

The many-to-many revolution

Advanced dimensional modeling with SQL Server Analysis Services 2005

Author: Marco Russo

Published: Version 1.0 Revision 93 – September 25, 2006

Contact: marco.russo@sqlbi.com – <http://www.sqlbi.com>

Summary: This paper describes how to leverage the many-to-many dimension relationships, a feature that debuted available with Analysis Services 2005. After introducing the main concepts, the paper discusses various implementation techniques in the form of design patterns: for each model, there is a description of a business scenario that could benefit from the model, followed by an explanation of its implementation.

A separate download (available on <http://www.sqlbi.com/manytomany.aspx>) contains SQL Server database and Analysis Services projects with the same sample data used in this paper

Acknowledgments: I would like to thank the many peer reviewers that helped me to make this document more readable: Bryan Batchelder, Chris Webb, Darren Gosbell, Greg Galloway, Jon Axon, Mark Hill, Peter Koller, Sanjay Nayyar, Scott Barrett.

A special mention goes to Teo Lachev, who made the deepest review.

I would also like to thank T.K. Anand and Marin Bezic, who answered to my questions.

Finally, special thanks are reserved for Claudio Civetta, Marcella Caviola and other people working for Cedacri spa (Italy): they trusted me enough to adopt many of the described patterns in a production environment and gave me an invaluable feedback.

Table of Contents

The many-to-many revolution	i
Table of Contents	iii
Introduction	1
Classical many-to-many relationship	2
Business scenario.....	2
Implementation	2
Cascading many-to-many relationship	7
Business scenario.....	7
Implementation	11
Survey	20
Business scenario.....	21
Implementation	22
Distinct Count	31
Business scenario.....	31
Implementation	32
Performance.....	47
Multiple Groups	50
Business scenario.....	50
Implementation	52
Cross-Time	57
Business scenario.....	57
Implementation	58
Transition Matrix	66
Business scenario.....	66
Implementation	68
Multiple Hierarchies	73
Business scenario.....	73
Implementation	77
Conclusions	84
Links	84

Introduction

Analysis Services 2005 (SSAS 2005) introduced the capability to handle many-to-many relationships between dimensions. At a first glance, you may tend to underscore the importance of this feature: after all, Analysis Services 2000 and many other OLAP engines do not offer many-to-many relationships. Yet, its lack did not limit their adoption and, apparently, only a few businesses really require it. However, as this paper shows, the UDM (Unified Dimensional Model) can leverage many-to-many relationships helping you to present data from different perspectives that are not feasible with a traditional star schema. This opens a brand new world of opportunities that transcends the limits of traditional OLAP.

We will explore many different uses of many-to-many relationships that give us more choices to model effectively business needs, including:

- Classical many-to-many
- Cascading many-to-many
- Survey
- Distinct Count
- Multiple Groups
- Cross-Time
- Transition Matrix
- Multiple Hierarchies

Although you do not have to do so, I recommend you to read the models in the order presented above, because often each one builds upon the previous models.

Each model has a brief introduction, followed by a business scenario that may benefit of its use and an explanation of its implementation. Each model uses only the minimal set of dimensions that are necessary to explain the concept behind it and a small dataset that demonstrates the underlying behavior.

Only the Distinct Count scenario contains a section discussing the impact on performance. Since the considerations presented there may be applied to other many-to-many relationship uses, I recommend you read this scenario if you are interested in performance evaluations.

An important warning has to be made if you are going to use VisualTotals MDX function (directly or through an OLAP browser): visual totals apply only to one level at a time with many-to-many dimensions. In the Links section, you will find a link to a document written by Richard Tkachuk that explains this limitation.

Classical many-to-many relationship

By the title "Classical many-to-many relationship", I mean the common situation that may benefit from many-to-many relationships. It is fundamental to understand very well how many-to-many relationships work within SSAS 2005 in order to be able to use them for different purposes: minor implementation details such as the relationships between dimensions and measure groups could have major design repercussions since small changes may lead to different results and confusion to the end users.

Business scenario

Here is a typical business scenario: you have a fact table that describes a measure (in this case an account balance taken at a given point of time) for a given entity (a bank account) that can be joined to many members of another dimension (a joint account owned by several customers). Those of you familiar with the "classical" multidimensional model can already see the difficulty, because it is not easy to describe the non-aggregability of measures joined to dimensions with a many-to-many relationship (in this case, each bank account can have one or more owners and each owner can have one or more accounts).

Implementation

With SSAS 2005, the issue simply does not exist. The "trick" is to introduce an intermediate fact table that, in the relational model, defines the many-to-many relationship. Typically, this "special" fact table has no measures. Following the Ralph Kimball's methodology we can name it a "factless fact table" or "bridge table"; for historical reasons I will use the term "factless" more than "bridge" in this document, but the term "bridge" is more appropriate. In Figure 1 fact tables are colored in yellow while dimension tables are shown in blue.

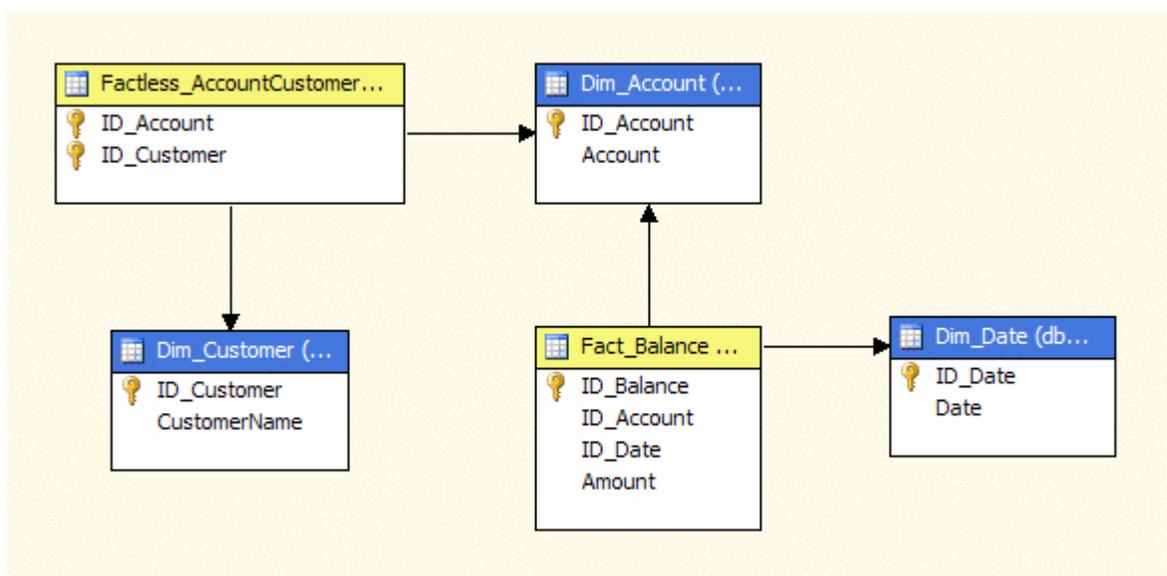


Figure 1 – Relational model with many-to-many relationship

When you define relationships between dimensions and measure groups, you specify that Dim Customer is joined to Fact Balance through the Factless Account Customer

measure group (as defined by the selected item in Figure 2). Please note that Figure 2 shows the results of the “auto build” feature of the Cube wizard: the wizard does a good job in this case, but in subsequent models we will take these relationships one-step further by adding other dimension-measure group relationships manually.

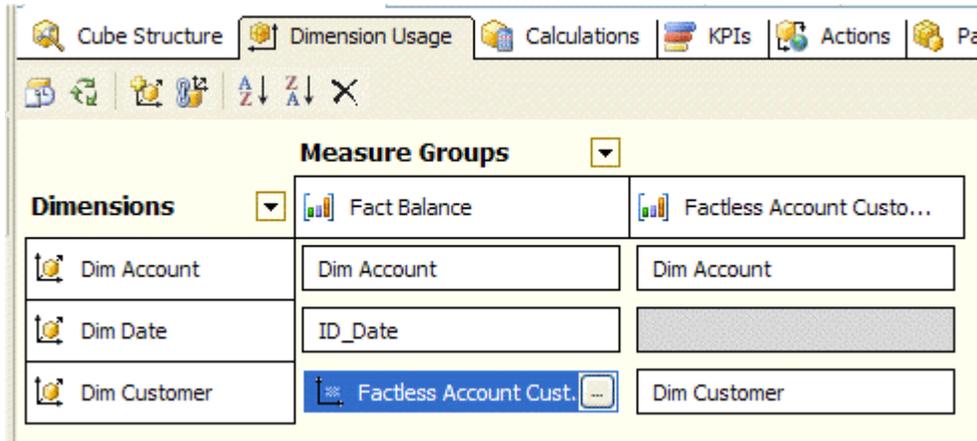


Figure 2 – UDM with many-to-many relationship

The many-to-many relationship is further described by the “Define Relationship” dialog box (Figure 3) that is displayed when you click on the button contained in the intersecting cell (the highlighted cell in Figure 2). This dialog box is available for every combination between dimensions and measure groups and it allows you to select the type of relationship.

