLAP + MOLAP
Relational Tuning	Low	Must get right
AS locking	Need to handle	Minimal
Cache Usage	Good	Poor
Relational Concurrency	N/A	RSCI
Data Storage	Best Compression	ROLAP sizes typically 2x MOLAP
Aggregation Management	SQL Server Profiler + UBO	Manual
Latency	Minutes	Seconds

2.5.4.4 ROLAP

In general, MOLAP is the preferred storage choice for Analysis Services; because MOLAP typically provides faster access to the data (especially if your disk subsystem is optimized for random I/O), it can handle attributes more efficiently and it is easier to manage. However, ROLAP against SQL Server can be a solid choice for very large cubes with excellent performance and the benefit of reducing or even removing the processing time of large cubes. As noted earlier, it is often a requirement if you need to have near real-time cubes. As can be seen in the following figure, the query performance of a ROLAP cube after usage-based optimization is applied can be comparable to MOLAP if the system is expertly tuned.

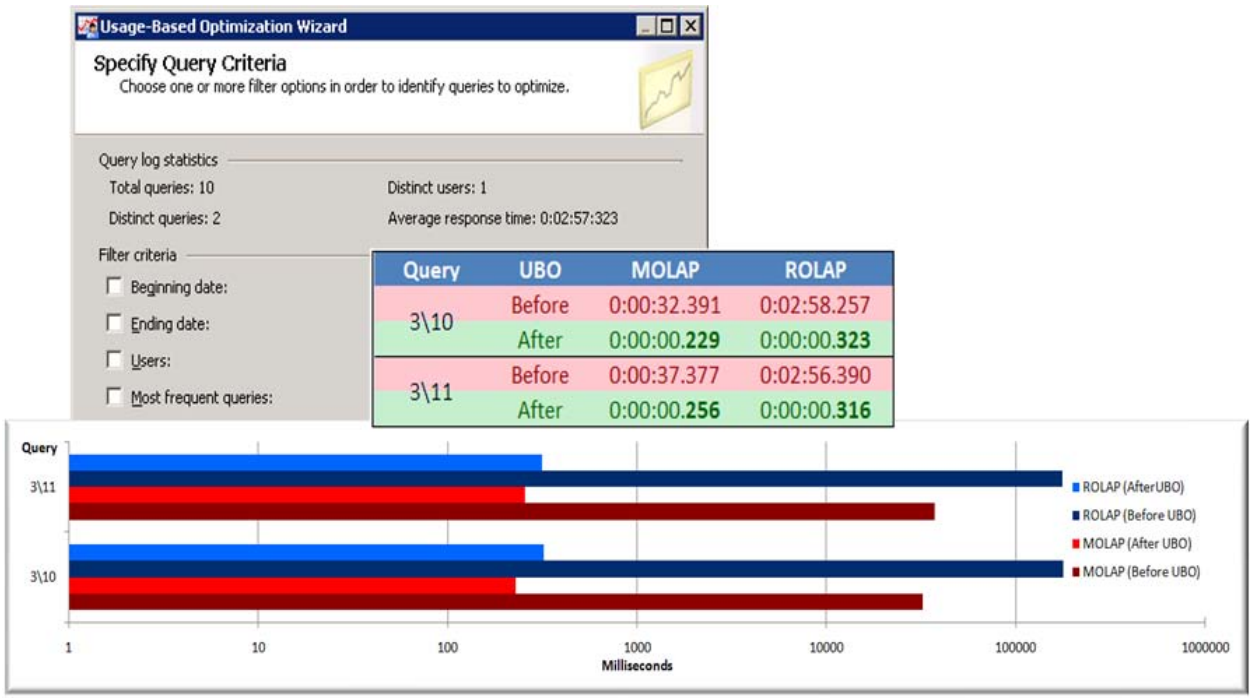


Figure 4343: Showcasing ROLAP vs. MOLAP performance before and after the application of usage-based optimization

2.5.4.4.1 ROLAP Design Recommendations

The recommendations for high performance querying of ROLAP cubes are listed here:

- Simplify the data structure of your underlying SQL data source to minimize page reads (for example, remove unused columns, try to use INT columns, and so on).
- Use a star schema without snowflaking, because joins can be expensive.
- Avoid scenarios such as many-to-many dimensions, parent-child dimensions, distinct count, and ROLAP dimensions.

2.5.4.4.2 ROLAP Aggregation Design Recommendations

When working with ROLAP partitions, you can create aggregations in two ways:

- Create cube-based aggregations by using the Analysis Services aggregations tools.
- Create your own transparent aggregations directly against the SQL Server database.

Both approaches rely on the creation of indexed views within SQL Server but offer different advantages and disadvantages. Often the most effective strategy is a combination of these two approaches as noted in the following table .

Aggregation Type	Advantages	Disadvantages
Cube-based	**Efficient query processing**: Analysis Services can use cube-based aggregations even if the query and	**Processing overhead**: Analysis Services drops and re-creates indexed views associated with cube-based aggregations

	aggregation granularities do not exactly match. For example, a query on [Month] can use an aggregation on [Day], which requires only the summarization of up to 31 numbers. **Aggregation design efficiency:** Analysis Services includes the Aggregation Design Wizard and the Usage-Based Optimization Wizard to create aggregation designs based on storage and percentage constraints or queries submitted by client applications.	during cube partition processing. Dropping and re-creating the indexes can take an excessive amount of time in a large-scale data warehouse.
Transparent	**Reuse of existing indexes across cubes:** While aggregate views can also be created by queries that do not know of their existence, the issue is that Analysis Services may unexpectedly drop the indexed views **Less overhead during cube processing**: Analysis Services is unaware of the aggregations and does not drop the indexed views during partition processing. There is no need to drop indexed views because the relational engine maintains the indexes continuously, such as during INSERT, UPDATE, and DELETE operations against the base tables.	**No sophisticated aggregation algorithms**: Indexed views must match query granularity. The query optimizer doesn't consider dimension hierarchies or aggregation granularities in the query execution plan. For example, an SQL query with GROUP BY on [Month] can't use an index on [Day]. **Maintenance overhead**: Database administrators must maintain aggregations by using SQL Server Management Studio or other tools. It is difficult to keep track of the relationships between indexed views and ROLAP cubes. **Design complexity**: Database Engine Tuning Advisor can help to facilitate aggregation design tasks by analyzing SQL Server Profiler traces, but it can't identify all possible candidates. Moreover, data warehouse(DW) architects must manually study SQL Server Profiler traces to determine effective aggregations.

Here are some general rules:

- Transparent aggregations have greater value in an environment where multiple cubes are referencing the same fact table.

- Transparent aggregations and cube-based aggregations could be used together to get the most efficient design:
    - Start with a set of transparent aggregations that will work for the most commonly run queries.
    - Add cube-based aggregations using usage-based optimization for important queries that are taking a long time to run.

2.5.4.4.3 Limitations of ROLAP Aggregations

While ROLAP is very powerful, there are some strict limitations that must be first considered before using this approach:

- You may have to design using table binding (and not query binding) to an actual table instead of a partition. The goal of this guidance is to ensure partition elimination.
    - This advice is specific to SQL Server as a data source. For other data sources, carefully evaluate the behavior of ROLAP queries when accessing a partitioned table.
    - It is not possible to create an indexed view on a view containing a subselect statement. This will prevent Analysis Services from creating index view aggregations.
- Relational partition elimination will generally not work:
    - Normally, DW best practice is to use partitioned fact tables.
    - If you need to use ROLAP aggregations, you must use separate tables in the relational database for each cube partition
    - Partitions require named queries, and those generate bad SQL plans. This may vary depending on the relational engine you use.
- You cannot use:
    - A named query or a view in the DSV.
    - Any feature that will cause Analysis Services to generate a subquery. For example, you cannot use a Count of Rows measure, because a subquery is always generated when this type of measure is used.
- The measure group cannot have:
    - Any measures that use Max or Min aggregation.
    - Any measures that are based on nullable fields in the relational data source.

References:

For more information about how to optimize your ROLAP design, see the white paper Analysis Services ROLAP for SQL Server Data Warehouses (http://sqlcat.com/sqlcat/b/whitepapers/archive/2010/08/23/analysis-services-rolap-for-sql-server-data-warehouses.aspx).

3 Part 2: Running a Cube in Production

In Part 2, you will find information on how to test and run Microsoft SQL Server Analysis Services in SQL Server 2005, SQL Server 2008, and SQL Server 2008 R2 in a production environment. The focus of this section is how you can test, monitor, diagnose, and remove production issues on even the largest scaled cubes. This book also provides guidance on how to configure the server for best possible performance.

Analysis Services cubes are a very powerful tool in the hands of the business intelligence (BI) developer. They provide an easy way to expose even large data models directly to business users. Unlike traditional, static reporting, where the query workload is known in advance, cubes support ad-hoc queries. Typically, such queries are generated by the Microsoft Excel spreadsheet software – without the business user being aware of the intricacies of the query engine. Because cubes allow such great freedom for users, the power they give to developers comes with responsibility. It is in the interaction between development and operations that the business value of the cubes is realized and where the proper steps can be taken to ensure that the power of the cubes is used responsibly.

It is the goal of this section to make your operations processes as painless as possible, and to have you run with the best possible performance without any additional development effort to your deployed cubes. In Part 2, you will learn how to get the best out of your existing data model by making changes transparent to the data model and by making configuration changes that improve the user experience of the cube.

However, no amount of operational readiness can cure a poorly designed cube. Although this section shows you where you can make changes transparent to end users, it is important to be aware that there are cases where design change is the only viable path to good performance and reliability. Cubes do not take away the ubiquitous need for informed data modeling. We hope that you'll make use of the insights and best practices in Part 1 of this book when starting new solutions from scratch.

Cubes do not exist in isolation – they rely on relational data sources to build their data structures. Although a full treatment of good relational data warehouse modeling out of scope for this document, it still provides some pointers on how to tune the database sources feeding the cube. Relational engines vary in their functionality; this book focuses on guidance for SQL Server data sources. Much of the information here should apply equally to other engines and your DBA should be able to transfer the guidance here to other database systems you run.

3.1 Configuring the Server

Installing Analysis Services is relatively straightforward. However, applying certain post-installation configuration options can improve performance and scalability of the cube. This section introduces these settings and provides guidance on how to configure them.

3.1.1 Operating System

Because Analysis Services uses the file system cache API, it partially relies on code in the Windows Server operating system for performance of its caches. Small changes are made to the file system cache in most versions of Windows Server, and this can have an effect on Analysis Services performance. We have found that patches related to the following software should be applied for best performance.

Windows Server 2008:

- KB 961719 – Applications that perform asynchronous cached I/O read requests and that use a disk array that has multiple spindles may encounter a low performance issue in Windows Server 2008, in Windows Essential Business Server 2008, or in Windows Vista SP1

Windows Server 2008 R2:

- KB 979149– A computer that is running Windows 7 or Windows Server 2008 R2 becomes unresponsive when you run a large application
- KB 976700 – An application stops responding, experiences low performance, or experiences high privileged CPU usage if many large I/O operations are performed in Windows 7 or Windows Server 2008 R2
- KB 982383 – You encounter a decrease in I/O performance under a heavy disk I/O load on a Windows Server 2008 R2-based or Windows 7-based computer

Kerberos-enabled systems:

For Kerberos-enabled systems, you may need the following additional patches:

- http://support.microsoft.com/kb/969083/
- http://support.microsoft.com/kb/2285736/

References:

- Configure Monitoring Server for Kerberos delegation - http://technet.microsoft.com/en-us/library/bb838742(office.12).aspx
- How to configure SQL Server 2008 Analysis Services and SQL Server 2005 Analysis Services to use Kerberos authentication - http://support.microsoft.com/kb/917409
- Manage Kerberos Authentication Issues in a Reporting Services Environment http://msdn.microsoft.com/en-us/library/ff679930.aspx

3.1.2 msmdsrv.ini

You can control the behavior of the Analysis Services engine to a great degree from the **msmdsrv.ini** file. This XML file is found in the **<instance dir>/OLAP/Config** folder. Many of the settings in **msmdsrv.ini** should not be changed without the assistance of Microsoft Product Support, but there are still a large number of configuration options that you can control yourself. It is the aim of this section to provide you with the guidance you need to properly configure these settings. The following table provides an overview of the settings available to you and can serve as starting point for exploring reconfiguration options.

Setting	Used for	Described in
<MemoryHeapType> **<HeapTypeForObjects>**	Increasing throughput of high concurrency system	Section 2.3.2.4
<HardMemoryLimit>	Preventing Analysis Services from consuming too much memory	Section 2.3.2
<LowMemoryLimt>	Reserving memory for the Analysis Services process	Section 2.3.2
<PreAllocate>	Locking Analysis Services memory allocation, and improving performance on Windows Server 2003 and potentially Windows Server 2008	Section 2.3.2.1
<TotalMemoryLimit>	Controlling when Analysis Services starts trimming working set	Section 2.3.2
<CoordinatorExecutionMode> **<CoordinatorQueryMaxThreads>**	Setting concurrency of queries in thread pool during query execution	Section 2.4.2
<CoordinatorBuildMaxThreads>	Increasing processing speeds	Section 7.2.1
<AggregationMemoryMin> **<AggregationMemoryMax>**	Increasing parallelism of partition processing	Section 7.2.2
<CoordinatorQueryBalancingFactor> **<CoordinatorQueryBoostPriorityLevel>**	Preventing single users from monopolizing the system	Section 2.4.3
<LimitSystemFileCacheSizeMB>	Controlling the size of the file system cache	Section 2.3.2
<ThreadPool> (and subsections)	Increasing thread pools for high concurrency systems	Section 2.4
<BufferMemoryLimit> **<BufferRecordLimit>**	Increasing compression during processing, but consuming more memory	Section 2.3.2.3
<DataStorePageSize> **<DataStoreHashPageSize>**	Increasing concurrency on a large machine	Section 2.3.2.4
<MinIdleSessionTimeout> **<MaxIdleSessionTimeout>**	Cleaning up idle sessions faster on systems that have many	Section 2.3.2.5

<IdleOrphanSessionTimeout> **<IdleConnectionTimeout>**	connect/disconnects	
<Port>	Changing the port that Analysis Services is listening on	Section 5.1
<AllowedBrowsingFolders>	Controlling which folders are viewable by the server administrator	Section 2.5 and Section 5.5
<TempDir>	Controlling where disk spills go	Section 2.5
<BuiltinAdminsAreServerAdmins>	Controlling segregation-of-duties scenarios	Section 5
<ServiceAccountIsServerAdmin>	Controlling segregation-of-dutiesscenarios	Section 5
<ForceCommitTimeout> **<CommitTimeout>**	Killing queries that are blocking a processing operation or killing the processing operation	Section 7.4.4.4

3.1.2.1 A Warning About Configuration Errors

One of the most common mistakes encountered when tuning Analysis Services is misconfiguration of the **msmdsrv.ini** file. Because of this, changing your **msmdsrv.ini** file settings should be done with extreme care. If you inherit the managing responsibilities of an existing Analysis Services server, one of the first things you should do is to run the **Windif.exe** utility, which comes with Windows Server, against the current **msmdsrv.ini** file with the default **msmdsrv.ini** file. Also, if you upgraded to SQL Server 2008 or SQL Server 2008 R2 from SQL Server 2005, you will likely want to compare the current **msmdsrv.ini** file settings with the default settings for SQL Server 2008 or SQL Server 2008 R2. As described in the Thread Pool section, for optimization, some settings were changed in SQL Server 2008 and SQL Server 2008 R2 from their default values in SQL Server 2005.

Take the following example from a customer configuration. The customer was experiencing extremely poor query/processing performance. When we looked at the memory counters, we saw this.

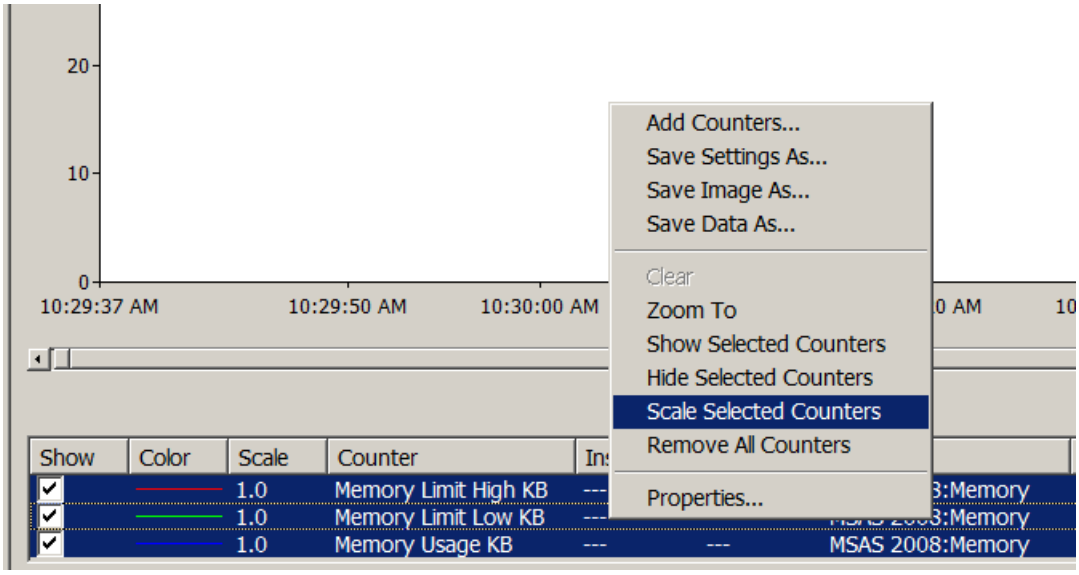


Figure 44 - Scaling Counters

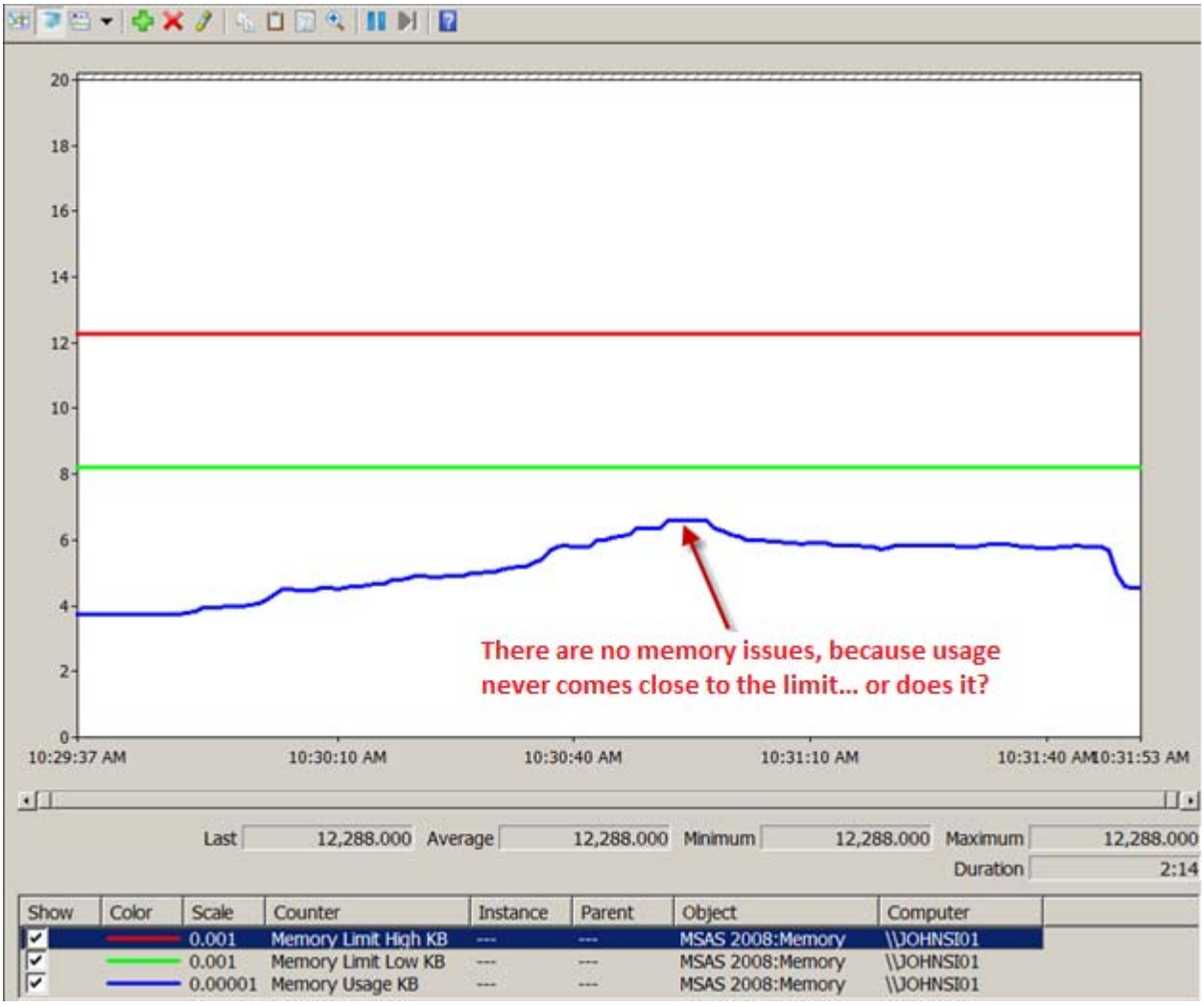


Figure 45 - Memory measurements with scaled counters

At first glance, this screenshot seems to indicate that everything is OK; there is no memory issue. However, in this case, because the customer used the **Scale Selected Counters** option in Windows Performance Monitor incorrectly, it was not possible to compare the three counters. A closer look, after the scale of the counters was corrected, shows this.

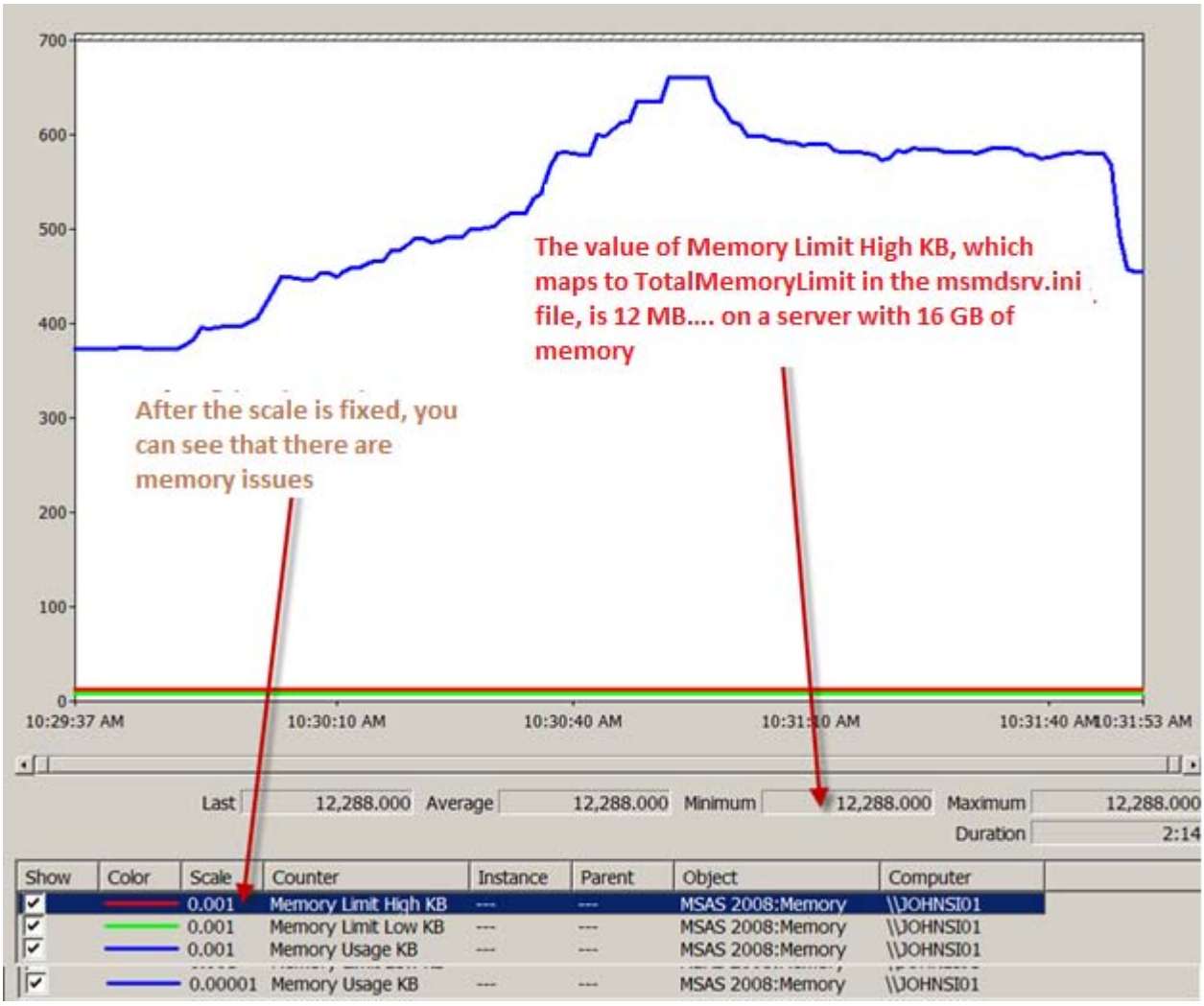


Figure 46 - Properly scaled counters

Now the fact that there is a serious memory problem is clear. Take a look at the actual **Memory Limit High KB** value. In a Performance Monitor log this and **Memory Limit Low KB** will always be constant values that reflect the value of **TotalMemoryLimit** and **LowMemoryLimit** respectively in the **msmdsrv.ini** file. So it appears that someone has modified the **TotalMemoryLimit** from the default 80 percent to an absolute value of 12,288 *bytes*, probably thinking the setting was in megabytes when in reality, the setting is in bytes. As you can see, the results of an incorrectly configured .ini file setting can be disastrous.

The moral of this story is this: Always be extremely careful when you modify your **msmdsrv.ini** file settings. One of the first things the Microsoft Customer Service and Support team does when working on an Analysis Services issue is grab the **msmdsrv.ini** file and compare it with the default .ini file for the customer's version of Analysis Services. There are numerous file comparison tools you can use, the most obvious being **Windiff.exe**, which comes with Windows Server.

References:

- How to Use the Windiff.exe Utility - http://support.microsoft.com/kb/159214

3.1.3 Memory Configuration

This section describes the memory model of Analysis Services and provides guidance on how to initially configure memory settings for a server. As will become clear in the Monitoring and Tuning section, it is often a good idea to readjust the memory settings as you learn more about the workload that is running on your server.

To understand the tradeoffs you will make during server configuration, it is useful to understand a bit more about how Analysis Services uses memory.

3.1.3.1 Understanding Memory Usage

Apart from the executable itself, Analysis Services has several data structures that consume physical memory, including caches that boost the performance of user queries. Memory is allocated using the standard Windows memory API (with the exception of a special case described later) and is part of the Msmdsrv.exe process private bytes and working set. Analysis Services does not use the AWE API to allocate memory, which means that on 32-bit systems you are restricted to either 2 GB or 3 GB of memory for Analysis Services. If you need more memory, we recommend that you move to a 64-bit system. The total memory used by Analysis Services can be monitored in Performance Monitor using either: **MSOLAP:Memory\Memory Usage Kb** or **Process\Private Bytes – msmdsrv.exe.**

Memory allocated by Analysis Services falls into two broad categories: shrinkable and non-shrinkable memory.

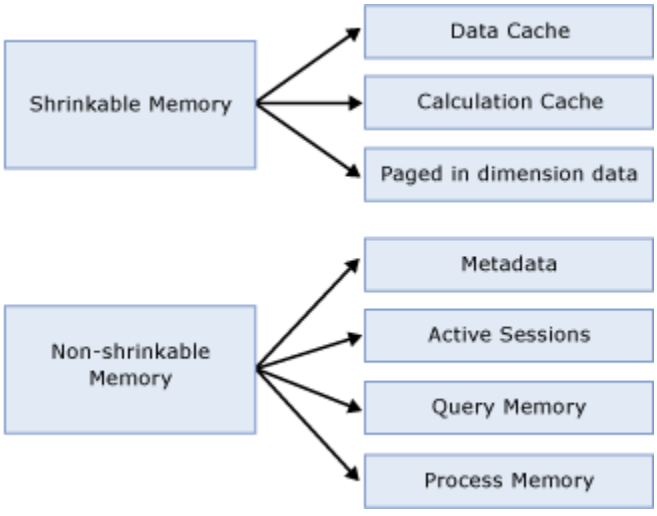


Figure 47 - Memory Structures

Non-shrinkable memory includes all the structures that are required to keep the server running: active user sessions, working memory of running queries, metadata about the objects on the server, and the

process itself. Analysis Services does not release memory in the non-shrinkable memory spaces – though this memory can see still be paged out by the operating system (but see the section on PreAllocate).

Shrinkable memory includes all the caches that gradually build up to increase performance: The formula engine has a calculation cache that tries to keep frequently accessed and calculated cells in memory. Dimensions that are accessed by users are also kept in cache, to speed up access to dimensions, attributes, and hierarchies. The storage engine also caches certain accessed subcubes. (Not all subcubes are cached. For example, the ones used by arbitrary shapes are not kept.) Analysis Services gradually trims its working set if the shrinkable memory caches are not used frequently. This means that the total memory usage of Analysis Services may fluctuate up and down, depending on the server state – this is expected behavior.

Outside of the Analysis Services process space, you have to consider the memory used by the operation system itself, other services, and the memory consumed by the file system cache. Ideally, you want to avoid paging important memory – and as we will see, this may require some configuration tweaks.

