Microsoft

SQL Server Technical Article

Title: Performance Tuning of Tabular Models in SQL Server 2012 Analysis Services

Writer: John Sirmon, Greg Galloway (Artis Consulting), Cindy Gross, Karan Gulati

Technical Reviewers: Wayne Robertson, Srinivasan Turuvekere, Jeffrey Wang, Lisa Liu, Heidi Steen, Olivier Matrat

Technical Editor: Jeannine Takaki

Published: July 2013

Applies to: SQL Server 2012

Summary: Tabular models hosted in SQL Server 2012 Analysis Service provide a comparatively lightweight, easy to build and deploy solution for business intelligence. However, as you increase the data load, add more users, or run complex queries, you need to have a strategy for maintaining and tuning performance. This paper describes strategies and specific techniques for getting the best performance from your tabular models, including processing and partitioning strategies, DAX query tuning, and server tuning for specific workloads.

Copyright

Contents

1. Introduction

Tabular models have grown in popularity since their introduction as part of SQL Server 2012 Analysis Services, because they are lightweight, flexible, and easy to learn, letting IT professionals and business users alike rapidly build and experiment with data models.

However, because the in-memory data model is a comparatively new development, best practices are just emerging. This paper shares what the authors have learned about how to design efficient models and optimize them for the desired performance.

The goals of this paper are as follows:

[1] Describe the tabular model architecture to increase understanding of the factors that affect performance.

We will provide enhanced visibility into the internals of tabular models and the usage of system resources by describing how processing and queries interact with internal objects in a tabular model.

[2] Prescribe tuning techniques and strategies specific to querying, processing, partitioning, and server configuration.

In addition to general performance advice, we have described some common contexts in which performance tuning is likely to be required.

[3] Provide a roadmap for optimizing performance in some common scenarios.

A set of use cases is mapped to the discussions in each major section (queries, processing, partitioning, and server configuration) to help you understand how to identify and solve some problems in some common scenarios. If you don't have much time to read and just want to get started solving a particular problem, see this section first.

Goals and Scope

The goal of this paper is to help designers of tabular models understand the decisions (and trade-offs) involved in getting better performance. The process of optimization includes these steps:

Understanding the behaviors for which the architecture is optimized. This paper briefly describes the architecture of an Analysis Services instance running in tabular mode, and explains the design decisions made by the product team. By understanding the behaviors for which the engine is optimized, as well the rationale behind them, you can design solutions that build on the unique strengths of the tabular model architecture, and avoid common design problems.

Optimizing the model design and avoiding common problems. A tabular model might be a blank slate, but it is not an unbounded one. This paper aims to help the model designer make informed choices

about what kind of data to use (and to avoid), how to optimize memory usage, and how to design and optimize DAX formulas.

Planning data refresh and partitioning. In very large models, reloading the model with fresh data within the refresh window requires that you understand the tradeoffs involved in achieving the best performance for your workload and design an appropriate partitioning and processing strategy.

Designing queries and monitoring their performance. An important part of managing performance is understanding user queries that are slow, and how to optimize them. The paper describes methods for monitoring and troubleshooting DAX and MDX queries on tabular models.

Scenarios

To make it easier to navigate this information, several common performance scenarios are listed, with links to the sections you should read first to understand and correct this particular performance problem:

1. Slow queries
2. Server optimization
3. Processing problems
4. Disproportionate memory usage
5. Working with large models
6. Tabular performance 101

Additional Tips

In addition to the detailed technical background and walkthroughs, we've highlighted some related tips using the following graphic.

If your tabular model is giving you headaches, join the Business Analytics Virtual Chapter of PASS. You'll find great resources, and learn from each other.

Scope Notes

This paper does not include the following topics:

- **Hardware sizing and capacity planning.** For information about how to determine the amount of memory or CPU resources required to accommodate query and processing workloads, download this white paper: Hardware Sizing a Tabular Solution (SSAS).
- **Performance of Multidimensional models**. The recommendations in this paper do not apply to Multidimensional models. See the Resources section for a list of the OLAP performance guides.
- **DAX against Multidimensional models.** See the SQLCAT site for additional articles to follow.

- **Performance tuning of PowerPivot models.** You can use some of the same tools described in this paper to monitor PowerPivot models that are hosted in SharePoint. However, this guide does not specifically address PowerPivot models.
- **Tabular Models in DirectQuery Mode.** Optimizing the performance of DirectQuery models requires knowledge of the network environment and specifics of the relational data source that are outside the scope of this paper. See this white paper for more information: Using DirectQuery in the Tabular BI Semantic Model (http://msdn.microsoft.com/en-us/library/hh965743.aspx).
- **Using Hadoop Data in Tabular Models.** You can access "Big Data" from a Hadoop cluster in a tabular model via the Hive ODBC driver, or the Simba ODBC drivers. See the Resources section for more information. Additionally, we recommend these resources:
    - White paper on SSIS with Hadoop: http://msdn.microsoft.com/en-us/library/jj720569.aspx
    - Best practices guide for Hadoop and data warehouses: (http://wag.codeplex.com/releases/view/103405)
- **Operating Tabular Models in Windows Azure.** Running a tabular model in Windows Azure using Infrastructure as a Service (IaaS) is an increasingly attractive and inexpensive way to operate a large tabular model on a dedicated environment, but out of scope of this paper. For additional evidence and experience on this topic, watch the SQLCAT site .

(Editorial note: Although it is not standard usage, in this paper we've opted to capitalize "Tabular" and "Multidimensional" throughout, since we'll be comparing them frequently.)

Requirements

SQL Server 2012 Analysis Services. Tabular models are included as part of the SQL Server 2012 release. You cannot create tabular models with any previous edition of SQL Server Analysis Services. Moreover, models that you created in an earlier edition of PowerPivot must be upgraded to work with SQL Server 2012.

SQL Server 2012 Service Pack 1. Additional performance improvements were released as part of SQL Server 2012 Service Pack 1. For information about the improvements specific to SP1, see this blog post by the Analysis Service product team. Note that Service Pack 1 is also required for compatibility with Excel 2013. In other words, if you want to prototype a model using the embedded tabular database (PowerPivot) provided in Excel 2013, and then move it to an Analysis Services server for production, you must install Service Pack 1 or higher on the Analysis Services instance.

For more information about this service pack, see the SP1 release notes, in the MSDN Library.

Tabular Performance Scenarios

Each of these scenarios tells a story about someone trying to solve a tabular model performance problem. You can use the stories as a shortcut to quickly find appropriate information for a problem that you might be facing. Use the links in the right-hand column to get more in-depth background and troubleshooting tips. The final two sections provides links to best practices for each scenario, and additional readings.

Use Case 1: Slow Queries

Queries are slow		
Problem Statement	"I have to demo to the CEO on Monday and the one key report is unbearably slow. I don't know what to do! My boss is breathing down my neck to speed up this key report. How do I figure out what's wrong?"	
Process and Findings	First, I used the four W's method—Who, What, When, Where—to get started with troubleshooting. I decided to concentrate on one typical query to simplify the analysis.	Troubleshooting
	Based on the advice in the query tuning section, I discovered that the main query bottleneck was in the storage engine.	Bottlenecks
	I studied the VertiPaq queries and saw that the LASTDATE function had caused the storage engine to need the formula engine's help by performing a callback. I realized that I can do the LASTDATE function earlier in the calculation so it happens once instead of millions of times during the VertiPaq scan.	Caching and callbacks Walkthrough
	We also set up a trace to monitor the longest queries and check memory usage.	Set up a trace
	Report performance is now snappy and I look like a hero.	
Best Practices	• Rule out network or client issues, then determine source of bottleneck: storage engine vs. formula engine. • Test query performance on a cold and warm cache. • Embed expensive calculations in the model itself rather than in the query. • Study the DAX Query Plan and VertiPaq queries. Imagine how you would go about evaluating this calculation, and then compare your method to the query plan.	Bottlenecks Clearing the cache Move calculations into model DAX Query Plan
Further Reading	• Read all of Section 3. • See posts in Jeffrey Wang's blog. • View "Inside DAX Query Plan" by Marco Russo (SQLPASS 2012; requires registration on PASS website) • Assess impact of security on performance	Tuning Query Performance http://mdxdax.blogspot.com http://www.sqlpass.org/summit/2012/Sessions/SessionDetails.aspx?sid=2956 Row filters used in role-based security

Use Case 2: Server Optimization

What can we do to optimize the server?		
Problem Statement	"I am the DBA responsible for Analysis Services. Users are calling me all week complaining that their reports are slow. I don't have the budget for new hardware. Is there anything I can do?"	
Process and Findings	I checked and the latest service pack was not applied. One bug fix included in the service pack fixed the performance of a few reports.	Applying the Latest Service Packs and Cumulative Updates
	I also noticed that users frequently complained about performance at the top of every hour. This timing corresponds to the hourly model processing we perform. I concluded that long-running queries were slowing down the processing commits and making other users queue up. Users can always rerun a query if it gets cancelled, so I lowered ForceCommitTimeout to 5 seconds to make sure users weren't queued up for very long before the long-running query is cancelled by the processing commit.	Locking and blocking during the commit of processing transactions
	Next, I reviewed the server configuration settings. Other than ForceCommitTimeout, the default settings are probably optimal for us.	Performance Related Server Configuration Settings Scale-Out
	I used the CPU-Z tool to check that processors on the server were running at top speed, but found that the server was in power-saving mode. I changed the BIOS settings to disable this setting.	Disable Power-Saving Mode
	Now everyone says queries are running 30% faster!	
Best Practices	• Apply latest service packs and cumulative updates. • Perform model processing when no users are querying. • Consider scale-out architectures if one server is overloaded. • Review server configuration settings. • Ensure power-saving mode is disabled.	See links in Process and Findings
Further Reading	Consider scale-out architectures, to prevent processing from impacting user queries.	Hardware Sizing a Tabular Solution Scale-Out Querying for Analysis Services with Read-Only Databases Scale-Out Querying with Analysis Services Using SAN Snapshots Analysis Services Synchronization Best Practices

Use Case 3: Processing Problems

Processing takes too long

Problem Statement	"I have spent the last month building a large enterprise-wide Tabular model and the users love it! However, no matter what I try, I can't get the model to finish refreshing until 10am or later each morning. I'm about to throw in the towel. Do you have any suggestions I should try?"	
Process and Findings	Reading about processing transactions and locking, I realized I was doing it all wrong. I was attempting to process multiple tables in parallel by running one processing batch per table in parallel, but I learned you can't process multiple tables in the same database in parallel unless they are in the same Parallel tag. I added the tags, and after this simple change, processing is completing by 8am!	Locking and blocking during the commit of processing transactions
	After that, I decided to read up on all the other processing options, and learned that a ProcessFull on each table will perform duplicate work unnecessarily. I switched it to ProcessData followed by one ProcessRecalc and now processing completes at 7:45am.	Example of ProcessData Plus ProcessRecalc
	Using Profiler, I was able to determine that the Customer dimension table, which has 1 million rows, was taking 30 minutes to process. The SQL query behind this table included about a dozen joins to add extra customer attributes to the dimension. I worked with our ETL developer to materialize all but one of these columns. This turned out to be much more efficient since only about 10,000 customers change each day.	Get Rid of Joins
	The last column we decided not to import at all, but instead converted it to a calculated column with a DAX expression in the model. Now the Customer dimension processes in 30 seconds!	Changing DAX Calculations
	Finally, I partitioned the largest fact table by month and am now able to incrementally process the table by only processing the current month.	Partitioning scenarios in Tabular models
	After all the tuning, processing completes by 6:30am!	
Best Practices	• Tune the relational layer. • Avoid paging. • Use Profiler to determine the slowest step in processing. • Do ProcessData on each table followed by one ProcessRecalc. • Partition to enable incremental loading. • Optimize the network and packet size.	Optimizing the Relational Database Layer Paging Profiler Trace Events to Watch during Processing Example of ProcessData Plus ProcessRecalc Partitioning scenarios in Tabular models Optimizing the Network

Further Reading	• Read all of Section 4. • View video by Ashvini Sharma and Allan Folting on model optimization. • Read blog by Marco Russo on incremental processing.	Processing and partitioning Optimizing Your BI Semantic Model for Performance and Scale Incremental Processing In Tabular Using Process Add

Use Case 4: Disproportionate Memory Usage

Memory use is heavier than expected		
Problem Statement	"I'm responsible for all the Windows Server deployments in my office. On a server with Analysis Services Tabular deployed, users are complaining that the data is several days stale and I see only 100MB of available RAM right now! I suspect that refreshing the data is failing. I don't know much about Analysis Services and our Analysis Services developer is on vacation. Can I do anything?"	
Process and Findings	I see the SQL Agent job that processes the model is failing with an out of memory error. The job does a ProcessFull on the whole database. On reading the processing section of this whitepaper, I learned that ProcessFull requires a lot of memory since it keeps one copy of the old data in memory and one copy of the new data in memory while it's loading.	ProcessFull
	I checked with our users and they never query the model during the 5am model refresh. Knowing this allowed me to run a ProcessClear before the ProcessFull, reducing overall memory usage by 2x.	ProcessClear
	Also, I just moved a large SQL Server database to this server yesterday, so it is possible that SQL Server is taking up quite a bit more memory on the server than before. I took the advice from the Server Memory Plan section and put a memory cap on SQL Server to leave more memory for Analysis Services while still leaving enough memory for SQL Server to perform acceptably.	Reducing memory usage Server Memory Plan
	Now processing succeeds and users are happy.	
Best Practices	• Add RAM. • Put a memory cap on other large consumers of memory like SQL Server. • Remove or optimize high cardinality columns. • If users don't need to query the model during processing, perform a ProcessClear before processing to reduce memory usage. • Create a server memory plan and set memory limits appropriately. • Use MaxParallel to reduce parallelism and reduce memory usage during processing. • Study the DAX Query Plan for any queries using too much memory and try to optimize them.	Reducing Memory Usage during Processing Server Memory Plan DAX Query Plan MaxParallel High cardinality columns
Further Reading	• Read all of Section 5. • Read tips on how to reduce memory during processing.	Server tuning Reducing Memory Usage during Processing

Use Case 5. Working with Big Models

Model might be too big to process		
Problem Statement	"We are already running into problems loading the model, and we're still in the development phase with a small percentage of the data size we will have in production. Will we even be able to process this model in production?"	
Process and Findings	After skimming this white paper, we realized that we had several options: make improvements to the model, optimize the environment, and create a processing plan.	Section 4
	First, we cleaned up resources on the development server, by removing debug builds and clients that were interfering with the model. We turned off Flight Recorder and adjusted memory limits.	Server configuration settings
	Next, we did some troubleshooting, and decided to track object usage, and find out which objects take the most memory.	Troubleshooting process Common processing problems
	We also identified the slowest queries among the SQL statements that load data into the model. In general, our relational design was a bottleneck on processing. So rather than get a full set of data directly from the source, we created a staging DW with a smaller set of representative data for use during the iterative development cycle.	Optimizing the relational layer Relational Db strategy
	Once we were able to load the model and build some reports, we instrumented the report queries, and found we had some inefficient calculations. We tried some different approaches to calculations and were able to eliminate the slowest IF statements. Also, once we realized that volatile functions must be refreshed each time a dependent calculation is executed, we replaced a NOW() statement with a hidden date column.	Changing the data model Known performance issues Known Power View issues
	The model developers can now alter the model in near real-time, plus we learned some valuable lessons about managing our data sources and about testing and optimizing calculations. Before we go into production with full data we'll also implement a partitioning plan to manage the data load.	
Best Practices	• Use only the columns you need, and only the data you need. • Minimize the amount of copy-pasted data in the model. • Create a partitioning strategy.	Compression Changing the data model Partitioning for tabular
Further Reading	• Learn tips for working with very large models. • Learn how to build an efficient data model.	http://go.microsoft.com/fwlink/?LinkId=313247 http://office.microsoft.com/en-us/excel-help/create-a-memory-efficient-data-model-using-excel-2013-and-the-powerpivot-add-in-HA103981538.aspx

Use Case 6. Tabular Performance 101

Get me started with tabular performance tuning		
Problem Statement	"We're pretty new to tabular models and to Analysis Services. What sort of information should we collect about the model and the system, and how? What sort of stats should we track from the beginning, to avoid trouble when user traffic increases?"	
Process and Findings	The white paper got pretty technical in places, so we focused on architecture, and read up on what happens under the covers during model loading and queries.	Architecture Query architecture Processing
	For starters, we need to determine which objects are being used, which take the longest to process, and which consume the most memory. We also learned where the tabular model files are, and how to find and interpret them.	Sample query on AS traces Model files
	Though we hadn't had any problems with memory or processing yet, we realized that we'd imported a lot of unnecessary columns. We took the opportunity to optimize the model design by removing unused keys and some high cardinality columns.	Change the Data Model
	Finally we read up on common issues with queries, so we know what scenarios to avoid, and how to tune the client.	Common query performance issues
	Our model design changes alone shrunk memory usage by almost half. Power View is new to our organization but we now have a much better idea of how to develop efficient reports and how to train report users.	
Best Practices	Get a performance baseline, by measuring queries on a warm cache and on a cold cache. Get started with single-instance monitoring by using one of the sample Excel workbooks. Figure out your security requirements and how this might affect performance.	Cold and warm cache Server tuning Row filters
Further Reading	Your model might be fine now, but take some time to figure out whether your server can handle future growth. Read about processing, to make sure you can handle more data.	Hardware sizing guide Section 4

2. Analysis Services Architecture and Tabular Model Design

This section provides background about the architecture of Tabular models, to help you understand the way that Analysis Services operates when in Tabular mode, and to highlight the key design decisions that make Tabular models behave differently from Multidimensional models.

Architecture Overview

Analysis Services operating in Tabular mode re-uses almost all of the framework that supports Multidimensional models. The query engine and formula engines are shared, and only a different storage engine is accessed. Thus, it should be almost transparent for client applications whether the connection is to an instance running in Multidimensional or Tabular mode. Aside from slight differences in functionality, commands from the interfaces (such as the XMLA Listener, HTTP pump, API calls through ADOMD.NET, etc.) should be accepted in terms of the traditional Multidimensional (UDM) objects and the server will execute those commands against the particular storage engine being used.

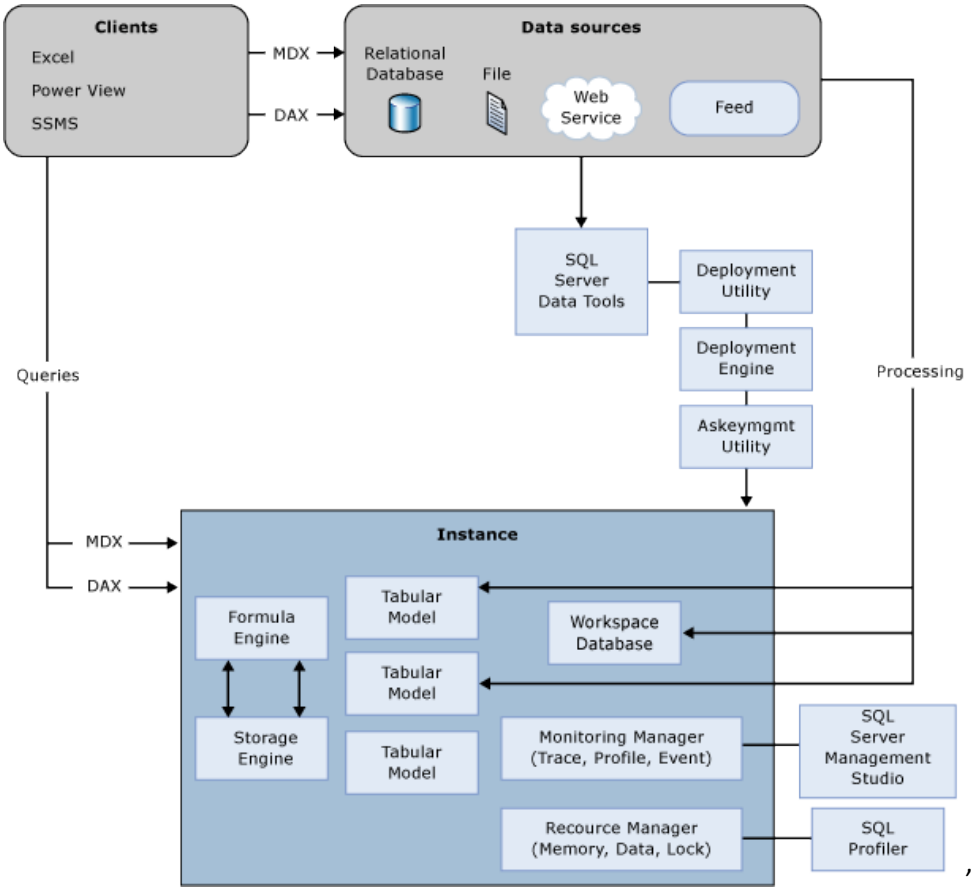


Figure 1. Single architecture for both multidimensional and tabular

The storage engine for Tabular models is officially named the "xVelocity in-memory analytics engine," which indicates that Tabular models are related to other in-memory technologies in the SQL Server product. However, this marketing nomenclature is fairly recent, replacing the term VertiPaq which had been extensively used in the underlying code base. Because the audience of this whitepaper will be seeing Profiler trace events and Performance Monitor counters that use the term VertiPaq, for clarity we will use the term VertiPaq in this white paper, instead of the official term, "xVelocity."

Differences between Multidimensional and Tabular

The Tabular storage engine is different from the Multidimensional storage engine in important ways. We will name a few to highlight the differences.

One catalog (database) = one model. Whereas a Multidimensional database can contain multiple cubes, in Tabular mode an Analysis Services database contains exactly one model. However, the model can contain perspectives.

When installing Analysis Services, you must choose whether to install the instance in Tabular mode or in Multidimensional mode. As such, hosting some Tabular models and some Multidimensional models requires two instances. (A third mode supports Analysis Services tabular models hosted within SharePoint, but is not considered in this paper.)

In-memory technology: The Tabular storage engine stores all data in memory and persists it to disk so that the model can be loaded into memory after a server restart. When a change is made to the model metadata or when the model is processed, the data is updated in memory and then also committed to disk at the end of a transaction. The model must fit in memory, however if memory needs are greater than available memory during processing or expensive queries, the server can also page portions of the model to pagefile.sys. This behavior is discussed in more detail in section 5.

In contrast, the traditional Multidimensional storage engine (MOLAP) is designed to efficiently retrieve massive amounts of pre-aggregated data from disk.

No aggregations. Column-based storage. The Multidimensional storage engine (MOLAP) uses row-based storage, keeping all the values from a single row together on disk. The MOLAP storage engine can pre-calculate and store aggregations on disk to minimize I/O and CPU cycles during queries.

In contrast, the Tabular storage engine uses column-based storage, which stores the distinct list of values from a column separate from other columns. Given that most analytic queries reference only a few columns from a table, and the column data is highly compressed, columnar storage is very efficient for summary-level analytic queries.

Both kinds of databases are described with XML for Analysis (XMLA) metadata. When you study the XMLA that defines a Tabular model, it becomes clear that the metadata language is shared with Multidimensional models. Under the covers, Tabular model objects are defined in terms of dimensions, attributes, and measure groups, and the Tabular terminology (tables and columns) is an abstraction layer on top of XMLA.

However, most developers and administrators will never need to be aware of the underlying schema. These mappings are relevant only when writing complex automation code using Analysis Management Objects (AMO), the management API for Analysis Services. Discussing AMO in any depth is out-of-scope for this whitepaper.

Files and File System

The data files that are generated are stored in the folder specified in the server configuration settings file, discussed in detail in section 5). Analysis Services stores its tabular model databases as thousands of XML and binary data files, using these rules:

- Each database gets its own folder.
- Each table gets its own folder, which ends in .dim.
- Columns in tables can be split into multiple files, depending on the column size.

By inspecting this file structure, you can easily determine which objects in the database used the most disk space. For example, in the following diagram, the list of files in the directory has been sorted by file size, so that you can clearly see which columns of the Reseller Sales fact table take up the most space.

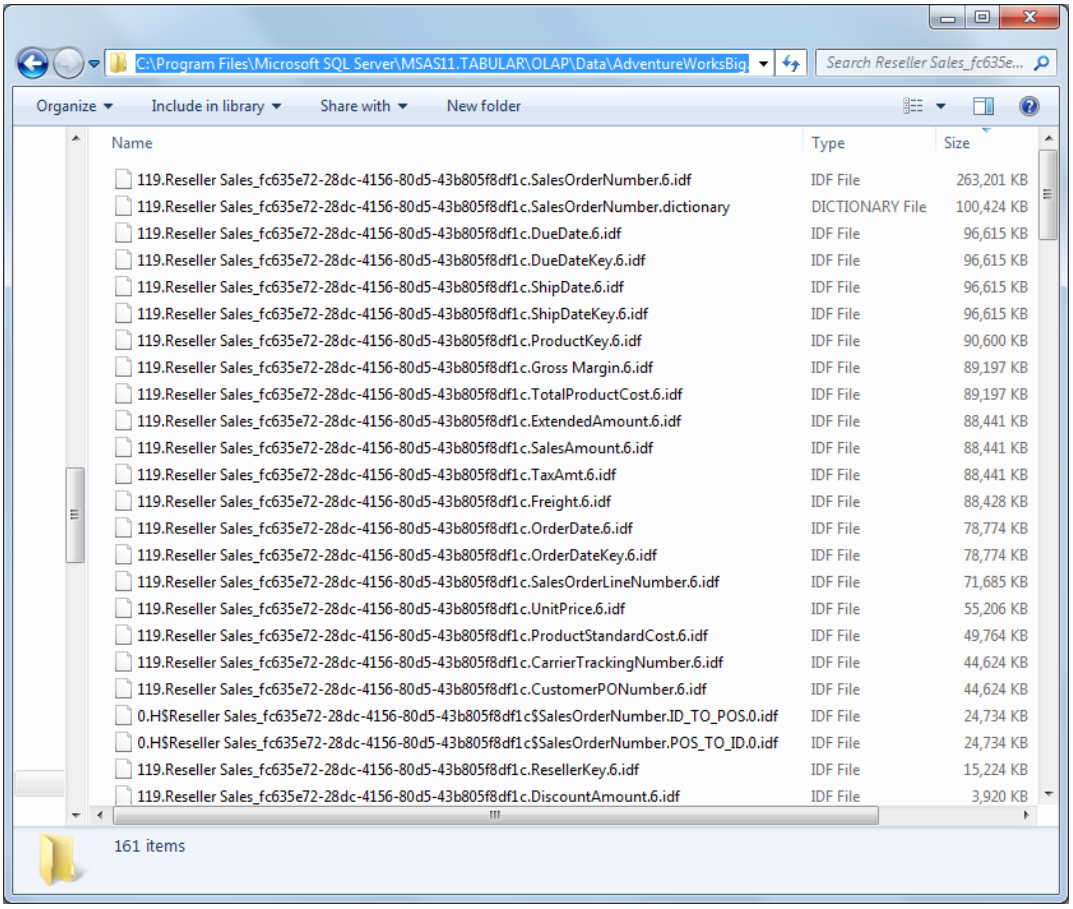


Figure 2. Files representing a tabular model

In this model, the SalesOrderNumber column clearly uses the most disk space, and by extension, the most memory. Looking at this list, you might wonder why that column is using so much memory.

This screenshot is from Adventure Works model where the Reseller Sales table has been expanded to contains 100 million rows. In this model, the SalesOrderNumber column has 6 million distinct values and is hash encoded, not value encoded. The high cardinality and hash encoding cause it to use the most memory of any column in the model.

You can use the following file name patterns when investigating files on disk:

Table 1. Files used to store tabular model

File Name Pattern	Description
***H$* files (e.g. 0.H$Reseller Sales*)**	Files related to the attribute hierarchies built on top of all columns
***U$* files (e.g. 44.U$Date*)**	User hierarchies which combine multiple columns into a single hierarchy.
***R$* files (e.g. 0.R$Reseller Sales*)**	Relationships between tables
***.dictionary**	The distinct list of values in a column which is encoded with hash encoding.
Other *.idf not included in the above	The segment data for a column.

For details on how to view these files and assess the amount of space used by the database —both in memory and on disk —see the Hardware Sizing Guide.

Columnar Storage

The following diagram is a high-level summary of the compressed columnar storage used in VertiPaq.

In this diagram, T1 and T2 represent two different tables in a tabular model.

- Table 1 has three data columns (C1, C2, C3 and one calculated column (CC1). A hierarchy is built using these columns and there is a relationship to another table (T2).
- Table 2 has only two columns, one calculated column, and one hierarchy. It participates in the relationship with T1.

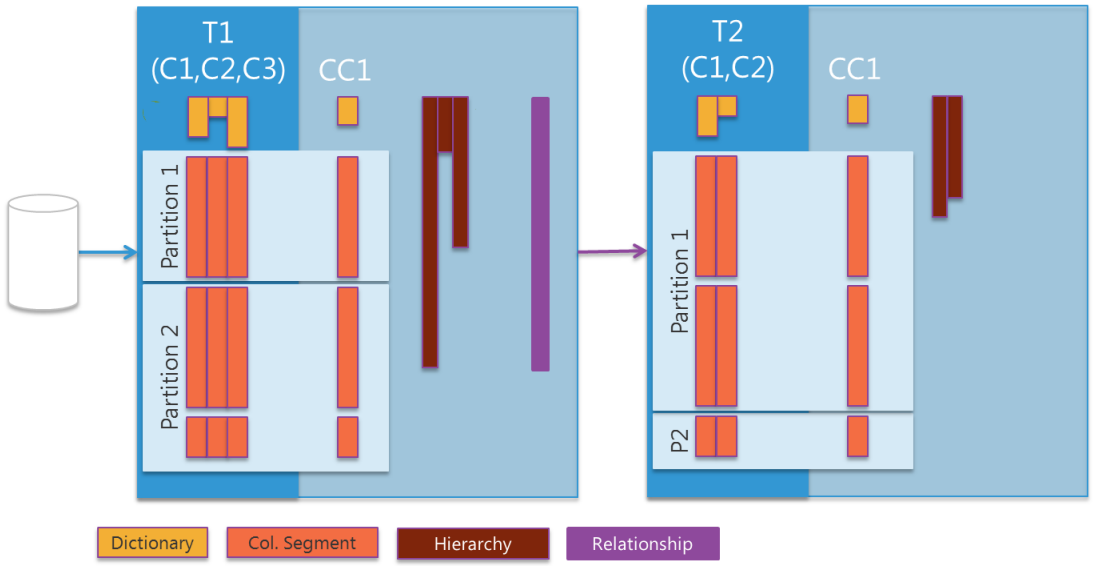


Figure 3. Objects in a tabular model

Dictionaries define the distinct values in a column and define an internal integer representation of every distinct value.

The **column segments** also include an index that identifies which value appears on each row.

Segments live within a **partition**. Each partition contain one or more segments per column depending on the number of rows. By default, segments contain about 8 million rows, however, there are a few exceptions to this rule, as discussed in section 4.

Calculated columns are evaluated and materialized during processing and thereafter are physically stored similar to other columns.

Hierarchies are built for all columns and for any multi-level user hierarchies defined by the developer.

Relationships represent foreign keys in the data model. Since T1 (the fact table) foreign keys to T2 (the dimension), relationship structures are added to T1 so that it can quickly refer to the related T2 rows.

Want to know more?
Section 4 describes in detail how and when each of these objects are built during model processing.
Section 4 also explains how the data loaded into the model is encoded, compressed, and stored in segments.

Summary—Tabular Architecture Design

Understanding the architecture can help you better align your model's design and performance with current hardware capacity, and determine the optimum balance between design-time performance and daily query and processing performance.

(1) In general, the developer should not need to tweak the system to get good performance.

Models should be fast, even without any sort of advanced tuning, because a Tabular model is just tables, columns, relationships, and calculated measures.

- **Pros:** The administrator doesn't need to know details of the model to optimize the system. For example, rather than force the administrator to know whether a particular column is used for aggregation vs. filtering, Analysis Services automatically builds a hierarchy for every column. As a result every column can be used as a slicer in a query.
- **Cons:** Less customization is possible than with MOLAP.

(2) Tabular requires different hardware than multidimensional.

Hardware requirements for a server that relies on in-memory technology are very different than the requirements for servers dedicated to traditional multidimensional models. If you need high performance from a tabular model, you might not want to run an instance of standard Analysis Services on the same computer.

(3) The system is optimized to support query speed.

A core design principle was that query performance is more important than processing performance. Query performance is achieved through scans of data in memory, rather than through painstaking optimization by the cube designer.

- **Pros:** Great performance out of the box.
- **Cons:** Loading data into a new model, or refreshing data in an existing model, can take a while. The administrator must consider the trade-offs – whether to optimize in-memory compression for better query performance, or to load and refresh the model faster at the expense of query time.

(4) The model design experience is fast and iterative.

To support working with real data in near real-time, the engine tracks dependencies and reprocesses only the tables that are affected by design changes.

- **Pros:** Because data is loaded into memory at design time, you can make changes to tables, columns, and calculations with less planning, and without requiring a reload of the entire model.
- **Cons:** The need to constantly recalculate and refresh data can make working with large models painful. You need different strategies when working with large amounts of data. If you

experience delays when modifying the model, it is best to work with small sets of data at first. For more information about how to manage large models, see the related white paper.

3. Tuning Query Performance

Tabular models use in-memory, columnar technology with high levels of compression and incredibly fast scan rates, which typically provides great query performance. However, there are many ways in which query performance can degrade due to complex and inefficient DAX calculations even on small models.

This section guides the model developer or administrator towards an understanding of how to troubleshoot query performance issues. We'll also provide a look under the covers into the DAX Query Plan, and discuss how you can optimize calculations.

Tabular Query Processing Architecture

Tabular models can be queried by using both MDX and DAX queries. The following diagram explains the underlying query processing architecture of Analysis Services when running in Tabular mode.

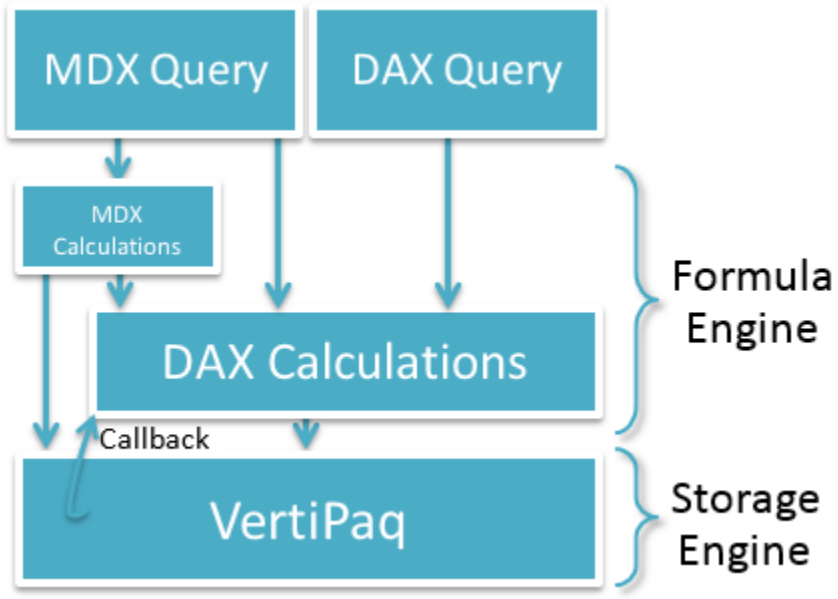


Figure 4. Query architecture

DAX queries can reference DAX calculations. These DAX calculation can reside in the model, in the session, or in the DEFINE clause of the DAX query. Power View reports generate DAX queries to gather the data they visualize.

MDX queries can reference **MDX calculations**, either in the session scope or in the WITH clause of the MDX query. MDX calculations can also reference DAX calculations, but the reverse is not true. MDX queries can directly refer to DAX calculations embedded in the model, and an MDX statement can also define new DAX calculations on the session scope or in the WITH clause of the MDX query.

MDX queries against Tabular models are not translated into DAX queries. MDX queries are resolved natively by the MDX formula engine, which can call into the DAX formula engine to resolve DAX measures. The MDX formula engine can also call directly into VertiPaq in cases such as querying dimensions. Excel generates MDX queries to support PivotTables connected to Analysis Services.

DAX calculations in the formula engine can request data from the VertiPaq storage engine as needed.

The **formula engine** allows very rich, expressive calculations. It is single-threaded per query.

The **storage engine** is designed to very efficiently scan compressed data in memory. During a scan, the entire column (all partitions and all segments) are scanned, even in the presence of filters. Given the columnar storage of VertiPaq, this scan is very fast. Because the data is in memory, no I/O is incurred.

Unlike the formula engine, a single storage engine query can be answered using multiple threads. One thread is used per segment per column.

Some very simple operations such as filters, sums, multiplication, and division can also be pushed down into the storage engine, allowing queries that use these calculations to run multi-threaded.

The formula engine commonly runs several VertiPaq storage engine scans, materializes the results in memory, joins the results together, and applies further calculations. However, if the formula engine determines that a particular calculation can be run more efficiently by doing the calculation as part of the scan, and if the calculation is too complex for the storage engine to compute on its own (for example, an IF function or the LASTDATE function), the VertiPaq storage engine can send a **callback** to the formula engine. Though the formula engine is single-threaded, it can be called in parallel from the multiple threads servicing a single VertiPaq scan.

Block Computation Mode

Block computation is an approach to performing calculations that is often more efficient than performing calculations in cell-by-cell mode.

- In block mode, sparsity in the data is efficiently handled so that unnecessary calculations over empty space are skipped
- In cell-by-cell mode, calculations are performed on every cell, even empty cells.
- In block mode, certain calculation steps are optimized to be performed once for a large range of cells
- In cell-by-cell mode, these calculation steps are repeated for each cell.

In most cases, block computation mode is orders of magnitude more efficient. However, not all MDX calculations can run in block computation mode. The set of optimized MDX functions is documented (http://msdn.microsoft.com/en-us/library/bb934106.aspx). When tuning an MDX calculation, refactoring the calculation to use only these optimized functions is an important step.

In contrast, all DAX functions can run in block computation mode.

Caching Architecture

Caching can occur in Tabular models in several places. Caching in Analysis Services improves the user experience when several similar queries are performed in sequence, such as a summary query and a drilldown. Caching is also beneficial to query performance when multiple users query the same data.

MDX Caching

MDX queries that are run against a Tabular model can take advantage of the rich caching already built into the MDX formula engine. *Calculated measures* using DAX that are embedded in the model definition and that are referenced in MDX queries can be cached by the MDX formula engine.

However, DAX queries (i.e. queries that begin with EVALUATE rather than SELECT) do not have calculations cached.

There are different MDX cache scopes. The most beneficial cache scope is the global cache scope which is shared across users. There are also session scoped caches and query scoped caches which cannot be shared across users.

Certain types of queries cannot benefit from MDX caching. For example, multi-select filters (represented in the MDX query with a subselect) prevent use of the global MDX formula engine cache. However, this limitation was lifted for most subselect scenarios as part of SQL Server 2012 SP1 CU4 (http://support.microsoft.com/kb/2833645/en-us). However, subselect global scope caching is still prevented for arbitrary shapes (http://blog.kejser.org/2006/11/16/arbitrary-shapes-in-as-2005/), transient calculations like NOW(), and queries without a dimension on rows or columns. If the business requirements for the report do not require visual totals and caching is not occurring for your query, consider changing the query to use a set in the WHERE clause instead of a subselect as this enables better caching in some scenarios.

A WITH clause in a query is another construct that prevents the use of the global MDX formula engine cache. The WITH clause not only prevents caching the calculations defined in the WITH clause, it even prevents caching of calculations that are defined in the model itself. If possible, refactor all WITH clause calculations as hidden DAX calculated measures defined inside the model itself.

VertiPaq Caching

Since the formula engine can be very chatty, with the VertiPaq storage engine frequently requesting the exact same VertiPaq query multiple times inside the same MDX/DAX query, a very simple and high performance caching layer was built into the VertiPaq storage engine. It enables identical VertiPaq queries to be retrieved from cache almost instantaneously. This cache is also beneficial when multiple users run the same query.

In the following screenshot of Profiler events, these VertiPaq cache hits can be seen with the VertiPaq SE Query Cache Match event in the green circle. The red circle shows a cache miss.

