

# Analysis Services Many-to-Many Dimensions: Query Performance Optimization Techniques

SQL Server Best Practices Article

Writers: Dan Hardan, Erik Veerman, Carl Rabeler

Contributors: Sanjay Mishra, Greg Galloway

Published: December 2007

Applies To: SQL Server 2005 Service Pack 2

Summary: Many-to-many dimension relationships in SQL Server 2005 Analysis Services (SSAS) enable you to easily model complex source schemas and provide great analytical capabilities. This capability frequently comes with a substantial cost in query performance due to the runtime join required by Analysis Services to resolve many-to-many queries. This best practices white paper discusses three many-to-many query performance optimization techniques, including how to implement them, and the performance testing results for each technique. It demonstrates that optimizing many-to-many relationships by compressing the common relationships between the many-to-many dimension and the data measure group, and then defining aggregations on both the data measure group and the intermediate measure group yields the best query performance. The results show dramatic improvement in the performance of many-to-many queries as the reduction in size of the intermediate measure group increases. Test results indicate that the greater the amount of compression, the greater the performance benefits—and that these benefits persist as additional fact data is added to the main fact table (and into the data measure group).

### Copyright

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

 2007 Microsoft Corporation. All rights reserved.

Microsoft and SQL Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

### Table of Contents

[Introduction 1](#_Toc185822016)

[Overview of Many-to-Many Relationships in SSAS 1](#_Toc185822017)

[Aggregation Optimization Technique 5](#_Toc185822018)

[Intermediate Measure Group Partitioning Optimization Technique 7](#_Toc185822019)

[Matrix Relationship Optimization Technique 8](#_Toc185822020)

[Compression and the Matrix Key 8](#_Toc185822021)

[Implementing the Matrix Optimization 10](#_Toc185822022)

[Relational Data Warehouse Implementation 10](#_Toc185822023)

[Cube Design 13](#_Toc185822024)

[ETL Implementation 16](#_Toc185822025)

[Test Objective 20](#_Toc185822026)

[Test Methodology 20](#_Toc185822027)

[Scaling Up the AdventureWorksDW Relational Data Warehouse 21](#_Toc185822028)

[Designing Multiple Cubes 22](#_Toc185822029)

[Cube Designs 22](#_Toc185822030)

[M2M Test Queries 23](#_Toc185822031)

[Test Results and Observations 25](#_Toc185822032)

[Scenario 1 25](#_Toc185822033)

[Scenario 2 28](#_Toc185822034)

[Scenario 3 30](#_Toc185822035)

[Scenario 4 32](#_Toc185822036)

[Conclusions 34](#_Toc185822037)

[Appendix A: Aggregation Percentage 36](#_Toc185822038)

[Appendix B: Calculating the Reduction % 37](#_Toc185822039)

[Appendix C: Cube Details 39](#_Toc185822040)

[Scenario 1 39](#_Toc185822041)

[Scenario 2 40](#_Toc185822042)

[Scenario 3 41](#_Toc185822043)

[Scenario 4 43](#_Toc185822044)

#### Introduction

In typical design scenarios, fact tables are joined to dimension tables via many-to-one relationships. Specifically, each fact table row in a typical design scenario can join to only one dimension table row, while each dimension table row joins to multiple fact table rows. Many-to-many (M2M) design scenarios occur when a fact table row can potentially join to multiple dimension table rows, with the join performed through the use of an intermediate fact table. In Microsoft® SQL Server™ 2005 Analysis Services (SSAS), this M2M design scenario is modeled through the use of an intermediate measure group and an intermediate dimension that link the data measure group and the M2M dimension (these terms are explained in more detail in the next section). In SSAS, when you query the data measure group by the M2M dimension, SSAS resolves the query by first retrieving the requested records from the appropriate measure groups and dimension, and then performing an in-memory run-time join between these retrieved records by using the granularity attributes of each dimension that the measure groups have in common. As the data size of the records that are being joined increases, the performance of this run-time join in SSAS suffers. Query performance with M2M queries in SSAS is intrinsically and linearly tied to the size of the data being joined between the data and intermediate measure groups.

This best practices white paper explores the use of the following three optimization techniques for improving the query performance of M2M dimension queries as data sizes increase:

* Defining aggregations in the data measure group and in the intermediate measure group
* Partitioning the intermediate measure group (partitioning the data measure group is a well-understood optimization technique and knowledge of these benefits on the part of the reader is assumed)
* Reducing the size of the intermediate measure group by defining common dimension surrogate keys to represent repeating M2M combinations and re‑using these keys (this technique is referred to as the matrix relationship optimization technique)

Note   While these optimization techniques can be used to improve query performance with dimensional designs that use M2M relationships in Analysis Services, alternative solutions are also available in certain scenarios to solve many analytical problems without the use of M2M relationships.

#### Overview of Many-to-Many Relationships in SSAS

Every M2M relationship in SSAS has the following four components:

* The **data measure group**, which is built upon the main fact table in the relational data warehouse. This measure group contains the transaction data to analyze; these transactions are called *measures* in SSAS.
* A **M2M dimension**, which is built upon a M2M dimension table in the relational data warehouse. Each record in this M2M dimension relates to many records in the data measure group and each record in the data measure group relates to many records in the M2M dimension.
* An **intermediate measure group**, which is built upon an intermediate fact table in the relational data warehouse. This fact table contains the details of the M2M relationship between the M2M dimension and the dimension table that contains the join keys. This fact table contains no measures and frequently contains more rows of data than the fact table itself. Ralph Kimball calls this type of intermediate fact table a [*factless fact table*](http://www.ralphkimball.com/html/designtipsPDF/DesignTips2003/KimballDT50FactlessFact.pdf).
* An **intermediate dimension**, which is often built upon the main fact table in the relational data warehouse. This intermediate dimension contains one or more join keys that relate the records in the data measure group to the records in the intermediate measure group. When this dimension is built upon the main fact table, it is called a *fact dimension* in SSAS. Ralph Kimball calls this type of dimension a [*degenerate dimension*](http://www.kimballgroup.com/html/designtipsPDF/DesignTips2003/KimballDT46AnotherLook.pdf).

Note   The terms *data measure group*, *M2M dimension*, *intermediate measure group*, and *intermediate dimension* are descriptive terms that are used to describe the function of each object in the context of the M2M dimension relationship. For example, an intermediate dimension in a M2M dimension relationship may also be a standard dimension in relation to some other measure group.

Figure 1 shows the relational model for the M2M relationship in the Adventure Works DW sample database between the DimSalesReason dimension table and the FactInternetSales fact table via the FactInternetSalesReason intermediate fact table. The relationship between the DimTime dimension table and the FactInternetSales table in Figure 1 illustrates a many-to-one dimension relationship.

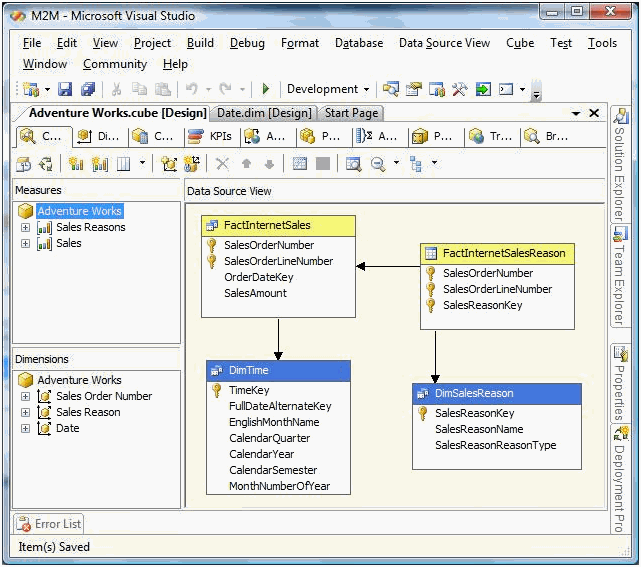


Figure 1: Relationship model of Adventure Works with M2M relationship

In Figure 1, notice the following as they will be used in some of the examples in this document:

* The Sales data measure group, which is based on the FactInternetSales main fact table
* The Sales Reasons intermediate measure group, which is based on the FactInternetSalesReasons intermediate fact table
* The Sales Reason M2M dimension, which is based on the DimSalesReason dimension table
* The Sales Order Number intermediate dimension, which is based on the FactInternetSales main fact table

Figure 2 displays the rows of data in the FactInternetSalesReasons intermediate fact table.

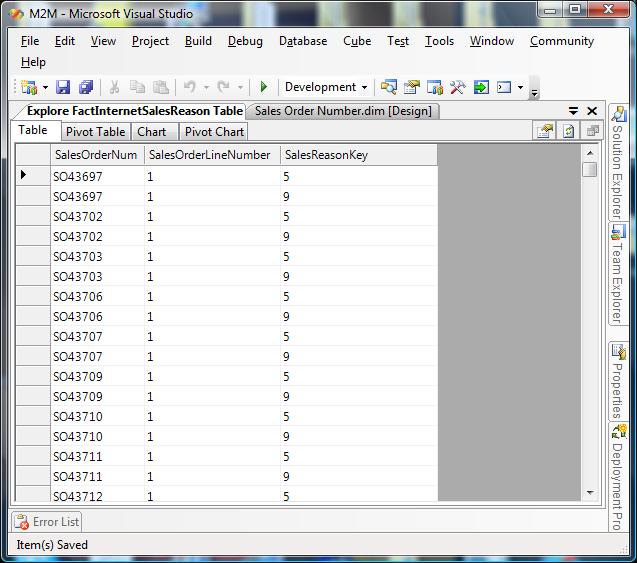


Figure 2: Contents of the FactInternetSalesReasons intermediate fact table

Notice that this table contains no transactional facts. Rather, it records the reason given by customers for each line item in each sales order. For example, for Line Number 1 for Sales Order Number SO43697, two different sales reasons appear. Because multiple rows are used for a single sales order line number, the number of rows in this intermediate fact table is larger than the number of rows in the main fact table.

To calculate the total sales for a given time period for a particular sales reason, data from the DimSalesReason dimension table is joined with data from the FactInternetSales table using the join key data in the FactInternetSalesReason table. In SSAS, the relationships between these objects are defined on the **Dimension Usage** tab in cube designer to enable SSAS to execute this join and resolve M2M queries. Figure 3 displays these relationship definitions.