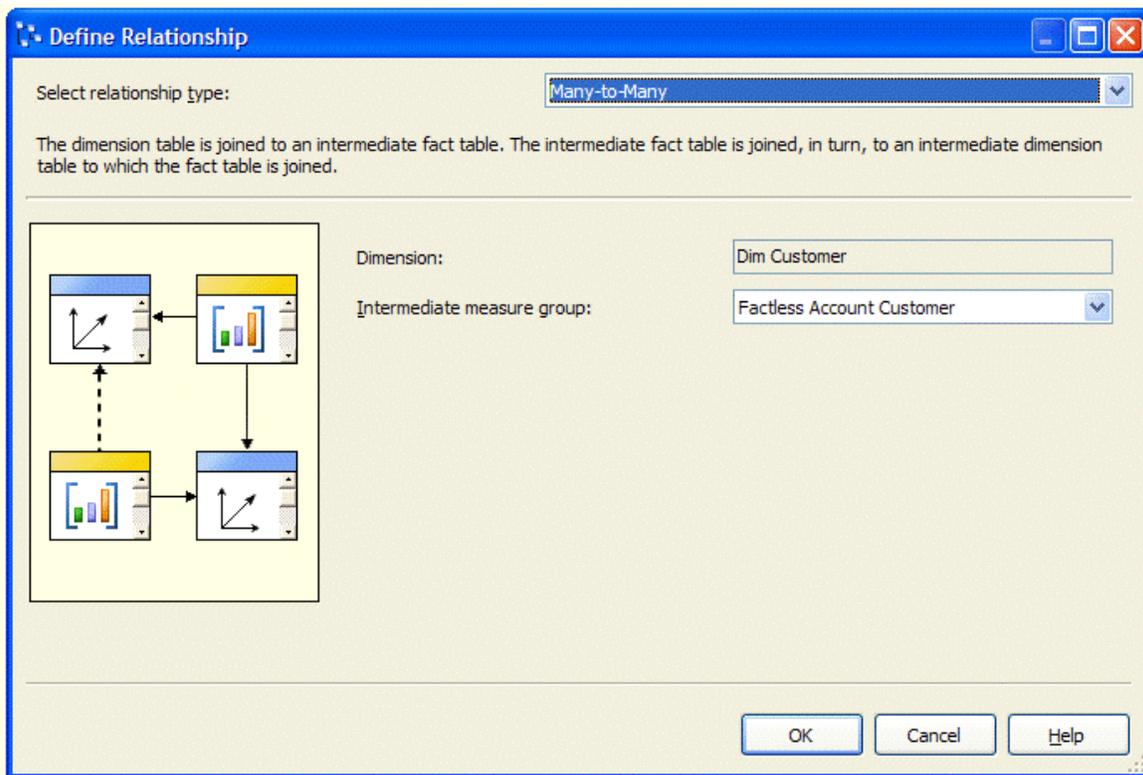


Figure 3 – Many-to-many relationship dialog box

The result is shown in Figure 4. I created 4 customers (Luke, Mark, Paul and Robert) and 6 accounts in the test tables. Each account is joined to one or more customers (the account name is a concatenation of the account holders) and the balance for each account is always 100 at the date used by the query.

Date		Customer Name				
2005-12-31 00:00:00		Luke	Mark	Paul	Robert	Grand Total
Account	Amount	Amount	Amount	Amount	Amount	Amount
Luke	100					100
Mark		100				100
Mark-Paul		100	100			100
Mark-Robert		100			100	100
Paul			100			100
Robert					100	100
Grand Total	100	300	200	200	600	

Figure 4 – Many-to-many relationship result

As you can see, for each customer you can identify the accounts that she owns and for each account you can see the balance repeated for each owner... but the total for each account (row) is always 100 (Grand Total row) and the balance for all accounts is 600 (100 * 6). Try to do that with many other OLAP tools and see what happens...

Figure 5 synthesizes the particular aggregability of some measures in respect of Dim Customer.

Date		Drop Column Fields Here	
2005-12-31 00:00:00		Customer Name	Amount
Customer Name	Amount	Fact Balance Count	
Luke	100	1	
Mark	300	3	
Paul	200	2	
Robert	200	2	
Grand Total	600	6	

Figure 5 – Measures aggregation by Customer Name

You can obtain the same result with AS2000 but only with some stunts and tradeoffs in terms of processing time or query performance (compared to results you can obtain with SSAS 2005).

Now we have to consider the count measure that is available in the Factless Amount Customer measure group. It seems to be very similar to the Fact Balance Count measure, but it has an important difference that we can better observe with different queries. Let us look at data related to Jan-06 in Figure 6.

Date					
2006-01-31 00:00:00					
Customer Name					
	Luke	Mark	Paul	Robert	Grand Total
Account	Amount	Amount	Amount	Amount	Amount
Luke	105				105
Mark		105			105
Mark-Robert		105		105	105
Paul			105		105
Robert				105	105
Grand Total	105	210	105	210	525

Figure 6 – Account Mark-Paul is missing in Jan-06 data

The balance for the account Mark-Paul is missing from January 2006 in the fact table, so we do not see a corresponding row.

Drop Filter Fields Here							
Date							
2005-12-31 00:00:00				2006-01-31 00:00:00		Grand Total	
Customer Name	Fact Balance Count	Factless Account Customer Count	Fact Balance Count	Factless Account Customer Count	Fact Balance Count	Factless Account Customer Count	
Luke	1	1	1	1	2	1	
Mark	3	3	2	3	5	3	
Paul	2	2	1	2	3	2	
Robert	2	2	2	2	4	2	
Grand Total	6	8	5	8	11	8	

Figure 7 – Counts with no relationship between Dim Date and Factless

Figure 7 shows query results for the different count measures we have in our cube. The Fact Balance Count measure counts rows in Fact Balance measure group: in this query it represents how many balances are present for each customer into a given period. Since each account has only one balance for each month, it could also be mistaken for the number of accounts that a customer has, but the Grand Total proves that this assumption is incorrect. Conversely, this information (the number of accounts for each customer) is correctly provided by the Factless Account Customer Count measure, which obtains this value directly counting the number of rows in Factless Account Customer measure group. This number is time invariant from a date, because its measure group has no relationship with the time dimension (Dim Date).

Drop Filter Fields Here							
Date							
2005-12-31 00:00:00				2006-01-31 00:00:00		Grand Total	
Customer Name	Fact Balance Count	Factless Account Customer Count	Fact Balance Count	Factless Account Customer Count	Fact Balance Count	Factless Account Customer Count	
Luke	1	1	1	1	2	1	
Mark	3	3	2	2	5	3	
Paul	2	2	1	1	3	2	
Robert	2	2	2	2	4	2	
Grand Total	6	8	5	6	11	8	

Figure 8 – Counts with many-to-many relationship between Dim Date and Factless

In Figure 8 we can see that numbers have changed somewhat (changes are highlighted) as a result of a relationship between Dim Date and Factless Account Customer (as we can see in Figure 9). Now the correct interpretation of the Factless Account Customer Count measure is that it represents the number of combinations between customers and accounts having at least one balance in the considered period. This explains the lower value in Jan-06 for Mark and Paul (a corresponding account balance is missing in that month) while the Grand Total is not affected (it includes both Dec-05 and Jan-06, so the account Mark-Paul has at least one balance).

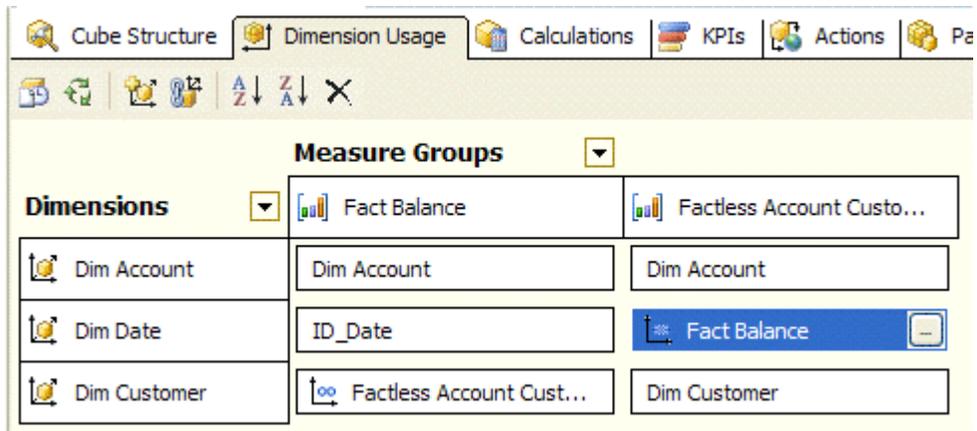


Figure 9 – Many-to-many relationship between Dim Date and Factless Account Customer

I encourage you to experiment with your data using many-to-many relationships. This will help you to understand the implications of not having a relationship between a dimension and a “factless” (or “bridge”) measure group. It is only when you really master this concept at this simple level (only two measure groups involved) that you can really go further with advanced dimensional modeling techniques, which leverages on many-to-many relationships.

Cascading many-to-many relationship

When we apply the many-to-many relationship several times in a cube, we have to pay attention if there is a chain of many-to-many relationships. As we have seen in the classical many-to-many relationship scenario, dimensions that apparently do not relate to a “factless” measure group could be meaningful and important to enhance the analytical capabilities of our model.

We will call this situation “cascading many-to-many relationship”.

Business scenario

A typical scenario is the case when a dimension far from the main fact table (a dimension that is only directly related to a factless fact table) is involved into an existing many-to-many relationship and has another many-to-many relationship with another dimension. This is a very common case when you have data based on questionnaire containing questions with multiple choices (fast forward to Figure 10 to see a sample dimensional schema).

For example, consider this slightly modified bank account scenario, with a different fact that we want to consider:

- Account transactions
 - ⇒ Transactions fact table related to Dim Date, Dim Account and Dim Type
- Each account can have one or more owners (customers)
 - ⇒ Dim Account has many-to-many relationship with Dim Customer
- Each customer can be classified into one or more categories
 - ⇒ Dim Customer has many-to-many relationship with Dim Categories

Although I could have used the previous balance accounts scenario, the new schema adds the Dim Type dimension so we need to use the many-to-many relationship in a bidirectional way.

I need to describe the sample data that we will use in our implementation. Table 1 shows the denormalized fact table we will use. Even if the Date dimension is not strictly necessary for this explanation, I will keep it in the model because it is a common dimension in a similar scenario and it is useful to see how it relates to the other dimensions.

Table 1 – Fact table transaction data

Account	Type	Date	Amount
Mark	Cash deposit	20051130	1000.00
Paul	Cash deposit	20051130	1000.00
Robert	Cash deposit	20051130	1000.00
Luke	Salary	20051130	1000.00
Mark-Robert	Salary	20051130	1000.00
Mark-Paul	Cash deposit	20051130	1000.00
Mark	ATM withdrawal	20051205	-200.00
Robert	Credit card statement	20051210	-300.00
Paul	Credit card statement	20051215	-300.00
Luke	ATM withdrawal	20051215	-200.00

More important for our purposes is the Type dimension: it describes the type of a transaction and it is useful to group transactions across other dimensions. For example, these are common question that a user could be interested to get an answer:

- What is the salary/income for the "IT enthusiast" category?
- How many different transaction types are used by the "Rally driver" category?
- What customer categories have ATM withdrawal transactions?