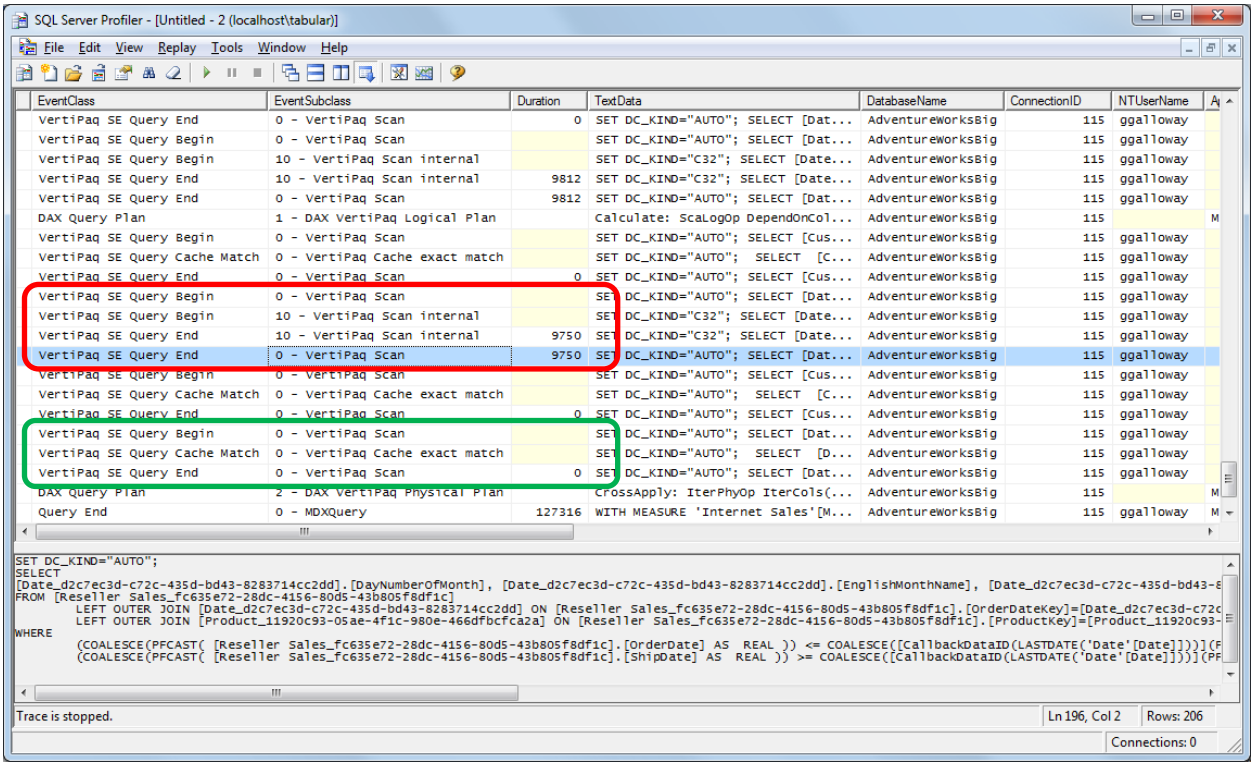


Figure 5. VertiPaq cache hits

The storage engine cache in a Multidimensional instance allows subsequent queries that are similar but not identical to quickly summarize up from the cached data. In contrast, the VertiPaq storage engine cache allows only identical subsequent queries to leverage the cache. This functionality difference is due to the fact that VertiPaq queries are answered from memory, which is orders of magnitudes faster than queries being answered from disk.

Some VertiPaq queries cannot be cached. For example, if a callback to the formula engine occurs in a VertiPaq scan, it prevents caching.

Note, however, that the VertiPaq cache does not use compressed storage; therefore, the cache can consume significant memory when the results of VertiPaq queries are large.

Clearing the Cache

When doing any query performance tests, be sure to record both cold cache and warm cache performance numbers.

1. Open an MDX window in SQL Server Management Studio, and paste the following XMLA.

```
<ClearCache xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">
 <Object>
  <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
 </Object>
</ClearCache>
```

2. Replace the highlighted DatabaseID with your DatabaseID.

Typically the DatabaseID matches the database name, but if not, you can find the DatabaseID by right-clicking on a database in the Object Explorer pane of Management Studio and choosing Properties. See the ID property in the following dialog box:

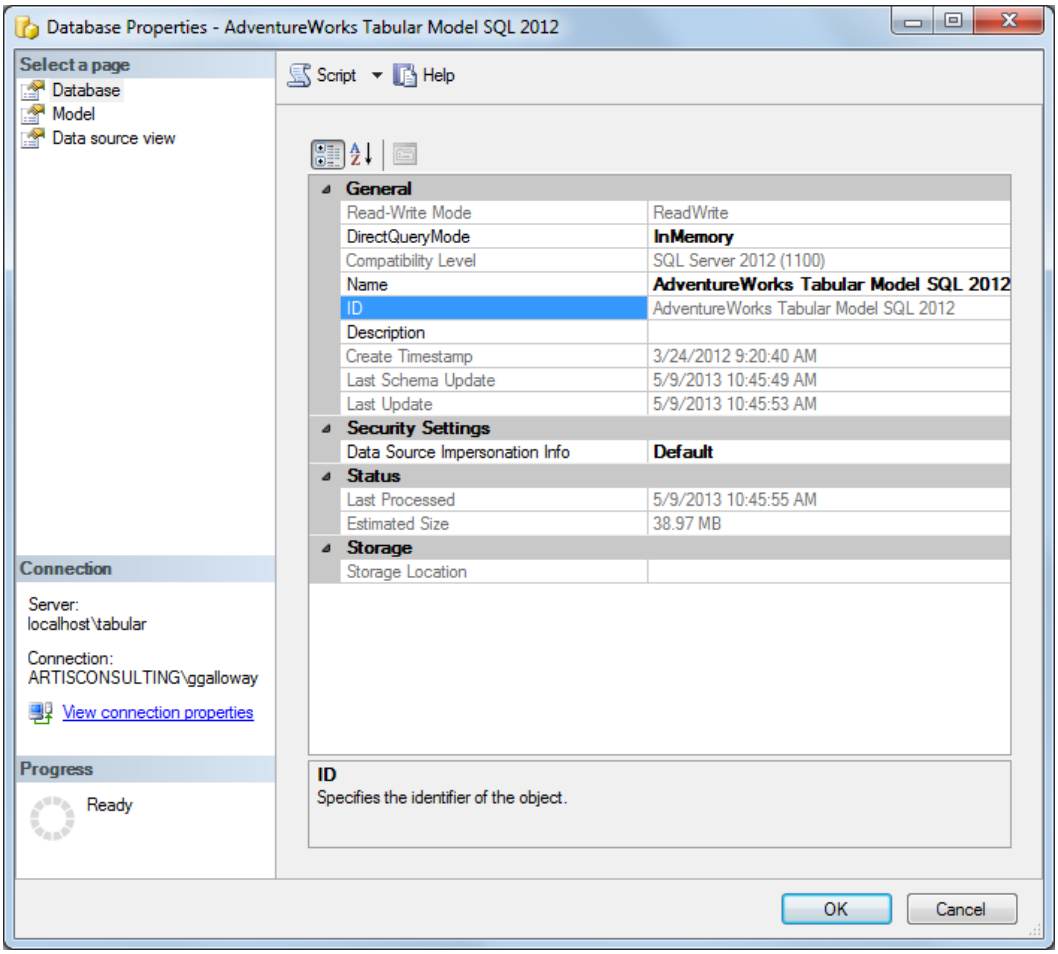


Figure 6. Locating the database ID

3. Run the statement, to clear both the MDX cache and the VertiPaq cache.
4. After the cache has been cleared successfully, run the query you want to test twice.
   - The first execution is cold cache performance.
   - The second execution is warm cache performance.
5. In Profiler, make a note of which VertiPaq queries hit cache, as discussed above.

Warning: Do not clear the cache on a production database as it harms production query performance.

Any processing command run against the database will clear both the MDX and VertiPaq caches for that database.

Cache Warming

If model processing is easily fitting within the refresh window, it is possible to spend that extra time warming the cache. That said, for Tabular models, cache warming may be of limited value. That is because the VertiPaq cache is purposefully made very small, to keep it efficient. VertiPaq cache entries are removed to make room for newer queries to be cached, so warming the VertiPaq cache is generally not worthwhile in typical scenarios.

Warming the MDX formula engine cache may be beneficial. To warm the cache, simply run common and slow performing MDX queries after model processing but before users begin querying the model.

Row Filters in Role-Based Security

Though a full discussion of the role-based security architecture is out-of-scope for this section, the impact of row-level security on query performance is important.

When you create roles for accessing a Tabular model, you can optionally define row filters on one or more tables in the model.

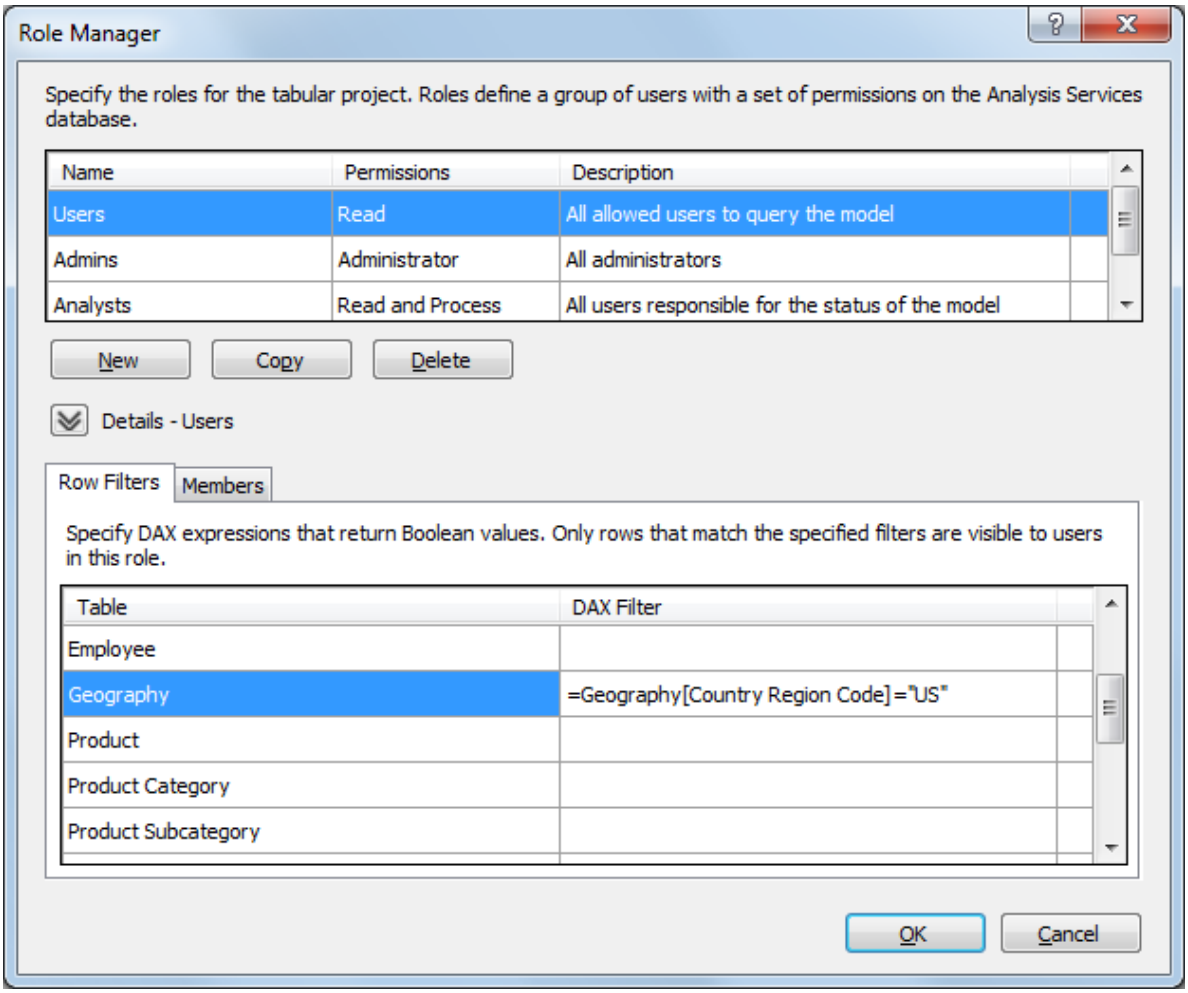


Figure 7. Defining a row-based filter for a role

These row filter expressions are evaluated the first time a user queries that table or any related table. For example, a row filter on the Product Subcategory table will be evaluated and cached the first time any downstream table (such as Product or Internet Sales) is queried.

However, queries on the Product Category table will not be affected by the row-level security on Product Subcategory, because cross-filtering operates in only one direction over relationships.

If User1 and User2 are both in the same set of roles, then a complex row filter expression will be evaluated and cached when User1 queries a related table. When User2 queries that same table, the row filter expression will not be evaluated again, as long as it doesn't reference a function such as USERNAME which differs between users.

Profiler will display any VertiPaq scans necessary to evaluate row filter expressions when they are evaluated. However, you will not see a DAX Query Plan for the row filter expression evaluation itself. If you need to optimize such queries to reduce the expense of the user's first query of the day, see a tip in Limitations of DAX Query Plans regarding capturing a query plan for row filter expressions. However, the row filters, once evaluated, will appear in the subsequent VertiPaq scans for user queries. Both VertiPaq scans and DAX Query Plans are discussed later in this section.

Comparing the Multidimensional and Tabular Storage Engines

It is helpful to compare data storage and retrieval in Multidimensional models versus Tabular models. **Multidimensional** models use row-based storage which is read from disk as needed in queries. Even on wide fact tables, all measures in the fact table are retrieved from storage even if only one measure is needed in a query. Models with hundreds of billions of fact rows are possible with Multidimensional models.

Tabular models use columnar storage which is already loaded into memory when needed by a query. During query execution, only the columns needed in the query are scanned. The VertiPaq engine underlying Tabular models achieves high levels of compression allowing the complete set of model data to be stored in memory, and it achieves blazingly fast data scan rates in memory, providing great query performance. Tests have shown amazing performance:

- **Commodity laptop** hardware can service VertiPaq scans at 5 billion rows per second or more and can store in memory billions of rows.
- **Commodity server** hardware tests have shown VertiPaq scan rates of 20 billion rows per second or more, with the ability to store tens of billions of rows in memory.

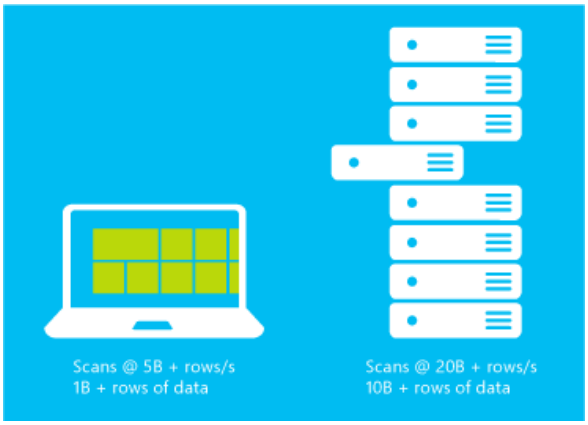


Figure 8. Comparing VertiPaq speed on commodity laptops and servers

To contrast these technologies, try loading a typical two billion row fact table into both a Multidimensional model and a Tabular model. Simple aggregation query performance tests would show results in the following order of magnitude:

Table 2. Comparison of query performance

Server	Scenario	Approximate Query Performance
Multidimensional	If query hits an aggregation	~ seconds
Multidimensional	If query misses aggregations but the fact data is cached in memory in the file system cache	~ minute
Multidimensional	If query misses aggregations and no fact data is cached in memory in the file system cache	~ minutes
Tabular		~ milliseconds

Notice that Multidimensional model query performance on large models is highly dependent on whether the administrator has anticipated query patterns and built aggregations. In contrast, Tabular model query performance requires no aggregations built during processing time in order to achieve great query performance.

Of course not every query achieves such optimal performance, and this whitepaper exists to guide the Analysis Services developer or administrator towards determining the bottleneck for poorly performing queries and applying the appropriate fixes.

Query Troubleshooting Methodology

Optimization involves finding the bottleneck and removing it. This section describes a methodology of troubleshooting queries in order to identify the bottleneck. The next section discusses different tuning techniques that you can apply to the bottleneck.

The Four W's

A recommended methodology for troubleshooting is the Four W's method: Who, What, When, and Where.

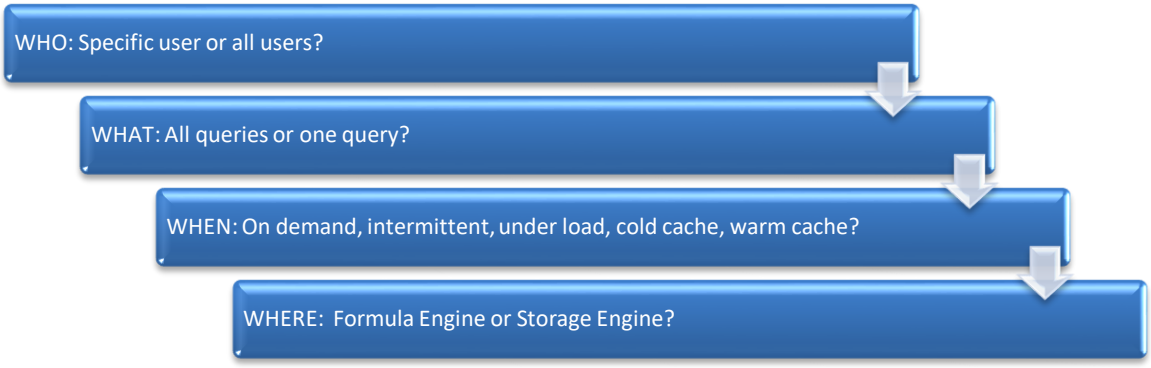


Figure 9. Troubleshooting using the four W's

Who?

First determine whether this query performance issue affects all users or just one user.

For example, you might ask these questions:

- Is this one user opening the report in Excel 2007 while most users are using Excel 2013?
- Does the report perform well until you convert the Excel PivotTable into CUBEVALUE Excel formulas?
- Does the query perform poorly for users in all roles or just users in a particular role with row-level security?
- Is the performance problem only seen from remote users connecting through VPN, or are users connected locally via Ethernet also experiencing the performance issue?

During this phase of troubleshooting, connect to the model as an member of the Server Administrator role in Analysis Services (as opposed to a local administrator on the server or client) and edit the connection string by adding ";EffectiveUserName=DOMAIN\username".
This connection string property allows an administrator to impersonate any user without knowing that user's password.

What?

After determining who is affected by the performance problem, determine which queries are problematic.

- Does every query perform poorly?
- Do queries only perform poorly if a particular DAX calculation is included?

- Do queries perform well if heavily filtered but perform poorly if not filtered well?

As part of this troubleshooting step, once slow queries have been identified, break them down by removing one piece at a time until the query begins to perform well.

- Does the query perform poorly until you manually edit the query to remove all subtotals?
- Does the query perform poorly only until you comment out a certain calculation?

In this way, the cause of the performance issue can quickly be identified.

For example, if you have a measure formula that seems slow, you can measure the duration of portions of the formula to see how they perform.

Original formula:

```
InventoryQty:=SUMX(VALUES(DimStore[StoreKey]),
SUMX(VALUES(DimProduct[ProductKey]), CALCULATE([BaseQty],
LASTNONBLANK(DATESBETWEEN(DimDate[Datekey],BLANK(),
MAX(DimDate[Datekey])),[BaseQty])))
```

Test formula 1:

```
=LASTNONBLANK(DATESBETWEEN(DimDate[Datekey],BLANK(),
MAX(DimDate[Datekey])),[BaseQty])
```

Test formula 2:

```
=  CALCULATE([BaseQty],
LASTNONBLANK(DATESBETWEEN(DimDate[Datekey],BLANK(),
MAX(DimDate[Datekey])),[BaseQty]))
```

Test formula 3:

```
=  SUMX(VALUES(DimProduct[ProductKey]),  CALCULATE([BaseQty],
LASTNONBLANK(DATESBETWEEN(DimDate[Datekey],BLANK(),
MAX(DimDate[Datekey])),[BaseQty])))
```

When?

After slow queries have been identified, perform some tests on a dedicated test server to determine if the performance issues are easy to reproduce on demand or if the problem is intermittent.

- If the problem is reproducible on demand, that makes it easier to troubleshoot.
- If the problem is intermittent, then you need to determine what else is happening when the problem occurs.
- Test slow queries on both a cold and warm cache.
- Test the queries while many other queries are running simultaneously. Tabular mode is not specifically optimized for multiple concurrent users. For example, it doesn't have settings to

prevent one user query from consuming all the resources on the server. Concurrent users will usually experience no problems, but you might wish to monitor this scenario to be sure.
- Test the queries while the model is being processed and after processing completes.
- Test queries running exactly when processing transactions commit to determine if blocking occurs (see Section 4).

Where?

Use SQL Server Profiler to capture trace events, and then tally up the duration of the events to decide whether the storage engine or the formula engine is the bottleneck. See Is the Bottleneck the Storage Engine or the Formula Engine for more details.

In an isolated test environment, observe the CPU and memory usage during the query, and ask these questions:

- Is one core pegged for the duration of the query (sustained 6% CPU usage on a 16 core machine, for example) indicating a formula engine bottleneck since it is single-threaded?
- Or are many processor cores doing work indicating a storage engine bottleneck?
- Is memory usage fairly unchanged during the query?
- Or is there a sudden huge spike in memory usage?
- How close is Analysis Services to the current values for **VertiPaqMemoryLimit** and **LowMemoryLimit**? (See section 5 for an explanation of why this might matter.)

Determining the Source of a Bottleneck

The first step is to determine whether the bottleneck is the Storage Engine or the Formula Engine. To do this, use SQL Server Profiler to capture query trace events.

1. Capture the default columns plus RequestID.
2. Add the **VertiPaq SE Query End** event and the **Query End** event.
3. Sum up the Duration column on the **VertiPaq SE Query End** event where EventSubclass = "0 - VertiPaq Scan". This represents the time spent in the storage engine.
4. Compare that number to the Duration on the **Query End** event.

If the time spent in the storage engine is more than 50% of the duration of the query, the bottleneck is the storage engine. Otherwise, the bottleneck is the formula engine.

To determine where the bottleneck is, set up a T-SQL query to check your traces and then analyze the problem for your query. You can use the following T-SQL query to automate this analysis.

Load the trace file output to a SQL Server table. Then run the query against the table where Profiler saved the trace events.

```
select DatabaseName
,NTUserName
,Query
```

```
,QueryExecutionCount = count(*)
,QueryDurationMS = sum(QueryDurationMS)
,FormulaEngineCPUTimeMS = sum(FormulaEngineCPUTimeMS)
,StorageEngineDurationMS = sum(StorageEngineDurationMS)
,StorageEngineCPUTimeMS = sum(StorageEngineCPUTimeMS)
,Bottleneck = case
               when sum(StorageEngineDurationMS) > sum(QueryDurationMS)/2.0
                then 'Storage Engine' else 'Formula Engine' end
from (
 select DatabaseName = min(DatabaseName)
 ,NTUserName = min(NTUserName)
 ,Query = max(case when EventClass = 10 then cast(TextData as varchar(max)) end)
 ,QueryDurationMS = sum(case when EventClass = 10 then Duration else 0 end)
 ,FormulaEngineCPUTimeMS = sum(case when EventClass = 10 then CPUTime else 0 end)
 ,StorageEngineDurationMS = sum(case when EventClass = 83 and EventSubclass = 0 then Duration else 0 end)
 ,StorageEngineCPUTimeMS = sum(case when EventClass = 83 and EventSubclass = 0 then CPUTime else 0 end)
 from tempdb.dbo.ProfilerTraceTable
 group by RequestID /*group by query*/
 having sum(case when EventClass = 10 and Error is null then 1 else 0 end) > 0
 /*skip non-queries and queries with errors */
) p
group by DatabaseName, NTUserName, Query
order by QueryDurationMS desc
```

That query produces results similar to the following:

	DatabaseName	NTUserName	Query	QueryExecutionCount	QueryDurationMS	FormulaEngineCPUTimeMS	StorageEngineDurationMS	StorageEngineCPUTimeMS	Bottleneck
1	AdventureWorksBig	ggalloway	WITH MEASURE 'Interne...	1	117741	2839	114950	712582	Storage Engine
2	AdventureWorksBig	ggalloway	with measure 'Internet Sal...	1	2607	718	1875	10796	Storage Engine
3	AdventureWorks Tab...	ggalloway	with measure 'Internet Sal...	2	203	140	26	63	Formula Engine

Figure 10. Sample query trace event summary

From these results you can see that the first query is bottlenecked on the storage engine. In the next section, you will learn how to investigate further.

If you're already having performance problems, beware of saving directly to a table – consider saving to a file first then loading to a table, by using a PowerShell script such as the one Boyan Penev discusses (http://www.bp-msbi.com/2012/02/counting-number-of-queries-executed-in-ssas/). Saving directly to a table can be a bottleneck, especially if you save to a SQL instance that is overloaded or remote with possible bandwidth issues.

Researching Storage Engine Bottlenecks

If the bottleneck is the storage engine, then focus on the slowest **VertiPaq SE Query End** events. The text emitted by these events is a pseudo-SQL language internally called *xmSQL*. xmSQL is not a real query language but rather just a textual representation of the VertiPaq scans, used to give visibility into how the formula engine is querying VertiPaq.

The **VertiPaq SE Query Begin/End** events have two subclasses: **0 – VertiPaq Scan** and **10 – VertiPaq Scan internal**. The first contains the formula engine's VertiPaq query. The second is usually nearly identical, though sometimes VertiPaq will choose to optimize the query by rewriting it. It is possible for one VertiPaq Scan to be rewritten into multiple internal scans.

For example, here is a DAX query against the Adventure Works Tabular model.

```
EVALUATE ROW(
 "TotalAmount"
 ,SUMX(
   'Reseller Sales'
   ,'Reseller Sales'[Unit Price] * 'Reseller Sales'[Order Quantity]
  )
)
```

Profiler shows that this DAX query generates one VertiPaq query. The GUIDs have been removed for readability.

```
SET DC_KIND="AUTO";
SELECT
SUM((PFCAST( [Reseller Sales].[UnitPrice] AS  INT ) * PFCAST( [Reseller Sales].[OrderQuantity] AS  INT )))
FROM [Reseller Sales];
```

This clearly shows that the multiplication of Unit Price and Order Quantity has been pushed down into the VertiPaq scan. Even on an expanded 100 million row version of the Reseller Sales table, this VertiPaq query finishes in 241 milliseconds.

(DC_KIND is intended only for Microsoft support engineers to debug VertiPaq query behavior. It is not modifiable. PFCAST is a cast operator that is sometimes injected as shown when the query contains an expression such as SUMX(ResellerSales, [UnitPrice]).)

For a slightly more complex example, here is another DAX query:

```
EVALUATE
ADDCOLUMNS(
 VALUES('Product'[Product Line])
 ,"TotalUnits", 'Reseller Sales'[Reseller Total Units]
)
```

Two VertiPaq scans are generated:

```
SET DC_KIND="AUTO";
SELECT
[Product].[ProductLine],
SUM([Reseller Sales].[OrderQuantity])
FROM [Reseller Sales]
 LEFT OUTER JOIN [Product] ON [Reseller Sales].[ProductKey]=[Product].[ProductKey];

SET DC_KIND="AUTO";
SELECT
[Product].[ProductLine]
FROM [Product];
```

Notice that the first scans the Reseller Sales table and sums up OrderQuantity. It also joins to the Product table (since the Adventure Works model contains a relationship between these tables) in order to group by ProductLine. If an inactive relationship were activated with the USERELATIONSHIP function, a similar join on a different column would be seen. The GROUP BY clause is omitted and assumed.

The second scan returns the distinct ProductLine values from the Product table. After the results of both queries are complete, the formula engine joins the results together in case there are ProductLine values not found in the Reseller Sales table.

The following example adds an IF statement, which cannot be evaluated by the storage engine.

```
EVALUATE
ADDCOLUMNS(
 VALUES('Product'[Product Line])
 ,"LargeOrdersAmount"
 ,CALCULATE(
   SUMX(
    'Reseller Sales'
    ,IF(
      'Reseller Sales'[Order Quantity] > 10
          ,'Reseller Sales'[Unit Price] * 'Reseller Sales'[Order Quantity]
     )
   )
  )
)
```

This query generates the following two VertiPaq scans:

```
SET DC_KIND="AUTO";
SELECT
[Product].[ProductLine],
SUM([CallbackDataID(IF(
 'Reseller Sales'[Order Quantity]] > 10
   ,'Reseller Sales'[Unit Price]] * 'Reseller Sales'[Order Quantity]]
    ))](PFDATAID( [Reseller Sales].[OrderQuantity] ), PFDATAID( [Reseller Sales].[UnitPrice] )))
FROM [Reseller Sales]
 LEFT OUTER JOIN [Product] ON [Reseller Sales].[ProductKey]=[Product].[ProductKey];

SET DC_KIND="AUTO";
SELECT
[Product].[ProductLine]
FROM [Product];
```

Note that the IF was not able to be evaluated by the storage engine, so the VertiPaq storage engine calls back to the formula engine to perform this calculation. This callback is indicated with **CallbackDataID**. Even on a 100 million row version of the Reseller Sales table, this query completes in 1.5 seconds.

The optimal filter in a VertiPaq query is written with a WHERE clause that looks like any of the following:

```
[Customer].[FirstName] = 'Roger'

([Customer].[LastName], [Product].[ProductLine]) IN {('Anand', 'M'), ('Murphy', 'T')}

[Customer].[CustomerKey] IN (15054, 21297) VAND [Geography].[StateProvinceCode] IN ('OR', 'HE')

COALESCE((PFDATAID( [Customer].[LastName] ) = 81))
```

(PFDATAID above just means that the filter is being performed on the DataID instead of on the value. For more information on DataIDs, see section 4.)

A predicate with a CallbackDataID is not as efficient as the simple predicates in the above examples. For example, a callback prevents VertiPaq caching.

Although less efficient than pure VertiPaq scans, callbacks may still be faster than evaluating the calculations inside the formula engine after VertiPaq results are returned.

- First, since callbacks occur from the multi-threaded VertiPaq storage engine, they can happen in parallel even within the same query. (Other than callbacks, operations in the formula engine are single-threaded per query.)
- Second, callbacks do avoid spooling of temporary results, which reduces memory usage.
- Third, the VertiPaq engine is able to efficiently calculate over large blocks of continuous rows with identical values for the columns referenced in calculations.

If it is possible to refactor a calculation to generate a VertiPaq scan with a simple predicate and without simply moving that same amount of work to the single-threaded formula engine, consider avoiding callbacks in order to achieve optimal query performance and caching.

Another common bottleneck in storage engine heavy queries is **materialization**. Queries may require cross-joining several columns and temporarily storing that new table in memory, to be passed to the formula engine for further calculation. This temporary storage is called a **spool** and it is not compressed in memory.

Depending on the size of the spool, performance could suffer and memory usage could rapidly increase. It is possible for a query against a very small 10MB model to consume many GB of memory due to materialization. If too much memory is being consumed, consider rewriting the query to avoid materialization.

The other type of bottleneck to look for when queries appear to be bottlenecked in the storage engine would be too many VertiPaq queries. Individual VertiPaq queries might be very fast, say, 50 milliseconds, but two hundred such VertiPaq queries would consume 10 seconds. Since the formula engine is single-threaded, these two hundred VertiPaq queries are run in serial. Often, this type of query pattern is caused by the formula engine requesting hundreds of datasets and then joining them together in the formula engine. If the DAX calculations can be rewritten to push more logic into VertiPaq scans, the number of VertiPaq queries may be reduced and overall query performance may improve.

If the VertiPaq queries do not suggest to you any DAX optimizations you could make, then you may need to study the VertiPaq queries in the context of the overall DAX Query Plan. The DAX Query Plan is explained in the next section.

Looking Up Column or Table Name for ID in VertiPaq Scans

Unfortunately, xmSQL currently displays unfriendly internal ID names for tables and columns. This makes it difficult to read because of the GUIDs in the table names, and it can be confusing if the internal ID for a table or column does not match its name.

If you are not clear which table or column is being referenced, you can look up the name as follows:

1. In SQL Server Data Tools, right click the .bim file
2. Select **View Code**.
3. Search for "<ID>The ID to Lookup</ID>"
4. Make a note of the corresponding Name next to the ID in the XML, as shown in the following screenshot:

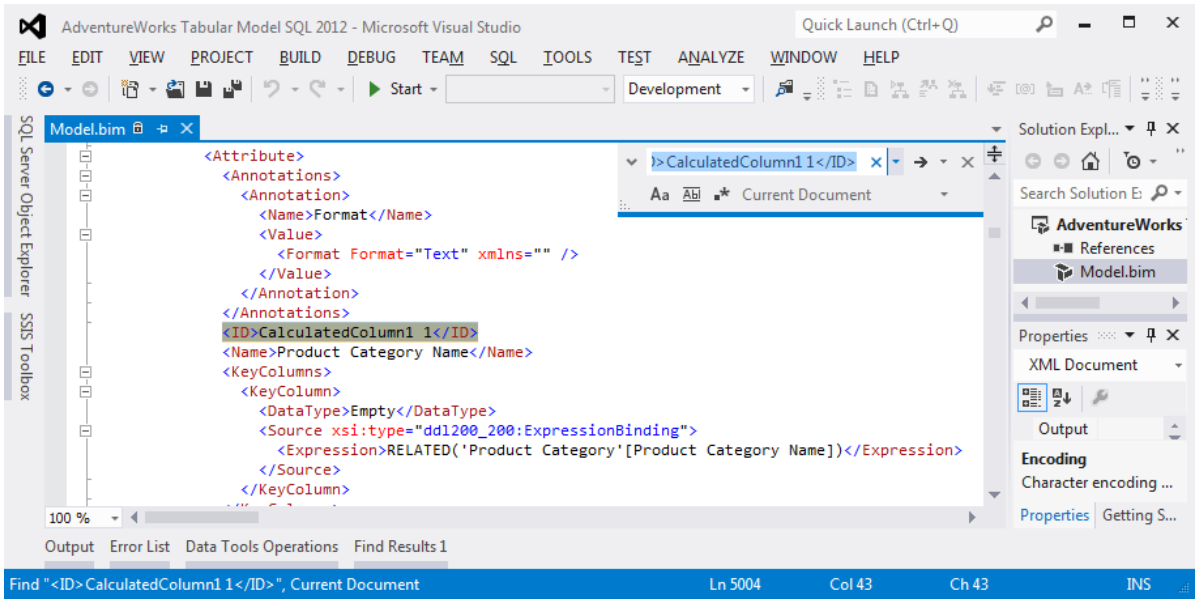


Figure 11. How to find the column ID

The DAX Query Plan discussed below does display friendly names for columns but not for tables.

Researching Formula Engine Bottlenecks

If the bottleneck is the formula engine, then study the DAX Query Plan further. The next section describes the DAX Query Plan.

In some scenarios, MDX calculations are the bottleneck instead of DAX calculations. You should investigate the MDX calculations if the following conditions are all met:

- The slow query is an MDX query.
- There are complex or expensive MDX calculations in the query.
- The DAX Query Plan(s) don't provide any clues on how to tune the query.

(Note: SQL Server 2012 Analysis Services also introduced a new Profiler event that provides some insight into the internals of MDX calculations. These two new events are called **Calculation Evaluation** and **Calculation Evaluation Detailed Information**. An explanation of these events is out-of-scope for this whitepaper since the calculations in a Tabular model are DAX calculations that can be investigated with the DAX Query Plan.)

Generally, Calculation Evaluation trace events are very expensive to capture, so it is recommended you avoid capturing them in a production environment.

Understanding and Using the DAX Query Plan

When trying to understand how Analysis Services interprets and evaluates a DAX calculation, the Profiler trace event **DAX Query Plan** is the place to look. This section provides a brief introduction to get you started troubleshooting DAX calculations.

This Profiler event has two subclasses:

- **1 - DAX VertiPaq Logical Plan**
- **2 - DAX VertiPaq Physical Plan**

It also has two **DirectQuery** subclasses, but they are not discussed in this whitepaper.

A DAX query will produce exactly one Logical Plan and one Physical Plan. An MDX query will produce multiple Logical/Physical Plan pairs in many scenarios, such as requesting a grand total or requesting multiple DAX measures.

Logical Plan

The Logical Plan usually closely resembles the actual DAX calculations. For example, comparing the following DAX query to its Logical Plan demonstrates the similarities.

DAX Query	DAX VertiPaq Logical Plan
EVALUATE CALCULATETABLE(ADDCOLUMNS(VALUES('Product'[Product Line]) ,"TotalUnits" , CALCULATE(SUM('Reseller Sales'[Order Quantity]))) ,'Product'[Days To Manufacture] >= 2)	CalculateTable: RelLogOp DependOnCols()() 1-2 RequiredCols(1, 2)('P AddColumns: RelLogOp DependOnCols()() 1-2 RequiredCols(1, 2)('P Scan_Vertipaq: RelLogOp DependOnCols()() 1-1 RequiredCols(1 Sum_Vertipaq: ScaLogOp DependOnCols(1)('Product'[Product Li Scan_Vertipaq: RelLogOp DependOnCols(1)('Product'[Produ 'Reseller Sales'[Order Quantity]: ScaLogOp DependOnCols Filter_Vertipaq: RelLogOp DependOnCols()() 0-0 RequiredCols(0)(Scan_Vertipaq: RelLogOp DependOnCols()() 0-0 RequiredCols(0 GreaterThanOrEqualTo: ScaLogOp DependOnCols(0)('Product'[Da 'Product'[Days To Manufacture]: ScaLogOp DependOnCols(0 Constant: ScaLogOp DependOnCols()() BigInt DominantValu

(Note: The plan is actually much wider than shown here, but the screenshot has been cropped for easier comparison.)

Note that the **CalculateTable**, **AddColumns**, and **GreaterThanOrEqualTo** functions map exactly. However, sometimes the Logical Plan doesn't closely resemble the DAX calculations because the

formula engine optimizes the Logical Plan to eliminate some extra joins by transferring constraints from one operator to another. In general, however, the Logical Plan contains these elements:

- The first word on the line is the operator.
- The next is the operator type. For the Logical Plan, the operator types are **ScaLogOp** which outputs a scalar value (i.e. number, date, string) or **RelLogOp** which outputs a table (for example, the results of a crossjoin).

If almost all the operators in the Logical Plan end in _Vertipaq (for example, **Scan_Vertipaq**, **Sum_Vertipaq**, **Filter_Vertipaq**) then your query will likely perform optimally. Pushing all operations possible into VertiPaq is the goal.

If the entire query can be reduced to a series of group by, filter, join, and aggregation operations, then you should be able to write calculations that are pushed down to the VertiPaq scan, since those are the operations VertiPaq supports. Other operations such as sorting (for example, TOPN) are not supported in VertiPaq so must be evaluated in the formula engine.

Sparse scalar operators and block computation mode are often the key ingredient in queries which perform well.

For a **ScaLogOp** operator, if the value of the **DominantValue** property is **NONE**, then Analysis Services has detected that this data is *dense*. If the value of the **DominantValue** property is anything other than **NONE**, Analysis Services has detected that the data is *sparse* and may be able to use block computation mode to efficiently calculate this particular scalar value. Specifically, Analysis Services may be able to use an iterator in the physical plan which is able to skip a large number of rows that would return the dominant value.

Physical Plan

The Physical Plan has a similar indented format that represents the call stack.

- The first word on a line is the operator.
- The next word is the operator type.

For the Physical Plan, the operator types are as follows:

- **LookupPhyOp**. Returns a scalar value given the row context.
- **IterPhyOp.** Iterates one row at a time through a table given the row context.
- **SpoolPhyOp.** Receives the results of a VertiPaq query or intermediate calculation result and materializes it in memory.

```
AddColumns: IterPhyOp IterCols(1, 2)('Product'[Product Line], ''[TotalUnits])
 Spool_Iterator<Spool>: IterPhyOp IterCols(1)('Product'[Product Line]) #Records=3 #KeyCols=238 #ValueCols=0
  AggregationSpool<Cache>: SpoolPhyOp #Records=3
   VertipaqResult: IterPhyOp #FieldCols=1 #ValueCols=0
 Spool: LookupPhyOp LookupCols(1)('Product'[Product Line]) BigInt #Records=3 #KeyCols=238 #ValueCols=1 DominantValue=BLANK
  AggregationSpool<Cache>: SpoolPhyOp #Records=3
   VertipaqResult: IterPhyOp #FieldCols=1 #ValueCols=1
```

Pay close attention to the **VertipaqResult** operator (highlighted in green) as it corresponds to one of the **VertiPaq SE Query Begin**/**End** events. Unfortunately, it is difficult to tell which VertiPaq query it corresponds to, other than by the number of fields and value columns (**#FieldCols** and **#ValueCols**, respectively) the query returns.