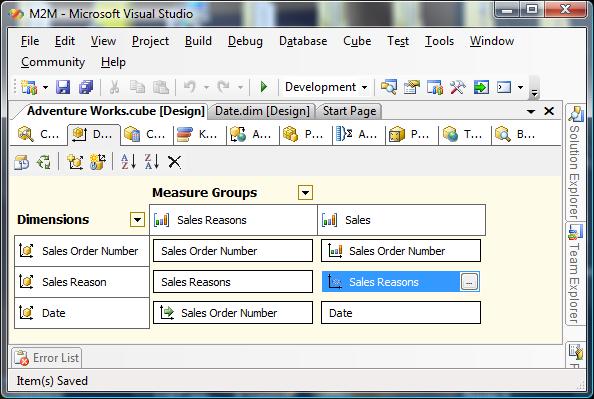


Figure 3: Dimension Usage tab for M2M relationship in SSAS

In Figure 3, the Sales Reason M2M dimension is related to the Sales data measure group via the Sales Reasons intermediate measure group. The join between these two measure groups is via the Sales Order Number intermediate dimension. Defining dimension and measure group relationships in this manner enables SSAS to resolve an MDX query that requests information about sales by time period based on sales reason type. The join required to resolve a M2M query is performed in memory at run time, joining sales reasons to sales fact data via the sales reasons intermediate fact table (using the information displayed in Figure 2 as an example).

**Note**   The underlying dimension and fact tables in the relational data warehouse are not involved in the M2M query to SSAS when MOLAP storage is used. When ROLAP storage is used, the data from each object that is using ROLAP storage is retrieved from the underlying relational data warehouse, but the in-memory run time join is still performed in SSAS.

For more information about defining many-to-many relationships, see [Many-to-Many Dimensions in Analysis Services 2005](http://technet.microsoft.com/en-us/library/ms345139.aspx) by Richard Tkachuk. For information about advanced dimensional modeling techniques using M2M relationships, see [Many-to-many dimensional modeling](http://www.sqlbi.eu/Default.aspx?tabid=80) by Marco Russo.

#### Aggregation Optimization Technique

One way to improve M2M query performance is to define aggregations on the data measure group and the intermediate measure group to support M2M queries. An aggregation is a pre-calculated summary of data that Analysis Services uses to enhance query performance (by moving the calculation to the processing phase). Specifically, an aggregation summarizes measures by a combination of dimension attributes. Another way to explain this is to say that an aggregation is actually another fact table that contains fact data at a higher grain than the fact table on which it is based, and generally containing fewer dimension keys—thus reducing the number of rows and the number of columns as compared to the non-aggregated fact table. Aggregations work by reducing the number of records that the storage engine needs to scan from disk in order to satisfy a query. However, an aggregation provides a significant benefit only if the size of the aggregation is significantly smaller than the size of the original table.

While pre-aggregating data to improve query performance is simple in typical design scenarios, it is more complex in M2M scenarios. How do you design aggregations to support M2M queries? You design aggregations to support M2M queries on both the data measure group and the intermediate measure group. SSAS uses an aggregation in the data measure group and/or the intermediate measure group to resolve a M2M query whenever possible, thereby, reducing the size of the in-memory run-time join. (The aggregation reduces the size of the data that is being join when the M2M query is covered by one or more aggregations.)

* **Designing data measure group aggregations for M2M queries**

To design an aggregation in the data measure group to support a M2M query, you must include in the aggregation the granularity attribute of all dimensions that join with the intermediate measure group. However, do not include the attribute from the M2M dimension; the join on this attribute in the data measure group and in the intermediate measure group occurs at query time as part of the run-time join. When a fact dimension is used as the intermediate dimension to join a M2M dimension to the data measure group (which is a very common scenario), you include the granularity attribute from the fact dimension in the aggregation—which generally causes the aggregation to be a significant fraction of the size of the facts themselves. The benefit of an aggregation in the data measure group for M2M queries is directly related to the size of this aggregation compared to the size of the data measure group. For an example of such an aggregation and information about tools to use to define the aggregation, see [Designing Multiple Cubes](#_Designing_Multiple_Cubes) later in this white paper.

Note: In general, an aggregation whose size is more than approximately 1/3 the size of the facts themselves is not considered to be useful for improving query performance. For more information, see the [SQL Server 2005 Analysis Services Performance Guide](http://www.microsoft.com/technet/prodtechnol/sql/2005/ssas2005perfguide.mspx).

* **Designing intermediate measure group aggregations for M2M queries**

To design an aggregation in the intermediate measure group, you must include in the aggregation the granularity attribute of the dimensions in the intermediate measure group that relates the measure group to the data measure group along with the attribute in the dimension that you wish to aggregate. The benefit of this aggregation is directly related to the resulting size of the aggregation compared to the size of intermediate measure group. In other words, if a significant percentage of the detail rows in the intermediate measure group can be aggregated, an aggregation in the intermediate measure group is useful.

Note   [Appendix A](#_Appendix_A:_Aggregation) includes an example of a Transact-SQL script that we used to calculate the percentage of detail rows that can be aggregated based on the AdventureWorksDW sample database and an aggregation of Sales Reasons to Sales Reasons Type.

#### Intermediate Measure Group Partitioning Optimization Technique

Another way to improve M2M query performance is to partition the intermediate measure group and the data measure group. Partitioning enables SSAS to retrieve data from only a portion of data in a measure group when the partitioning scheme matches the query pattern and to parallelize data retrieval when data must be retrieved from multiple partitions.

Note   Partitioning the data measure group is already a well‑understood technique whose benefits are assumed for the purpose of this article. The data measure group should generally be partitioned by the same dimension attribute members as the intermediate measure group.

While best practices around the partitioning of a data measure group are well understood, the best practices around the partitioning of an intermediate measure group may not be immediately obvious. How do you partition an intermediate measure group to support M2M queries? For example, if your M2M queries are filtered or sliced by time and region, you could partition the intermediate measure group by time and region. When partitioning an intermediate measure group, you must relate the partitioning dimensions to the intermediate measure group in the Dimension Usage tab in Business Intelligence Development Studio.

The benefit of this optimization technique is that it reduces the size of the data from the intermediate measure group that is used by the run-time join with the data measure group when the M2M query can be resolved from only a few (or one) partitions. This approach can leverage the fact that M2M queries are resolved through all shared dimensions between the data measure group and the intermediate measure group, not just a single common dimension. A query that filters the intermediate measure group partitions will reduce the records in the run-time join, but this is even more advantageous if the filter is applied to both measure groups and partitions on both sides are limited.

It is important to note that if the M2M query must be resolved by retrieving data from many or all partitions, the technique provides little or no value and may even increase response times. If, however, your queries do target specific partitions of the intermediate measure group, you can achieve measurable gains in M2M query response times.

To summarize, the following design patterns are required:

* The intermediate measure group must be partitioned by one or more common dimensions used in queries. The Date dimension is often the best choice for this as it provides balanced partitioning options and is also suitable for range or set queries.
* The dimension usage grid must have an association defined from the intermediate measure group to each dimension that is used for partitioning. This is important because in order for the partitions to be selected in the query, the dimension must have a relationship to the intermediate measure group. Otherwise SSAS will not know that a query can be optimized by filtering out partitions defined in the intermediate measure group.
* The previous two implementation requisites assume that the intermediate fact table contains the dimension surrogate key that is used for the measure group partitioning so that the SSAS partitions can be filtered by a defined query (using a WHERE clause) and so that the relationship of the intermediate measure group to the dimension can be defined through the dimension surrogate key that is in the fact table. Alternatively, you can use a named query or Transact‑SQL view to relate the surrogate key to the intermediate fact table, but this may have performance implications during the partition processing because of the joins required.

For more implementation details, see Erik Veerman's blog on the Solid Quality Mentors site at <http://blogs.solidq.com/EN/Erik/Lists/Posts/Post.aspx?ID=3>.

#### Matrix Relationship Optimization Technique

Another way to improve M2M query performance is to use a technique called the *matrix relationship optimization technique*. This optimization technique increases M2M query performance by reducing the size of the run-time join. The run-time join is reduced by using a process of compression to eliminate unnecessary repetitiveness in the intermediate fact table. With this technique, common M2M dimension member combinations in the intermediate fact table are identified and replaced with a surrogate key (called a *matrix key*). This technique collapses the size of the intermediate fact table, which results in a linear increase in M2M query performance. The degree to which the size of the intermediate fact is reduced is known as the reduction percentage—the higher the percentage the better.

Note   [Appendix B](#_Appendix_B:_Calculating) includes a Transact‑SQL script that enables you to calculate the intermediate fact table reduction percentage before you implement the matrix relationship optimization technique. If the percentage is not high, this technique will provide little performance benefit.

##### Compression and the Matrix Key

The matrix relationship optimization technique creates a compressed intermediate fact table by taking the following steps:

* The common dimension member combinations in the intermediate fact table are identified.
* Each set of common dimension member combinations is assigned a surrogate key called the matrix key.
* Repeated combinations are eliminated.

Each row in the main fact table is then associated with this matrix key. By identifying this combination with a matrix key, we can use the matrix key as the join key linking the main fact to the M2M dimension table via the intermediate fact table.

Figure 4 illustrates this concept by applying this technique to the FactInternetSalesReason intermediate fact table and the FactInternetSales main fact table in the AdventureWorksDW relational data warehouse sample.

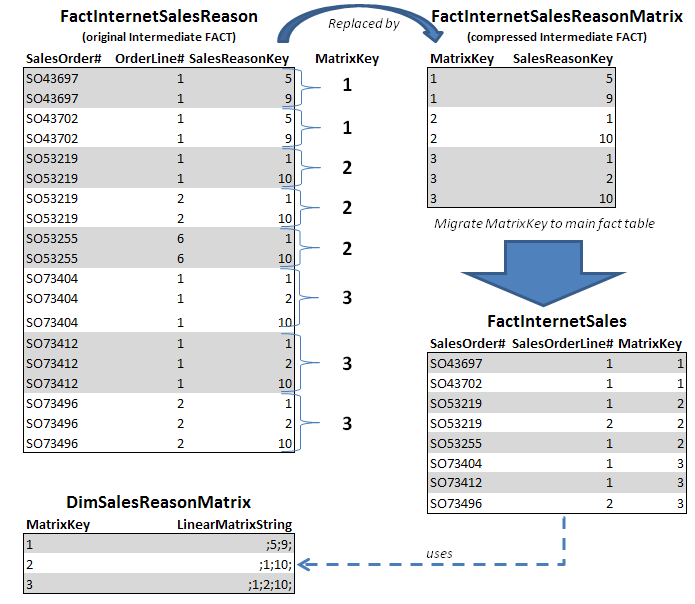


Figure 4: Compressing the FactInternetSalesReason intermediate fact table

In Figure 4, three common sales reason combinations are identified in this subset of the original FactInternetSalesReason table. These unique sets were assigned a surrogate key value and each set was added to the DimSalesReasonMatrix table as a concatenated string. By using the matrix key assigned to these unique sets in a new intermediate fact table (the FactInternetSalesReasonMatrix table), we are able to replace the 19 rows in the original intermediate fact table with 7 rows in the new intermediate fact table. Indeed, there are only 12 unique sets in the entire FactInternetSalesReason table in the sample database, which enables the compressed FactInternetSalesReasonMatrix table to be reduced to 20 rows from 64,515 rows in the original intermediate fact table (the FactInternetSalesReason table). Finally, notice that this matrix key is added to the main fact table (the FactInternetSales table), which enables the M2M join to be accomplished by using the matrix key as the join key rather than the sales order number and sales order line number.