Within the transaction fact data, there is not enough information to provide answers to those questions. Therefore, we need a supplementary table holding further information. Table 2 contains the relationship existing between customers and categories in our sample data.

Table 2 – Customers-categories relationship

Customer	Category
Mark	IT enthusiast
Robert	IT enthusiast
Paul	Rally driver
Robert	Rally driver
Luke	Traveler
Mark	Traveler
Paul	Traveler
Robert	Traveler

To give an answer to the first question we need an additional clarification. If we consider the accounts owned by only one person, then there are no customers belonging to the "IT enthusiast" category who get a salary income; but if we consider joint accounts (e.g. Mark and Robert both own the same account), then their owners receive a salary income. From Mark's perspective, he receives a salary income of 1000.

On the other side, Robert gets a salary income of 1000 too! However, unfortunately for them, from the perspective of "IT enthusiast" category we cannot count the same salary income two times, so the "IT enthusiast" salary income is still 1000 and not 2000! So the tough reality is that Mark and Robert have to share this

```
SELECT SUM( ft.Amount ) AS Amount
FROM Fact_Transaction ft
INNER JOIN Dim_Type dt
  ON dt.ID_Type = ft.ID_Type
  AND dt.Type = 'Salary'
WHERE ID_Account IN (
  SELECT ID_Account
  FROM Factless_CustomerCategory fcc
  INNER JOIN Dim_Category dc
    ON dc.ID_Category = fcc.ID_Category
  INNER JOIN Factless_AccountCustomer ac
    ON ac.ID_Customer = fcc.ID_Customer
  WHERE CategoryName = 'IT enthusiast'
)
```

single salary income because we have no other way to know (basing on our data) which of them is really receiving this income as the transaction is recorded against their joint account. This problem is very common in bank environment: one of the possible SQL query solutions presented beside demonstrates how this kind of problems cannot be easily tackled with a generic query builder that many users are used to working with (see the subquery in the WHERE condition of the main SQL query). For this reason, we would like to resolve similar questions with a pivot table.

Now we can try to solve the second question. There are two customers who belong to the "Rally driver" category, Paul and Robert; these two customers own 4 accounts, which in our fact table gets any transaction type other than "ATM withdrawal". Therefore, the answer will be 3 transaction types: Cash deposit (for an amount of 3000), Salary (1000) and Credit card statement (-600.00). The SQL query construct could be very similar to the previous one.

```
SELECT COUNT( DISTINCT ft.ID_Type ) AS
TransactionTypes
FROM Fact_Transaction ft
WHERE ID_Account IN (
    SELECT ID_Account
    FROM Factless_CustomerCategory fcc
    INNER JOIN Dim_Category dc
        ON dc.ID_Category = fcc.ID_Category
    INNER JOIN Factless_AccountCustomer ac
        ON ac.ID_Customer = fcc.ID_Customer
    WHERE CategoryName = 'Rally driver'
)
```

The third question requires a different approach: starting from a set of transactions (filtered by type) we need to get related customers and then related categories. In such a case a query builder could give us a working query, but it should be noted how potentially slow the query could be, because it could generate a large set of rows before applying the DISTINCT clause. The SQL query could be optimized but in a way that is difficult to obtain with a generic query builder. Even this time a working pivot table would be a dream that becomes reality for an end user.

```
SELECT DISTINCT dc.CategoryName
FROM Fact_Transaction ft
INNER JOIN Dim_Type dt
    ON dt.ID_Type = ft.ID_Type
    AND dt.Type = 'ATM withdrawal'
INNER JOIN Factless_AccountCustomer fac
    ON fac.ID_Account = ft.ID_Account
INNER JOIN Factless_CustomerCategory
fcc
    ON fcc.ID_Customer = fac.ID_Customer
INNER JOIN Dim_Category dc
    ON dc.ID_Category = fcc.ID_Category
```

Now we have enough requirements to design and test a multidimensional model that enables a pivot table to solve this kind of problems with a few clicks.

Implementation

Figure 10 shows the relational schema of our model: we have two bridge tables (or factless fact tables) that join two “cascading” many-to-many relationships, the first one between Dim Account and Dim Customer and the second one between Dim Customer and Dim Category.

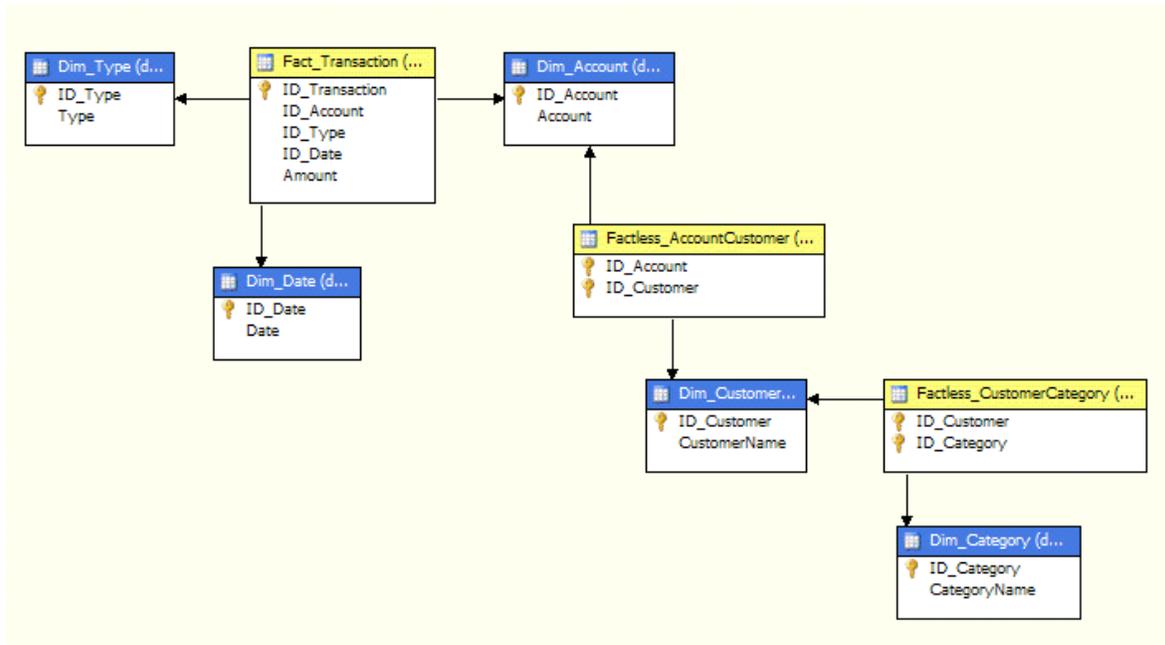


Figure 10 – Relational model with cascading many-to-many relationships

If we create the cube with the auto build feature of Cube Wizard we end up with a model that correctly identify dimension and fact tables. However, the problem of missing relationships between dimensions and measure groups we have already seen in the previous scenario is amplified here, as we can see in Figure 11. The wizard is not able to find cascading many-to-many relationships and a reason for this behavior is that defining all the many-to-many relationships could negatively affect performance.

Dimensions	Fact Transaction	Factless Account Customer	Factless Customer Categ...
Dim Type	Dim Type		
Dim Date	ID_Date		
Dim Account	Dim Account	Dim Account	Factless Account Custo...
Dim Customer	Factless Account Custo...	Dim Customer	Dim Customer
Dim Category			Dim Category

Figure 11 – Dimension relationship obtained by cube wizard/auto build feature

Unfortunately, the many gray boxes that are present in Figure 11 produce meaningless results when you query dimension and measure group at corresponding coordinates. For example (see Figure 12), we cannot see the amount of transactions for each customer category and things are worse when we try to split the Amount measure by transaction type (see Figure 13).

Drop Filter Fields Here	
	Drop Colour
Category Name ▾	Amount
IT enthusiast	5000
Rally driver	5000
Traveler	5000
Grand Total	5000

Figure 12 – Categories are not related to amount measure

Drop Filter Fields Here						
	Type					
Category Name	Amount	Amount	Amount	Amount	Amount	Amount
IT enthusiast	-400	4000	-600		2000	5000
Rally driver	-400	4000	-600		2000	5000
Traveler	-400	4000	-600		2000	5000
Grand Total	-400	4000	-600		2000	5000

Figure 13 – Categories still do not split amount measure

At this point, the problem seems to be the missing relationship between Dim Category and Fact Transaction measure group: so we click on the button in the gray box and in the Define Relationship dialog box select the only available intermediate measure group for the Many-to-Many relationship type we choose (Figure 14 better summarize this selection).