References:

- Russinovich, Mark and David Solomon: *Windows Internals, 5th edition*. http://technet.microsoft.com/en-us/sysinternals/bb963901
- Webb, Chris, Marco Russo, and Alberto Ferrari: *Expert Cube Development with Microsoft SQL Server 2008 Analysis Services* – Chapter 11
- Arbitrary Shapes in AS 2005 - https://kejserbi.wordpress.com/2006/11/16/arbitrary-shapes-in-as-2005/

3.1.3.2 Memory Setting for the Analysis Service Process

The Analysis Services memory behavior can be controlled by using parameters available in the **msmdsrv.ini** file. Except for the **LimitFileSystemCacheMB** setting, all the memory settings behave like this:

- If the setting has a value below 100, the running value is that percent of total RAM on the machine.
- If the setting has a value above 100, the running value is that amount of KB.

For ease of use, we recommend that you use the percentage values. The total memory allocations of Analysis Services process are controlled by the following settings.

LowMemoryLimit–This is the amount of memory that Analysis Services will hold on to, without trimming its working set. After memory goes above this value, Analysis Services begins to slowly deallocate memory that is not being used. The configuration value can be read from the Performance Monitor counter: **MSOLAP:Memory\LowMemoryLimit.** Analysis Services does not allocate this memory at startup; however, after the memory is allocated, Analysis Services does not release it.

PreAllocate–This optional parameter (with a default of 0) enables you to preallocate memory at service startup. It is described in more detail later in this section. **PreAllocate** should be set to a value less than or equal to **LowMemoryLimit**.

TotalMemoryLimit–When Analysis Services exceeds this memory value,it starts deallocating memory aggressively from shrinkable memory and trimming its working set. Note that this setting is not an upper memory limit for the service; if a large query consumes a lot of resources, the service can still consume memory above this value. This value can also be read from the Performance Monitor counter: **MSOLAP:Memory\TotalMemoryLimit**.

HardMemoryLimit–This setting is only available in SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services – not SQL Server 2005 Analysis Services. It is a more aggressive version of **TotalMemoryLimit**. If Analysis Services goes above **HardMemoryLimit**, user sessions are selectively killed to force memory consumption down.

LimitSystemFileCacheMB – The Windows file system cache (described later) is actively used by Analysis Services to store frequently used blocks on disk. For some scan-intensive workloads, the file system cache can grow so large that Analysis Services is forced to trim its working set. To avoid this, Analysis Services exposes the Windows API to limit the size of the file system cache with a configuration setting in Msmdsrv.ini. If you choose to use this setting, note that it limits the file system cache for the entire server, not just for the Analysis Services files. This also means that if you run more than one instance of Analysis Services on the server, you should use the same value for **LimitSystemFileCacheMB**for each instance.

3.1.3.2.1 PreAllocate and Locked Pages

The **PreAllocate** setting found in **msmdsrv.ini** can be used to reserve virtual and/or physical memory for Analysis Services. For installations where Analysis Services coexists with other services on the same machine, setting **PreAllocate** can provide a more stable memory configuration and system throughput.

Note that if the service account used to run Analysis Services also has the **Lock pages in Memory** privilege, **PreAllocate** causes Analysis Services to use large memory pages. Large memory pages are more efficient on a big memory system, but they take longer to allocate. **Lock pages in Memory** is set using Gpedit.msc. Bear in mind that large memory pages cannot be swapped out to the page file. While this can be an advantage from a performance perspective, a high number of allocated large pages can cause the system to become unresponsive.

Important:PreAllocate has the largest impact on the Windows Server 2003 operating system. With the introduction of Windows Server 2008, memory management has been much improved. We have been testing this setting on Windows Server 2008, but have not measured any significant performance benefits of using **PreAllocate** on this platform. However, you may want still want to make use of the locked pages functionality in Window Server 2008.

To learn more about the effects of **PreAllocate**, see the following technical note:

- Running Microsoft SQL Server 2008 Analysis Services on Windows Server 2008 vs. Windows Server 2003 and Memory Preallocation: Lessons Learned - (http://sqlcat.com/technicalnotes/archive/2008/07/16/running-microsoft-sql-server-2008-analysis-services-on-windows-server-2008-vs-windows-server-2003-and-memory-preallocation-lessons-learned.aspx)

References:

- How to: Enable the Lock Pages in Memory Option (Windows) - http://msdn.microsoft.com/en-us/library/ms190730.aspx

3.1.3.2.2 AggregationMemory Max/Min

Under the server properties of SQL Server Management Studio and in **msmdsrv.ini**, you will find the following settings:

- **OLAP\Process\AggregationMemoryLimitMin**
- **OLAP\Process\AggregationMemoryLimitMax**

These two settings determine how much memory is allocated for the creation of aggregations and indexes in each partition. When Analysis Services starts partition processing, parallelism is throttled based on the **AggregationMemoryMin/Max** setting. The setting is per partition. For example, if you start five concurrent partition processing jobs with **AggregationMemoryMin** = 10, an estimated 50 percent (5 x 10%) of reserved memory is allocated for processing. If memory runs out, new partition processing jobs are blocked while they wait for memory to become available. On a large memory system, allocating 10 percent of available memory per partition may be too much. In addition, Analysis Services may sometimes misestimate the maximum memory required for the creation of aggregates and indexes. If you process many partitions in parallel on a large memory system, lowering the value of **AggregationMemoryLimitMin** and **AggregationMemoryMax** may increase processing speed. This works because you can drive a higher degree of parallelism during the process index phase.

Like the other Analysis Services memory settings, if this setting has a value greater than 100 it is interpreted as a fixed amount of kilobytes, and if is lower than 100, it is interpreted as a percentage of the memory available to Analysis Services. For machines with large amounts of memory and many partitions, using an absolute kilobyte value for these settings may provide a better control of memory than using a percentage value.

3.1.3.2.3 BufferMemoryLimit and BufferRecordLimit

OLAP\Process\BufferMemoryLimit determines the size of the fact data buffers used during partition data processing. While the default value of the **OLAP\Process\BufferMemoryLimit** is sufficient for most deployments, you may find it useful to alter the property in the following scenario.

If the granularity of your measure group is more summarized than the relational source fact table, you may want to consider increasing the size of the buffers to facilitate data grouping. For example, if the source data has a granularity of day and the measure group has a granularity of month; Analysis Services

must group the daily data by month before writing to disk. This grouping occurs within a single buffer and it is flushed to disk after it is full. By increasing the size of the buffer, you decrease the number of times that the buffers are swapped to disk. Because the increased buffer size supports a higher compression ratio, the size of the fact data on disk is decreased, which provides higher performance. However, be aware that high values for the **BufferMemoryLimit** use more memory. If memory runs out, parallelism is decreased.

You can use another configuration setting to control this behavior: **BufferRecordLimit**. This setting is expressed in received records from the data source instead of a **Memory %/Kb** size. The lower of the two takes precedence. For example, if **BufferMemoryLimit** is set to 10 percent of a 32-GB memory space and **BufferRecordLimit** is set to 10 million rows, either 3.2 GB or 10,000,000 times the rowsize is allocated for the merge buffer, whichever is smaller.

3.1.3.2.4 Heap Settings, DataStorePageSize, and DataStoreHashPageSize

During query execution, Analysis Services generally touches a lot of data and performs many memory allocations. Analysis Services has historically relied on its own heap implementation to achieve the best performance. However, since Windows Server 2003, advances in the Windows Server operating system mean that memory can now be managed more efficiently by the operating system. This turns out to be especially true for multi-user scenarios. Because of this, servers that have many users should generally apply the following changes to the **msmdsrv.ini**file.

Setting	Default	Multi-user, faster heap
<MemoryHeapType>	1	2
<HeapTypeForObjects>	1	0

It is also possible to increase the page size that is used for managing the hash tables Analysis Services uses to look up values. Especially on modern hardware, we have seen the following change yield a significant increase in throughput during query execution.

Setting	Default	Bigger pages
<DataStorePageSize>	8192	65536
<DataStoreHashPageSize>	8192	65536

References:

- **KB2004668** - You experience poor performance during indexing and aggregation operations when using SQL Server 2008 Analysis Services - http://support.microsoft.com/kb/2004668

3.1.3.2.5 Clean Up Idle Sessions

Client tools may not always clean up sessions opened on the server. The memory consumed by active sessions is non-shrinkable, and hence is not returned the Analysis Services or operating system process for other purposes. After a session has been idle for some time, Analysis Services considers the session expired and move the memory used to the shrinkable memory space. But the session still consumes memory until it is cleaned up.

Because idle sessions consume valuable memory, you may want to configure the server to be more aggressive about cleaning up idle sessions. There are some **msmdsrv.ini**settings that control how Analysis Services behaves with respect to idle sessions. Note that a value of zero for any of the following settings means that the sessions or connection is kept alive indefinitely.

Setting	Description
<MinIdleSessionTimeOut>	This is the minimum amount of time a session is allowed to be idle before Analysis Services considers it ready to destroy. Sessions are destroyed only if there is memory pressure.
<MaxIdleSessionTimeout>	This is the time after which the server forcibly destroys the session, regardless of memory pressure.
<IdleOrphanSessionTimeout>	This is the timeout that controls sessions that no longer have a connection associated with them. Examples of these are users running a query and then disconnecting from the server.
<IdleConnectionTimeout>	This timeout controls how long a connection can be kept open and idle until Analysis Services destroys it. Note that if the connection has no active sessions, **MaxIdleSessionTimeout** eventually marks the session for cleaning and the connection is cleaned with it.

If your server is under memory pressure and has many users connected, but few executing queries, you should consider lowering **MinIdleSessionTimeOut** and **IdleOrphanSessionTimeout** to clean up idle sessions faster.

3.1.4 Thread Pool and CPU Configuration

Analysis Services uses thread pools to manage the threads used for queries and processing. The thread management subsystem that Analysis Services uses enables you to fine-tune the number of threads that are created. Tuning the thread pool is a balancing act between CPU overutilization and underutilization: If too many threads are created, unnecessary context switches and contention for system resources lower performance, and too few threads can lead to CPU and disk underutilization, which means that performance is not optimal on the hardware allocated to the process. When you tune your thread pools, it is essential that you benchmark your performance before and after your configuration changes; misconfiguration of the thread pool can often cause unforeseen performance issues.

3.1.4.1 Thread Pool Sizes

This section discusses the settings that control thread pool sizes.

ThreadPool\Process\MaxThreads determines the maximum number of available threads to Analysis Services during processing and for accessing the I/O system during queries. On large, multiple-CPU machines, the default value in SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services of 64 or 10 multiplied by the number of CPU cores (whichever is higher) may be too low to take advantage of all CPU cores. In SQL Server 2005 Analysis Services the settings for process thread pool were set to the static value of 64. If you are running an installation of SQL Server 2005 Analysis Services,

or if you upgraded from SQL Server 2005 Analysis Services to SQL Server 2008 Analysis Services or SQL Server 2008 R2 Analysis Services, it might be a good idea to increase the thread pool.

ThreadPool\Query\MaxThreads determines the maximum number of threads available to the Analysis Services formula engine for answering queries. In SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services, the default is 2 multiplied by the number of logical CPU or 10, whichever is higher. In SQL Server 2005 Analysis Services, the default value was fixed at 10. Again, if you are running on SQL Server 2005 Analysis Services or an upgraded SQL Server 2005 Analysis Services, you may want to use the new settings.

3.1.4.2 CoordinatorExecutionMode and CoordinatorQueryMaxThreads

Analysis Services uses a centralized storage engine job architecture for both querying and processing operations. When a subcube request or processing command is executed, a coordinator thread is responsible for gathering the data needed to satisfy the request.

The value of **CoordinatorExecutionMode** limits the total number of coordinator jobs that can be executed in parallel by a subcube request in the storage engine. A negative value specifies the maximum number of parallel coordinator jobs that can start per processing core per subcube request. By default **CoordinatorExecutionMode** is set to -4, which means the server is limited to 4 jobs in parallel per core. For example, on a 16-core machine with the default values of **CoordinatorExecutionMode** = -4, a total of 64 concurrent threads can execute per subcube request.

When a subcube is requested, a coordinator thread starts up to satisfy the request. The coordinator first queues up one job for each partition that must be touched. Each of those jobs then continues to queue up more jobs, depending on the total number of segments that must be scanned in the partition. The value of **CoordinatorQueryMaxThreads**, with a default of 16, limits the number of partition jobs that can be executed in parallel for each partition. For example, if the formula engine requests a subcube that requires two partitions to be scanned, the storage engine is limited to a maximum of 32 threads for scanning. Also note that both the coordinator jobs and the scan threads are limited by the maximum number of threads configured in **ThreadPool/Processing/MaxThreads.** The following diagram illustrates the relationship between the different threadpools and the coordinator settings.

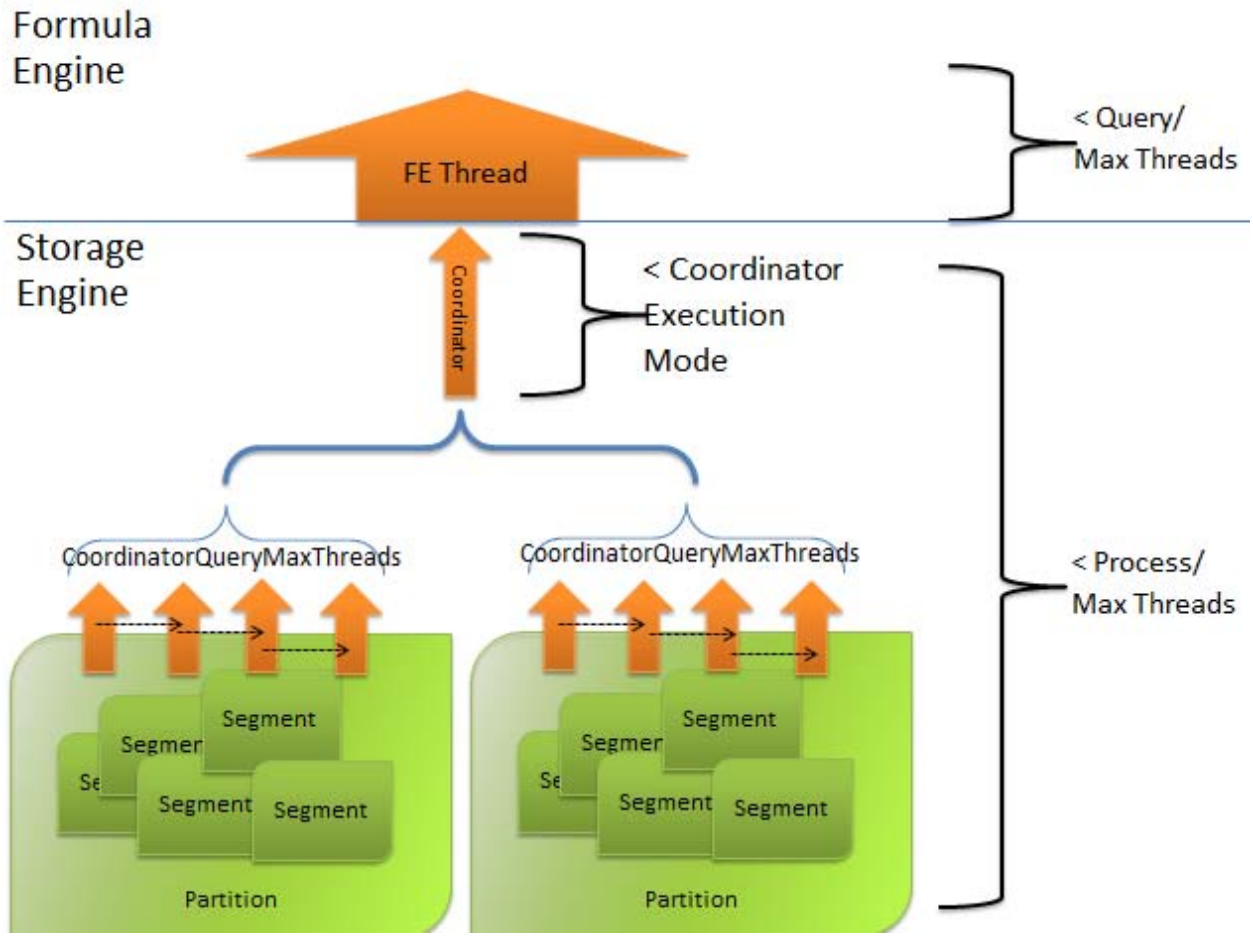


Figure 48 - Coordinator Queries

If you increase the processing thread pool, you should make sure that the **CoordinatorExecutionMode** settings, as well as the **CoordinatorQueryMaxThreads** settings, have values that enable you to make full use of the thread pool.

If the typical query in your system touches many partitions, you should be careful with the **CoordinatorQueryMaxThreads**. For example, if every query touches 10 partitions, a total 160 threads can be used just to answer that query. It will not take many queries to run the thread pool dry under those conditions. In such a case, consider lowering the setting of **CoordinatorQueryMaxThreads**.

3.1.4.3 Multi-User Process Pool Settings

In multi-user scenarios, long-running queries can starve other queries; specifically they can consume all available threads, blocking execution of other queries until the longer-running queries complete.

You can reduce the aggressiveness of how each coordinator job allocates threads per segment by modifying **CoordinatorQueryBalancingFactor** and **CoordinatorQueryBoostPriorityLevel** as follows.

Setting	Default	Multi-user nonblocking settings

CoordinatorQueryBalancingFactor	-1	1
CoordinatorQueryBoostPriorityLevel	3	0

If you want to understand exactly what these settings do, you need to know a little about the internals of Analysis Services. First, a word of warning: The remainder of this section looks at Analysis Services at a very detailed level. It is perfectly acceptable to take a query workload and test with the default settings and then with the multi-user settings to decide if it is worth making these changes.

With the disclaimer out of the way, let's look at an example to explain this behavior. On a 45-core Windows Server 2008 server with default .ini file settings, you have a long running query that appears to be blocking many of the other queries being executed by other users. Behind the scenes in Analysis Services, multiple segment jobs (different from coordinator jobs) are created to query the respective segments. A segment of data in Analysis Services is composed of roughly 64,000 records, which are subdivided into pages. There are 256 pages in a segment and 256 records in a page. These records are brought into memory in chunks upon request by queries. Analysis Services determines which pages need to be scanned to retrieve the relevant records for the data requested. These jobs are chained, meaning Job 1 queues Job 2 to the thread pool, Job 1 performs its own job, Job 2 queues Job 3 to the thread pool, Job 2 performs its own job, and so on. Each job has its own thread, or *segment job*.

In our example, using the default settings the first query fires off 720 jobs, scanning a lot of data. The second query fires off 1 job. The first 720 jobs fire off their own 720 jobs, using up all of the threads. This prevents the second query from executing, because no threads are available. This behavior causes blocking of the second and subsequent queries that need threads from the process pool. The multi-user settings (**CoordinatorQueryBalancingFactor**=1, **CoordinatorQueryBoostPriorityLevel**=0) prevent all of the threads from being allocated so the second and third queries can execute their jobs.

Again, each segment job uses one thread. If the long-running query requires scanning of multiple segments, Analysis Services creates the necessary number of threads. In single-user mode, the first query fires off 720 jobs, and the second query fires off 1 job. Each segment job is immediately queued up before the prior segment job begins scanning data. The first 720 jobs fire off their own 720 jobs, preventing second query from executing. In multi-user mode, not all threads are allocated, allowing second query (and third) to execute their jobs.

Be careful modifying these settings; although these settings reduce or stop the blocking of shorter-running queries by longer-running ones, it may slow the response times of individual queries. (In the figure, SSAS stands for SQL Server Analysis Services.)

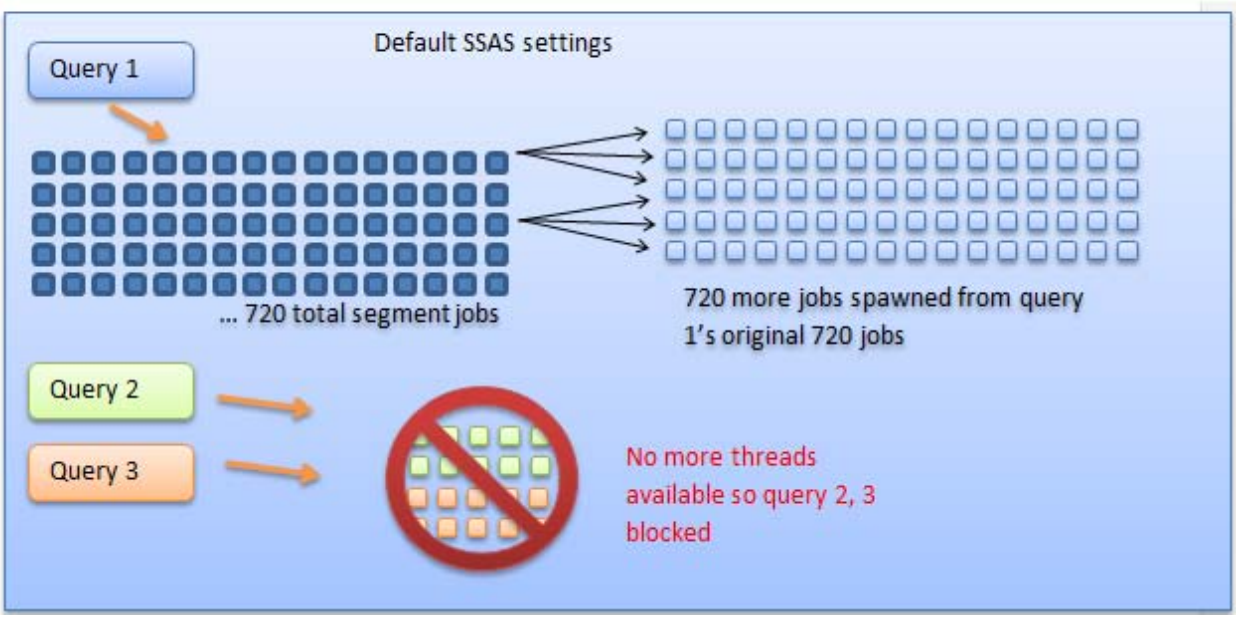


Figure 49 - Default settings

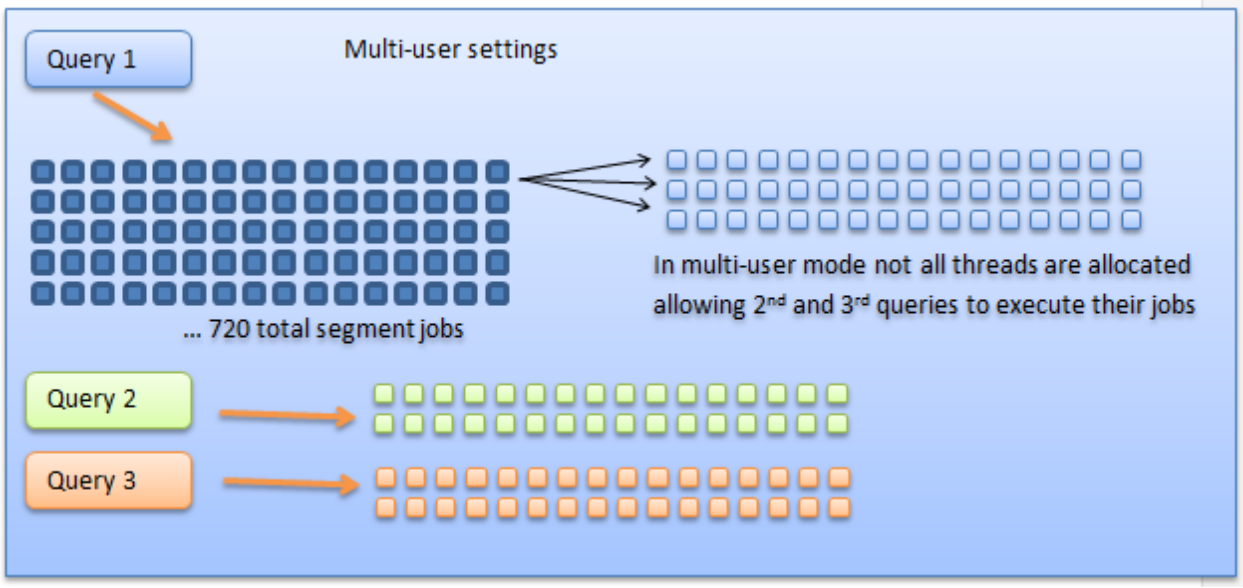


Figure 50 - Multi-user settings

3.1.4.4 Hyperthreading

We receive numerous questions from customers around hyperthreading. The Analysis Services development team has no official recommendation on hyperthreading; it is included in this book only

because our customers ask about it frequently. With that said, we have made the following observations in installations in which hyperthreading is used with Analysis Services:

- If the load on your server is more CPU-bound, turning on hyperthreading offers no improvement, in our experience.
- If the load on your server is more I/O-bound, there may be some benefit to turning on hyperthreading.
- Processors are constantly changing, and many characteristics of behavior and performance related to hyperthreading are going to be specific to the processor.

3.1.5 Partitioning, Storage Provisioning and Striping

When you configure the storage of a server you are often presented with a series of LUNs for data storage. These are provisioned from either your SAN, internal drives, or NAND memory. NAND memory, in some configuration also known as Solid State Disks/Devices (SSD) is typically a good fit for large cubes, because the latency and I/O pattern favored by these drives are a good match for the Analysis Services storage engine workload.

The question is how to make the best use of the storage for your Analysis Services instance.

Apart from performance and capacity, the following factors must also be considered when designing the disk volumes for Analysis Services:

- Is clustering of the Analysis Services instance required?
- Will you be building a scaled-out environment?

Consult the subsections that follow for guidance in these specific setups. However, the following general guidance applies.

SAN Mega LUNs: If you are using a SAN, your storage administrator may be able to provision you a single, large LUN for your cubes. Unless you plan to build a consolidation environment, having such a single, mega-LUN is probably the easiest and most manageable way to provision your storage. First of all, it provides the IOPS as a general resource to the server. Secondly, and additional advantage of a mega-LUN is that you can easily disk align this in Windows Server 2003. For Windows Server 2008 you do not need to worry about disk alignment on newly created volumes.

Windows Serverdynamic disk stripes: Using Disk Manager it is possible to combine multiple LUNs into a single Windows volume. This is a very simple way to combine multiple, similarly sized, similarly performing LUNs into a single mount point or drive letter. We have tested dynamic disk stripes in Windows Server 2008 R2 all the way up to 100.000 IOPS – and the performance overhead to that level is negligible.

> **Note:** You *cannot* use dynamic disk stripes in a cluster. This is discussed in greater detail later in Part 2.

Drive letters versus mount points: Both drive letters and mount points will work with Analysis Services cubes. If the server is dedicated to a single Analysis Services instance, a drive letter may be the simplest solution. Choosing drive letters versus mount points is often a matter of personal preference, or it can be driven by the standards of your operations team. From the perspective of performance, one is not superior to the other.

AllowedBrowsingFolders: Remember that in order for a directory to be visible to administrators of Analysis Services in SQL Server Management Studio, it must be listed in **AllowedBrowsingFolders**, which is available in SQL Server Management Studio by clicking **Server Properties** and then **Advanced Properties**.

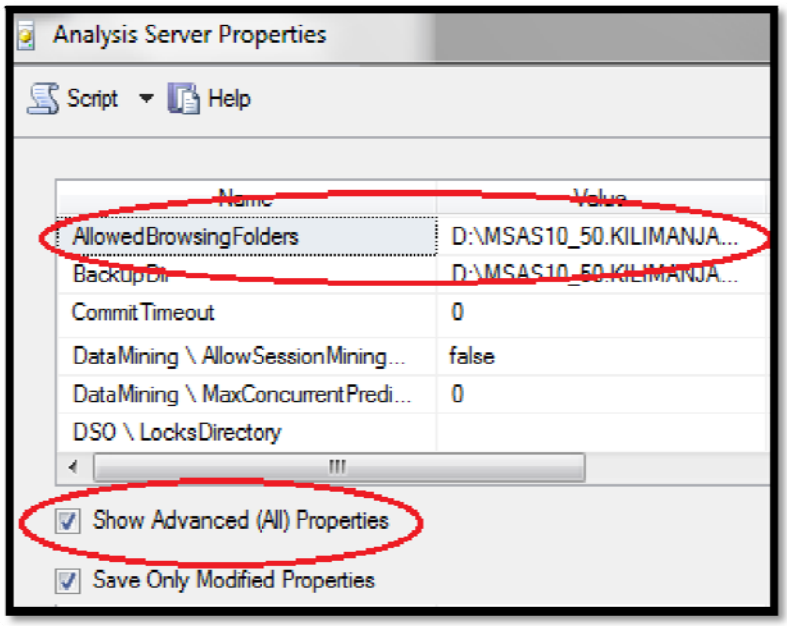


Figure 51 – AllowedBrowsingFolders

The Analysis Services account must also have permission to both read and write to these folders. Just adding them to the **AllowedBrowsingFolders** list is not enough – you must also assign file system permissions.

MFT versus GPT disk: If you expect your cubes to be larger than 2 terabytes, you should use a GPT disk. The default, an MFT disk, only allows 2-terabyte partitions.

TempDir Folder: The **TempDir** folder, configured in the Advanced Properties of the server, should be moved to the fastest volume you have. This may mean you will share TempDir with cube data, which is fine if you plan capacity accordingly. For more information, see the section about the **TempDir** folder.

NTFS Allocation Unit Size (AUS): With careful optimization, it is sometimes possible to get slightly better performance by smartly choosing between 32K or 64K (a few percent). But, unless you are hunting for benchmark performance, just go with 32K. If you have standardized on 64K for other SQL Server services, that will work too.

Don't suboptimize storage: There are a few cases where it makes sense to split your data into multiple volumes – for example if you have different storage types attached to the server (such as NAND for latest data and SATA drives for historical data). However, it is generally *not* a good idea to suboptimize the storage layout of Analysis Services by creating complicated data distributions that span different storage types. The rule of thumb for optimal disk layout is to utilize all disk drives for all cube partitions. Create large pools of disk, presented as single, large volumes.

Exclude Analysis Services folders from virus scanners: If you are running a virus scanner on the server, make sure the **Data** folder, **TempDir**, and backup folders are not being scanned or touched by the filter drivers of the antivirus tool. There are no executable files in these Analysis Services folders, and enabling virus scanners on the folder may slow down the disk access speeds significantly.