Not every scenario will produce a high degree of common M2M combinations. The best candidates are generic dimensions that commonly apply to multiple fact rows, such as [mini dimensions](http://www.kimballgroup.com/html/designtipsPDF/KimballDT53Dimension.pdf). Situations where a dimension member rarely, or minimally, applies to multiple fact rows are not good candidates for a high degree of compression.

Since the performance challenges of M2M relationships are due to the run‑time join performed between the data measure group and the M2M dimensions through the intermediate measure group, a substantial reduction in the size of the intermediate fact table on which the intermediate measure group is based yields significant performance improvements for MDX queries that use the M2M relationship. The long-term performance behavior of matrix-based M2M relationships is also interesting in scenarios where the growth in the size of the intermediate fact table is directly related to the growth in the size of the main fact table. Since the matrix optimization is based on re‑using common M2M combinations, over time fewer and fewer new M2M combinations are found as the intermediate fact table grows, and the percentage of the matrix keys that are re‑used increases rather than resulting in an increase in the size of the intermediate fact table. As more and more data is sampled, the growth rate of the intermediate fact approaches zero in this scenario and the net result is better compression and better performance.

##### Implementing the Matrix Optimization

Implementing the matrix relationship optimization technique requires changes to aspects of the M2M relationship at the following three levels:

* The relational data warehouse implementation level
* The cube design level
* The ETL implementation level

###### Relational Data Warehouse Implementation

The abstract model shown in Figure 5 depicts the key relational data warehouse elements of the matrix relationship optimization technique:



Figure 5: Diagram of relational data warehouse elements

The following table describes each of these elements.

Table 1: Description of relational data warehouse elements

|  |  |
| --- | --- |
| Fact Table | |
| Description | The only difference between the main fact table in the matrix model and a standard M2M model is the addition of the matrix key in the fact table (either directly or through a view). |
| Columns | The MatrixKey column is a foreign key from the MatrixDimension table of a particular combination of dimension members participating in the M2M relationship. |
| Intermediate Fact Table | |
| Description | The join table between the main fact table and one or more dimensions in a M2M relationship with the main fact table. In this model, this table represents a normalized view of the common combinations of the M2M dimension members.  In certain cases, M2M relationships are defined between multiple dimension members. As with a standard M2M relationship, the key of each dimension participating in the relationship must exist in the intermediate fact table on the same row. |
| Columns | The MatrixKey column is a foreign key from the MatrixDimension table of a particular combination of dimension members participating in the M2M relationship.  The DimensionKey columns(s) are the dimension keys for one or more dimensions participating in the M2M relationship. |
| Grain | One row per matrix key per relationship (where each relationship could involve multiple dimensions) |
| Dimension | |
| Description | One or more dimension tables for the M2M dimensions participating in the M2M relationship |
| Columns | The DimensionKey column contains the surrogate key of each dimension member in the M2M dimension |
| Grain | One row per dimension member |
| Matrix Dimension | |
| Description | The table containing the matrix primary key for each unique combination of M2M dimension members. Used strictly for defining, maintaining, and looking up matrix definitions. |
| Columns | The MatrixKey column contains the primary key identifying a common combination of dimension members  The LinearMatrixString column contains a concatenated list of all the dimension key values of the dimension member combination |
| Grain | One row per dimension member combination, regardless of the number of dimension members involved in the relationship |

Implementing the matrix relationship optimization technique in the relational data warehouse requires the addition of the tables used for the matrix dimension and intermediate fact table and the addition of a matrix column in the base fact table for the matrix relationships. The details include:

1. The first table needed is a Matrix Dimension table. This table has two columns—a Matrix Key column and a Linear Matrix String column.

The Linear Matrix String column will contain a sorted list of the M2M keys that are related, separated by a delimiter, such as a semi-colon. If there is more than one M2M dimension related to the intermediate fact table, you can include all the dimensions in the string, ordered and separated by a grouping delimiter, such as a vertical bar. A string entry with multiple dimensions might look like this:

;8;11;250|20;411;5612;

The following table illustrates the content of a Matrix Dimension table.

Table 2: Illustration of the content of a Matrix Dimension table

| MatrixKey | LinearMatrixString |
| --- | --- |
| 1 | ;8;11; 250; |
| 2 | ;5;8; |
| 3 | ;11; |
| 4 | ;15;60;899; |
| 5 | ;561; |

Note: This table does not provide any analytical value, but exists solely to support the string concatenation of surrogate keys from the M2M dimension.

For our tests, we used the following CREATE TABLE script to create this table in the AdventureWorksDW database:

CREATE TABLE [dbo].[DimSalesReasonMatrix]

([SalesReasonMatrixKey] [int] NOT NULL,

[LinearMatrixString] [varchar](255) NULL,

CONSTRAINT [PK\_DimSalesReasonMatrix] PRIMARY KEY CLUSTERED

([SalesReasonMatrixKey] ASC))

--add IDENTITY if a T-SQL sript is used to add rows rather than an

--SSIS package that adds the unique identity value

1. The matrix relationship optimization technique requires an intermediate fact table that relates the M2M dimensions to the base fact through the Matrix Key. This intermediate fact table will have the Matrix Key column and one or more Dimension Key columns. Using the first two Matrix Key values from the table in step 1 (Table 2), these columns in the intermediate fact table will look like the table in Table 3.

Table 3: Illustration of the Matrix intermediate fact table

| MatrixKey | Dimension Key |
| --- | --- |
| 1 | 8 |
| 1 | 11 |
| 1 | 250 |
| 2 | 5 |
| 2 | 8 |

The Matrix Key is repeated for each related dimension key from the string concatenation in step 1.

For our tests, we used the following CREATE TABLE script to create this table in the AdventureWorksDW database:

CREATE TABLE [dbo].[FactInternetSalesReasonMatrix]

([SalesReasonMatrixKey] [int] NULL,

[SalesReasonKey] [int] NULL)

1. The final relational data warehouse step for the matrix relationship optimization technique is to add the Matrix Key column to the base fact table. This is required to tie the fact table to the intermediate fact table.

For this task, you could use the following ALTER TABLE script to alter the FactInternetSales table in the AdventureWorksDW database:

ALTER TABLE dbo.FactInternetSales

Add [SalesReasonMatrixKey] [int] NULL

However, for our tests, we created a view that related the SalesOrderNumber and SalesOrderLineNumber to the SalesReasonMatrixKey, and then performed the necessary join for the processing query in the processing query for each partition. This technique enabled us to avoid altering the FactInternetSales table in our sample database.

###### Cube Design

The implementation of the matrix relationship optimization technique within SSAS works in the same way that any M2M dimension relationship is created. The difference is that the shared dimension between the base measure group and the intermediate measure group is the matrix dimension, rather than the fact dimension. The details of the cube design implementation are:

1. In Data Source View, add two tables—one for the matrix dimension and a second for the new intermediate fact table. Update the data source view to ensure that the base fact table referenced in the data source view includes the new matrix dimension key column that you added to this table (unless you add the matrix dimension key column via a join in the processing query for each partition). Figure 6 shows these changes in the Adventure Works data source view.

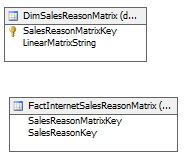
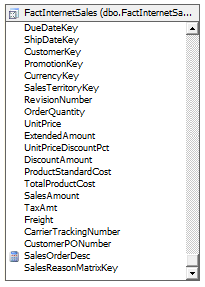


Figure 6: Adventure Works data source view changes

1. Next, add a new SSAS database dimension based on the matrix dimension table. Add this dimension to the cube that contains the base measure group. This SSAS dimension will have only one attribute—the Key attribute, which is the surrogate dimension key. This dimension does not need the key concatenation string included as an attribute because it is merely used to generate the matrix key. In the cube, set the **Visible** property for the matrix dimension to False; this dimension won’t be used for analytics, it is used only to resolve the relationship. Figure 7 shows this new dimension in the Adventure Works cube that we used for our testing.

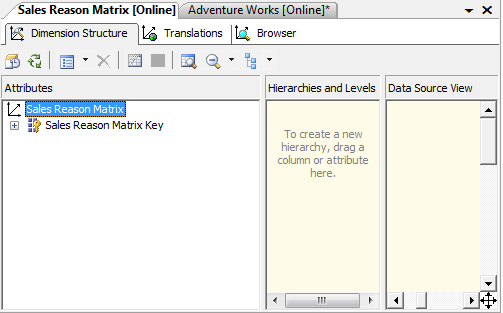


Figure 7: SalesReasonMatrix dimension

1. Add a new SSAS measure group to act as the intermediate measure group for resolving the M2M relationships (and remove the existing Sales Reasons measure group). This measure group is based on the new intermediate fact table (containing the matrix key) from step 2 of the relational data warehouse implementation. Since each measure group needs at least one measure, use the default Count measure that is created and set its **Visible** property to False (unless you wish to use the count measure to display the number of dimension records to which the base measures relate). Figure 8 shows this new measure group in the Adventure Works cube that we used for our testing, along with the new cube dimension from step 2.

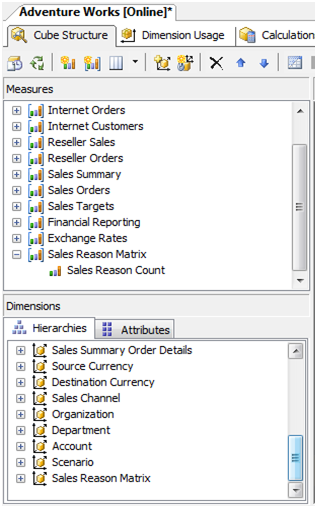
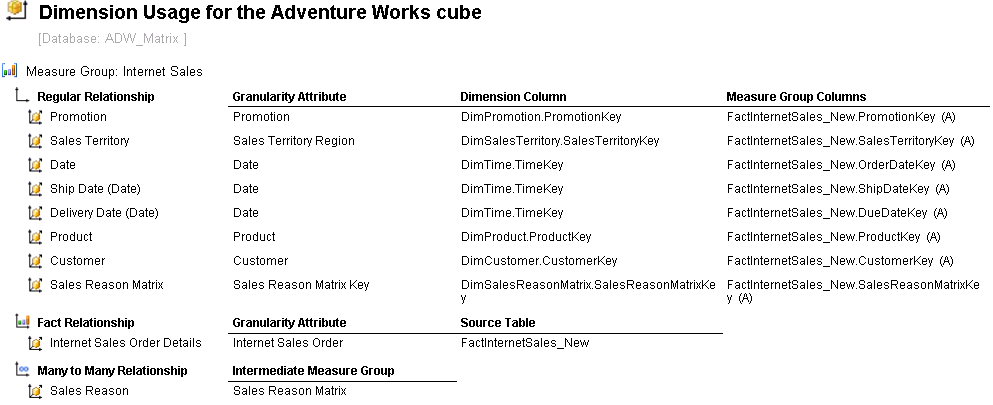


Figure 8: Sales Reason Matrix measure group and Sales Reason Matrix cube dimension

1. Define the Dimension Usage to associate the dimension tables to the measure groups.
2. Both the base measure group and the intermediate measure group must have a direct relationship to the matrix dimension. This relationship should be the only dimension that the two measure groups share. (The old intermediate measure group relationship is removed when the old intermediate measure group is dropped from the cube design.)
3. The intermediate measure group also needs a direct relationship to the dimension or dimensions that participate in the M2M relationship (the Sales Reason dimension in our sample cube). In your intermediate fact table, you have these dimension key(s) included along with the matrix dimension key. Therefore, you can relate the intermediate measure group to the dimensions through the dimension key(s).
4. The dimensions referenced in step b must also be related to the base measure group in step a. To do this, set the relationship between the base measure group and the dimension(s) to the Many-to-Many Relationship type, and set the new intermediate measure group as the resolving measure group (called an intermediate measure group in the Dimension Relationship editor).

