SSAS Tabular as Analytical Engine

USING TABULAR AS PART OF YOUR SOFTWARE SOLUTION

produced by

sqlbi

SSAS Tabular as Analytical Engine

Many companies have adopted SQL Server Analysis Services Tabular as the analytical engine for their product or service. Some have created a single large database deployed manually and queried by their front end. Others provide a multi-tenant service—creating databases on demand, and then accessing them through standard clients. Other scenarios use a mix of these two approaches.

The point of view is unlike that of creating a "classic" Business Intelligence solution. When the solution serves other software instead of a human user, the challenges are completely different. This article provides several reasons why Tabular could be the right choice for the analytical engine embedded in a service or an application.

Author: Marco Russo

Published: Version 1.0 – September 2014

Contact: marco.russo@sqlbi.com – www.sqlbi.com

Summary

This article describes pros and cons of using SQL Server Analysis Services Tabular as the analytical engine in a service or application, based on the experience of companies who have adopted it.

Contents

Introduction

In these early years of SQL Server Analysis Services Tabular (SSAS Tabular, or Tabular hereinafter) adoption, SQLBI has helped several customers in their first implementation using the new xVelocity in-memory engine and the DAX language. Existing customers of SSAS Multidimensional (Multidimensional hereinafter) have many reasons for not adopting Tabular: missing features, skills shortage, lack of tools, legacy (!) OLAP ecosystems. This is expected and very natural, as Multidimensional is complementary to Tabular and is not going to disappear. Thus, users have adopted Tabular mainly for new projects—especially in companies that had not used Multidimensional before.

However, in this adoption process, something surprising happened. My guess is that Microsoft was surprised by it as well.

I have seen many companies adopt Tabular as the analytical engine for their product or service without actually creating a "BI project" in the canonical way (gather requirements, design a prototype, get feedback, improve the model, and loop until it is done). Instead, they integrated Tabular features into their existing software as a back-end server for their analytical needs. In other words, users do not even know that Tabular is powering their reports.

The usage may vary: some companies created a single large database deployed manually and queried by their front end (this made it necessary to write their own DAX query builder). Others provide a multi-tenant service, creating databases on demand, and then offering access them through standard clients (such as Excel, Reporting Services, and other third-party tools). In other scenarios that we have helped to build, companies use a mix of both approaches.

Until today, we have never been proactive in promoting Tabular for such scenarios. Companies contacted us when they were looking for a solution to a specific problem—specifically, looking for an engine for their product or service with good performance and low maintenance cost. These companies usually considered Multidimensional, but chose not to adopt it for two main reasons: MDX complexity (from the developers' point of view) and performance. Not to say that Multidimensional is slow, but creating an efficient Multidimensional database requires a major effort: it is hard to automate and it is complex to optimize when an application automatically generates the model. Other common issues are DISTINCT COUNT measures, many-to-many relationships, and leaf-level calculations: all of these are features where obtaining good performance is a challenge.

Often, when customers approached us asking for an engine, we developed both Multidimensional and Tabular proof of concepts. In nearly 90% of these cases, the customer chose Tabular because of its effectiveness and simplicity. One of these customers is inContact, for which we provided a full case study in this paper: "Using Tabular Models in a Large-Scale Commercial Solution."

A few times, we had requests for proof of concept of a Tabular solution by companies that were already using Multidimensional (and finding some limitations because of their requirements) or that had discarded Multidimensional after building a prototype. They mainly asked whether the new analytical engine they saw at a conference or read about in articles and blogs (because Microsoft never really pushed Tabular adoption) was good enough for their requirements. Our discovery was that yes, most of the time it was much more than "good enough."

In this article, we would like to share in more detail some of the reasons why Tabular could be the right choice for the analytical engine embedded in a service or application. The point of view is unlike that of creating a “classic” BI solution. When the solution serves other software instead of a human user, the challenges are completely different. We will try to help you make an initial evaluation, highlighting that, before taking any decision, you need to create a proof of concept for your specific scenario (by the way, SQLBI can help you do that, but—trust me—this is the only commercial you will find here).