Consider Defragmenting the Data Folder: Analysis Services cubes get very fragmented over time. We have seen cases where defragmenting cube data folders, using the standard disk defragment utility, yielded a substantial performance benefit. Note that defragmenting a drive also has impact on the performance of that drive as it moves the files around. If you choose to defragment an Analysis Services drive, do so in batch window where the service can be taken offline while the defragmentation happens, or plan disk speeds accordingly to make sure the performance impact is acceptable.

References:

- White paper: Configuring Dynamic Volumes - http://technet.microsoft.com/en-us/library/bb727122.aspx

3.1.5.1 I/O Pattern

Because Analysis Services uses bitmap indexes to quickly locate fact data, the I/O generated is mostly random reads. I/O sizes will typically average around 32-KB block sizes.

As with all SQL Server databases systems, we recommend that you test your I/O subsystem before deploying the database. This allows you to measure how close the production system is to your maximum potential throughput. Not running preproduction I/O testing of an Analysis Services server is the equivalent of not knowing how much memory or how many CPU cores your server has.

Due to the threading architecture of the storage engine, the I/O pattern will also be highly multi threaded. Because the NTFS file system cache issues synchronous I/O, each thread will have only one or two outstanding I/O requests. Incidentally, this means that cubes will often run very well on NAND storage that favor this type of I/O pattern.

The Analysis Services I/O pattern can be easily simulated and tested using SQLIO.exe. The following command-line parameter will give you a good indication of the expected performance:

```
sqlio -b32-frandom-o1-s30-t256-kR<path of file>
```

Make sure you run on a sufficiently large test file. For more information, refer to the links in the References section.

References:

- SQLIO download: http://www.microsoft.com/downloads/en/details.aspx?familyid=9a8b005b-84e4-4f24-8d65-cb53442d9e19&displaylang=en
- White paper: Analyzing I/O characteristics and sizing storage for SQL Server Database Applications – http://sqlcat.com/whitepapers/archive/2010/05/10/analyzing-i-o-characteristics-and-sizing-storage-systems-for-sql-server-database-applications.aspx
- Blog: The Memory Shell Game – http://blogs.msdn.com/b/ntdebugging/archive/2007/10/10/the-memory-shell-game.aspx

3.1.6 Network Configuration

During the **ProcessData** phase of Analysis Services processing, rows are transferred from the relational database specified in the data source to Analysis Services. If your data source is on a remote server, the data should be retrieved over TCP/IP. It is important to make sure all networking components are configured to support the optimal throughput. If your Ethernet throughput is consistently close to 80 percent of your maximum capacity, adding more network capacity typically speeds up **ProcessData**.

Creating a high-speed network is fairly easy with the networking hardware available today. Specifically most servers come with a Gigabit NICs out of the box. Infiniband and 10-gigabit NICs are also available for even more throughput. With that said, your overall throughput can be limited by routers and other hardware in your network that don't support the speed of your NIC. You can use the **Networking** tab in Task Manager to quickly determine your link speed. In the following screenshot you can see that the NIC and switch support a maximum of 1Gbps.

Adapter Name	Network Utilization	Link Speed	State
Local Area Connection	0 %	1 Gbps	Connected

Figure 52 - Viewing NIC speed

If you have a 1-Gbps NIC, but only a 100-Mbps switch, Task Manager displays 100 Mbps. Depending on your network topology there may be more to determining your link speed than this, but using Task Manager is a simple way to get a rough idea of the configuration.

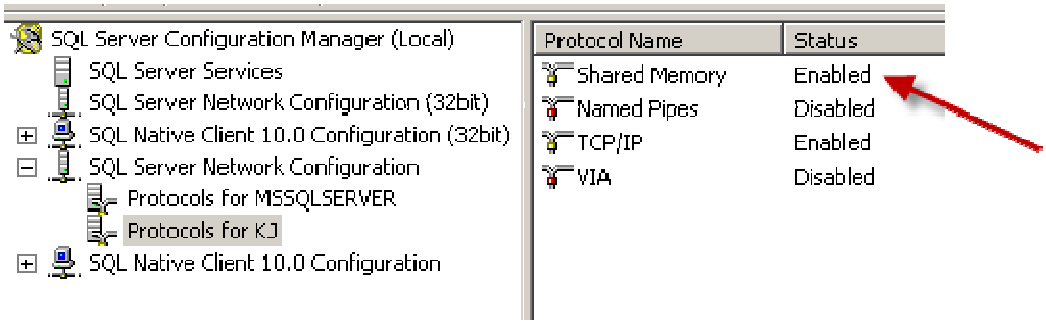
In addition to creating a high-speed network, there are some additional configurations you can change to further speed up network traffic.

3.1.6.1 Use Shared Memory for Local SQL Server Data Sources

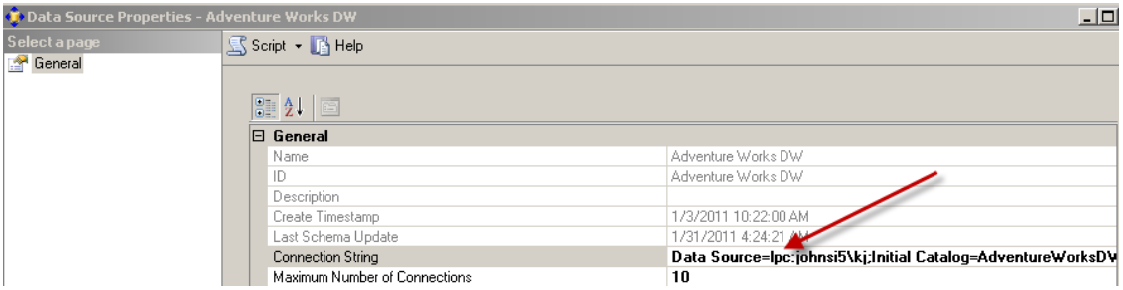
If Analysis Services is on the same physical machine as the data source, and the data source is SQL Server, you should make sure you are exchanging data over the shared memory protocol. Shared memory is much faster than TCP/IP, as it avoids the overhead of the translation layers in the network stack. Shared memory is only possible if the SQL Server database is on the same physical machine as Analysis Services. If you cannot get a high speed network set up in your organization, you can sometimes benefit from running SQL Server and Analysis Services on the same physical machine.

To modify your data source connection to specify shared memory:

1. First, make sure that the Shared Memory protocol is enabled in SQL Server Configuration Manager.



2. Next, add lpc: to your data source in the connection string to force shared memory.



3. After you start processing, you can verify your connection is using shared memory by executing the following SELECT statement.

```
SELECT session_id, net_transport, net_packet_size
FROMsys.dm_exec_connections
```

The **net_transport** for the Analysis Services SPID should show: **Shared memory**.

For more information about shared memory connections, see “Creating a Valid Connection String Using Shared Memory Protocol” (http://msdn.microsoft.com/en-us/library/ms187662.aspx).

To compare TCP/IP and shared memory, we ran the following two processing commands.

```
<!--SQL Server Native Client with TCP-->
<Batchxmlns="http://schemas.microsoft.com/analysisservices/2003/engine">
<Parallel>
<Process…
<Object>
<DatabaseID>Adventure Works DW 2008R2</DatabaseID>
<CubeID>Adventure Works</CubeID>
</Object>
<Type>ProcessFull</Type>
<DataSourcexsi:type="RelationalDataSource">
<ID>Adventure Works DW</ID>
<Name>Adventure Works DW</Name>
```

```
      <ConnectionString>
      Provider=SQLNCLI10.1;Data Source=tcp:johnsi5\kj;
      Integrated Security=SSPI;Initial Catalog=AdventureWorksDW2008R2;
      </ConnectionString>
<ImpersonationInfo>
<ImpersonationMode>ImpersonateCurrentUser</ImpersonationMode>
</ImpersonationInfo>
<Timeout>PT0S</Timeout>
</DataSource>
</Process>
</Parallel>
</Batch>

<!--SQL Native Client with shared memory-->
<Batchxmlns="http://schemas.microsoft.com/analysisservices/2003/engine">
<Parallel>
<Process…
<Object>
<DatabaseID>Adventure Works DW 2008R2</DatabaseID>
<CubeID>Adventure Works</CubeID>
</Object>
<Type>ProcessFull</Type>
<DataSourcexsi:type="RelationalDataSource">
<ID>Adventure Works DW</ID>
<Name>Adventure Works DW</Name>
<ConnectionString>
      Provider=SQLNCLI10.1;Data Source=lpc:johnsi5\kj;
      Integrated Security=SSPI;Initial Catalog=AdventureWorksDW2008R2;
      </ConnectionString>
<ImpersonationInfo>
<ImpersonationMode>ImpersonateCurrentUser</ImpersonationMode>
</ImpersonationInfo>
<Timeout>PT0S</Timeout>
</DataSource>
</Process>
</Parallel>
</Batch>
```

The first command using TCP/IP maxed out at 112,000 rows per second. Because the data source for the cube was on the same machine as Analysis Services, we were able to use shared memory in the second processing command and get much better throughput: 180,000 max rows/sec.

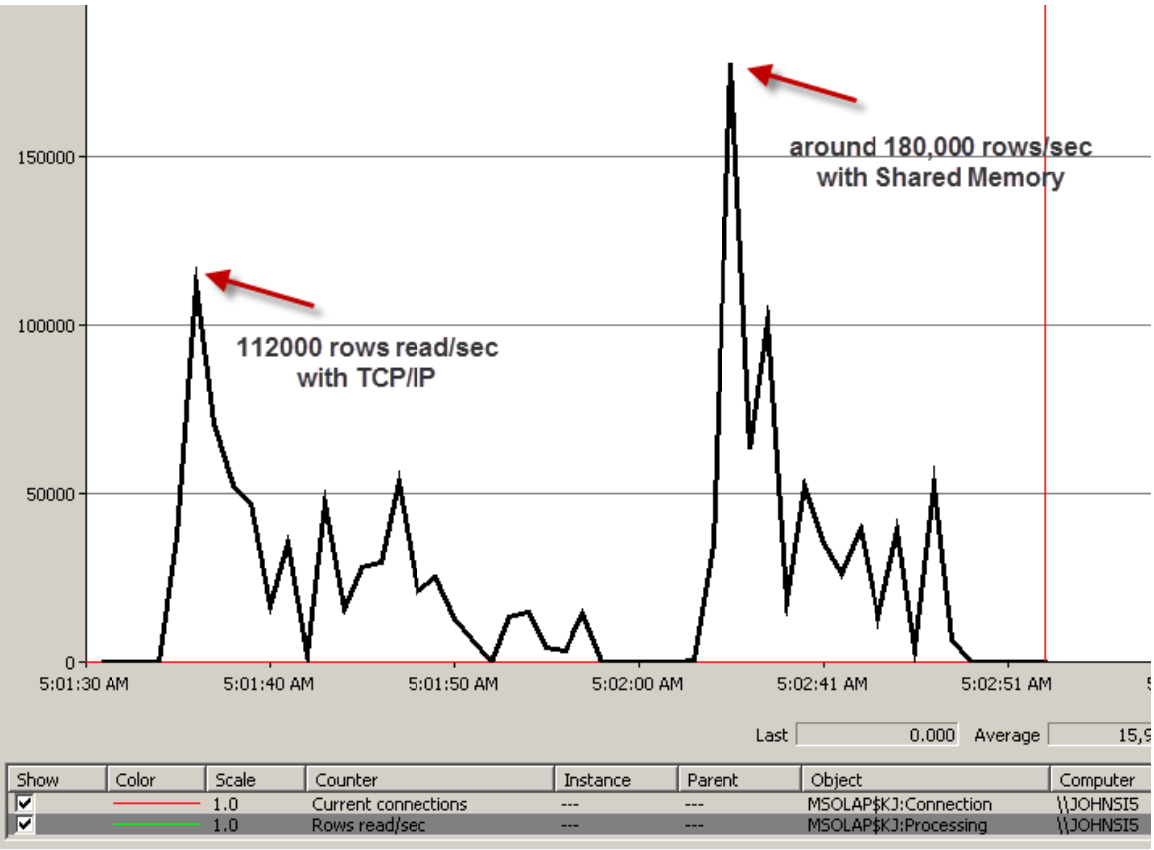


Figure 53 - Comparing rows/sec throughput

3.1.6.2 High-Speed Networking Features

Windows Server 2008 R2 has numerous improvements in network virtualization support that enable enhanced networking support. Windows Server 2008 R2 has also enhanced the support of jumbo packets and TCP offloading that was introduced in Windows Server 2008. Additionally Virtual Machine Queue (VMQ) support was added to allow network routing and data copy processing to be offloaded to a physical NIC. These features were introduced to take advantage of the capabilities found in the 10GbE server NICs.

3.1.6.2.1 TCP Chimney

TCP Chimney is a networking technology that transfers TCP/IP protocol processing from the CPU to a network adapter during the network data transfer. There have been some issues with enabling TCP Chimney in the past, and for that reason many people recommended turning it off. The technology has matured and many of the issues reported were specific to the NIC manufacturer. Applications that have long-lived connections transferring a lot of data benefit the most from the TCP Chimney feature. Processing Analysis Services data from a remote server falls into this category, so we recommend that you make sure that TCP Chimney is enabled and configured correctly. TCP Chimney can be enabled and disabled in the operating system and in the advanced properties of the network adapter.

To enable TCP Chimney you need to perform the following:

1. Enable TCP chimney in the operating system using the **netsh** commands.
2. Ensure the physical network adapter supports TCP Chimney offload, and then enable it for the adapter in the network driver.

TCP chimney has three modes of operation: Automatic (new in Windows Server 2008 R2), Enabled, and Disabled. The default mode in Windows Server 2008 R2, Automatic, checks to make sure the connections considered for offloading meet the following criteria:

- 10Gbps Ethernet NIC installed and connection established through 10GbE adapter
- Mean round trip link latency is less than 20 milliseconds
- Connection has exchanged at least 130 KB of data

To determine whether TCP Chimney is enabled, run the following from an elevated command prompt.

netsh int tcp show global

In the results, check the Chimney Offload State setting.

```
TCP Global Parameters
----------------------------------------------
Receive-Side Scaling State          : enabled
Chimney Offload State               : automatic
NetDMA State                        : enabled
Direct Cache Access (DCA)           : disabled
Receive Window Auto-Tuning Level    : normal
Add-On Congestion Control Provider  : none
ECN Capability                      : disabled
RFC 1323 Timestamps                 : disabled
```

In this example, the results show that TCP Chimney offloading is set to automatic. This means that it is enabled as long as the requirements mentioned earlier are met.

After you verify that TCP Chimney is enabled at the operating system level, check the NIC settings in device manager:

1. Go to start | run and type **devmgmt.msc**.
2. In Device Manager, expand Network Adapters, and right-click the name of the physical NIC adapter, and then click **Properties**.
3. On the **Advanced** tab, under **Property**, click**TCP Chimney Offload** or **TCP Connection Offload**, and then under **Value**, verify that **Enabled** is displayed. You may need to do this for both IPv4 and IPv6. Note that **TCP Checksum Offload** is not the same as **TCP Chimney Offload**.

If the system has IPsec enabled, no TCP connections are offloaded, and TCP Chimney provides no benefit. There are numerous different configuration options for TCP Chimney offloading that our outside the scope of this book. See the references section for a deeper treatment.

References:

- Windows Server 2008 R2 Networking Deployment Guide: Deploying High-Speed Networking Features http://download.microsoft.com/download/8/E/D/8EDE21BC-0E3B-4E14-AAEA-9E2B03917A09/HSN_Deployment_Guide.doc
- Windows Server 2008 R2 Networking Deployment Guide http://download.microsoft.com/download/8/E/D/8EDE21BC-0E3B-4E14-AAEA-9E2B03917A09/HSN_Deployment_Guide.doc

3.1.6.2.2 Jumbo Frames

Jumbo frames are Ethernet frames with more than 1,500 and up to 9,000 bytes of payload. Jumbo frames have been shown to yield a significant throughput improvement during Analysis Services processing, specifically Process Data. Network throughput is increased while CPU usage is minimized. Jumbo frames are only available on gigabit networks, and all devices in the network path must support them (switches, routers, NICs, and so on).

Most default Ethernet set ups are configured to have MTU sizes of up to 1,500 bytes. Jumbo frames allow you to go up to 9,000 bytes in a single transfer. To enable jumbo frames:

1. Configure all routers to support jumbo frames.
2. Configure the NICs on the Analysis Services machine and the data source machine to support jumbo frames.
    a) Click the **Start**button, point to **Run**, and then type **ncpa.cpl**.
    b) Right-click your network connection, and then click **Properties**.
    c) Click **Configure,** and then click the **Advanced** tab. Under **Property**, click **Jumbo Frame**, and then under **Value**, change the value from **Disabled** to **9kb MTU** or **9014**, depending on the NIC.
    d) Click **OK** on all the dialog boxes. After you make the change, the NIC loses network connectivity for a few seconds and you should reboot.

3. After you configure jumbo frames, you can easily test the change by pinging the server with a large transfer:

    Ping <servername> -f –l 9000

    You should only measure one network packet per ping request in Network Monitor.

3.1.6.3 Hyper-V and Networking

If you are running Analysis Services and SQL Server in a Hyper-V virtual machine, there are a few things you should be aware of specific to networking.

When you enable Hyper-V and create an external virtual network, both the guest and the host machine go through a virtual network adapter to connect to the physical network. All traffic goes through your virtual network adapter, and there is additional latency overhead associated with this.

3.1.6.4 Increase Network Packet Size

Under the properties of your data source, increasing the network packet size for SQL Server minimizes the protocol overhead require to build many, small packages. The default value for SQL Server 2008 is 4096. With a data warehouse load, a packet size of 32K (in SQL Server, this means assigning the value 32767) can benefit processing. Don't change the value in SQL Server using **sp_configure**; instead override it in your data source. This can be set whether you are using TCP/IP or Shared Memory.

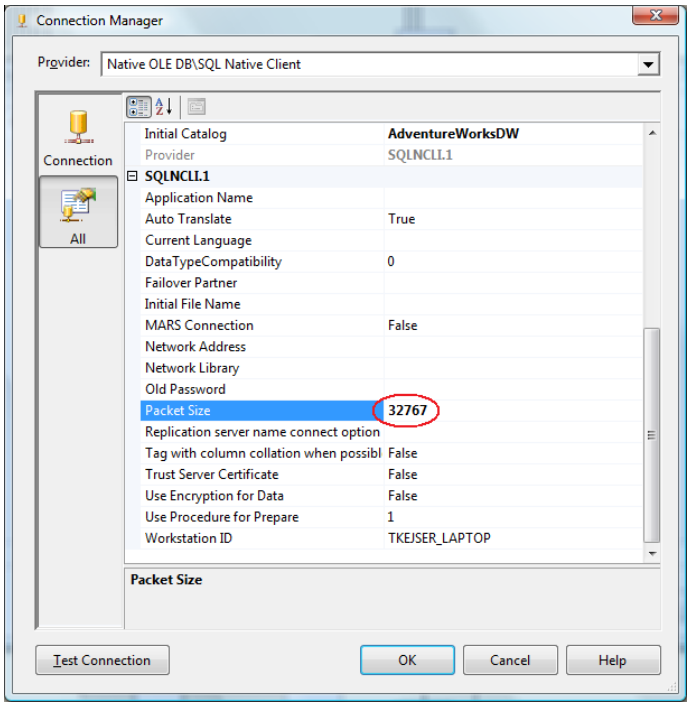


Figure 54 Tuning network packet size

3.1.6.5 Using Multiple NICs

If you are hitting a network bottleneck when you process your cube, you can add additional NICs to increase throughput. There are two basic configurations for using multiple NICs on a server. The first and default option is to use the NICs separately. The second option is to use NIC teaming.

Multiple NICs can be used individually to concurrently run many multipartition processing commands. Each partition in Analysis Services can refer to a different data source, and it is this feature you make use of. To use multiple NICs, follow these steps:

1. Set up NICs with different IP numbers.
2. Add new data sources.
3. Change your partition setup to reference the new data sources.

First, set up multiple NICs in the source and the target, each with its own IP number. Match each NIC on the cube server with a corresponding NIC on the relational database. Set up the subnet mask so only the desired NIC on the source can be reached from its partner on the cube server. If you are limited in bandwidth on the switches or want to create a very simple setup, you can use this technique with a crossover cable, which we have done successfully

Second, add a data source for each NIC on the data source, and set up each data source to point to its own source NIC.

Third, configure partitions in the cube so that they are balanced across all data sources. For example, if you have 16 partitions and 4 data sources per NIC – point 4 partitions to each data source.

Schematically, the setup looks like this.

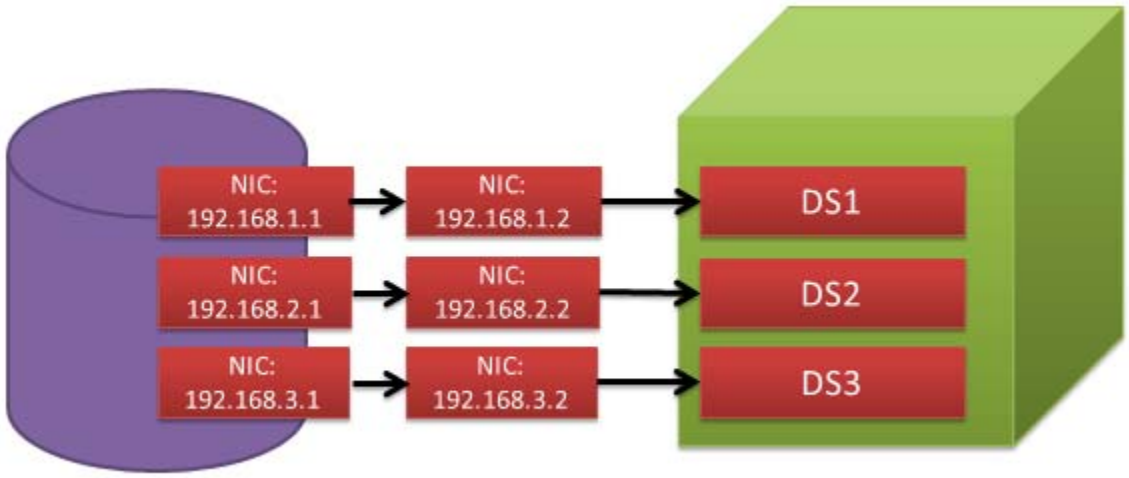


Figure 55 - Using multiple NICs for processing

NIC teaming: Another option is to team multiple NICs so they appear as one NIC to Windows Server. It is difficult to make specific recommendations around using teaming, because performance is specific to the hardware and drivers used for teaming. To determine whether NIC teaming would be beneficial to your processing workload, test with it both enabled and disabled, and measure your performance results with both settings.

3.1.7 Disabling Flight Recorder

Flight Recorder provides a mechanism to record Analysis Services server activity into a short-term log. Flight Recorder provides a great deal of benefit when you are trying to troubleshoot specific querying and processing problems by logging snapshots of common DMV into the log file. However, it introduces an amount of I/O overheard on the server itself.

If you are in a production environment and you are already monitoring the Analysis Services instance using the information in Part 2, you do not need Flight Recorder capabilities and you can disable its

logging and remove the I/O overhead. The server property that controls whether Flight Recorder is enabled is the **Log\Flight Recorder\Enabled** property. By default, this property is set to **true**.

3.2 Monitoring and Tuning the Server

As part of healthy operations management, you must collect data that allows both reactive and proactive behaviors that increase system stability, performance, and integrity. The temptation is to collect a lot of data in the belief that with more knowledge comes better decisions. This is not always the case, because you may either overload the server with data collection overhead or collect data points that you cannot take any action on.

Hence, for every data point you collect, you should have an idea about what that data collection helps you achieve. Part 2 of this book provides you with the knowledge you need to interpret the measurements you configure and how to take action on them. For ease of reference, this section summarizes the data collection required. You can use it as a checklist for data collection.

The tool used to collect the data does not matter much; it is often a question of preference and operational procedures. What this section provides is the source of the data points – how you aggregate and collect them will depend on your organization.

3.2.1 Performance Monitor

On an Analysis Services server, the minimal collection of data is this.

Performance object	Counter	Description
Logical Disk	All / All	Collects data about disk load and utilization.
Process	Private Bytes – msmdsrv.exe	The memory consumed by the Analysis Services process.
MSOLAP:Memory	Memory Usage Kb	Alternative to **Process**.
MSOLAP:Connection	Current Connections	Measures the number of open connections to gauge concurrency.
MSOLAP:Threads	*	Allows tuning of thread pools.
Memory	Cache Bytes Standby Cache Normal Priority Bytes Standby Cache Core Bytes Standby Cache Reserve Bytes	Estimates the size of the file system cache.
Memory	Page Faults / sec	Tests for excessive paging of the server.
Memory	Available Bytes	Used to tune memory settings.
MSOLAP: Proc Aggregations	temp file bytes written	Measures the spill from processing operations. Ideally, cube and hardware should be balanced to make this 0.
MSOLAP: Proc Indexes	Current Partitions Rows/sec	Measures speed and concurrency of process index.

MSOLAP: Processing	Rows read/sec Rows written/sec	Measures speed of relation read and efficiency of merge buffers.
MSOLAP:MDX	*	Used by cube tuners to determine whether calculation scripts or MDX queries can be improved.
System	File Read Bytes/sec	Measure bytes read from the file system cache.
System	File Read Operations/sec	Measures IOPS from file system cache
Network Interface	Bytes Received/sec Bytes Sent/sec	Contains the capacity plan to follow if NIC speed slows. See section 2.6.5.
TCPv4 and **TCPv6**	Segments / sec Segments Retransmitted/sec	Discovers unstable connections.

In most cases, you can collect this information every 15 seconds without causing measurable impact to the system.

If you are using Systems Center Operations Manager (SCOM) to monitor servers, it is a good idea to add this data collection to your monitoring. Alternatively, you can use the built in performance monitor in Windows Server, but that will require that you harvest the counters from the servers yourself.

3.2.2 Dynamic Management Views

Starting with SQL Server 2008 Analysis Services, there is a new set of monitoring tools available to you: dynamic management views (DMVs). These views provide information about the operations of the service that you cannot find in SQL Server Profiler or Performance Monitor.

The following table lists the most useful DMVs collect on a regular basis from the server.

DMV	Description
$system.discover_commands	Lists all currently running commands on the server, including commands run by the server itself.
$system.discover_sessions	List all sessions on the server. It is used to map commands and connections together.
$system.discover_connections	Lists current open connections.
$system.discover_memoryusage	Lists all memory allocations in the server.
$system.discover_locks	Lists currently held and requested locks.

Your capture rate of the data depends on the system you are running and how quickly your operations team response to events. **$system.discover_locks** and **$system.discover_memoryusage** are expensive DMVs to query – and gathering them too often consumes significant CPU resources on the server. Capturing the DMVs once every few minutes is probably enough for most operations management purposes. If you choose to capture them more often, measure the impact on your server, which will depend on the concurrency of executing sessions, memory sizes, and cube design.

Depending on how often you query the DMV, you can generate a lot of data. It is often a good idea to keep the recent data at a high granularity and aggregate older data. Using a tool like Microsoft StreamInsight enables you to perform such historical aggregation in real time.

Capturing the DMV data enables you to monitor the progress of queries and alert you to heavy resource consumers early. Here are some examples of issues you can identify when you use DMVs:

- Queries that consume a lot of memory
- Queries that have been blocked for a long time
- Locks that are held for a long time
- Sessions that have run for a long time, or consumed a lot of I/O
- Connections that are transmitting a large amount of data over the network

One way to collect this data is to use a linked server from the SQL Server Database Engine to Analysis Services and use the Database Engine as the storage for the data you collect. You may also be able to configure your favorite monitoring tool to do the same. Unfortunately, Analysis Services does not ship with a fully automated tool for this data collection.

The following example collection scripts can get you started with a basic data collection framework:

```
select SESSION_SPID /* Key in commands */
/* Monitor for large values and large different with below */
, COMMAND_ELAPSED_TIME_MS
, COMMAND_CPU_TIME_MS /* Monitor for large values */
, COMMAND_READS   /* Monitor for large values */
, COMMAND_WRITES /* Monitor for large values */
, COMMAND_TEXT /* Track any problems to the query */
from $system.discover_commands

select SESSION_SPID     /* Join to SPID in Sessions */
, SESSION_CONNECTION_ID /* Join to connection_id in Connections */
, SESSION_USER_NAME /* Finds the authenticated user */
, SESSION_CURRENT_DATABASE
from $system.discover_sessions

select CONNECTION_ID    /* Key in connections */
, CONNECTION_HOST_NAME /* Find the machine the user is coming from */
, CONNECTION_ELAPSED_TIME_MS
, CONNECTION_IDLE_TIME_MS /* Find clients not closing connections */
/* Monitor the below for large values */
, CONNECTION_DATA_BYTES_SENT
, CONNECTION_DATA_BYTES_RECEIVED
from $system.discover_connections

select SPID
, MemoryName
, MemoryAllocated
, MemoryUsed
, Shrinkable
, ObjectParentPath
, ObjectId
from $system.discover_memoryusage
```

```
where SPID <> 0
```

References:

- MSDN library reference on Analysis Services DMVs: http://msdn.microsoft.com/en-us/library/ee301466.aspx
- CodePlex ResMon tool, which captures detailed information about cubes: http://sqlsrvanalysissrvcs.codeplex.com/wikipage?title=ResMon%20Cube%20Documentation

3.2.3 Profiler Traces

Analysis Services exposes a large number of events through the profiler API. Collecting every single one during normal operations not only generates high data collection volumes, it also consumes significant CPU and network traffic. While it is possible to dig into great detail of the server, you should measure to impact on the system while running the trace. If every event is traced, a lot of trace information is generated: Make sure you either clean up historical records or have enough space to hold the trace data. Very detailed SQL Server Profiler traces are best reserved for situations where you need to do diagnostics on the server or where you have enough CPU and storage capacity to collect lots of details.