Figure 9 shows this dimension usage definition in the Adventure Works cube that we used for our testing.



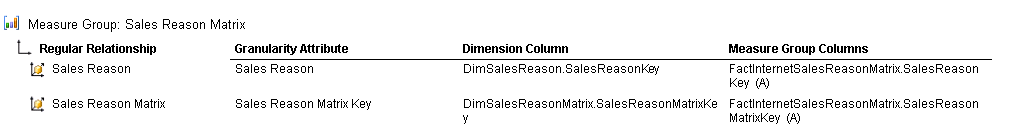


Figure 9: Affected objects on the Dimension Usage tab

Note: Figure 9 was produced by using the BIDS Helper community sample application, available on CodePlex at <http://www.codeplex.com/bidshelper>

###### ETL Implementation

The ETL implementation for the matrix relationship optimization technique involves updating the relational data warehouse tables with new matrix relationships and associating the fact tables to the matrix dimensions. This sounds easier than it actually is. The challenges involve pivoting the dimension keys into a linear string, and then making sure that the base fact table knows which matrix key to associate with each of its records, and building the intermediate fact table with the matrix key and dimension key, row by row. The general steps are:

1. Adding new key concatenation strings to the matrix dimension table for new records being added to the fact table. The keys need to be grouped by the transaction ID of the source and sorted before the concatenation. Sorting ensures that only one matrix dimension record is added for a unique set. For example, ;4;12; is the same as ;12;4; and the sorting would enforce ;4;12; in all instances of the keys. This step also requires that only new matrix records are added to the matrix dimension because it is probable that the same matrix key set will appear over time.
2. Adding the matrix dimension key associated with each transaction ID to the base fact table. This means that in step 1, ideally, the dimension keys are stored with their associated fact table transaction key in a staging table that can be used in this step to update the fact table or to use when adding new fact rows.
3. Adding the key of the matrix dimension with each M2M dimension key record (or dimension keys if more than one dimension participates in the matrix) to the intermediate fact table.

While these tasks can be accomplished by using Transact‑SQL scripts, we prefer to use SQL Server 2005 Integration Services (SSIS) for this task. With the previous steps in mind, the following SSIS approach models how this can be done efficiently. The SSIS approach has the following characteristics:

* The matrix dimension surrogate key is generated in the SSIS package.
* The matrix dimension and intermediate fact tables are loaded simultaneously, since the dimension key is self generated.
* A staging table is included in the outcome; this staging table can be used to update the base table either through a set update or a lookup transformation in a subsequent data flow.
* The package uses Script Components in the data flow, which provides an easy way for concatenation across rows and optimization of the check for existing records.

Figure 10 highlights the control flow of the package that we created for our testing on the AdventureWorksDW sample database.

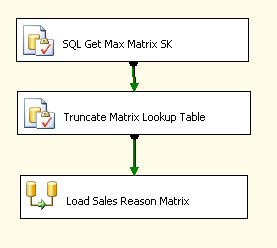


Figure 10: SSIS Control Flow

Three steps are used:

1. The first step, the SQL Get Max Matrix SK task, is an Execute SQL task that executes a MAX operation on the SalesReasonMatrixKey from the DimSalesReasonMatrix table. The resulting value is stored in the package variable named *New\_SalesReasonMatrixKey*.
2. The second step, Truncate Matrix Lookup Table, is an Execute SQL task that truncates a staging table, which we named SalesOrderToMatrix\_Lookup. The staging table has the following DDL for our testing using the AdventureWorksDW relational data warehouse:

CREATE TABLE [dbo].[SalesOrderToMatrix\_Lookup]

([SalesReasonMatrixKey] [int] NULL,

[SalesOrderID] [int] NULL)

The SalesOrderID comes from the AdventureWorksDW source, which is the transaction key of the sale and will be used to associate the base fact table to the matrix dimension.

1. The third step, the Load Sales Reason Matrix task, is a Data Flow task that performs the majority of the processing logic. Figure 11 highlights the Data Flow layout.

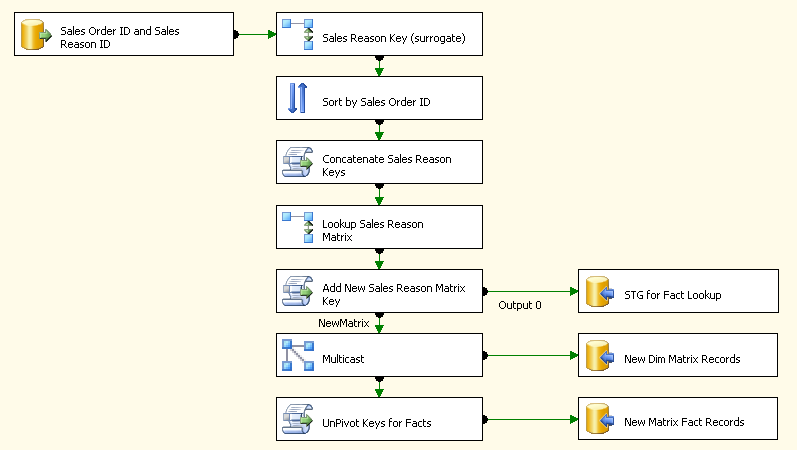


Figure 11: Load Sales Reason Matrix data flow task layout

To decompose this, let’s start with the sources and destinations. The source adapter extracts rows from the AdventureWorks sample database and pulls the Sales Order ID and Sales Reason ID by using the following query:

SELECT SalesOrderID, SalesReasonID

FROM Sales.SalesOrderHeaderSalesReason

The destinations are inserting rows into the staging table (described above), the matrix dimension table, and the matrix intermediate fact table. As you can see, this single data flow is handling steps 1 and 2 of the ETL process described earlier.

The more complex aspect of the Data Flow is in the transformation logic. Table 4 lists the data flow tasks and describes the transforms in more detail.

Table 4: Load Sales Reason Matrix data flow task details

| Transform Name | Description of Transform |
| --- | --- |
| Sales Reason Key (surrogate) | This Lookup transformation caches the Sales Reason dimension table in memory and uses the Sales Reason ID from the source to get the Sales Reason Key, which is the surrogate key of the dimension. It is assumed that all the Sales Reason dimension members exist when this SSIS package is run. |
| Sort by Sales Order ID | Since the primary purpose of the data flow is to concatenate the Sales Reason Keys by each sale, a sort is first needed to put the Sales groupings in order. The sort includes Sales Order ID and Sales Reason Key in that order, so we can also sort the surrogate key in order (and prevent duplicates in the matrix tables). |
| Concatenate Sales Reason Keys | This Script Component evaluates the Sales Order IDs (in order, since they are sent to this component this way) and for all the same grouping, the Sales Reason Keys are concatenated. The output of this script is a single row per Sales Order ID, with the concatenated string. In other words, this transformation performs a pivot with a concatenation. |
| Lookup Sales Reason Matrix | Another Lookup is needed to determine if the concatenated string already exists in the Matrix Dimension table. If a record does exist, the surrogate key is returned. If no match exists, a NULL is added as the matrix dimension surrogate key (the Lookup is configured to ignore errors). |
| Add New Sales Reason Matrix Key | This script task is the most complicated, as it performs the following steps:   1. It adds a new surrogate key for any row with NULLs in the matrix dimension key. An internal hash table is persisted to assure that the same concatenation string is not added more than once, since more than one new Sales Order ID group may have the same Sales Reason keys. 2. All the rows are sent out the “Output 0” and are stored in the SalesOrderToMatrix\_Lookup staging table used to update the base fact table. 3. The NewMatrix output has only records that did not already exist in the dimension table, so that these records can be added to both the matrix dimension table and the intermediate fact table. |
| Multicast | The Multicast transformation simply sends all the NewMatrix rows from the previous script component to the matrix dimension table (since they are all new dimension records) and the next downstream transformation described immediately below. |
| UnPivot Keys for Facts | The final transformation step is to unpivot the keys. Since we have the matrix dimension key and a concatenated string of Sales Reason keys, and these are new records, if we unpivot the keys, they can be added to the intermediate fact table. Therefore, the script goes row by row and outputs multiple rows if the concatenation string has more than one Sales Reason key. These Sales Reason keys are sent to the intermediate fact table along with the matrix dimension key. |

Although not shown, a final step is needed to complete the ETL. As mentioned earlier, the matrix dimension key must be in the main fact table. To accomplish this association, one of the following approaches can be used:

* **Update the fact table with the matrix dimension key.** The staging table SalesOrderToMatrix\_Lookup can used to update the FactInternetSales fact table. The join between the two tables is across the SalesOrderID to SalesOrderNumber (the SalesOrderNumber has an SO appended to the beginning of the SalesOrderID).
* **Lookup the matrix dimension key during the ETL**. If you are loading the FactInternetSales after the above package is run, the staging table SalesOrderToMatrix\_Lookup can be used in a Lookup transformation in SSIS to get the matrix dimension key before loading the new records in the FactInternetSales table.
* **Use a view that associates the matrix dimension key**. If you prefer not to put the matrix dimension key into the fact table, but to keep it in a separate table, the SalesOrderToMatrix\_Lookup table can be used in a view to bring these two tables together. For this approach, do NOT truncate the table for every ETL (if you run the package incrementally) and make sure to index the staging table on the SalesOrderID.

Note   Sample packages for accomplishing these tasks on an existing fact table as well as adding them on an ongoing basis as new records flow into the relational data warehouse from the transactional system are included as a download with this best practices article.