DAX functions such as the IF function support *short-circuit evaluation*. Analysis Services calls this "strict" mode compared to "eager" mode. DAX functions always use strict evaluation during the physical evaluation phase. The short-circuit evaluation will be evident in the Physical Plan since only the branches executed are shown. However, the Logical Plan will show both branches in scenarios when the IF condition depends on a VertiPaq scan which doesn't execute until after the Logical Plan is generated.

One of the most important details to focus on is the **#Records** property (highlighted in yellow in the example above) that displays towards the right side of some lines. Often, performance and memory usage of a query will be most tied to the operators that involve large record counts.

For example, in the simple example just cited, the second line of the physical plan uses a **Spool_Iterator** operator which will iterate over 3 records in a temporary in-memory structure called a spool.

One common place to watch for materialization or slowness in the Physical Plan is to look for an Apply operator with multiple VertiPaq queries underneath returning a large number of records (such as 100,000 or more). Run the following example query against Adventure Works.

```
EVALUATE
ROW(
 "cnt"
 ,COUNTROWS(
   CROSSJOIN(
    VALUES('Reseller Sales'[Discount Amount])
    ,VALUES('Reseller Sales'[Sales Amount])
   )
  )
)
```

This query returns the following Physical Plan. Some higher level operators were omitted for brevity. Notice the **CrossApply** operator with multiple **VertipaqResult** operators beneath. Note the record counts and realize that if these record counts were larger, this query could require a long time to execute as it iterates row by row through all the combinations. The row by row iteration requires very little memory but a significant amount of time. However, in other queries, the output of a large CrossApply is spooled and can consume a large amount of memory.

```
ApplyRemap: IterPhyOp IterCols(0, 1)('Reseller Sales'[Discount Amount], 'Reseller Sales'[Sales Amount])
 CrossApply: IterPhyOp IterCols(0, 1)('Reseller Sales'[Discount Amount], 'Reseller Sales'[Sales Amount])
  Spool_Iterator<Spool>: IterPhyOp IterCols(0)('Reseller Sales'[Discount Amount]) #Records=514 #KeyCols=238 #ValueCols=0
   AggregationSpool<Cache>: SpoolPhyOp #Records=514
    VertipaqResult: IterPhyOp #FieldCols=1 #ValueCols=0
  Spool_Iterator<Spool>: IterPhyOp IterCols(1)('Reseller Sales'[Sales Amount]) #Records=1423 #KeyCols=238 #ValueCols=0
   AggregationSpool<Cache>: SpoolPhyOp #Records=1423
    VertipaqResult: IterPhyOp #FieldCols=1 #ValueCols=0
```

DAX Query Plan Events

When a query is executed:

1. The DAX expressions are parsed.
2. The Logical Plan is built.
3. The Logical Plan is simplified by rewriting some functions.

   In order to simplify the Logical Plan, some VertiPaq queries and parts of the expressions may need to be executed. In SQL Server, the query optimizer uses statistics to weigh query plan decisions. In Tabular models, the data is already compressed and stored in memory in a very efficient format, so Analysis Services can query the data in order to make some query plan decisions.

   After the Logical Plan has been simplified, the trace event fires. Depending on the cost of the expressions and VertiPaq queries executed when simplifying the plan, there may be a significant delay before the event fires, though the delay is typically small.

4. The rest of the expressions and VertiPaq queries are executed.
5. Once all VertiPaq queries are complete the Physical Plan trace event fires.

   This Physical Plan trace event represents the actual execution plan.

The following diagram visualizes the sequence of events during query execution.

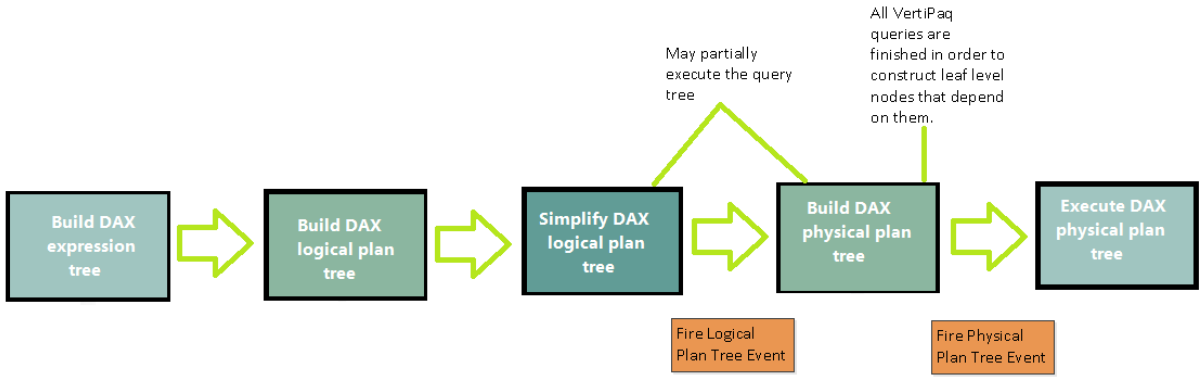


Figure 12. Inside the DAX query plan

For more information on DAX Query Plans, we highly recommend that you read this series of blog posts by Jeffrey Wang, Principal Development Lead on the Analysis Services team:

Post 1 (http://mdxdax.blogspot.com/2011/12/dax-query-plan-part-1-introduction.html)
Post 2 (http://mdxdax.blogspot.com/2012/01/dax-query-plan-part-2-operator.html)
Post 3 (http://mdxdax.blogspot.com/2012/03/dax-query-plan-part-3-vertipaq.html)

Also, see the following white paper on DAX Query Plans from Alberto Ferrari and related tools:

http://www.sqlbi.com/topics/query-plans/

DAX Query Plans for Calculated Columns and Role-Based Security Expressions

Based on the guidance from Section 4, you might have determined that the bottleneck during ProcessRecalc is a particular calculated column. Unfortunately, there is no way to see the DAX Query Plan generated for a calculated column during ProcessRecalc. However, you can construct the following DAX query which will provide you with a DAX Query Plan:

```
EVALUATE ADDCOLUMNS(YourTable, "x", <your calculated column expression here>)
```

The same principle applies to row filters in role-based security. No DAX Query Plan is provided for the row filter expression in each role. If this role-based security causes the first query of a table related to the row filter to be slow, then consider testing your row filter expression in the DAX query above so you can see a DAX Query Plan and optimize it.

Limitations in DAX Query Plans

DAX Query Plans provide great insight into how queries are evaluated. However, there are a few known limitations. First, when VertiPaq performs a callback to the formula engine, the query plan for that callback is not seen. In some cases, the callback itself requires further VertiPaq scans not visible in the DAX Query Plan.

Second, query plans do not display when slices are transferred. For example, the following query uses a many-to-many calculation pattern which filters Internet Sales by the Reseller table through the Reseller Sales bridge table.

```
DEFINE
MEASURE 'Internet Sales'[M2M] =
 CALCULATE(
  'Internet Sales'[Internet Total Units]
  ,SUMMARIZE(
    'Reseller Sales'
   ,'Product'[ProductKey]
  )
 )
EVALUATE
ADDCOLUMNS(
 VALUES(Reseller[Reseller Name])
 ,"m2m", 'Internet Sales'[M2M]
)
```

In this example, first there is a VertiPaq scan of the Reseller Sales table to determine which ProductKey and Reseller Name combinations exist. Then it transfers the list of ProductKey values into the predicate on the VertiPaq scan of the Internet Sales table.

```
SELECT
[Product].[ProductKey],
SUM([Internet Sales].[OrderQuantity])
FROM [Internet Sales]
 LEFT OUTER JOIN [Product] ON [Internet Sales].[ProductKey]=[Product].[ProductKey]
WHERE
 [Product].[ProductKey] IN (531, 380, 494, 323, 457, 571, 286...[334 total values, not all displayed]);
```

This process of transferring slices is not apparent from the DAX Query Plan itself. However, the time it takes to spool the list of ProductKey values will be captured in the **VertiPaq SE Query End** trace event's **Duration** column. Transferring slices can be expensive if you must materialize millions of ProductKey values in memory, however the formula engine performs this optimization to reduce the number of joins.

DAX Query Plans on Production Servers

You should avoid requesting DAX Query Plans on production servers.

SQL Server 2012 SP1 CU3 fixed a bug (described in KB2821686 http://support.microsoft.com/kb/2821686) that was seen mainly during scheduled data refresh in PowerPivot for SharePoint. The problem was that, as a result of a bug, the Analysis Services session trace that was launched to monitor processing during a scheduled data refresh incorrectly included the DAX Query Plan event. Thus, if a user ran a report using a complex DAX query, the time spent generating the DAX query plan could cause the server to consume 100% of available processor resources and become unresponsive.

This bug also affected Tabular servers, whenever a Profiler trace or session trace requested the DAX Query Plan event. Although the bug has been fixed, to get the best performance, always apply the latest service packs and cumulative updates.

DAX query plans can be very expensive to generate on complex queries. Avoid requesting the DAX Query Plan trace event on your production server.

Query Tuning Techniques

In SQL Server, tuning a query may involve adding indexes or hints. In Multidimensional models, tuning a query may involve building aggregations. In Tabular models, there are no indexes, hints, or aggregations. To tune a query against a Tabular model, you have two choices:

- Change the DAX calculations.
- Change the data model.

Changing DAX Calculations

One way to tune a query against a Tabular model is to refactor the DAX calculations to perform more optimally.

There are many ways that you can optimize DAX. For example, moving a filter to a different part of the calculation (as described below in Using CALCULATE Instead of FILTER) may improve performance. Often the best optimization available is refactoring the calculation so that it can be pushed down into a VertiPaq scan.

The following section provides some common examples of optimizing DAX calculations, and some best practices. The section is followed by a walkthrough that demonstrates how you might investigate a DAX query plan.

Make the most of DAX:

- Prefer CALCULATE() over FILTER()
- Move FILTERS toward the outside in nested calculations
- Avoid functions that result in storage engine callbacks or cell-by-cell computation.
- Use SUMMARIZE only for grouping; use ADDCOLUMN to add new values.
- Use ISEMPTY instead of ISBLANK where appropriate.

WARNING: In order to know which calculations to change and how to change them, you must understand how the calculations are evaluated under the covers. If you don't know how to read a DAX Query Plan, we recommend that you review the preceding section, which discusses DAX query plans in detail.

Convert calculations into calculated columns

One way to make a calculation run faster during a query is to move the calculation into the processing phase. That is, if a certain calculation is too slow when run at query time and if processing is fitting well within the processing window, consider changing expensive parts of the calculated measure to a calculated column which will be evaluated at processing time and stored like a regular column.

By converting query-time calculations into calculated columns, you are leveraging a common optimization technique— trading a scarce resource for a plentiful resource. In this case, query time is scarce, but processing time and memory usage are plentiful.

Beware, however: calculated columns are evaluated without respect to row-based security. For example, consider a calculated column called ManagerSalary on the Employee table. This column looks at the manager of the employee on the current row and saves the manager's salary as a calculated column. Later, if row-based security is built to ensure this employee can only see his or her own row, this employee will be able to see his or her manager's salary since this calculated column has already been evaluated at processing time with no respect for row-based security.

Avoid error handling functions

In the DAX language that shipped with the first version of PowerPivot, certain functions like SEARCH and FIND would return an error if they did not find the string they were looking for. Thus, the ISERROR or

IFERROR functions were required to catch the error and handle it. However, in the version of the DAX language that is included with SQL Server 2012 Analysis Services, both the SEARCH and FIND functions have a new fourth parameter that specifies the value to return if the string is not found. Therefore, if you have SQL Server 2012, never use ISERROR and IFERROR. Catching an error severely impacts query performance because it causes the rest of the query to run in cell-by-cell mode. In addition, there are some scenarios where ISERROR or IFERROR will not actually catch every error. Avoid these error handling functions.

The DIVIDE function is a good example of one way of avoiding divide by zero errors. It may also perform better than the equivalent IF function in some cases, as described in this blog by Rob Collie: (http://www.powerpivotpro.com/2013/07/divide-and-conquer/).

Detect filters using optimized functions

When possible, instead of detecting whether you are filtered to one row by using an expression like COUNTROWS(VALUES(TableName[ColumnName]))=1, use the functions HASONEVALUE or HASONEFILTER. These functions are optimized for this task, whereas COUNTROWS necessarily performs several scans.

Decide when to use SUMMARIZE versus ADDCOLUMNS

Although the SUMMARIZE function is convenient, in many cases you can get the same results more efficiently by using ADDCOLUMNS. For example the following formulas are equivalent in terms of results:

```
SUMMARIZE(VALUES(Product), Product[Color], "Total Amount", 'Internet Sales'[Internet Total Sales])
```

```
ADDCOLUMNS(VALUES(Product[Color]), "Total Amount", 'Internet Sales'[Internet Total Sales])
```

In this simple example, the Physical Plan is identical. However, in more complex examples, the ADDCOLUMNS approach produces a more efficient physical plan.

The rule of thumb is that SUMMARIZE is fine when you are just using it to do a group by operation. But if you use SUMMARIZE to add scalar values (in the example above, the `"Total Amount", 'Internet Sales'[Internet Total Sales]` parameters) then prefer ADDCOLUMNS instead.

The first exception to this rule is when the table expression being summarized is a complex, arbitrary shape which would require putting the filter in multiple places in the query. The second exception to this rule occurs when you use the ROLLUP function inside SUMMARIZE. In these two exceptions, SUMMARIZE is the best choice. For more information, see this article by Marco Russo: Best Practices using SUMMARIZE and ADDCOLUMNS (http://www.sqlbi.com/articles/best-practices-using-summarize-and-addcolumns/).

Use CALCULATE instead of FILTER

DAX beginners will naturally lean towards the simple FILTER function and avoid the powerful but complex CALCULATE and CALCULATETABLE functions. This is a mistake. The CALCULATE and

CALCULATETABLE functions are more efficient because they set the filter context for all operations inside the parentheses, and should be used instead of FILTER where possible.

For example, the following query takes 22 seconds to run against Adventure Works. Watching processor usage in Task Manager reveals the typical behavior of a query bottlenecked in the formula engine, with just one processor working for the duration of the query.

```
EVALUATE
ADDCOLUMNS(
 'Date',
 "HistoryToDateTotal",
 SUMX(
  FILTER('Reseller Sales', [Order Date] <= [Date]),
  [Sales Amount]
 )
)
```

Internally, the following VertiPaq query is executed, which returns every row in the Reseller Sales table. Notice that it includes the RowNumber internal column, which ensures that it returns every row.

```
SELECT
[Reseller Sales].[RowNumber], [Reseller Sales].[SalesAmount], [Reseller Sales].[OrderDate]
FROM [Reseller Sales];
```

Studying the Physical Plan reveals #Records=60855, which is the record count for Reseller Sales. On a version of the Adventure Works model with a 100 million row Reseller Sales table, this query would consume many GB of memory, because 100 million rows are cached in memory uncompressed.

However, you can rewrite the query to use CALCULATE instead of FILTER, as shown here. The use of CALCULATE allows the hard work to be pushed down into a VertiPaq scan.

```
EVALUATE
ADDCOLUMNS(
 'Date',
 "HistoryToDateTotal",
 CALCULATE(
  SUMX('Reseller Sales', [Sales Amount]),
  'Reseller Sales'[Order Date] <= EARLIER([Date]),
  ALL('Date')
 )
)
```

As a result, this query executes in under one second. Even on a 100 million row Reseller Sales table, the query executes in two seconds without consuming much memory.

Internally, the rewritten query generates the following VertiPaq scan, which returns the total sales amount per day. The formula engine then uses only these values for further history-to-date summation:

```
SELECT
[Reseller Sales].[OrderDate],
SUM([Reseller Sales].[SalesAmount])
FROM [Reseller Sales]
WHERE
 [Reseller Sales].[OrderDate] IN (39264.000000, 38687.000000...[40 total values, not all displayed]);
```

Decide whether to use ISBLANK versus ISEMPTY

As part of SQL Server 2012 SP1 CU4 (http://support.microsoft.com/kb/2833645/en-us), a new ISEMPTY function was introduced to the DAX language. In some scenarios, using ISEMPTY will perform better than ISBLANK. The reason ISEMPTY is faster is that whereas ISBLANK will evaluate the measure, ISEMPTY will just look for the presence of a row.

For example, the following formula returns a list of cities having a value for Internet Total Freight:

```
FILTER(
 VALUES(Geography[City]),
 NOT(ISBLANK('Internet Sales'[Internet Total Freight]))
)
```

However, the formula could be rewritten as follows:

```
FILTER(
 VALUES(Geography[City]),
 CALCULATE(NOT(ISEMPTY('Internet Sales')))
)
```

Instead of evaluating the Internet Total Freight measure (which could have been an expensive DAX expression) the second formula just finds cities with Internet Sales rows.

A DAX Calculation Tuning Walkthrough

On a copy of the Adventure Works model with the Reseller Sales table expanded to 100 million rows, a certain calculation was performing poorly. The calculation looks at each date period and customer, and then gets the number of Internet sales units for products which also had Reseller sales in progress on the last day of that date range.

The MDX query with DAX calculation in the WITH clause (for ease of troubleshooting) reads as follows:

```
WITH MEASURE 'Internet Sales'[M2M] =
 CALCULATE(
  'Internet Sales'[Internet Total Units]
  ,SUMMARIZE(
    FILTER(
     'Reseller Sales'
     ,'Reseller Sales'[Order Date]<=LASTDATE('Date'[Date])
      && 'Reseller Sales'[Ship Date]>=LASTDATE('Date'[Date])
    )
    ,'Product'[Product Id]
   )
  )
SELECT [Measures].[M2M] on 0,
NON EMPTY [Date].[Calendar].Members
* [Customer].[First Name].[First Name].Members on 1
from [Model]
```

On a cold cache, the query took 130 seconds. The query was run on a cold cache by using the ClearCache XMLA and separating it from the query with a GO keyword, as shown in the following screenshot:

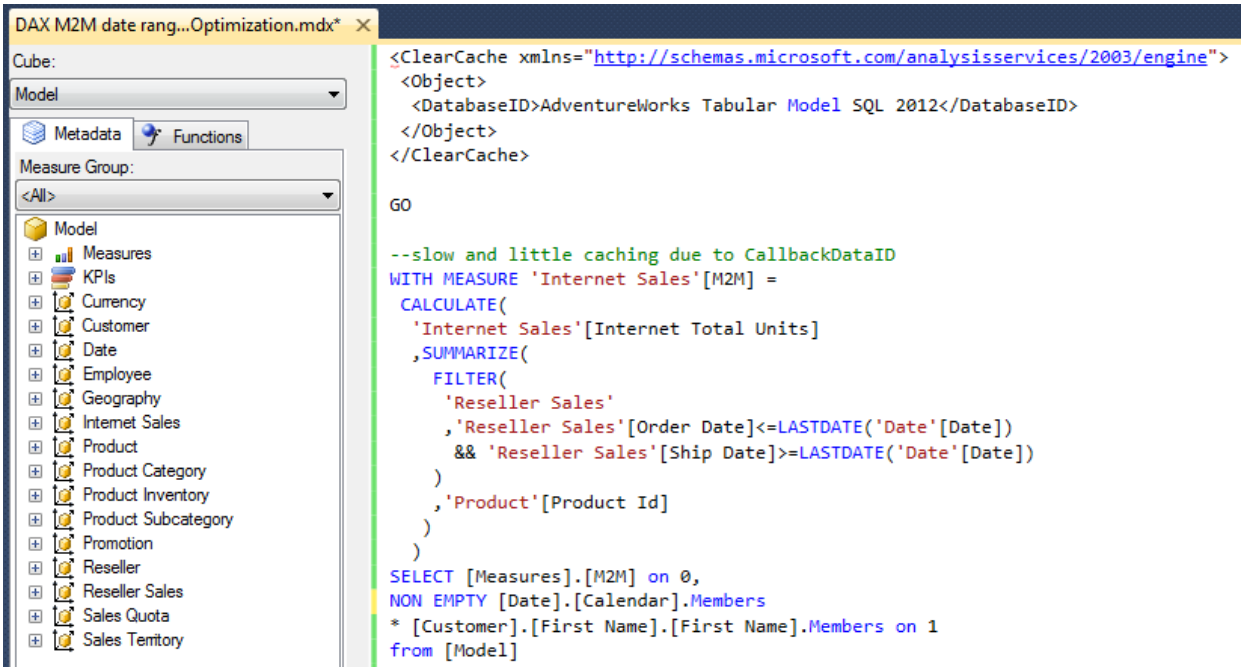


Figure 13. Sample query

Capturing the **VertiPaq SE Query End** Profiler event where EventSubclass = "0 - VertiPaq Scan" and summing the Duration column up revealed that 127 seconds were spent in the storage engine. Given that total duration (from the **Query End** event) was 130 seconds, the bottleneck is clearly in the storage engine.

Next, we looked at Task Manager, and found that almost all 8 processor cores were active for the entirety of the query. When you are testing in an isolated, single-user environment, this CPU usage pattern indicates that the multi-threaded storage engine is in use.

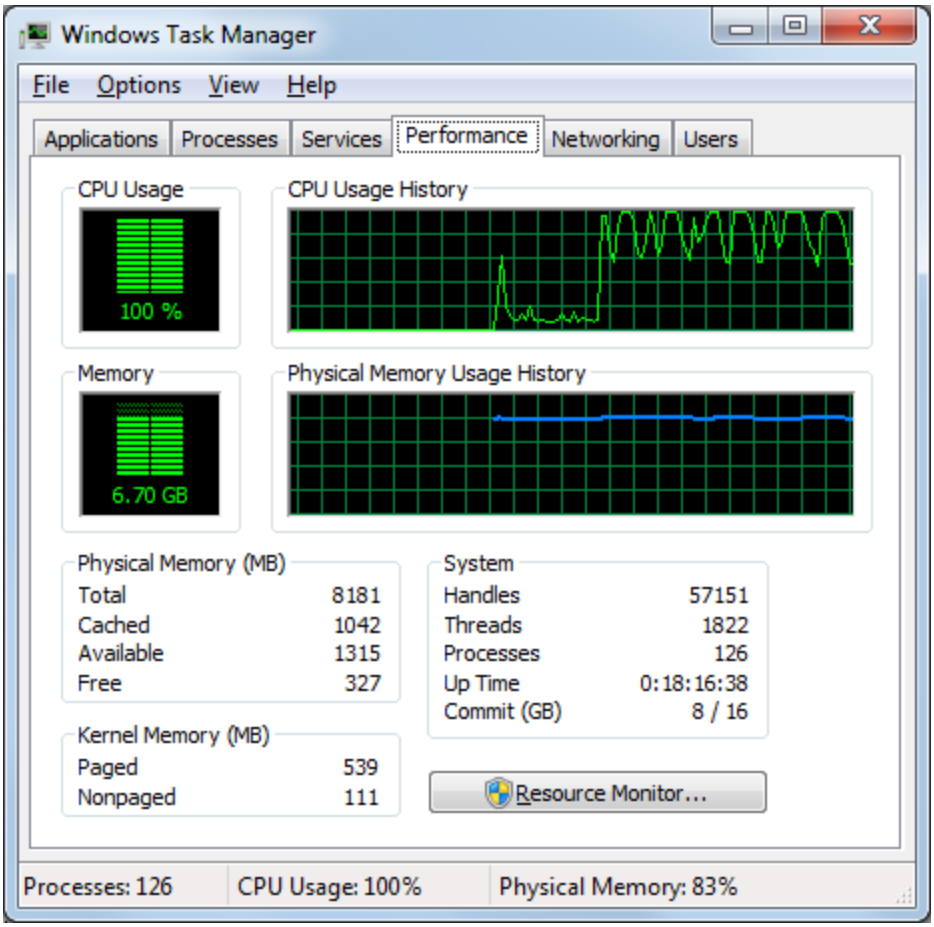


Figure 14. CPU usage on an 8 core machine

Next, we ran the query again without clearing the cache, to test the query on a warm cache. This time the query took 129 seconds. This tells us that the VertiPaq storage engine cache is not being used effectively to improve warm cache performance.

However, if the 'Internet Sales'[M2M] calculated measure definition is moved into the model (rather than inside the WITH clause of the MDX query), the warm cache performance of this query drops to 3 seconds, because the MDX formula engine cache is being utilized. This is a nice optimization, but it is possible to improve the cold cache performance of this calculation.

To find the problem in the warm cache query, we studied the Profiler trace, and found that some VertiPaq scans hit the cache and were fast (see the green circle), but some slow scans do not hit the cache (see the red circle).

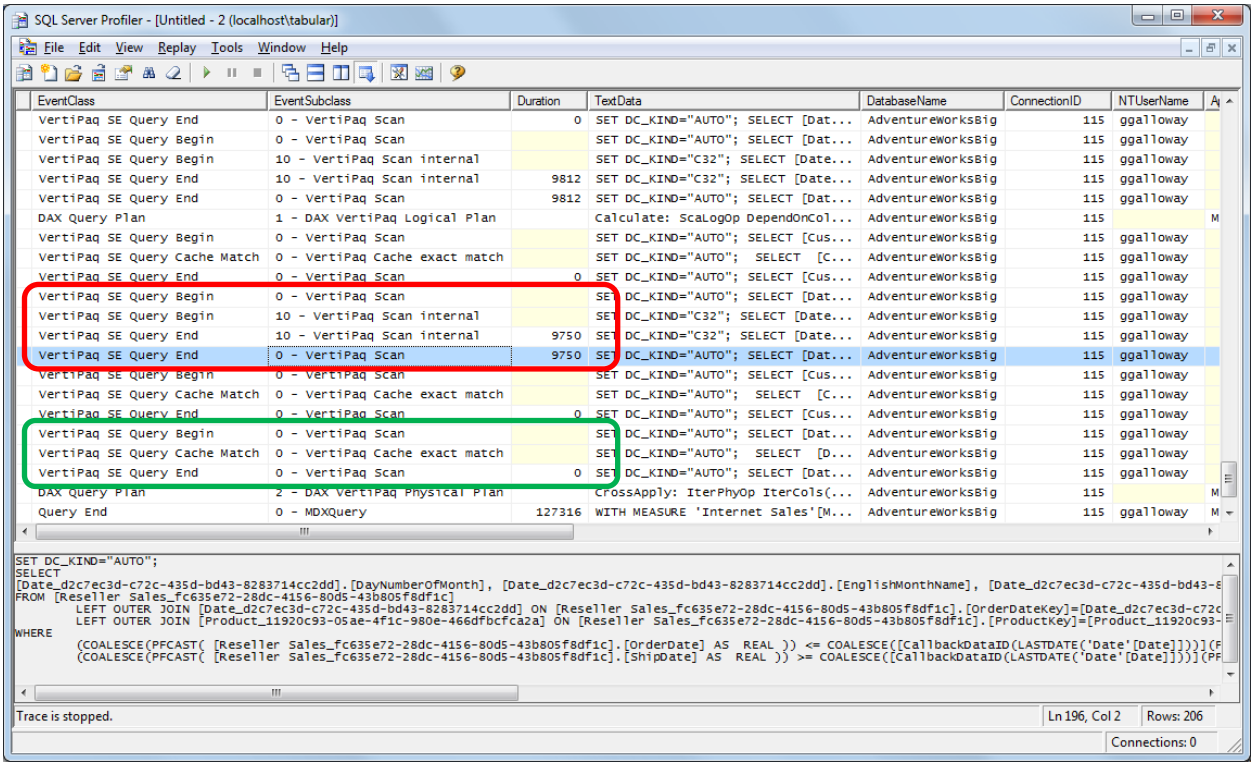


Figure 15. VertiPaq cache hits (green) are faster than non hits (red)

You can get some insight into how data is being retrieved from the VertiPaq engine during the scan by looking at the TextData for the VertiPaq Scan, which was not cached) This pseudo-SQL syntax is very useful to the DAX developer, though it has some elements that are unfamiliar, such as PFCAST, which is a cast operator that is sometimes injected in expressions.

Here we've removed the GUIDs to make it more readable. :

```
SET DC_KIND="AUTO";
SELECT
[Date].[DayNumberOfMonth], [Date].[EnglishMonthName], [Date].[CalendarQuarter], [Date].[CalendarYear],
[Date].[CalendarSemester], [Product].[ProductAlternateKey]
FROM [Reseller Sales]
 LEFT OUTER JOIN [Date] ON [Reseller Sales].[OrderDateKey]=[Date].[DateKey]
 LEFT OUTER JOIN [Product] ON [Reseller Sales].[ProductKey]=[Product].[ProductKey]
WHERE
 (COALESCE(PFCAST( [Reseller Sales].[OrderDate] AS  REAL ))
  <= COALESCE([CallbackDataID(LASTDATE('Date'[Date]))](PFDATAID( [Date].[FullDateAlternateKey] )))) VAND
 (COALESCE(PFCAST( [Reseller Sales].[ShipDate] AS  REAL ))
  >= COALESCE([CallbackDataID(LASTDATE('Date'[Date]))](PFDATAID( [Date].[FullDateAlternateKey] ))));
```

The most interesting part is the **CallbackDataID** for the LASTDATE function. The VertiPaq storage engine is having to send a callback to the formula engine in order to resolve the LASTDATE function. In other words, the LASTDATE function didn't get pushed down into the storage engine. This callback is expensive. Clearly, the LASTDATE function inside the FILTER function is expensive in this query.

In the case of this calculation, the last date doesn't actually vary per Reseller Sales row, so it might improve performance to move the LASTDATE function earlier in the calculation (towards the outside, since it is nested on the very inside of the query).

In the following example, the calculation was rewritten to find the LASTDATE before calling SUMMARIZE and FILTER. By doing so you can guarantee the filter context would contain only one date, which allows the VALUES function to replace the LASTDATE function inside the FILTER expression:

```
MEASURE 'Internet Sales'[M2M] =
CALCULATE(
 CALCULATE(
  'Internet Sales'[Internet Total Units]
  ,SUMMARIZE(
    FILTER(
     'Reseller Sales'
     ,'Reseller Sales'[Order Date]<=VALUES('Date'[Date])
      && 'Reseller Sales'[Ship Date]>=VALUES('Date'[Date])
    )
    ,'Product'[Product Id]
   )
  )
 ,LASTDATE('Date'[Date])
)
```

Running the same query with the updated calculation on a cold cache took only 5 seconds, and on a warm cache it took less than 1 second.

Changing the Data Model

Poor data models can lead to poor query performance. Model optimizations can include:

- Using a good star schema
- Taking advantage of the strengths of VertiPaq
- Utilize DAX time intelligence
- Leverage compression
- Avoid many-to-many filters
- Find and optimize for many-to-many patterns
- Denormalize if it helps

In general, a star schema following Kimball modeling techniques is the optimal data model to build into a Tabular model. It is possible to build a Tabular model on top of a third normal form (3NF) data model such as an OLTP database, but typically such a model will perform worse than a model with fact tables at the proper grain and properly denormalized, conformed dimensions.

For example, if two tables have a one-to-one relationship and queries tend to often navigate the relationship between the tables to compare rows, refactoring these two tables into one may make sense.

Taking into account the performance characteristics of VertiPaq might also influence the data model design.

For example, in an inventory fact table, a common approach is to take a daily snapshot of inventory levels per product per store. However, some products do not turn over quickly, so the inventory level might stay the same for weeks on end. In this case, rather than create a snapshot fact table, you could often build a much smaller table by creating a fact table with one row per product per store per day and only including rows when the inventory level changes. Since VertiPaq is incredibly efficient at scanning data, DAX calculations can do history-to-date style summation of transactions to arrive at the ending inventory level at query time. The following is an example of a calculation rewritten to take advantage of VertiPaq and time intelligence:

```
CALCULATE(
 SUM( InventoryTransactions[Transaction Units] ),
 DATESBETWEEN( 'Date'[Date], BLANK(), LASTDATE( 'Date'[Date] ) )
)
```

Another way to change the data model is to improve the compression rate. Higher compression leads to faster scans and faster queries.

Compression can be improved by changing datatypes and reducing the number of distinct values in a column. For more information, see the section, Do datatypes matter (and the rest of section 4 on encoding).

Another way to improve compression is by spending more time during processing by changing the ProcessingTimeboxSecPerMRow setting discussed in section 4.

Optimize many-to-many calculations

Many-to-many bridge tables in a model, along with DAX calculated measures to filter through the bridge table, can be expensive when the bridge table is sizeable. This design pattern is described well in The Many-to-Many Revolution (http://www.sqlbi.com/articles/many2many/).

It is important both for performance reasons and accuracy reasons to wrap any calculations that use expensive many-to-many logic in an IF function to avoid the many-to-many filter when possible. When you add the IF, DAX will short-circuit and avoid the expensive filter, assuming the user has not set a slice on the many-to-many dimension.

For example, in the following example, a many-to-many calculation filters on a many-to-many Customer dimension. Be to add the highlighted IF function in cases like these!

```
IF(
 ISCROSSFILTERED(Customer[CustomerID]),
 CALCULATE(
  SUM(Fact_Transaction[Amount]),
   SUMMARIZE(
    Bridge_AccountCustomer,
    Account[AccountID]
  )
 ),
 SUM(Fact_Transaction[Amount])
)
```

In some cases, changing the data model can lead to better query performance. Most many-to-many relationships contain a high percentage of common patterns.

For example, in the medical world, a many-to-many diagnosis dimension may contain a number of common patterns, such as patients diagnosed with both high blood pressure and high cholesterol. Instead of using a bridge table that contains one row per patient encounter per diagnosis, the bridge table can be compressed by finding these common diagnosis signatures, assigning them a surrogate key called DiagnosisSignatureKey, and adding this key to the main fact table. This optimization is called the Matrix Relationship Optimization technique and has been well described in a previous whitepaper (http://www.microsoft.com/en-us/download/details.aspx?id=137). Though that whitepaper discusses how to implement many-to-many dimensions in Multidimensional models, the optimization technique applies to optimizing the data model in a Tabular model to compress the bridge table.

In other cases, it is possible to avoid the many-to-many performance problem by simply increasing the number of rows in the main fact table and ensuring that all DAX measures referencing that table avoid double counting the extra rows. In many scenarios, denormalized tables perform better than several normalized tables.

Use natural hierarchies if possible

Multidimensional models contain the concept of attribute relationships inside dimensions. If attribute relationships support a hierarchy, the hierarchy is considered “natural.” However, if attribute relationships do not support a hierarchy (for example, the case of the hierarchy generated where “red” products fall into many different product categories), the hierarchy is declared “unnatural.”

Though Tabular models do not contain the concept of attribute relationships, they do check the data in columns involved in a hierarchy. During the ProcessRecalc phase of model processing, each hierarchy is categorized as natural or unnatural. As Chris Webb describes, this information is exposed in the following DMV:

```
SELECT *
FROM $SYSTEM.MDSCHEMA_HIERARCHIES
WHERE [CUBE_NAME] = 'Model'
AND [HIERARCHY_ORIGIN] = 1
```

The STRUCTURE_TYPE column returned by this DMV contains two values, either **Natural** or **Unnatural**, indicating whether the data in the columns for that hierarchy supports a natural hierarchy.

In Tabular models, querying natural hierarchies may perform better than querying unnatural hierarchies since certain MDX and formula engine code paths are not optimized for unnatural hierarchies. For this reason, consider building natural hierarchies by changing the data. For example, instead of making “Red” a child of “Bikes”, make “Red Bikes” a child of “Bikes.”

Common Query Performance Issues

Though good query performance is typical in Tabular models, there are some common query performance issues. Some of these issues are easy to resolve and some will require improvements in future versions of SQL Server.

This section describes both client- specific issues and some common calculations known to be inefficient. These include:

- Power View queries and visualizations
- Excel client problems
- Miscellaneous performance issues

Common Power View Query Performance Issues

Power View is a new data exploration and interactive reporting experience new with SQL Server 2012 and with Excel 2013. Under the covers, it executes DAX queries.

However, some DAX queries generated by Power View are very expensive, and you have no way of changing the way it writes DAX queries. Therefore, ensure that models used in Power View contain calculated measures which have been optimized to be as efficient as possible and train Power View users to understand some of the following common problems:

Visualizations that return huge data sets

Some visualizations within Power View can generate DAX queries that return very large result sets.

For example, if you build a table visualization which returns millions of rows, the DAX query will request all the data. Power View will begin reading the first several hundred rows, and then Power View will cancel the query to prevent it from consuming further memory.

When the user scrolls down in the table, the same DAX query will be executed again with the START AT clause used to force Analysis Services to begin returning rows starting where Power View previously stopped reading.

These large queries can consume significant memory and processor resources inside Analysis Services.

Visualizations that sample data

Other visualizations, such as the scatter chart, use the SAMPLE function in DAX to ensure that a reasonable number of rows are returned. Another example is the TOPN function, which is used in charts to show the largest values and ignore the rest.

Multiple evaluation of measures

Another performance bottleneck often seen in DAX queries generated by Power View is that measures are evaluated multiple times.

Typically the innermost part of a query will perform a FILTER and a NOT(ISBLANK()) on the calculated measure. Then one or more ADDCOLUMNS functions will reevaluate the measure in the proper filter

context. This type of query displays the correct results on the screen but does prevent the VertiPaq cache from being hit during the multiple evaluations of the same measure, causing performance to be slower than expected.

Finally, when adding fields from multiple tables to the same visualization without first adding a measure to that visualization, Power View will determine which combinations to show by looking for combinations which are valid in any related fact table in the model.

For example, in the following Power View visualization, the combinations of Model Name and Calendar year that appear in this list are the combinations that have rows in one or more of the Internet Sales, Reseller Sales, or Product Inventory fact tables. To generate this simple visualization, all three tables are scanned.

Model Name	Calendar Year
	2005
	2006
	2007
	2008
All-Purpose Bike Stand	2005
All-Purpose Bike Stand	2006
All-Purpose Bike Stand	2007
All-Purpose Bike Stand	2008
Bike Wash	2005
Bike Wash	2006
Bike Wash	2007
Bike Wash	2008
Cable Lock	2005
Cable Lock	2006
Cable Lock	2007
Cable Lock	2008
Chain	2005
Chain	2006
Chain	2007

Figure 16. A table without a measure in Power View

These checks are usually fast thanks to VertiPaq, but in large models with many fact tables, the query used to build this table is usually much more expensive than the one generated when a user adds a measure to the visualization first.

- If users are experiencing performance issues, start design of your chart or table by immediately adding the main measure users want to visualize, and then adding slicers or filters.
- Teach users which visualizations are likely to have worse performance.

Common Excel Query Performance Issues

Given the popularity of Excel as an analysis tool, Excel PivotTables are a very common way to query Tabular models. When you work with Tabular model data in Excel, Excel generates one MDX query per PivotTable, requesting whatever summary data is currently displayed.

Cross-product in PivotTable without a measure

When starting a PivotTable by adding a field from a table to rows or columns, Excel will decide which values from that field to show based upon the default measure. In Tabular models, the default measure is a hidden calculated measure named "__No measures defined" which always return a non-null value.

In other words, until you add a measure to the Values area of a PivotTable, the cross-product of every field you put on rows will be displayed. Note this behavior differs from Power View as described above. For example, in the following PivotTable where no measure has yet been added, every Model Name and Calendar Year combination shows regardless of whether there are any fact tables containing that combination. This behavior cannot be changed, so train users to start PivotTables by first adding a measure they wish to analyze.