References:

- ASTrace – a tool to collect profiler traces into SQL Server for further analysis: http://msftasprodsamples.codeplex.com/wikipage?title=SS2005%21Readme%20for%20ASTrace%20Utility%20Sample&referringTitle=Home

3.2.4 Data Collection Checklist

You can use the following checklist to make sure you have covered the basic data collection needs of the server:

- ✓ Windows Server-specific Performance Monitor counters
- ✓ Analysis Services-specific Performance Monitor counters
- ✓ Data Management Views collected:
  - $system.discover_commands
  - $system.discover_sessions
  - $system.discover_connections
  - $system.discover_memoryusage
  - $system.discover_locks

3.2.5 Monitoring and Configuring Memory

To discover the best memory configuration for your Analysis Services instance, you need to collect some data about the typical usage of the system.

First of all, you should start up the server with Analysis Services disabled and make sure that all the other services you need for normal operations in a state that is typical for your standard server configuration. What typical is varies by environment –you are looking to measure how much memory will be available to Analysis Services after the operating system and other services (such as virus scanners and monitoring tools) have taken their share. Note down the value of the Performance Monitor counter **Memory/Available Bytes**.

Secondly, start Analysis Services and run a typical user workload on the server. You can for example use queries from your test harness. For this test, also make sure **PreAllocate** is set to 0. While the workload is running, run the following query.

```
SELECT * FROM $SYSTEM.DISCOVER_MEMORYUSAGE
```

If you have any long-running, high-memory consuming queries, you should also measure how much memory they consume while they execute.

Copy the data into an Excel spreadsheet for further analysis. You can also use the CodePlex project ResMon cube (see reference section) to periodically log snapshots of this DMV and browse memory usage trends summarized in a cube.

With the data collected in the previous section, you can define some values needed to set the memory:

[Physical RAM] = The total physical memory on the box

[Available RAM] = The value of the Performance Monitor counter **Memory/Available** Bytes as noted down earlier.

[Non Shrinkable Memory] = The sum of the **MemoryUsed** column from $SYSTEM.DISCOVER_MEMORYUSAGE where the column **Shrinkable** is **False.**

[Valuable Objects] = The sum of all objects that you want to reserve memory for from $SYSTEM.DISCOVER_MEMORYUSAGE where column **Shrinkable** is **True**. For example, you will most likely want to reserve memory for all dimensions and attributes. If you have a small cube, this value may simply be everything that is marked as shrinkable.

[Worst Queries Memory] = The amount of memory used by the all the most demanding queries you expect will run concurrently. You can use the DMV **$system.discover_memoryusage** to measure this on a known workload.

With the preceding values you can calculate the memory configuration for Analysis Services:

LowMemoryLimit = [Non Shrinkable Memory] + [Valuable Objects] / [Total RAM]But keep a gap of at least 20 percent between this value and **HighMemoryLimit** to allow the memory cleaners to release memory at a good rate.

HighMemoryLimit = ([Available RAM] - [Worst Queries Memory]) / [Total RAM]

HardMemoryLimit = [Available RAM] / [Total RAM]

LimitSystemFileCacheMB = [Available RAM] - LowMemoryLimit

Note that it may not always be possible to measure all the components that make up these equations. In such case, your best bet is to guesstimate them. The idea behind this method is that Analysis Services will always hold on to enough memory to keep the valuable objects in memory. What is valuable depends on your particular setup and query set. The rest of the available memory is shared between the file system cache and the Analysis Services caches; if you use the operating systems memory usage optimizations, the ideal balance between Analysis Services and the file system cache is adjusted dynamically.

If Analysis Services coexists with other services on the machine, take their maximum memory consumption into consideration. When you calculate **[Available RAM]**, subtract the maximum memory use by other large memory consumers, such as the SQL Server relational engine. Also, make sure those other memory consumers have their maximum memory settings adjusted to allow space for Analysis Services.

The following diagram illustrates the different uses of memory.

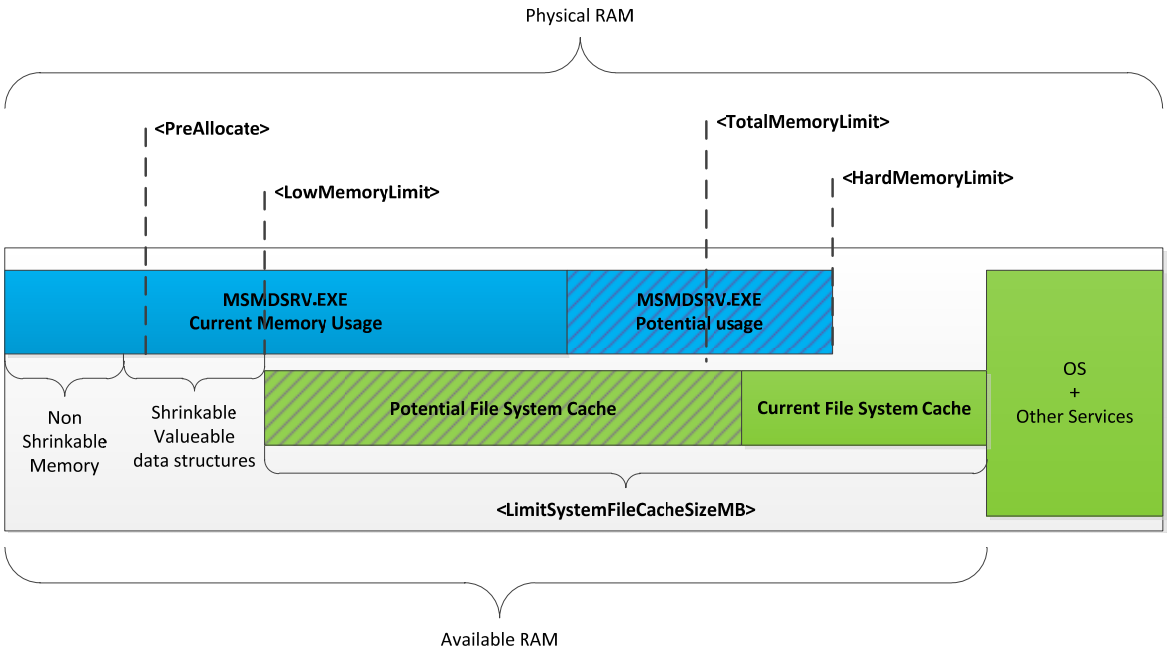


Figure 56 - Memory Settings

References:

- CodePlex ResMon project - http://sqlsrvanalysissrvcs.codeplex.com/wikipage?title=ResMon%20Cube%20Documentation

3.2.6 Monitoring and Tuning I/O and File System Caches

To tune the Analysis Services I/O subsystem, it is helpful to understand how a user query is translated into I/O requests. The following diagram illustrates the caching and I/O architecture of Analysis Services.

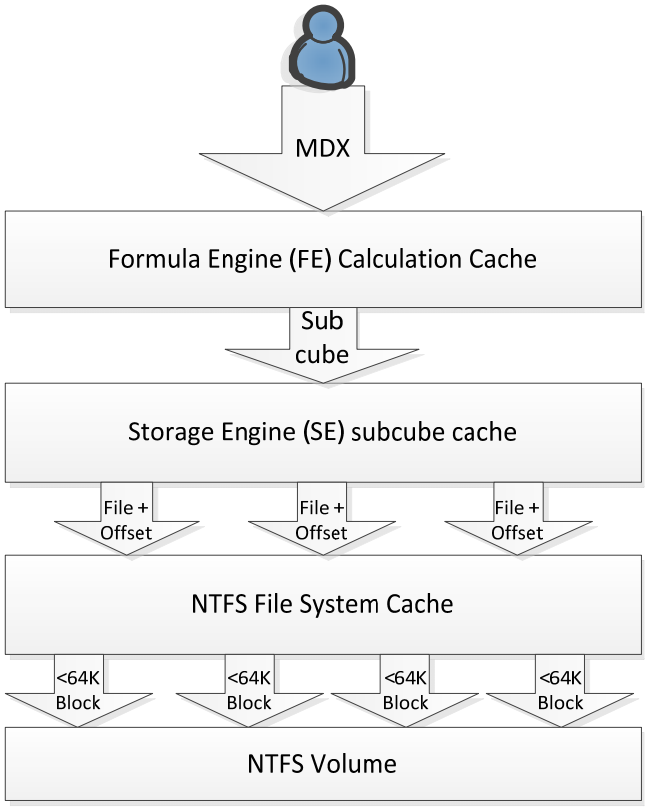


Figure 57 - I/O stack

From the perspective of the I/O system, the storage engine requests data from an offset in a file. The building blocks of the storage used by Analysis Services are the files used to store dimension and fact data. Dimension data is typically cached by the storage engine. Hence, the most frequently requested files from the storage system are measure group data, and they have the following file names: ***.fact.data, *.aggs.rigid.data** and ***.agg.flex.data**. Because Analysis Services uses buffered I/O, frequently requested blocks in files are typically retained by the NTFS file system cache. Note that this means that the memory used for the file system caching of data comes from outside the Analysis Services working set.

When a block from a file is requested by Analysis Services, the path taken by the operating system depends on the state of the cache.

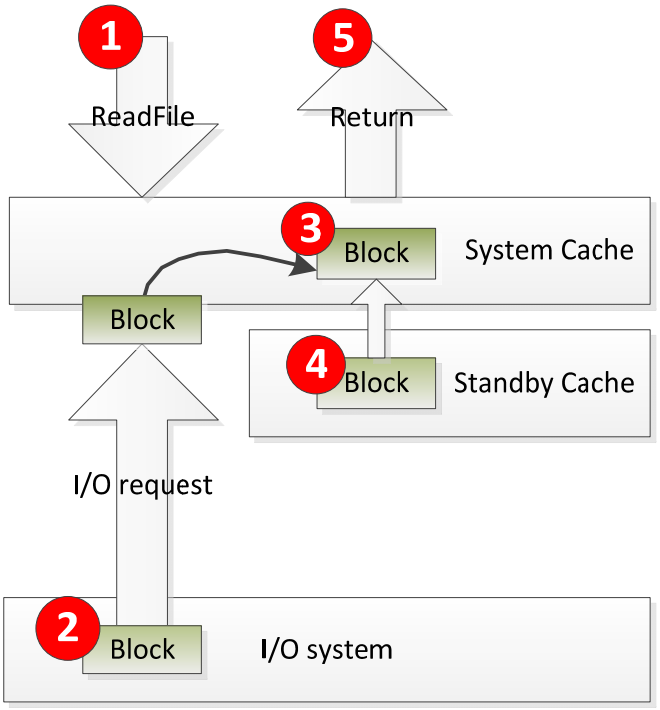


Figure 58 - Accessing a file in the NTFS cache

In the preceding illustration, Analysis Services executes the ReadFileEx call to the Windows API.

- If the block is not in the cache, an I/O request is issued (2), the file is put in the cache (3), and the data is returned to Analysis Services (5).
- If he block is already in the cache (3), it is simply returned to Analysis Services (5).
- If a block is not frequently accessed or if there is memory pressure, the NTFS file system cache may choose to place the block in the Standby Cache (4) and eventually remove the file from memory. If the file is requested before it is removed, it is brought back into the System Cache (3) and then returned to Analysis Services (5).
    - Note that in Windows Server 2003, the file is simply removed from the system cache, bypassing the standby cache.

3.2.6.1 System Cache vs. Physical I/O

Because Analysis Services uses the NTFS cache, not every I/O request reaches the I/O subsystem. This means that even with a clear storage engine and formula engine cache, query performance will still vary depending on the state of the file system cache. In the worst case, a query will run on a clean file system cache and every I/O request will hit the physical disk. In the best case, every I/O request will be served by the memory cache. This depends on the amount of data requested by the query.

It is useful to know the ratio between the I/O issues to the disk and the I/O served by the file system cache. This helps you capacity plan for growing cube. Small cubes, less than the size of the server memory, are typically served mostly from the file system cache. However, as cubes grow larger and

exceed the file system cache size, you will eventually have to assist cube performance with a fast disk system. Measuring the current cache hit ratio helps shed light on this.

To measure the I/O served by the file system cache, use Performance Monitor to measure **System:File Read Bytes/sec** and **System File Read Operations/sec**. These counters will give you an estimate of the number of I/O operations and file bytes that are served from memory. By comparing this with **Logical Disk / Disk reads /sec** you can measure the ratio between memory buffered and physical I/O. As the amount of data in the cube grows, you will often see that the amount of disk access begins to grow in proportion to memory access. By extrapolating from historical trends, you can then decide what the best ratio between IOPS and RAM sizes should be going forward.

To measure the total memory consumption of Analysis Services, you will also have to take the file system cache into account. The total memory used by analysis services is the sum of:

- **Process – msmdsrv.exe / Private Bytes** – The memory used by Analysis Services itself
- **Memory – Cache Bytes** – The files currently live in cache
- **Memory – Standby Cache Normal Priority Bytes** – The files that can be freed from the cache

However, note that the memory counters also measure other files' caches in the NTFS file system cache. If you are running Analysis Services together with other services on the machine, the file system counter may not accurately reflect the caches used by Analysis Services.

3.2.6.2 TempDir Folder

When memory is scarce on the server, Analysis Services spills operations to disk. Examples of such operation are:

- Dimension processing
    - ByTable processing commands that don't fit memory
    - Processing of large dimension where the hash tables created exceed available memory
- Aggregation processing
- ROLAP dimension attribute stores

Dimension processing: To optimize dimension processing, we recommend that you use the techniques described later in this document to avoid spills and speed up processing. **ByTable** should generally only be used if you can keep the dimension data in memory during processing.

Aggregation processing: During cube processing, aggregation buffers described in configuration section determine the amount of memory that is available to build aggregations for a given partition. If the aggregation buffers are too small, Analysis Services supplements the aggregation buffers with temporary files. To monitor any temporary files used during aggregation, review **MSOLAP:Proc Aggregations\Temp file bytes written/sec**. You should try to design your cube in such a way that memory spills to temp files does not occur, that is, keep the temp bytes written at zero. There are several techniques available to avoid memory spills during aggregation processing:

- Only create aggregations that are small enough to fit in memory during processing.
- Process fewer partitions in parallel.
- Add more memory to the machine or allocating more memory to Analysis Services.

It is generally a good idea to split **ProcessData** and **ProcessIndex** operations into two different processing jobs. **ProcessData** typically consumes less memory than **ProcessIndex** and you run many **ProcessData** operations in parallel. During **ProcessIndex**, you can then decrease concurrency if you are short on memory, to avoid the disk spill.

ROLAP dimensions: In general, you should try to avoid ROLAP dimensions; MOLAP stores are much faster for dimension access. However, the requirement for ROLAP dimensions is often driven by a lack of sufficient memory to hold the attribute stores requires for drillthrough actions, which returns data using a degenerate ROLAP dimension. If this is your scenario, you may not be able to avoid spills to the **TempDir** folder.

If you cannot design the cube to fit all operations in memory, spilling to **TempDir** may be your only option. If that is the case, we recommend that you place the **TempDir** folder on a fast LUN. You can either use a LUN backed by caches (for example in a SAN) or one that sits on fast storage – for example, NAND devices.

3.2.7 Monitoring the Network

One way to easily monitor network throughput is through a performance monitor trace. Windows Performance Monitor requests the current value of performance counters at specified time intervals and logs them into a trace log (.blg or .csv).

The following counters will help you isolate any problems in the network layer.

Processing: Rows read/sec – this is the rate of rows read from your data source. This is one of the most important counters to monitor when you measure your throughput from Analysis Services to your relational data source. When you monitor this counter, you will want to view the trace using a line chart, as it gives you a better idea of your throughput relative to time. It is reasonable to expect tens of thousands of rows per second per network connection to a SQL Server data source. Third-party sources may experience significantly slower throughput.

Bytes received/sec and **Bytes sent/sec** - These two counters enable you to measure how far you are from the theoretical NIC speeds. It allows capacity planning for faster NIC.

TCPv4 and TCPv6 Segments/sec and **Segments retransmitted/sec** – These counters enable you to discover unstable network connections. The ratio between segments retransmitted and segments for both TCPv4 and TCPv6 should not exceed 3-4 percent on a stable network.

3.3 Security and Auditing

Cubes often contain the very heart of your business data – the source of your decision making. That means that you have to consider potential attack vectors that a malicious user or intruder could use to acquire access to your data. The following table provides an overview of the potential vulnerabilities and the countermeasures you can take. Note that most environments will not need all of these countermeasures – it depends on the data security and on the attack vectors that are possible on your network.

Attack vector	Countermeasure
Other services listening on the server	Firewall all ports other than those used by Analysis Services
Sniff TCP/IP packets to client **Sniff TCP/IP packets during processing**	Configure IPsec or SSL encryption Configure SQL Server Protocol Encryption
Steal physical media containing cubes	Encrypt file system used to store cubes
Compromise the service account	Configure minimum privileges to the service account Require a strong password
Access the file system as a logged-in user	Secure file system with minimal privileges

A full treatment of security configuration is outside the scope of this book, but the following sections provide references that serve as a starting point for further reading and give you an overview of the options available to you.

3.3.1 Firewalling the Server

By default, Analysis Services communicates with clients on port 2383. If you want to change this, access the properties of the server.

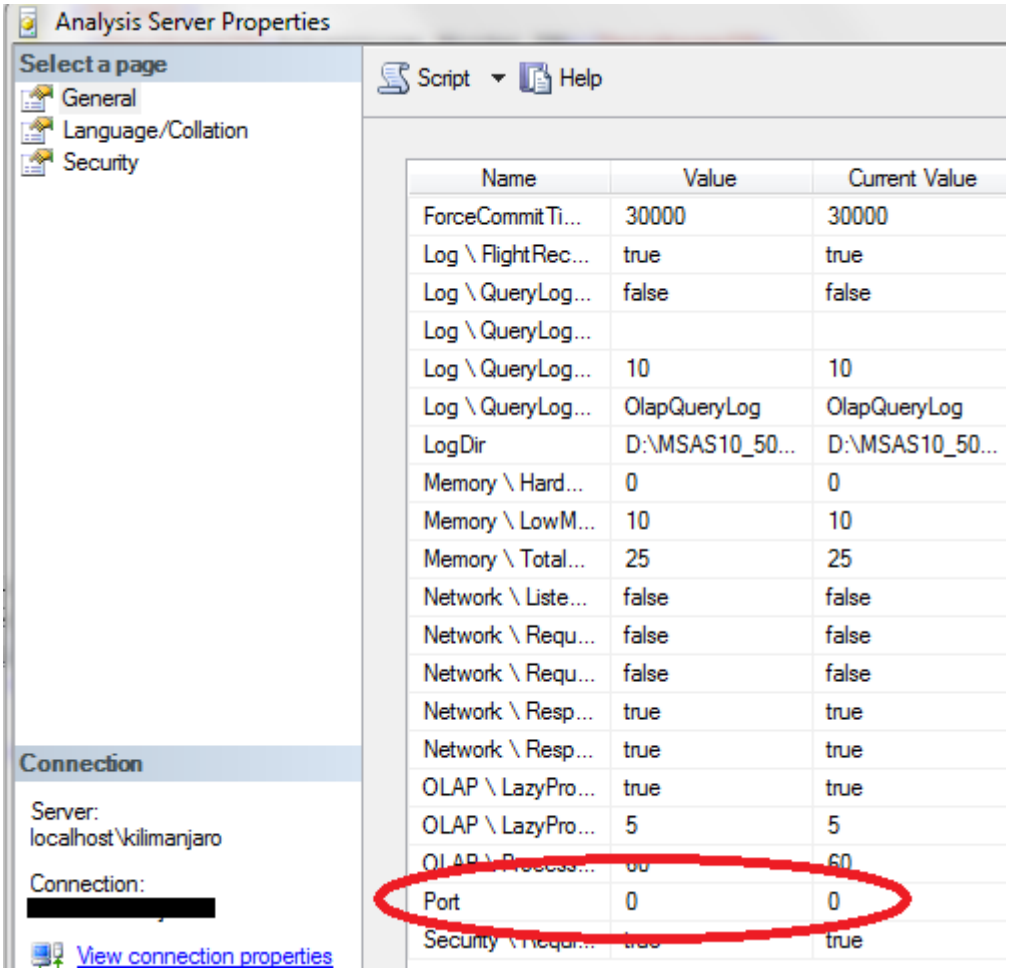


Figure 59 - Changing the port used for Analysis Services

Bear in mind that you will need to open the port assigned here in your firewall for TCP/IP traffic. If you leave the default value of 0, Analysis Services will use port 2382.

If you are using named instances, your client application may also need to access the SQL Server browser service. The browser service allows clients to resolve instance names to port numbers, and it listens on TCP port 2382. Note that it is possible to configure the connection string for Analysis Services in such a way that you will not need the browser service port open. To connect directly to the port that Analysis Services is listening on, use this format **[Server name}:[Port].** For example, to connect to an instance listening on port 2384 on server MyServer, use **MyServer:2384**.

Analysis Services can be set up to use HTTP to communicate with clients. If you choose this option, follow the guidelines for configuring Microsoft Internet Information Services. In this case, you will typically need to open either port 80 or port 443.

References:

- How to: Configure Windows Firewall for Analysis Services Access - http://msdn.microsoft.com/en-us/library/ms174937.aspx
- Resolving Common Connectivity Issues in SQL Server 2005 Analysis Services Connectivity Scenarios - http://msdn.microsoft.com/en-us/library/cc917670.aspx
    - Also applies to SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services
- Configuring HTTP Access to SQL Server 2005 Analysis Services on Microsoft Windows 2003 - http://technet.microsoft.com/en-us/library/cc917711.aspx
    - Also applies to SQL Server 2008 Analysis Services, SQL Server 2008 R2 Analysis Services, Windows Server 2008, and Windows Server 2008 R2
- Analysis Services 2005 protocol - XMLA over TCP/IP - http://www.mosha.com/msolap/articles/as2005_protocol.htm

3.3.2 Encrypting Transmissions on the Network

Analysis Services communicates in a compressed and encrypted format on the network. You may still want to use IPsec to restrict which machines can listen in on the network traffic, but the communication channel itself is already encrypted by Analysis Services.

If you have configured Analysis Services to communicate over HTTP, the communication can be secured using the SSL protocol. However, be aware that you may have to acquire a certificate to use SSL encryption over public networks. Also, note that SSL encryption normally uses port 443, not port 80, to communicate. This difference may require changes in the firewall configuration. Using the HTTP protocol also allows you to run secured lines to parties outside the corporate network – for example, in an extranet setup.

Depending on your network configuration, you may also be concerned about network packets getting sniffed during cube processing. To avoid this, you again have to encrypt traffic. Again, you can use IPsec to achieve this. Another option is to use protocol encryption in SQL Server, which is described in the references.

References:

- How to configure SQL Server 2008 Analysis Services and SQL Server 2005 Analysis Services to use Kerberos authentication - http://support.microsoft.com/kb/917409
- Windows Firewall with Advanced Security: Step-by-Step Guide: Deploying Windows Firewall and IPsec Policies - http://www.microsoft.com/downloads/en/details.aspx?FamilyID=0b937897-ce39-498e-bb37-751c00f197d9&displaylang=en
- How To Configure IPsec Tunneling in Windows Server 2003 - http://support.microsoft.com/kb/816514
- How to enable SSL encryption for an instance of SQL Server by using Microsoft Management Console - http://support.microsoft.com/kb/316898

3.3.3 Encrypting Cube Files

Some security standards require you to secure the media that the data is stored on, to prevent intruders from physically stealing the data. If you have MOLAP cubes on the server, media security may be a concern to you. Because Analysis Services, unlike the relational engine, does not ship with native encryption of MOLAP cube data, you must rely on encryption outside the engine. You can use Windows File System encryption (Windows Server 2003) or BitLocker (on Windows Server 2008 and Windows Server 2008 R2) to encrypt the drive used to store cubes. Be careful, though; encrypting MOLAP data can have a big impact on performance, especially during processing and schema modification of the cube. We have seen performance drops of up to 50 percent when processing dimensions on encrypted drives – though less so for fact processing. Weigh the requirement to encrypt data carefully against the desired performance characteristics of the cube.

References:

- Encrypting File System in Windows XP and Windows Server 2003 - http://technet.microsoft.com/en-us/library/bb457065.aspx
- BitLocker Drive Encryption - http://technet.microsoft.com/en-us/library/cc731549%28WS.10%29.aspx

3.3.4 Securing the Service Account and Segregation of Duties

To secure the service account for Analysis Services, it is useful to first understand the different security roles that exist at a server level.

Theservice account is the account that runs the msmdsrv.exe file. It is configured during installation and can be changed using SQL Server Configuration Manager. The service account is the account used to access all files required by Analysis Services, including MOLAP stores. For your convenience, a local group **SQLServerMSASUser$<Instance Name>**iscreated that has the right privileges on the binaries required to run Analysis Services. If you configure the service account during installation, the account will automatically be added to this group. If you later choose to change the service account in SQL Server Configuration Manager, you must manually update the group membership.

The server administrator role has privileges to change server settings, and to create, back up, restore, and delete databases. Members of this account can also control security on all databases in the instance. It is the DBA role of the instance and almost equivalent to the **sysadmin** role for SQL Server. Membership in the server administrator role is configured in the properties of the server in SQL Server Management Studio.

In a secure environment, you should run Analysis Services under a dedicated service account. You can configure this during installation of the service.

If your environment requires segregation of duties between those who configure the service account and those who administer the server, you need to make some changes to the msmdsrv.ini file:

- By default, local administrators are members of the server administrator role. To remove this association, set **<BuiltinAdminsAreServerAdmins>** to 0.
- By default, the service account for Analysis Services is a member of the server administrator role. Remove this association by setting **<ServiceAccountIsServerAdmin>** to 0.

However, keep in mind that a local administrator could still access the msmdsrv.ini file and alter the changes you have made, so you should audit for this possibility.

3.3.5 File System Permissions

As mentioned earlier, the Analysis Services installer will create the Windows NT group**SQLServerMSASUser$<Instance Name>**and add the service account to this group. At install time, this group is also granted access to the **Data**, backup, log, and **TempDir** folders.

If you at a later time add more folders to the instance to hold data, backups, or log files, you will need to grant the **SQLServerMSASUser$<Instance Name>**groupread and write access to the new folders. If you move the TempDir folder, you will also need to assign the group read/write permission to the new location. No other users need permissions on these folders for Analysis Services to operate.

In the configuration of the server you will also find the **AllowedBrowsingFolders** setting. This setting, a pipe-separated list of directories, limits the visible folders that server administrators can see when they configure storage locations Analysis Services data. **AllowedBrowsingFolders** is *not* a security feature, because server administrator can change the values in it to reflect anything that the service account can see. However, it does serve as a convenient filter to display only a subset of the folders that the service account can access. Note that server administrators cannot directly access the data visible through the **AllowedBrowsingFolders**setting, but they can write backups in the folders, restore form the folders, move **TempDir**there, and set the storage location of databases, dimensions, and partitions to those folders.

3.4 High Availability and Disaster Recovery

High availability and disaster recovery are not robustly integrated as part of Analysis Services for large-scale cubes. In this section, you learn about the combination of methods you can use to achieve these goals within an enterprise environment.

To ensure disaster recovery, use the built-in Backup/Restore method, Analysis Services Synch, or Robocopy to ensure that you have multiple copies of the database. You can achieve high availability by deploying your Analysis Services environment onto clustered servers. Also note that by using a combination of multiple copies, clustering, and scale out (section 7.4), you can achieve both high availability and disaster recovery for your Analysis Services environment. In a large-scale environment, the scale-out method generally provides the best of use hardware resources while also providing both backup and high availability.

3.4.1 Backup/Restore

Cubes are data structures on disk and as such, they contain information that you may want to back up.

Analysis Services backup – Analysis Services has a built-in backup functionality that generates a single backup file from a cube database. Analysis Services backup speeds have been significantly improved in SQL Server 2008 and for most solutions, you can simply use this built-in backup and restore functionality. As with all backup solutions, you should of course test the restore speed.

SAN based backup – If you use a SAN, you can often make backups of a LUN using the storage system itself. This backup process is transparent to Analysis Services and typically operates on the LUN level. You should coordinate with your SAN administrator to make sure the correct LUNs are backed up, including all relevant data folders used to store the cube. You should also make sure that the SAN backup utility uses a VDI/VSS compliant tool to call to Windows before the backup is taken. If you do not use a VDI/VSS tool, you risk getting chkdsk errors when the LUN is restored.

File based backup copy – In SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services, you can attach database if you have a copy of the data folder the database resides in. This means that a detached copy of files in a stale cube can serve as a backup of the database. This option is available only with SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services. You can use the same tools that you use for scale-out cubes (for example, Robocopy or NiceCopy). Restoring in this case means copying the files back to the server and attaching the database.

Don't backup – While this may sound like a silly option, it does have merit in some disaster recovery scenarios. Cubes are built on relational data sources. If these sources are guaranteed to be available and securely backed up, it may be faster to simply reprocess the cube than to restore it from backup. If you use this option, make sure that no data resides in the cubes that cannot be re-created from the relational source (for example, data loaded via the SQL Server Integration Services Analysis Services destination). Of course, you should also make sure that the cube structure itself is available, including all aggregation and partition designs that have been changed on the production server from the standard deployment script.

This is particularly true for ROLAP cubes. In this case (as in all the previous scenarios), you should *always* maintain an updated backup of your Analysis Services project that allows for a redeployment of the solution (with the subsequent required processing), in case your backup media suffers any kind of physical corruption.

3.4.1.1 Synchronization as Backup

If you are backing up small or medium-size databases, the Analysis Services synchronization feature is an operationally easy method. It synchronizes databases from source to target Analysis Service servers. The process scans the differences between the two databases and transfers only the files that have been modified. There is overhead associated with scanning and verifying the metadata between the two different databases, which can increase the time it takes to synchronize. This overhead becomes increasingly apparent in relation to the size and number of partitions of the databases.

Here are some important factors to consider when you work with synchronization:

- At the end of the synchronization process, a Write lock must be applied to the target server. If this lock is queued up behind a long running query, it can prevent users from querying the target database.
- During synchronization, a read commit lock is applied to the source server, which prevents users from processing new data but allows multiple synchronizations to occur from the same source server.
- For enterprise-size databases, the synchronization method can be expensive and low-performing. Because some operations are single-threaded (such as the delete operation), having high-performance servers and disk subsystems may not result in faster synchronization times. Because of this, we recommended that for enterprise size databases you use alternate methods, such as Robocopy (discussed later in this book) or hardware-based solutions (for example, SAN clones and snapshots).
- When executing multiple synchronization operations against a single server, you may get the best performance (by minimizing lock contentions) by queuing up the synchronization requests and then running them serially.
- For the same locking contention reasons, plan your synchronizations for down times, when querying and processing are not occurring on the affected servers. While there are locks in place to prevent overwrites, processes such as long-running queries and processing may prevent the synchronization from completing in a timely fashion.