#### Test Objective

Our test objective was to test each of these optimization techniques independently and in conjunction with each other (where appropriate).

#### Test Methodology

To achieve our test objective, we scaled up the relevant tables in the AdventureWorksDW database based on four relational data warehouse scale-up scenarios, modified the AdventureWorksDW database to support the matrix optimization technique, and then tested multiple M2M queries against each of three cubes. The first cube used the default M2M design for the Adventure Works sample cube; the second cube used the intermediate measure group partitioning optimization technique; and the third cube used that matrix relationship optimization technique. We tested each of these cubes with and without aggregations.

##### Scaling Up the AdventureWorksDW Relational Data Warehouse

To test the performance of each M2M optimization technique, we scaled up the data in three tables in the AdventureWorksDW relational data warehouse based on the following four relational data warehouse scenarios:

* Scenario 1 – Medium fact volume, low dimension volume with a low M2M ratio
* Scenario 2 – Same dimension volume and M2M ratio as Scenario 1, but with three times the number of facts
* Scenario 3 – Medium fact volume, medium dimension volume with a high M2M ratio
* Scenario 4 – High fact volume, high dimension volume with a low M2M ratio

The table in Table 5 describes the details of the scale-out sizes of the tables in each of these four scenarios before we implemented the matrix relationship optimization technique.

Table 5: AdventureWorksDW scale-out table sizes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Scenario | DimSalesReason count | FactInternetSales count | FactInternetSalesReason count | Ratio of Sales Reasons to Sales Reason Types |
| 1 | 20,000 | 10,086,466 | 15,285,991 | 1:1.5 |
| 2 | 20,000 | 30,199,000 | 46,150,062 | 1:1.5 |
| 3 | 2,000,000 | 2,053,532 | 41,070,640 | 1:20 |
| 4 | 50,311,534 | 50,000,000 | 75,066,873 | 1:1.5 |

Table 6 describes the details of the scale-out sizes of these three tables used for the matrix relationship optimization technique in each of these four scenarios, as well as the compression percentage.

Table 6: Matrix relationship optimization technique scale-out table sizes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Scenario | DimSalesReason Matrix count | FactInternetSales Reason count | FactInternetSalesReason Matrix count | Compression % |
| 1 | 348,695 | 15,285,991 | 851,084 | 94% |
| 2 | 348,695 | 46,150,062 | 851,084 | 98% |
| 3 | 941,495 | 41,070,640 | 18,829,880 | 54% |
| 4 | 29,919,504 | 75,066,873 | 57,509,738 | 23% |

Notice the significant reduction in size of the intermediate fact table (the FactInternetSalesReasonMatrix table) in scenarios 1 and 2 compared to the size of the intermediate fact table (the FactInternetSalesReason) with the original fact table. The performance improvement associated with matrix relationship optimization technique is directly related to this compression.

Notice also that the number of unique combinations of sales reasons to line items, as shown in the DimSalesReasonMatrix count column, is identical in scenarios 1 and 2 (we added sales to the fact tables, but did not add any new sales reasons. This results in an identical intermediate fact table for the matrix relationship optimization technique in scenarios 1 and 2.

##### Designing Multiple Cubes

Next, we used a modified version of the Adventure Works DW-Simple cube to test the performance of six M2M queries against cubes based on these four relational data warehouse scenarios. This simplified version of the sample Adventure Works cube is from the [Identifying and Resolving MDX Query Performance Bottlenecks in SQL Server 2005 Analysis Services](http://technet.microsoft.com/en-us/sqlserver/bb331794.aspx) best practices article with the following modifications to improve processing performance with these large table sizes. We removed the user hierarchies from the fact dimensions and changed the value of the **AttributeHierarchyOrdered** property for each of the attributes in these fact dimensions to False. It is important to point out that the storage type of each of these fact dimensions is MOLAP (in the original sample Adventure Works cube, these fact dimensions use ROLAP storage). MOLAP storage for fact dimensions, as well as all for other Analysis Services objects, is dramatically faster than ROLAP storage. This performance difference was particularly noticeable with our M2M queries since the keys in the fact dimension are the join keys for our default cube.

###### Cube Designs

For each relational data warehouse scenario, we designed three different SSAS cubes:

* Cube design 1 – This design is based on the modified Adventure Works DW-Simple cube.
* Cube design 2 – This design is based on the modified Adventure Works DW-Simple cube, and then modified to support partitioning of the intermediate measure group by time.
* Cube Design 3 – This design is based on the modified Adventure Works DW-Simple cube, and then modified to support the matrix relationship optimization technique discussed previously.

For each of these cubes, we designed a single aggregation in the data measure group and in the intermediate measure group to support the M2M queries that we were testing, and tested our MDX queries both with and without these aggregations.

* **Cube design 1** – For this design, we defined an aggregation in each of the partitions in the Internet Sales data measure group and defined an aggregation in the partition in the Sales Reasons intermediate measure group. We defined the aggregation in the partitions in the data measure group on the Month Name attribute in the Date dimension and on the Internet Sales Order (granularity) attribute in the Internet Sales Order Details fact dimension.

Important   The granularity (key) attribute in the Internet Sales Order fact dimension is required for SSAS to use the aggregation to resolve the M2M queries, which results in a large aggregation (see Tables 8-11 in [Appendix C](#_Appendix_C:_Cube)). When you design an aggregation in the data measure group to support a M2M query, each dimension in the M2M query must be included in the aggregation. In the aggregation, you must include the granularity attribute (generally the key attribute) from each dimension (in the query) joining the data measure group with the intermediate measure group and you must not include the M2M attribute itself. However, the subcube vector reported by the Query Subcube event in SQL Server Profiler for M2M queries includes the M2M attribute and does not include the granularity attribute (this is due to a product oversight). As a result, the only way to create this aggregation definition is to define a custom aggregation by using the Aggregation Manager sample utility. This program is available on CodePlex at <http://www.codeplex.com/MSFTASProdSamples>; a community-enhanced version of this utility is available on CodePlex at <http://www.codeplex.com/bidshelper>.

We also defined the aggregation in the partition in the intermediate measure group on the Sales Reason Type attribute as well as the Internet Sales Order (granularity) attribute in the Internet Sales Order Details fact dimension.

* **Cube design 2** – For this design, we defined an aggregation in each of the partitions in the Internet Sales data measure group and also defined an aggregation in each of the partitions in the Sales Reasons intermediate measure group. We defined the aggregation in the partitions in the data measure group on the Date (granularity) attribute in the Date dimension and on the Internet Sales Order (granularity) attribute in the Internet Sales Order Details fact dimension.

Note: The granularity attribute of the Date dimension as well as the granularity attribute of the Internet Sales Order Details fact dimension is required for SSAS to use the aggregation to resolve the M2M queries that we used for our tests. When you design an aggregation in the data measure group to support a M2M query, each dimension in the M2M query must be included in the aggregation. In the aggregation, you must include the granularity attribute (generally the key attribute) from each dimension (in the query) joining the data measure group with the intermediate measure group and you must not include the M2M attribute itself. Because the granularity attribute of the Date dimensions is required rather than the Month Name attribute used in the aggregation design for Cube design 1, the size of this aggregation is larger than the aggregation defined for the first cube design (see Tables 8-11 in [Appendix C](#_Appendix_C:_Cube)).

We also defined the aggregation in each of the partitions in the intermediate measure group on the Sales Reason Type attribute, the Date (granularity) attribute in the Date dimension as well as the Internet Sales Order (granularity) attribute in the Internet Sales Order Details fact dimension.

* **Cube design 3** – For this design, we defined an aggregation in each of the partitions in the Internet Sales data measure group and on the partition in the Sales Reason Matrix intermediate measure group. We defined the aggregation in the partitions in the data measure group on the Month Name attribute in the Date dimension and on the Sales Reason Matrix Key (granularity) attribute in the Sales Reason Matrix dimension. We also defined the aggregation in the partition in the Sales Reason Matrix intermediate measure group on the Sales Reason Type attribute as well as on the Sales Reason Matrix Key (granularity) attribute in the Sales Reason Matrix dimension. see Tables 8-11 in [Appendix C](#_Appendix_C:_Cube).

##### M2M Test Queries

Table 7 contains the six MDX queries we used for our tests. The first two queries request data from all years in the cube. The third query requests data from a specific calendar year. The fourth query requests data from a specific calendar quarter within a calendar year. The fifth query requests data from two calendar years, and the Internet Sales measure group is partitioned by year. The sixth query requests data from four calendar quarters within two different calendar years.

Table 7: MDX queries used in our tests

|  |  |
| --- | --- |
| **Query #** | **Query text** |
| 1 | SELECT CROSSJOIN (  { [Sales Reason].[Sales Reason Type].Members },  {[Measures].[Internet Sales Amount],  [Measures].[Internet Total Product Cost],  [Measures].[Internet Gross Profit],  [Measures].[Internet Gross Profit Margin] } ) ON 0,  [Date].[Fiscal].[Month] ON 1  FROM [Adventure Works] |
| 2 | SELECT [Measures].[Internet Sales Amount] ON 0,  [Date].[Calendar Year].Members  \* [Sales Reason].[Sales Reason Type].[Sales Reason Type].Members ON 1  FROM [Adventure Works] |
| 3 | SELECT [Measures].[Internet Order Quantity] ON 0  , [Sales Reason].[Sales Reason Type].[Sales Reason Type] ON 1  FROM [Adventure Works]  WHERE [Date].[Calendar].[Calendar Year].&[2003] |
| 4 | SELECT [Measures].[Internet Order Quantity] ON 0  , [Sales Reason].[Sales Reason Type].[Sales Reason Type] ON 1  FROM [Adventure Works]  WHERE [Date].[Calendar].[Calendar Quarter].&[2003]&[4] |
| 5 | SELECT {[Date].[Calendar].[Calendar Year].&[2003]:  [Date].[Calendar].[Calendar Year].&[2004]} ON 0  , [Sales Reason].[Sales Reason Type].[Sales Reason Type] ON 1  FROM [Adventure Works]  WHERE [Measures].[Internet Order Quantity] |
| 6 | SELECT {[Date].[Calendar].[Calendar Quarter].&[2003]&[3]:  [Date].[Calendar].[Calendar Quarter].&[2004]&[2]} ON 0  , [Sales Reason].[Sales Reason Type].[Sales Reason Type] ON 1  FROM [Adventure Works]  WHERE [Measures].[Internet Order Quantity] |

These queries of data within one, two, or all partitions will enable us to see the effect of the partitioning optimization technique as well as the matrix relationship optimization technique, both with and without aggregations.