	A	B
1	**Model Name**	**Calendar Year**
2		2005
3		2006
4		2007
5		2008
6		2009
7		2010
8	**All-Purpose Bike Stand**	2005
9	**All-Purpose Bike Stand**	2006
10	**All-Purpose Bike Stand**	2007
11	**All-Purpose Bike Stand**	2008
12	**All-Purpose Bike Stand**	2009
13	**All-Purpose Bike Stand**	2010
14	**Bike Wash**	2005
15	**Bike Wash**	2006
16	**Bike Wash**	2007
17	**Bike Wash**	2008
18	**Bike Wash**	2009
19	**Bike Wash**	2010

Figure 17. A sample Excel PivotTable with no measure

Too many unnecessary subtotals in Compatibility mode

Unlike Multidimensional models, Analysis Services 2012 instances in Tabular mode enable a hidden setting, **PreferredQueryPatterns**=1. This setting, in combination with Excel 2010 or later, causes Excel PivotTables to optimize the MDX queries it sends so that fewer unnecessary subtotals are returned. This setting is exposed to Excel through the MDSCHEMA_CUBES discover command:

```
SELECT CUBE_NAME, PREFERRED_QUERY_PATTERNS
FROM $SYSTEM.MDSCHEMA_CUBES
WHERE CUBE_SOURCE = 1
```

For example, the following PivotTable in Excel 2010 was built in an .xls workbook in Compatibility Mode:

	A	B	C
1	Internet Total Sales		
2	Product Category Name	Color	Total
3	Accessories	Black	$72,954.15
4		Blue	$74,353.75
5		NA	$435,116.69
6		Red	$78,027.70
7		Silver	$40,307.67
8	Accessories Total		$700,759.96
9	Bikes	Black	$8,659,117.30
10		Blue	$2,169,055.53
11		Red	$7,646,302.82
12		Silver	$5,073,081.41
13		Yellow	$4,770,587.59
14	Bikes Total		$28,318,144.65
15	Clothing	Black	$106,340.51
16		Blue	$35,687.00
17		Multi	$106,470.74
18		White	$5,106.32
19		Yellow	$86,168.04
20	Clothing Total		$339,772.61
21	Grand Total		$29,358,677.22

Figure 18. Workbook in Compatibility Mode

Note that the PivotTable contains two header rows and 19 rows of data. The MDX query behind this PivotTable actually returns 27 rows, because it returns useless subtotals for each color across all Product Categories:

```
SELECT NON EMPTY CrossJoin (
```

```
 Hierarchize (
  AddCalculatedMembers (
   {
    DrilldownLevel (
     { [Product].[Product Category Name].[All] }
    )
   }
  )
 )
 , Hierarchize (
  AddCalculatedMembers (
   {
    DrilldownLevel ( { [Product].[Color].[All] } )
   }
  )
 )
) DIMENSION PROPERTIES PARENT_UNIQUE_NAME
, HIERARCHY_UNIQUE_NAME ON COLUMNS
FROM [Model]
WHERE ( [Measures].[Internet Total Sales] )
CELL PROPERTIES VALUE, FORMAT_STRING, LANGUAGE, BACK_COLOR, FORE_COLOR, FONT_FLAGS
```

You can use the **About** tab in a tool called [OLAP PivotTable Extensions](https://olappivottableextend.codeplex.com/wikipage?title=About%20Tab) (https://olappivottableextend.codeplex.com/wikipage?title=About%20Tab) to confirm that this PivotTable is an Excel 2002 version format. The tool also offers guidance on how to upgrade the PivotTable.

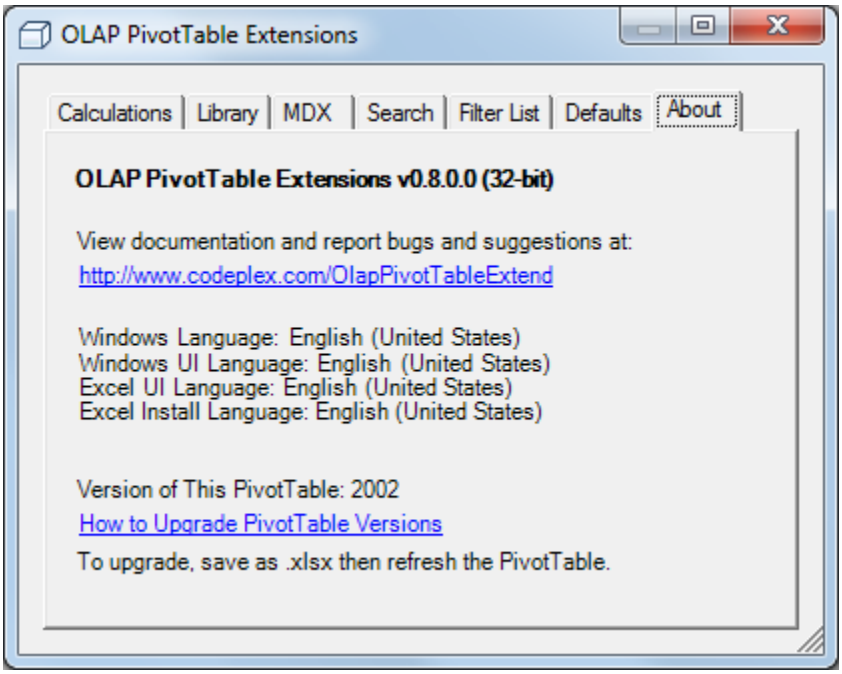


Figure 19. Detecting an older version of a PivotTable

1. Save the file as an .xlsx file (the Open XML file format), and then refresh the PivotTable.
2. When you open the About tab again, the OLAP PivotTable Extensions tool confirms the PivotTable version is now 2010:

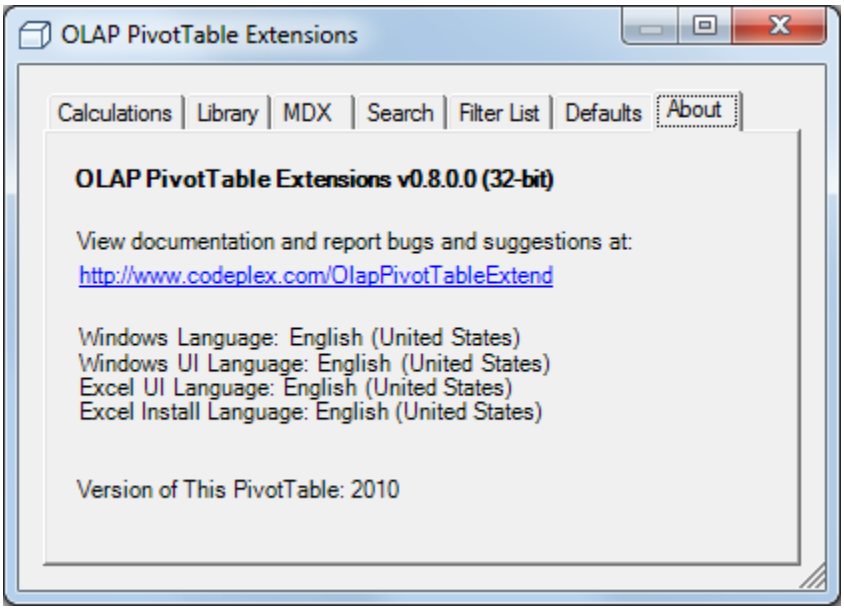


Figure 20. How you can tell that a PivotTable has been updated

After the update, the MDX query for the PivotTable returns 19 rows and reads as follows.

```
SELECT NON EMPTY Hierarchize (
 DrilldownMember (
  CrossJoin (
   { [Product].[Product Category Name].[All],
     [Product].[Product Category Name].[Product Category Name].AllMembers }
   ,
   { ( [Product].[Color].[All] ) }
  )
  , [Product].[Product Category Name].[Product Category Name].AllMembers
  , [Product].[Color]
 )
) DIMENSION PROPERTIES PARENT_UNIQUE_NAME
, HIERARCHY_UNIQUE_NAME ON COLUMNS
FROM [Model]
WHERE ( [Measures].[Internet Total Sales] )
CELL PROPERTIES VALUE, FORMAT_STRING, LANGUAGE, BACK_COLOR, FORE_COLOR, FONT_FLAGS
```

Notice the new DrilldownMember syntax that causes the query to return only useful subtotals.

Of course, returning 27 rows versus 19 will not impact performance substantially, but if your workbook is in Compatibility Mode, with each new column added to the rows axis of the PivotTable, the percentage of useless subtotals returned grows exponentially. For detailed reporting,

PreferredQueryPatterns makes a substantial difference. Ensure Compatibility Mode is not preventing this optimization from taking place.

The current PreferredQueryPatterns implementation in Excel 2010 and Excel 2013 still requests subtotals even when the user chooses not to show subtotals. This shortcoming does make detailed reporting connected to a Tabular model perform more slowly than expected.

If performance of detailed reports is not adequate because of the subtotals in the query, the report designer can create a named set using MDX in Excel 2010 and later put that set on rows. The following example MDX requests no subtotals:

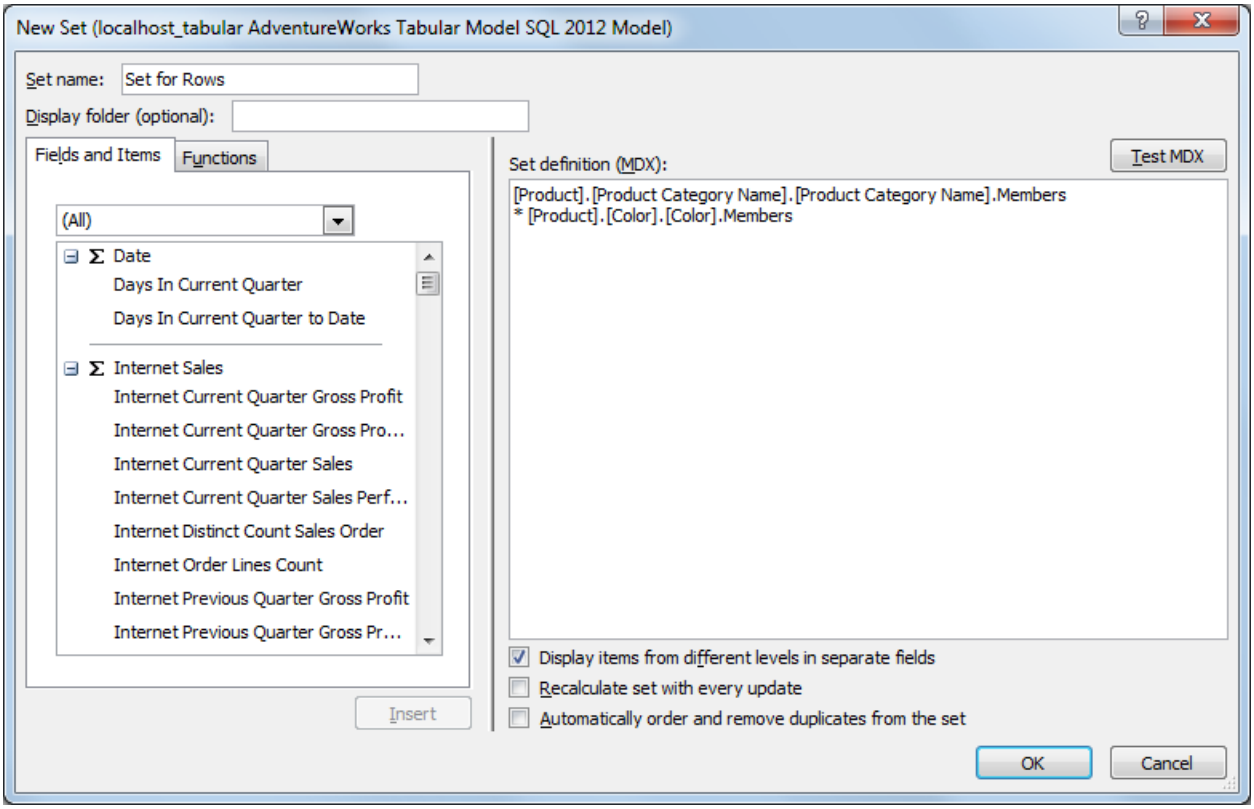


Figure 21. A named set without subtotals

```
[Product].[Product Category Name].[Product Category Name].Members
* [Product].[Color].[Color].Members
```

Another alternative is to use an Excel QueryTable to execute a hardcoded DAX query to return the detailed reporting results. Chris Webb describes the steps to take to [bind an Excel table to an MDX query](http://cwebbbi.wordpress.com/2009/12/18/binding-an-excel-table-to-the-results-of-an-mdx-query/) (http://cwebbbi.wordpress.com/2009/12/18/binding-an-excel-table-to-the-results-of-an-mdx-query/) and the same steps apply to DAX queries.

Slow performance when converting a large PivotTable to CUBEVALUE formulas

Excel PivotTables are objects that expand and contract on the worksheet as filters and data change in the PivotTable. If you need to create a fixed layout, Excel does support converting the PivotTable to CUBEVALUE formulas. This also lets you have a highly customized format, since each cell is independent.

Behind the scenes, Excel batches together several hundred CUBEVALUE formulas into one MDX query requesting a hardcoded list of tuples. Unfortunately, this type of MDX query typically generates hundreds of VertiPaq queries which are run in serial. Converting very large (for example, 5,000 or more cells) PivotTables into CUBEVALUE formulas may perform poorly against a Tabular model.

Slow performance with many PivotTables connected to slicers

Excel 2010 introduced a type of filter object called a slicer, which can be used in multiple PivotTables or PivotCharts, in addition to cross-filtering other slicers. However, when a user builds a very big Excel workbook with many slicers connected to many PivotTables, performance can degrade significantly.

Keep slicers simple:

- Consider reducing the number of slicers.
- Use a PivotTable filter instead of slicers.
- Disable cross-filtering if necessary.

These options are discussed in more detail by Rob Collie (http://www.powerpivotpro.com/2010/07/slicers-and-pivot-update-performance/).

Other Known Query Performance Issues

The following are some miscellaneous known query performance issues. In SQL Server 2012 Analysis Services, there is no silver-bullet fix for these issues. These issues are presented to explain the underlying complexities.

Strict mode in combination with arbitrary shapes can be inefficient

DAX always performs short-circuit evaluation of functions. As mentioned in the discussion of the DAX Query Plan above, this is known as strict mode.

However, in some cases, this behavior causes both branches to be evaluated, each over a subset of the space. This subset of space for each branch is often an inefficient shape called an arbitrary shape. In such situations it could be more efficient to evaluate both branches over the entire calculation space than to use strict mode and operate on arbitrary shapes. However, in DAX in SQL Server 2012 Analysis Services, it is not possible to control this behavior.

For details about what an arbitrary shape is, see the blog post by Thomas Kejser: http://kejserbi.wordpress.com/2006/11/16/arbitrary-shapes-in-as-2005/.

Group By occurs even when querying at the lowest level

VertiPaq currently does not have a projection operator like the one in T-SQL. Currently, all VertiPaq queries perform a group by, even if querying the lowest granularity data in a table, and produce a hash table with all the related keys. This operation can be expensive on large tables.

VertiPaq requires callbacks for some types of currency arithmetic

While most simple arithmetic can be pushed down into VertiPaq, some arithmetic operations that require rounding or datatype rescaling will utilize a callback. Most of the arithmetic scenarios requiring a callback involve the Currency datatype.

For example, the following query multiplies Unit Price (Currency datatype) and Unit Price Discount Pct (floating point datatype). This operation requires a callback.

```
EVALUATE
ROW(
 "Discount",
 SUMX(
  'Reseller Sales',
  'Reseller Sales'[Unit Price] * 'Reseller Sales'[Unit Price Discount Pct]
 )
)
```

Callbacks are more expensive than native VertiPaq arithmetic, and also prevent VertiPaq caching. The recommendation below to prefer Currency datatypes where possible still stands. However, in some exceptional situations, changing Currency datatypes to floating point datatypes (Data Type=Decimal Number in SQL Server Data Tools) may improve query performance by eliminating the callback.

Jeffrey Wang, Principal Development Lead on the Analysis Services team, blogged (http://mdxdax.blogspot.com/2013/06/the-currency-data-type-and-vertipaq.html) more specifics about which datatype combinations require callbacks.

Large memory usage because filters are not pushed down to VertiPaq in MDX query

This issue affects only MDX queries, but can have a significant effect on performance in large models. To see how it works, using the Adventure Works Tabular model with a Reseller Sales table expanded to 100 million rows, run the following MDX query:

```
select {[Measures].[Reseller Order Lines Count] ,
 [Measures].[Reseller Total Freight] ,
 [Measures].[Reseller Total Sales]} on 0,
NON EMPTY [Reseller Sales].[Sales Order Number].[Sales Order Number].AllMembers
*[Reseller Sales].[Carrier Tracking Number].[Carrier Tracking Number].AllMembers
*[Reseller Sales].[Sales Order Line Number].[Sales Order Line Number].AllMembers
on Rows
from [Model]
where (
[Product].[Color].&[Black]
,[Date].[Date].&[2005-07-01T00:00:00]
,[Reseller Sales].[Revision Number].&[1]
)
```

This query consumes many GB of memory while spooling the following VertiPaq query:

```
SELECT
```

```
[Reseller Sales].[SalesOrderNumber], [Reseller Sales].[SalesOrderLineNumber],
 [Reseller Sales].[CarrierTrackingNumber]
FROM [Reseller Sales]
WHERE
 [Reseller Sales].[RevisionNumber] = 1;
```

Note that it is not passing in the Date or Product dimension filters to this VertiPaq query, so the query returns millions of rows. The filter on RevisionNumber=1 doesn't eliminate many rows. And even if you exclude that filter, the query behaves the same.

In contrast, the equivalent DAX query does push all filters down to the scan of Reseller Sales and performs great:

```
EVALUATE
CALCULATETABLE(
 ADDCOLUMNS(
  SUMMARIZE(
   'Reseller Sales'
   ,'Reseller Sales'[Sales Order Number]
   ,'Reseller Sales'[Carrier Tracking Number]
   ,'Reseller Sales'[Sales Order Line Number]
  )
  ,"measure1", [Reseller Order Lines Count]
  ,"measure2", [Reseller Total Freight]
  ,"measure3", [Reseller Total Sales]
 )
 ,Product[Color] = "Black"
 ,'Date'[Date] = DATE(2005,7,1)
 ,'Reseller Sales'[Revision Number] = 1
)
```

The DAX query generates the following VertiPaq query, which does include the Product and Date filters:

```
SELECT
[Reseller Sales].[SalesOrderNumber], [Reseller Sales].[SalesOrderLineNumber],
 [Reseller Sales].[CarrierTrackingNumber]
FROM [Reseller Sales]
 LEFT OUTER JOIN [Date] ON [Reseller Sales].[OrderDateKey]=[Date].[DateKey]
 LEFT OUTER JOIN [Product] ON [Reseller Sales].[ProductKey]=[Product].[ProductKey]
WHERE
 [Date].[FullDateAlternateKey] = 38534.000000 VAND
 [Product].[Color] = 'Black' VAND
 [Reseller Sales].[RevisionNumber] = 1;
```

This is a known issue. Consider voting for it on Connect (https://connect.microsoft.com/SQLServer/feedback/details/787006/large-dimension-on-rows-of-mdx-query-causes-huge-vertipaq-query) if you are experiencing a similar issue in your model.

Possible issues when querying large dimension tables

On models with huge dimension tables (for example, 100 million rows or more), take care in how those dimensions are queried. For example, try this query against a copy of AdventureWorks with a 100 million row Product dimension:

```
EVALUATE
CALCULATETABLE(
 FILTER(
  CROSSJOIN(
   VALUES('Date'[Date]),
   SUMMARIZE(
```

```
    'Product',
    'Product'[Product Id],
    'Product'[Product Name]
   )
  ),
  NOT(ISBLANK('Reseller Sales'[Reseller Total Sales]))
 ),
 'Sales Territory'[Sales Territory Country] = "United States",
 'Reseller Sales'[DueDateKey] = 20060910
)
```

The following VertiPaq query is generated with no filter due to use of the SUMMARIZE function:

```
SELECT
[Product].[ProductAlternateKey], [Product].[EnglishProductName]
FROM [Product];
```

That VertiPaq query will return 100 million rows and consume a tremendous amount of memory.

However, the query can be optimized by summarizing the fact table. This step also allows you to remove the NOT(ISBLANK()) filter, because the Sales Amount column is never null.

```
EVALUATE
CALCULATETABLE(
 SUMMARIZE(
  'Reseller Sales',
  'Date'[Date],
  'Product'[Product Id],
  'Product'[Product Name]
 ),
 'Sales Territory'[Sales Territory Country] = "United States",
 'Reseller Sales'[DueDateKey] = 20060910
)
```

An alternate approach is to combine the large dimension's attributes into the fact table, as described in this article by Teo Lachev (http://prologika.com/CS/blogs/blog/archive/2013/03/17/transactional-reporting-with-tabular.aspx).

Heap fragmentation in models with thousands of columns

Analysis Services in SQL Server 2012 ships with default memory settings (MemoryHeapType=2 and HeapTypeForObjects=0) that enable it to use the Windows Low Fragmentation Heap for memory allocations. This memory heap performs well under concurrency and is not as prone to fragmentation as other types of memory heaps. However, Windows has a hard-coded limit that prevents memory allocations larger than 16KB from using the Windows Low Fragmentation Memory Heap. When many memory allocations larger than 16KB occur, this can lead to severe heap fragmentation over time and cause Analysis Services performance to degrade until an administrator restarts the Analysis Services process. This problem can occur in both Multidimensional and Tabular models.

While it is uncommon to have Tabular models with thousands of columns since the model must fit in memory, it is technically possible. Columns in tables and user-defined hierarchies are both referred to as "hierarchies." If a model has 2,045 "hierarchies" or more, certain allocations will require more than 16KB and will cause heap fragmentation.

Analysis Services administrators can see how many hierarchies there are in a particular model by counting the rows returned from this DMV:

```
SELECT [HIERARCHY_UNIQUE_NAME]
FROM $SYSTEM.MDSCHEMA_HIERARCHIES
WHERE [CUBE_NAME] = 'Model'
```

In addition to the 2,045 hierarchies threshold, there is also a 4,092 levels threshold per model. The Analysis Services administrator can see how many levels there are in a particular model by counting the rows returned from this DMV:

```
SELECT [LEVEL_UNIQUE_NAME]
FROM $SYSTEM.MDSCHEMA_LEVELS
WHERE [CUBE_NAME] = 'Model'
```

Even if the model is within these threshold limits, the number of hierarchies (columns and user hierarchies) defines the "cube space." Currently there is no way to mark a column (such as a numeric column in a fact table) so that it does ***not*** build a hierarchy. Thus each new column adds width to internal memory structures, caches, and coordinates. Lookups have extra width to compare, and iterations have extra width to update. The point is that very wide models add significant extra cost to calculations.

Model developers should be judicious about each new table and column they add to a model. Developers should generally strive to keep their models under 1,000 hierarchies for optimal performance.

Query Troubleshooting Tips

The following tips will help you isolate the problem with your queries.

Determining which process is out of memory

When executing queries that return large result sets, it is possible to run out of memory in the client application. For example, suppose you are in SQL Server Management Studio running a query that returns 117,000 rows and dozens of columns, and get the error, System.OutOfMemoryException. Did the Analysis Services instance run out of memory and return an error, or did Management Studio run out of memory?

The text of the error message and additional information in Task Manager can help you determine the cause. In this case, the System.OutOfMemoryException is a .NET error message, which is a different error message than the message generated when the server runs out of memory. This indicates that it

is possibly the client application running out of memory. Client applications that use .NET include SSMS connections to Analysis Services and grid views of SSAS data.

In Task Manager, Management Studio is listed as “Ssms.exe *32” which means it is a 32-bit process. As the query executes and streams results to Management Studio, the peak working set rises to nearly 1GB, and then the OutOfMemoryException is displayed.

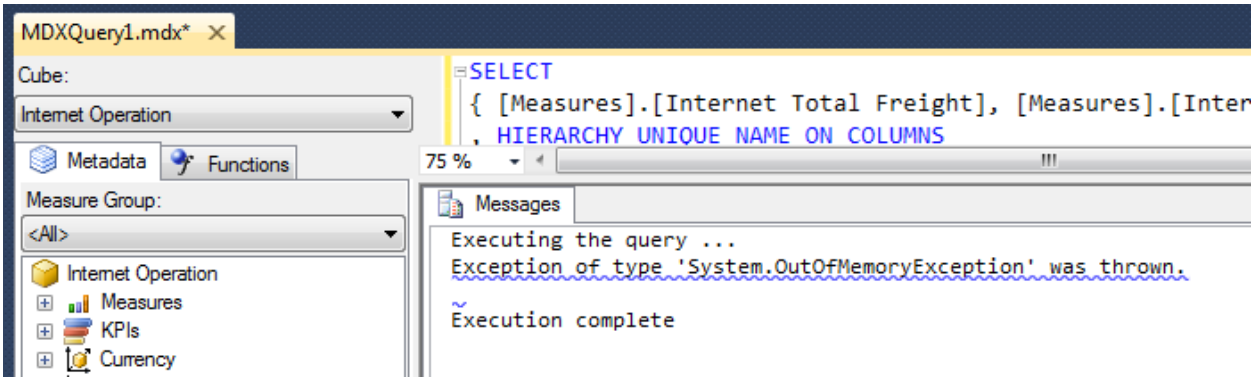


Figure 22. Error for out of memory condition in client

If the out-of-memory error occurred within Analysis Services, the error message would read, “The operation has been cancelled because there is not enough memory available for the application. If using a 32-bit version of the product, consider upgrading to the 64-bit version or increasing the amount of memory available on the machine.” Another option may be to tune the query.

Determining whether the query, network, or provider is slow

On queries that return large result sets, most of the time required to transfer the response over the network is included in the **Duration** column of the **Query End** event. Because serialization is different per provider (OLEDB, ADOMD.NET, and provider versions) and per server settings, perform some tests in your environment.

Also, because each client tool performs differently with large result sets, use a timer to measure the client-side experience, watch network usage using Performance Monitor, and compare these timings to the **Duration** column of the **Query End** event. If the client application is not able to keep up reading the network packets, Analysis Services will slow down serialization of the result set.

For example, a Reporting Services report executing an MDX query that returns 117,000 very wide rows in the result set had a Duration value of 9 minutes on the Query End event when run over a VPN network connection between Reporting Services and Analysis Services. However, when run with a fast network connection, that same query from Reporting Services had a Duration of 2 minutes.

The type of client application and driver used to run the query also makes a difference.

For example, Reporting Services reports that use a data source of type “Microsoft SQL Server Analysis Services” use an ADOMD.NET data reader to read the MDX or DAX query results. ADOMD.NET can

receive query results in compressed XML format, but not in compressed binary XML format. Excel PivotTables uses the Analysis Services OLE DB provider, which is able to receive query responses in compressed binary XML. In one test, compressed binary XML resulted in 10x fewer network packets than compressed XML.

The SQL Server 2012 Analysis Services OLE DB provider is the default provider for Excel 2013 and can be installed from the SQL Server 2012 SP1 Feature Pack (http://www.microsoft.com/en-us/download/details.aspx?id=35580) for use with prior versions of Excel. This version of the driver is used when the Excel connection string specifies Provider=MSOLAP.5. This version of the driver includes an optimized serialization format which helps significantly on query results which contain repetition of dimension members on many rows or columns due to a crossjoin of many fields. In one test over a VPN connection, using the MSOLAP.3 driver (SQL Server 2005 Analysis Services, which is the driver installed with Excel 2007) refreshing a large PivotTable with many fields on the rows axis took 75 seconds, while using the MSOLAP.5 driver took 51 seconds.

Use the SQL Server 2012 Analysis Services OLE DB provider (MSOLAP.5) wherever possible. This driver includes an optimized serialization format which helps significantly on query results.
For earlier versions of Excel, you can install the provider from the SQL Server 2012 SP1 Feature Pack (http://www.microsoft.com/en-us/download/details.aspx?id=35580)

Summary—Tuning Query Performance

In summary, query performance is typically fast against Tabular models, but there are many ways in which performance can degrade. One way to optimize DAX calculations is to push as much calculation logic down into the VertiPaq scan as possible, to achieve optimal query performance. Understanding DAX Query Plans and VertiPaq queries will also help the model developer tune DAX calculations.

4. Processing and Partitioning

Processing is the term used for loading or reloading the data in a model. Processing typically occurs during model deployment, on demand, or on a schedule determined by the model administrator.

Partitioning of tables in a Tabular model allows the administrator to refresh different subsets of a table independently. Partitions in Tabular models do not impact query performance, and they do not allow parallel processing within a single table. Therefore, partitions are solely intended for administrative purposes. Partitioning is one way of accomplishing incremental loading to minimize processing time.

This section contains practical advice, solutions to common problems, and recommendations.

If model processing is consuming too much time, too much memory, or causing downtime for users, this section can help.
The reverse is also true; if processing is well within the time limits but queries are too slow, the administrator could use this section to change settings to spend *more* time during processing in order to increase compression and query performance.

To guide you in making the right decision, the architecture and terminology of processing is provided as background before the recommendations are made.

This section first discusses permissions and processing options. Next, common processing methodologies are explained. Then the internals encoding and compression phases of processing are explained. Finally, some optimization suggestions are made.

Understanding Processing

This section describes how Analysis Services loads the physical dictionary, hierarchy, segment and relationship structures presented in the architecture overview in section 2.

An understanding of Tabular model processing will allow the administrator to optimize performance to meet refresh and uptime requirements.
Remember that processing is typically repeated on a schedule, such as weekly, daily, or more often; therefore, optimizing processing is important.
In addition to such scheduled periodic refreshes, processing also takes place both when model changes are deployed to the server and during development of the model.

Permissions Required to Process a Model

Processing is an administrative operation that can only be performed by administrators of the Analysis Services instance or by users in a role that grants the Process, Read and Process, or Administrator permission within that Tabular model database.

The following figure shows the Role Manager dialog inside SQL Server Data Tools. This interface provides an easy way to assign the permissions most useful for tabular models.

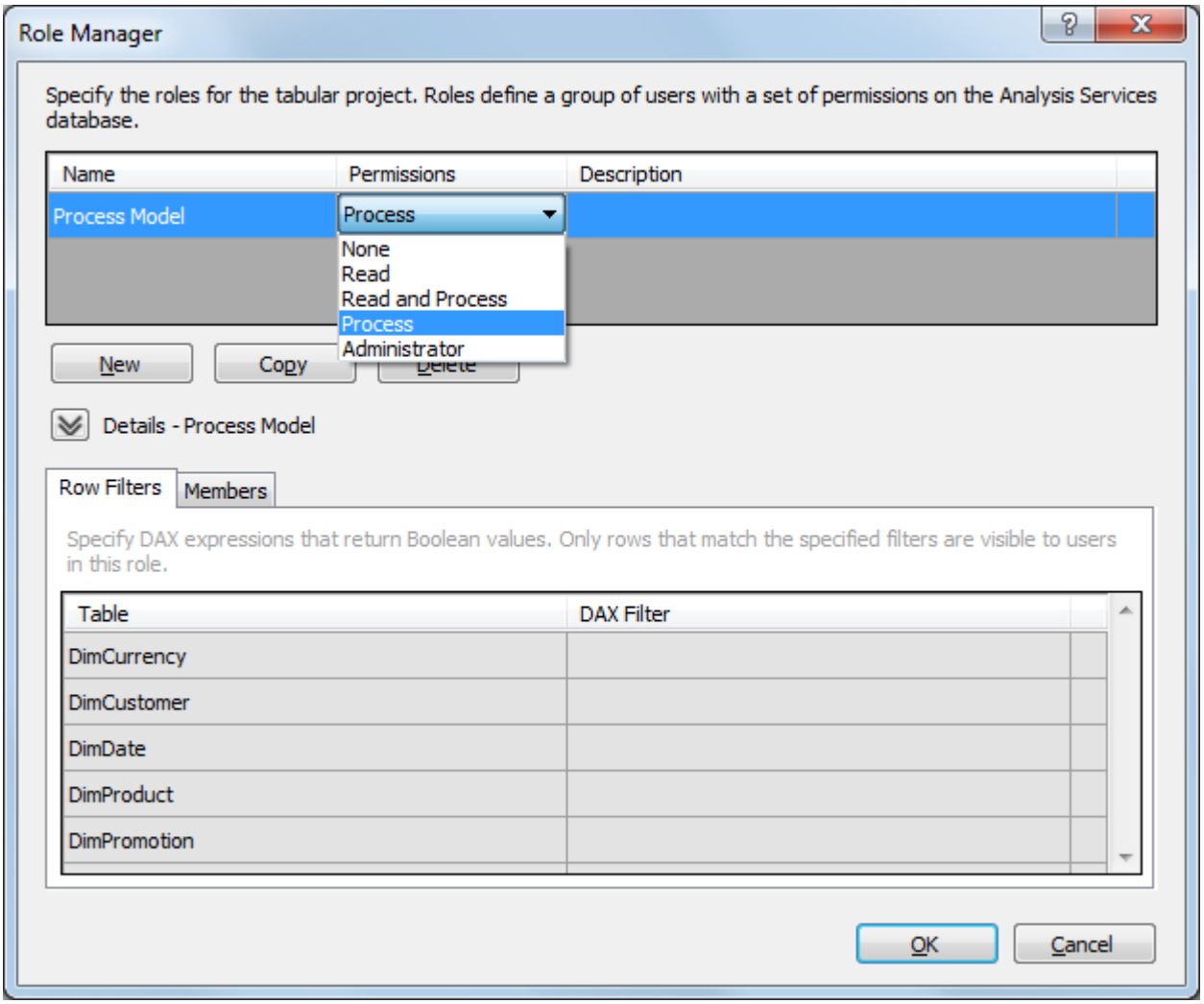


Figure 23. The Role Manager dialog inside the development environment, SQL Server Data Tools

Processing Options for Tabular Models

The following table summarizes the different processing operations available to the model administrator.

Table 3. Processing options described

Processing Option	Description	Connects to Data Source?	Includes ProcessRecalc?	Affected Objects
ProcessData	Reads data from the relational source, encodes and compresses it into dictionaries. Replaces existing data.	Yes	No	Dictionaries
ProcessRecalc	Builds any hierarchies, calculated columns, and relationships that need to be built. Finishes immediately if all of the above are already built. Converts unprocessed tables into queryable but empty tables.	No	Yes	Calculated columns Hierarchies Relationships
ProcessFull	ProcessData and ProcessRecalc	Yes	Yes	Dictionaries Calculated columns Hierarchies Relationships
ProcessAdd	Reads data from the relational source, encodes and compresses it into dictionaries. Appends to existing data. Performs a ProcessRecalc.	Yes	Yes	Dictionaries Calculated columns Hierarchies Relationships
ProcessDefault	Processes any unprocessed tables, partitions, hierarchies, calculated columns, and relationships. Finishes immediately if all objects are processed.	If necessary	Only if necessary	Only if necessary
ProcessDefrag	Rebuilds dictionaries to reduce memory usage by removing items from the dictionary which no longer exist.	No	Yes	Dictionaries
ProcessClear	Drops all the data in the object.	No	No	Dictionaries Calculated columns Hierarchies Relationships

Different processing operations are applicable to the whole database, a whole table, or an individual partition:

Table 4. Objects touched by different processing options

Processing Option	Database	Table	Partition
ProcessData		Available	Available
ProcessRecalc	Available		
ProcessFull	Available	Available	Available
ProcessAdd			Not in UI
ProcessDefault	Available	Available	Available
ProcessDefrag	Not in UI	Available	
ProcessClear	Available	Available	Available

Because the timing and manner of processing has a huge impact on the availability and freshness of your model, we recommend that you take a minute to learn more about each processing option, and how each affects performance.

ProcessData

ProcessData can be run on a table or a partition. This processing operation connects to the relational data source, reads uncompressed data over the network, encodes it into integers, and then compresses it into dictionaries. Other auxiliary objects like hierarchies, calculated columns, and relationships are not built until ProcessRecalc is run. Thus the model will not be queryable until ProcessRecalc is run.

If run on a table, all column dictionaries are dropped and rebuilt inside a transaction.

If users could be querying the model during processing and ample memory is available, make sure to use transactional processing and to include ProcessRecalc in the processing batch. See the Example of ProcessData Plus ProcessRecalc section for more information.

ProcessData will use resources in the relational database in addition to consuming about 1.5 to 2 cores in Analysis Services for reading, encoding, and compressing data. Network bandwidth will be used for remote data sources and network speeds can impact processing performance. Analysis Services needs enough memory to hold about 16 million (twice the DefaultSegmentRowCount setting) uncompressed rows of data, to choose an encoding scheme, and to compress it. It also needs enough memory to hold the distinct list of values (the dictionary) in memory for all columns for the whole table. If there is available memory and CPU on the Tabular server, focus on optimizing the SQL query, as described below.

ProcessRecalc

ProcessRecalc can be performed only on the whole database. It never connects to the relational source system. It analyzes the dependencies between calculated columns, detects which relationships are not yet built, and sequences processing tasks to do as much in parallel as possible. As such, it can be is very

memory and processor intensive depending on the size of the data and the complexity of the calculated columns. ProcessRecalc is incremental in nature so that it returns instantly if there is no work to be done.

If ProcessRecalc is consuming too much memory or processor resources, consider leveraging MaxParallel in XMLA to lower parallelism and memory usage. For example, the following script runs a ProcessRecalc with a MaxParallel value of 1, meaning suppress parallel plan generation and run all tasks in serial:

```
<Batch xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">
 <Parallel MaxParallel="1">
  <Process>
   <Type>ProcessRecalc</Type>
   <Object>
    <DatabaseID>AdventureWorks</DatabaseID>
   </Object>
  </Process>
 </Parallel>
</Batch>
```

Always use ProcessRecalc at the end of any processing batches to ensure the model ends up in a queryable state.
If ProcessRecalc is consuming too much time to fit in your processing window, monitor the Duration column of Profiler events carefully to determine which objects or calculated columns are the bottleneck.

ProcessFull

ProcessFull can be performed on the whole database, a whole table, or a partition. It performs all the work of ProcessData and ProcessRecalc.

Running ProcessFull on the **whole database** is the most straightforward way of reloading every table while keeping the old data queryable by users until the transaction commits and the new data is queryable. However, it is also uses the most memory of any processing option. If a model consumes 10GB of RAM at rest, during a ProcessFull on the whole database, the server must have enough memory to hold 10GB for the old data, 10GB for the new data, and approximately another 5GB-10GB for various processing overhead structures and operations.

If users will never query the model during processing, the model administrator should consider running ProcessClear in a separate transaction prior to running a ProcessFull. This reduces memory usage, because it avoids having two copies of the data in memory during processing.

ProcessFull on a **table** will fully reload all partitions from the relational database, and then perform a ProcessRecalc only on dependent structures. Thus, if subsequent processing transactions run ProcessFull on other tables, some ProcessRecalc work may be done twice.

For example, if you perform ProcessFull on Table A and Table A contains a calculated column referring to Table B, that calculated column will be computed at the end of the ProcessFull on Table A. However, if you run ProcessFull on Table B in a separate transaction (or even in a separate XMLA Parallel tag in the same transaction), that same calculated column in Table A will be recomputed again.

To avoid this problem, instead of running ProcessFull on multiple tables, consider running ProcessData on those tables followed by a ProcessRecalc in the same transaction. (See an example below.)

- If ProcessRecalc is run in the same transaction, there will be no downtime.
- If ProcessRecalc is run in a separate transaction, there will be model downtime until ProcessRecalc completes.

ProcessFull on a **partition** performs a ProcessData on that partition and then a ProcessRecalc on the whole table and any dependent objects. Therefore, it is recommended that instead of performing ProcessFull on multiple partitions in the same table, you run ProcessData on individual partitions and then run ProcessRecalc on the database. Realize that if ProcessRecalc is run in a separate transaction, there will be some model downtime until ProcessRecalc completes.

If ProcessFull is performing poorly, perform ProcessData and ProcessRecalc separately as a troubleshooting step to isolate which operation is the bottleneck, and then refer to that section for optimization suggestions.
Alternately, monitor the Duration column in a Profiler trace to see which operations and columns are the most costly.

ProcessAdd

ProcessAdd is not available in the Management Studio user interface, however, it is available programmatically in XMLA, AMO, and PowerShell as described in the article by Marco Russo, Incremental Processing In Tabular Using Process Add (http://www.sqlbi.com/articles/incremental-processing-in-tabular-using-process-add/).

ProcessAdd provides an out-of-line query binding which specifies a SQL query that returns only the new rows in the data source that must be appended to the partition. It is the responsibility of the administrator to ensure that the query binding returns only those rows currently not in the Tabular model. In other words, ProcessAdd supports inserts to a partition, but does not support updates or deletes.

- If no column is marked as a Row Identifier, then duplicate rows will be added without error.

- If a Row Identifier column is set, then an error similar to the following will be received during ProcessAdd:

  Column 'GeographyKey' in Table 'Geography' contains a duplicate value '292' and this is not allowed for columns on the one side of a many-to-one relationship or for columns that are used as the primary key of a table.

ProcessAdd can only be run on a partition. Like ProcessFull, ProcessAdd maintains the column dictionaries and then performs a ProcessRecalc on dependent structures. Unlike ProcessFull, it does not clear the existing rows in the partition at the beginning of the command, making it a very efficient way to perform incremental processing.

ProcessDefault

ProcessDefault does the minimum work necessary to make an object queryable. If necessary, it will perform ProcessData and ProcessRecalc. It performs the work incrementally, so if the object it is run against is already processed and queryable (all dictionaries, hierarchies, relationships and calculated columns are built), then ProcessDefault returns instantly.

One important point is that ProcessDefault does not have insight into whether data in the source system has changed. If a table is already processed, it will not know whether the underlying data has changed, and it will not reprocess the table. This task is the responsibility of the model developer and administrator.