For more information, see the “Analysis Services Synch Method” section in Analysis Services Synchronization Best Practices technical note (http://sqlcat.com/technicalnotes/archive/2008/03/16/analysis-services-synchronization-best-practices.aspx).

3.4.1.2 Robocopy

The basic principle behind the Robocopy method is to use a fast copy utility, such as Robocopy, to copy the OLAP data folder from one server to another. By following the sample script noted in the technical note, Sample Robocopy Script to customer synchronize Analysis Services databases (http://sqlcat.com/technicalnotes/archive/2008/01/17/sample-robocopy-script-to-customer-synchronize-analysis-services-databases.aspx), you can perform delta copies (that is, copy only the data that has changed) of the OLAP data folder from one source to another target source in parallel. This method is often employed in enterprise environments because the key factor here is fast, robust data file transfer.

However, the key disadvantages of using this approach include:

- You must stop and then restart (in SQL Server 2005 Analysis Services) or detach (in SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services) your Analysis Services servers when you use a fast copy utility.
- You cannot use the synchronization feature and Robocopy together.

- This approach makes the assumption that there is only one database on that instance; this is usually okay because of its size.
- Some functionality is lost if you use this method, including, but not limited to, writeback, ROLAP, and real-time updates.

Nevertheless, this is an especially effective method for query server / processing server architectures that involve only one database per server.

References:

- Scale-Out Querying with Analysis Services (http://sqlcat.com/whitepapers/archive/2007/12/16/scale-out-querying-with-analysis-services.aspx)

3.4.2 Clustered Servers

Analysis Services can participate in a Windows failover cluster using a shared disk subsystem, such as a SAN. When provisioning storage for a cluster, there are some things you should be aware of.

It is not possible to use dynamic disk stripes. In this case, you have two options for spreading the cubes over all available LUN:

- Have your SAN administrator configure a mega-LUN.
- Selectively place partitions on different LUNs and then manually balance the load between these LUNs.

Mega LUN: Most SANs today can stripe multiple, smaller LUNs into a large mega-LUN. Talk to your SAN administrator about this option.

Selective placement: If you need to use multiple LUNs for a single cube, you should try to manually balance I/O traffic between these LUNs. One way to achieve this is to partition the cube into roughly equal-sized slices based on a dimension key. Very often, the only way to achieve roughly equal balance like this is to implement a two-layer partitioning on both date and the secondary, balancing key. For more information, see the Repartitioningsection in Part 1.

3.5 Diagnosing and Optimizing

This section discusses how to troubleshoot problems and implement changes that can be made transparently in the cube structures to improve performance. Many of these changes are already documented in Part 1, but some additional considerations apply from an operations perspective.

3.5.1 Tuning Processing Data

During **ProcessData** operations, Analysis Services uses the processing thread pool for worker threads used by the storage engine.

ProcessData operations use three parallel threads to perform the following tasks:

- Query the data source and extract data
- Look up dimension keys in dimension stores and populate the processing buffer
- Write the processing buffer to disk when it is full

You can usually increase throughput of **ProcessData** by tuning the network stack and processing more partitions in parallel. However, you may still benefit from tuning the process pool.

To optimize these settings for the **ProcessData** phase, check your Performance Monitor counter on the object **MSOLAP: Threads** and use the following table for guidance.

Situation	Action
Processing pool job queue length> 0 and **Processing pool idle threads** = 0 for longer periods during processing.	Increase **ThreadPool\Process\MaxThreads** and then retest.
Both **Processing pool job queue length** > 0and **Processing pool idle threads** > 0 at same time during processing.	Decrease **CoordinatorExecutionMode** (for example, change it from -4 to -8) and then retest.

You can use the **Processor –% Processor Time – Total** counter as a rough indicator of how much you should change these settings. For example, if your CPU load is 50 percent, you can double **ThreadPool\Process\MaxThreads** to see whether this also doubles your CPU usage. It is possible to get to 100 percent CPU load in a system without bottlenecks, though you may want to leave some headroom for growth. Keep in mind that increased parallelism of processing also has an effect on queries running at the system. Ideally, use a separate processing server or a dedicated processing time window where no one is querying Analysis Services. If this is not an option, as you dedicate more CPU power and threads to processing, less CPU will be used for query responses. Because processing consumes threads from the same pool as query subcube requests, you should also be careful that you don’t run the process thread pool dry if you process and query at the same time. If you are processing the cubes during a set processing window with no users on the box, this will of course not be an issue.

References:

- SQL Server 2005 Analysis Services (SSAS) Server Properties (http://technet.microsoft.com/en-us/library/cc966526.aspx)

3.5.1.1 Optimizing the Relational Engine

In addition to looking at Analysis Services configurations, and settings, you can also look at the relational engine when you are planning improvements to your Analysis Services installation. This section focuses on working with relational data from SQL Server 2005, SQL Server 2008, or SQL Server 2008 R2. Although Analysis Services can be used with any OLE DB or .NET driver enabled database (such as Oracle or Teradata), the advice here may not apply to such third-party environments. However, if you are a third-party DBA, you may be able to translate the techniques discussed here to similar ones in your own environment.

3.5.1.1.1 Relational Indexing for Partition Processing

While you generally want each cube partition to touch at most one relational partition, the reverse is not true. It is perfectly viable to have to have more than one cube partition accessing the same relational partition. As an example, a relational source that is partitioned by year with a cube that is partitioned by month can still provide optimal processing performance.

If you do not have a one-to-one relationship between relational and cube partitions, you generally want an index to support the fact processing query. The best choice of index for this purpose is a clustered index; if your load strategy allows you to maintain such an index, this is what you should aim for.

When a partition processing query is supported by an index the plan should look like this.

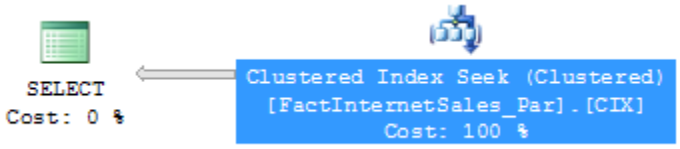


Figure 17 Supporting Measure Group processing with an index

References:

- Top 10 SQL Server 2005 Performance Issues for Data Warehouse and Reporting Applications (http://sqlcat.com/top10lists/archive/2007/11/21/top-10-sql-server-2005-performance-issues-for-data-warehouse-and-reporting-applications.aspx)
- Ben-Gan, Itzik and Lubor Kollar, *Inside Microsoft SQL Server 2005: T-SQL Querying.* Redmond, Washington: Microsoft Press, 2006.

3.5.1.1.2 Relational Indexing for Dimension Processing

If you follow a dimensional star schema design (which we recommend for large cubes), most dimension processing queries should run relatively fast and take only a tiny portion of the total cube processing time. But if you have very large dimensions with millions of rows or dimensions with lots of attributes, some performance can be still be gained by indexing the relational dimension table. To understand the best indexing strategy, it is useful to know which form dimensions processing queries take. The number of queries generated by Analysis Services depends on the attribute relationships defined in the dimension. For each attribute in the dimension, the following query is generated during processing.

> **SELECT DISTINCT**<attribute>, [<related attribute> […n]]
> **FROM**<dimension table>

Consider the following example dimension, with **CustomerID** as the key attribute.

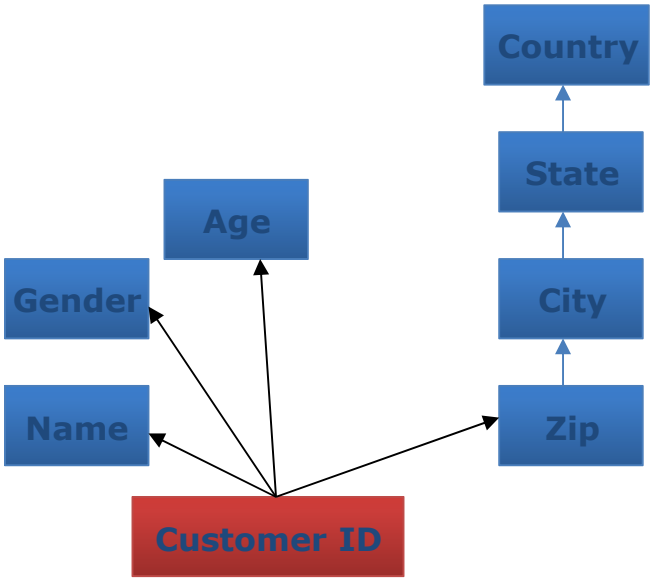


Figure 60 – Example Customer Dimension – Attribute relationships

The following queries are generated during dimension processing.

```
SELECTDISTINCT Country FROM Dim.Customer
SELECTDISTINCTState, Country FROM Dim.Customer
SELECTDISTINCT City,StateFROM Dim.Customer
SELECTDISTINCT Zip, City FROM Dim.Customer
SELECTDISTINCT Name FROM Dim.Customer
SELECTDISTINCT Gender FROM Dim.Customer
SELECTDISTINCT Age FROM Dim.Customer
SELECTDISTINCT CustomerID, Name, Gender, Age, Zip FROM Dim.Customer
```

The indexing strategy you apply depends on which attribute you are trying to optimize for. To illustrate a tuning process, this section walks through some typical cases.

Key attribute: the key attribute in a dimension, in this example **CustomerID,** can be optimized by creating a unique, clustered index on the key. Typically, using such a clustered index is also the best strategy for relational user access to the table – so your DBA will be happy if you do this. In this example, the following index helps with key processing.

```
CREATEUNIQUECLUSTEREDINDEX CIX_CustomerID ON Dim.Customer(CustomerID)
```

This will create the following, optimal query plan.

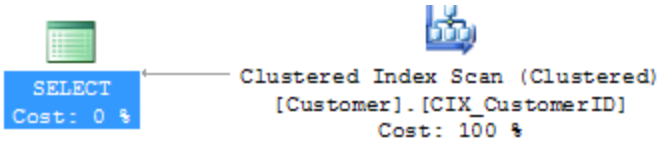


Figure 61 - A good key processing plan

High cardinality attributes: For high cardinality attributes, like **Name**, you need a nonclustered index. Notice the DISTINCT in the SELECT query generated by Analysis Services.

```
SELECTDISTINCT Name FROM Dim.Customer
```

DISTINCT generally forces the relational engine to perform a sort to remove duplicates in the returned dataset. The sort operation results in a plan that looks like this.

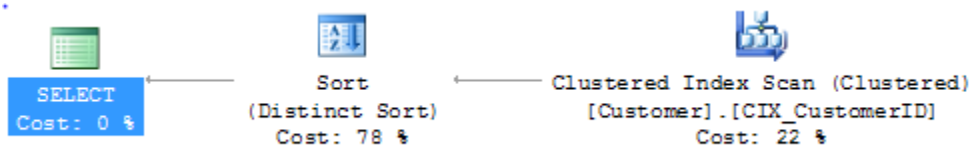


Figure 20 - Expensive sort plan during dimension processing

If this plan takes a long time to run, which could be the case for large dimension, consider creating an index that helps remove the sort. In this example, you can create this index.

```
CREATEINDEX IX_Customer_Name ON Dim.Customer(Name)
```

This index generates the following, much better plan.

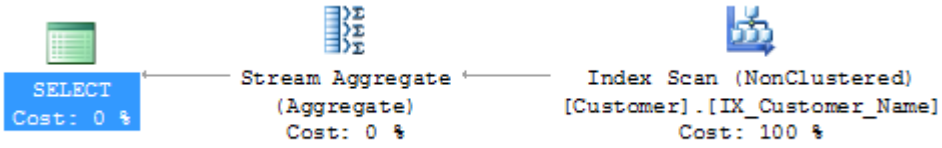


Figure 62 - A fast high cardinality attribute processing plan

Low cardinality attributes: For attributes that are part of a large dimension but low granularity, even the preceding index optimization may result in an expensive plan. Consider the **City** attribute in the customer dimension example. There are very few cities compared to the total number of customers in the dimension. For example, you want to optimize for the following query.

```
SELECTDISTINCT City,StateFROM Dim.Customer
```

Creating a multi-column index on **City** and **State** removes the sort operation required to return DISTINCT rows – resulting in a plan very similar to the optimization performed earlier with the **Name** attribute. This is better than running the query with no indexed access. But it still results in touching one row per customer – which is far from optimal considering that there are very few cities in the table.

If you have SQL Server Enterprise, you can optimize the SELECT query even further by creating an indexed view like this.

```
CREATEVIEW Dim.Customer_CityState
WITHSCHEMABINDING
AS
SELECT City,State,COUNT_BIG(*)AS NumRows FROM Dim.Customer
GROUPBY City,State

GO

CREATEUNIQUECLUSTEREDINDEX CIX_CityState
ON Dim.Customer_CityState(City,State)
```

SQL Server maintains this aggregate view on disk, and it stores only the distinct values of **City** and **State** – exactly the data structure you are looking for. Running the dimension processing query now results in the following optimal plan.

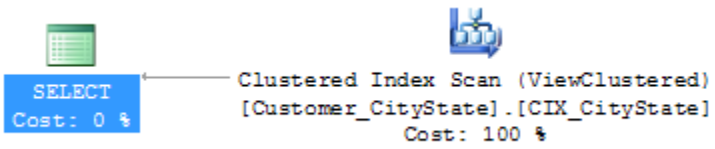


Figure 63 - Using indexed views to optimize for low-cardinality attributes

3.5.1.1.3 Overoptimizing and Wasting Time

It is possible to tune the relational engine to cut down time on processing significantly, especially for partition processing and large dimensions. However, bear in mind that every time you add an index to a table, you add one more data structure that must be maintained when users modify rows in that table. Relational indexing, like aggregation design in a cube and much of BI and data warehousing, is tradeoff between data modification speed and user query performance. There is typically a sweet spot in this space that will depend on your workload. Different people have different skills, and the perception of where that sweet spot lies will change with experience. As you get closer to the optimal solution, increased the tuning effort will often reach a point of diminishing returns where the speed of the system moves asymptotically towards the optimum balance. Monitor your own tuning efforts and try to understand when you are getting close to that flatline behavior. As tempting as full tuning exercises can be to the technically savvy, not every system needs benchmark performance.

3.5.1.1.4 Using Index FILLFACTOR = 100 and Data Compression

If page splitting occurs in an index, the pages of the index may end up less than 100 percent full. The effect is that SQL Server will be reading more database pages than necessary when scanning the index.

You can check for index pages are not full by querying the SQL Server DMV **sys.dm_db_index_physical_stats**. If the column **avg_page_space_used_in_percent** is significantly lower than 100 percent, a FILLFACTOR 100 rebuild of the index may be in order. It is not always possible to rebuild the index like this, but this trick has the ability to reduce I/O. For stale data, rebuilding the indexes on the table is often a good idea before you mark the data as read-only.

In SQL Server 2008 you can use either row or page compression to further reduce the amount of I/O required by the relational database to serve the fact process query. Compression has a CPU overhead, but reduction in I/O operations is often worth it.

References:

- Data Compression: Strategy, Capacity Planning and Best Practices - http://msdn.microsoft.com/en-us/library/dd894051%28v=sql.100%29.aspx

3.5.1.1.5 Eliminating Database Locking Overhead

When SQL Server scans an index or table, page locks are acquired while the rows are being read. This ensures that many users can access the table concurrently. However, for data warehouse workloads, this page level locking is not always the optimal strategy – especially when large data retrieval queries like fact processing access the data.

By measuring the Perfmon counter **MSSQL:Locks – Lock Requests / Sec** and looking for **LCK** events in **sys.dm_os_wait_stats**, you can see how much locking overhead you are incurring during processing.

To eliminate this locking overhead, you have three options:

- Option 1: Set the relational database in Read Only mode before processing.
- Option 2: Build the fact indexes with `ALLOW_PAGE_LOCKS=OFF` and `ALLOW_ROW_LOCKS=OFF`.
- Option 3: Process through a view, specifying the `WITH (NOLOCK)` or `WITH (TABLOCK)` query hint.

Option 1 may not always fit your scenario, because setting the database to read-only mode requires exclusive access to the database. However, it is a quick and easy way to completely remove any lock waits you may have.

Option 2 is often a good strategy for data warehouses. Because SQL Server Read locks (S-locks) are compatible with other S-locks, two readers can access the same table twice, without requiring the fine granularity of page and row locking. If insert operations are only done during batch time, relying solely on table locks may be a viable option. To disable row and page locking on a table and index, rebuild ALL by using a statement like this one.

```
ALTERINDEXALLON FactInternetSales REBUILD
WITH (ALLOW_PAGE_LOCKS=OFF,ALLOW_ROW_LOCKS=OFF)
```

Option 3 is a very useful technique. Processing through a view provides you with an extra layer of abstraction on top of the database –a good design strategy. In the view definition you can add a NOLOCK or TABLOCK hint to remove database locking overhead during processing. This has the advantage of making your locking elimination independent of how indexes are built and managed.

```
CREATEVIEW vFactInternetSales
AS
SELECT [ProductKey], [OrderDateKey], [DueDateKey]
,[ShipDateKey], [CustomerKey], [PromotionKey]
,[CurrencyKey], [SalesTerritoryKey], [SalesOrderNumber]
,[SalesOrderLineNumber], [RevisionNumber], [OrderQuantity]
,[UnitPrice], [ExtendedAmount], [UnitPriceDiscountPct]
,[DiscountAmount], [ProductStandardCost], [TotalProductCost]
,[SalesAmount], [TaxAmt], [Freight]
,[CarrierTrackingNumber] ,[CustomerPONumber]
FROM [dbo].[FactInternetSales] WITH (NOLOCK)
```

If you use the **NOLOCK** hint, beware of the dirty reads that can occur. For more information about locking behaviors, see SET TRANSACTION ISOLATION LEVEL (http://technet.microsoft.com/en-us/library/ms173763.aspx) in SQL Server Books Online.

3.5.1.1.6 Forcing Degree of Parallelism in SQL Server

During **ProcessData**, a partition processing task is limited by the speed achievable from a single network connection the data source. These speeds can vary between a few thousand rows per second for legacy data sources, to around 100,000 rows per second from SQL Server. Perhaps that is not fast enough for you, and you have followed the guidance in this book to allow multiple partitions to process in parallel – scaling **ProcessData** nearly linearly.

We have seen customer reach 6.5 million rows per second into a cube by processing 64 partitions concurrently. Bear in mind what it takes to transport millions of rows out of a relational database every second. Each processing query will be selecting data from a big table, aggressively fetching data as fast as the network stack and I/O subsystem can deliver them. When SQL Server receives just a single request for all rows in a large fact table, it will spawn multiple threads inside the database engine to serve that it as fast as possible – utilizing all server resources. But what the DBA of the relational engine may not know, is that in a few milliseconds, your cube design is set up to ask for *another* one of those large tables – concurrently. This presents the relational engine with a problem: how many threads each query should be assigned to optimize the total throughput of the system, without sacrificing overall performance of individual processing commands. It is easy to see that race conditions may create all sorts on interesting situations to further complicate this. If parallelism overloads the Database Engine, SQL Server must resort to context switching between the active tasks, continuously redistributing the scarce CPU resources and wasting CPU time with scheduling. You can measure this happening as a high **SOS_SCHEDULER_YIELD** wait in **sys.dm_os_wait_stats**.

Fortunately, you can defend yourself against excessive parallelism by working together with the cube designer to understand how many partitions are expected to process at the same time. You can then assign a smaller subset of the CPU cores in the relational database for each partition. Carefully designing for this parallelism can have a large impact. We have seen cube process data speeds more than double when the assigned CPU resources are carefully controlled.

You have one or two ways to partition server resources for optimal process data speeds, depending on which version of SQL Server you run.

Instance reconfiguration – Using **sp_configure**, you can limit the number of CPU resources a single query can consume. For example, consider a cube that processes eight partitions in parallel from a computer running SQL Server with 16 cores. To distribute the processing tasks equally across cores, you would configure like this.

```
EXECsp_configure'show advanced options', 1
RECONFIGURE
EXECsp_configure'max degree of parallelism', 2
RECONFIGURE
```

Unfortunately, this is a brute-force approach, which has the side effect of changing the behavior of the entire instance of SQL Server. It may not be a problem if the instance is dedicated for cube processing, but it is still a crude technique. Unfortunately, this is the only option available to you on SQL Server 2005.

Resource Governor – If you run SQL Server 2008 or SQL Server 2008 R2, you can use Resource Governor to control processing queries. This is the most elegant solution, because it can operate on each data source view individually.

The first step is to create a resource pool and a workload group to control the Analysis Services connections. For example, to the following statement limits each ProcessData task to 2 CPU cores and a maximum memory grant of 10 percent per query.

```
CREATERESOURCEPOOL [cube_process] WITH(
min_cpu_percent=0
, max_cpu_percent=100
, min_memory_percent=0
, max_memory_percent=100)
GO

CREATEWORKLOADGROUP [process_group] WITH(
    group_max_requests=0
, importance=Medium
, request_max_cpu_time_sec=0
, request_max_memory_grant_percent=10
, request_memory_grant_timeout_sec=0
, max_dop=2)
USING [cube_process]
GO
```

The next step is to design a classifier function that recognizes the incoming cube process requests. There are several ways to recognize cube connection. One is to use the host name. Another is to use application names (which you can set in the connection string in the data source view in the cube). The following example recognizes all Analysis Services connections that use the default values in the connection string.

```
USE [master]
GO
CREATEFUNCTION fnClassifier()
RETURNSsysname
WITHSCHEMABINDING
AS
BEGIN
DECLARE @group SYSNAME
IFAPP_NAME()LIKE'%Analysis Services%'BEGIN
SET @group='process_group'
END
ELSEBEGIN
SET @group ='default'
END
RETURN (@group)
END
```

Make sure you test the classifier function before applying it in production. After you are certain that the function works, enable it.

```
ALTERRESOURCEGOVERNORWITH (CLASSIFIER_FUNCTION = [dbo].[fnClassifier]);
GO
ALTERRESOURCEGOVERNORRECONFIGURE;
```

3.5.1.1.7 Loading from Oracle

Analysis Services is commonly deployed in heterogeneous environments, with Oracle being one of the main data sources. Because cubes often need to read a lot of data from the source system during processing, it is important that you use high speed drivers to extract this data. We have found that the native Oracle drivers provide a reasonable performance, especially if many partitions are processed in parallel.

However, we have also found that data can be extracted from a SQL Server data source at 5-10 times the speed of the same Oracle source. The SQL Server driver SQLNLCI is optimized for very high extraction speeds – sometimes reaching up to 80,000-100,000 rows per second from a single TCP/IP connection, depending on the source schema). We have tested processing speeds on top of SQL Server all the way to 6.1 million rows per second.

Consider that SQL Server Integration Services, part of the same SKU as Analysis Services, has a high-speed Oracle driver available for download. This driver is optimized for high-speed extraction from

Oracle. We have found that the following architecture often provides faster processing performance than processing directly on top of Oracle.

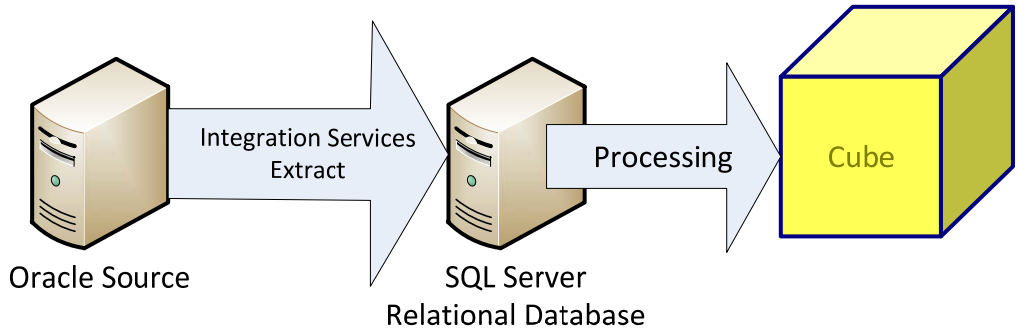


Figure 64 - Fast Processing on Oracle

References:

- Microsoft Connectors Version 1.1 for Oracle and Teradata by Attunity - http://www.microsoft.com/downloads/en/details.aspx?FamilyID=6732934c-2eea-4a7f-85a8-8ba102e6b631
- The Data Loading Performance Guide - http://msdn.microsoft.com/en-us/library/dd425070.aspx
  - Describes how to move lots of data into SQL Server

3.5.2 Tuning Process Index

As with **ProcessData** workloads, you can often increase speed of **ProcessIndex** by running more partitions in parallel. However, if this option is not available to you, the thread settings can provide an extra benefit. When you measure CPU utilization with the counter **Processor –% Processor Time – Total** and you find utilization is less than 100 percent with no I/O bottlenecks, there is a good chance that you can increase the speed of the **ProcessIndex** phase further by using the techniques in this section.

3.5.2.1 CoordinatorBuildMaxThreads

When a single partition is scanned, the amount of threads used to scan extents is limited by the **CoordinatorBuildMaxThreads** setting. The setting determines the maximum number of threads allocated per aggregation processing job. It is the absolute number of threads that can be run by an aggregation processing job. Keep in mind that you are still limited by the number of threads in the process thread pool when processing, so increasing this value may require increasing the threads available to the process thread pool too.

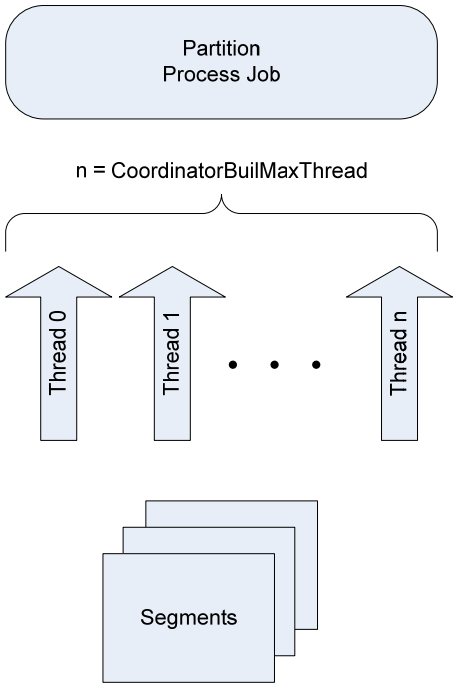


Figure 24 CoordinatorBuildMaxThreads

If you are not able to drive high parallelism by using more partitions, you can change the **CoordinatorBuildMaxThreads** value. Increasing this allows you to use more threads per partition while building aggregations for each partition during the **ProcessIndex** phase

3.5.2.2 AggregationMemoryMin and Max

As you increase parallelism of **ProcessIndex**, you may run into memory bottlenecks. On a server with more than around ten **Process ndex** jobs running in parallel, you may need to adjust **AggregationMemoryMin** and **AggregationMemoryMax** to get optimal results. For more information, see the Memory section in Part 2.

3.5.3 Tuning Queries

Query optimization is mostly covered in Part 1 and will generally involve design or application level changes. However, there are some optimizations that can be made in a production cube that are transparent to users and BI developers. These are described here.

3.5.3.1 Thread Pool Tuning

Analysis Services uses the parsing thread pools, the query thread pool, and the process thread pool for query workloads on Analysis Services. A listener thread listens for a client request on the TCP/IP port specified in the Analysis Services properties. When a query comes in, the listener thread brokers the request to one of the parsing thread pools. The parsing thread pools either execute the request immediately or send the request off to the query or process thread pool.

Worker threads from the query pool check the data and calculation caches to see whether the request can be served from cache. If the request contains calculations that need to be handled by the formula engine, worker threads in the query pool are use to perform the calculation and store the results. If the query needs to go to disk to retrieve aggregations or scan a partition, worker threads from the processing pool are used to satisfy the request and store the results in cache.

There are settings in the **msmdsrv.ini** file that allow users to tune the behavior of the thread pools involved in query workloads. Guidance on using them is provided in this section.

3.5.3.1.1 Parsing Thread Pools

The Analysis Services protocol uses Simple Object Analysis Protocol (SOAP) and XMLA for Analysis (XMLA) with TCP/IP or HTTP/HTTPS as the underlying transport mechanism. After a client connects to Analysis Services and establishes a connection, commands are then forwarded from the connection's input buffers to one of the two parsing thread pools where the XMLA parser begins parsing the XMLA while analyzing the SOAP headers to maintain the session state of the command.

There are two parsing thread pools: the short-command parsing pool and the long-command parsing pool. The short-command parsing pool is used for commands that can be executed immediately and the long-command parsing pool is used for commands that require more system resources and generally take longer to complete. Requests longer than one package are dispatched to the long-parsing thread pool, and one-package requests go to the short-parsing pool. If a request is a quick command, like a DISCOVER, the parsing thread is used to execute it. If the request is a larger operation, like an MDX query or an MDSCHEMA_MEMBERS, it is queued up to the query thread pool.

For most Analysis Services configurations you should not modify any of the settings in the short-parsing or long-parsing thread pools. However, with the information provided here, you can imagine a workload where either the short-parsing or long-parsing thread pools run dry. We have only seen one such workload, at very high concurrency, so it is unlikely that you will need to tune the parsing thread pool. If you do have a problem with one the parsing thread pools, it will show up as values consistently higher than zero in **MSOLAP/Thread - Short parsing job queue length** or **MSOLAP/Thread – Long parsing job queue length.**

3.5.3.1.2 Query Thread Pool Settings

Although modifying the **ThreadPool\Process\MaxThreads** and **ThreadPool\Query\MaxThreads** properties can increase parallelism during querying, you must also take into account the additional impact of **CoordinatorExecutionMode** as described in the configuration section.