Important   To insure reproducibility with our test queries when running against the each of the 12 Analysis Services databases (four relational data warehouse scenarios times three cube designs for each scenario), we set the value of the **PreAllocate** property in the msmdsrv.ini file to 33. This property causes SSAS to preallocate 33% of the available memory on the SSAS query server to Analysis Services at startup. We discovered that the memory manager in the Windows® operating system does not scale well with many small virtual memory allocations. Worker threads related to user queries can be blocked while waiting for the memory manager to allocate virtual memory. We noticed that this behavior resulted in slower performance for queries that were executing at a time when additional virtual memory was being allocated. Using the **PreAllocate** property resulted in consistency for our query tests.

#### Test Results and Observations

Our tests of the six MDX queries against each of the 12 Analysis Services databases (four relational data warehouse scenarios times three cube designs for each scenario) yielded the following results:

* Cube design 3, the matrix relationship optimization technique, yielded substantial performance benefits for all four scenarios and for all queries.
* Cube design 3, in conjunction with aggregations on both the main and intermediate measure groups yielded the best results for all four scenarios and for all queries.
* Cube design 2 yielded better query performance results than either cube designs 1 or 3 in only limited situations, and sometimes yielded worse performance numbers than either cube designs 1 and 3.
* Aggregations on the data measure group yielded performance benefits only in conjunction with cube design 3.
* Aggregations on the intermediate measure groups yielded performance benefits only when the aggregation is significantly smaller than the intermediate measure group itself.

Our results and observations from each of the six MDX queries for each of the four scenarios are described in the following sections.

##### Scenario 1

This section covers the results for each of these six queries for relational data warehouse scenario 1. This scenario has 20,000 members in the dimension table, 10 million rows in the main fact table, with a ratio of sales reasons to sales reasons types of 1 to 1.5.

Queries 1 and 2

Figure 12 displays the performance numbers for queries 1 and 2 for relational data warehouse scenario 1 for all three cube designs.

Figure 12: Queries 1 and 2 performance in Scenario 1

Notice that each of these queries is dramatically faster with cube design 3, which uses the matrix relationship optimization technique, than with either cube designs 1 or 2. Without the use of aggregations, these queries are approximately 70% faster with cube design 3 than with cube design 1. Notice also that these queries are actually slower with cube design 2 than with cube design 1; partitioning of the intermediate measure group actually hurts the performance of these two queries.

With an aggregation in the data measure group with cube design 3, these queries are approximately 80% faster than without this aggregation (the aggregation in the intermediate measure group has little effect on these queries in this scenario). The aggregation in the data measure group is faster with cube design 3 only because, with the matrix relationship optimization technique, the aggregation in each partition is approximately 90% smaller than facts themselves. Contrast this to cube design 1 where the aggregations are only about 20% smaller (and only 10% smaller with cube design 2) than the facts themselves (see Table 8 in [Appendix C](#_Appendix_C:_Cube)). The aggregation in the intermediate measure group for cube designs 1 and 2 provide a small amount of performance improvement because the effect of the aggregation in the intermediate measure group is to reduce the size of the real-time join between the data measure group and the intermediate measure group.

In summary, cube design 3 with an aggregation in both measure groups yields the best performance with queries 1 and 2—approximately 95% total performance improvement, from slightly more than 20 seconds to approximately 1 second.

Queries 3 and 4

Figure 13 shows the performance numbers for queries 3 and 4 for relational data warehouse scenario 1 for all three cube designs.

Figure 13: Queries 3 and 4 performance in Scenario 1

Notice that again each query is dramatically faster with cube design 3 than with either cube designs 1 or 2. Without the use of aggregations, these queries are approximately 95% faster with cube design 3 than with cube design 1. Also, these queries are faster with cube design 2 than with cube design 1. The query performance of queries 3 and 4 improve with cube design 2, the intermediate measure group partitioning optimization technique, compared to queries 1 and 2. This is because queries 3 and 4 only touch a single partition. By comparing queries 3 and 4 with queries 1 and 2, we see that cube design 2 provides a benefit when a single partition is involved—but cube design 3 is still the performance winner.

Similar to queries 1 and 2, an aggregation in the data measure group for cube design 3 results in a significant performance improvement (although not as significant as with queries 1 and 2 because queries 3 and 4 are retrieving data from a single partition, which itself yields a significant performance improvement).

In summary, cube design 3 with an aggregation in both measure groups yields the best performance with queries 3 and 4—approximately 97% total performance improvement, from approximately 15 seconds to approximately 1/2 second.

Queries 5 and 6

Figure 14 shows the performance of queries 5 and 6 for relational data warehouse scenario 1 for all three cube designs.

Figure 14: Queries 5 and 6 performance in Scenario 1

Again, each of the queries is dramatically faster with cube design 3, which uses the matrix relationship optimization technique, than with cube designs 1 or 2. Without the use of aggregations, these queries are approximately 80% faster with cube design 3 than with cube design 1. An aggregation in the data measure group for cube design 3 results in a significant performance improvement, similar to queries 1 and 2.

Queries 5 and 6 are slower with cube design 2 than with cube design 1. The query performance of these queries does not improve with cube design 2 because these queries touch two partitions rather than a single partition (unlike queries 3 and 4). By comparing queries 5 and 6 with queries 3 and 4, we see that cube design 2 provides a benefit when only a single partition is involved—but cube design 3 remains the performance winner.

In summary, cube design 3 with an aggregation in both measure groups yields the best performance with queries 5 and 6—approximately 96% total performance improvement, from approximately 20 seconds to approximately 1 second.

##### Scenario 2

The results for each of these six queries for relational data warehouse scenario 2 are discussed in this section. This scenario has 20,000 members in the dimension table, 30 million rows in the main fact table, with a ratio of sales reasons to sales reasons types of 1 to 1.5.

Queries 1 and 2

Figure 15 shows the performance numbers for queries 1 and 2 for relational data warehouse scenario 2 for all three cube designs.

Figure 15: Queries 1 and 2 performance in Scenario 2

Similar to scenario 1, each query is dramatically faster with cube design 3, which uses the matrix relationship optimization technique, than with cube designs 1 or 2. With cube design 3 with an aggregation in the data measure group, query performance improves from approximately 75 seconds to approximately 1 second. In particular, notice that when we increased the number of rows in the fact table by a factor or 3 with scenario 2, query performance with cube designs 1 and 2 also increased by a factor of approximately 3. However, query performance with cube design 3 remained constant at approximately 1 second.

As the number of rows in the intermediate measure group increases, the benefit of an aggregation in the intermediate measure increases. The performance of cube design 2 is worse than cube design 1—cube design 3 remains the performance winner.

In summary, cube design 3 with an aggregation in both measure groups yields the best performance with queries 1 and 2—a total performance improvement of approximately 99%, from approximately 75 seconds to approximately 1 second.

Queries 3 and 4

Figure 16 shows the performance numbers for queries 3 and 4 for relational data warehouse scenario 2 for all three cube designs.

Figure 16: Queries 3 and 4 performance in Scenario 2

Once again queries 3 and 4 are dramatically faster with cube design 3 than with either cube designs 1 or 2. Similar to scenario 1, cube design 2 yields a performance increase because only a single partition is being touched. But, cube design 3 remains the performance winner.

In summary, cube design 3 with an aggregation in both measure groups yields the best performance with queries 3 and 4—a total performance improvement of approximately 99%, from approximately 60 seconds to approximately 1/2 second.

Queries 5 and 6

Figure 17 shows the performance numbers for queries 5 and 6 for relational data warehouse scenario 1 for all three cube designs.

Figure 17: Queries 5 and 6 performance in Scenario 2

Queries 5 and 6 are once again dramatically faster with cube design 3 than with either cube designs 1 or 2. Similar to scenario 1, cube design 2 does not yield a performance increase when two partitions are being touched rather than a single partition; in fact, performance is slightly degraded. Cube design 3 remains the performance winner.

In summary, cube design 3 with an aggregation in both measure groups yields the best performance with queries 5 and 6—a total performance improvement of approximately 98%, from approximately 70 seconds to approximately 1 second.

##### Scenario 3

The results for each of these six queries for relational data warehouse scenario 3 are discussed in this section. This scenario has 2 million members in the dimension table, 2 million rows in the main fact table, with a ratio of sales reasons to sales reasons types of 1 to 20.

Queries 1 and 2

Figure 18 shows the performance numbers for queries 1 and 2 for relational data warehouse scenario 2 for all three cube designs.

Figure 18: Queries 1 and 2 performance in Scenario 3

Again, each query is dramatically faster with cube design 3 than with cube designs 1 or 2. Notice, however, that the effect of aggregations is different in scenario 3 than in scenarios 1 or 2. The aggregation in the data measure group is of much less benefit because the aggregation is not substantially smaller than the facts themselves (see Figure 18). The aggregation in the intermediate measure group, however, is of substantial benefit with all three cube designs. This is because the size of the aggregation is a small fraction of the size of the facts themselves (see Figure 18). The aggregation is approximately 90% smaller than the fact themselves due to the change we made to the ratio of sales reasons to sales reason types from 1:1.5 to 1:20. With this 1:20 ratio, many more sales reasons can be aggregated to sales reason types with scenario 3 than with either scenario 1 or 2. But, cube design 3 remains the performance winner.

In summary, cube design 3 with an aggregation in both measure groups yields the best performance with queries 1 and 2—a total performance improvement of approximately 90%, from approximately 30 seconds to approximately 3 seconds. The performance of cube design 3 with both aggregations is still approximately twice as fast as cube design 1 with both aggregations.

Queries 3 and 4

Figure 19 shows the performance numbers for queries 3 and 4 for relational data warehouse scenario 2 for all three cube designs.

Figure 19: Queries 3 and 4 performance in Scenario 3

Each of these queries is dramatically faster with cube design 3 than with either cube designs 1 or 2, and the effect of the intermediate aggregation with queries 3 and 4. Notice also that, with cube design 2 and with the intermediate aggregation, queries 3 and 4 have approximately the same performance as cube design 3 with the same aggregation. Furthermore, as the amount of data retrieved from the partition decreases the better the performance of cube design 2 (compare the results for query 3 versus query 4). Indeed, if we query for a single quarter in 2002 rather than 2003 (2002 is a smaller partition than 2003 – see Figure 18), cube design 2 performs almost four times faster than cube design 3. However, as you will see with queries 5 and 6 in Figure 29, this performance gain disappears when multiple partitions are touched.

In summary, cube design 3 with an aggregation in both measure groups yields the best performance with queries 3 and 4 – approximately 92% total performance improvement, from approximately 30 seconds to approximately 2 seconds. The performance of cube design 3 with both aggregations yields approximately the same performance with these two queries as cube design 2 with both aggregations.

Queries 5 and 6

Figure 20 shows the performance numbers for queries 3 and 4 for relational data warehouse scenario 2 for all three cube designs.

Figure 20: Queries 5 and 6 performance in Scenario 3

Notice again that, in similar fashion to queries 5 and 6 with scenarios 1 and 2, when a query touches more than one partition, cube design 3 with aggregations on both measure groups yields the best performance. In summary, cube design 3 with an aggregation in both measure groups yields the best performance with queries 5 and 6—a total performance improvement of approximately 90%, from approximately 30 seconds to approximately 3 seconds.