Choosing an Embedded Analytical Engine

Among many other products, Analysis Services is a possible choice for an embedded analytical engine in an application or service. Moreover, it provides two different engines and model types: Tabular and Multidimensional. To make your choice, you need to perform a deep analysis of the requirements for each specific scenario. In this section, you can find a description of the general requirements to consider and the reasons why Tabular is the best choice in many cases.

Requirements

Software products and services that offer analytical features typically need some of the following elements:

- A semantic model that allows a generic client to browse data and dynamically create queries.
- High scalability for concurrent users' workload.
- Top query performance when aggregating huge volumes of data.
- A query language that simplifies the automated generation of queries.
- The capability to script and automate several database activities (model definition, provisioning, backup, and maintenance).
- Low total cost of ownership (TCO) to reduce the costs over time of their solution.

Most applications use a relational database as a repository for their data. Often this becomes the data source for all of the reports and queries. Normally this is not a brilliant idea, because the application database is seldom designed to respond quickly to analytical queries. Nevertheless, there are some aspects (skills, licensing costs, hardware, and maintenance) for which using the same relational engine looks like a good choice. Here are some of the most important aspects to consider when deciding which database to use for your application analytics:

- Relational databases do not have a **semantic model**. The semantic model is needed by query builders (either Microsoft Excel or the application's query builder) to let users interact with the database. Without a semantic model, you may need to build an external data model to drive the client application in data navigation within the relational model.
- Serving many concurrent users requires a good cache policy. Usually, relational databases do not cache the intermediate results of a query; they mainly use the cache just to lower the number of I/O operations. Once data is in memory, a high concurrent workload might create a bottleneck in RAM access. Microsoft SQL Server's cache can have a **scalability** issue with high concurrent workload, even when using column store indexes.
- Relational databases offer index technologies that improve **performance** when a query aggregates data. For example, column store indexes in Microsoft SQL Server might be able to satisfy such a requirement. In an application database, however, adding indexes might create performance issues

for the application, because indexes need to be updated whenever the underlying data changes. Thus, adding indexes until the analytics are fast enough is a dangerous technique to use.

- SQL is a flexible and general-purpose **query language**. There are tons of existing libraries to generate queries automatically (for example, .NET Framework uses LINQ). However, the SQL language can be verbose and hard to optimize for the query engine, especially when an object-relational mapping (ORM) library is used to generate the SQL query.

- Standard SQL includes a rich set of statements to generate the data model (Data Definition Language, DDL). Product-specific extensions to SQL are available to manage a SQL Server database lifecycle. Several database products, such as Microsoft SQL Server, also include custom APIs for other languages (e.g., C#) to **automate deployment**, backup, and maintenance. Thus, a product like Microsoft SQL Server is already compliant with scriptability/automation requirements.

- In order to obtain good performance for queries that aggregate huge volumes of data, a relational database uses additional structures that duplicate data in a format that makes query execution faster (such as indexes). These structures require **maintenance** over time, which might include additional operations (such as periodic reviews of the index schema because distribution of data changes over time). This results in a "hidden" additional **cost of ownership**.

This is a recap of these considerations in a small matrix:

Feature	SQL Server
Semantic model	N/A
High scalability	Good, but SSAS is better
Query performance for huge volumes	Fair, but SSAS is far better
Query language	Good, but verbose
Automated handling of the database	Good
Total cost of ownership	Good, but might have hidden costs

As you can see, sometimes there is the need for a different analytical engine than the standard relational database used to store an application's data.

Why Analysis Services Tabular

After considering the requirements for an analytical engine, Analysis Services is usually a good candidate, even just for licensing reasons: many SQL Server licenses also include Analysis Services, if installed on the same server. However, the licensing cost is only part of the equation. Analysis Services also offers a semantic model with a cache system that provides performance and scalability. It has two model types, with different internal engines: Tabular and Multidimensional. Analysis Services 2012 introduced the Tabular model, whereas the Multidimensional model has been around since Analysis Services 2005.

Other than using different engines, the main differences between Tabular and Multidimensional are the languages and tools used to define the models. A complete description of the differences is outside of the scope of this article; more details are available in the "Choosing a Tabular or Multidimensional Modeling Experience in SQL Server 2012 Analysis Services" white paper (http://msdn.microsoft.com/en-us/library/hh994774.aspx).