In practical terms, the balancing of jobs and threads can be tricky. If you want to increase parallelism, it is important to first narrow down parallelism as the bottleneck. To help you determine whether this is the case, it is helpful to monitor the following performance counters:

- **Threads\Query pool job queue length**—The number of jobs in the queue of the query thread pool. A nonzero value means that the number of query jobs has exceeded the number of available query threads. In this scenario, you may consider increasing the number of query

threads. However, if CPU utilization is already very high, increasing the number of threads only adds to context switches and degrades performance.

- **Threads\Query pool busy threads**—The number of busy threads in the query thread pool. Note that this counter is broken in some versions of Analysis Services and does not display the correct value. The value can also be derived from the size of the thread pool minus the **Threads\Query pool idle threads** counter.
- **Threads\Query pool idle threads**—The number of idle threads in the query thread pool.

Any indication of queues in any of these thread pools with a CPU load less than 100 percent indicate a potential option for tuning the thread pool.

References

- Analysis Services Query Performance Top 10 Best Practices (http://www.microsoft.com/technet/prodtechnol/sql/bestpractice/ssasqptb.mspx)

3.5.3.2 Aggregations

Aggregations behave very much like indexes in a relational database. They are physical data structures that are used to answer frequently asked queries in the database. Adding aggregations to a cube enables you to speed up the overall throughput, at the cost of more disk space and increased processing times.

Note that just as with relational indexing, not every single, potentially helpful aggregate should be created. When the space of potential aggregates grows large, one aggregate being read may push another out of memory and case disk thrashing. Analysis Services may also struggle to find the best aggregate among many matching a given query.

A good rule of thumb is that no measure group should have more than 30 percent of its storage space dedicated to aggregates. To discover which aggregates are most useful, you should collect the **query subcube verbose** event using SQL Server Profiler while running a representative workload. By summing the run time of each unique subcube you can get a good overview of the most valuable aggregations to create. For more information about how to create aggregates, see Part 1.

There are some aggregates that you generally always want to create, namely the ones that are used directly by sets defined in the calculation script. You can identify those by tracing queries going to the cube after a process event. The first user to run a query or connect to the cube after a processing event may trigger **query subcube verbose** events that have SPID = 0 – this is the calculation script itself requesting data to instantiate any named sets defined there.

References:

- Analysis Services 2005 Aggregation Design Strategy - http://sqlcat.com/technicalnotes/archive/2007/09/11/analysis-services-2005-aggregation-design-strategy.aspx
    - Also applies to SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services
- Aggregation Manager on CodePlex - http://bidshelper.codeplex.com/wikipage?title=Aggregation%20Manager&referringTitle=Home
    - Makes the task of designing aggregations easier
- BIDS Helper - http://bidshelper.codeplex
    - Assists with many common tasks, including aggregation design

3.5.3.3 Optimizing Dimensions

Good dimension design is key to good performance of large cubes. Your developers should design dimensions that fit the needs of the business users. However, there are some minor changes you can often make on a live system that can yield a significant performance benefit. The changes are not 100 percent transparent to business users, but they may well be acceptable or even improve the behavior of the cube. Be sure to coordinate with your BI developers to understand whether you can make any of the changes discussed here in the cube.

Removing the (All) Level – In some dimensions, it does not make sense to ask for the (All) level in a query. The classic example is a date dimension. If the (All) level is present, and no default has been set, a user connecting to the cube may inadvertently ask for the sum of all years. This number is rarely a useful value and just wastes resources at the server.

Setting default members – Whenever a user issues an MDX statement, every hierarchy not explicitly mentioned in the query uses its default member value. By default, this member is the (All) level in the hierarchy. The (All) level may not be the typical usage scenario, which will cause the user to reissue the query with a new value. It is sometimes useful to set another default value in the dimension that more accurately reflects the most common usage scenario. This can be done with a simple modification to the calculation script of the cube. For example, the following command sets a new default member in the Date dimension.

```
ALTERCUBE [Adventure Works]UPDATE
DIMENSION [Date], DEFAULT_MEMBER='[Date].[Date].&[2000]'
```

Scope the All Level to NULL –Removing the (All) level and setting default members can be confusing to users of Excel. Another option is to force the (All) level to be NULL for the dimensions where querying that level makes no sense. You can use a SCOPE statement for that. For more information, see Part 1.

AttributeHierarchyEnabled – This property, when set to **false**, makes the property invisible as an attribute hierarchy to users browsing the cube. This reduces the metadata overhead of the cube. Not all attributes can be disabled like this, but you may be able to remove some of them after working with the users.

Note that there are many other optimizations that can be done on dimensions. Part 1 of this book contains more detailed information.

3.5.3.4 Calculation Script Changes

The calculation script of a cube contains the MDX statements that make up the nonmaterialized part of the cube. Optimizing the calculation script is a very large topic that is outside the scope of this document. What you should know it that such changes can often be made transparently to the user and that the gains, depending on the initial design, can often be substantial.

You should generally collect the performance counters under **MSOLAP:MDX** because these can be used by cube developers to determine whether there are potential gains to be had. A typical indicator of a slow calculation script is a large ratio between the **MSOLAP:MDX/Number of cell-by-cell evaluation nodes** and **MSOLAP:MDX/Number of bulk-mode evaluation nodes.**

3.5.3.5 Repartitioning

Partitioning of cubes can be used to both increase processing and query speeds. For example, if you struggle with process data speeds, splitting up the nightly batch into multiple partitions can increase concurrency of the processing operation. This technique is documented in Part 1 of this book.

Partitions can also be used to selectively move data to different I/O subsystems. An example of this is a customer that keeps the latest data in the cube on NAND devices and moves old and infrequently accessed data to SATA disks. You can move partition around using SQL Server Management Studio in the properties pane of the partition. Alternatively, you can use XML or scripting to move partition data by executing a query like this.

```
<AlterObjectExpansion="ExpandFull"
xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">
<Object>
<DatabaseID>Adventure Works DW</DatabaseID>
<CubeID>Adventure Works DW</CubeID>
<MeasureGroupID>Fact Reseller Sales</MeasureGroupID>
<PartitionID>Reseller_Sales_2001</PartitionID>
</Object>
<ObjectDefinition>
<Partition>
<ID>Reseller_Sales_2001</ID>
<Name>Reseller_Sales_2001</Name>
<StorageLocation>D:\MSAS10_50.KILIMANJARO\OLAP\2001\</StorageLocation>
</Partition>
</ObjectDefinition>
</Alter>
```

Be aware that after you have changed the storage location of the partition that marks the partition as unprocessed and empty, you must reprocess the partition to physically move the data. Note that Analysis Services creates a folder under the path you specify, and this folder is named using a GUID – not

easy to decode for a human. To keep track of where your moved partitions are, it is therefore an advantage to precreate folders with human-readable names to hold the data.

For large cubes, it is often a good idea to implement a “matrix” partitioning scheme: partition on both date and some other key. The date partitioning is used to selectively delete or merge old partitions. The other key can be used to achieve parallelism during partition processing and to restrict certain users to a subset of the partitions. For example, consider a retailer that operates in US, Europe, and Asia. You might decide to partition like this.

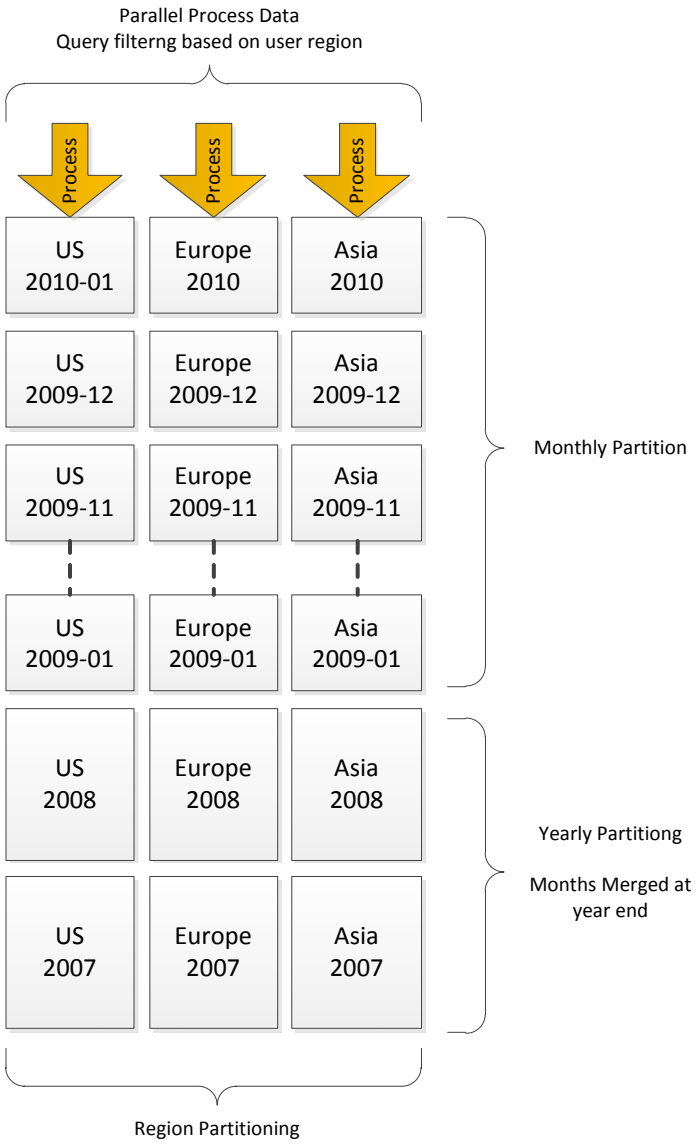


Figure 65 - Example of matrix partitioning

If the retailer grows, they may choose to split the region partitions into smaller partitions to increase parallelism of load further and to limit the worst-case scans that a user can perform. For cubes that are expected to grow dramatically, it is a good idea to choose a partition key that grows with the business

and gives you options for extending the matrix partitioning strategy appropriately. The following table contains examples of such partitioning keys.

Industry	Example partition key	Source of data proliferation
Web Retail	Customer key	Adding customers and transactions
Store Retail	Store key	Adding new stores
Data Hosting	Host ID or rack location	Adding a new server
Telecommunications	Switch ID or country code or area code	Expanding into new geographical regions or adding new services
Computerized manufacturing	Production Line ID or Machine ID	Adding production lines or (for machines) sensors
Investment Banking	Stock Exchange or financial instrument	Adding new financial instruments, products, or markets
Retail Banking	Credit Card Number or Customer Key	Increasing customer transactions
Online Gaming	Game Key or Player Key	Adding new games or players

Sometimes it is not possible to come up with a good distribution of the keys across the partitions. Perhaps you just don't have a good key candidate that fits the description in the previous paragraph, or perhaps the distribution of the key is unknown at design time. In such cases, a brute-force approach can be used: Partition on the hash value of a key that has a high enough cardinality and where there is little skew. Users will have to touch all the hash buckets as they query the cube, but at least you can perform parallel loading. If you expect every query to touch many partitions, it is important that you pay special attention to the **CoordinatorQueryBalancingFactor** described earlier.

As you add more partitions, the metadata overhead of managing the cube grows exponentially. As a rule of thumb, you should therefore seek to keep the number of partitions in the cube in the low thousands. This affects **ProcessUpdate** and **ProcessAdd** operations on dimensions, which have to traverse the metadata dependencies to update the cube when dimensions change. For large cubes, prefer larger partitions over creating too many partitions. This also means that you can safely ignore the Analysis Management Objectswarning (AMO) in Visual Studio that partition sizes should not exceed 20 million rows. We have measured the effect of large partition sizes, and found that they show negligible performance differences compared to smaller partition sizes. Therefore, the reduction in partition management overhead justifies relaxing the guidelines on partition sizes, particularly when you have large numbers of partitions.

3.5.3.5.1 Adding and Managing Partitions

Managing partitions on a large cube quickly becomes a large administrative task if done manually through SQL Server Management Studio. We recommend that you create scripts to help you manage partitioning of the cube in an automated manner instead.

The best way to manage a large cube is to use a relational database to hold the metadata about the desired partitioning scheme of the cube and use XMLA to keep the cube structure in sync with the

metadata in the relational source. Every time you request metadata from a cube, you have to run a DISCOVER command - some of these commands are expensive on a large cube with many partitions. You should design partition management code to extract the metadata from the cube in a single bulk operation instead of multiple small DISCOVER commands. Note that when you use AMO (and the SQL Server Integration Services task) to access the cube, some amount of error handling is built into the .NET library. This error handling code executes DISCOVER commands and be quite chatty with the server. A good example of this is the partition add function in AMO. This code will first check to see whether the partition already exists (it raises an error if it does) using a DISCOVER command. After that it will issue the CREATE command, creating the partition in the cube. This technique turns out to be a highly inefficient way to add many new partitions on cube with thousands of existing partitions. Instead, it is faster to first read all the partitions using a single XMLA command, discover which partitions are missing ,and then run an XMLA command to create each missing partition directly – avoiding the error checking that AMO.NET does.

In summary, design partition management code carefully and test it using SQL Server Profiler to trace the commands you generate as the code runs. Because it provides so much feedback to the server, partition management code is very expensive, and it is often surprising to customers how easy it is to create inefficient operations on a big cube. That is the price of the safety net that ADOMD.NET gives you. Using XMLA to carefully manage partitions is often a better solution for a large cube.

3.5.4 Locking and Blocking

Locks are used to manage concurrent operations. In an Analysis Services deployment, you'll find that locks are used pervasively in discovery and execution to ensure the stability of underlying data structures while a query or process is running.

Occasionally, locks taken for one operation can block other processes or queries from running. A common example is when a query takes a long time to execute, and another session issues a write operation involving the same objects. The query holds locks, causing the write operation to wait for the query to complete. Additional new queries issued on other sessions are then blocked as they wait for the original query operation to complete, potentially causing the entire service to stop handling commands until the original long-running query completes.

This section takes a closer look at how locks are used in different scenarios, and how to diagnose and address any performance problems or service disruptions that arise from locking behaviors. One of the tools at your disposal is the new SQL Server Profiler lock events that were introduced in SQL Server 2008 R2 Service Pack 1 (SP1). Finally, this section looks at deadlocks and provides some recommendations for resolving them if they begin to occur too frequently.

3.5.4.1 Lock Types

In Analysis Services, the most commonly used lock types are Read, Write, Commit_Read, and Commit_Write. These locks are used for discovery and execution on the server. SQL Server Profiler, queries, and dynamic management views (DMVs) use them to synchronize their work. Other lock types

(such as LOCK_NONE, LOCK_SESSION_LOCK, LOCK_ABORTABLE, LOCK_INPROGRESS, LOCK_INVALID) are used peripherally as helper functions for concurrency management. Because they are peripheral to this discussion, this section focuses on the main lock types instead.

The following table lists these locks and briefly describes how they are used.

Lock type	Object	Usage
Read	All	Read locks are used for reading metadata objects, ensuring that these objects cannot be modified while they are being used. Read locks are shared, meaning that multiple transactions can take Read locks on the same object.
Write	All	Write locks are used for create, alter, and update operations. Write locks are exclusive. Exclusive locks prevent concurrent transactions from taking Read or Write locks on the object at the same time.
Commit_Read	Database, Server, Server Proxy	Commit_Read locks are shared, but they block write operations that are waiting to commit changes to disk. Database Objects: Commit_Read locks are used for the following: - Query operations to ensure that no commit transactions overwrite any of the metadata objects used in the query. The lock is held for the duration of a query. - During processing, while acquiring Read and Write locks on other objects. - At the beginning of a session to calculate user permissions for a given database. - During some discover operations, such as those that read from a database object. Server Objects: Commit_Read locks are used at the beginning of a session to compute session security. Commit_Read is also used at the start of a transaction to read the master version map. Server Proxy Objects: Commit_Read locks are used at the beginning of SQL, DMX, and MDX queries to prevent changes to assemblies or administrative role while the objects are retrieved.
Commit_Write	Database, Server, Server Proxy	Commit_Write locks are exclusive locks on an object, taken to prevent access to that object while it is being updated. Commit_Write is the primary mechanism for ensuring “one version of the truth” for queries.

	Database Objects: A Commit_Write is used in a commit transaction that creates, updates, or deletes DDL structures in the database. In a commit transaction, Commit_Write locks can be taken on multiple databases at the same time, assuming the transaction includes them (for example, deleting multiple databases in one transaction). Server Object: A Commit_Write is used to update the master version map. More information about the master version map is in the next section on server locks. Server Proxy Object: A Commit_Write is taken whenever there are changes to assemblies or membership of the server role.

3.5.4.2 Server Locks

In Analysis Services, there are two server lock objects, the Server and the Server Proxy, each used for very specific purposes.

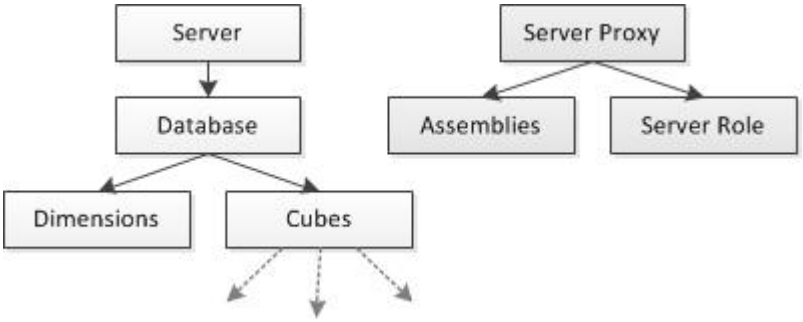


Figure 66 - Server Lock Hierarchies

The Server object takes a commit lock for operations such as processing or querying that traverse the object hierarchy.Typically, commit locks on the server are very brief, taken primarily whenever the server reads the master version map (Master.vmp) file or updates it so that it contains newer versions of objects changed by a transaction.

The master version map is a list of object identifiers and current version number for all of the major objects recognized by the server (that is,Database, Cube, Measure Groups, Partitions, Dimensions and Assemblies). Minor objects, such as hierarchy levels or attributes, do not appear in the list. The master version map identifies which object version to use at the start of a query or process. The version number of a major object changes each time you process or update it.Only the Server object reads and writes to the Master.vmp file.

When reading the master version map, the server takes a Commit_Read to protect against changes to the file while it is being read. A Commit_Write is taken on the Server object to update the master version map, by merging newer version information that was created in a transaction.

The Server Proxy object is used for administrative locks, for example when you make changes to the membership of the Server role or to assemblies used by the database.

3.5.4.3 Lock Fundamentals

This section explores some of the foundational concepts that describe locking behavior in Analysis Services. Like all locking mechanisms, the purpose of locks in Analysis Services is to protect metadata or data from the effects of concurrency.

The rules or principles of how locks are used can be distilled into the following points:

- Locks protect metadata; latches protect data. Latches are lightweight locks that are rapidly acquired and released. They are used for atomic operations, like reading a data value out of system data.
- Read locks are taken for objects that feed into a transaction; Write locks are taken for objects that are changed by the transaction. Throughout the rest of this section, we'll see how this principle is applied in different operations.

Consider the case of dimension processing. Objects like data sources and data source views that provide data to the dimension (that is, objects that the dimension depends on) take a Read lock. Objects that are changed by the operation, such as partitions, take a Write lock. When a dimension is processed, the partitions need to be unprocessed and then reprocessed, so a Write lock is taken on the partitions to update the dimension-related data within each partition.

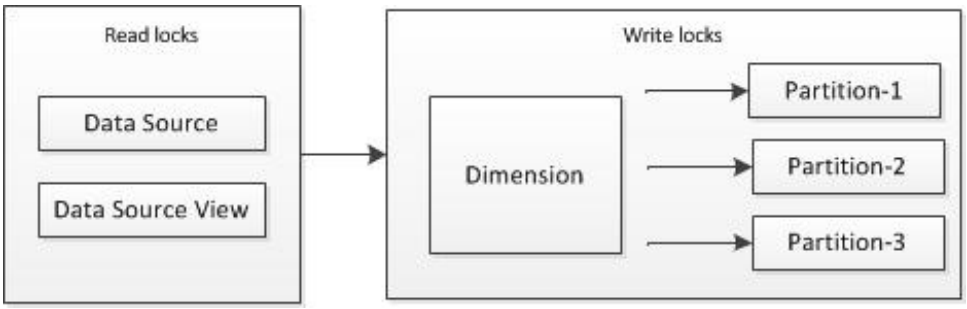


Figure 67 - Locks during dimension processing

Compare the preceding illustration to the following one, which shows only Read locks used for partition processing. Because no major objects depend on a partition, the only Write lock in play is on the partition itself.

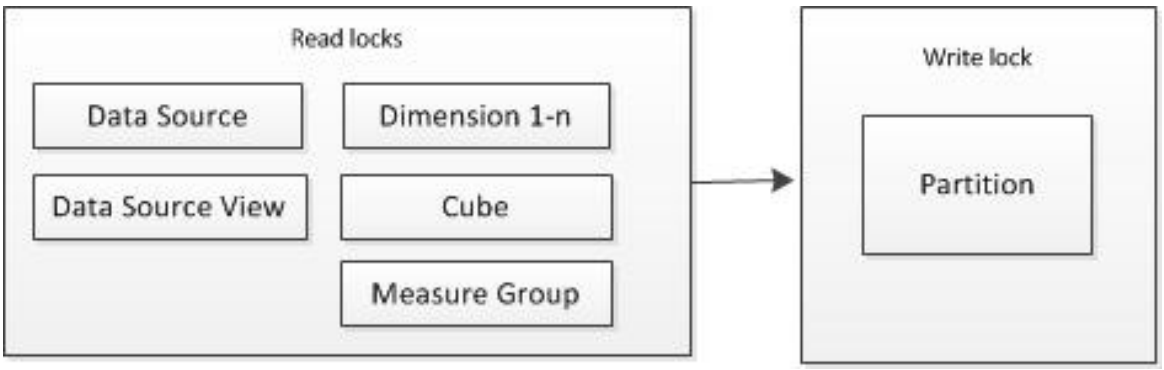


Figure 68 - Locks during partition processing

3.5.4.4 Locks by Operation

This section explains the locks taken by different operations on the server. This list of locks is not exhaustive:

- Begin Session
- Queries
- Discover
- Processing
- DDL operations
- Rollback

Other operations such as lazy processing, sessions, and proactive caching pose interesting challenges in terms of understanding locks. The reason is discussed at the end of this section, but these other operations are not discussed in detail. Also missing from the list is writeback, Because writeback is a hybrid of query and process operations, which are covered individually.

3.5.4.4.1 Begin Session

When a session starts, Analysis Services determines which databases the user has permission to use. Performing this task requires taking a Commit_Read lock on the database.If a database is not specified, session security is computed for each database on the server until a database is found that the session has read access to. Locks are released after session security is computed. Occasionally, the commit lock is acquired and released so quickly that it never shows up in a trace.

3.5.4.4.2 Queries

All queries run on a database and in a session.A Commit_Read lock is taken simultaneously on the Server Proxy object and database when the query starts. The lock is held for the duration of the query.

The Commit_Read lock on a database held by query is sometimes blocks processing operations.

Figure 69 - Write locks blocking queries

In the preceding example, the long-running query prevents the Commit_Write lock required by the processing operation (yellow arrow) from being taken. New readers requiring the Commit_Read lock on the database then become blocked behind the processing command. It is this combination of events that can turn a three-second query into a three-minute (or longer) query.

3.5.4.4.3 Discovers

There are two types of discover operations that use locks: those that request metadata about instances or objects (such as MDSCHEMA_CUBE), and those that are part of a dynamic management view (DMV) query.

The Discover method on metadata objects behaves very similarly to a query in that it takes a Commit_Read lock on the database.

Discovers issued for DMV operations don't take Read locks. Instead, DMVs use interlocking methods to synchronize access to system data that is created and maintained by the server. This approach is sufficient because DMV discovers do not read from the database structure. As such, the protection offered by locks is overkill. Consider DISCOVER_OBJECT_MEMORY_USAGE. It is a DMV query that returns shrinkable and unshrinkable memory used by all objects on the server at given point in time. Because it is a fast read of system data directly on the server, DISCOVER_OBJECT_MEMORY_USAGE uses interlocked access over memory objects to retrieve this information.

3.5.4.4.4 Processing

For processing, locks are acquired in two phases, first during object acquisition and again during schedule processing. After these first two phases, the data can be processed and the changes committed.

Phase 1: Object Acquisition

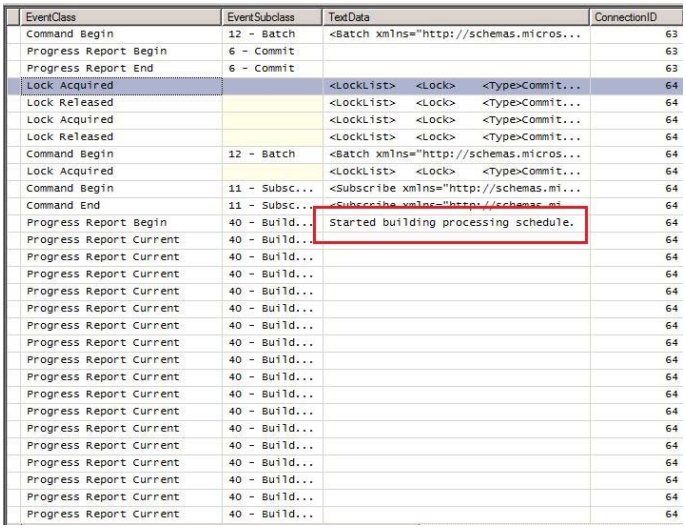
1. A Commit_Read lock is acquired on the database while the objects are retrieved.
2. The objects used in processing are looked up.
3. The objects are identified, and then the Commit_Read lock on the database is released.

Phase 2: Schedule Processing

This phase finds the objects on which the dimension or partition depends, as well as those objects that depend on it.In practice, building the schedule is an iterative process to ensure that all of the dependencies are identified. After all of the dependencies are understood, the execution workflow

moves forward to the processing phase. The following list summarizes the events in schedule processing.

1. Scheduler builds a dependency graph that identifies all of the objects involved in the processing command. Based on this graph, it builds a schedule of operations.



EventClass	EventSubclass	TextData	ConnectionID
Command Begin	12 - Batch	<Batch xmlns="http://schemas.micros...	63
Progress Report Begin	6 - Commit		63
Progress Report End	6 - Commit		63
Lock Acquired		<LockList> <Lock> <Type>Commit...	64
Lock Released		<LockList> <Lock> <Type>Commit...	64
Lock Acquired		<LockList> <Lock> <Type>Commit...	64
Lock Released		<LockList> <Lock> <Type>Commit...	64
Command Begin	12 - Batch	<Batch xmlns="http://schemas.micros...	64
Lock Acquired		<LockList> <Lock> <Type>Commit...	64
Command Begin	11 - Subsc...	<Subscribe xmlns="http://schemas.mi...	64
Command End	11 - Subsc...	<Subscribe xmlns="http://schemas.mi...	64
Progress Report Begin	40 - Build...	Started building processing schedule.	64
Progress Report Current	40 - Build...		64
Progress Report Current	40 - Build...		64
Progress Report Current	40 - Build...		64
Progress Report Current	40 - Build...		64
Progress Report Current	40 - Build...		64
Progress Report Current	40 - Build...		64
Progress Report Current	40 - Build...		64
Progress Report Current	40 - Build...		64
Progress Report Current	40 - Build...		64
Progress Report Current	40 - Build...		64
Progress Report Current	40 - Build...		64
Progress Report Current	40 - Build...		64
Progress Report Current	40 - Build...		64
Progress Report Current	40 - Build...		64
Progress Report Current	40 - Build...		64
Progress Report Current	40 - Build...		64
Progress Report Current	40 - Build...		64

2. Scheduleracquires a Commit_Read on the database, and then it acquires Read and Write locks on the objects.
3. Scheduler generates the jobs and is now free to start executing the jobs.

Phase 3:Process

After scheduling, there are no Commit_Read locks, just Write locks on the affected objects, and read locks on objects. Only Read and Write locks are used to do the work. Read and Write locks prevent other transactions from updating any objects used in the transaction. At the same time, objects that are related but not included in the current transaction can be updated. For example, suppose two different dimensions are being processed in parallel. If each dimension is referenced by different cubes and if each dimension is fully independent of the other, there is no conflict when the transaction is committed. Had a Commit_Write lock been held on the database, this type of parallel processing would not be possible.

Phase 4: Commit

When a Commit transaction starts, a Commit_Write lock is taken on the database and held for the duration of the transaction. No new Commit_Read locks are accepted while Commit_Write is pending. Any new Begin Sessions may encounter a connection timeout. New queries are queued or canceled, depending on timeout settings.

The Commit transaction waits for the query queue to drain (that is, it waits for Commit_Read locks to be released). At this point, reading from the property settings defined on the server, it might use one of the following properties to cancel a query if it is taking too long:

ForceCommitTimeout cancels transactions that hold Commit_Read locks. By default, this property is set to 30 seconds. However, it is important to remember that cancelations are not always instantaneous. Sometimes it takes several minutes to release commit locks.

CommitTimeout cancels transactions requesting Commit_Write locks, in effect prioritizing queries over processing. The server uses whichever timeout occurs first. If **ForceCommitTimeout** occurs sooner than **CommitTimeout**, cancelation is called on long-running queries instead of the write operation.

Note: If the client application has its own retry logic, it can reissue a command in response to a connection timeout and the connection will succeed if there are available threads.

A commit transaction also takes a Commit_Write lock on the server. This is very short and occurs when the transaction version map is merged into the master version map that keeps track of which object versions are the master versions. Because this is a Write lock, it must wait for Read locks to clear – this wait time is controlled by **ForceCommitTimeout** and **CommitTimeout**. A commit transaction creates new versions of the master objects, ensuring consistent results for all queries.

After the Master.vmp file is updated, the server deletes unused version data files. At this point, the Commit_Write lock is released.

3.5.4.4.5 DDL Operations

For operations that create, alter, or delete metadata objects, locks are acquired once. It is similar to the processing workflow but without the scheduling phase. It finds the objects that the object depends on and locks them using Read locks, and Write locks are taken on the objects that depend on the object that is being modified. Write locks are also acquired on the objects that are created, updated, or deleted in the transaction.