##### Scenario 4

The results for each of these six queries for relational data warehouse scenario 4 are discussed below. This scenario has 50 million members in the dimension table, 50 million rows in the main fact table, with a ratio of sales reasons to sales reasons types of 1 to 1.5.

Queries 1 and 2

Figure 21 shows the performance numbers for queries 1 and 2 for relational data warehouse scenario 4 for all three cube designs.

Figure 21: Queries 1 and 2 performance in Scenario 4

Each of these queries is somewhat faster with cube design 3, which uses the matrix relationship optimization technique, than with either cube design 1 or 2. Notice, however, that unlike scenarios 1, 2, or 3, performance improvement is much less dramatic, with or without aggregations. The aggregation in the data measure group is of little benefit because the aggregation is still very large, as is the aggregation in the intermediate fact table; the run-time join simply takes a long time given the large data sizes. The aggregation in the intermediate measure group has little benefit because the M2M ratio is close to one (1:1.5) in contrast to scenario 3 (1:20). Notice that with cube design 2, query times increase over cube designs 1 and 3, with our without aggregations.

In summary, cube design 3 with an aggregation in both measure groups yields the best performance with queries 1 and 2—a total performance improvement of approximately 40%, from approximately 110 seconds to approximately 70 seconds.

Queries 3 and 4

Figure 22 shows the performance numbers for queries 3 and 4 for relational data warehouse scenario 4 for all three cube designs.

Figure 22: Queries 3 and 4 performance in Scenario 4

Notice again with queries 3 and 4, similar to scenarios 1, 2 and 3, that when only a single partition is being touched by a query, cube design 2 yields excellent performance. As mentioned earlier, the performance footprint for cube design 2 improves as the amount of data retrieved from a single partition is small compared to the total amount of data in all partitions. Query 4 shows the most difference because query 4 retrieves the least data, which results in the fast join.

In summary, cube design 3 with an aggregation in both measure groups yields a performance improvement with queries 3 and 4 over cube design 1 of approximately 42% total performance improvement, from approximately 85-90 seconds to approximately 50 seconds. The performance of cube design 2 with both aggregations yields somewhat better performance, reducing the response times to approximately 40 and 22 seconds respectively for queries 3 and 4.

Queries 5 and 6

Figure 23 shows the performance numbers for queries 5 and 6 for relational data warehouse scenario 4 for all three cube designs.

Figure 23: Queries 5 and 6 performance in Scenario 4

In similar fashion to queries 5 and 6 with scenarios 1, 2 and 3, when a query touches more than one partition, cube design 3 with aggregations on both measure groups yields the best performance and cube design 2 suffers. In summary, cube design 3 with an aggregation in both measure groups yields the best performance with queries 3 and 4—a total performance improvement of approximately 38%, from approximately 108 seconds to approximately 66 seconds.

#### Conclusions

M2M support in SSAS is invaluable for many business scenarios, but the reality is that when dealing with large volumes of data, the performance implications of M2M queries can be debilitating when implemented at scale. This best practices article demonstrates that optimizing M2M relationships through a method of compressing the common relationships between the M2M dimension and the data measure group, and then defining aggregations on both the data measure group and the intermediate measure group yields the best query performance.

The results show dramatic improvement in M2M query performance as the reduction in the size of the intermediate measure group increases. This reduction in size can be achieved by using the matrix relationship optimization technique, aggregations, or both. The greater the amount of compression, the greater the performance benefits—particularly when both optimization techniques are used together. These benefits persist as additional fact data is added to the main fact table (and into the data measure group).

The results show that the improvement in M2M query performance based on the partitioning of the intermediate measure group to be dramatic for queries that only touch a single partition, but these benefits disappear quickly when multiple partitions are touched. However, for queries that touch many partitions, this optimization technique yields a small reduction in performance.

The performance gains achieved through these two optimization techniques reduce the overall impact of M2M queries on the SSAS query server, which leads to more scalability and overall query performance. The performance of small (or normally fast‑running) queries can be impacted by long running queries (including long-running M2M queries). For more information on the interaction of concurrently running queries, see [SSAS small/big query interaction](http://blogs.msdn.com/sqlcat/archive/2007/10/19/ssas-small-big-query-interaction.aspx).

Although exemplified through a version of the Adventure Works sample database and cubes in this white paper, these approaches have been tried and proven in several customer scenarios. Data validation has been performed and proven both here and within customer environments. The benefits are realized by users who need these SSAS data relationships to understand corporate information and make better decisions, without performance inhibitors.

For more information:

[SQL Server Customer Advisory Team blog](http://blogs.msdn.com/sqlcat)

[SQL Server Customer Advisory Team Best Practices site](http://www.microsoft.com/technet/prodtechnol/sql/bestpractice)

[SQL Customer Advisory Team site](http://sqlcat.com)

[SQL Server Web site](http://www.microsoft.com/sql/default.mspx)

[SQL Server TechCenter](http://technet.microsoft.com/en-us/sqlserver/default.aspx)

[SQL Server DevCenter](http://msdn2.microsoft.com/en-us/sqlserver/default.aspx)

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 screenshots, clear writing, or another reason?
* Are you rating it low due to poor examples, fuzzy screenshots, unclear writing?

This feedback will help us improve the quality of best practices articles we release. [Send feedback](mailto:sqlfback@microsoft.com?subject=White%20Paper%20Feedback:%20Analysis%20Services%20Many-to-Many%20Dimensions:%20Query%20Performance%20Optimization%20Techniques).

#### Appendix A: Aggregation Percentage

You can calculate the percentage of detail rows that can be aggregated in a fact table. The following Transact-SQL query calculates the expected aggregation percentage for the FactInternetSalesReason table in the AdventureWorksDW sample database when defining an aggregation that includes the Sales Reason Type attribute.

SELECT ((SELECT COUNT (\*)FROM FactInternetSalesReason) - (SELECT COUNT (\*)FROM (

SELECT f1.SalesOrderNumber, f1.SalesOrderLineNumber, d.SalesReasonReasonType, COUNT (\*) c\_srt

FROM FactInternetSalesReason f1, DimSalesReason d

WHERE f1.SalesReasonKey = d.SalesReasonKey

GROUP BY f1.SalesOrderNumber, f1.SalesOrderLineNumber, d.SalesReasonReasonType

) a)) \* 100.0 / (SELECT COUNT (\*)FROM FactInternetSalesReason)

#### Appendix B: Calculating the Reduction %

Before implementing the matrix relationship optimization technique, you can calculate the potential reduction percentage of the M2M relationship before undertaking the implementation costs. The following Transact-SQL query can be executed against any dataset containing a M2M relationship and will calculate the expected reduction percentage:

SELECT (SELECT COUNT(\*) FROM [INT MG]) OriginalRecordCount

, COUNT(MatrixKey) \* AVG(CAST(KeyCount AS FLOAT)) CompressedRecordCount

, (1 - CAST(COUNT(MatrixKey) \* AVG(CAST(KeyCount AS FLOAT)) AS FLOAT)/(SELECT COUNT(\*) FROM [INT MG])) \* 100 ReductionPerc

FROM ( SELECT DISTINCT COUNT(\*) KeyCount, MatrixKey =

( SELECT [M2M DIM #1 KEYS] AS [data()]

FROM [INT MG]

WHERE [INT MG].[JOIN KEY #1] = s.[JOIN KEY #1]

AND [INT MG].[JOIN KEY #2] = s.[JOIN KEY #2]

ORDER BY [M2M DIM #1 KEYS]

FOR XML PATH ('') )

FROM [INT MG] s

GROUP BY [JOIN KEY #1], [JOIN KEY #2]

) SUBQ

Here’s a description of each query part:

|  |  |
| --- | --- |
| Query Part | Description |
| [INT MG] | The intermediate measure group that is a candidate for matrix |
| [M2M DIM #1 KEYS] | Key(s) of the M2M dimension (assumes the primary key and foreign key are the same name) |
| [JOIN KEY #1] | The key(s) that join the intermediate back to the main fact table |

For the AdventureWorksDW relational data warehouse sample, the query looks like this:

SELECT (SELECT COUNT(\*) FROM FactInternetSalesReason) OriginalRecordCount

, COUNT(MatrixKey) \* AVG(CAST(KeyCount AS FLOAT)) CompressedRecordCount

, (1 - CAST(COUNT(MatrixKey) \* AVG(CAST(KeyCount AS FLOAT)) AS FLOAT)/(SELECT COUNT(\*) FROM FactInternetSalesReason)) \* 100 ReductionPerc

FROM ( SELECT DISTINCT COUNT(\*) KeyCount, MatrixKey =

( SELECT SalesReasonKey AS [data()]

FROM FactInternetSalesReason

WHERE

FactInternetSalesReason.SalesOrderNumber = s.SalesOrderNumber

AND FactInternetSalesReason.SalesOrderLineNumber = s.SalesOrderLineNumber

ORDER BY SalesReasonKey

FOR XML PATH ('') )

FROM FactInternetSalesReason s

GROUP BY SalesOrderNumber, SalesOrderLineNumber

) SUBQ

Here are the results:

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | Original Record Count | Compressed Record Count | Reduction Percent |
| Original sample | 64,515 | 20 | 99% |
| Scenario 1 | 15,285,991 | 851,084 | 94% |
| Scenario 2 | 46,150,062 | 851,084 | 98% |
| Scenario 3 | 41,070,640 | 18,829,880 | 54% |
| Scenario 4 | 75,066,873 | 57,509738 | 23% |

If you compare these values to the actual compression that we achieved (see Table 6), you notice that this query accurately predicts the amount of compression of the intermediate fact table through the use of the matrix relationship optimization technique.

#### Appendix C: Cube Details

The following tables describe the data and aggregation sizes for each of the four scenarios in our test for each of the three cube designs.

##### Scenario 1

Table 8 describes the details of the data and aggregation sizes for each of the partitions in the main and intermediate measure groups for these three cube designs for scenario 1.