Several companies that we have worked with have adopted SQL Server Analysis Services Tabular as the analytical engine for their products or solutions. The reasons why Tabular was preferred to Multidimensional are performance, maintenance, cost of ownership, and flexibility in data model design. This flexibility is very important considering the requirements to automate the provisioning of the data model. One of the limits of Tabular adoption is the licensing cost, because it requires a Business Intelligence or Enterprise edition of SQL Server. The cheaper Standard edition includes Multidimensional but not Tabular. For this reason, to reduce the final price of their products, many small independent software vendors (ISVs) choose Multidimensional over Tabular, even if Tabular could be a better choice from a technical point of view.

The following section presents the experiences of several companies who chose Tabular as the analytical engine for their products or solutions.

Tabular Experiences from the Field

In the previous section, we explained why choosing an Analysis Services data model can be a good option for software analytics. This section explores the issues of an embedded analytical engine in more detail, sharing the experience of early adopters of Analysis Services Tabular and some of the reasons why they made this choice.

Automated Deployment

In multi-tenant scenarios, the system provides a database for each customer. Moreover, if the customer has the option of customizing entities (for example, adding custom columns or calculations), you end up having a different schema for each database. Classic relational database solutions used for this requirement, such as entity-attribute-value (EAV) models or XML columns, are not compatible with Tabular and the requirements of a semantic model, which must clearly identify each attribute.

As of SQL Server 2014, Tabular does not have a real data definition language (DDL). An application can deploy a database either by creating XMLA statements or by using the Tabular AMO 2012 library. With it, you can use C# and any other .NET language to automate the deployment operation.

A DDL for Tabular would be very welcome, because it would make it easier to compare the differences between databases in different tenants by just looking at the differences between two human-readable text files. Unfortunately, as of now, this is not feasible, but BISM Normalizer is a useful tool that offers a similar feature.

Dynamic Query Generation

Many applications have embedded features that allow users to create custom reports. This requires the dynamic generation of queries sent to the database engine. SQL is the language of choice for a relational database, whereas Tabular offers DAX and MDX.

The most common choice for Tabular is DAX. It is more similar to SQL than MDX, and it is a functional language, which makes it easier to compose and encapsulate operations in sub-queries. The LINQ to DAX library (http://linqtodax.codeplex.com/) simplifies the generation of a query tree and the production of the DAX code in a .NET application. One of our customers (Dealogic) created this library and then released it as an open source project on CodePlex.

No Maintenance Required

If an application needs to generate a SQL Server database schema automatically, then it faces a further complexity: creating indexes and tracking statistics to maintain good performance. With automated deployment, we expect that there is no database administrator (DBA) working on planning and implementing maintenance. Thus, the automation must include this activity too. A similar level of complexity would be present even with Multidimensional—at least for the definition and maintenance of aggregations that might change for different databases.

Tabular, on the other hand, does not have indexes or aggregations. The Tabular engine gets statistics on demand while running a query and requires no additional data structures. These internal queries are very fast and do not interfere with the overall query performance, thanks to the optimized in-memory storage engine. The important factors that determine system performance are the model design and the query structure. Different queries producing the same result might have different query plans with different response times, so the only possible optimization is using the best DAX constructs, because there are no indexes or external "helps" that can produce a different query from the same DAX query.

Thus, there is no need to plan manual or automatic maintenance of auxiliary structures used as performance optimization tools, resulting in a lower TCO for Tabular compared to SQL Server or Multidimensional.

Performance

Every query in Tabular might perform a complete scan of the columns involved in the request. A small internal cache avoids running the same storage engine request multiple times within the same DAX query. Even when there is a complete scan of a column, the execution time of a storage engine request is usually in the milliseconds range because of optimizations based mainly on data compression, which lower the amount of RAM read for each request.

Thanks to these continuous scans, leaf-level calculations on data do not affect performance (unlike what happens in Multidimensional). Tabular always works at the most granular level and its design provides outstanding speed.