ProcessDefrag

ProcessDefrag rebuilds dictionaries to remove any items which no longer exist in the table. It is only relevant on tables with multiple partitions where partitions are removed or where individual partitions are processed. The most common scenario where ProcessDefrag may help is after deleting a partition. For example, suppose one of the retail stores in your database closes. After the last partition containing data for the closed store has been deleted, that StoreID will continue to occupy space in the table's dictionary until a ProcessDefrag operation is run on the table.

Even if no partitions are deleted, reprocessing individual partitions can cause fragmentation due to data in the source system changing over time. Fragmented dictionaries do not cause queries to return incorrect results, but they do waste memory and degrade query performance.

While ProcessDefrag can be run on the entire database, by design it will first performs a ProcessData on any unprocessed tables. For this reason, you might prefer running it on individual tables.

ProcessDefrag does not need to reconnect to the relational database to reload data. It simply rebuilds dictionaries and column segments. However, it is expensive in terms of memory and time. To detect whether a column has a fragmented dictionary and ProcessDefrag would help, see Running ProcessDefrag.

ProcessDefrag requires a subsequent ProcessRecalc to leave your model in a queryable state. Therefore, in situations where users may be querying the model during ProcessDefrag, the best practice is to include ProcessRecalc in the transaction with one or more ProcessDefrag commands, as in the following example:

```
<Batch xmlns='http://schemas.microsoft.com/analysisservices/2003/engine' Transaction='true'>
 <Process>
  <Type>ProcessDefrag</Type>
  <Object>
   <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
   <DimensionID>Internet Sales_fdac9a13-4019-4773-b193-7cca3a4883eb</DimensionID>
  </Object>
 </Process>
 <Process>
  <Type>ProcessDefrag</Type>
  <Object>
   <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
   <DimensionID>Product Inventory_271ff796-8f0b-4f89-8add-d281264ec6d8</DimensionID>
  </Object>
 </Process>
 <Process>
  <Type>ProcessRecalc</Type>
  <Object>
   <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
  </Object>
 </Process>
</Batch>
```

ProcessClear

ProcessClear drops all data and makes that object unqueryable. It can be run on a partition, a table, or the whole database. On servers where memory is tight and the model will not be queried during processing, the administrator can run a ProcessClear in a separate transaction to drop data and have more memory available for subsequent processing transactions.

Determining Whether a Model is Processed

The easiest way to determine if a model is processed is to view the model properties in SQL Server Management Studio.

1. In SQL Server Management Studio, in Object Explorer, open a connection to the Analysis Services Tabular instance you want to check.
2. Select and then right-click the database node, and then select Properties.
3. Click the **Model** tab.

If the **State** property is **Processed,** then all partitions in all tables are processed. If the value of the **State** property is **PartiallyProcessed**, then some partitions are processed. If the value of **State** is **Unprocessed**, then no partitions are processed.

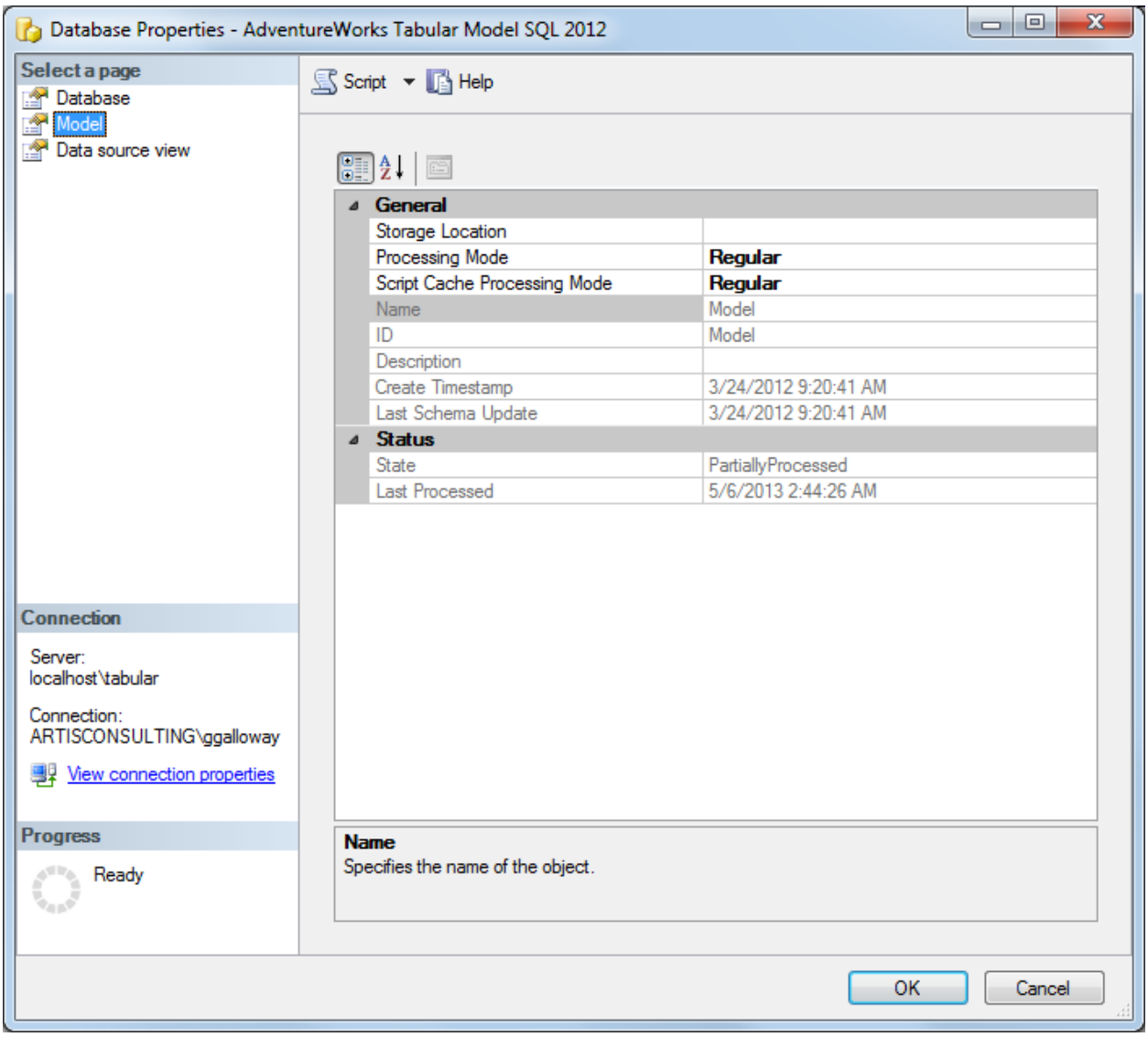


Figure 24. Determining the processing status of a database

If only some tables are processed, you can find out which ones.

1. If the **State** property is **PartiallyProcessed**, expand the **Tables** node in Object Explorer.
2. Right click on any table and select **Properties**.
3. Look for partitions where the value of **LastProcessed** is **Never**, and then select the next table from the **Table** dropdown list.
4. Continue clicking the down arrow to review the list of tables, looking for partitions that aren't processed.

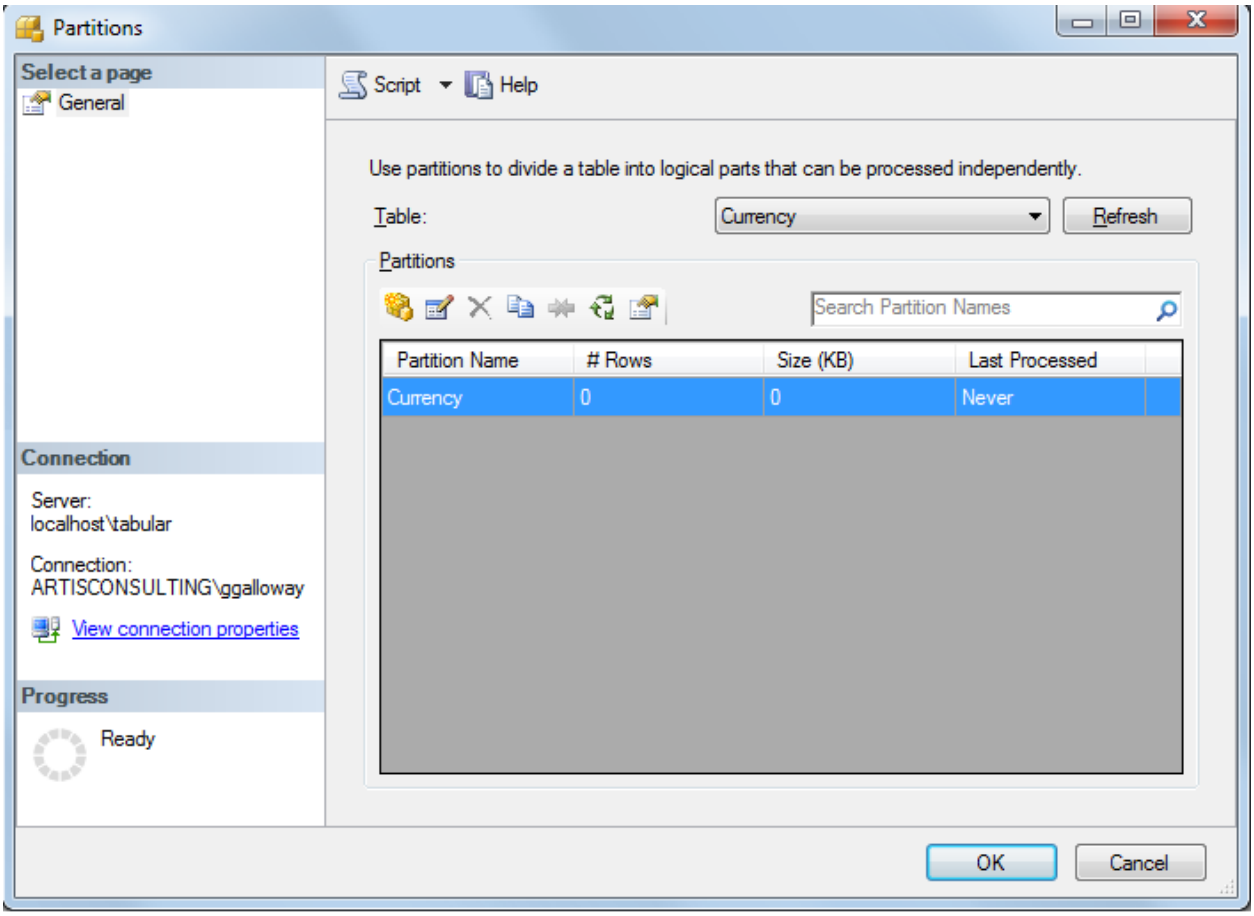


Figure 25. Partitions dialog box

Even if the model appears to have been processed, it is possible that the model is currently unqueryable, and you need to run ProcessRecalc. The easiest way to determine whether ProcessRecalc is needed is simply to run the command, ProcessRecalc. Because ProcessRecalc works incrementally, it will only do any work if any hierarchies, relationships, or calculated columns need to be built. If all of these objects are up-to-date, ProcessRecalc will complete almost instantaneously.

Calculated columns that use volatile DAX functions such as NOW() must be rebuilt every time ProcessRecalc is run.
For more information, see Troubleshoot Recalculation.

Detecting exactly which objects need ProcessRecalc is more complex, but the following PowerShell script uses AMO to perform this task. Note that this script assumes that all columns will have

hierarchies; however, in future versions of Analysis Services (after SQL Server 2012), this assumption may not be correct.

To use this script:

1. Replace the highlighted text with your server name and database name.
2. Save the file with the file name, IsProcessRecalcNeeded.ps1.
3. In Windows Explorer, right click the file you just saved, and select **Run with PowerShell**.

```
[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.AnalysisServices")

$srv = new-object Microsoft.AnalysisServices.Server

#Connect to the server.
$srv.Connect("localhost\tabular")
$dbname = "AdventureWorks Tabular Model SQL 2012"
$db = $srv.Databases.GetByName($dbname)

$bProcessRecalcNecessary = $false

foreach ($dim in $db.Dimensions)
{
 #are the attribute hierarchies on each column built?
 #also detects whether calculated columns are built
 foreach ($da in $dim.Attributes)
 {
  if ($da.AttributeHierarchyProcessingState -ne "Processed")
  {
   write-host "Table '" $dim.Name "' column [" $da.Name "] " $da.AttributeHierarchyProcessingState
   $bProcessRecalcNecessary = $true
  }
 }

 #are the user hierarchies built?
 foreach ($h in $dim.Hierarchies)
 {
  if ($h.ProcessingState -ne "Processed")
  {
   write-host "Table '" $dim.Name "' hierarchy [" $h.Name "] " $h.ProcessingState
   $bProcessRecalcNecessary = $true
  }
 }

 #are the relationships built?
 foreach ($r in $dim.Relationships)
 {
  if ($r.ProcessingState -ne "Processed")
  {
   $fromTable = $db.Dimensions[$r.FromRelationshipEnd.DimensionID]
   $fromColumn = $fromTable.Attributes[$r.FromRelationshipEnd.Attributes[0].AttributeID].Name
   $toTable = $db.Dimensions[$r.ToRelationshipEnd.DimensionID]
   write-host "Table '" $fromTable "'[" $fromColumn "] relationship to '" $toTable "' " $r.ProcessingState
   $bProcessRecalcNecessary = $true
  }
 }
}

if ($bProcessRecalcNecessary -eq $false)
{
 write-host "ProcessRecalc does not currently need to be run."
}

if ($svr.Connected)
{
```

```
$svr.Disconnect()
}

$null = read-host "Press enter to continue..."
```

Understanding the Phases of Processing

Understanding the internal operations performed by a Tabular server during processing can help the model administrator choose the best processing scheme and optimize processing.

Phase 1: Opening the connection and running the query

The first step in processing is reading data from the relational source. A connection is opened using the connection string and credentials that are stored in the data source. That connection must succeed within 60 seconds, per the **ExternalConnectionTimeout** server setting, unless overridden by the **Connect Timeout** property on the connection string.

If multiple tables are being processed in parallel, a hidden property on the data source called **MaxActiveConnections** comes into play. By default, it specifies that no more than 10 connections for each data source can be opened at the same time. This property is not visible in SSDT, but it can be changed by editing the **Maximum Number of Connections** property on the data source in SQL Server Management Studio.

Beware!! Any changes that you make in SQL Server Management Studio can be lost on the next model deployment.
To avoid losing your customizations, consider creating a new project in SSDT using the **Import from Server (Tabular)** option to import your settings back into your source code.

However, be warned that increasing the number of connections can swamp the relational data source. For example, when many queries are run against SQL Server and each query uses a high degree of parallelism, SQL Server may throw an error saying it doesn't have enough worker threads available. Test overall throughput when changing MaxActiveConnections and consider changing MAXDOP settings in SQL Server.

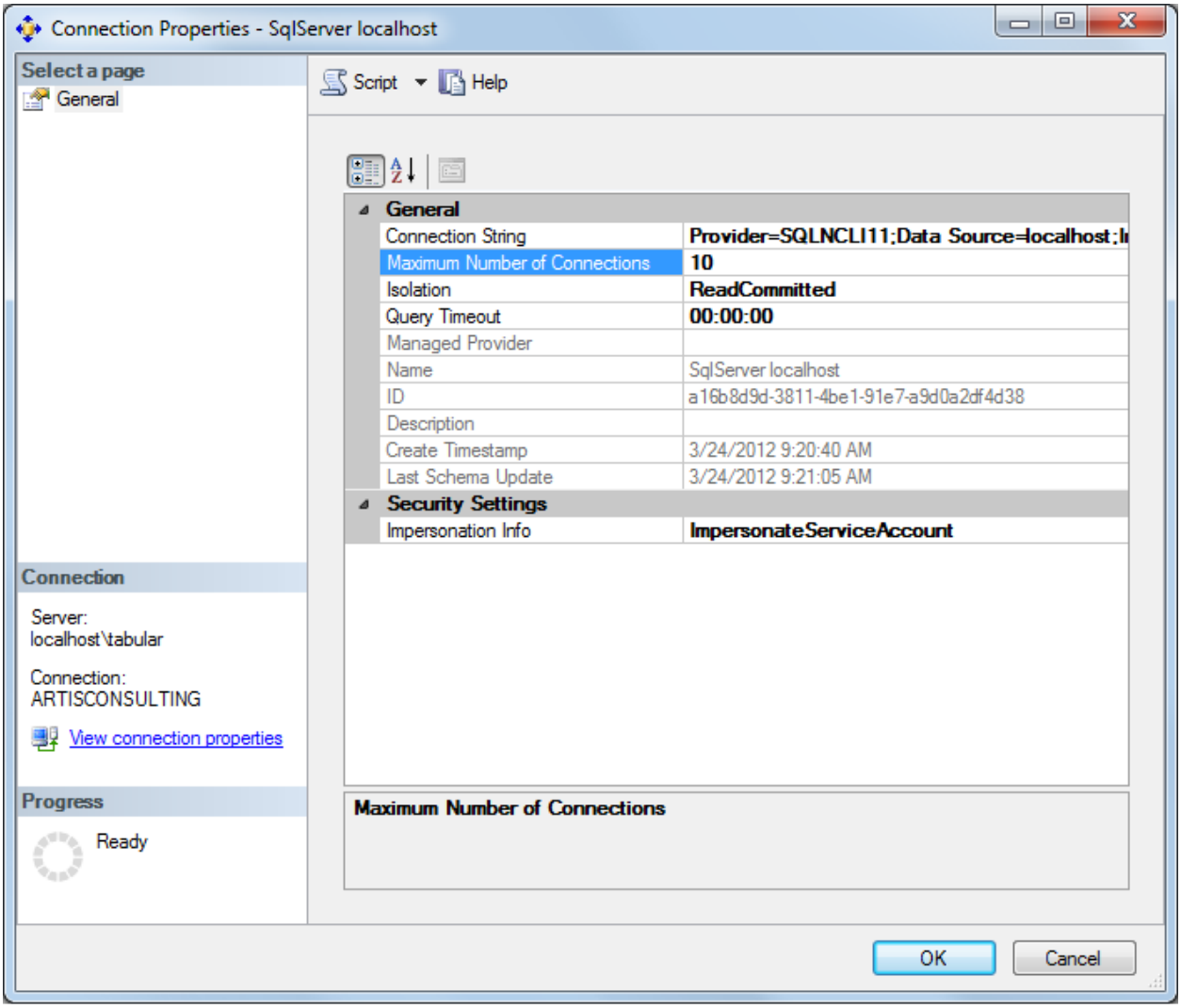


Figure 26. Connection Properties dialog box

If the model contains multiple data sources, or if multiple databases are being processed in parallel, the server setting **OLAP\Process\DatabaseConnectionPoolMax** may be a limiting factor. This setting has a default of 50 data source connections per Analysis Services instance. However, you should ensure that the relational data source can handle more than 50 parallel queries before considering changing this server setting.

After the connection is opened to the data source, the SQL query for a partition is executed. This query is executed with a 60 minute timeout setting by default (according to the **ExternalCommandTimeout** server setting which defaults to 3600 seconds). Consider increasing the value of **ExternalCommandTimeout** if it is expected that model processing could ever take more than an hour.

Performance of this phase of processing is highly dependent upon the performance of the SQL query. Optimization recommendations for SQL queries are made later under Optimizing the Relational Database Layer in this whitepaper.

Consider the downstream impact on the relational source during ProcessData. If fully processing a 300 million row table, all 300 million detailed rows will be returned from the relational source and transferred over the network. This operation can slow down performance of the source relational database and causing locking in the relational database.

On the other end of the spectrum, if the relational data source is sending rows to Analysis Services faster than it can compress them, Analysis Services will pause reading until compression catches up. This pipelining behavior is by design, and avoids the problem of too much memory being used by Analysis Services as rows accumulate.

Phase 2: Encoding

Once Analysis Services begins receiving rows, the encoding phase begins. Encoding refers to converting data into integers. These integers are internally referred to as the *DataID*. All column datatypes (even strings and dates and decimals) are converted into integers, because integers offer the best performance.

Analysis Services samples the values in each column and applies heuristics to decide on the encoding method for each column—either "value encoding" or "hash encoding". This sampling operation occurs only when a table is empty or during ProcessFull (since ProcessFull clears then reprocesses a table inside a transaction).

Value encoding

Value encoding uses the value itself (with or without some simple mathematical operation applied) as the DataID. For example, if the unique values in a column are 900, 901, and 902, Analysis Services may decide that it can subtract 900 from each value and fit the resulting values inside a two bit data type.

Value encoding is used whenever possible because it provides better query performance compared to hash encoding. The reason it provides superior performance is because computations in queries can operate directly on the DataID without a separate hash lookup step. For example, Analysis Services can answer queries that sum up values in a value-encoded column, by simply adding 900 to the DataID and proceeding to sum. Value encoding works well for dense values.

Hash encoding

Hash encoding creates a new, meaningless DataID for each distinct value in a column. Hash encoding is used for string columns and for any other columns where value encoding will be less efficient than hash encoding. For example, numeric columns with very sparse distribution will typically be hash encoded.

If a query references a hash-encoded column using a count formula, the DataID can be used. However, if a queries references a hash-encoded column using a sum formula (or any non-count aggregation), the actual value must be decompressed and retrieved before it can be summed. Because of this added step during queries and because of the added structure taking up RAM, hash encoding is less efficient than value encoding.

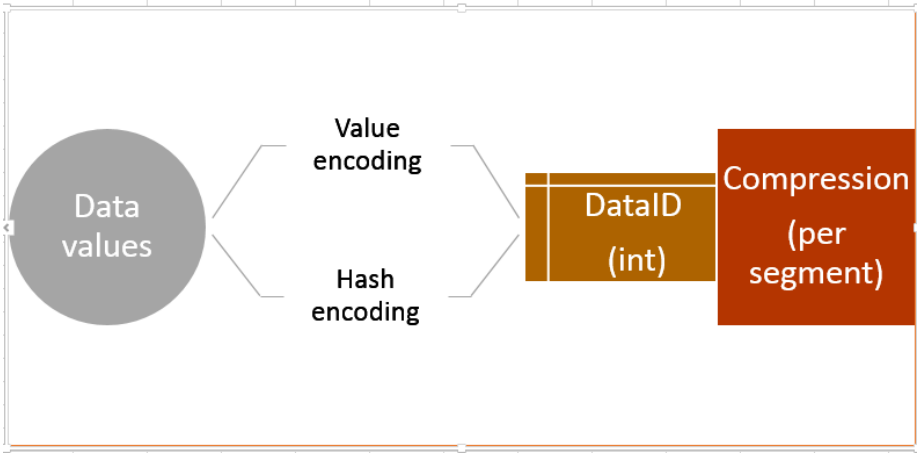


Figure 27.Encoding process

Detecting the encoding type for a column

To determine which encoding type is used for a particular column, you can run the following DMV query in SQL Server Management Studio, and look at the results in the COLUMN_ENCODING column. A value of 1 for a column means “hash encoding” and a value of 2 means “value encoding”. Note that one encoding type is applied per column per table, not per partition. Therefore you cannot distribute your data among partitions to optimize for different types of encoding.

```
SELECT *
FROM $SYSTEM.DISCOVER_STORAGE_TABLE_COLUMNS
WHERE COLUMN_TYPE = 'BASIC_DATA'
```

Monitor for re-encoding and consider avoiding by ordering data

One potentially expensive processing issue involves changing the encoding method that is applied to a column, after the column has been partially processed. The following explains how to detect re-encoding, determine how costly it is, and decide whether it is worth trying to avoid in the future.

Re-encoding may occur in a scenario like the following. In this data set, the PurchaseQuantity column in the first 60 million rows of a table contains only the values 1, 2, or 3. Analysis Services Tabular may choose to use *value encoding* for this column. However, if the next row contains a bogus sales transaction with a PurchaseQuantity of 99,999,999 then Analysis Services Tabular may need to re-encode this column using hash encoding. When this occurs, a Progress Report End trace event with an EventSubclass of “55 - Switching dictionary” will appear, as seen in the following figure.

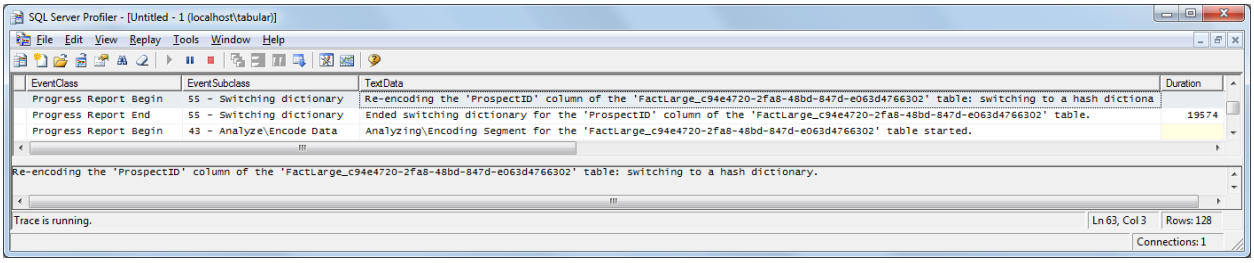


Figure 28. Event subclass showing encoding

In a sample test table with 60 million rows and 60 million distinct values in a column, switching to hash encoding took between 19 to 50 seconds depending on the test scenario. This time can be seen in the duration of the Profiler event above.

If processing is taking longer than expected, watch for the "Switching dictionary" trace event. If the duration is significant, then investigate the source data further.
If the table in question is partitioned, process one partition at a time, then determine which encoding method is used for a column by running the DMV query listed in the prior section.

The main reason that we want you to understand re-encoding is so that you can assess whether it is affecting the performance of your model processing. Changing your model or data to avoid re-encoding is difficult; do not attempt to do this unless you find that re-encoding repeatedly accounts for a significant amount of time.

How can you avoid re-encoding? The key is to make sure that a representative range of column values is seen in the first set of rows presented to a table. This could be accomplished in several ways:

- Ensuring that the query binding for the first partition returns a representative set of rows. For example, if the bogus sales transactions appear under a store called "For User Training," the query binding for the first partition could be restricted to this store.
- Create a fake first partition with representative data, processing all the partitions, then dropping the fake partition.
- Change the order of rows with an ORDER BY clause or a clustered index to sort based upon a different column.

However, typically the cost of ordering the rows outweighs the cost of re-encoding.

When adding a partition, be sure to ensure those rows are not also returned in the queries of subsequent partitions. If your table has a Row Identifier you will receive a duplicate key error. If your table does not have a Row Identifier, you will not receive an error and queries may double count those rows.

Do datatypes matter?

For numeric columns, it is recommended that columns containing decimals be changed to the Currency data type in the model—unless business requirements call for more than 4 decimal places. You can change the data type for a column in SSDT, by clicking the **Data Type** property for the column, the Properties pane, and changing the type to Currency. This change allows Analysis Services to treat the column as an integer, with an assumed 4 decimal places, and may make value encoding more likely.

Using a string data type in a column forces hash encoding.
If it is possible to change a string to a numeric data type without losing functionality, we recommend making the change.

Note that the **Data Type** property affects physical storage, whereas the **Data Format** property affects the formatting when the data is presented to the client. Thus, it is possible to store a number as Data Type=Currency and display it to users without a currency symbol such as a dollar sign ($) by setting the **Data Format** property to Decimal Number.

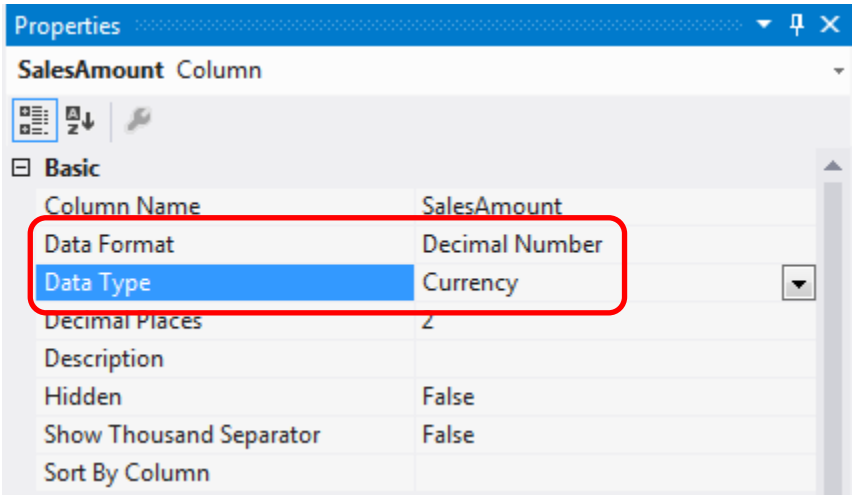


Figure 29. Determining or changing the data type

The exact data type coming from the source system matters very little except in how efficiently data is copied over the network from the source. The data type property of the column inside the Tabular models is what does matter. However, because Tabular models use columnar storage with dictionaries holding the distinct values, the actual number of distinct values in the column is most important factor. The absolute number of values determines the size of the dictionary and the amount of memory required to keep that data in memory. Query performance and memory usage also depend on the number of distinct values that must be handled. Therefore, reducing the number of distinct values through datatype changes is beneficial whenever business requirements allow. For more information on

reducing memory usage on specific datatypes, see Reducing Memory Consumption in High Cardinality Columns in section 5.

Phase 3: Compression

After a segment of data is read, the compression phase begins in parallel with the next segment of data being read and encoded.

Segment size

The notion of a column segment has already been explained, in Section 2. The size of each segment is generally determined by server settings:

- In Tabular models, the VertiPaq\DefaultSegmentRowCount setting defaults to 0, which means 8 million rows per segment.
- In PowerPivot, the default is 1 million rows, due to client machines typically having less memory available.

If you want to change this setting, see Section 5 for more details on how to modify server configuration settings. This DefaultSegmentRowCount setting is an instance-wide setting which applies to all databases and all tables in that Analysis Services instance. It is not possible to control the segment size of each table individually with this setting. However, partitions can be used to create smaller segments since partitions are made up of one or more segments.

Changing segment size

The value of this setting must be a power of 2; therefore, the default setting of 0 actually represents a value of 8388608, to Analysis Services.

Suppose you lower the setting to 1048576, 2097152, or 4194304 (1, 2, or 4 million)? As a result, the segments will require less memory during processing.

Suppose you raised the value, for example to 16777216 (16 million)? This change might provide better compression and faster queries. Queries might be faster both because of better compression and also because of fewer segments.

As discussed in the section on queries, scans of data utilize one processor core per segment. Therefore, to optimally tune a model for a particular server, there should be a rough correlation between the number of segments and the number of cores on the server. For example, in a 240 million row table, it is a bad idea to have 240 segments containing 1 million rows each since the processors will be doing short bursts of activity with lots of overhead in between compared to longer productive bursts of scanning on fewer larger partitions.

VertiPaq checks for query cancellation at the beginning of each segment scan. Therefore, increasing the segment size might make query cancellation noticeably slower, which could extend the blocking period during a processing commit.

Compression

Compression happens per segment. After enough data has been read and encoded, the segment is compressed. The following figure diagrams the sequence and parallelism that occurs.

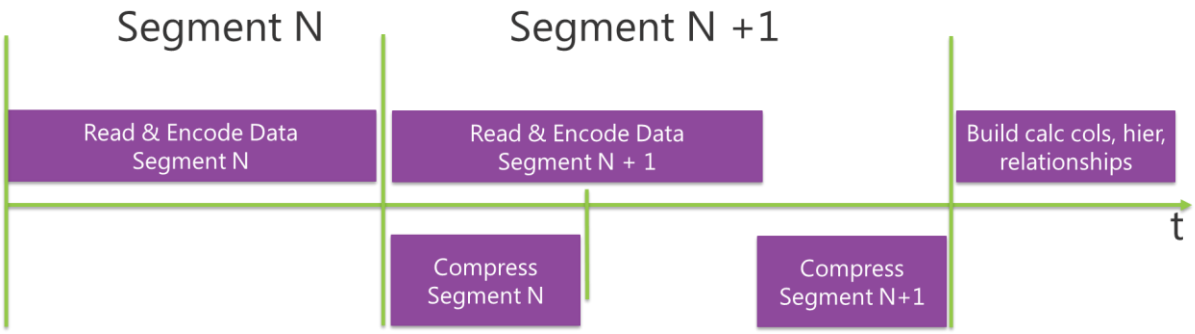


Figure 30. How encoding is interleaved with compressions

Therefore, the number of distinct values in each column that are present in each segment have an effect on the level of compression.

For example, it might be possible to improve compression by ordering the data coming from the source system such that each segment has the minimum number of distinct values. However, such ordering in the source is very expensive and is not worth the time saved in typical models. Also, sorting the incoming relational data may cause re-encoding to occur more frequently.

Compression timebox

The compression phase can be very CPU-intensive as the storage engine tries to compress the data as much as possible. The **VertiPaq\ProcessingTimeboxSecPerMRow** setting defaults to -1, which lets the server decide how much time it spends on compression. Currently, -1 allows up to 10 seconds per million rows for compression. For models with very wide tables (about 200 or more columns), you might consider raising the value of this setting. For example, by raising the value to 60, you would increase the maximum compression time per segment to 60 seconds. By setting the value to 0, you give the engine unlimited time for compression. For information on changing server configuration settings, see section 5.

If processing is completing well within the acceptable processing window, consider experimenting with the setting, **ProcessingTimeboxSecPerMRow**, to see if it increases compression, reduces memory required to hold the model in memory, and increases query performance.

The following example shows the change in processing time and compression on a 4 billion row table.

- With the default setting, the table compressed to 41GB with the default settings
- When the value of **ProcessingTimeboxSecPerMRow** was set to 0 and the value of **DefaultSegmentRowCount** was increased to 16 million rows, the table compressed to 25GB.
- With the default, processing completed in 5.5 hours. With the changes, processing completed in 21 hours (results for one test run).

Not every possible combination of settings was run, but the experiment indicates that these settings can make a significant impact on processing duration, memory usage, and compression.

Table 5. Processing time and timeboxing

Working set max (GB)	Duration (hours)	Folder size (GB)	VertiPaq \ DefaultSegmentRowCount	VertiPaq \ ProcessingTimeboxSecPerMRow
43.7	5:28:57	41.3	0 (2^23 = 8388608)	-1 (10 seconds per million rows)
44.4	6:43:25	41.1	16,777,216 (2^24)	-1 (10 seconds per million rows)
45.2	8:44:12	40.8	33,554,432 (2^25)	-1 (10 seconds per million rows)
45.8	13:42:09	40.5	67,108,864 (2^26)	-1 (10 seconds per million rows)
47.4	15:26:43	40.3	134,217,728 (2^27)	-1 (10 seconds per million rows)
29.0	20:58:17	24.7	16,777,216 (2^24)	0 (as long as necessary)

If you wanted to reduce processing time at the expense of worse compression, you might consider lowering the value of **ProcessingTimeboxSecPerMRow** to between 2 and 5. However, this change might have little impact on the overall processing time if reading the data from the relational database is the bottleneck.

Monitoring timeboxing

A Profiler trace will report a Progress Report End with EventSubclass of “53 – VertiPaq” when timeboxing is occurring. If timeboxing does not occur, a message similar to the following will be seen and the Duration for the event reports how many milliseconds were spent on compression.

Compression for the '<TableID>' table completed in '10758' steps with '3209' clusters computed in total. Execution hasn't been timeboxed.

If timeboxing does occur, a message similar to the following will be seen.

Compression for the '<TableID>' table completed in '40256' steps with '29502' clusters computed in total. Execution has been timeboxed.

Another way of detecting timeboxing after the fact is by using a DMV. If the VERTIPAQ_STATE column in the following DMV says TIMEBOXED, then that particular segment was timeboxed:

```
SELECT *
FROM $SYSTEM.DISCOVER_STORAGE_TABLE_COLUMN_SEGMENTS
```

Compression results

The details of the compression algorithms used by Tabular models are proprietary; however, it does use a greedy algorithm which should achieve most of the gains in the beginning. Compression varies greatly depending on data types, the density of the source data, and the settings mentioned above. The following chart shows how much compression a Tabular model achieved compared to the uncompressed size of data in the relational source database.

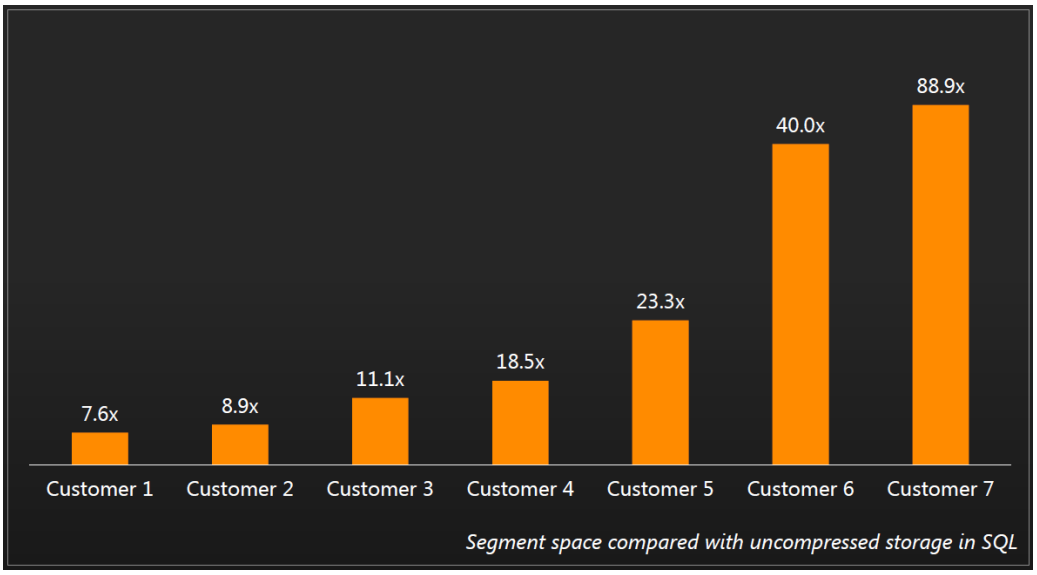


Figure 31. Real world compression results

If you would like to verify this with your own data, you can measure the size of data in SQL Server (or other data store) and then compare it with the tabular model.

- The size of data in a SQL Server relational database can be measured by looking at the size of the heap or clustered index (exclude nonclustered indexes) on a version of the table with no row, page, or columnstore index compression.
- For the many other supported data sources (http://technet.microsoft.com/en-us/library/gg492165.aspx), your database experts can guide you on similar measurements.

- A model administrator can determine the compressed size in a Tabular model by following these instructions.

We strongly recommend that, at the beginning of a project, you build a small prototype model with representative data, in order to determine approximate levels of compression that can be achieved on the dataset in question. Because Tabular models use in-memory technology and proprietary compression algorithms, the creation of a small prototype at the very beginning of a project will give you a much better estimate of server memory requirements.

In the absence of such a prototype, 10x compression is a good rule of thumb assumption for initial planning, though there is much more to memory capacity planning than just the raw database size. Capacity planning and hardware sizing is discussed more fully in the Hardware Sizing a Tabular Solution whitepaper and will not be repeated here.

Segment size of first segment

The first partition processed in a table can be processed according to a special rule that differs slightly from the above explanation. We will explain this special case so that you can understand results that vary from the general rules for segment size.

1. Analysis Services will continue reading the first rows it sees until 16 million rows (or twice DefaultSegmentRowCount) are accumulated. For example, if 14 million rows are received, then the first segment will contain 14 million rows, making that segment larger than normal.
2. If more than 16 million rows are received, then it creates one segment out of the first 8 million rows and a second segment out of the next 8 million rows.

This optimization is intended to avoid the expense of building two segments unless multiple segments are really needed. This optimization also requires extra memory, because as the engine accumulates 16 million uncompressed rows and then splits and compresses the first two segments in parallel.

The following diagram illustrates this process:

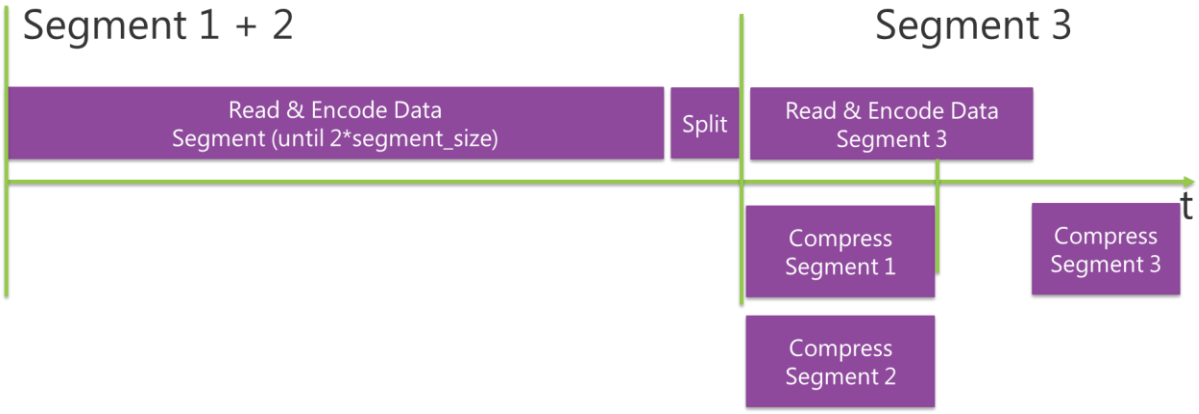


Figure 32. Special case of the large first segment

This special case is particularly relevant to ProcessAdd. For example, assume that a segment contains 7 million rows. If another 7 million rows are appended during ProcessAdd, you will end up with a 14 million row segment, even if there are other processed partitions. However, once the segment hits 16 million rows, it will be split into two segments of 8 million rows.