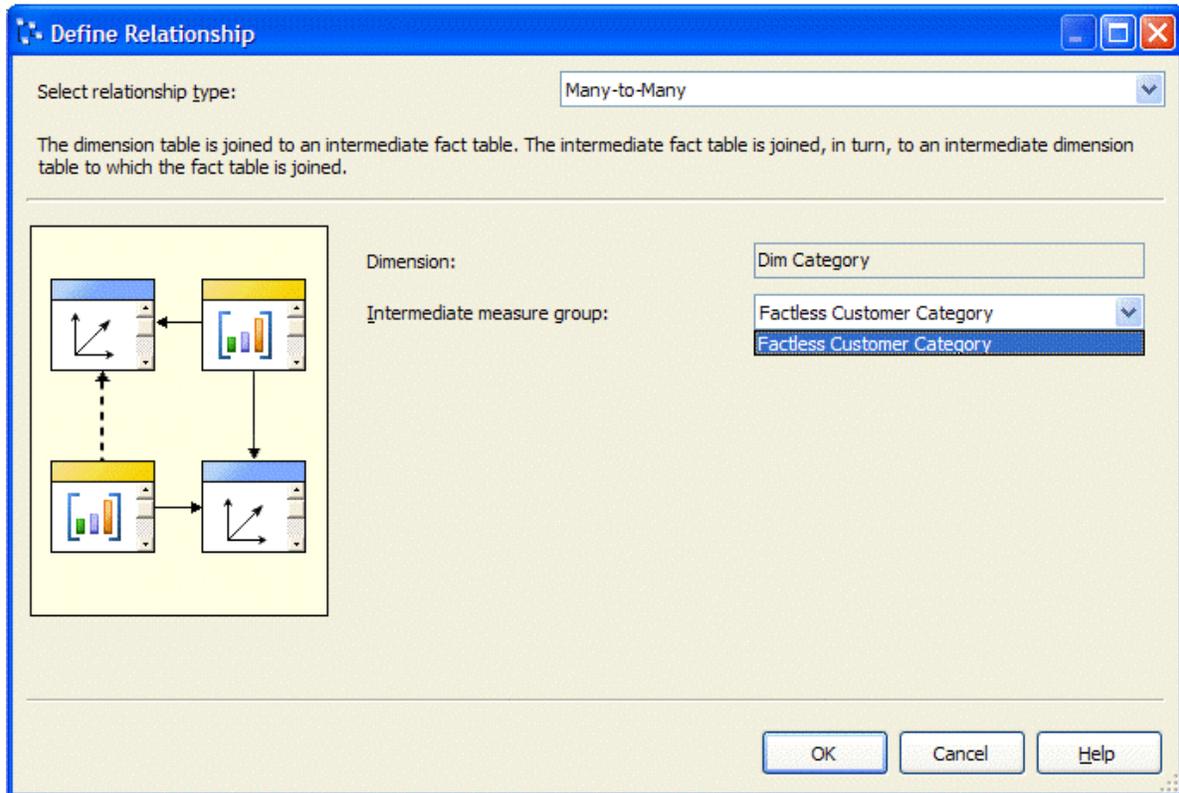


Figure 14 – Intermediate measure groups available for Dim Category

Now we can reprocess the cube, but the results will be the same and wrong as Figure 12 and Figure 13 show. Before claiming it is a bug of Analysis Services (it is not), look at the new dimension relationship summary in Figure 15. There are still a lot of gray boxes and the intermediate measure group between Dim Category and Fact Transaction is different than the one between Dim Customer and Fact Transaction (one is Factless Customer Category and the other is Factless Account Customer).

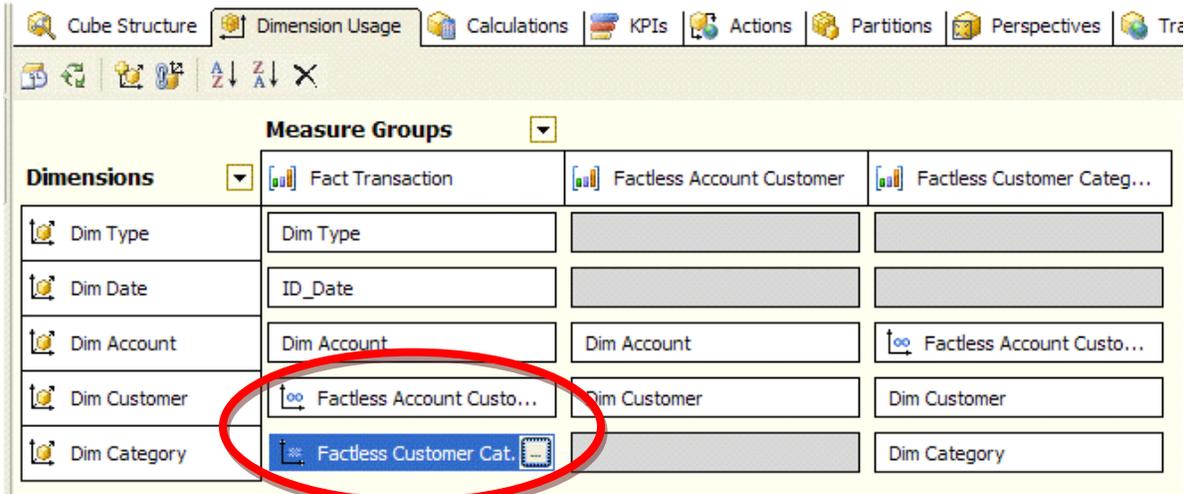


Figure 15 - Dimension relationship after Dim Category manual definition

To understand what is happening and why, you need to realize that Analysis Services entities like dimensions and measure groups are totally separated and disconnected from the underlying relational schema. Subsequently, Analysis Services has no sufficient information to relate correctly customer categories with account transactions. We told Analysis Services that a category is related to account transactions through the Factless Customer Category measure group, but to go from a category to a transaction we need to get all the customers for that category (Factless Customer Category) and then all the accounts for this set of customers (through Factless Account Customer). Now the problem should be clear: when have not informed Analysis Services about the relationship between Dim Category and Factless Account Customer. For this reason, it is still a gray box. We can fill this void by clicking on the ... button: this time our dialog box shows up two possible intermediate measure groups (Figure 16).

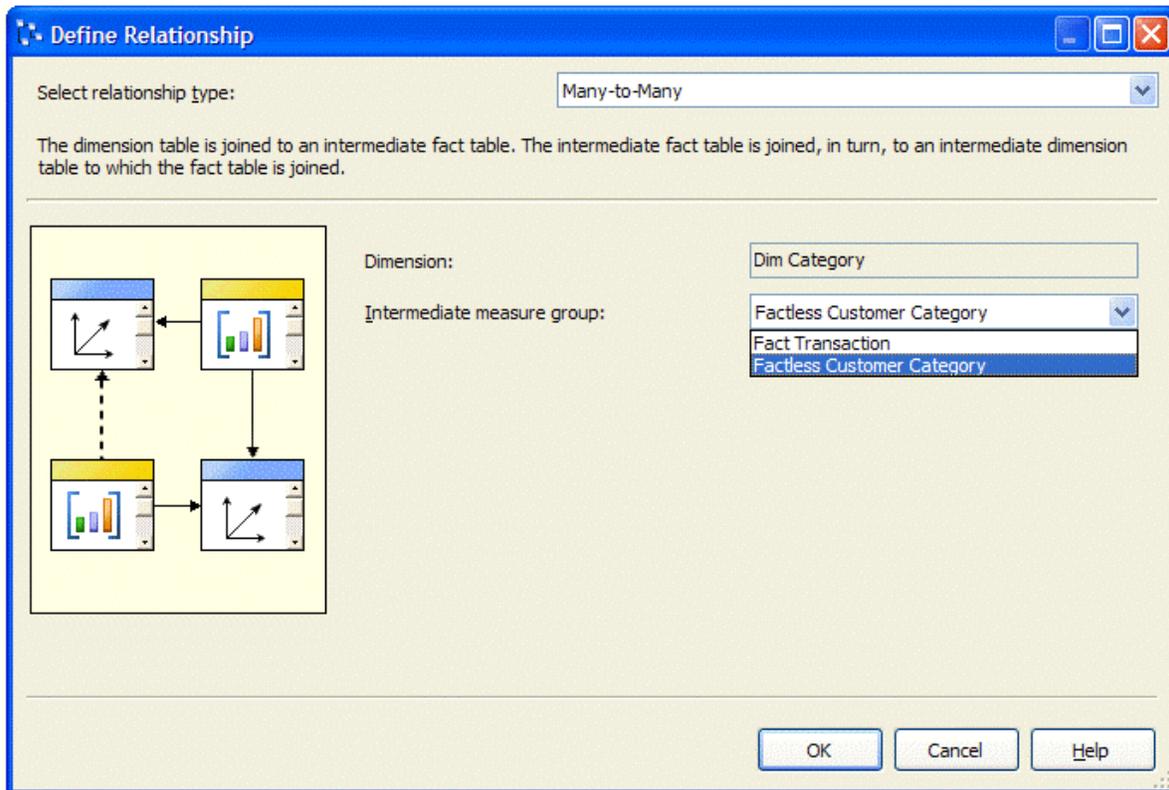


Figure 16 – Difficult choice for Dim Category intermediate measure group

We need to choose Factless Customer Category as intermediate measure group, because this is the only possible factless fact table that we traverse walking from Dim Customer to Factless Account Customer into the relational schema (Figure 10). However, why does the “Intermediate measure group” dropdown include the Fact Transaction as a possible intermediate measure group? Simply because we previously defined a (wrong) relationship between Dim Category and Fact Transaction using Factless Customer Category as intermediate measure group (review Figure 14). If we would return at the stage immediately after the Cube Wizard, we would have seen only one choice (the right one) defining a many-to-many relationship between Dim Category and Factless Account Customer.

At this point, we still need to correct the relationship between Dim Category and Fact Transaction: it has to be Factless Account Customer instead of Factless Customer Category we have previously chosen. Now if we redefine this relationship the dropdown lists both choices, because Dim Category has many relationships with other measure groups. The resulting dimension usage schema is summarized in Figure 17.

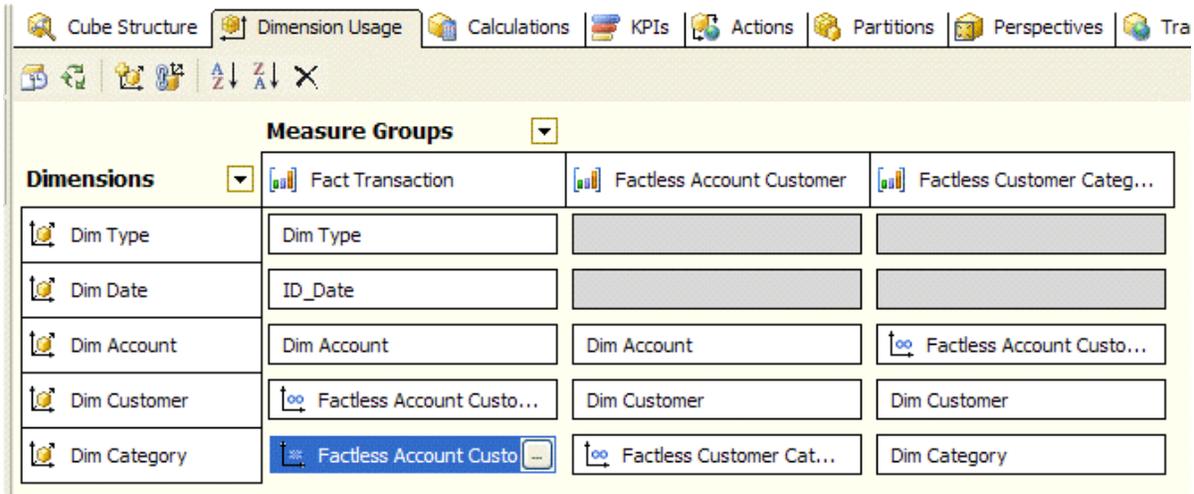


Figure 17 – Correct Dim Category many-to-many relationship assignments

To verify that this is correct, we can retry the queries we failed in Figure 12 and Figure 13. This time we get the correct numbers, as we can see in Figure 18 and Figure 19.