3.5.4.4.6 Rollback

For rollback, there is a server latch that protects Master.vmp, but a rollback by itself does not take a Commit_Read or Commit_Write lock on the database. In a rollback, nothing is changing, so no commit locks are required. However, any Read and Write locks that were taken by the transaction are still held during the rollback, but they are released after the rollback has been completed.Rollback deletes the unused versions of any objects created by the original transaction.

3.5.4.4.7 Session Transactions, Proactive Caching, and Lazy Processing

In terms of locking, sessions, proactive caching and lazy processing are tricky because they have breakable locks. Pending a write operation from an administrator, Lock Manager cancels Write locks in a session, proactive caching, or lazy processing, and it lets the Write Commit prevail.

Sessions can have Write locks that don’t conflict. Snapshots and checkpoints are used to manage write operations for session cubes.The classic example is the grouping behavior in Excel where new dimensions are created in-session, on the fly, to support ad-hoc data structures. The new dimensions are always rolled back eventually, but they can be problematic during their lifetime. If an error occurs in

session, the server might perform a partial rollback to achieve a stable state; sometimes these rollback operations have unintended consequences.

3.5.4.4.8 Synchronization, Backup and Restore, and Attach

The locking mechanism for synchronization, restore, and attach are already documented in the “Analysis Services Synchronization Best Practices” article on the SQLCAT web site (http://sqlcat.com). Restated from that article, the basic workflow of lock acquisition and release for synchronization is as follows.

During synchronization, a Write lock is applied before the merge of the files on the target server, and a read commit lock is applied to the source databasewhile files are transferred.

- The Write lock is applied on the target server (when a target database exists), preventing users from querying and/or writing in database. The lock is released from the target database after the metadata is validated, the merge has been performed and the transaction is committed.
- The read commit lock is taken on the source server to prevent processing from committing new data, but it allows queries to run, including other source synchronization. Because of this, multiple servers can be synchronized at the same time from the same source. The lock is released at about the same moment as the Write lock, as it is taken in this distributed transaction.

For synchronization only, there is also an additional Commit_Write lock while the databases are merged. This lock can be held for a long period of time while the database is created and overwritten.

For backup, there is only a Commit_Read on the database.

A restore operation uses a more complex series of locks. It acquires a Write lock on the database that is being restored, ensuring the integrity of the database as its being restored, but blocking queries while Restore is being processed. Best practice recommendations suggest using two databases with different names so that you can minimize query downtime.

The basic workflow for all of these operations is as follows:

1. Write locks are taken on the database that is being synchronized, restored, or attached.
2. Files are extracted and changes committed while processing.
3. The locks are released.

3.5.4.5 Investigating Locking and Blocking on the Server

This section describes the tools and techniques for monitoring locks and removing locking problems. One of the tools discussed is the new lock events that are introduced in SQL Server 2008 R2 SP1. The other tool is DMVs (and the DISCOVER_LOCKS schema rowset in particular), which are discussed with emphasis on how it fits into a troubleshooting scenario.

Locking and blocking problems manifest themselves in a server environment in different ways. As with all performance problems, this one is a matter of degree. You might have a blocking issue in your environment that resolves so quickly, it never registers as a problem that requires a solution. At the

other end of the spectrum, blocking can become so severe that users are locked out of the system. Connection requests fail; queries either time out or fail to start altogether.

In these extreme scenarios, it is difficult to clearly diagnose the problem if you do not have an available thread to connect SQL Server Profiler or if you are unable to run a DMV query that tells you what is going on. In this case, the only way to confirm a locking problem is to work with a Microsoft support engineer to analyze a memory dump. The engineer can tell you whether your server unavailability is due to locks, indicated by **PCLockManager::Wait** on multiple threads.

3.5.4.5.1 Using DMV Queries and XMLA to Cancel a Blocked Transaction

If your situation is less dire, you can use a DMV query and XML for Analysis (XMLA) to unblock your server. Given a free thread for processinga new connection request, you can connect to the server using an administrator account, run a DMV query to get the list of locks, and then try to kill the blocking SPID by running this XMLA command.

```
<Cancel xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">
      <SPID>nnnn</SPID>
</Cancel>
```

To get the SPID, you can use SQL Server Management Studio to connect to the server and then issue a DISCOVER_LOCKS statement as an MDX query.

```
Select * from $SYSTEM.DISCOVER_LOCKS
```

This DMV query returns a snapshot of the locks used at specific point in time. DISCOVER_LOCKS returns a rowset directly from Lock Manager. As such, the data you get back might not be as clear or easy to follow as the trace events in SQL Server Profiler. The following example contains a snapshot of the locks held during a query but no information about the query itself.

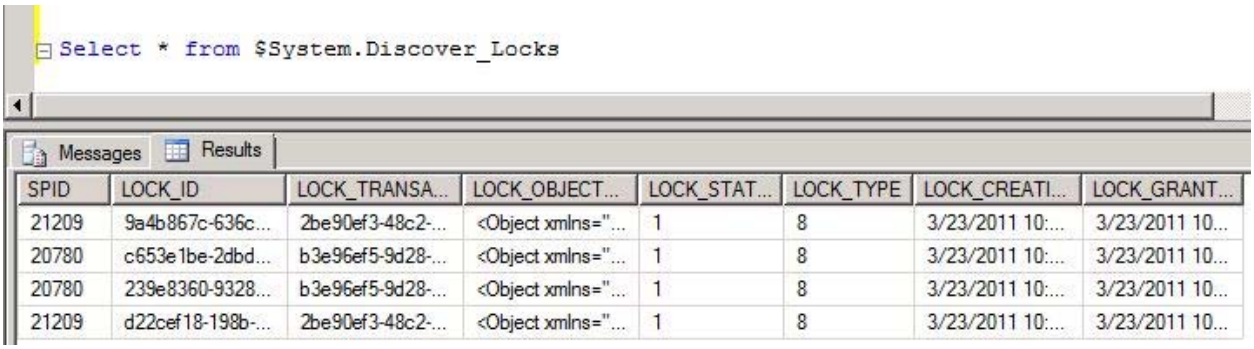


Figure 70 - Output of $system.discover_locks

Unfortunately, there is no mitigation for this. You cannot run DISCOVER_LOCKS in conjunction with other DMV statements to get additional insight into the timing of the lock acquisition-release lifecycle

relative to the transactions running on your server. You can only run the MDX SELECT statement and then act on the information it provides.

3.5.4.5.2 Using SQL Server Profiler to Analyze Lock Contention

SQL Server Profiler offers numerous advantages over DMV queries in terms of depth and breadth of information, but it comes at a cost. It is often not feasible to run SQL Server Profiler in a production environment because of the additional resource demands it places on server. But if you can use it, and if you are running SQL Server 2008 R2 SP1, you can add the new Lock Acquired, Lock Released, and Lock Waiting events to a trace to understand the locking activity on your server.

SQL Server 2008 R2 SP1adds the following events to the Locks event category in SQL Server Profiler: Lock Acquired, Lock Released, Lock Waiting.

Like other lock events, they are not enabled by default. You must select the **Show all columns** check box before you can select events in the Locks category. To get an idea of how locks are acquired and released in the course of a transaction be sure to add Command Begin and Command End to the trace. If you want to view the MDX executed, you should also include the Query Begin and Query End.

In Analysis Services, it is normal to see a large number of acquired and released lock events. Every transaction that includes Read or Write operations on a major object requests a lock to perform that action. Lock Waiting is less common, but by itself is not symptomatic of a problem. It merely indicates that a queue has formed for transactions that are requesting the same object. If the queue is not long and operations complete quickly, the queue drains and query or processing tasks proceed with only a small delay.

The following illustration shows a Lock Waiting event and the XML structures that identify which transaction currently holds a lock on the object, and which transactions are waiting for that same object. In the illustration:

- **<HoldList>** shows which transaction is currently holding a Commit_Write lock on the database.
- **<WaitList>** indicates that another transaction is waiting for a Commit_Write lock on the same database.

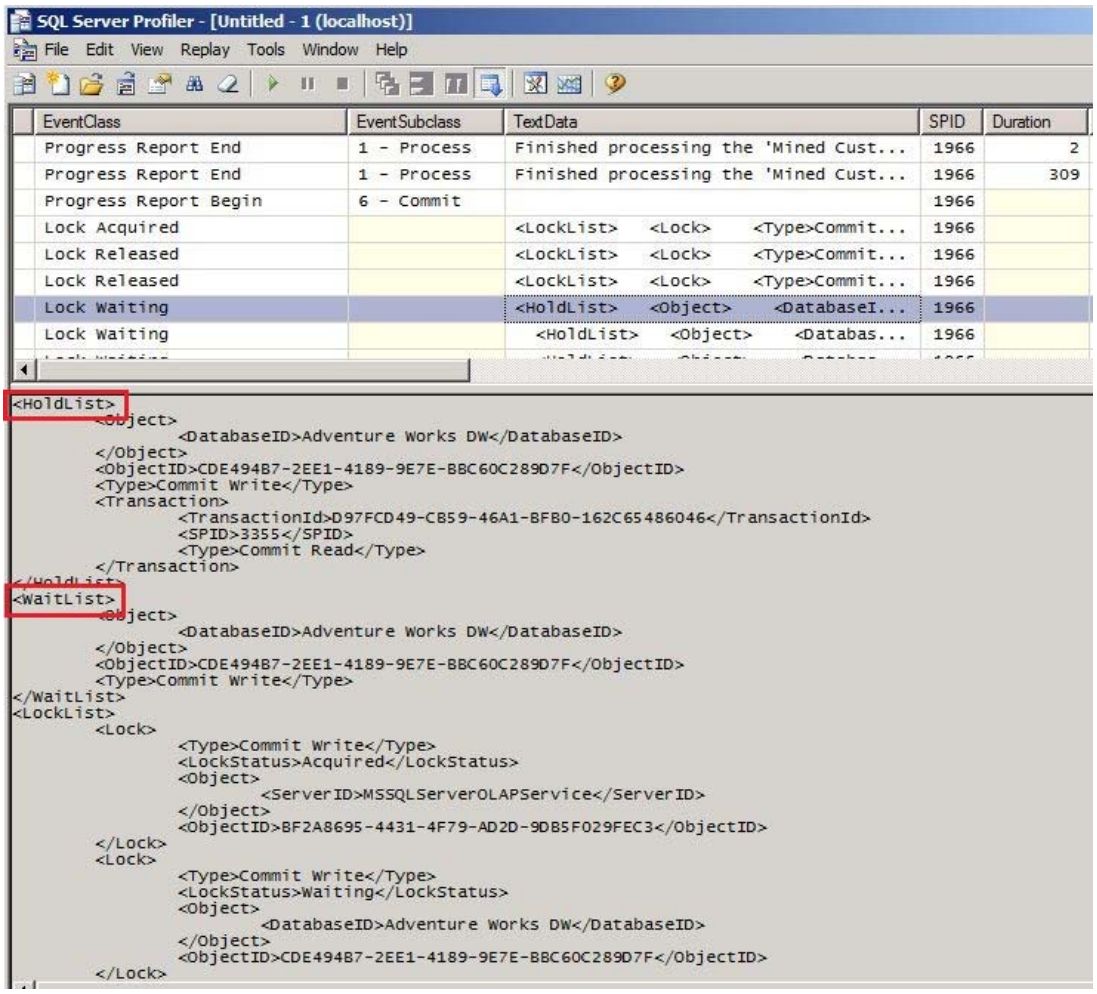


Figure 31 – SQL Server Profiler output of lock events

3.5.4.6 Deadlocks

A deadlock in Analysis Services is lock contention between two or more sessions, where operations in either session can't move forward because each is waiting to acquire a lock held by the other session.

Analysis Services has deadlock detection but it only works for locks (Read, Write, Commit_Read, Commit_Write). It won't detect deadlocks for latches. When a deadlock is detected, Analysis Services stops the transaction in one of the sessions so that the other transaction can complete.

Deadlocks are typically a boundary case. If you can reproduce it, you can run a SQL Server Profiler trace and use the Deadlock event to determine the point of contention. In SQL Server Profiler, a deadlock looks like the following:

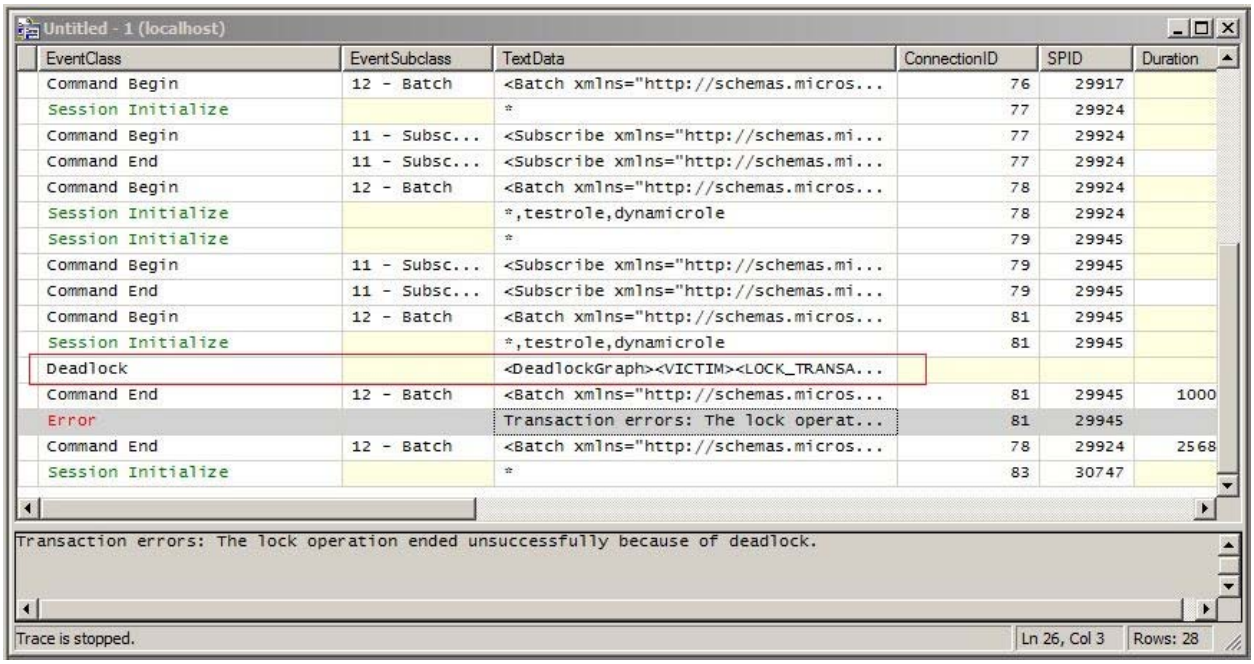


Figure 71 - Deadlock events in profiler

A deadlock event uses an XML structure called a deadlockgraph to report on which sessions, transactions, and objects created the event. The <VICTIM> node in the first few lines identifies which transaction was sacrificed for the purpose of ending the deadlock. The remainder of the graph enumerates the lock type and status for each object. In the following example, you can see which objects are requested and whether the lock was granted or waiting.

```
<DeadlockGraph>
<VICTIM>
<LOCK_TRANSACTION_ID>E0BF8927-F827-4814-83C1-98CA4C7F5413</LOCK_TRANSACTION_ID>
<SPID>29945</SPID>
</VICTIM>
<LOCKS>

<Lock>s
<LOCK_OBJECT_ID><Object><DatabaseID>FoodMart
2008</DatabaseID><DimensionID>Promotion</DimensionID></Object></LOCK_OBJECT_ID>
<LOCK_ID>326321DF-4C08-43AA-9AEC-6C73440814F4</LOCK_ID>
<LOCK_TRANSACTION_ID>E0BF8927-F827-4814-83C1-98CA4C7F5413</LOCK_TRANSACTION_ID>
<SPID>29945</SPID>
<LOCK_TYPE>2</LOCK_TYPE>      ---- Lock_Type 2 is a Read lock
<LOCK_STATUS>1</LOCK_STATUS>  ---- Lock_Status 1 is 'acquired'
</Lock>

<Lock>
<LOCK_OBJECT_ID><Object><DatabaseID>FoodMart
2008</DatabaseID><DimensionID>Product</DimensionID></Object></LOCK_OBJECT_ID>
<LOCK_ID>21C7722B-C759-461A-8195-1C4F5A88C227</LOCK_ID>
<LOCK_TRANSACTION_ID>E0BF8927-F827-4814-83C1-98CA4C7F5413</LOCK_TRANSACTION_ID>
<SPID>29945</SPID>
<LOCK_TYPE>2</LOCK_TYPE>      ---- Lock_Read
<LOCK_STATUS>0</LOCK_STATUS>  ---- waiting on Product, which is locked by 29924
</Lock>
```

```
<Lock>
<LOCK_OBJECT_ID><Object><DatabaseID>FoodMart
2008</DatabaseID><DimensionID>Promotion</DimensionID></Object></LOCK_OBJECT_ID>
<LOCK_ID>7F15875F-4CCB-4717-AE11-5F8DD48229D0</LOCK_ID>
<LOCK_TRANSACTION_ID>1D3C42F3-E875-409E-96A0-B4911355675D</LOCK_TRANSACTION_ID>
<SPID>29924</SPID>
<LOCK_TYPE>4</LOCK_TYPE>      ---- Lock_Write
<LOCK_STATUS>0</LOCK_STATUS>  ---- waiting on Promotion, which is locked by 29945
</Lock>

<Lock><LOCK_OBJECT_ID><Object><DatabaseID>FoodMart
2008</DatabaseID><DimensionID>Product</DimensionID></Object></LOCK_OBJECT_ID>
<LOCK_ID>96B09D08-F0AE-4FD2-8703-D33CAD6B90F1</LOCK_ID>
<LOCK_TRANSACTION_ID>1D3C42F3-E875-409E-96A0-B4911355675D</LOCK_TRANSACTION_ID>
<SPID>29924</SPID>
<LOCK_TYPE>4</LOCK_TYPE>      ---- Lock_Write
<LOCK_STATUS>1</LOCK_STATUS>  ---- granted
</Lock><
/LOCKS>
</DeadlockGraph>
```

Deadlocks should be rare events, if they occur at all. Persistent deadlocks indicate a need for redesigning your processing strategy. You might need to speed up processing by making greater use of partitions, or you might need to take a closer look at other processing options or schedules to see whether you can eliminate the conflict. For more information about these recommendations, see Part 1 of this book or the Analysis Services Performance Guide (http://www.microsoft.com/downloads/en/details.aspx?FamilyID=3be0488d-e7aa-4078-a050-ae39912d2e43&displaylang=en).

References:

- Analysis Services Synchronization Best Practices - http://sqlcat.com/technicalnotes/archive/2008/03/16/analysis-services-synchronization-best-practices.aspx
- Deadlock Troubleshooting in SQL Server Analysis Services (SSAS) - http://blogs.msdn.com/b/sql_pfe_blog/archive/2009/08/27/deadlock-troubleshooting-in-sql-server-analysis-services-ssas.aspx
- SSAS: Processing, ForceCommitTimeout and "the operation has been cancelled" - http://geekswithblogs.net/darrengosbell/archive/2007/04/24/SSAS-Processing-ForceCommitTimeout-and-quotthe-operation-has-been-cancelledquot.aspx
- Locking and Unlocking Databases (XMLA) - http://msdn.microsoft.com/en-us/library/ms186690.aspx

3.5.5 Scale Out

Analysis Services currently supports up to 64 cores in a scale-up configuration. If you want to go beyond that scale, you will have to design for scale-out. There are also other drivers for scale-out; for example, it may simply be cheaper to use multiple, smaller machines to achieve high user concurrency. Another consideration is processing and query workloads. If you expect to spend a lot of time processing or if you are designing for real time, it is often useful to scale-out the processing on a different server than the query servers.

Scale-out architectures can also be used to achieve high availability. If you have multiple query servers in a scale-out farm, some of them can fail but the system will remain online.

Although the full details of designing a scale-out Analysis Services farm is outside the scope of this book, it is useful to understand the tradeoffs and potential architectures that can be applied.

3.5.5.1 Provisioning in a Scaled Out Environment

In a scale-out environment you can use either read-only LUN and the attach and detach functionality of SQL Server 2008 Analysis Services or SAN snapshots (which can also be used in SQL Server 2005 Analysis Services) to host multiple copies of the same database on many machines.

When you update a scale-out read-only farm, a Windows volume has to be dismounted and mounted every time you update an Analysis Services database. This means that no matter which disk technology you use the scale out, the smallest unit you can update without disturbing other cubes is a Windows volume. If you have multiple databases in the scale-out environment, it is therefore an advantage to have each database live on its own Windows volume. This separation allows you to update the databases independently of each other if you are running SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services.

3.5.5.2 Scale-out Processing Architectures

There are basically three different architectures you can use in a scale-out configuration:

- Dedicated processing architecture
- Query/processing flipping architecture
- ROLAP

The three architectures have different tradeoffs that you have to consider, which this section describes. Note: It is also possible to combine these architectures in different hybrids, but that is outside the scope of this document. Understanding the tradeoffs for each will help you make the right design decisions.

3.5.5.2.1 Dedicated Processing Architecture

In the dedicated processing architecture, an instance of Analysis Services is reserved to process all new, incoming data. After processing is done, the result is copied to query servers. The advantage of this architecture is that the query servers can respond to the queries without being affected by the processing operation. A lock is required only when data is updated or added to the cube.

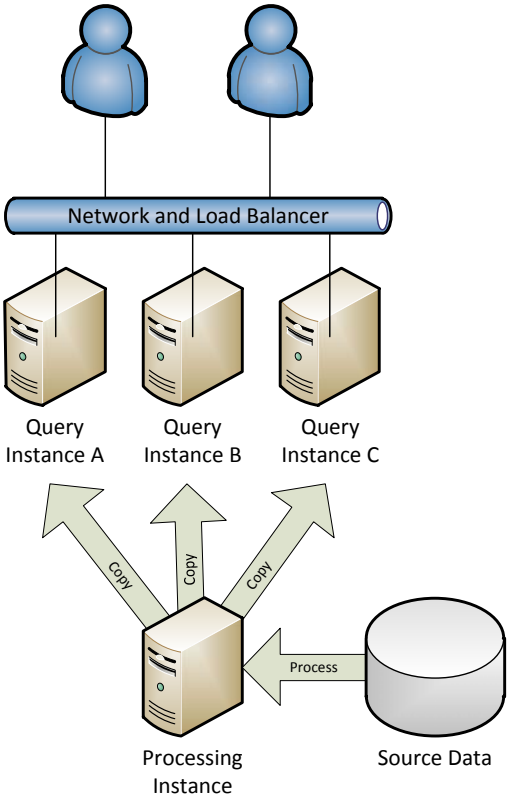


Figure 72 - Dedicated Processing Architecture

In a dedicated processing architecture, consider how to get the data from the processing instance to the query servers. There are several ways to achieve this.

Analysis Services Cube Synchronization: By using this built-in Analysis Services functionality, you can move the delta data directly to the query servers.

Robocopy or NiceCopy: By using a high-speed copying program, you can quickly synchronize each query instance with its own copy of the changed data. This method is generally faster than cube synchronization, but it requires you to set up your own copy scripts.

SAN Snapshots or Storage Mirrors: Using SAN technology, it is possible to automatically maintain copies of the data LUN on the processing servers. These copies can then be mounted on the query servers when the data is updated.

SAN Read -Only LUN: Using this technique, which is available only in SQL Server 2008 and SQL Server 2008 R2, you can use read-only LUN to move the data from the processing instance to the query servers. A read-only LUN can be shared between multiple servers, and hence enables you to use more than query server on the same physical disk.

Both SAN snapshots and read-only LUN strategies may require careful design of the storage system bandwidth. If your cube is small enough to fit in memory, you will not see much I/O activity and this

technique will work very well out of the box. However, if the cube is large and cannot fit in memory, Analysis Services will have to do I/O operations. As you add more and more query servers to the same SAN, you may end up creating a bottleneck in the storage processors on the SAN to serve all this I/O. You should make sure that the SAN is capable of supporting the required throughput. If the I/O throughput is not sufficient, you may end up with a scale-out solution that performs worse than a scale-up.

If you are worried about I/O bandwidth, the Robocopy/NiceCopy solution or the cube synchronization solution may work better for you. In these solutions you can have dedicated storage on each query server . However, you have to make sure there is enough bandwidth on the network to run multiple copies over the network. You may have to use dedicated network cards for such a setup.

The dedicated processing architectures can also be used to achieve high availability. However, you need a way to protect the processing server, to avoid a single point of failure. You can haveeither a standby processing server or an extra (disabled) instance on one of the query servers that can be used to take over the role of processing service in the case of hardware failure. Another alternative is to use clustering on the processing server ,although this may waste hardware resources on the passive node.

References:

- Scale-Out Querying for Analysis Services with Read-Only Databases - http://sqlcat.com/whitepapers/archive/2010/06/08/scale-out-querying-for-analysis-services-with-read-only-databases.aspx
- Sample Robocopy Script to Synchronize Analysis Services Databases - http://sqlcat.com/technicalnotes/archive/2008/01/17/sample-robocopy-script-to-customer-synchronize-analysis-services-databases.aspx
- Scale-Out Querying with Analysis Services - http://sqlcat.com/whitepapers/archive/2007/12/16/scale-out-querying-with-analysis-services.aspx
- Scale-Out Querying with Analysis Services Using SAN Snapshots - http://sqlcat.com/whitepapers/archive/2007/11/19/scale-out-querying-with-analysis-services-using-san-snapshots.aspx
- Analysis Services Synchronization Best Practices - http://sqlcat.com/technicalnotes/archive/2008/03/16/analysis-services-synchronization-best-practices.aspx
- SQL Velocity – Scalable Shared Data base - http://sqlvelocity.typepad.com/blog/2010/09/scalable-shared-data-base-part-1.html

3.5.5.2.2 Query/Processing Flipping Architecture

The dedicated processing server architecture solves most scale-out cases. However, the time required to move the data from the processing server to the query servers may be restrictive if updates happen at intervals shorter than a few hours. Even with SAN snapshots or read-only LUN, it will still take some time to dismount the LUN, set it online on the query server, and finally mount the updated cube on the query

servers. In the query/processing flipping architecture, each instance of Analysis Services performs its own processing, as illustrated in the following figure.

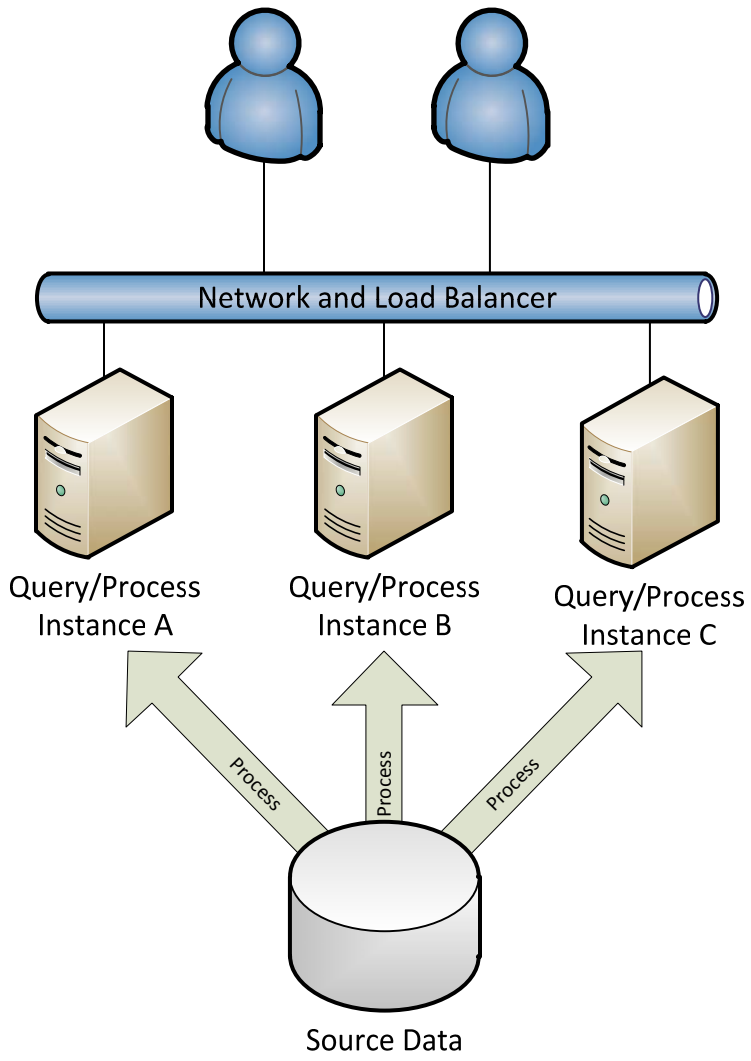


Figure 73 - Query and processing flipping

Because each server does it own processing, it is possible that some servers will have more recent data than others. This means that one user executing a query may get a later version of the data than another use executing the same query concurrently. However, for many scenarios where you are going near-real time, such a state of loose synchronization may is an acceptable tradeoff. If the tradeoff is not acceptable, you can work around it with careful load balancing – at the price of adding some extra latency to the updates.

In the preceding diagram, you can see that the source system receives more processing requests than in the dedicated processing architecture. You should scale the source accordingly and consider the network bandwidth required to read the source data more than once.

The query/processing flipping architecture also has a build in high availability solution. If one of the servers fails, the system can remain online but with additional load on the rest of the servers.

3.5.5.3 Query Load Balancing

In any scale-out architecture with more than one query server, you need to have a load balancing mechanism in place. The load balancing mechanism serves two purposes. First, it enables you to distribute queries equally across all query servers, achieving the scale-out effect. Second, it enables you to selectively take query servers offline, gracefully draining them, while they are being refreshed with new data.

When you use any load-balancing solution, be aware that the data caches on each of the servers in the load-balancing architecture will be in different states depending on the clients it is currently serving. This results in differences in response times for the same query, depending on where it executes.