Phase 4: ProcessRecalc

A ProcessRecalc can be included in a transaction either through the implicit recalculation performed as part of ProcessFull or ProcessAdd, or through an explicit ProcessRecalc command.

The actual work of ProcessRecalc begins when all the segments have finished compressing. This phase does the work described in the following sections:

Building calculated columns

As described above, ProcessRecalc analyzes dependencies between calculated columns and calculates the proper sequence of processing with as much parallelism as possible. The calculations are evaluated at processing time, not at query time. Calculated columns are stored just like any other column.

Calculated columns are stored just like any other column, in segments, except that calculated columns are built after compression completes, which means that calculated columns are not compressed. Therefore, if it is possible to perform the same calculation in the SQL query inexpensively, you should generate the value by using the SQL query, since doing so leads to better compression. That is to say, data values coming from the data source are compressed, whereas data values generated in the form of calculated columns are not compressed.

However, there are many calculated columns using complex DAX expressions that evaluate multiple rows across multiple tables, which would require many joins and would be much more expensive to calculate in SQL. So there are many scenarios where calculated columns are preferred both for processing performance and for simplicity. In your model design, you will have to balance formula requirements such as these against the reduced memory usage and improved query performance when calculated columns are converted to data values that can be compressed.

For additional tips on how to optimize DAX formulas to improve query performance, see section 3.
For example, you can improve the speed of a slow DAX expression by converting expensive portions of the calculation into calculated columns which can be evaluated once at processing time instead of query time.

Building hierarchies

The ProcessRecalc phase also builds hierarchies. Hierarchies are useful for optimizing MDX functions such as Ancestor and NextMember. In SQL Server 2012 Tabular Models, every column in every table (whether the table is a “fact table” or a “dimension table”) has an attribute hierarchy built by the

engine. And, in keeping with a design principle for Tabular models which calls for simplicity and for fewer knobs to adjust, there is currently no way to disable building attribute hierarchies on a column.

This makes the model more flexible, as it allows any column to be used in aggregation or as a slicer. However, building hierarchies on high cardinality columns can consume significant memory, time, and disk space. Removing unnecessary columns or reducing their cardinality is the only way to optimize hierarchy building.

ProcessRecalc also builds user-defined hierarchies. User-defined hierarchies are the hierarchies that are added in the model designer, to provide navigation paths and combine multiple columns into levels.

The cost of building a user-defined hierarchy during processing depends on the cardinality of the columns used in the hierarchy; small hierarchies are fairly cheap to build, but large hierarchies on top of columns with millions of distinct values can be expensive to build.

Building relationships

The ProcessRecalc phase also builds relationship structures that allow one table to quickly lookup the related row in another table. Relationships are built for both active and inactive relationships.

As Analysis Services is analyzing the dependencies among calculated columns, it is also analyzing the dependencies on relationships and choosing to build the relationships needed for calculated columns first.

Since relationships are built as ancillary structures, new relationships can be added to the model without reloading the tables from the relational database. As discussed more fully in the section comparing Tabular to Multidimensional processing, this behavior differs from Multidimensional models where the internal storage of measure group data uses the DataIDs of the related dimensions. Adding a dimension relationship will unprocess the measure group in Multidimensional models. Furthermore, fully reprocessing a dimension will reassign DataIDs, and in Multidimensional models, this operation will have the side effect of unprocessing any related measure groups. Not so in Tabular models. New relationships can be added and dimension tables can be fully processed without causing the fact tables to become unprocessed. A simple ProcessRecalc must be run to rebuild the relationships after either of these changes take place.

Nuances of Processing via the UI

When using SQL Server Management Studio to process tables, the model administrator can process tables and receive progress reports in the UI, or the administrator can click the **Script** button to generate an XMLA script which can be run on demand or scheduled with a SQL Agent job. Each option behaves differently.

Processing Tables via the UI

For example, clicking **OK** in the following **Process Tables** dialog will run one ProcessFull batch per table in serial, which will commit results after each table and will repeat work as mentioned above. However,

clicking **Script** will generate one transactional batch which processes all tables in serial but commits changes to all tables in one commit.

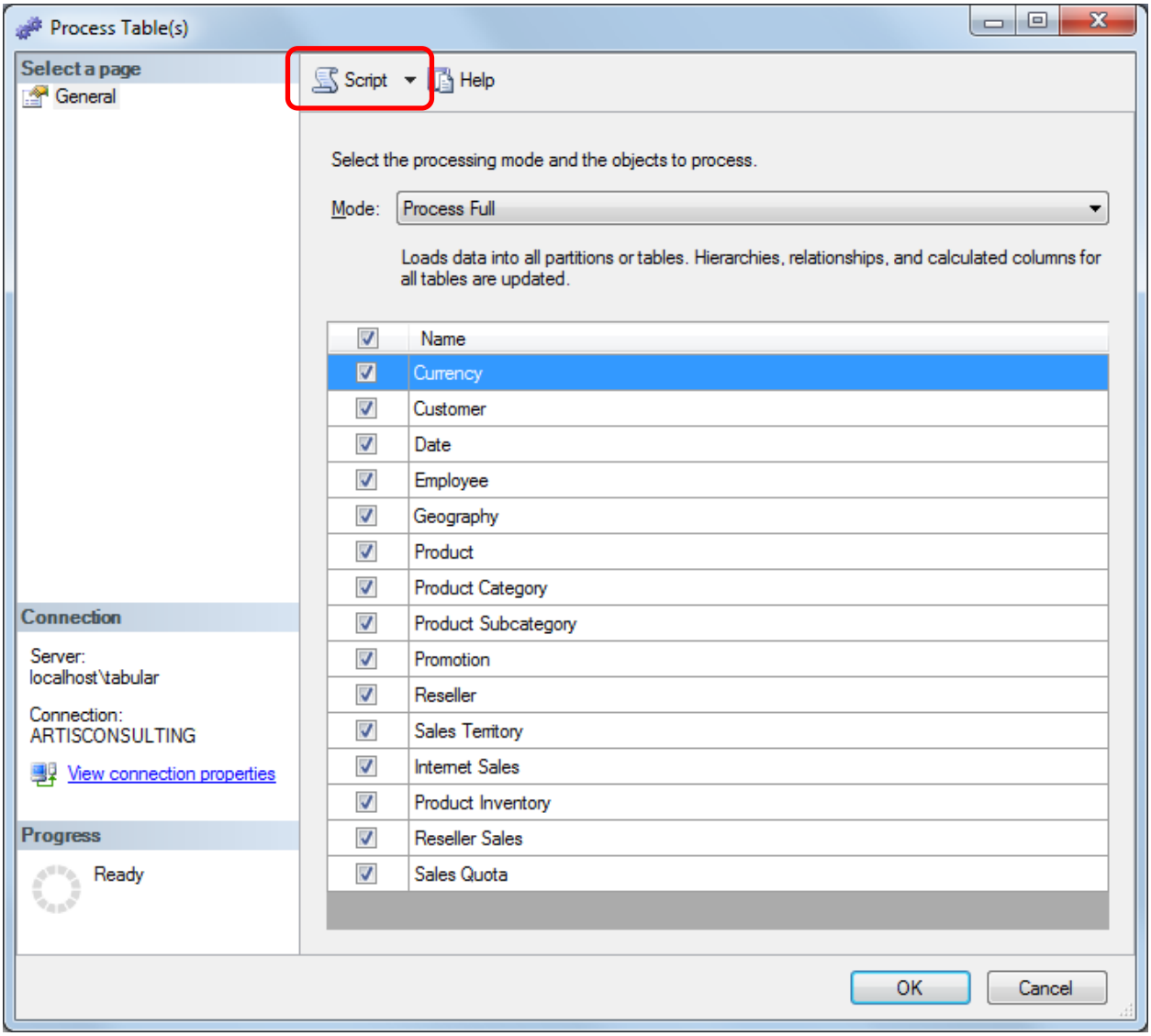


Figure 33. Creating a script to process tables

Example of ProcessData Plus ProcessRecalc

In large models, if you are processing multiple tables and have the memory available to do parallel processing to reduce processing time, it is highly recommended that you use parallel processing.

How to do parallel processing of partitions

1. In Management Studio, right click on a table, and choose **Process.**
2. Change **Mode** to **Process Data**.
3. Select the tables you wish to process.
4. Click the **Script** button.

5. Open the script file, and add Parallel tags (<Parallel></Parallel>) around all processing commands, as shown in the following example.
6. Finally, add a ProcessRecalc command to the end.

The final XMLA command should resemble the following example:

```
<Batch xmlns='http://schemas.microsoft.com/analysisservices/2003/engine' Transaction='true'>
 <Parallel>
  <Process>
   <Type>ProcessData</Type>
   <Object>
    <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
    <DimensionID>Customer_08b41bba-8733-438e-a584-9894998b2f69</DimensionID>
   </Object>
  </Process>
  <Process>
   <Type>ProcessData</Type>
   <Object>
    <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
    <DimensionID>Internet Sales_fdac9a13-4019-4773-b193-7cca3a4883eb</DimensionID>
   </Object>
  </Process>
  <Process>
   <Type>ProcessRecalc</Type>
   <Object>
    <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
   </Object>
  </Process>
 </Parallel>
</Batch>
```

Processing Partitions via the UI

The preceding example discussed how the Management Studio UI processes multiple tables. However, when processing multiple partitions via the UI, regardless of whether you click OK in the dialog box or click the **Script** button, the same processing batch is generated. Both methods will process the checked partitions in one transaction. However, it is highly recommended that you script the operation and change it to ProcessData followed with one ProcessRecalc to avoid repeating work after each partition.

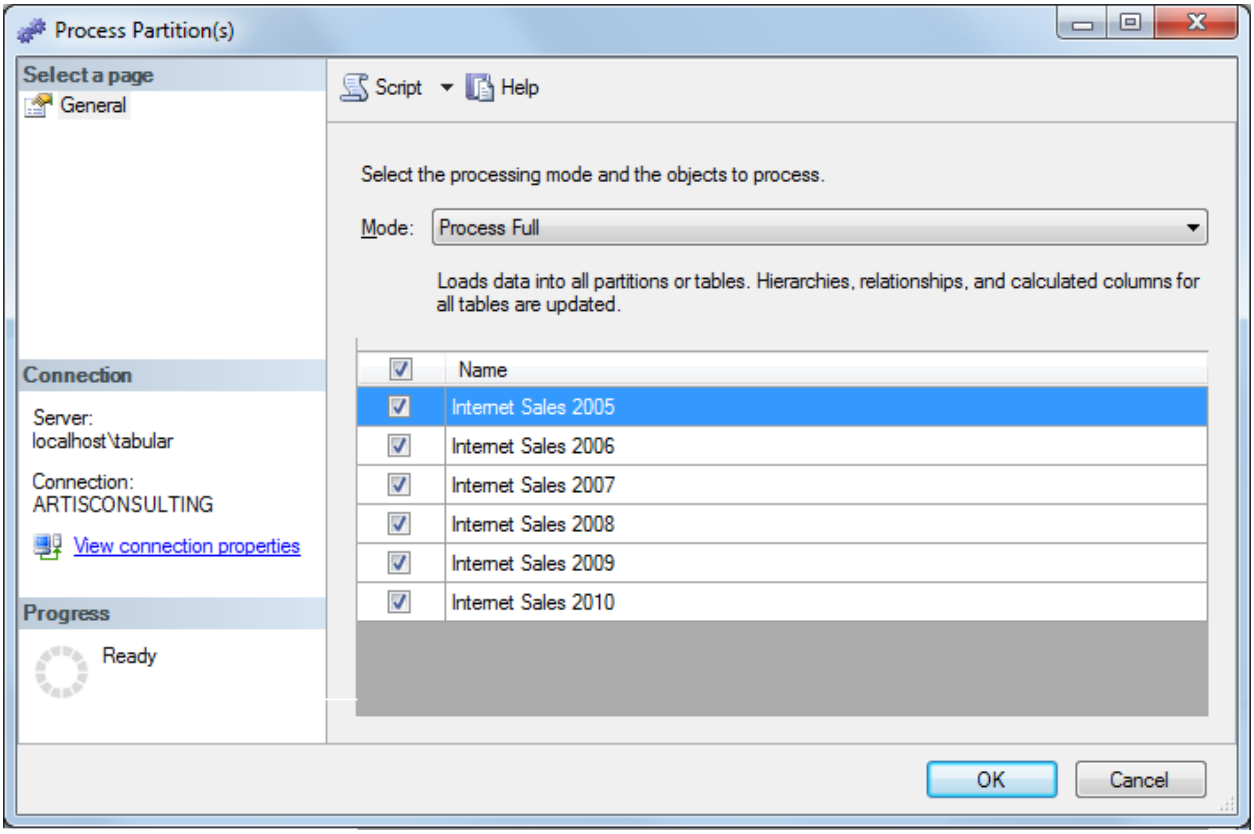


Figure 34. Process Partitions dialog box

Common Processing Methodologies

The seven processing commands above are the tools in the tool bag of the model administrator. Choosing the best processing methodology for each scenario depends on the business requirements, available server resources, and level of complexity that is acceptable. The following sections describe common processing methodologies. They are listed in order of increasing complexity.

ProcessFull to refresh the entire database

If the size of the data is small enough, there is enough memory on the server, and it completes fast enough to meet requirements, ProcessFull on the database is the most straightforward way to refresh a model. As discussed above, it causes no model downtime since it leaves the old data queryable until the processing transaction commits. Try this option first to see if it meets the business needs. If it does not, consider the following strategies.

ProcessData to touch the fewest partitions possible, then ProcessRecalc

If ProcessFull on the whole database is not completing within the desired refresh window, consider whether the largest tables can be partitioned so that only a small subset of the partitions every night must be processed. For instance, if the partition scheme is by month, it may be possible to just process the current month's partition nightly. The goal of this processing scheme is to determine which rows are

new, changed, or deleted in the relational database and to only process partitions containing those rows.

These ProcessData commands and one ProcessRecalc command should be placed inside one batch marked with Transaction="true" (or not marked with a Transaction attribute since Transaction=true is the default). This will cause the processing to be done in a transaction leaving the old data queryable until the entire processing batch commits. Optimally, to increase parallelism, put all the Process commands inside one Parallel tag:

```
<Batch xmlns="http://schemas.microsoft.com/analysisservices/2003/engine" Transaction="true">
 <Parallel>

  <!--process the whole Customer table-->
  <Process>
   <Type>ProcessData</Type>
   <Object>
    <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
    <DimensionID>Customer_08b41bba-8733-438e-a584-9894998b2f69</DimensionID>
   </Object>
  </Process>

  <!--process the one partition in the Internet Sales table-->
  <Process>
   <Type>ProcessData</Type>
   <Object>
    <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
    <CubeID>Model</CubeID>
    <MeasureGroupID>Internet Sales_fdac9a13-4019-4773-b193-7cca3a4883eb</MeasureGroupID>
    <PartitionID>Internet Sales_0231d131-30b8-4d3c-90cd-2d4858e7b9bf</PartitionID>
   </Object>
  </Process>

  <!--ProcessRecalc once the above is complete-->
  <Process>
   <Type>ProcessRecalc</Type>
   <Object>
    <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
   </Object>
  </Process>
 </Parallel>
</Batch>
```

Though this processing methodology can save tremendous amounts of time in some scenarios, it does raise the complexity level. If the set of partitions that need to be processed can change over time, the model developer will need to write some code to automate this processing scheme. You might need to use an ETL auditing framework that marks rows which have changed, or you might need to periodically perform a ProcessFull to capture the occasional changes outside of the designated partitions.

ProcessAdd to add just the new rows

If the processing methodology described above takes too long to complete, or if some tables receive only inserts (not updates or deletes) in the relational database, consider using ProcessAdd instead of ProcessData. ProcessAdd defines an out-of-line query binding which returns only new rows from the relational database which are not already loaded into the partition. This is typically the most efficient way to load new data into a model assuming the underlying source system can return the new rows

more efficiently than returning all rows. Implementation details for this approach are described in Incremental Processing in Tabular Using Process Add, by Marco Russo.

Another similar methodology is to create a new partition containing a query binding which returns new data, ProcessData that new partition, merge that new partition with an existing partition, then perform ProcessRecalc.

This processing methodology requires even more complexity as it requires code to automate the above steps. It is recommended that before investing in ProcessAdd, first consider optimizing the SQL query and relational database layer behind the partitions which are processing too slowly for ProcessData.

Processing in multiple transactions to minimize memory requirements

If the processing methodologies above result in out-of-memory errors, consider processing the model in multiple transactions. This can be accomplished in multiple batches run in serial. Or it can be accomplished in one batch by marking it Transaction=false. For example, the following batch will process the Customer table, commit that operation, then process the Internet Sales table and commit that operation:

```
<Batch xmlns='http://schemas.microsoft.com/analysisservices/2003/engine' Transaction='false'>
 <Process>
  <Type>ProcessFull</Type>
  <Object>
   <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
   <DimensionID>Customer_08b41bba-8733-438e-a584-9894998b2f69</DimensionID>
  </Object>
 </Process>
 <Process>
  <Type>ProcessFull</Type>
  <Object>
   <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
   <DimensionID>Internet Sales_fdac9a13-4019-4773-b193-7cca3a4883eb</DimensionID>
  </Object>
 </Process>
</Batch>
```

Multiple process commands inside a batch with Transaction=false will run in serial. This reduces the memory requirements by reducing parallelism. It also reduces memory by using smaller transactions which mean less memory is required to hold the shadow copy of the old data that's currently being reprocessed. Finally, it reduces memory usage because ProcessFull on a table does a ProcessRecalc only on dependent objects, not on the whole model.

Note that multiple commits means that if users are querying this model while it is processing, they may see inconsistent or incomplete data.

Unfortunately, this processing scheme does some work multiple times. For example, if the Internet Sales table relates to the Customer table, that relationship will be rebuilt two times by the implicit ProcessRecalc that is part of the ProcessFull on each table. Furthermore, calculated columns that depend on multiple tables may be calculated multiple times. If this is wasting significant time in your

model, and no users will be querying the model during processing, consider using ProcessData and saving ProcessRecalc until the end, as in the following example, so that work is done only once. Just be aware that once the first ProcessData transaction is committed, the model will not be queryable by users until the ProcessRecalc step at the end is complete.

```
<Batch xmlns='http://schemas.microsoft.com/analysisservices/2003/engine' Transaction='false'>
 <Process>
  <Type>ProcessData</Type>
  <Object>
   <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
   <DimensionID>Customer_08b41bba-8733-438e-a584-9894998b2f69</DimensionID>
  </Object>
 </Process>
 <Process>
  <Type>ProcessData</Type>
  <Object>
   <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
   <DimensionID>Internet Sales_fdac9a13-4019-4773-b193-7cca3a4883eb</DimensionID>
  </Object>
 </Process>
 <Process>
  <Type>ProcessRecalc</Type>
  <Object>
   <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
  </Object>
 </Process>
</Batch>
```

While transactional processing is generally recommended, on servers where there is not enough memory to process the entire model in a transaction, the above workarounds can help processing succeed.

Processing Problems and Tips

This section lists some additional issues to watch for, as well as some tips for monitoring what is happening during processing. We also explain how processing is different in Tabular and Multidimensional models.

Locking and Blocking during the Commit of Processing Transactions

It is important to understand how locking affects processing, for two reasons.

- First, two processing transactions on the same database cannot run in parallel. If two separate processing batches are run at the same time, the first will run, then the second will wait to run until the first completes. To resolve this situation, combine all processing commands for the same database into one processing batch in the same Parallel tag. See the following example.
- Second, during the commit of a processing transaction, Analysis Services requires an exclusive lock on the database. A long-running query can cause the processing commit to wait up to 30 seconds (according to the **ForceCommitTimeout** setting) before the query is signaled to cancel. Typically, the long-running query is cancelled quickly after it is signaled to cancel, however in

some scenarios the query doesn't cancel immediately, which prolongs the blocking period. During this blocking period, all other connections and queries will be blocked, causing the server to appear hung until the long-running query cancels and processing commits.

So you want to avoid a situation where processing causes queries to be cancelled or blocked. There are a couple different ways to avoid long-running queries (which potentially intrude into the processing window):

- First, ensure that queries and calculations are efficiently written.
- If optimization is not possible, consider processing the model during periods of the day when users are not querying the model.
- If this is not possible, consider one of the scale-out approaches (discussed in the scale-out section) which reduces or avoids locking.

On the other hand, if you are processing the model frequently (such as every 5 minutes), it may be preferable to let processing commits time out, rather than cause the long-running queries to time out. In this case, set the **CommitTimeout** setting to 30 (seconds) and set **ForceCommitTimeout** to 0. This configuration is unusual, because long-running queries might prevent processing from ever committing, but in certain circumstances it is desirable.

Detecting, avoiding, and troubleshooting locking has been explained in great detail in the SQL Server 2008 R2 Analysis Services Operations Guide in section 7.4. Although the guide was written about Multidimensional models, everything said applies to Tabular models since the locking architecture is identical.

Profiler Trace Events to Watch during Processing

Watching in Profiler, you can observe pairs of the following **Progress Report Begin** and **Progress Report End** events. The Progress Report End event contains a **Duration** column. Focus on the largest Duration events.

Table 6. List of related trace events

EventSubclass	Progress Report Begin and End TextData	Explanation
1 – Process	Processing of the 'YourPartitionName' partition has started. Finished processing the 'YourPartitionName' partition.	The Duration on the Progress Report End event tells you how long the ProcessData phase took. The ProcessRecalc phase begins later.
25 – ExecuteSQL	select * from YourTableName	The SQL query run for that partition
17 – ReadData	Reading data for the 'YourTableName_GUID' table started. Finished reading data for the 'YourTableName_GUID' table.	The Progress Report Begin fires when the first row is read from the relational data source. If the CurrentTime on this event is much delayed from when partition processing started, the SQL query

		was slow to return rows. If the duration on the Progress Report End event is longer than expected, optimize the SQL query. The Progress Report Current displays the number of rows returned in the IntegerData column.
53 – VertiPaq	Compression started for the 'YourTableName_GUID' table. Compression for the 'YourTableName_GUID' table completed in '141509' steps with '60108' clusters computed in total. Execution has been timeboxed.	Despite the text mentioning that compression is starting for the table, this pair of events is seen once per segment. See above for explanation of compression and timeboxing.
44 – Compress Segment	Compressing segment 0 of column 'YourColumnName' for the 'YourTableName_GUID' table started. Finished compressing segment 0 of column 'YourColumnName' for the 'YourTableName_GUID' table	Detecting which columns take a long time to compress can be seen by monitoring the Duration column on the Progress Report End event.
54 – Hierarchy processing	Started processing the 'YourColumnName' hierarchy. Finished processing the 'YourColumnName' hierarchy.	A long Duration on the Progress Report End indicates a very high cardinality column for which building the hierarchy is expensive. Consider removing the column or reducing its cardinality.
1 – Process	Started processing the 'YourCalculatedColumn' calculated column. Finished processing the 'YourCalculatedColumn' calculated column.	A long Duration on the Progress Report End event indicates a slow and complex DAX calculation in the calculated column.
46 – Relationship Build Prepare	Building and preparing relationship for the 'YourTableName_GUID' table started.	A long Duration on the Progress Report End event indicates this relationship was expensive to build. Unfortunately, this event does not identify which table is on the other end of the relationship or which columns are related.
6 – Commit		A long duration on the Progress Report End event may indicate a longrunning query is blocking the commit, or it may indicate an I/O bottleneck on committing a very

		large processing operation.

Server Resource Usage during Processing

Now that the steps taken during processing have been described, it is possible to discuss how these processing steps utilize network, memory, processor, and disk. The following diagram shows processing a single 24 million row table with one partition. The purple overlays describe what phase is happening at that moment in time.

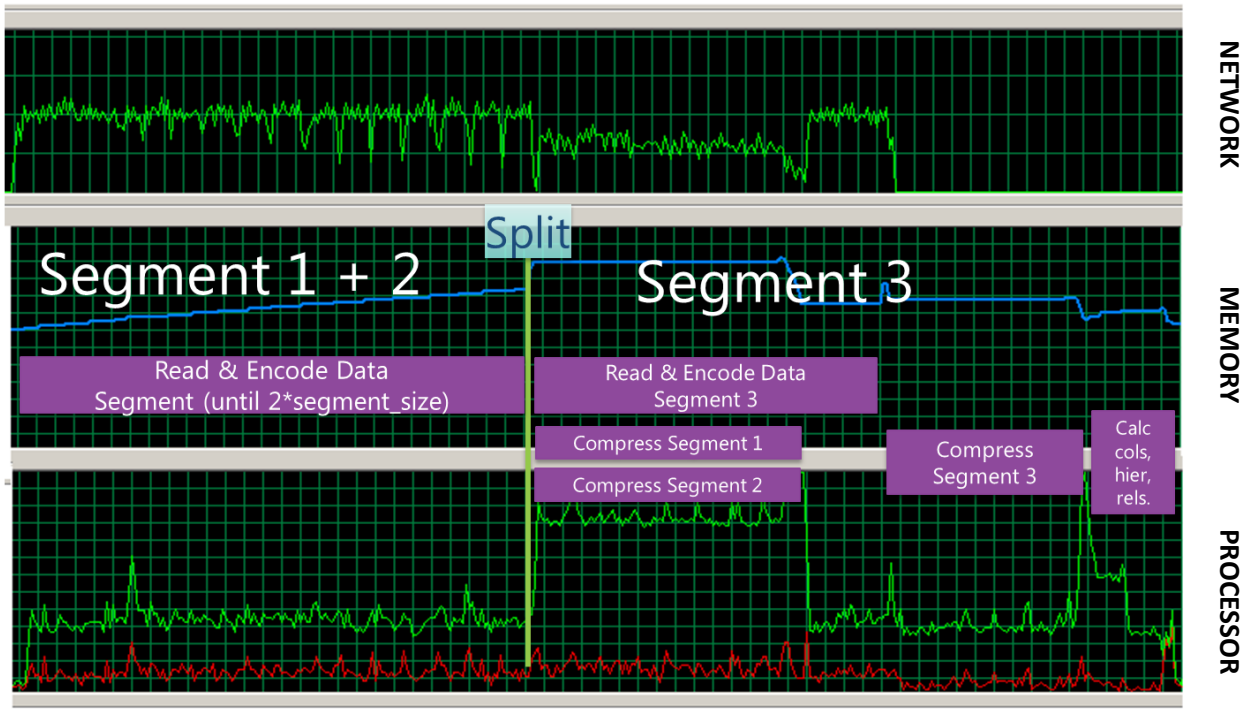


Figure 35. Processing memory and CPU usage

The line chart at the top with the single green line represents **network usage**. In this scenario, the SQL server was located on a separate server from Tabular. Therefore, network resources are used until all 24 million rows are read and encoded.

The second line chart represents **memory usage**. Notice that memory usage steadily grows as 16 million rows of uncompressed data are read and encoded. Then a spike occurs as the 16 million rows are split into two segments and compressed. Once segment 1 and 2 are finished compressing, memory usage drops slightly. If the table were hundreds of millions of rows, memory usage would continue to grow gradually depending upon how well the data compresses and how much repetition of column values is seen. Finally, the ProcessRecalc phase begins and memory usage will be highly dependent on the complexity of calculated columns and on the size of data.

The third line chart represents **processor usage**. The server in this test is a four core machine. While it is reading and encoding the first 16 million rows, approximately 1 core (25% CPU) is used. Then reading

and encoding of segment 3 begins in parallel with segment 1 and 2 being compressed. During this special case phase, three cores are used. Beyond 24 million rows, if compression is completing faster than the next segment of 8 million rows arrives, then typically 1.5 to 2 cores will be used until all segments are compressed. However, if rows are arriving faster than they can be compressed, more than 2 cores can be used. Once all segments are compressed, the ProcessRecalc phase begins. Depending on the number, dependencies, and complexity of calculated columns, a spike of CPU can been seen until those calculated columns, hierarchies, and relationship structures are built.

Disk I/O is not pictured. All of the work above is done in memory until the transaction commits. At that point, disk I/O occurs to write these structures to disk and delete the outdated structures. Once the commit completes, the new data can be queried by users. Typically processing is not extremely intensive on the disk storage of the Tabular server due to the efficient compression being used. However, in real-time scenarios where every second counts, optimizing I/O by ensuring the DataDir for the instance is located on fast drives such as SSD or multiple spindles may be required. I/O optimization is the exception rather than the norm for Tabular models, compared to Multidimensional models where I/O performance is important on large models.

The above pattern pictures server resources used during processing one table with one partition. If multiple tables are processed in parallel, imagine overlaying multiple copies of the chart to represent the compounding of resource usage due to parallelism. If users are simultaneously querying the old data in the model, overlay the resources used in those query patterns. If optimizing processing performance or choosing hardware, spec out servers with ample memory and processor resources. Hardware selection has been fully discussed in the Hardware Sizing a Tabular Solution whitepaper and will not be repeated here.

Comparing Tabular and Multidimensional Processing

For those Analysis Services administrators experienced with Multidimensional models, comparing Multidimensional to Tabular processing will be helpful.

No parallel processing of partitions in one table

In a Multidimensional model you can reduce processing time by processing partitions within a fact table (a measure group, in Multidimensional terminology) in parallel. However, in Tabular models all partitions within a single table are processed serially. In Tabular models it is possible to process multiple tables in parallel inside the same transaction, thus reducing processing time.

Each table is independent

In Multidimensional models, unprocessing a dimension will unprocess all related measure groups. For example, in a large Multidimensional cube, if the model developer needs to change the key structure of a dimension attribute, deploying this change might cause billion-row fact tables to need reprocessing.

In Tabular models, each table is independent. Altering the structure of one table, whether it is a fact or dimension table, will not unprocess related tables. Only the relationships and any calculated columns dependent on the changed table will need to be rebuilt.

Furthermore, in real-time refresh scenarios, the bottleneck of Multidimensional models is dimension processing, because if any data relationships change in the dimension, indexes and aggregations on measure group partitions also need to be refreshed in order to stay up-to-date.

Since the columnar storage and in-memory scanning of data is so fast, even on billions of rows, Tabular models do not require aggregations to be built during processing time. In Tabular models, relationships between tables are typically less expensive to rebuild than indexes and aggregations in Multidimensional models.

Partitioning dimension tables

In Multidimensional models, only measure groups (i.e. fact tables) can be partitioned. Dimensions cannot be partitioned.

However, in Tabular models, any table can be partitioned. Any table can have measures that can be aggregated and "dimension attributes" that can slice or filter content. This feature provides much flexibility in incrementally processing "dimension" tables.

Table query bindings are run without alterations

In Multidimensional models, Analysis Services alters the SQL query in various ways. During dimension processing, multiple SELECT DISTINCT queries are run per dimension, pushing the expensive work of building the distinct list of members into the relational engine. During measure group partition processing, the query is wrapped in a subselect and only the necessary columns are actually returned from the relational source.

Because Multidimensional models alter the query and use subqueries, certain T-SQL features such as ORDER BY, common table expressions (CTEs), and stored procedures cannot be used in the query binding.

In Tabular models running in the in-memory mode (not in DirectQuery mode), the query binding for each table is run unaltered. This means that it is possible to use T-SQL features such as ORDER BY, CTEs, and stored procedures. But this also means that Analysis Services does not alter the query to ensure only the necessary columns are returned. The burden of removing unnecessary columns from the query rests on the shoulders of the model developer. If unnecessary columns are returned from the query behind a table, processing will be less efficient due to networking and memory overhead.

In Tabular models, instead of deleting a column from the model, remove the column from the SQL query in the Table Properties dialog as this causes the column to be removed from both the SQL query results and the model table.

Handling of relational integrity violations

In Multidimensional models, there are many options for handling fact rows that reference a dimension member that does not exist. These fact rows can be skipped, logged, assigned to an UnknownMember,

or cause processing to fail. Some of these options such as logging can cause processing performance to degrade.

In Tabular models, the UnknownMember is hidden in every table until another table's foreign key refers to a row that does not exist. At that point, the UnknownMember (or blank row) is made visible in the table and the related row references it. This handling of relational integrity violations has little impact on performance since the individual problems are not logged. Tables where the blank row is visible can be detected by running the following DMV.

```
SELECT *
FROM $SYSTEM.DISCOVER_STORAGE_TABLES
WHERE RIVIOLATION_COUNT > 0
```

Then the problem tables can be queried to determine which rows are violating relational integrity. For example, some Product rows reference a ProductSubcategoryKey which does not exist:

```
EVALUATE(
 FILTER(
  'Product'
  ,ISBLANK(RELATED('Product Subcategory'[ProductSubcategoryKey]))
 )
)
```

Optimizing Processing

The design principles of Tabular models prioritize query performance over processing performance. Consequently, time and resources are spent during processing to achieve the highest levels of compression and query performance possible. Depending on the size of the relational data, expect that processing could take many hours. However, if processing is not fitting within the refresh window or if processing is taking too much memory, the following advice can help.

The techniques that you can use include:

- Optimizing the relational database layer.
- Managing memory usage during processing.
- Reducing the overall time spent on processing.
- Improving compression.
- Optimizing the Analysis Services server settings.

Optimizing the Relational Database Layer

Optimal relational database design and performance is required to get optimal processing performance in Analysis Services. Although it is not always possible to control or optimize the design of the data sources or the network layer, we have provided some guidance about how you might tweak these external factors to improve processing performance.

In general, these optimizations might include:

- Removing unused columns.
- Denormalizing your schema.
- Inspecting your indexes.
- Creating views.
- Modifying your connection strings.

In the subsection below, we will 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. We also explain how you might assess current processing speed.

Assess current processing speed

A good rule of thumb for SQL Server data sources is to target reading at least 80,000 rows per second per table being processed in parallel for typically sized tables. If you think the ProcessData phase of processing is running too slowly, check the **MSOLAP:Processing - Rows read/sec** performance counter. If the throughput of that counter averaged over the duration of ProcessData is much lower than the target value, then you should optimize the relational database layer.

You should also refer to the Profiler Trace Events to Watch during Processing section for information on determining how the SQL query for each partition performed.

Remove unused columns from the query

During processing, the query bindings for each table are run without alterations. If some columns are being returned from the query but not used in the table, change the query not to return unused columns. These unused columns waste network, memory and processor during processing.

Get rid of joins

If you are using a database view or a query as the basis of partitions, you should seek to eliminate joins in the query sent to the relational database. If some joins are not necessary in the query, eliminate them. If some joins are necessary, consider refactoring the relational data model and consider performing joins during the ETL instead of during Tabular model processing. 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.

Get relational partitioning right

If you use partitioning on the relational side, you should ensure that each Tabular model 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.

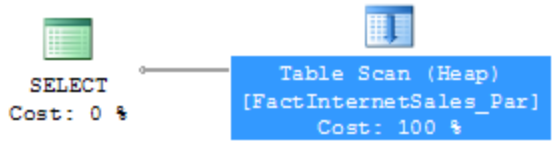


Figure 36: An optimal partition processing query

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

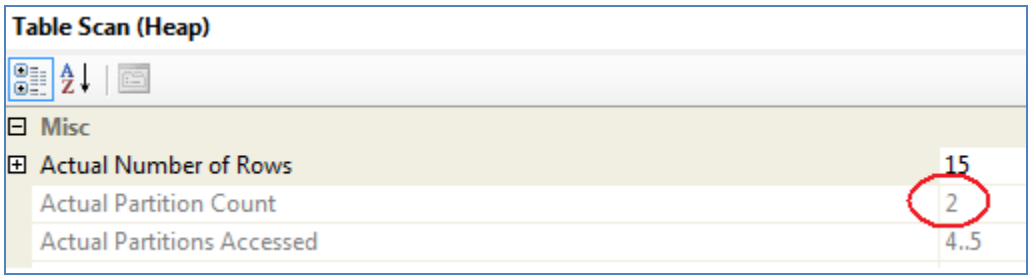


Figure 37: Too many partitions accessed

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), consider repartitioning the relational source data so that each Tabular model partition touches at most one relational partition.

Choose appropriate index FILLFACTOR

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.

On tables that are truncated and reloaded during the ETL, be sure to use FILLFACTOR=100 on all indexes so that SQL Server doesn't intentionally leave free space in every page of data. The same advice applies to

- Tables that only receive inserts, but no updates.
- Indexes that are dropped and recreated during the ETL.

However on tables which receive regular updates, FILLFACTOR=100 may cause [fragmentation](http://blogs.msdn.com/b/cindygross/archive/2009/11/20/sql-server-and-fragmentation.aspx) (http://blogs.msdn.com/b/cindygross/archive/2009/11/20/sql-server-and-fragmentation.aspx).

If eliminating free space in pages will be beneficial in your data loading scenario, you can check for index pages that 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.

Data compression

In SQL Server 2012 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 unless the SQL Server instance is CPU bound it is very likely compression will improve query performance.

Though SQL Server 2012 offers a nonclustered columnstore index which provides high levels of compression, this columnar storage is optimized for summary level queries with aggregation, not for streaming millions of detailed rows. The SQL Server optimizer chooses which queries will benefit from use of the nonclustered columnstore index, and typical queries sent by Analysis Services during processing will not benefit from a columnstore index. For more information about how to use columnstore indexes, we recommend the FAQ, or Books Online.

Eliminate 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 incur during processing.

To eliminate this locking overhead (at the cost of potentially blocking other queries), you have three options:

- Option 1: Set the relational database to **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.

```
ALTER INDEX ALL ON 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.

```
CREATE VIEW 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.

Perform other standard relational database tuning best practices

Any normal relational database tuning should be applied. Such best practices for SQL Server may include the following:

- Run the SQL Server Best Practices Analyzer (BPA) and/or System Center Advisor (SCA) and implement relevant recommendations.
- If you are not doing scans of the entire underlying table you may be able to improve performance of the SQL queries by tuning the SQL Server indexes.
- Reduce fragmentation.
- Optimize configuration with settings such as optimize for ad hoc workloads, Lock Pages in Memory, Max Server Memory, MAXDOP, and cost threshold for parallelism.
- Verify you have appropriate power settings at the OS and hardware levels.
- Ensure you have sufficient hardware resources to meet the demands of the SQL Server instance during peak processing times.
- Reduce the load on the SQL Server instance during processing times.

Modify data feeds

Data that is imported via an Excel data feed may also get embedded in the script depending on how the feed was configured in Excel or PowerPivot. Specifically if the data was imported using the Excel import data feed option, the data import uses a push model which behaves just like copy/pasted data. As a result, feed data gets embedded in the database script which can hurt performance for more than 50 rows of data.

If data was imported from a feed using PowerPivot, it uses the OData Provider that comes with the .Net framework 4.0. This data goes directly into the tabular engine and does not get embedded into the tabular model's metadata. If you are creating a tabular model from a PowerPivot workbook that contains a data feed, check how the feed data was imported; if it uses the Excel data feed method, you should open PowerPivot and use the PowerPivot import from data feed method instead.

Consider using SQL Server Integration Services to stage the data feed data to SQL Server to avoid the time and Azure licensing costs of reading from the data feed every time the Tabular model processes.

Optimizing the Network

On large models, the network between the relational data source and Analysis Services can cause slowdowns in processing. Optimizations for the network layer have been extensively documented in the SQL Server 2008 R2 Analysis Services Operations Guide in section 2.6.

One specific recommendation is worth calling out, because the dialog box for setting the data source properties looks slightly different than in Multidimensional models. When the data source is SQL Server, the default packet size for SQL Server is 4K. Editing the Packet Size connection string property in Analysis Services allows more efficient communication with less packet overhead.

To edit the connection properties:

1. In SQL Server Data tools, from the **Model** menu, select **Existing Connections**.
2. Edit the advanced properties of each connection, setting **Packet Size** to 32KB (32767 as seen in the following figure).