Applications that benefit more from the Tabular engine have the following characteristics:

- Extensive use of leaf-level calculation, such as simulation or other dynamic calculations based on external parameters of algorithms applied row by row on large tables.
- Distinct count calculations on any column in any table of the data model.
- Calculations over many-to-many relationships (also combined with distinct count calculations).
- Free-form queries, where filtering and grouping can happen on any combination of columns and values for any query.

That said, there are two main points of attention for performance:

1. The **data model design** affects the compression, which affects query performance. For example, high cardinality columns should be avoided (as in the case of timestamp, not required for analysis) or split (as in the case of a datetime value, which works better if stored in two columns—one for the date and the other for the time).
2. **Quality of DAX code** is key for both queries and measures. Different formulations of DAX queries and calculations might produce the same result, but with different speed. In fact, the query plan generated by the Tabular engine is very sensitive to the quality of the DAX code, because its query

optimizer is not as mature and advanced as the SQL Server one. Adopting best practices and generating DAX code in a smart way is essential to obtain the optimal level of performance. Being able to read DAX query plans and other events provided by the profiler is an important skill required to optimize measures and queries.

Tabular usually provides good performance, but it is always important to plan tests and optimizations as part of the development process, because of the impact this could have on the database design. Since optimization does not depend on additional structures, it is important that you do not delegate this activity to a support team after model deployment. Instead, you should constantly check the impact on performance of each modeling decision.

Bottlenecks

In our experience, you can solve most of the performance issues of a Tabular model by rewriting the DAX expression of formulas and queries. Experience and education usually train developers so that this type of optimization is not necessary, once they learn to write DAX formulas in the right way at first draft.

With that said, two issues may require more complex optimizations based on the specific design of the data model:

- High-cardinality columns
- Complex formula engine calculations

High-Cardinality Columns

As a general rule of thumb, a high-cardinality column has more than a million unique values. High-cardinality columns affect performance of several operations in a query. In general, any scan of such a column tends to be slow because of its relatively low compression rate. In particular, this usually affects distinct count calculations and relationships between tables based on high-density columns. But even a simple sum of high-cardinality columns is much slower than the same operation on a low-cardinality column for the same number of rows in a table.

A distinct count calculation may be relatively slow compared to other calculations such as sum, average, min, and max. The release of Cumulative Update 9 for Analysis Services 2012 Service Pack 1 greatly alleviated this issue (also documented in KB2927844). It is important to use a build of Analysis Services greater than or equal to 11.00.3412 for Analysis Services 2012 and greater than or equal to 12.00.2342 for Analysis Services 2014 (which has implemented this fix in Cumulative Update 1 for RTM).

High-cardinality columns can be still a performance issue when they define a relationship between two tables. In this case, low selective queries might show slower performance. For example, consider a database that has sales transactions in a fact table related to a dimension with 10 million customers. Calculating sales for male customers will require filtering 5 million customers over the fact table, which may be slow. However, this type of issue does not affect queries that filter a low number of customers (for example, thousands of customers living in a particular zip code).

Complex Formula Engine Calculations

The DAX engine evaluates queries using an architecture that includes a formula engine and a storage engine. The storage engine (also known as VertiPaq or xVelocity in-memory analytical engine) performs simple calculations leveraging a scalable multi-threaded architecture. The formula engine executes more complex operations (especially filters and calculations) in a single-threaded environment.

When a DAX query requests a big effort of the formula engine, it can result in lower performance. When this happens, it is usually necessary to rewrite the DAX query to obtain a different (and faster) query plan by moving most of the computation to the storage engine or by reducing the computation complexity. This type of DAX optimization might be possible by using different DAX syntax or by adapting the data model to produce a better query plan. However, these performance issues can be hard to solve in a system that generates the data model and/or the DAX queries automatically. The lack of some features in DAX (such as temporary tables, or set operations like union, except, and intersect) makes it difficult to express certain operations in an efficient way.

Development Tools

The development environment for a Tabular model is the SQL Server Data Tools for Business Intelligence. This environment requires the use of a user interface for creating each measure and calculated column in the data model. The entire model is a single XML file, which makes it hard to compare different versions of the same project. Moreover, hosting the full project in a single file is a big obstacle if different programmers need to edit the same data model at the same time. Finally, editing DAX formulas in Visual Studio is not very productive and there is a need for improvement in future releases.