There are several load balancing strategies to consider. These are treated in the following subsections. As you choose the load balancer, bear in mind the granularity of the load balancing and how this affects the process to query server switching. This is especially important if you use a dedicated processing architecture. For example, the Windows Network Load Balancing solution balances users between each Analysis Services Instance in the scale-out farm. This means that when you have to drain users from a query server and update the server with the latest version of the cube, the entire instance has to be drained. If you host more than one database per instance, this means that if one database is updated the other databases in the same instance must also be taken offline. Client load balancing and Analysis Services Load Balancer may be better solutions for you if you want to load-balance databases individually.

3.5.5.3.1 Client Load Balancing

In the client load balancing, each client knows which query server it will use. Implementing this strategy requires client-side code that can intelligently choose the right query server and then modify the connection string accordingly. Excel add-ins are an example of this type of client-side code. Note that you will have to develop your own load balancer to achieve this.

3.5.5.3.2 Hardware-Level Load Balancing

Using technologies such as load balancers from F5, it is possible to implement load balancing directly in the network layer of the architecture. This makes the load balancing transparent to both Analysis Services and the client application. If you choose to go down this route, make sure that the load-balance appliance enables you to affinitize client connections. When clients create session objects, state is stored on Analysis Services. If a later client request, relying on the same session state, is redirected to a different server, the OLE DB provider throws an error. However, even if you run affinity, you may still have to force clients off the server when processing needs to commit. For more information about the **ForceCommitTimeout** setting, see the locking section.

3.5.5.3.3 Windows Network Load Balancing

The Microsoft load-balancing solution is Network Load Balancing (NLB), which is a feature of the Windows Server operating system. With NLB, you can create an NLB cluster of Analysis Services servers running in multiple-host mode. When an NLB cluster of Analysis Services servers is running in multiple-host mode, incoming requests are load balanced among the Analysis Services servers.

3.5.5.3.4 Analysis Services Load Balancer

Analysis Services is used extensively inside Microsoft to serve our business users with data. As part of the initiative to scale out our Analysis Services farms a new load balancing solution was built. The advantages of this solution are that you can load balance on the database level and that you use a web API to control each database and the users connected to it. This customized Analysis Services load balancing solution also allows fine control over the load balancing algorithm used. Be aware that moving large datasets over the web API has a bandwidth overhead, depending on how much data is requested by user queries. Measure this bandwidth overhead as part of the cube test phase.

References:

- Analysis Services Load Balancing Solution - http://sqlcat.com/technicalnotes/archive/2010/02/08/aslb-setup.aspx

3.5.5.4 ROLAP Scale Out

With the query/processing architecture you can get the update latency of cubes down to around 30 minutes, depending on workload. But if you want to refresh data faster than that, you have to either go fully ROLAP or use a hybrid of one of the strategies discussed earlier and ROLAP partitions.

In a pure ROLAP setup, you only process the dimensions and redirect all measure group queries directly to a relational database. The following diagram illustrates this.

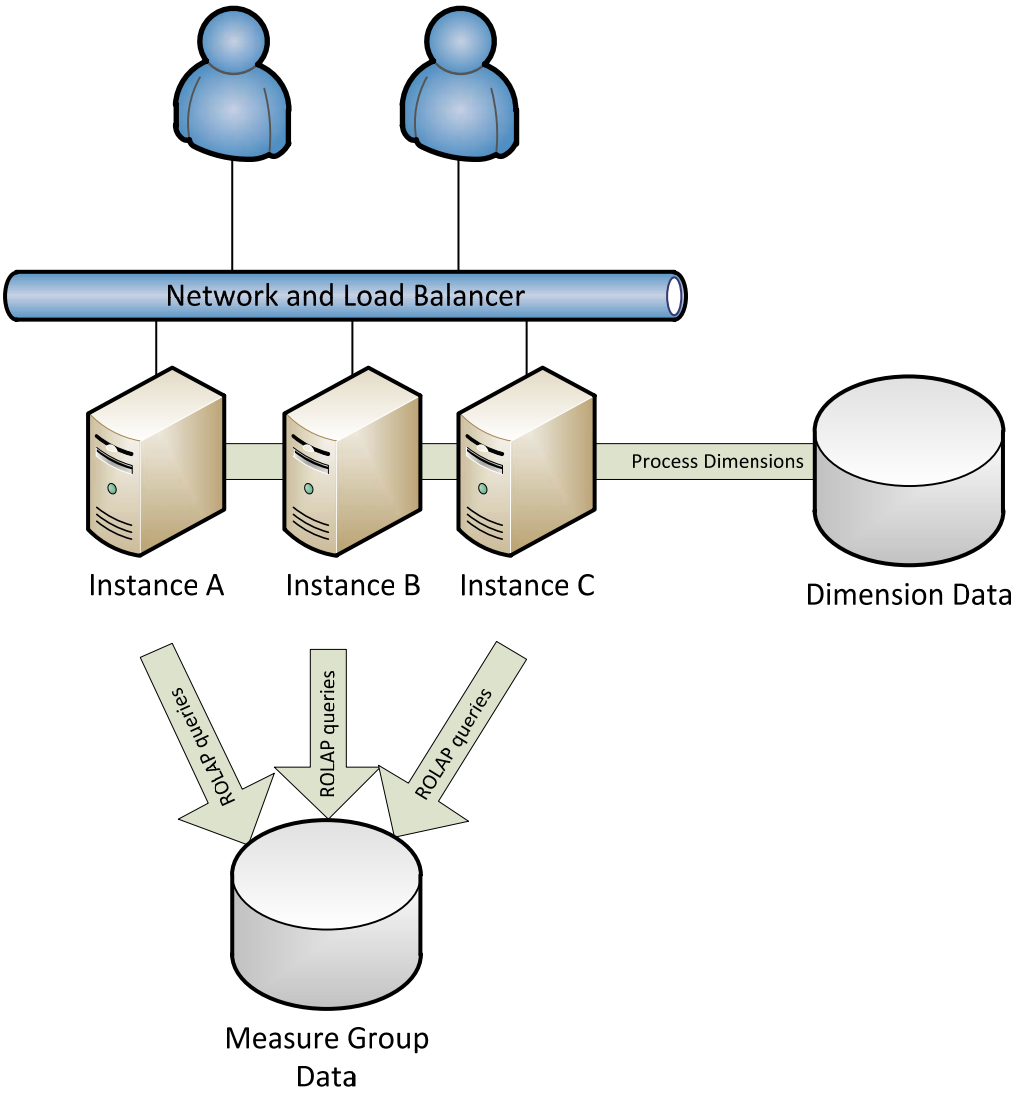


Figure 74 - ROLAP scale-out

In a ROLAP system like this, you have to consider the special requirements mentioned later in this document. You should also make sure that your relational data store is scaled to support multiple Analysis Services query servers.

An interesting hybrid between MOLAP and ROLAP can be built by combining the ROLAP scale-out with either the dedicated processing architecture or the query/processing flipping architecture. You can store data that changes less frequently in MOLAP partitions, which you either process or copy to the query servers. Data that changes frequently can be stored in ROLAP partitions, redirecting queries directly to the relational source. Such a setup can achieve very low update latencies, all the way down to a few seconds, while maintaining the benefits of MOLAP compression.

References:

- Analysis Services ROLAP for SQL Server Data Warehouses - http://sqlcat.com/whitepapers/archive/2010/08/23/analysis-services-rolap-for-sql-server-data-warehouses.aspx

3.6 Server Maintenance

When you move an Analysis Services instance to production, there are some regular maintenance tasks you should configure. This section describes those tasks.

3.6.1 Clearing Log Files and Dumps

During server operations, Analysis Services generates a log file containing data about the operation. This log file is located in the folder described by the LogDir in Msmdsrv.ini – the default location being <Install dir>\OLAP\Log. This log grows extremely slowly and you should generally not need to clean it up. If you *do* need to reclaim the disk spaced used by the log file, you have to stop the service to delete it.

In the <Install Dir>\OLAP\log folder, you may also find files with extension *.mdmp. These are minidump files generated by the Analysis Services and are typically a few megabytes each. These files get generated when undetected deadlocks happen inside the process or when there is aproblem with the service. The files are used by Microsoft Support to investigate stability issues and errors in the server. If you are experiencing any such behavior, you should collect these minidump files for use during case investigation. Periodically check for these files, and clean them up after any Microsoft Support case you have open is resolved.

3.6.2 Windows Event Log

Analysis Services uses the Windows event logs to report server errors, warnings, and information. The Application Log is used for most messages, but the System Log is also used for events that are related to Service Manager.

Depending on your server configuration, event logs may be configured to be cleaned manually. Make sure that this is a regular part of your maintenance. Alternatively, you can configure the event log to overwrite older events when the log is close to full. In both cases, make sure you have enough disk space to hold the full event log. The following illustration shows how to configure the event log to overwrite older events.

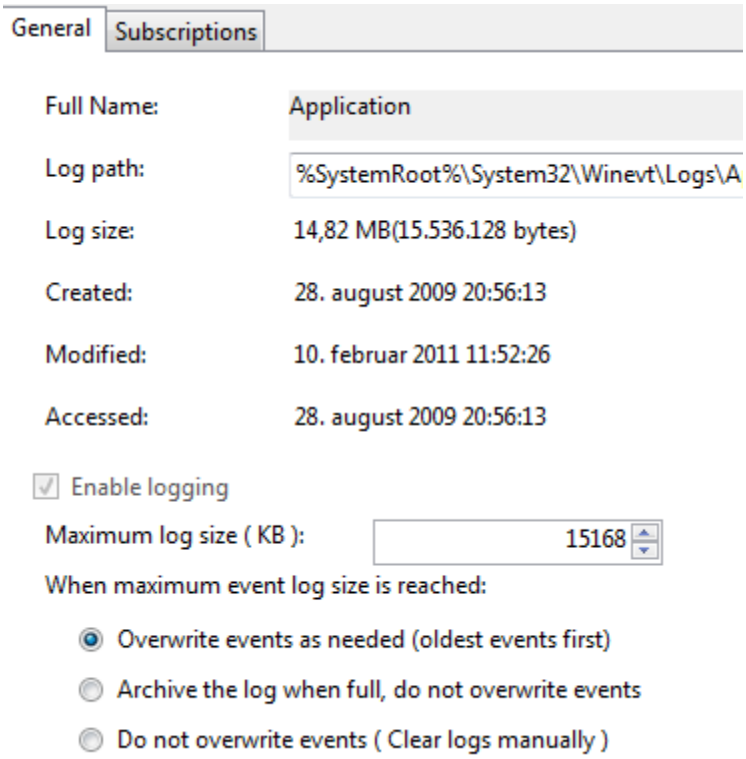


Figure 75 - Recycling the event log

3.6.3 Defragmenting the File System

As described in the I/O section, there can be an advantage in defragmenting the files storing a cube, especially after a lot of changes to the partitions and dimensions. Running disk defrag will have a measurable impact on your disk subsystem performance, and depending on the hardware you run on, this may affect user response time. You can consider running defragmentation on the server during off-peak hours or in batch windows.

Note that the defrag utility retains the work done, even when it does not run to completion. This means that you can do partial defragmentation spread over time.

3.6.4 Running Disk Checks

Running disk checks (using ChkDsk.exe) on the Analysis Services volume gives you the confidence that no undetected I/O corruption has occurred. How often you want to do this depends on how often you expect the I/O subsystem to create such errors without detecting them – this varies by vendor and disk model.

Note that ChkDsk.exe can run for a very long time on a large disk volume, and that it will have an impact on your I/O speeds. Because of this, you may want to use a SAN snapshot of the LUN and run ChkDsk.exe on another machine that mounts the snapshot.

In both cases, you should be able to detect disk corruption without touching the live system. If you detect irreparable corruption, you should consider restoring the backups as per the previous section.

References:

- Chkdsk (http://technet.microsoft.com/en-us/library/bb491051.aspx)

3.7 Special Considerations

Using certain features of Analysis Services cubes can lead you down some design paths that require extra attention to succeed. This section describes these special scenarios and the considerations that apply when you encounter them.

3.7.1 ROLAP and Real Time

This section deals with issues that are specific to BI environments that do not have clearly defined batch windows for loading data. If you have a cube that updates data at the same time that ETL jobs are running on the source data or while users are connected, you need to pay special attention to certain configuration parameters.

As described in the Locking and Blocking section, processing operations generally take an instance-wide lock. This will typically prevent you from designing MOLAP systems that are updated more frequently than approximately every 30 minutes . Such a refresh frequency may not be enough, and if this is your scenario, ROLAP is the path forward, and you should be aware of the special considerations that apply. You should also be aware that a ROLAP partition can put significant load on the underlying relational source, which means you should involve the DBA function to understand this workload and tune for it.

3.7.1.1 Cache Coherency

By default, the storage engine of Analysis Services caches ROLAP subcubes in the same way it caches MOLAP subcubes. If the relational data changes frequently, this means that queries that touch ROLAP partitions may use a combination of the relational source and the storage engine cache to generate the response. If the relational source has changed since the cache entry was added to the storage engine, – this combination of source data can lead to results that are transactionally inconsistent from the perspective of the user because they represent an intermediate state of the system. There are of course ways to resolve this coherency issue, depending on your scenario.

Changing the connection string– It is possible to add the parameter **Real Time OLAP=true** to the connection string when the cube is accessed. Setting this parameter to **true** causes all relevant storage caches to be refreshed for every query run on that connection – including the caches generated by MOLAP queries. Note that this change can cause a significant impact on both query performance and concurrency. You should test it carefully. However, it gives you the most up-to-date results possible from the cube, because Analysis Services is essentially used as a thin MDX wrapper on top of the relational source in this mode.

Blowing away cachesat regular intervals – you can either use an XMLA script to clear the cache or you can use query notifications (set in the ProActive caching properties of the partition). This allows you to clear the storage engine caches at regular intervals. Assuming you time this cache clearing with relational data loads, this gives users a consistent view that is updated every time the cache is cleared. Although this does not give you the same refresh frequency as the **Real Time OLAP=true** setting, it does have a smaller impact on user query performance and concurrency.

As you can see from these two options, going towards real-time cubes requires you to carefully consider tradeoffs between refresh frequencies and performance. Full coherency is possible but expensive. However, you can get a loose coherency that is much cheaper. Analysis Services supports both paradigms.

3.7.1.2 Manually Set Partition Slicers on ROLAP Partitions

When Analysis Services processes the index on a MOLAP partition, it collects data about the attributes in that partition. Assuming the data matches only one attribute value, an automatic slicer is set on the partition, eliminating it from scans that do not include that attribute value. For example, if you process a partition that has data from December 2008 only, Analysis Services detects this slicing and only accesses that partition when queries request data in that time range.

Because ROLAP data resides outside of Analysis Services, the automatic slicer functionality is not used. Unless you set the slicer manually (which can be done from both Visual Studio and SQL Server Management Studio) every query has to touch every ROLAP partition. It is therefore a good practice to always set slices on ROLAP partitions when they are available.

3.7.1.3 UDM Design

When you design for ROLAP access, it is generally a good idea to keep the UDM as simple as possible. The gives the relational engine the best possible conditions for optimizing for the query workload. The following table lists some optimizations you should consider when switching a cube to ROLAP mode.

Existing feature usage	ROLAP redesign
Reference dimensions	Switch to a pure star schema to eliminate unnecessary joins and provide relational engine with optimal conditions for query execution.
Parent/child dimensions	Normalize the parent-child dimension (for more information about how to do this, see the references for this section).
Many-to-many dimensions	Reduce intermediate table sizes using matrix compression.
Query binding of partitions	Switch to table binding. Consider binding to a view if queries are needed.
Query binding of dimensions	Implement the result of the query in the relational source instead.
Aggregates	Consider reducing the number of aggregates. Be aware of conditions and features in the relational engine so that you fully understand the tradeoffs. For example, in SQL Server 2008 R2, it is often a good idea to focus on aggregates that are targeted only at the leaf level and/or **[all]** level of attributes).
MDX calculations	Optimize carefully, and avoid cell-by-cell operations in large ROLAP partitions.
ROLAP dimensions	If at all possible, use MOLAP dimensions. MOLAP dimensions have much better performance than ROLAP dimensions and if you run regular **ProcessAdd** operations, you can keep them up to date at short refresh intervals.

You should work closely with the BI developers when troubleshooting ROLAP cubes. It is imperative to get the design right and follow the guidance in Part 1 of this book.

References:

- Analysis Services Parent-Child Dimension Naturalizer on CodePlex – http://pcdimnaturalize.codeplex.com/
    - Also available from BIDS helper: http://bidshelper.codeplex.com
- Analysis Services Many-to-Many Dimensions: Query Performance Optimization Techniques – http://www.microsoft.com/downloads/en/details.aspx?FamilyID=3494E712-C90B-4A4E-AD45-01009C15C665&displaylang=en
- Analysis Services ROLAP for SQL Server Data Warehouses- http://sqlcat.com/whitepapers/archive/2010/08/23/analysis-services-rolap-for-sql-server-data-warehouses.aspx

3.7.1.4 Dimension Processing

Real-time dimension processing can present a special challenge. In a batch-style warehouse, you are in control of when inserts and updates happen – which means you can typically process the dimension after the relational source is done refreshing data. But if you are designing a cube on top of a real-time source, the relational data may change while you are processing a dimension. Dimension processing is by default executed as many concurrent SQL Server queries, as described in the Optimizing Processing section. Consider this sequence of events:

1. The customer dimension contains customers from all of the United States, but no customers from the rest of the world.
2. Dimension **ProcessAdd** starts.
3. Analysis Services sends query **SELECT DISTINCT Zip, City FROM Dimension** to the relational source. This query reads all current attribute **Zip** and **City** values.
4. The relational source inserts a new row, **City** = **Copenhagen**, in **Country = Denmark.**
5. Analysis Services, reading the next level of the hierarchy, sends the query **SELECT DISTINCT City, Country**.
6. The **City** member **Copenhagen** Is returned in the second query, but because it was not returned in the first, Analysis Services throws an error.

While this scenario may sound uncommon, we have seen it at several customers that design real-time systems. There are some ways to avoid these conditions.

ByTable processing- By setting the **ProcessingGroup** property of the dimension to be **ByTable** you will change howAnalysis Services behaves during dimension processing. Instead of sending multiple SELECT DISTINCT queries, the processing task will instead request the entire table with one query. This allows you to get a consistent view of the dimension, even under concurrency in the relational source. However, this setting has a drawback, namely that you will need to keep all the source dimension data in memory while the dimension is processing. If the server is under memory while this happens, paging can occur, which may cause a slowdown of the entire system.

MARS and Snapshot - If you are processing on top of a SQL Server data source, you can use Multiple Active Result Sets (MARS) and snapshot isolation to process the dimension and get a consistent view of the data even under updates.

Configuring MARS and Snapshot processing requires a few configuration changes to the data source view and relational database. First, in the data source properties, change the data source view to use snapshot isolation.

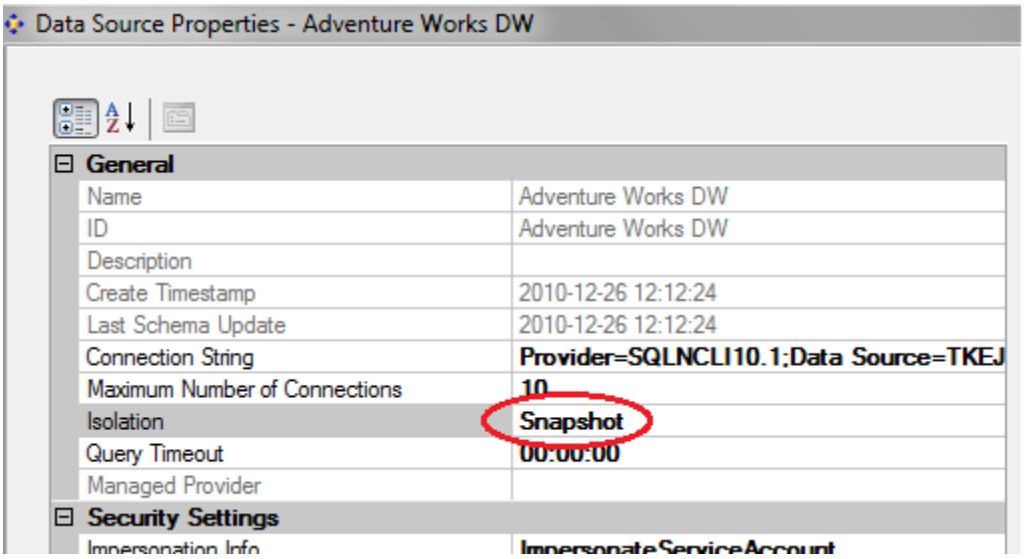


Figure 76 - Setting the Data Source to Snapshot

Second, enable MARS in the connection string of the data source view.

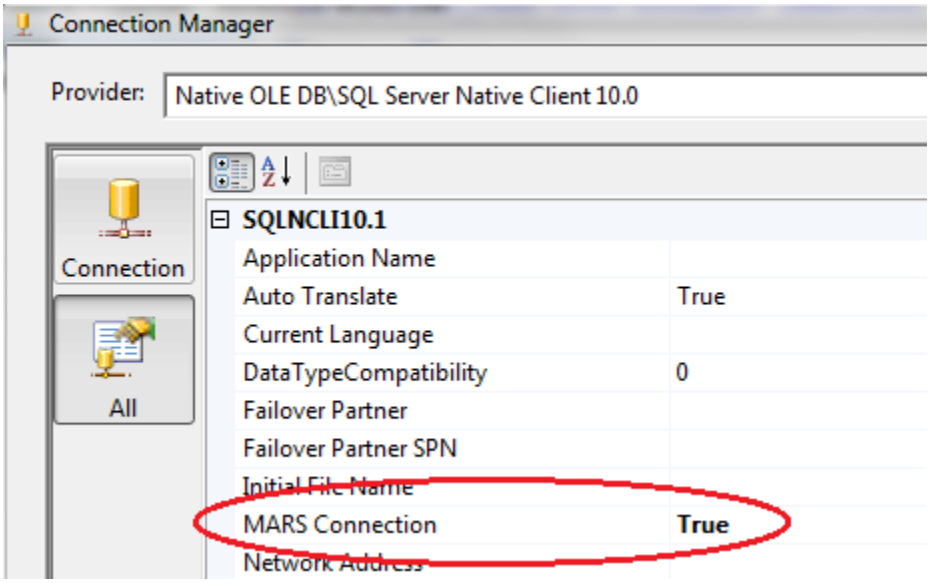


Figure 77 - Setting MARS in a DSV

And finally, enable either snapshot or read committed snapshot isolation in the SQL Server database.

```
ALTERDATABASE [Database]
SETREAD_COMMITTED_SNAPSHOTON
```

Processing now uses MARS, and snapshots generate a consistent view of the dimension during processing.

Understand that maintaining the snapshot during processing, as well as streaming the data through MARS, does not provide the same performance as the default processing option.

Maintaining consistency relationally – If you want to both maintain the processing speed and avoid memory consumption in the Analysis Services service, you have to design your data model to support real-time processing. There are several ways to do this, including the following:

- Add a timestamp to the rows in the dimension table that shows when they are inserted. During processing, only read the rows higher than a certain timestamp.
- Create a database snapshot of the relational source before processing.
- Manually create a copy of the source table before processing on top of the copy. The original can then be updated while the copy is being accessed by the cube.

References:

- Multiple Active Result Sets (MARS) - http://msdn.microsoft.com/en-us/library/ms345109(v=sql.90).aspx
- Using ByAttribute or ByTable Processing Group Property with Analysis Services 2005 - http://blogs.msdn.com/b/sqlcat/archive/2007/10/19/using-byattribute-or-bytable-processing-group-property-with-analysis-services-2005.aspx?wa=wsignin1.0

3.7.2 Distinct Count

Distinct count measures behave differently than other measures. Because data in a distinct count measure is not additive, a lot more information must be stored on disk. According to best practice, a measure group that has a distinct count measure should only have that single measure and no others. While additive measures compress very well, the same is not true for distinct count measures. This means that leaf-level data of the measure group takes up more disk space.

Targeting good aggregates for distinct count measures can also be difficult. Although aggregates for additive measures can be used by queries at higher granularities than the aggregate, the same does not apply for distinct count measures.

The combined effects of big measure groups and less useful aggregates means that queries that run against distinct count data often cause a significant amount of I/O operations and simply run longer than other queries. This is expected and part of the nature of distinct count data. However, there are some optimizations you can make that can greatly speed up both queries and processing of distinct count measures.

3.7.2.1 Partitioning for Distinct Count

Recall that for additive measures, it is generally recommended that you partition by time and sometimes by another dimension. This partition strategy is described in the Nonbreaking Cube Changes section. Distinct count measures are an exception to this rule of thumb. When it comes to distinct count, it is often a good idea to partition by the values of the distinct count measure itself. Analysis Services keeps track of the measure values in each partition, and assuming the intervals are not overlapping, it can

benefit from some parallelism optimizations. The basic idea is to create partitions, typically one per CPU core in the machine, that each contain an equal-sized, nonoverlapping interval of measure values. The following picture illustrates these partitions.

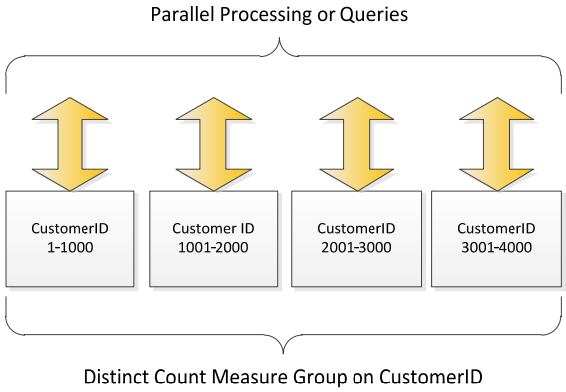


Figure 78 - Distinct Count Partitioning on a 4-Core Server

You can still apply a date based partition schema in addition to the distinct count partitioning. But if you do, make sure that queries do not cross the granularity level of this date range, or you lose part of the optimization. For example, if you do not have queries across years, you may benefit by partitioning by both year and the distinct count measure. Conversely, if you have queries that ask for data at the year level, you should not partition by month and the distinct count measure.

The white paper in the References section describes the partition strategy for distinct count measures in much more detail.

References:

- Analysis Services Distinct Count Optimization - http://www.microsoft.com/downloads/en/details.aspx?FamilyID=65df6ebf-9d1c-405f-84b1-08f492af52dd&displaylang=en
    - Describes the partition strategy that speeds up queries and processing for Distinct Count Measure groups

3.7.2.2 Optimizing Relational Indexes for Distinct Count

Analysis Services adds an ORDER BY clause to the distinct count processing queries have. For example, if you create a distinct count measure on **CustomerPONumber** in **FactInternetSales,** you get this query while processing.

```
SELECT…FROM FactInternetSales
ORDERBY [CustomerPONumber]
```

If your partition contains a large amount of rows, ordering the data can take a long time. Without supporting indexes, the query plan looks something like this.

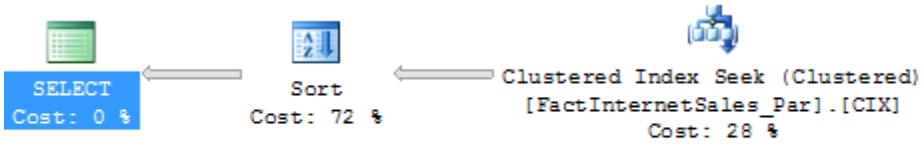


Figure 40 Relational sorting caused by distinct count

Notice the long time spent on the Sort operation? By creating a clustered index sorted on the distinct count column (in this case **CustomerPONumber**), you can eliminate this sort operation and get a query plan that looks like this.

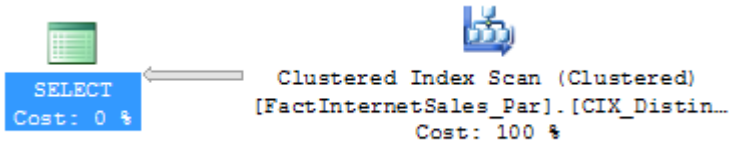


Figure 41 Distinct count query supported by a good index

Of course, this index needs to be maintained. But having it in place speeds up the processing queries.

3.7.3 Many-to-Many Dimensions

Many-to-many dimensions are a powerful feature of Analysis Services cubes. They enable easy solutions for some complex, yet common scenarios in dimensional modeling. When cubes resolve many-to-many queries, the join with the intermediate table is done in Analysis Services memory and during query time. If the intermediate table is large, especially if it larger than memory, these queries can take a long time to respond. We recommend that you use many-to-many dimensions only if the intermediate table fits in memory. The links in the References section describe some techniques that enable you to reduce the memory consumption of the intermediate table.

References:

- Analysis Services Many-to-Many Dimensions: Query Performance Optimization Techniques – http://www.microsoft.com/downloads/en/details.aspx?FamilyID=3494E712-C90B-4A4E-AD45-01009C15C665&displaylang=en
- BIDS Helper has tools to estimate benefits of the Many-to-many compression described in the section:
    - http://bidshelper.codeplex.com/wikipage?title=Many-to-Many%20Matrix%20Compression

- Many-to-many project- http://www.sqlbi.com/manytomany.aspx
    - Design patterns for many-to-many dimensions

4 Conclusion

This document provides guidance for creating and maintaining Analysis Services cubes that run in a production environment. In Part 1, you learned best practices for developing cubes and dimensions that are fast to process and query. Part 2 explored performance tuning from the standpoint of cubes that are already in production and not easily modified.

For more information, see:

http://sqlcat.com/: SQL Customer Advisory Team

http://www.microsoft.com/sqlserver/: SQL Server Web site

http://technet.microsoft.com/en-us/sqlserver/: SQL Server TechCenter

http://msdn.microsoft.com/en-us/sqlserver/: SQL Server DevCenter

If you have any suggestions or comments, please do not hesitate to contact the authors. You can reach Thomas Kejser at tkejser@microsoft.com and Denny Lee at dennyl@microsoft.com.

Did this paper help you? Please give us your feedback. Tell us on a scale of 1 (poor) to 5 (excellent), how would you rate this paper and why have you given it this rating? For example:

- Are you rating it high due to having good examples, excellent screen shots, clear writing, or another reason?
- Are you rating it low due to poor examples, fuzzy screen shots, or unclear writing?

This feedback will help us improve the quality of white papers we release.

Send feedback.