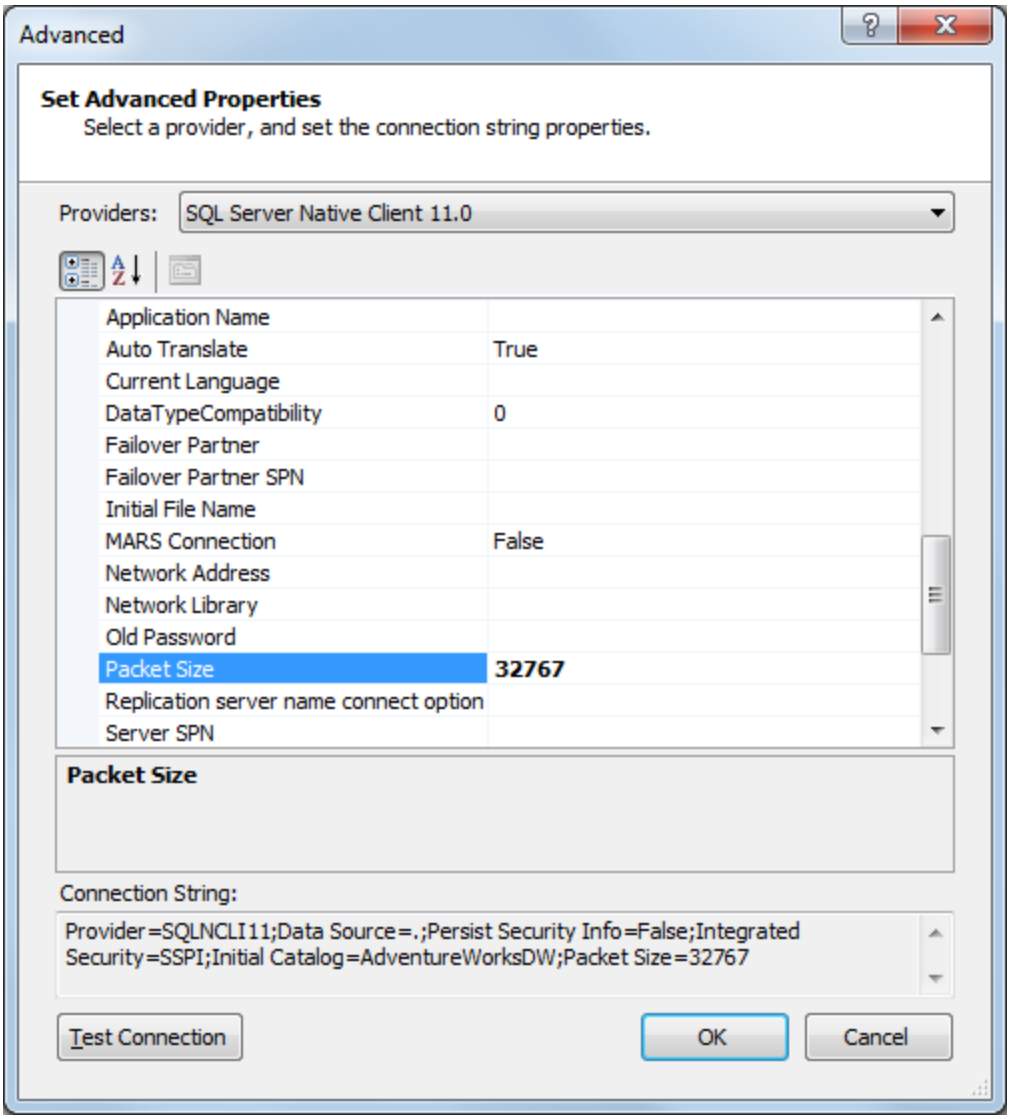


Figure 38 Tuning network packet size

Reducing Memory Usage during Processing

Processing is a memory intensive operation. The following bullet points are pointers to relevant content elsewhere in this whitepaper.

If you encounter a memory constraint you may decide to increase the RAM available to Analysis Services and/or you may choose to tune the system to reduce memory consumption. Consider which approach is more cost effective for your scenario.

Increase memory available to Analysis Services

- Add physical RAM to the system.

- Reduce the memory used by other applications or remove those other applications from the server.

Reduce memory usage during ProcessData

- Follow the advice in the section on Compression about lowering **DefaultSegmentRowCount**, realizing this may result in worse compression and worse query performance.
- Study the server memory limits and decide if they should change.
- Reduce the number of tables that are being processed in parallel using the **MaxParallel** property of the Parallel tag, or by avoiding the Parallel tag.
- Remove or optimize high cardinality columns.

Reduce memory usage during ProcessRecalc

- Follow the advice about **MaxParallel** in the section on ProcessRecalc above.
- Optimize the DAX in calculated columns.

Reducing Time Spent during Processing

If you decide to focus on reducing the total time spent processing, rather than memory usage, try these techniques:

- Tune the relational layer using the advice provided in the preceding section.
- Avoid paging. Utilize perfmon counters to detect when paging is occurring and slowing processing performance. Then attempt to avoid paging by reducing memory usage.
- Avoid doing the same work multiple times by avoiding running multiple ProcessFull operations. Instead, prefer multiple ProcessData commands with one ProcessRecalc command and ensure the Parallel tag is used to achieve parallel processing.
- Study the common processing methodologies and consider processing a subset of partitions or consider using ProcessAdd.
- Study common partitioning strategies to better enable incremental loading.
- Study Processing thread pool perfmon counters to determine if settings changes are needed.

Improving Compression

See the section on Compression Timebox for information about spending more time during processing to achieve greater compression.

Increasing Available Threads

SQL Server 2012 Analysis Services includes an intelligent default for its Processing thread pool. The **ThreadPool\Process\MaxThreads** setting defaults to 0 which currently means -1. The value -1 means "1 multiplied by the number of cores", with a minimum of 64 threads. To confirm this, look at msmdsrv.log in the LogDir folder which contains a new informational message written each time the service starts:

Message: The Processing thread pool now has 1 minimum threads, 64 maximum threads, and a concurrency of 16. Its thread pool affinity mask is 0x00000000000000ff.

Due to the intelligent default, changing the size of this thread pools is only necessary in extreme workloads. In order to know whether the size of a thread pool is limiting parallelism, monitor the **MSOLAP:Threads – Processing pool job queue length** perfmon counter and if it is > 0 while the **MSOLAP:Threads – Processing pool idle non-I/O threads** counter = 0, then consider changing the **ThreadPool\Process\MaxThreads** setting to a higher value, for example to 128. As with any setting change, measure processing performance before and after.

The new IOProcess thread pool is not relevant to Tabular models, only multidimensional models.

Partitioning

Partitioning describes the process of taking one large table and breaking it into several sets of rows. For example, in a 120 million row table covering one year of data, twelve monthly partitions could be built, each containing 10 million rows of data.

The Role of Partitioning in Performance Tuning

Unlike Multidimensional models where partitioning can improve query performance and can increase parallelism during processing, in Tabular models, partitioning does not improve query performance or processing parallelism. Specifically, no partition elimination currently takes place during querying and partitions within a single table cannot be processed in parallel. (Multiple tables can be processed in parallel, though.) In fact, excessive partitioning could create many small segments and reduce query performance, as discussed above. Therefore, the main purpose of partitioning in Tabular models is to aid in incremental loading and administration.

Partition size does not generally matter in Tabular models. Splitting a large table into multiple partitions will not significantly decrease the memory requirements during processing and will not increase query performance. This may come as a surprise to administrators used to managing Multidimensional models where they have learned specific partition sizing recommendations.

Partitioning Scenarios in Tabular Models

When faced with a large model or a tight refresh window, the model administrator should review the following common partition design patterns to help decide on the proper partitioning scheme for the model.

Rolling off the oldest data monthly

A common requirement in business intelligence is to provide the users with a rolling period of data such as providing the most recent 36 months of data. This allows the data mart and Tabular model to avoid constant growth over time and to provide more predictable processing and query times over time. Without partitioning, the model administrator would need to reprocess the entire table each month in order to purge the oldest month of data. However, if the table is partitioned by month in the Tabular model the administrator can automate deleting the oldest partition. Unlike reprocessing the entire table, deleting a partition is a very lightweight operation. Be sure to run a ProcessRecalc inside the same transaction as the partition delete to leave the model queryable:

```
<Batch xmlns="http://schemas.microsoft.com/analysisservices/2003/engine" Transaction="true">
```

```
<Delete>
 <Object>
  <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
  <CubeID>Model</CubeID>
  <MeasureGroupID>Reseller Sales_fc635e72-28dc-4156-80d5-43b805f8df1c</MeasureGroupID>
  <PartitionID>Reseller Sales_5e10967a-f64b-4818-aebe-043d813d5abc</PartitionID>
 </Object>
</Delete>
<Process>
 <Type>ProcessRecalc</Type>
 <Object>
  <DatabaseID>AdventureWorks Tabular Model SQL 2012</DatabaseID>
 </Object>
</Process>
</Batch>
```

When dropping partitions, over time the table's dictionaries can begin to contain many items which no longer exist. For example, once the last partition containing data for a closed store is deleted, that StoreID will continue to occupy space in the table's dictionary until a ProcessDefrag processing operation is run on the table. ProcessDefrag is expensive in terms of memory and time, but it usually necessary in this rolling period partitioning scenario. See Running ProcessDefrag for further advice.

Partitioning in order to perform incremental loading

For large Tabular models, fully reprocessing the entire model daily may not be feasible. In this scenario, partitioning can be used in several ways to aid in incremental loading.

Partitioning to minimize which partitions are processed. One method of using partitioning to aid in incremental loading is to partition so that only a few partitions need to be reprocessed daily. In scenarios when only the data in the most recent month's partition changed, processing only that one partition rather than the whole table will decrease processing time. In many data marts, it is not possible to guarantee which range of data will typically change daily. In that case, consider decorating fact and dimension rows with metadata that tracks the last time they were changed by the ETL. This metadata allows the Tabular model administrator to determine which partitions contain changed data and reprocess the minimum partitions.

Geographic partitioning. Another partitioning method is to build a partitioning scheme which mirrors the data availability by geographic region. For example, in international companies, the work day is finished in Europe many hours before the work day in the United States, so separate ETL processes can extract Europe data from the source system and load the data mart many hours before United States stores close. In this scenario, partitioning fact tables by geographic region to mirror the nightly ETL schedule may provide fresher data if the proper partitions are processed as soon as that geographic region's data is ready.

Creating new partitions and merging. Another partitioning method to aid in incremental processing can be used in tables which only receive inserts (not updates or deletes). In this scenario, create a new partition with a query binding which picks up only new records. Also consider merging this new partition with an existing partition so that the partition count does not grow indefinitely. Tabular models should not have thousands of partitions due to the overhead of managing that amount of partition metadata.

Furthermore, hundreds of small partitions (such as one million rows each) will create hundreds of small segments which will be less efficient during querying than fewer larger segments. For these reasons, merging small partitions is recommended. Note that creating a new partition and then merging it could alternately be accomplished by performing a ProcessAdd on an existing partition to add new records.

Converting updates into journal style inserts. Some fact tables which currently update rows nightly could be transformed into journal style inserts. For example, if a customer orders 10 units and then changes the order to 12 units the next day, this update could be refactored into an insert of a row representing an order of 2 units. Besides the benefit of improved traceability in the data warehouse, refactoring updates into inserts allows the Tabular model administrator to avoid reloading existing rows and spend processing time on loading new rows. If this refactoring is done, ensure that queries do not make incorrect assumptions about an order occurring on only one row.

Fragmentation in Partitions

Fragmentation in hash dictionaries can occur when partitions are deleted over time or individual partitions are processed over time. You can potentially improve processing performance by re-processing partitions that have been fragmented, using **ProcessDefrag**.

The following MDX query can detect the level of fragmentation in a particular column. Run the following query to check for fragmentation:

1. Replace the yellow-highlighted text (in this example, [Reseller Sales]) with the name of the table
2. Replace green-highlighted text (in this example, [Extended Amount]) with the name of the column.
3. You should only run this code for any columns which are hash encoded, as described in the Detecting the encoding type for a column section.

```
WITH
MEMBER [Measures].[Cardinality] as
 [Reseller Sales].[Extended Amount].Members.Count
MEMBER [Measures].[Max DataID] as
 Max(
  [Reseller Sales].[Extended Amount].[Extended Amount].Members
  , DataID([Reseller Sales].[Extended Amount].CurrentMember)
 )
MEMBER [Measures].[Fragmentation Percent] as
 1 - [Measures].[Cardinality] / [Measures].[Max DataID]
,FORMAT_STRING = "0%"
SELECT {
 [Measures].[Cardinality]
 , [Measures].[Max DataID]
 , [Measures].[Fragmentation Percent]
} on COLUMNS
FROM [Model]
```

If there is a lot of churn in partitions, check for fragmentation. When the fragmentation value gets to be higher than 10% on columns with high cardinality, consider running ProcessDefrag.

Summary—Processing and Partitioning

In this section we reviewed the internals of processing and partitioning in Tabular models and discussed best practices.

- Processing allows the administrator to refresh the data in a model.
- Partitioning allows the administrator to perform incremental processing, loading a subset of the data.
- Optimizing processing involves performing incremental loading, parallel processing, and optimizing the relational database layer.
- Common mistakes such as serial processing or repeating ProcessRecalc work in multiple transactions can be avoided by understanding how to structure processing batches.
- Processing can also be optimized to increase compression and improve query performance through changing several settings.
- Partitions are useful as a management tool, not as a tool to achieve parallel processing or to improve query performance.

5. Tuning the System

Tuning a server to perform optimally for the particular models deployed requires an understanding of service packs, server configuration settings, and memory management. This section summarizes common tasks for server and model administrators to tune the system.

Apply the Latest Service Packs and Cumulative Updates

Before tuning a server, ensure that the latest service pack has been applied. Service packs contain important bug fixes and enhancements which improve performance, accuracy, and stability. Service packs undergo thorough regression testing within Microsoft and among Microsoft partners before being released. We recommend that you always install a service pack in your development or test environment first and validate that it behaves as expected with your model, and then install the latest service pack in production.

Cumulative updates are bi-monthly rollups of important fixes that appear between service packs. With the frequency of cumulative update releases, they are not tested with quite the rigor that service packs receive. Many companies opt to install the latest service pack but to only install cumulative updates if one or more of the fixes in them are needed on their project. Other companies that have their own rigorous testing procedures prefer to stay current by applying cumulative updates on a regular cycle. Be aware that typically the most recent few cumulative updates do not make it into the next service pack. For example, although SQL Server 2012 Cumulative Update 4 was released before Service Pack 1, the fixes included in this RTM based CU4 were not present in Service Pack 1, but instead were released soon after SP1, as part of Service Pack 1 Cumulative Update 1 (SP1 CU1).

To stay informed of service pack and cumulative update release, subscribe to the SQL Server Release Services blog (http://blogs.msdn.com/b/sqlreleaseservices/).

To determine which version of Analysis Services you are running, open SQL Server Management Studio and connect Object Explorer to your Analysis Services instance. The icon next to the server name will resemble the following figure, if it is in Tabular mode. The version number (in this case, 11.0.3000) appears to the right of the server name.

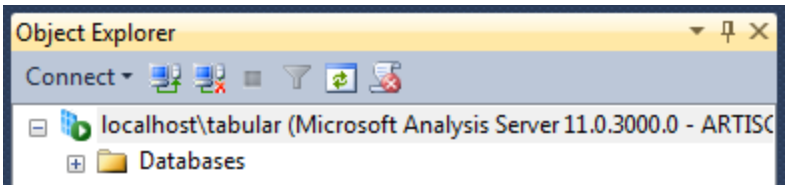


Figure 39. Finding the SSAS version number

You can then cross-reference the version number against this helpful list of SQL Server Builds, to find out which updates are included in your version: (http://sqlserverbuilds.blogspot.com/). For example, version 11.0.3000 is SQL Server 2012 Service Pack 1 (SP1) and version 11.0.3368 is SQL Server 2012 Service Pack 1 Cumulative Update 4 (SP1 CU4). This is not an official Microsoft page, but it's always very up-to-date and is very easy to use.

When looking at version numbers, don't confuse the version number of the server or backend with the version of the SSMS client. Often they won't match.

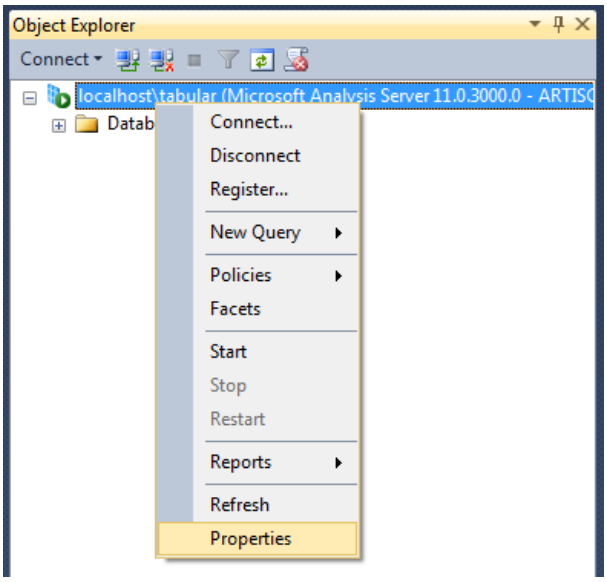
Configure Analysis Services Server Settings

Analysis Services has server configuration properties that determine thresholds, feature availability, and server behaviors. Defaults for these properties target a middle of the road solution, determined by repeated testing and customer feedback. For solutions that fall outside the norm, particularly large solutions or systems that are under heavy load, you might want to modify server configuration to better fit the requirements of the solutions you are supporting.

Note: Server configuration properties apply to the instance as whole, and not to individual databases. If you are supporting multiple solutions that run best on different server configurations, you'll either need to optimize for the solution that is most important to your business, or install multiple instances that you can configure independently.

Changing Server Settings in SQL Server Management Studio

The easiest way to change common server settings is through the server properties dialog inside SQL Server Management Studio. Connect Object Explorer to your Analysis Services instance running in Tabular mode, then right click on the server node and choose **Properties**:



The **General** tab of the **Properties** pane contains common server settings, their current values, their default values, and whether changing them requires a restart.

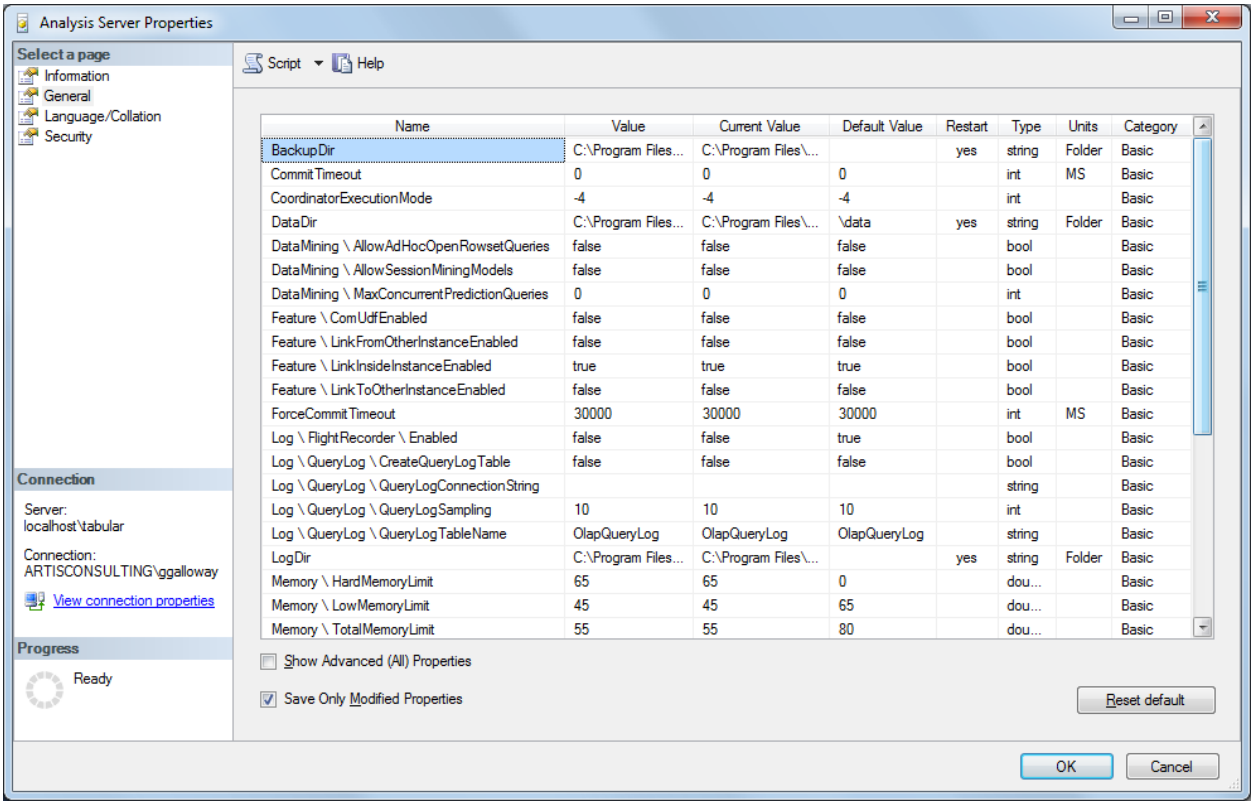


Figure 40. Analysis Services advanced properties

Clicking the link, **Show Advanced (All) Properties**, will show some additional properties that you can set at the server level, some of which are less commonly changed.

Performance Related Server Configuration Settings

There are over one hundred settings in this dialog box, and hundreds more that are available only in the msmdsrv.ini file discussed below. However, most of the settings are relevant only to Multidimensional instances.

The following table lists the settings most relevant to performance tuning Tabular instances, along with notes on how to manage the property. These properties are all available in the **Server Properties** dialog. Where not otherwise specified, the properties are listed under **General** server properties.

Table 7. List of server configuration settings

Setting Name	Comments
DataDir	Identifies the name of the directory where data is stored All database files are persisted to disk in this directory so that they can be loaded back into memory after the service starts when the first user connects. • Make sure the DataDir is set to a folder on disks which contain redundancy through RAID. • Put the DataDir on the fastest disks available but realize that disk performance is typically unimportant for Tabular models. Disk performance is only important in real-time refresh scenarios where performance of the processing transaction commit is important. • Otherwise, don't spend your budget on expensive SAN or direct attached storage for Tabular models.
LogDir	Identifies the name of the directory that contains server logs. The LogDir is used for minidumps and flight recorder and a few other log files. Unlike SQL transaction logs, the information in this file is purely informational, and is not used for rollback. However, if you are not careful the minidump files can grow to consume considerable disk space, causing out of disk space errors. So make sure this directory is on a drive with ample free space (in case you begin generating minidump files due to a bug) and periodically monitor it.
BackupDir	Identifies the name of the directory where backup files are stored by default. If performing Analysis Services database backups, make sure this directory is on separate physical media than the DataDir.

ExternalConnectionTimeout	Defines the timeout, in seconds, for creating connections to external servers. See section 4 for performance tuning advice.
ExternalCommandTimeout	Defines the timeout, in seconds, for commands issued to external servers See section 4 for performance tuning advice.
OLAP\Process\DatabaseConnectionPoolMax	Specifies the maximum number of pooled database connections. See section 4 for performance tuning advice.
ServerTimeout	Defines the timeout for queries. The default is 3600 seconds (or 60 minutes). Zero (0) specifies that no queries will timeout.
ForceCommitTimeout	Specifies how long a write commit operation should wait before canceling other commands that preceded the current command, including queries. See section 4 for performance tuning advice.
CommitTimeout	Specifies how long the server will wait to acquire a write lock for the purpose of committing a transaction. See section 4 for performance tuning advice.
Log\FlightRecorder\Enabled	Indicates whether the flight recorder feature is enabled. You should change this setting to 0 on a production server to not waste resources logging this information.
Memory\HardMemoryLimit	Specifies a memory threshold after which the instance aggressively terminates active user sessions to reduce memory usage. See the following section on Memory Management.
Memory\TotalMemoryLimit	Defines a threshold that when reached, causes the server to deallocate unused caches more aggressively. See the following section on Memory Management.
Memory\LowMemoryLimit	Specifies the threshold amount of memory allocated by Analysis Services at which it will begin cleaning caches to lower memory usage. See the following section on Memory Management.
Memory\VertiPaqMemoryLimit	Specifies the limit for VertiPaq memory consumption. Depending on paging settings discussed below, above this limit paging starts or memory allocation errors occur. See the following section on Memory Management.
Memory\VertiPaqPagingPolicy	Specifies the paging behavior in the event the server runs low on memory. See the following section on Memory Management.

ThreadPool\Query\MaxThreads	Defines the maximum number of threads for servicing queries. Each user query requires one thread from this thread pool. The default of 0 currently means -2 which means 2 threads per core with a minimum of 10 threads. This default makes it unlikely that this setting will need to be changed often. On servers with many concurrent users running queries, if the **MSOLAP:Threads\Query pool job queue length** counter is greater than 0 then consider lowering this setting to **-3** or **-4** in order to increase the number of threads per core. This will cause more queries to share time on the same cores and potentially slow down individual queries.
ThreadPool\Process\MaxThreads	Typically it is not recommended that you change this property. See section 4 for performance tuning advice.
VertiPaq\DefaultSegmentRowCount	Define the size of the segments used for storing column values. See section 4 for performance tuning advice.
VertiPaq\ProcessingTimeboxSecPerMRow	Defines a time limit for processing rows. See section 4 for performance tuning advice.
Feature\ResourceMonitoringEnabled	Indicates whether internal resource monitoring counters are enabled. On a production server, you might set this property to 0, to skip certain kinds of resource monitoring that can be costly, especially on a NUMA server. However, some DMVs will not report correct information if this property is disabled.

For additional information about server properties, see Books Online: http://msdn.microsoft.com/en-us/library/ms174556.aspx.

Finding the Config Directory and the msmdsrv.ini File

Behind the scenes, changes to the server properties made in the **Server Properties** dialog are persisted to disk in a file named **msmdsrv.ini**. Some less commonly changed properties can only be changed in that file. Even if the administrator has no intention of changing less common settings, finding the msmdsrv.ini file does enable the administrator to quickly compare the configuration files from different servers using windiff.exe (http://support.microsoft.com/kb/159214) or other file diff utilities. You may also want to periodically back up this file so you can refer to it when troubleshooting or comparing present and past behavior.

In general, you can look in the DataDir property to find the location of the configuration files, as described in Books Online: http://msdn.microsoft.com/en-us/library/ms174556.aspx. However, it is possible to have multiple instances of Analysis Services on the same server. It is even possible to have

orphaned directories from old uninstalled instances. The following instructions will ensure that you find and edit the correct msmdsrv.ini configuration file.

1. Log into to the server (either directly or by using a Remote Desktop connection), and then open Task Manager.
2. In Task Manager, on the **Processes** tab, click **Show processes from all users**.
3. From the **View** menu, choose **Select Columns** and check the **Command Line** option. Click **OK**.
4. On the **Services** tab, click the column heading to sort by **Description**, and find the appropriate instance of SQL Server Analysis Services.

For example, the following graphic shows the named instance called TABULAR, which has the name MSOLAP$TABULAR and the description "SQL Server Analysis Services (TABULAR):

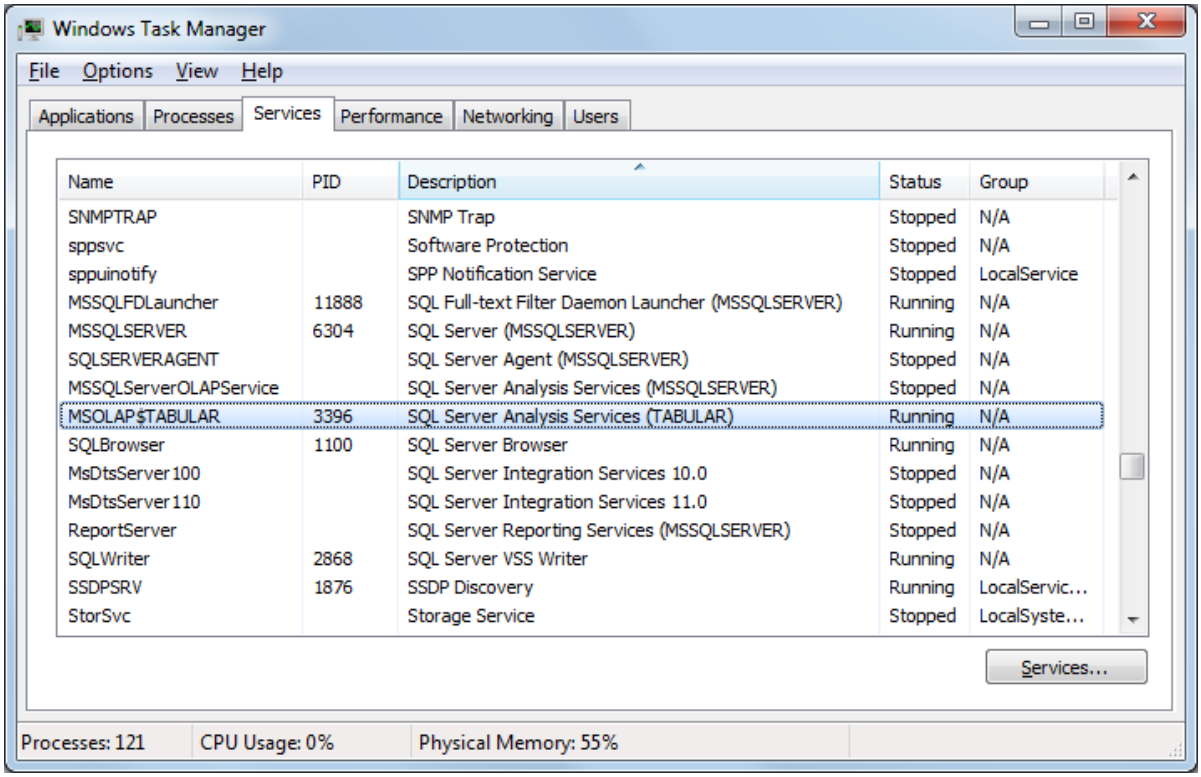


Figure 41. Locating the tabular instance in Task Manager

5. Right click the service and select **Go to Process**.
6. In the highlighted process, review the text in the **Command Line** column. The **-s** switch will provide the path to the **Config** directory.

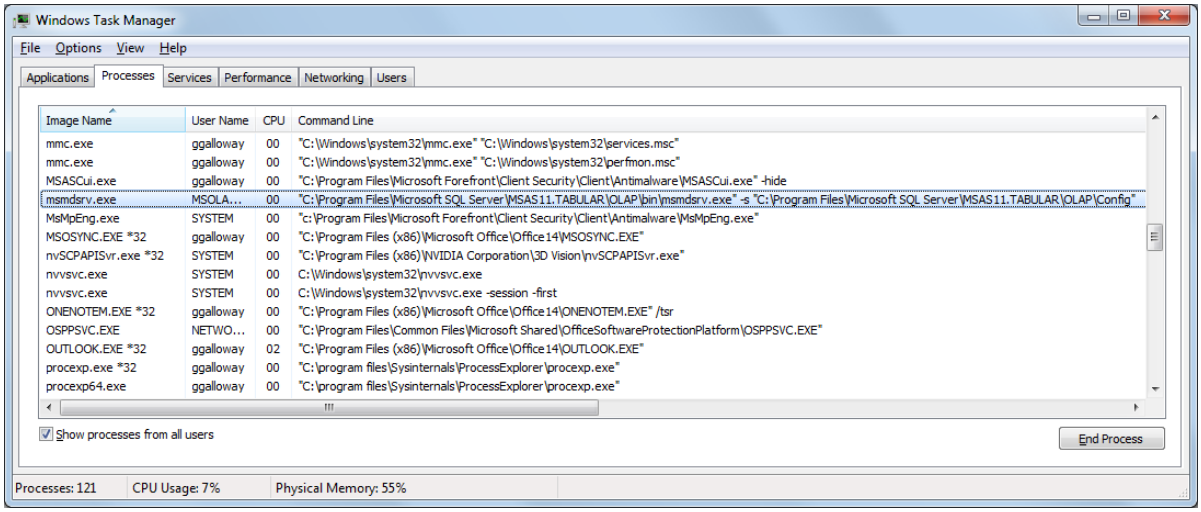


Figure 42. Location of server files in Task Manager

7. From the Start menu, locate Notepad.
8. Right-click Notepad, and choose **Run as Administrator**.
9. From the Notepad **File** menu, select **Open**.
10. Locate the msmdsrv.ini (***not*** the msmdsrv.ini.bak) file in the **Config** directory.

Note that administrative permissions are required to read and make changes.

All server configuration settings are visible in the msmdsrv.ini file, which is an XML file. Similar to the properties that you can set in SQL Server Management Studio, some properties require a restart before they take effect.

Always make a backup of your INI files!
If you need to alter properties in this file, it is best to stop the Analysis Services service, make a backup of the msmdsrv.ini file, change the msmdsrv.ini property, and then restart the instance.
Need to look at your INI file remotely? Usually the instance id, instance name, and folder name match each other, and it's easy to view them remotely without the overhead, security, and trouble of connecting directly to the server.

Understanding Memory Management

Because Tabular models employ in-memory technology, memory usage is a tremendously important concept to understand. This section describes how Analysis Services manages memory and paging.

Memory Usage When Analysis Services Starts

When a Tabular instance of Analysis Services starts, it loads some lightweight metadata of the databases that exist on that instance. Then when the first user runs a query against a particular database, that database in its entirety is loaded into memory.

Tabular models aren't loaded into memory when the server starts (or restarts) – they are loaded into memory when the first user queries a particular database.

The experience differs somewhat when you are using administrative tools such as SQL Server Management Studio or other connections to the server with the Analysis Management Objects (AMO or Microsoft.AnalysisServices namespace). These tools cause all databases to be loaded into memory during the DISCOVER_XML_METADATA command.

Memory Limit Settings

Memory limits are derived from the server configuration settings listed below, following these rules:

- If the settings are expressed as a number between 1 and 100, then the value represents the percentage of total server RAM that will be allowed for that Analysis Services instance.
- If the settings are expressed as a number above 100, the value represents the exact number of ***bytes*** to allow.

Be very careful in your calculations. For example, if you want to use 8GB, you must compute the number of bytes, for example:

```
8 * 1024 * 1024 * 1024 = 8,589,934,592 bytes
```

Then, be sure to enter the value without the commas. If you accidentally express the memory limit as KB or MB, Analysis Services will be extremely short on memory.

VertiPaqMemoryLimit controls the maximum amount of memory that can be used for VertiPaq objects (such as dictionaries and segments) except when paging is enabled. Paging is discussed below. VertiPaqMemoryLimit defaults to 60%.

If memory passes above **LowMemoryLimit** then Analysis Services will begin cleaning caches to lower memory usage. If memory usage continues to climb, then cleaning will grow progressively more aggressive. LowMemoryLimit defaults to 65%.

If memory passes above **TotalMemoryLimit**, then all caches not in use will be evicted. This setting defaults to 80%. Setting LowMemoryLimit to about 80% or less of **TotalMemoryLimit** is a best practice so the cleaner has some runway to clean memory before it exceeds **TotalMemoryLimit**.

If memory passes above **HardMemoryLimit**, then active sessions will be cancelled to reduce memory. This setting defaults to 0 which means halfway between **TotalMemoryLimit** and total server memory (or 90% which is halfway between 100% and the default **TotalMemoryLimit** of 80%).

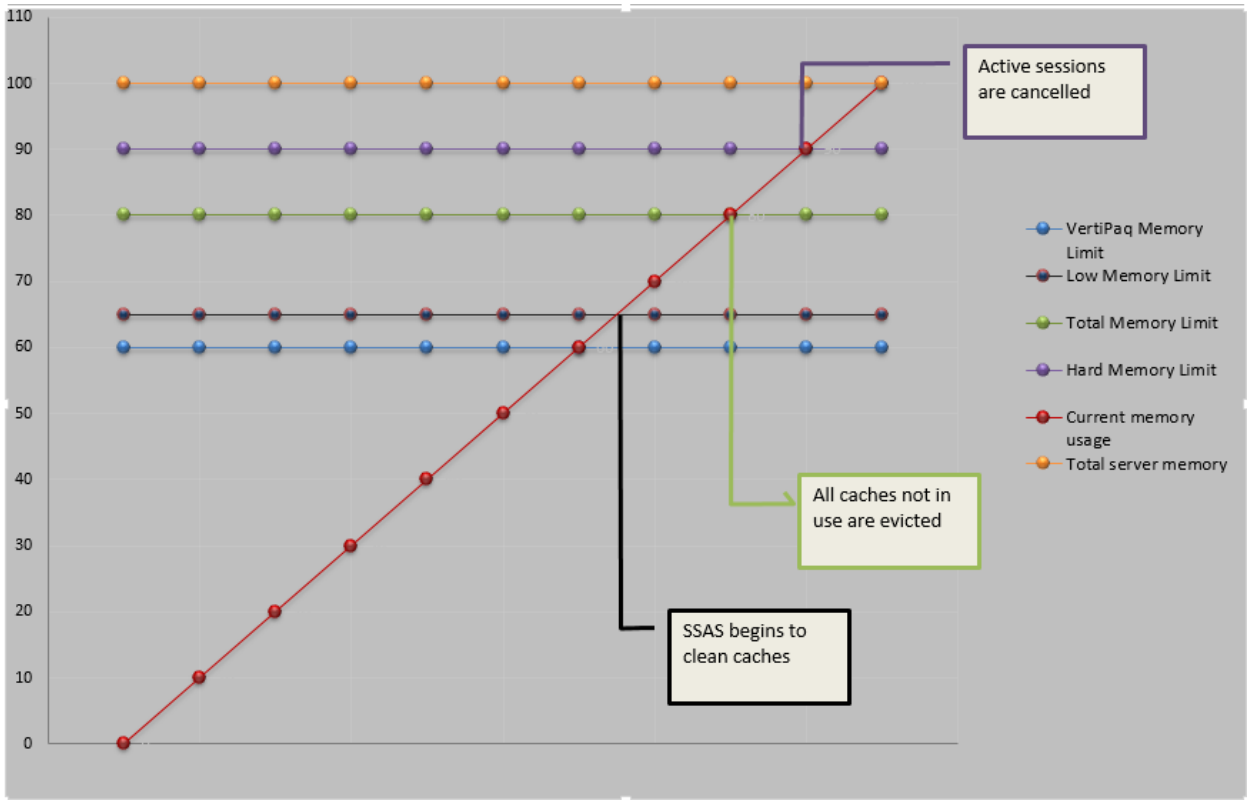


Figure 43. Memory limits and current memory usage

On servers with a small amount of RAM (such as 32GB) and no services other consuming memory besides the single Analysis Services instance, these settings are reasonable defaults since these defaults leave about 6GB of RAM for the operating system.

On servers with large amounts of RAM and no other memory consuming services (such as SQL Server database engine) running on them, these defaults should probably be raised. For example, on a server with 256GB of RAM, if you use the default settings, the server will begin cleaning caches once memory usage reaches LowMemoryLimit of 167GB, and this is probably too aggressive.

Paging

In the event that VertiPaq objects (column dictionaries, segments, hierarchies, and relationship objects) consume all the memory allowed by VertiPaqMemoryLimit, depending upon the paging setting, memory allocations will either receive out-of-memory errors (causing queries or processing to fail) or will begin paging. A server setting **Memory\VertiPaqPagingPolicy** controls whether paging of VertiPaq objects is allowed. This setting has three possible options:

- 0 means no paging is allowed. Once VertiPaqMemoryLimit has been exhausted, then processing or queries will fail.
- 1 (the default) will allow paging to the Windows page file (pagefile.sys). Dictionaries must remain in memory, though.

- 2 is currently not supported. It is mentioned to make it clear it is not supported since some blog posts on the Internet say otherwise. If supported in the future, this paging policy will allow paging through memory mapped files. Dictionaries must remain in memory.

Paging is not intended to be an ongoing state in Tabular models. Paging was implemented to allow brief periods where memory exceeds limits during processing or queries. Paging drastically hurts performance. To achieve predictable performance, paging should be avoided. If paging is unavoidable, and if policy 1 is set, then ensure pagefile.sys resides on the fastest local disks.