Drop Filter Fields Here	
Drop Column Fields Here	
Category Name ▼	Amount
IT enthusiast	3500
Rally driver	3400
Traveler	5000
Grand Total	5000

Figure 18 – Categories are correctly related to amount measure

Drop Filter Fields Here		Type ▼				
		ATM withdrawal	Cash deposit	Credit card statement	Salary	Grand Total
Category Name ▼	Amount	Amount	Amount	Amount	Amount	Amount
IT enthusiast	-200	3000	-300	1000	3500	
Rally driver		3000	-600	1000	3400	
Traveler	-400	4000	-600	2000	5000	
Grand Total	-400	4000	-600	2000	5000	

Figure 19 – Categories correctly split amount measure

Now, take a cup of your favorite coffee and fix well in your mind what you are learning here: it will save you a lot of time when your favorite cube will start to have several cascading many-to-many relationship. The cascading many-to-many relationships is a fundamental concept on which the rest of the models described in this document are built.

The relationship between a dimension and a measure group is used to tell Analysis Services how to relate dimension members with fact measures. When the relationship is regular, it is simple. When the relationship is many-to-many, **the intermediate measure group must refer to a measure group that contains a valid relationship with a dimension that is related via a regular relationship to the target measure group.** This should explain why choices were good or bad in my previous examples. In this example, Factless Account Customer measure group had to be used to relate Dim Category to Fact Transaction measure group as that is the only measure group that has a dimension (Account) that is directly related to the Fact Transaction measure group.

Official documentation explains this concept of granularity, which is formally correct but much less intuitive. In other words, when you define a many-to-many relationship between a measure group and a dimension, you have to choose the intermediate measure group (the factless fact table, or bridge table) that is nearest to the measure group, considering all the possible measure groups that you can cross going from the measure group to the considered dimension.

I think that Define Relationship dialog could be more clear and smart. For example, it could filter out the choices that are probably wrong. However, you have to consider that the cascading many-to-many relationship feature was introduced very late in the development cycle. I guess that there was not enough feedback to improve the user interface in the release version.

We should check if all other business requirements are met. Figure 20 shows the right answer to the second question (How many different transaction types are used by the "Rally driver" category?). Figure 21 answers correctly to the third question (What categories of customers have ATM withdrawal transactions?). Note that, in both cases, the Grand Total row is not the sum of previous rows and that it is coherent with the nature of the many-to-many relationship.

Category Name ▾	
Rally driver	
Drop Column Fields Here	
Type ▾	Amount
Cash deposit	3000
Credit card statement	-600
Salary	1000
Grand Total	3400

Figure 20 – Transaction types for Rally driver category

Type ▾	
ATM withdrawal	
Drop Column Fields Here	
Category Name ▾	Amount
IT enthusiast	-200
Traveler	-400
Grand Total	-400

Figure 21 – Category of customers who did ATM withdrawals

At this point, we should determine if the remaining gray boxes could still lead to issues with other queries. In fact, if we are interested in the count measure produced by the

factless fact table, they do. For example, if for whatever reason you would choose to address the third question using the Factless Customer Category Count measure instead of the Amount measure (it is not such a useful number, but you should not ask whether a number is useful while it is wrong), you would obtain the strange result of Figure 22.

Type	
ATM withdrawal	
	Drop Column Fields Here
Category Name	Factless Customer Category Count
IT enthusiast	2
Rally driver	2
Traveler	4
Grand Total	8

Figure 22 – Wrong results using Factless Customer Category Count measure

Numbers aside (in this query the measure should represent the number of customers for each category that did at least one ATM withdrawal transaction, but it does not), the category list is wrong. The reason should be obvious: there is not a valid relationship between Dim Type and Factless Customer Category measure group, which contains the measure we used in our query. At this point we must choose between making this measure invisible or fixing this measure: I prefer the second approach, because in future I may need to expand my UDM and having more defined relationships makes the cube easier to explain. I can always make these strange measures invisibles, but this is not a good excuse to leave UDM incomplete.

Dimensions	Fact Transaction	Factless Account Customer	Factless Customer Categ...
Dim Type	Dim Type	Fact Transaction	Factless Account Custo...
Dim Date	ID_Date	Fact Transaction	Factless Account Custo...
Dim Account	Dim Account	Dim Account	Factless Account Custo...
Dim Customer	Factless Account Custo...	Dim Customer	Dim Customer
Dim Category	Factless Account Custo...	Factless Customer Cat...	Dim Category

Figure 23 – Complete cube model for cascading many-to-many relationships

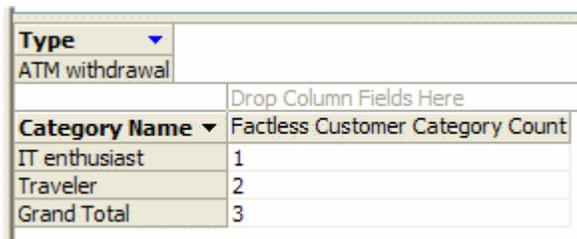
In Figure 23 I finalized the UDM dimension usage by defining relationships between all dimensions and all measure groups. Oftentimes, all many-to-many relationships (all cells) of a dimension usage column point to the same intermediate measure group. This is common because only the measure groups that are based on true factless fact table have different intermediate measure groups for different dimensions, e.g. the Factless Account Customer measure group.

I worked on complex UDMs that have different intermediate measure groups for different dimensions linked with many-to-many relationships. Sometimes it happens

even for “standard” measure groups containing real fact measures (and not only a Count measure as in the case of a factless fact table). Once you understand how to choose the correct intermediate measure group for a dimension, you should be able to handle similar situations.

It is important to point out that removing all “gray cells” in the dimension usage matrix is not necessarily a “best practice” you should follow in all cases. Maintaining all these relationships in an evolving cube (it is normal to add dimensions and measure groups over time in the real life) could be extremely difficult and error-prone. Do it only when it is necessary. Even in this paper, there are scenarios that do not require a complete dimension usage matrix. A simple rule of thumb follows: if you want to make visible any measure derived by an intermediate measure group (corresponding to a bridge table), you have to define dimensions relationships for all intermediate measure groups in a chain that connect the measure to other interesting dimensions, even if the visible measure is only a row count (the only measure you should get from a real factless fact table).

Now we can get the right answer for the third question (What customer categories have ATM withdrawal transactions?) even with the Factless Customer Category Count measure, as we can see in Figure 24.



Type	
ATM withdrawal	
	Drop Column Fields Here
Category Name	Factless Customer Category Count
IT enthusiast	1
Traveler	2
Grand Total	3

Figure 24 – Right results using Factless Customer Category Count measure

Once you have mastered cascading many-to-many relationships, you definitely gain the ability to create richer multidimensional models.

Survey

The survey scenario is a common example of a more general case where you have a lot of attributes associated to a case (one customer, one product, and so on) and you want to normalize the model because you do not want to change the UDM each time you add a new attribute to data (as adding a new dimension or changing an existing one). One common scenario is a questionnaire consisting of questions that have predefined answers with both simple and multiple choices. The typical star schema model (one fact table with answers joined with a questions/answers dimension and a case dimension) is fully queryable using SQL. However, once you move to UDM that things become harder: while it is very simple to compare different answers to the same question, it could be very difficult to relate answers to more than one question. For example, if we have a question asking for sports practiced (multiple choices) and another one asking for job performed, probably we would like to know what relationships exists between those two attributes. The normal way to solve it is to have two different attributes (or dimensions) that users can combine on rows and columns of a pivot table. Unfortunately, having an attribute for each question is not very flexible; more important, you have to change your star schema to accommodate having a single row into the fact table for each case. This makes it very difficult to handle any multiple choices question.

Instead, we can change our perspective and leverage many-to-many relationships. We can build a finite number (as many as we want) of questions/answers dimensions, duplicating many times the original one and providing to the user a number of "filter" dimensions that can be crossed into a pivot table or can be used to filter data that, for each case, satisfy defined conditions for different questions.

Remember that the survey scenario is usable in many similar circumstances: classification of product characteristics and basket analysis are two other applicable examples of many others.

Business scenario

Let us explore the survey scenario in more details. Data was loaded into the star schema shown in Figure 25. Dim_QuestionsAnswers contains questions and answers. I could have defined two independent dimensions (resulting in a snowflake schema) but it is a choice I do not recommend for two reasons: one is the maintenance cost to update surrogate keys, the second is that there are no reasons to query Dim_Questions without Dim_Answer (typically, you will make visible only a hierarchy Question-Answer on the UDM).

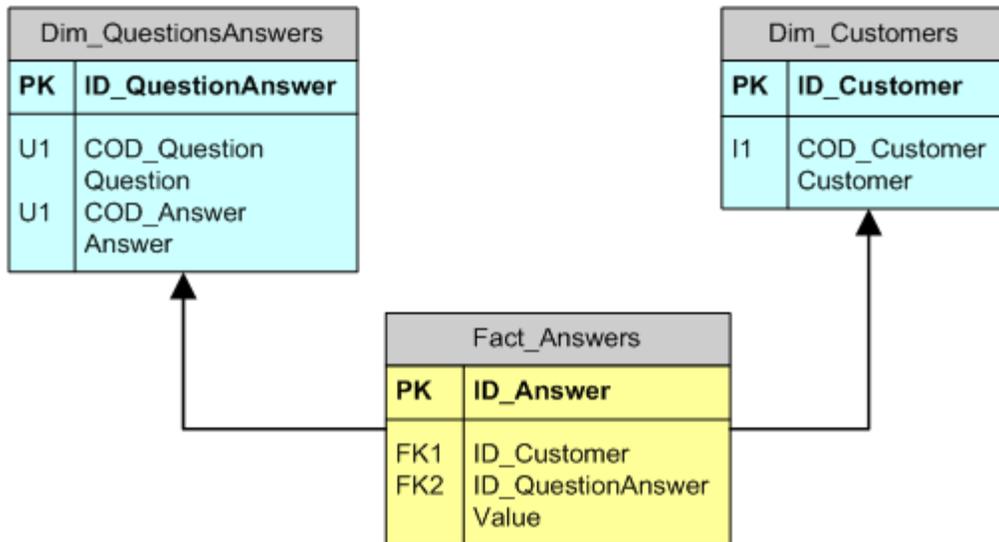


Figure 25 – Relational Survey star schema

Our users may decide to query this model in a PivotTable (like the one provided by Excel) and without the need to write a single row of MDX. A typical query could be “How many customers do play soccer and hockey?”; the side box shows a SQL solution that use a COUNT(*) expression, while a

```

SELECT COUNT(COUNT *)
FROM Fact_Answers a1
INNER JOIN Dim_QuestionsAnswers q1
  ON q1.ID_QuestionAnswer = a1.ID_QuestionAnswer
INNER JOIN Fact_Answers a2
  ON a2.ID_Customer = a1.ID_Customer
INNER JOIN Dim_QuestionsAnswers q2
  ON q2.ID_QuestionAnswer = a2.ID_QuestionAnswer
WHERE q1.Answer = 'Soccer'
      AND q2.Answer = 'Hockey'
  
```

more correct one could be COUNT(DISTINCT ID_Customer) in a more general case (useful if you add more complex filter conditions). Adding more conditions requires new INNER JOIN (two for each condition) to the query: in other ways, it is very difficult to get a parameterized query that automatically solves this problem for us.