Table 8: Data and aggregation sizes for scenario 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Cube Design** | **Aggregation Design** | **2001** | **2002** | **2003** | **2004** | **All Years** |
| 1 | Data measure group fact size | 4,957 | 13,535 | 122,479 | 142,060 |  |
| Data measure group aggregation size | 4,131 | 10,915 | 95,672 | 115,764 |  |
| Data measure group aggregation reduction in size compared to fact size | 17% | 19% | 22% | 19% |  |
| Intermediate measure group fact size |  |  |  |  | 74,191 |
| Intermediate measure group aggregation size |  |  |  |  | 45,864 |
| Intermediate measure group aggregation reduction in size compared to fact size |  |  |  |  | 38% |
| 2 | Data measure group fact size | 4,957 | 13,535 | 122,479 | 142,060 |  |
| Data measure group aggregation size | 4,461 | 12,225 | 107,631 | 126,288 |  |
| Data measure group aggregation reduction in size compared to fact size | 10% | 10% | 12% | 11% |  |
| Intermediate measure group fact size | 3,273 | 5,466 | 51,416 | 64,161 |  |
| Intermediate measure group aggregation size | 1,514 | 2,574 | 32,010 | 43,135 |  |
| Intermediate measure group aggregation reduction in size compared to fact size | 54% | 53% | 38% | 32% |  |
| 3 | Data measure group fact size | 4,108 | 13,885 | 131,494 | 157,732 |  |
| Data measure group aggregation size | 527 | 1070 | 8572 | 10792 |  |
| Data measure group aggregation reduction in size compared to fact size | 87% | 92% | 93% | 93% |  |
| Intermediate measure group fact size |  |  |  |  | 3,517 |
| Intermediate measure group aggregation size |  |  |  |  | 1,704 |
| Intermediate measure group aggregation reduction in size compared to fact size |  |  |  |  | 52% |
| Intermediate measure group size comparison | Intermediate measure group fact size reduction between cube design 1 and 3 |  |  |  |  | 95% |
| Intermediate measure group aggregation size reduction between cube design 1 and 3 |  |  |  |  | 96% |

With Scenario 1, notice the following:

* The size of the intermediate fact table with cube design 3 is 95% smaller than with cube designs 1 and 2 (and the aggregation in the intermediate measure group is 96% smaller than the aggregation in cube design 1).
* The aggregation sizes for the partitions in the data measure group for cube design 3 are a small fraction of the size of their corresponding facts whereas the aggregation sizes for the partitions with cube designs 1 and 2 are a substantial fraction of the size of their corresponding facts.
* The aggregation sizes for the partitions in the intermediate measure group for all three cube designs are a substantial fraction of the size of their corresponding facts.

These observations are directly related to the performance numbers that we saw in our tests.

##### Scenario 2

Table 9 describes the details of the data and aggregation sizes for each of the partitions in the main and intermediate measure groups for these three cube designs for scenario 2.

Table 9: Data and aggregation sizes for scenario 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Cube Design** | **Aggregation Design** | **2001** | **2002** | **2003** | **2004** | **All Years** |
| 1 | Data measure group fact size | 14,839 | 40,522 | 368,548 | 425,369 |  |
| Data measure group aggregation size | 12,366 | 32,679 | 298,377 | 346,597 |  |
| Data measure group aggregation reduction in size compared to fact size | 17% | 19% | 19% | 19% |  |
| Intermediate measure group fact size |  |  |  |  | 223,551 |
| Intermediate measure group aggregation size |  |  |  |  | 136,871 |
| Intermediate measure group aggregation reduction in size compared to fact size |  |  |  |  | 39% |
| 2 | Data measure group fact size | 14,839 | 40,522 | 368,548 | 425,369 |  |
| Data measure group aggregation size | 13,356 | 36,600 | 322,247 | 378,106 |  |
| Data measure group aggregation reduction in size compared to fact size | 10% | 10% | 13% | 11% |  |
| Intermediate measure group fact size | 9,881 | 16,502 | 159,701 | 197,263 |  |
| Intermediate measure group aggregation size | 4,576 | 7,807 | 87,524 | 116,768 |  |
| Intermediate measure group aggregation reduction in size compared to fact size | 54% | 53% | 45% | 40% |  |
| 3 | Data measure group fact size | 9,603 | 42,570 | 393,858 | 472,568 |  |
| Data measure group aggregation size | 527 | 1,123 | 9,024 | 11,409 |  |
| Data measure group aggregation reduction in size compared to fact size | 95% | 97% | 98% | 98% |  |
| Intermediate measure group fact size |  |  |  |  | 4,093 |
| Intermediate measure group aggregation size |  |  |  |  | 1,704 |
| Intermediate measure group aggregation reduction in size compared to fact size |  |  |  |  | 58% |
| Intermediate measure group size comparison | Intermediate measure group fact size reduction between cube design 1 and 3 |  |  |  |  | 98% |
| Intermediate measure group aggregation size reduction between cube design 1 and 3 |  |  |  |  | 99% |

With Scenario 2 (very similar to Scenario 1), notice the following:

* The size of the intermediate fact table with cube design 3 is 98% smaller than with cube designs 1 and 2 (and the aggregation in the intermediate measure group is 99% smaller than the aggregation in cube design 1).
* The aggregation sizes for the partitions in the data measure group for cube design 3 are a small fraction of the size of their corresponding facts whereas the aggregation sizes for the partitions with cube designs 1 and 2 are a substantial fraction of the size of their corresponding facts.
* The aggregation sizes for the partitions in the intermediate measure group for all three cube designs are a substantial fraction of the size of their corresponding facts.

These observations are directly related to the performance numbers that we saw in our tests.

##### Scenario 3

Table 10 describes the details of the data and aggregation sizes for each of the partitions in the main and intermediate measure groups for these three cube designs for scenario 3.

Table 10: Data and aggregation sizes for scenario 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Cube Design** | **Aggregation Design** | **2001** | **2002** | **2003** | **2004** | **All Years** |
| 1 | Data measure group fact size | 1,010 | 2,756 | 25,052 | 28,926 |  |
| Data measure group aggregation size | 808 | 2134 | 19479 | 22,498 |  |
| Data measure group aggregation reduction in size compared to fact size | 20% | 23% | 22% | 22% |  |
| Intermediate measure group fact size |  |  |  |  | 240,329 |
| Intermediate measure group aggregation size |  |  |  |  | 18,946 |
| Intermediate measure group aggregation reduction in size compared to fact size |  |  |  |  | 92% |
| 2 | Data measure group fact size | 1,010 | 2,756 | 25,052 | 28,926 |  |
| Data measure group aggregation size | 909 | 2,400 | 21,913 | 25,712 |  |
| Data measure group aggregation reduction in size compared to fact size | 10% | 13% | 13% | 11% |  |
| Intermediate measure group fact size | 5,543 | 15,616 | 145,958 | 192,322 |  |
| Intermediate measure group aggregation size | 557 | 1,470 | 13,418 | 17,711 |  |
| Intermediate measure group aggregation reduction in size compared to fact size | 90% | 91% | 91% | 91% |  |
| 3 | Data measure group fact size | 1,111 | 2,934 | 26,783 | 32,140 |  |
| Data measure group aggregation size | 808 | 2134 | 9,246 | 9,982 |  |
| Data measure group aggregation reduction in size compared to fact size | 27% | 27% | 65% | 69% |  |
| Intermediate measure group fact size |  |  |  |  | 93,587 |
| Intermediate measure group aggregation size |  |  |  |  | 8,686 |
| Intermediate measure group aggregation reduction in size compared to fact size |  |  |  |  | 91% |
| Intermediate measure group size comparison | Intermediate measure group fact size reduction between cube design 1 and 3 |  |  |  |  | 61% |
| Intermediate measure group aggregation size reduction between cube design 1 and 3 |  |  |  |  | 54% |

With Scenario 3, notice the following:

* The size of the intermediate fact table with cube design 3 is 61% smaller than with cube designs 1 and 2 (and the aggregation in the intermediate measure group is 54% smaller than the aggregation in cube design 1). This reduction is size is less than with scenarios 1 and 2, but is still significant.
* The aggregation sizes for the partitions in the data measure group for cube design 3 are a larger fraction of the size of their corresponding facts than we saw with scenarios 1 and 2, whereas the aggregation sizes for the partitions with cube designs 1 and 2 continue to constitute a substantial fraction of the size of their corresponding facts.
* The aggregation sizes for the partitions in the intermediate measure group for all three cube designs, unlike scenarios 1 and 2, are a small fraction of the size of their corresponding facts. This is because of the change in the size ratio between the main fact table and intermediate fact tables in this scenario.

These observations are directly related to the performance numbers that we saw in our tests.

##### Scenario 4

Table 11 describes the details of the data and aggregation sizes for each of the partitions in the main and intermediate measure groups for these three cube designs for scenario 4.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Cube Design** | **Aggregation Design** | **2001** | **2002** | **2003** | **2004** | **All Years** |
| 1 | Data measure group fact size | 24,722 | 67,509 | 620,175 | 708,665 |  |
| Data measure group aggregation size | 20,602 | 54,442 | 497,096 | 577,431 |  |
| Data measure group aggregation reduction in size compared to fact size | 17% | 19% | 20% | 19% |  |
| Intermediate measure group fact size |  |  |  |  | 440,742 |
| Intermediate measure group aggregation size |  |  |  |  | 218,752 |
| Intermediate measure group aggregation reduction in size compared to fact size |  |  |  |  | 50% |
| 2 | Data measure group fact size | 24,906 | 67,505 | 616,355 | 708,665 |  |
| Data measure group aggregation size | 23,074 | 60,976 | 556,747 | 656,171 |  |
| Data measure group aggregation reduction in size compared to fact size | 7% | 10% | 10% | 7% |  |
| Intermediate measure group fact size | 18,656 | 29,824 | 289,314 | 355,042 |  |
| Intermediate measure group aggregation size | 7144 | 13917 | 137106 | 177325 |  |
| Intermediate measure group aggregation reduction in size compared to fact size | 62% | 53% | 53% | 50% |  |
| 3 | Data measure group fact size | 16,714 | 70,069 | 673,807 | 790,100 |  |
| Data measure group aggregation size | 7,870 | 26,594 | 301,112 | 396,962 |  |
| Data measure group aggregation reduction in size compared to fact size | 53% | 62% | 55% | 50% |  |
| Intermediate measure group fact size |  |  |  |  | 336,972 |
| Intermediate measure group aggregation size |  |  |  |  | 162,219 |
| Intermediate measure group aggregation reduction in size compared to fact size |  |  |  |  | 52% |
| Intermediate measure group size comparison | Intermediate measure group fact size reduction between cube design 1 and 3 |  |  |  |  | 24% |
| Intermediate measure group aggregation size reduction between cube design 1 and 3 |  |  |  |  | 26% |

Table 11: Data and aggregation sizes for scenario 4

With Scenario 4, notice the following:

* The size of the intermediate fact table with cube design 3 is only 24% smaller than with cube designs 1 and 2 (and the aggregation in the intermediate measure group is only 26% smaller than the aggregation in cube design 1).
* The aggregation sizes for the partitions in the data measure group for cube design 3 are approximately one-half the size of their corresponding facts whereas the aggregation sizes for the partitions with cube designs 1 and 2 are a substantial fraction of the size of their corresponding facts.
* The aggregation sizes for the partitions in the intermediate measure group for all three cube designs are a substantial fraction of the size of their corresponding facts.

These observations are directly related to the performance numbers that we observe in our tests of the MDX queries against each of these scenarios.