Use the following formula to detect how close you are to the VertiPaqMemoryLimit:

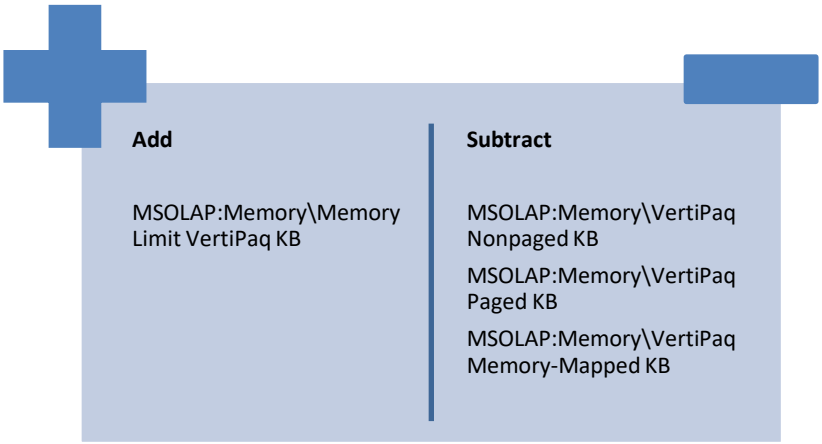


Figure 44. Formula for estimating remaining memory

- If VertiPaqPagingPolicy=0, then all memory will show under **MSOLAP:Memory\VertiPaq Nonpaged KB**.
- If VertiPaqPagingPolicy=1, dictionaries will be locked in memory and the **MSOLAP:Memory\VertiPaq Nonpaged KB** perfmon counter will show that, while all other VertiPaq memory will be pageable and will be shown under the **MSOLAP:Memory\VertiPaq Paged KB** counter.
- If VertiPaqPagingPolicy=2, you should see zero under **MSOLAP:Memory\VertiPaq Memory-Mapped KB**. This setting is unsupported it is included in the formula for completeness.

(Unfortunately, as of this writing, a bug is preventing these counters from returning values on SQL Server 2012 SP1. A fix for this issue is targeted for a future cumulative update or service pack.)

To monitor Analysis Services memory usage overall (VertiPaq and other memory usage), use Performance Monitor to watch the **MSOLAP:Memory\Memory Usage KB** counter and correlate it with a Profiler trace to determine which processing operation or query is causing memory usage to increase.

When VertiPaqPagingPolicy is set to 0, then the memory limits and cleaner behave as described in Memory Limit Settings. However, if VertiPaqPagingPolicy allows paging, then memory limits calculations

are more complex. In this situation, in order to allow paging of VertiPaq structures, the cleaner ignores VertiPaq memory usage and focuses on other memory usage such as sessions and caches.

If paging is disabled and processing or queries require more memory than is allowed, out-of-memory errors will be received by the calling application that say the following: "*The operation has been cancelled because there is not enough memory available for the application. If using a 32-bit version of the product, consider upgrading to the 64-bit version or increasing the amount of memory available on the machine.*"

Creating a Server Memory Plan

Now that you understand how Analysis Services manages memory, on servers where Analysis Services is co-located with the SQL Server database engine, create a server memory plan that accounts for all services, such as the following:

Table 8. A sample server memory plan

Memory Consumer	Memory Limit
Analysis Services	124GB
SQL Server Database Engine	124GB
Operating System	6GB
Programs like SSMS	2GB
Total	**256GB**

In this example, monitoring indicated that 6GB was sufficient for the operating system and built-in Windows services. The amount of memory to leave for the operating system and other services and applications will vary depending on how your server is configured. The operating system needs its own resources to service some of the requests by Analysis Services, and monitoring, backup, scheduling, data movement, troubleshooting, and other applications, utilities, and services can all use memory. Monitor the memory used by Analysis Services and the total memory used at various times to estimate how much is enough to leave outside of the Analysis Services instance(s). Make sure all unnecessary services and applications are stopped, disabled, or configured to use no more memory and other resources than absolutely necessary.

In the example above, several GB was planned for programs like Management Studio to allow one administrator to Remote Desktop to the server and perform administrative actions directly on the server, rather than through Management Studio on the administrator's laptop.

Try to reduce or eliminate usage of desktop GUI applications such as SQL Server Management Studio, SQL Server Profiler, and SQL Server Data Tools on the server.
Script out your actions or connect from a remote machine whenever possible.
However, if usage of such applications on the server is necessary or preferred, plan accordingly in your server memory plan.

Memory usage for Analysis Services was capped using the MemoryLimit properties discussed above. Memory limits were also set for other large memory consumers (such as SQL Server in this example) to ensure they don't consume more than their fair share. To cap the amount of memory used by SQL Server, use the **Maximum server memory** (http://support.microsoft.com/kb/2663912) property in the **Server Properties** dialog box:

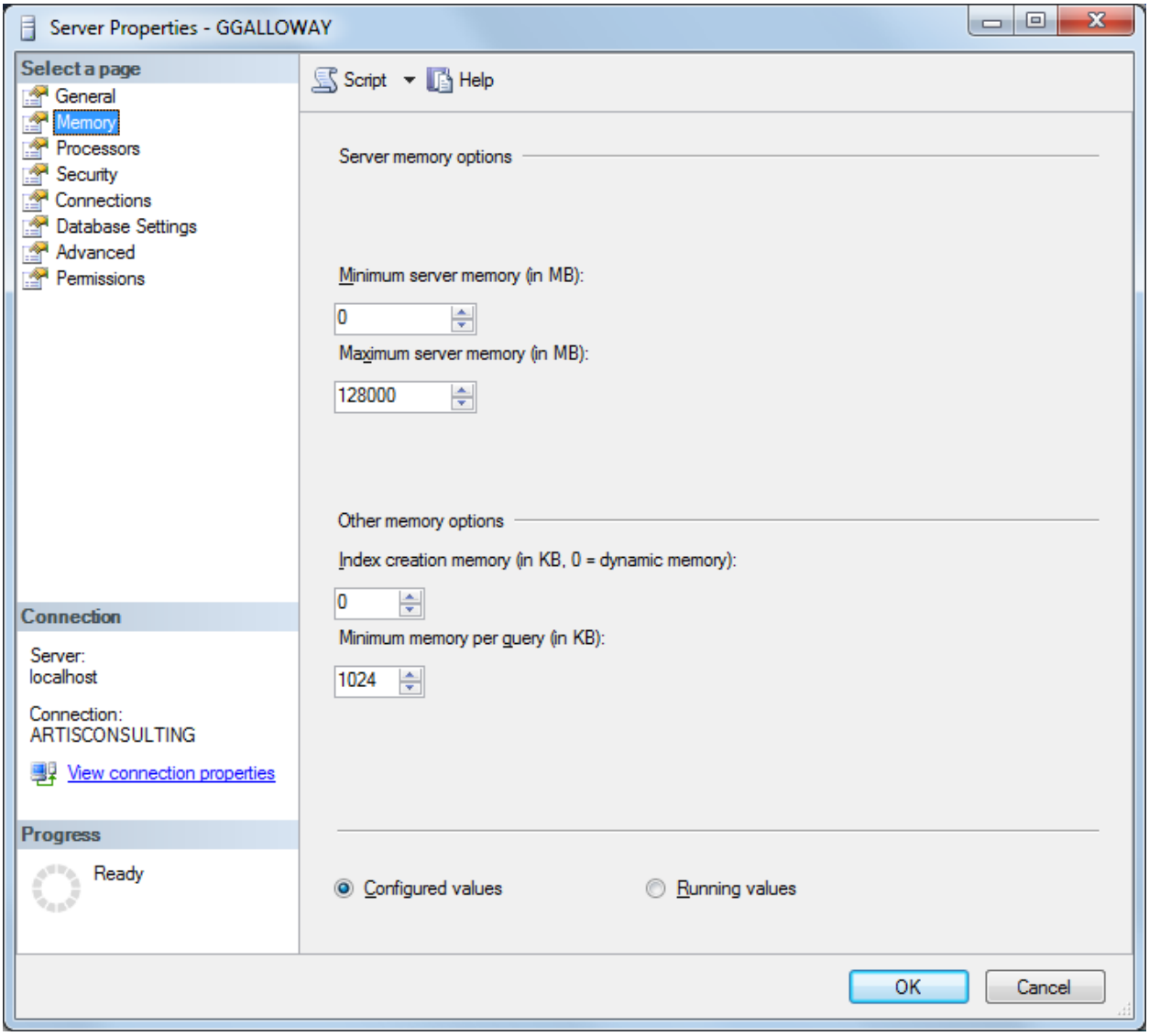


Figure 45. Server memory properties

Choosing the Memory Limit for Analysis Services

What should be the memory cap for Analysis Services? You can determine the approximate amount as follows:

- As described in the Hardware Sizing a Tabular Solution whitepaper, estimate the amount of memory needed to hold your data at rest.
- Detect the largest consumers of memory, as described below.
- Then allow another 2-3x for transactional processing, processing overhead, user sessions, caches, and the msmdsrv.exe process.

To explain why 2-3x memory is needed, consider the following. If the chosen processing methodology involves transactional processing (such as ProcessFull on the whole database), the operation will require

a second copy of all the VertiPaq data structures in memory until the transaction commits. Two copies are maintained so that the old data can be queried while processing is running, as seen in the following diagram. For example, if one processing transaction is a ProcessFull on a single table, then the old copy of the entire database will be in memory and the new copy of that one table being processed will be built in memory. Once processing completes and commits the transaction, the old data for the objects just processed is removed from memory. Thus, if no users will query your model during processing, consider performing a ProcessClear before processing as described above.

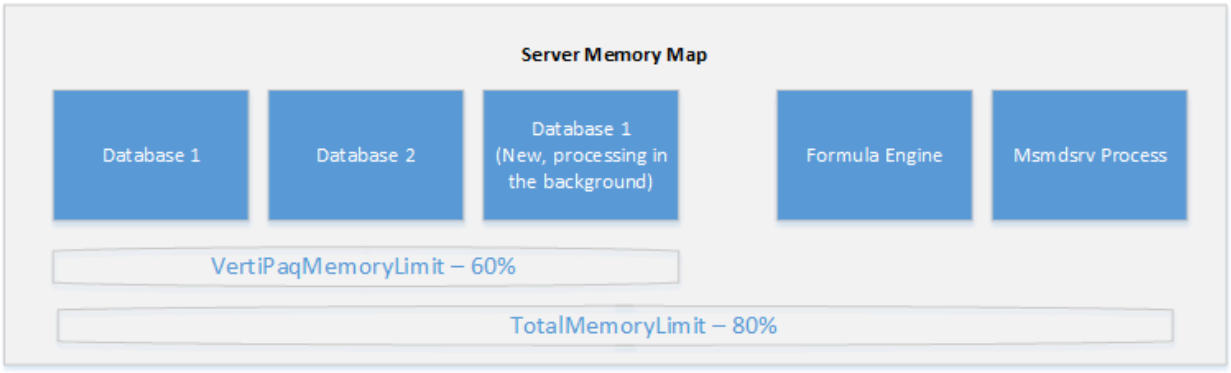


Figure 46. SSAS server memory map

If low on memory, consider studying best practices for reducing memory usage during processing and reduce model size by detecting the largest consumers of memory.

Multiple Instances in the Server Memory Plan

If multiple Analysis Services instances are deployed on the same server, we recommend that each instance have a memory cap so that one instance doesn't consume memory that another instance expects to be available.

Multiple Databases in the Server Memory Plan

On Analysis Services instances housing multiple databases, install enough memory on the server to allow all models to be loaded into memory at the same time. When planning the processing schedule for multiple databases, if tight on memory, consider staggering processing so databases are not being processed at the same time. If processing concurrently, then the 2-3x multiplication factor should be applied to each database being processed concurrently.

Development Servers and Workspace Servers

During model development, SQL Server Data Tools (SSDT) creates a workspace database so that the design-time experience is interactive and calculation results are visible without having to manually deploy the model to the server. If multiple developers edit the model, a separate workspace database is created for each developer. By default, these workspace databases are detached once SSDT is closed, leaving them on disk on the workspace server, but removing them from Analysis Services memory. However, if the developer suspends their laptop without closing SSDT, the workspace database will be left attached to the workspace server.

- Periodically look for workspace databases that aren't being used and which can be deleted or detached.
- Plan overall memory to account for workspace databases.
- Consider using a subset of data during development by leveraging views or filters.

An alternate approach is to outfit developer laptops with adequate RAM and processing power so that each developer can use the Analysis Services instance installed on their laptop as the workspace server, thus freeing up resources on the development server for deployed databases. There are also several advantages to developing against a local workspace server which Cathy Dumas discusses in Configuring a workspace database server (http://blogs.msdn.com/b/cathyk/archive/2011/10/03/configuring-a-workspace-database-server.aspx).

Adjusting Server Memory Limits Programmatically

Adjusting the memory limit for SQL can be automated with the following T-SQL script:

```
EXEC sys.sp_configure N'show advanced options', N'1'  RECONFIGURE WITH OVERRIDE
GO
EXEC sys.sp_configure N'max server memory (MB)', N'128000'
GO
RECONFIGURE WITH OVERRIDE
GO
EXEC sys.sp_configure N'show advanced options', N'0'  RECONFIGURE WITH OVERRIDE
GO
```

Adjusting the memory limit for Analysis Services can be automated with the following XMLA script:

```
<Alter ObjectExpansion="ObjectProperties"
  xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">
 <Object />
 <ObjectDefinition>
  <Server>
   <ID>SERVERNAME\INSTANCENAME</ID>
   <Name>SERVERNAME\INSTANCENAME</Name>
   <ServerProperties>
    <ServerProperty>
     <Name>Memory\HardMemoryLimit</Name>
     <Value>60</Value>
    </ServerProperty>
    <ServerProperty>
     <Name>Memory\TotalMemoryLimit</Name>
     <Value>55</Value>
    </ServerProperty>
    <ServerProperty>
     <Name>Memory\LowMemoryLimit</Name>
     <Value>40</Value>
    </ServerProperty>
    <ServerProperty>
     <Name>Memory\VertiPaqMemoryLimit</Name>
     <Value>35</Value>
    </ServerProperty>
   </ServerProperties>
  </Server>
 </ObjectDefinition>
```

```
</Alter>
```

Because memory limit changes can be scripted, one option on servers that are short on memory is to dynamically change the limit for different periods of the day. For example, during the ETL, raise the memory limit on SQL Server and lower it for Analysis Services. Then during processing, lower it for SQL Server and raise it for Analysis Services.

Detecting the Largest Consumers of Memory

If VertiPaq memory usage is approaching the VertiPaqMemoryLimit, detecting which objects are consuming the most memory may help the model developer reduce memory usage.

While there are several approaches for detecting large memory consumers discussed in the Hardware Sizing a Tabular Solution whitepaper, we recommend that you download and use a workbook created and published by Kasper De Jonge, an Analysis Services program manager (http://www.powerpivotblog.nl/what-is-using-all-that-memory-on-my-analysis-server-instance).

Memory usage dashboard for SSAS server instance:

Top 10 tables on the server instance by memory usage

Row Labels	Rank	Object Memory Usage MB
Test-ContosoRetail - FactSales	1	60.19
Test-ContosoRetail - Model	2	2.33
Test-ContosoRetail - DimProduct	3	2.10
Test-ContosoRetail - Measures	4	2.07
Test-ContosoRetail - DimDate	5	1.76
Test-ContosoRetail - DimStore	6	1.18
Test-ContosoRetail - DimPromotion	7	0.76
Test-ContosoRetail - DimChannel	8	0.48
Test-ContosoRetail - DimCurrency	9	0.48
Test-ContosoRetail -	10	0.10
Grand Total	**1**	**71.44**

Memory usage per Database

Row Labels	Object Memory Usage MB
Test-ContosoRetail	71.56
(blank)	0.00
Grand Total	**71.56**

Figure 47. Sample dashboard in tabular monitoring workbook

His workbook uses DMV queries to report memory usage by object, summarize the numbers by database, and allow drill down into the details.

Reducing Memory Consumption in High Cardinality Columns

Reducing the memory consumption of a model leaves more memory for queries, transactional processing, and caches. Reduced memory consumption usually leads to faster query and processing performance, too.

If the largest consumer of memory is an identity column that is not necessary for calculations or user queries, then delete that column. For other columns, consider how to reduce the number of distinct values in that column following the advice in Optimizing High Cardinality Columns in VertiPaq and Optimize Memory in PowerPivot and SSAS Tabular by Marco Russo. If importing a PowerPivot model into a Tabular instance, first run the PowerPivot Workbook Size Optimizer.

Another way to reduce memory consumption is to tweak the data types, if possible, to get better compression. For more information, see the section titled "Do datatypes matter?" in Section 4 for guidance on using the Currency datatype.

- Optimize your models from the beginning – get users to run the optimization wizard in PowerPivot.
- Clean up embedded data from linked tables. This data is loaded as part of the database script and can hurt performance if you have too many rows of copy-pasted data.
- Remove all identity columns not explicitly needed.
- Change data types if necessary to get better compression.

Other Best Practices

Avoid Co-Locating Tabular and Multidimensional Instances on the Same Server

While it is possible to install instances running in Tabular mode and in Multidimensional mode on the same server, and while this is acceptable for small solutions or projects with a small budget, if the budget is available, separate hardware is recommended. The reason is that different hardware is optimal for Tabular than for Multidimensional. For example, it is important to spend money on fast disks on a server hosting large Multidimensional models, but fast disks are not important for Tabular servers.

The following table summarizes the differences in requirements.

Table 9. Suggested specs for tabular vs. multidimensional servers

Hardware resource	Multidimensional	Tabular
RAM	Does not have to fit in memory but performs better if it does	Must fit in memory
RAM speed	Important	Crucial
Processors	Good processors	Fastest processors with largest L2 cache
Disks	SSD or fast disks important	Not important
Network speed	Important	Important
NUMA	NUMA-aware	Not NUMA-aware. Fast memory and processors is more important than lots of processors.

Disable Power-Saving Mode

Many servers ship with a default BIOS setting that throttles processor speed in order to save power. If your goal is achieving optimal performance rather than saving power, disable this setting, at least during hours the server is active. Specify the "performance BIOS setting" (or similar for your hardware vendor) when ordering new servers. For existing servers, look for processor performance or power-saving mode settings in the BIOS.

Also ensure the power settings in Windows are not in power-saving mode. For example, the following server is set to power-saving mode and should be changed to the "High performance" plan.

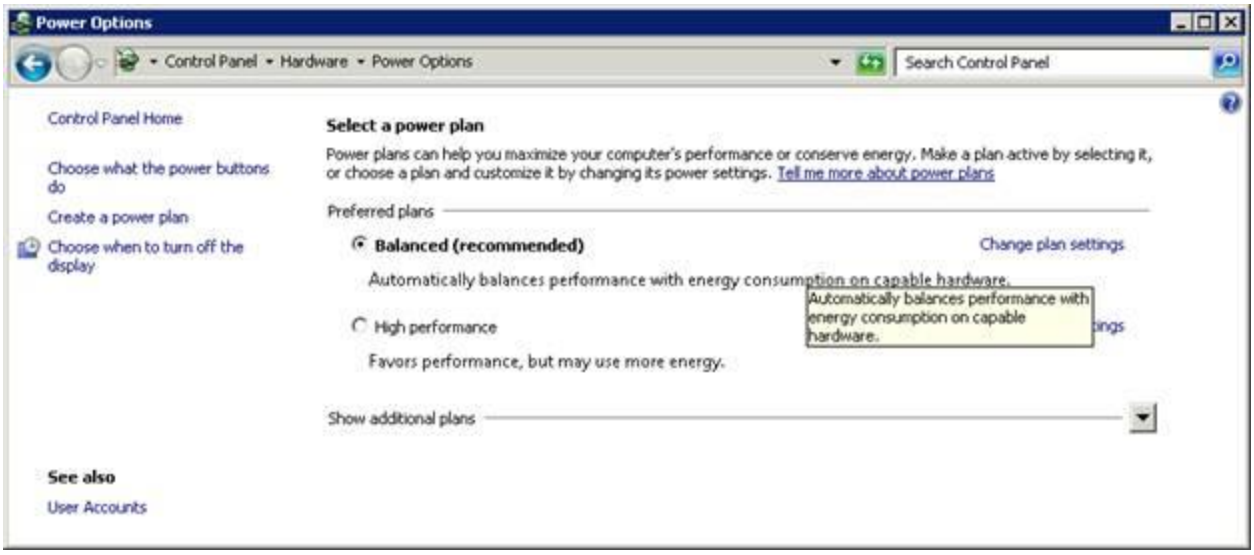


Figure 48. Modifying the power plan

The CPU-Z tool (http://cpuid.com) can be used to monitor processor speeds and detect power-saving mode. For example, on the following server, the processor specification speed is 3.47GHz (or 3553MHz). However, the core speed is just 1595MHz, which indicates that the server is in power saving mode and therefore the processors are not running at top speed.

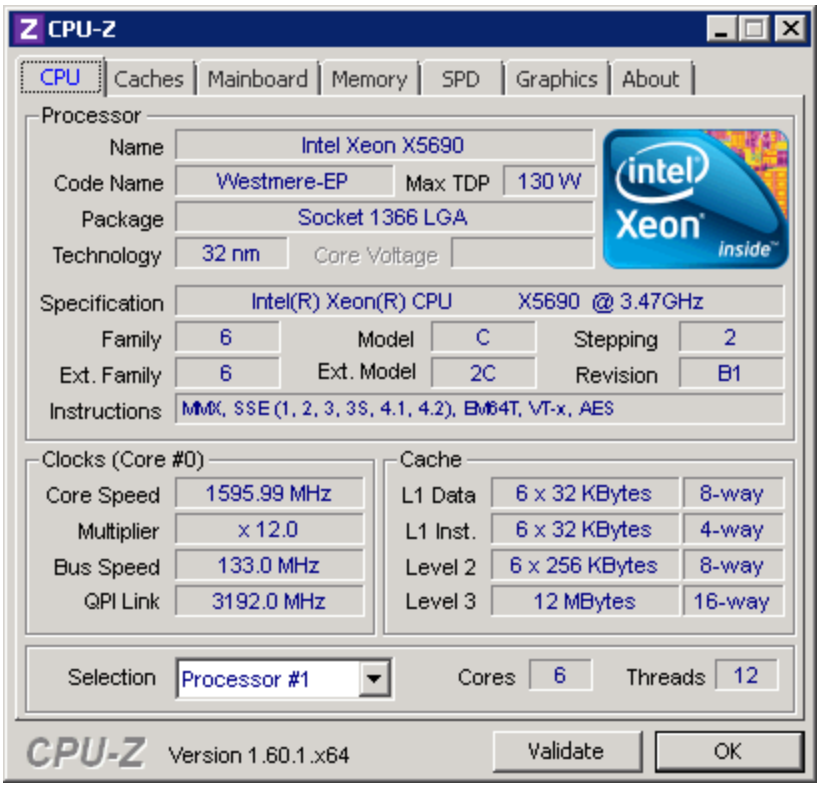


Figure 49. CPU-Z tool for monitoring processor speed

Scale Out

Optimizing calculations and scaling up a server can improve performance only so far. As more and more concurrent users query a model, scaling out to multiple servers will eventually become necessary. Luckily, scale-out options for Tabular models are identical to those for Multidimensional models, and you can easily find many helpful resources that discuss different scale-out architectures. Some examples are listed here:

Scale-Out Querying for Analysis Services with Read-Only Databases

Scale-Out Querying with Analysis Services Using SAN Snapshots

Analysis Services Synchronization Best Practices

The following high-level summary of scale-out architectures serves to introduce an administrator to the concepts and to distinguish how Tabular models may favor different scale-out architectures than Multidimensional models.

Scale-Out Reasons

There are a number of reasons to employ a scale-out architecture:

- Processing a Tabular model is an extremely resource intensive process which can consume nearly all CPU and memory resources on a server to the detriment of simultaneous user queries. Scaling out could separate the processing server from the query server, avoiding this problem.
- Processing a Tabular model requires an exclusive lock on the database at the end of processing. User queries can create a blocking problem where processing is waiting on a long-running query while all other users are waiting on processing to commit. Some scale-out architectures eliminate this blocking problem.
- In the case of a hardware failure, certain scale-out architectures can provide guaranteed high availability.
- In the case of a physical disaster such as a data center flooding, certain scale-out architectures can provide disaster recovery capabilities.
- Querying a server halfway across the globe can provide less than optimal performance. Scale-out architectures can help to host Analysis Services geographically near users, improving the user experience.
- As many concurrent users query Analysis Services, scale-out architectures provide the hardware resources necessary to service those queries.
- When a single server can't possibly load data fast enough, leveraging multiple servers in a staggered fashion can provide users access to fresher data.
- In a consolidation environment, dozens of independent Analysis Services databases can be consolidated onto a few powerful servers. This provides cost savings through economies of scale and it allows the administrator to be nimble in moving heavily used, resource intensive models to servers with more resources available.

After determining that a scale-out architecture can solve one of the business needs listed above, three primary decisions must be made in finalizing an architecture:

- Deciding how to copy the model.
- Determining the best approach to network load balancing.
- Selecting the hardware.

Scale-Out Model Copy Approaches

In order to scale out to multiple servers, all servers need to be able to access a copy of the Tabular model on disk in order to load it into memory. Unlike Multidimensional models, I/O performance is fairly unimportant for Tabular models, except in extreme real-time scenarios. However, a scale-out copy approach is still required:

- Analysis Services Synchronize command provides a built-in way of incrementally updating a model on a query server from the fresher model on a processing server. It physically copies the files that have changed and does this inside a transaction so that the older data can be queried on the query server while the **Synchronize** command is running. It does require an exclusive lock to commit. Performance of the **Synchronize** command is mainly determined by I/O and network performance. It is not very processor intensive, so it serves well when a scale-out architecture is being used to prevent processing from slowing down querying. However, it does require approximately double memory (depending on how many tables have been reprocessed since the last synchronize) to keep the old copy and the new copy of the model in memory until the new synchronize is committed. Physically copying the model to a second I/O subsystem can be

advantageous to I/O intensive Multidimensional models, however I/O performance is not a factor in query performance in Tabular models.

- Windows clustering in an active/standby configuration can provide automated failover in the event of a hardware failure. During failover, the SAN volume can be unmounted from one server and mounted on the other server, giving it nearly instant access to the Tabular model files on disk. This approach does require about a minute of model downtime during the failover. Unlike active/standby configurations where one server is idle and unused, active/active configurations have services running on each server. For example, the SQL Server might run on one server, and Analysis Services might run on the other. Given that Tabular models require the data be stored in memory, you should carefully calculate worst-case memory requirements when choosing an active/active clustering configuration.
- As described in Scale-Out Querying for Analysis Services with Read-Only Databases, one common approach for presenting a Tabular model to multiple servers is to attach a LUN as read-only to one or more servers. For Multidimensional models, this approach works well for calculation-heavy models with modest I/O requirements. For Tabular models, this approach works well since disk I/O is not necessary except when loading the Tabular model into memory after the service starts when the first user connects.
- As described in Scale-Out Querying with Analysis Services Using SAN Snapshots, you can leverage built-in SAN technology for quickly creating a "snapshot" (or virtual copy) of files. It does require the model be taken offline before the copy begins, causing model downtime unless the network load balancing layer sends queries to other servers during the copy.

Network Load Balancing Approaches

In order to scale-out to multiple query servers, network load balancing may be required. The following options should be considered:

- Hardware load balancers such as BIG-IP
- Software load balancers such as Network Load Balancing in Windows Server
- ASLB or Open ASLB is a customizable load balancing and redirection layer on top of Analysis Services HTTP connectivity, which can be very helpful in consolidation environments to ensure the connection string remains the same even if the model is moved to a different server.
- Client load balancers such as custom Excel add-ins which intelligently set the connection string to query the server with the freshest data

Any network load balancing solution should take into account that sessions in Analysis Services outlive individual connections. As described in Scale-Out Querying for Analysis Services with Read-Only Databases, the network load balancer should support cookie-based affinity to ensure all network traffic for a particular session is routed to the same Analysis Services server.

The following diagrams illustrate several scale-out topologies described above which include network load balancing.

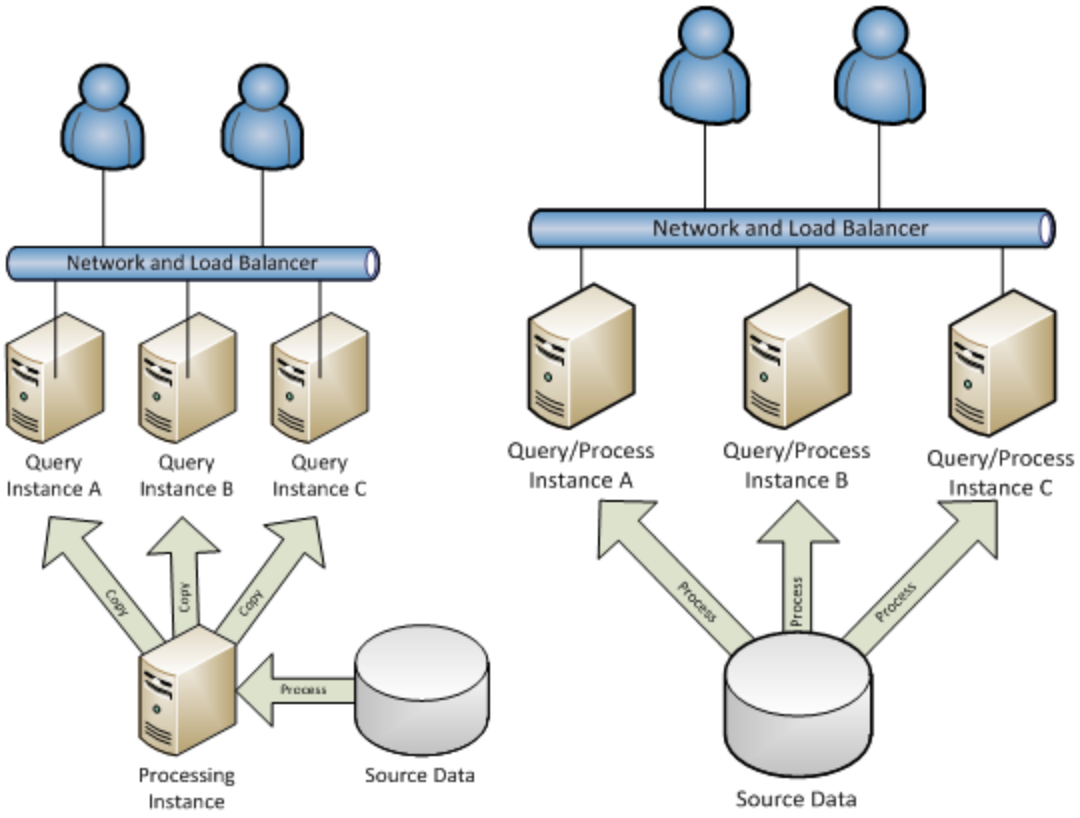


Figure 50. Two scale-out topologies

Additional Hardware for Scale-Out

Additional hardware is implicit in scale-out architectures. For a full treatment of hardware needs in Tabular models, refer to the whitepaper entitled Hardware Sizing a Tabular Solution.

Summary—Server Configuration and Tuning

This section discussed tuning a server to perform optimally for the particular models deployed. Once the latest service pack has been installed, server configuration settings should be studied and a server memory plan can be established. Power-saving mode and scale-out architectures were also discussed.

6. Conclusion

Although this paper describes tuning and troubleshooting strategies in considerable detail, the decisions that you make to optimize your tabular model are actually very straightforward:

- Choose the right hardware – optimize for single-processor speed rather than parallel processing or shared architectures.
- Avoid sharing the server with traditional applications.
- Configure the server to optimize for memory-intensive operations.
- Select the data used in the models wisely to avoid bottlenecks in processing and enhance compression.
- Prepare your data for efficient encoding and retrieval.
- Design your model and your DAX formulas to take advantage of the efficiencies of the engine.
- Set up a window for processing as you would in any data warehousing environment.
- Monitor usage and tune your formulas and model.

As always there are trade-offs, but in general optimizing tabular models is easier and requires less planning than does optimization of multidimensional models. Given some attention to these core principles of good data modeling, query tuning, and server optimization, we think you can achieve remarkable performance with solutions designed to take advantage of xVelocity technology.

For more information:

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

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.

Appendix

Resources

This section contain links to resources that were cited in the text or that were used in compiling this paper.

Tools and scripts

This section provides links to tools available on CodePlex and some utilities used by the authors.

[1] Memory usage dashboard for SSAS, by Kasper De Jonge

Analysis Services Program Manager Kasper De Jonge created a PowerPivot workbook that allows you to investigate the memory usage on your server instance.

The report can summarize memory usage of a Tabular or Multidimensional instance, and contains two dashboards: the first displays the top 10 tables on the server instance by memory usage; the second displays memory usage across the instance.

Excel 2010 (https://skydrive.live.com/view.aspx?resid=7F4E0559CC74581A!272&app=Excel)

Excel 2013 (https://skydrive.live.com/view.aspx?resid=7F4E0559CC74581A!1089&app=Excel)

[2] CodePlex SSAS Community Samples

Codeplex page that lists multiple code samples and projects related to SQL Server Analysis Services.

http://sqlsrvanalysissrvcs.codeplex.com/

[3] Codeplex: ASTrace

Utility that helps you capture an Analysis Services trace and log it into a SQL Server table. The table can be queried later or read using SQL Server Profiler. Versions available for 2008, 2008 R2 and 2012.

http://sqlsrvanalysissrvcs.codeplex.com/

[4] Codeplex: DAX Editor

Provides IntelliSense and syntax highlighting for DAX queries. This extension is helpful when writing complex DAX queries.

http://daxeditor.codeplex.com/

[5] CodePlex: DAX Studio

Community members Marco Russo, Darren Gosbell, and Paul te Braak created this tool to assist with DAX formula editing and troubleshooting. Includes integrated tracing and query execution breakdowns.

http://daxstudio.codeplex.com/

[6] Extended Events (xEvents)

SQL Server Profiler for use with Analysis Services is expected to be deprecated in an upcoming release, in favor of the more lightweight Windows Extended Events architecture. In general, although more work is required upfront for customization, the Extended Events framework uses fewer resources, and handles all the required events across both server and client applications.

Extended Events might be a less intrusive monitoring solution for tabular servers that are under already memory pressure.

Use SQL Server Extended Events (XEvents) to Monitor Analysis Services

Product team blog discusses the planned migration from Profiler to Extended Events.

http://blogs.msdn.com/b/extended_events/archive/2010/12/10/migrating-from-sql-trace-to-extended-events.aspx

This blog by Jonathan Kehayias compares the performance impact on SQL Server Database Engine of tracing vs. Extended Events

http://www.sqlperformance.com/2012/10/sql-trace/observer-overhead-trace-extended-events

Miscellaneous Reading and Reference

This section provides links to additional reading that was not directly cited in this white paper but might be useful when troubleshooting or tuning performance.

[7] DAX over Multidimensional

Documentation on the TechNet Wiki relating the recent release of DAX over multidimensional models. Note the mapping of objects between tabular and multidimensional in the IMBI schema.

http://msdn.microsoft.com/en-us/library/dn198310.aspx

[8] Troubleshooting Process Data (Books Online)

This article in Books Online provides tips on how to determine whether a data source has been processed and troubleshoot basic processing issues.

http://technet.microsoft.com/en-us/library/gg492128.aspx

[9] How to build a memory efficient data model

Senior Program Manager Dany Hoter provides some tips on how to build an efficient data model using PowerPivot in Excel 2013 that are very applicable to tabular models too.

http://office.microsoft.com/en-us/excel-help/create-a-memory-efficient-data-model-using-excel-2013-and-the-powerpivot-add-in-HA103981538.aspx

[10] 2012 OLAP Performance Guide

No separate performance guide will be released for multidimensional models built using SQL Server 2012 Analysis Services. For general guidance, see the 2008 R2 performance and operations guides. Performance related enhancements to Multidimensional models in SQL Server 2012 include:

- 4GB string store limitation lifted
- support for more than 64 processors and NUMA and ability to affinitize to processors
- EnableRolapDistinctCountOnDataSource server setting
- Calculations using named sets can run in block computation mode in most cases

SQL Server 2012 SP1 also includes the following performance related enhancements:

- Improved performance on more VBA functions for use in MDX queries
- Enhancements to common Excel PivotTable scenarios like label filters, value filters, top N filters, and complex DrilldownLevel expressions

SQL Server 2012 SP1 CU4 also includes the following enhancements:

- Power View connectivity for Multidimensional models
- Subselects no longer prevent caching in most scenarios

SQL Server 2012 SP1 CU5 has also been released, so if you were going to apply and test SP1 CU4 in your environment, please consider applying and testing SP1 CU5 or future cumulative updates instead.

[11] 2008 R2 Operations Guide

http://msdn.microsoft.com/en-us/library/hh226085.aspx

[12] 2008 R2 Performance Guide

http://sqlcat.com/sqlcat/b/whitepapers/archive/2011/10/10/analysis-services-2008-r2-performance-guide.aspx

[13] Hardware Sizing a Tabular Solution

This white paper, released in 2013, provides guidance for estimating the hardware requirements needed to support processing and query workloads for an Analysis Services tabular solution.

http://msdn.microsoft.com/en-us/library/jj874401.aspx

[14] White paper on DAX Query Plans from Alberto Ferrari

http://www.sqlbi.com/topics/query-plans/

[15] Using Hadoop data in tabular models via the ODBC driver

The Microsoft ODBC driver for Hive enables users to access HDInsight Hive datastores through Microsoft's various client BI applications.

The driver is supported on the following operating system versions:

- Windows Server 2008 R2
- Windows Server 2012
- Windows 7 (32-bit and 64-bit)
- Windows 8 (32-bit and 64-bit)

On 64-bit systems, only 64-bit driver needs to be installed.

Connectivity from and retrieving result sets into the following applications has been validated and is supported by this driver:

- SQL Server 2008 R2 SP2 Analysis Services in Tabular mode
- SQL Server 2012 Analysis Services in Tabular mode
- Excel 2010 (and PowerPivot)
- Excel 2013 (and PowerPivot)

For all other scenarios, we recommend that you use the ODBC drivers or toolkit provided by our partners:

http://simbaodbc.com/

http://www.businesspress24.com/pressrelease1240499.html?utm_source=dlvr.it&utm_medium=twitter

Known Issues and Bug Fixes

This section provides a list of issues related to performance monitoring that were unresolved at the time of this writing, or the authors could not verify a fix.

Issue 1. Vertipaq Paged KB always returns zero in SP1

Problem: The following performance counters always show zero in SQL Server 2012 SP1:

MSOLAP:Memory\VertiPaq Nonpaged KB
MSOLAP:Memory\VertiPaq Paged KB
MSOLAP:Memory\VertiPaq Memory-Mapped KB

Expected behavior: The expected behavior for these counters is described in section 5.
Cause: Bug.

Workaround: None

Fix: Bug fix to be released in a future cumulative update or service pack.

Issue 2. Tabular mode does not perform well in NUMA

Problem: Tabular models do not provide the expected performance, or even the usual adequate performance, on machines that are NUMA-enabled. It is also difficult to determine how memory is allocated across NUMA nodes and how much of the query time is related to foreign memory access slowdowns.

Cause: Tabular models are not NUMA-aware.

Workaround: The extra cores and memory that are available on NUMA machines can mitigate the reduced performance to some degree, so you can sometimes get better throughput. However, typically performance is somewhat reduced in a NUMA environment. This is because neither the Formula Engine nor the Storage Engine will modify execution when running on NUMA machines.

Users should perform an evaluation in each scenarios and determine which hardware meets their needs, based on both performance and scalability results. For tabular models, you can get increased performance by choosing hardware with a fast memory bus, large L2 cache, and fast processor speeds, rather than investing in very large NUMA-aware servers.

When choosing between systems that have the same number of cores and RAM, pick a non-NUMA system if you can.

Issue 3: SQL Server Management Studio (SSMS) ProcessFull UI bug

Problem: When you use SQL Server Management Studio and perform ProcessFull on a Tabular model database, the user interface incorrectly displays ProcessRecalc when you click the Details link. Despite what is shown, it actually does perform a ProcessFull.

Expected behavior: User interface displays the correct processing status.

Cause: Bug

Fix: Unknown.

Issue 4: Other known query performance issues

Section 3 discusses several known query performance issues in Power View, Excel, and other general query performance issues. Short-circuit evaluation can cause worse query performance than eager mode evaluation, but currently DAX does not support anything but short-circuit evaluation of functions. Querying at the lowest grain still performs a GROUP BY in VertiPaq. Some currency arithmetic requires a callback. Certain MDX queries against large fact tables use large amounts of memory. Similarly, certain DAX queries against extremely large dimension tables can use large amounts of memory.

For more detailed information on these issues, see section 3.

Issue 5: No multi-user settings provided for VertiPaq

Problem: Analysis Services in tabular mode is not optimized for multiuser scenarios. As a result, an expensive query could starve all other user queries of resources.

Cause: By design, each user's query should be so fast that the engine should just finish one query before moving on to the next on. This design gives the CPU and caches and threads the least amount of context switching.

Workaround: Design queries so that they do not run too long. Also, note that the server properties to support multiple users in Multidimensional mode do not apply to Tabular mode.