We want to change surveys in the future, keeping them compatible with existing data (at least for identical questions that use the same answers). One day we could add more questions and answers, without requiring a cube or dimension to full process, allowing incremental updates of any entity.

SQL does not satisfy our requirements in a simple way.

Implementation

To implement a cube based on the star schema shown in Figure 25 we define a single QuestionsAnswers dimension (see Figure 26). In this way, the user can filter rows of Fact_Answers table (or cells of the derived cube). However, what we need is completely different. Instead of filter the answers, we need to filter customers that satisfy a given condition, then filter customers that satisfy another condition and at the end we need to get the intersection between these two sets of customers.

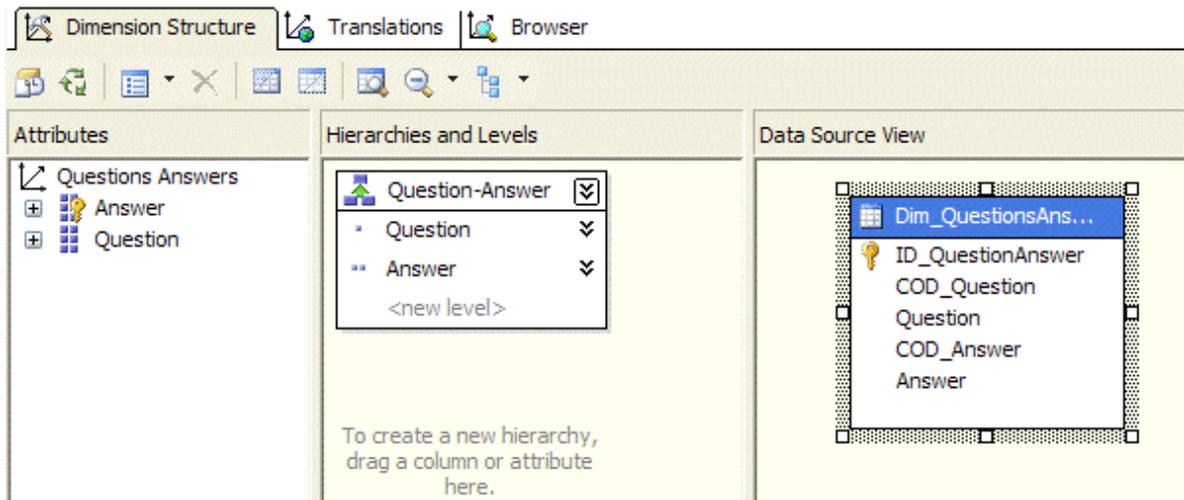


Figure 26 – Dimension QuestionsAnswers

We want to design a UDM that uses the same QuestionsAnswers dimension several times, allowing us to combine different answers and questions for the same customer. We will call the resulting dimensions as “Filter 1”, “Filter 2”, and so on. Users will be able to select any combination of those dimensions and filter on them. This will result in a query that applies all the filters (logical AND). However, the AND condition will be applied only to those fact rows that belong to the same customer. Note that we seek to evaluate the number of customers that have specific characteristics based on survey: our main fact table, in the cube, is not the Fact_Answers fact table, but the Dim_Customers itself!

To model our cube, we need to relate each customer to all answers for that customer, as we would denormalize the Fact_Answers fact table to have a column for each QuestionsAnswers member. From a practical point of view, there is a many-to-many relationship between Customers and each QuestionsAnswers dimensions (renamed to “Filter n”) we added to the cube. To do that, we use the Fact_Answers fact table as the factless fact table of a many-to-many relationship, and we use the Dim_Customers dimension table as a fact table (to get the customers count). Each “Filter” dimension use the same physical factless fact table to reference the QuestionsAnswers dimension. It is convenient to define a logical view (named query) into the Data Source View (DSV) to create N different measures groups in the cube (each one has to be related to a different table), Here, N is the number of “Filter” dimensions we have chosen. The factless fact table is repeated in the DSV, so we can use the Cube Editor for this model: normally, Visual Studio editor would not allow you to create many different measure groups based on the same fact table, unless you define a Distinct Count measure.

Alternatively, you could manually define different measure groups related to the same fact table by changing the cube XML definition.

In my example, I will use three “Filter” dimensions. Therefore, I need three aliases for the Fact_Answers fact table. I defined a named query view for each one instead of using the real fact table. Figure 27 shows the resulting DSV. The query used in the vFact_AnswersN named queries is shown aside.

```
SELECT
  ID_Answer,
  ID_Customer,
  ID_QuestionAnswer,
  Value
FROM Fact_Answers
```

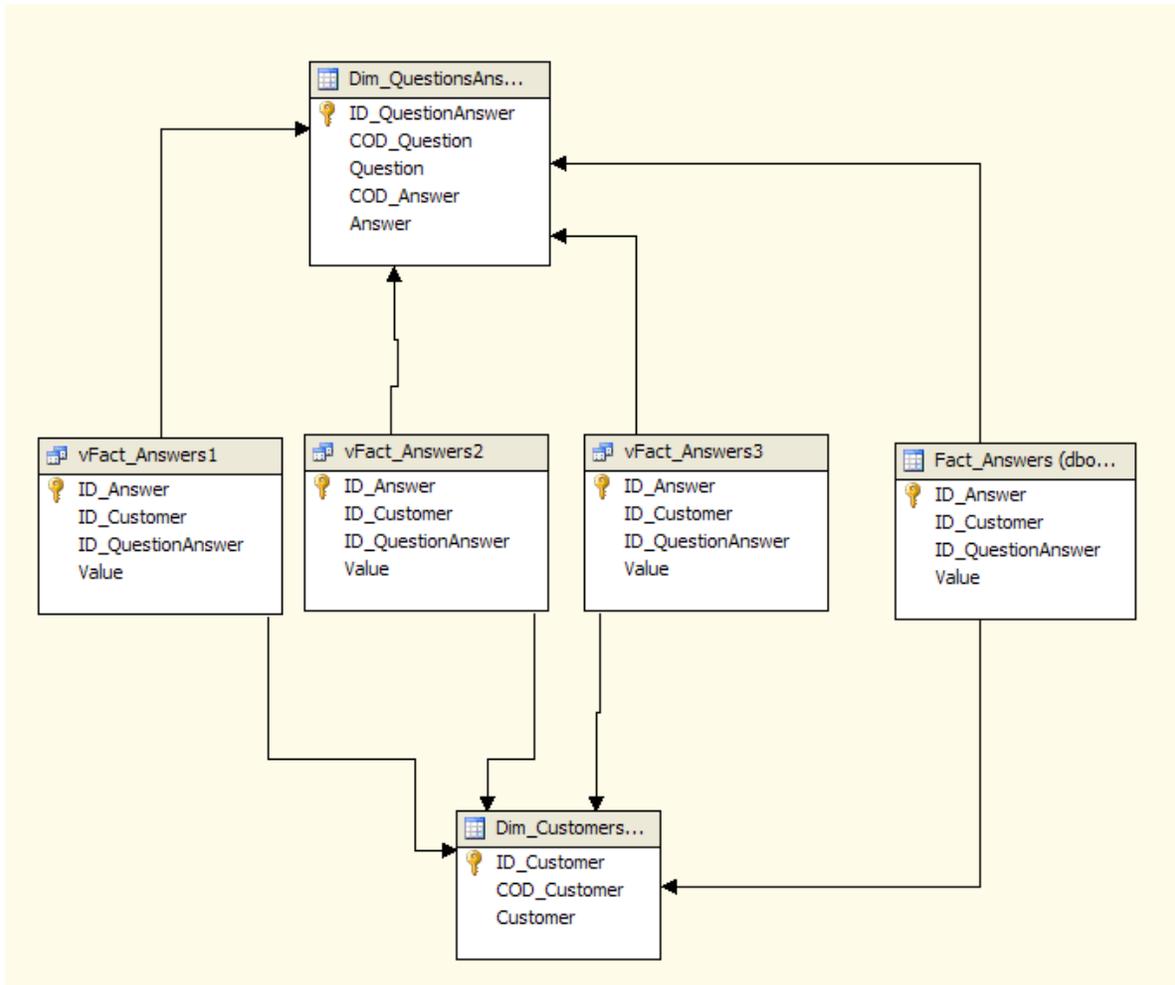


Figure 27 – Survey model Data Source View

We can use the Cube Wizard to start the cube modeling. After the first two steps (accept the defaults) we come to the Identify Fact and Dimension Tables step. We need to change the suggested selection as shown in Figure 28. We use Dim_Customers as Fact and Dimension and we excluded the Fact_Answers table (instead, we will use several named queries based on the vFact_Answers views).