The community has created several tools to improve productivity in Visual Studio and in testing/debugging Tabular models:

- **BIDS Helper**: a Visual Studio add-in that improves productivity when editing Tabular models. It provides access to several features implemented in the Tabular engine but not available in the Visual Studio user interface (for example, drillthrough, translations, display folders, and editing of HideMemberIf property) - http://bidshelper.codeplex.com/

- **BISM Normalizer**: a Visual Studio extension that helps when comparing two Tabular databases, enabling the creation of scripts with changes between two versions - http://visualstudiogallery.msdn.microsoft.com/5be8704f-3412-4048-bfb9-01a78f475c64

- **DAX Studio**: an Excel add-in for Power Pivot and Tabular that provides an object browser, editor, and execution environment for DAX queries - http://daxstudio.codeplex.com/

- **DAX Editor**: a Visual Studio extension that simplifies editing of DAX measures in a Tabular project - http://visualstudiogallery.msdn.microsoft.com/9b18b6f7-a3b8-4adc-af82-4b4ec23f98b6

- **DAX Formatter**: an online tool that formats any DAX expression (query, measure, and calculated column) applying SQLBI code formatting guidelines - http://www.daxformatter.com/

These limitations usually do not affect companies that adopt Tabular as an embedded analytical engine, because most of the time the software/service generates and deploys the data model automatically. Most of these companies use Visual Studio only to create the "reference" data model and/or a prototype. Important tools to automate the provisioning of a Tabular model are:

- **Tabular AMO 2012**: a .NET library exposing an object model to manipulate a Tabular model - https://tabularamo2012.codeplex.com/

- **Tabular Database PowerShell Cmdlets**: a library of PowerShell cmdlets (based on Tabular AMO) that automates Tabular model manipulation from PowerShell scripts - http://tabularcmdlets.codeplex.com/

- **XMLA:** the lowest-level layer for manipulating an Analysis Services database. The Tabular AMO library internally generates XMLA commands. Using XMLA directly does not have any dependency on external libraries.

Hardware

The hardware choice for Tabular is a very important aspect for the performance of the database. Tabular requires a fast CPU with a large amount of L2 cache. Because the formula engine executes operations in a single thread, it is better to have eight cores at 3.8 GHz than sixteen cores at 2.0 GHz. The RAM speed is very important also. Tabular does not make any use of I/O during queries, so storing data in the L2 cache is very important (a large L2 cache is better) and access to RAM is the new I/O bottleneck. For this reason, the ideal hardware for Tabular (as of July 2014) has a clock speed of at least 3.30 GHz, at least 2 Mb of L2 cache per core, and mount memory with a speed of at least 1866 MHz.

Considering that Tabular is very sensitive to CPU clock and RAM speed, and does not produce a significant I/O workload, you can host it on faster hardware that uses a less expensive I/O system compared to the typical hardware used in data centers.

Tabular runs just fine on virtual machines. It is important to have the RAM committed to the virtual machine to avoid paging. By adopting a multi-tenant architecture, it is possible to run Tabular in different instances on different virtual machines, with maximum flexibility in workload distribution across the available hardware (in this situation, do not forget to evaluate the consequences on the licensing cost). In any case, even in a virtual machine it is very important to know the CPU clock and the RAM speed of the host system. Moreover, running Tabular on virtual machines hosted on a larger server, which is normally a NUMA server, you need to verify that a single NUMA node hosts the virtual machine to avoid memory access issues (i.e., bad performance).

Good white papers about this topic are "Hardware Sizing a Tabular Solution (SQL Server Analysis Services)" and "Using Tabular Models in a Large-Scale Commercial Solution."

Conclusion

If you have an application or a service that stores data and provides embedded analytical and reporting features, you might consider the adoption of SQL Server Analysis Services Tabular to get better performance, higher scalability, and the benefits of a semantic model that can be queried by standard analytical tools such as Excel and Power View.

Although many features and improvements are still possible, Tabular is a technology where Microsoft is all in, already using it in Power Pivot and as the pillar of the entire Power BI platform. Many of the missing tools are a minor issue for developers looking for a platform open to development instead of a product more "closed" to extensions and integration.