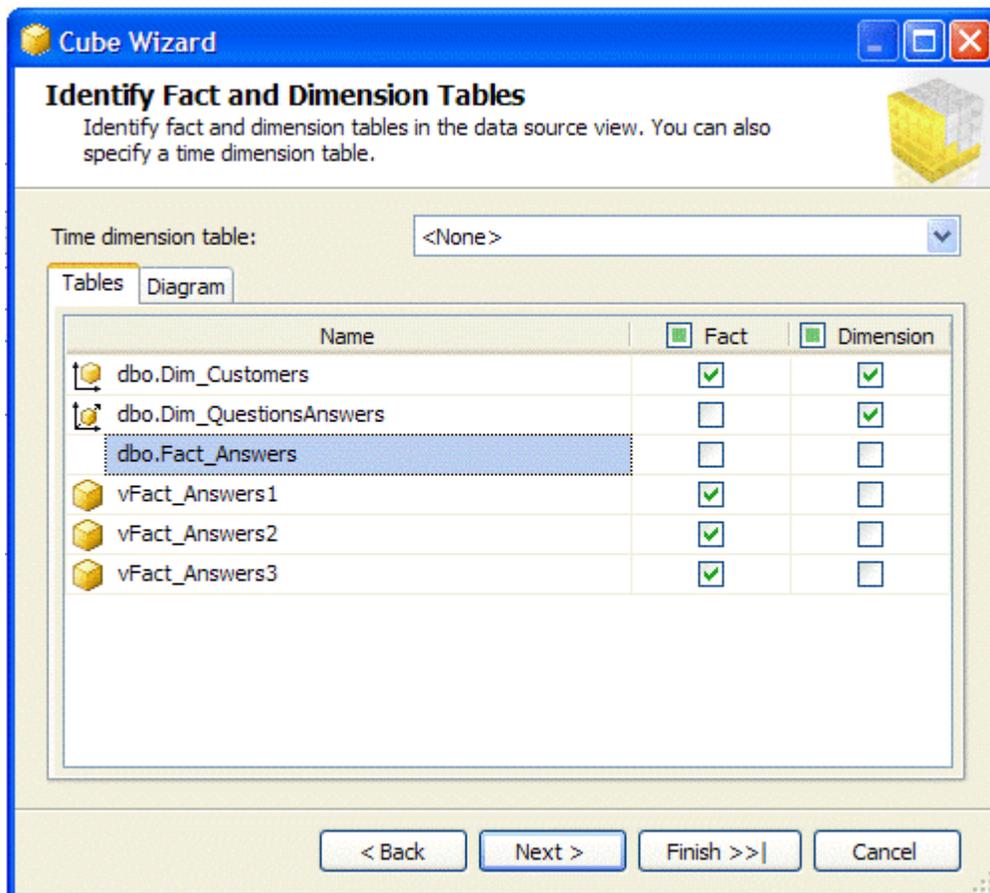


Figure 28 – Cube Wizard selection for Survey Cube

In the next step (Review Shared Dimensions), we choose all dimensions from the “available dimensions” list. In the Select Measures step that follows, we make a lifting to default measure names, as shown in Figure 29.

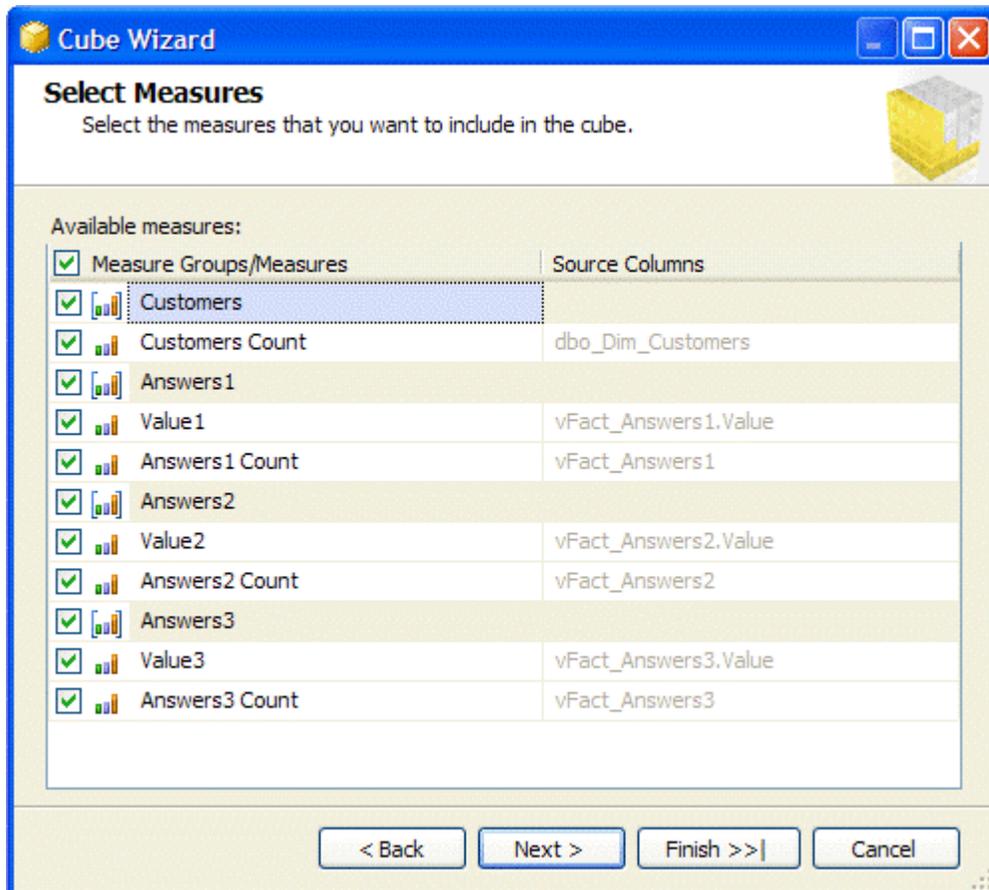


Figure 29 – Esthetic changes to measure names

We accept the defaults in the steps that follow and we name the cube Survey. Once we complete the Cube Wizard, the Cube Designer opens and shows the resulting cube structure (see Figure 30). Each AnswersN measure group will contain data needed to build three different Filter dimensions based on Questions Answers dimension.

We need to add “role-playing dimensions” to the cube to build the three Filter dimensions (shown in the Dimension pane in Figure 30).

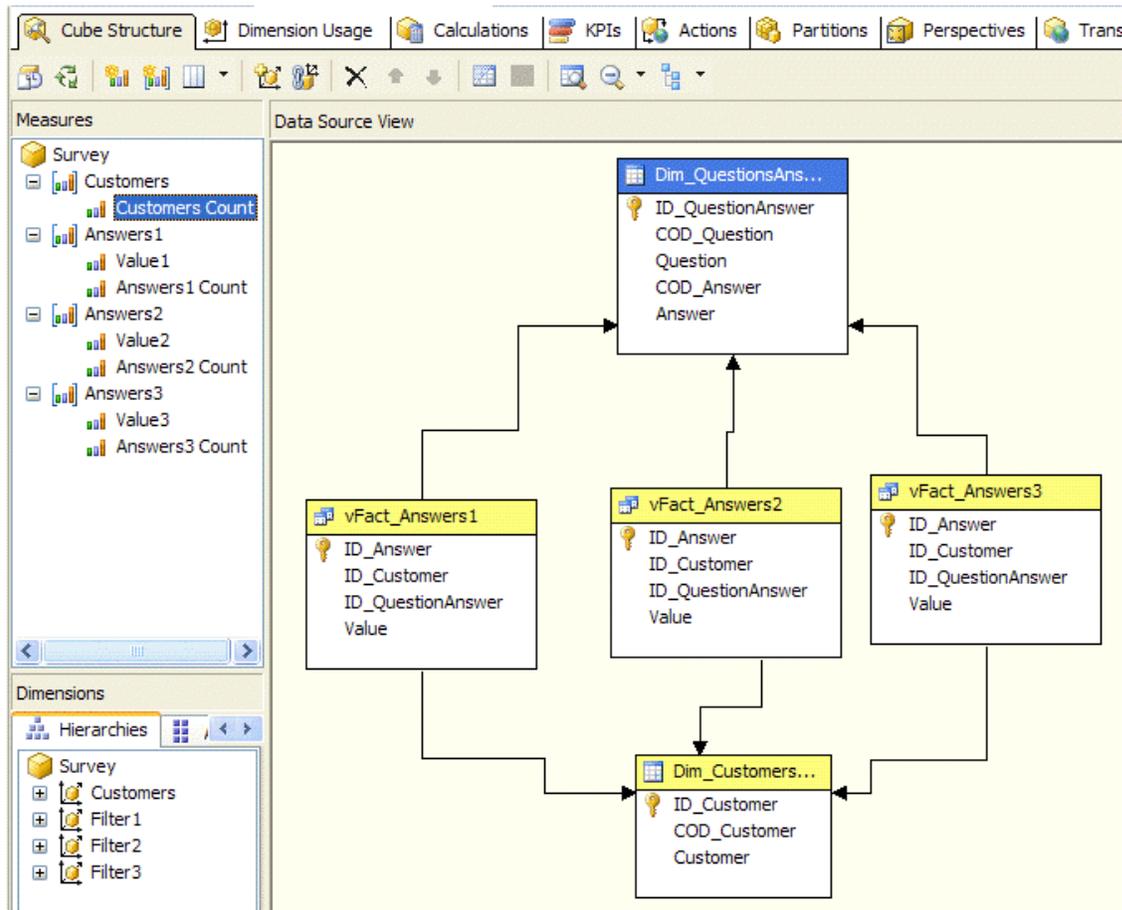


Figure 30 – Resulting Survey Cube Structure

To add a dimension we can use the Dimension Usage tab and click on Add Cube Dimension button. We add the Questions Answers dimension three times and we rename it to “FilterN”, where N is a progressive number to distinguish the filter dimension (in this case ranging from 2 to 3). The original Questions Answers dimension has to be renamed to Filter1.

As we learned in previous scenarios, we have to set up useful relationships between dimensions and measures groups. Figure 31 shows the relationships we need. If you consider the Customer Measure Group only, you realize that we have a fact dimension (Dim_Customers) related many times to Questions Answers (used three times as a role-playing dimension) through a different factless fact table each time.

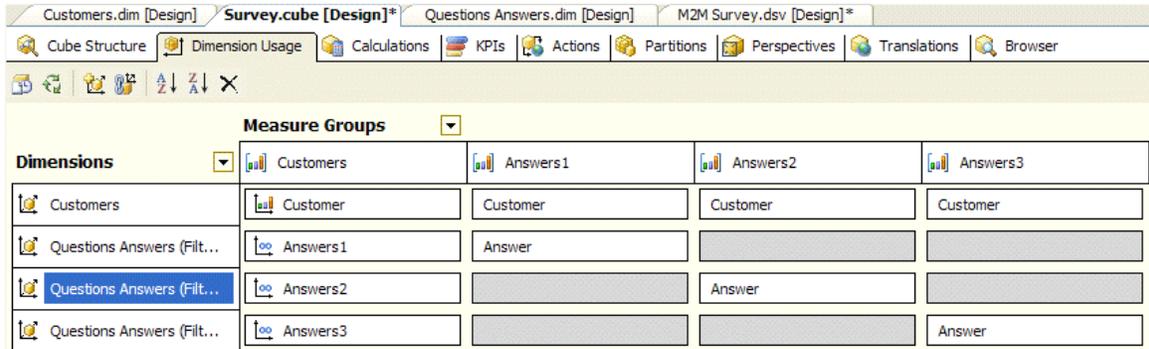


Figure 31 – Dimension Usage for Survey Cube

Before analyzing the results on Pivot Table, look at sample data I used in my test. We have four customers (Bill, Elisabeth, John and Mark) and a survey for each customer. Possible questions and answers of the survey are presented in Table 3.

Table 3 – Dim_QuestionsAnswers data

Customer	Question	Answer
Sports	Tennis	Tennis
Sports	Golf	Golf
Sports	Soccer	Soccer
Sports	Hockey	Hockey
Job	Employee	Employee
Job	Student	Student
Job	Designer	Designer
Age	Age	Age

The survey data is denormalized into a single view, visible in Table 4.

Table 4 – vFact_AnswersN data

Customer	Question	Answer
Bill	Age	28
Bill	Job	Designer
Bill	Sports	Hockey
Bill	Sports	Soccer
Elisabeth	Age	31
Elisabeth	Job	Designer
Elisabeth	Sports	Golf
Elisabeth	Sports	Tennis
John	Age	29
John	Job	Student
John	Sports	Soccer
Mark	Age	30
Mark	Job	Employee
Mark	Sports	Golf
Mark	Sports	Soccer
Mark	Sports	Tennis

We can see that only Bill plays both Soccer and Hockey.

Now we can process the cube and see if it works as expected. In the Browser pivot table we put a Filter dimension on rows and another Filter dimension on columns. In Figure 32 I selected only the answer Hockey for rows and only the answer Soccer for columns because I wanted to limit the results to a specific case.

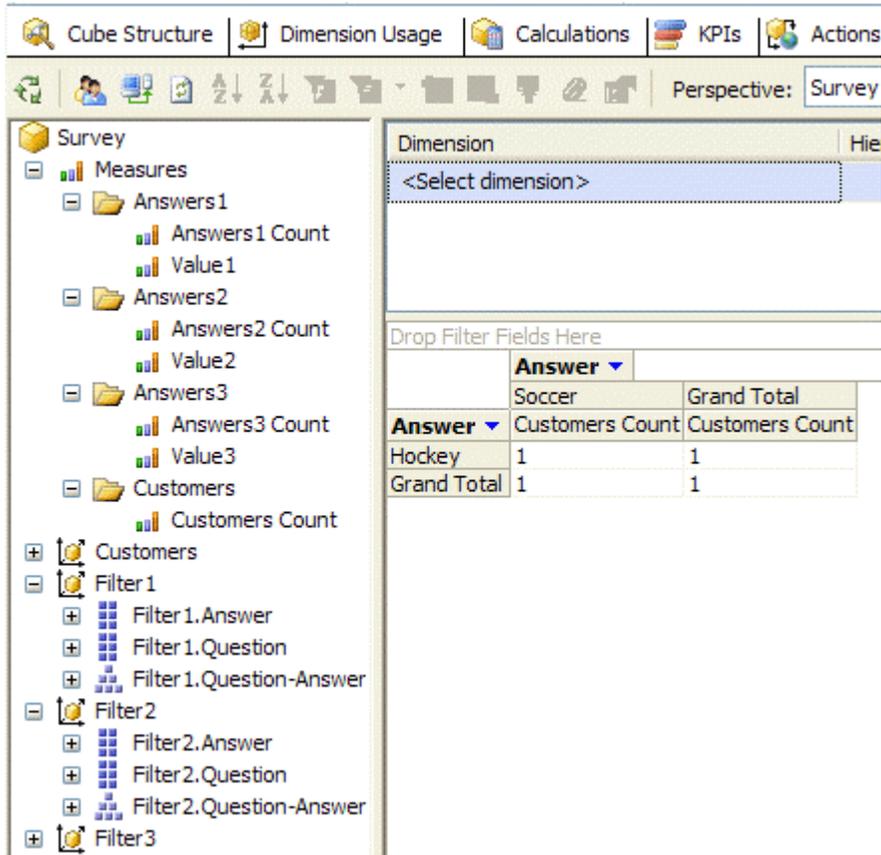


Figure 32 – Query between members of the same dimension

We can also intersect more answers and questions into the same pivot table report. Figure 33 shows that many customers plays two sports and what, what is the relationship between jobs and sports, and so on. There is a certain data redundancy because the data is mirrored diagonally from top-left to bottom-right defines. This kind of analysis is bidirectional and the order of answers provided by customer is not meaningful.

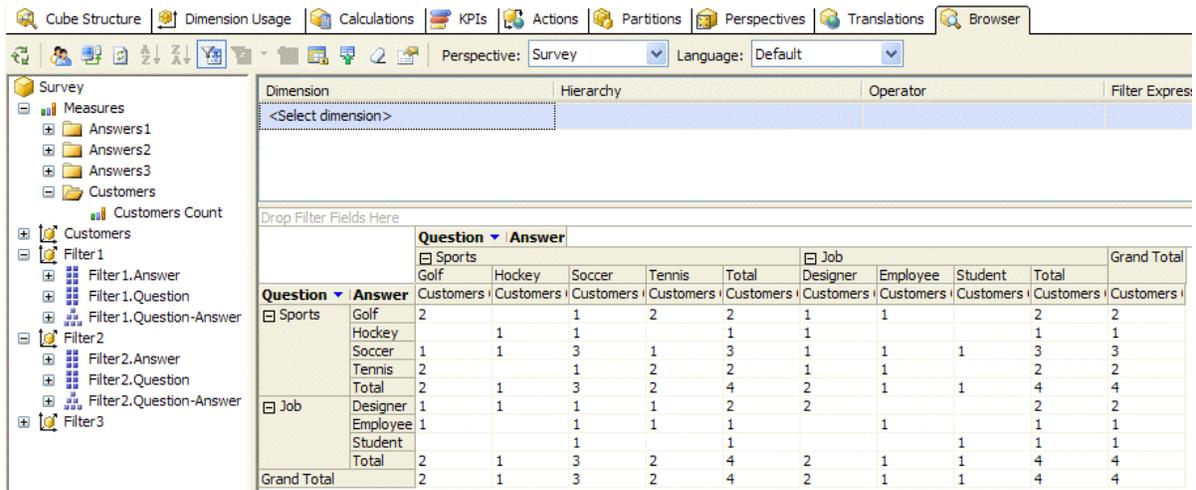


Figure 33 – Cross selection between members of the same dimension

The final touch is to query customers with specific characteristics are. In Figure 34 I initiated the Customer List action (that I previously defined as a drillthrough action that returns Customer attribute from dimension Customers) by right clicking on the intersection between column Golf and row Tennis. You can check in Table 4 that the result is correct.

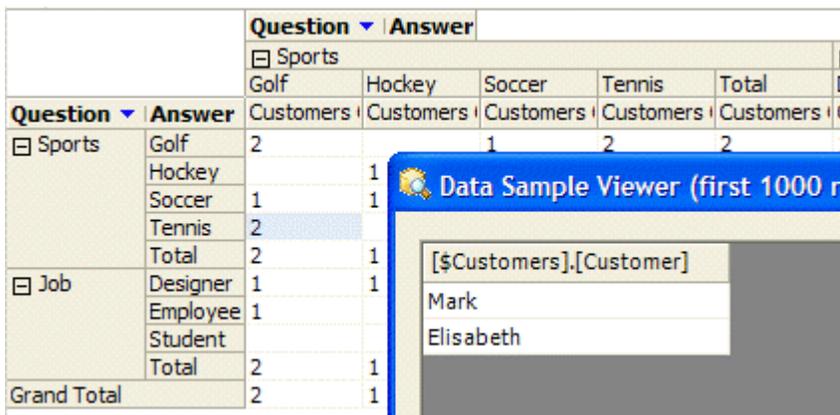


Figure 34 – Drillthrough Action on Golf-Tennis cell

It is possible to use the Survey model for many scenarios that present similar challenges. For example, we could apply the same technique to alarms and/or diagnostics generated on items (customers, cars). Another scenario is cross-sell opportunities There are data mining models to do that, but sometime a graphical output helps to visualize all of the relationships between specific items and the pivot table is the simpler way to obtain it.

Distinct Count

Distinct count measures are very useful and common requirement. Unfortunately, Analysis Services implementation is very resource-intensive. The algorithm used to calculate distinct count queries the source fact table data with an ORDER BY clause. For this reason, a separate measure group is required for each distinct count measure (SSAS generates a query for each partition/measure group). Not only this requires a long processing time and strains on the source RDBMS when the cube is fully processed (assuming no incremental update), but also results in relatively slow response times when the end user queries the distinct count measure.

Surprisingly, instead of using the UDM native distinct count support, we can build an alternative model based on many-to-many relationship that produces the same results but with faster processing times and equivalent or even faster response times.

The use of many-to-many relationships is particularly useful when you want to build a distinct count on a slowly changing dimension (SCD) dimension.

Business scenario

Marketing analysis often requires distinct count measures for customers and products sold. These measures are important to evaluate averages as sales for distinct customer, sales for distinct product, and so on.

For simplicity, we define a relational schema with only two dimensions: Date and Customers. To describe better the changing set of attributes related to it, the Customers dimension is implemented as a slowly changing dimension (SCD). The relational model is shown in Figure 35. For the sake of simplicity, my dimensions here have only the essential attributes; a real model would have many more attributes that would justify the presence of a Type II SCD for Customers (in the Kimball terminology a Type II SCD holds a row for each version of a member, holding all attributes history change).

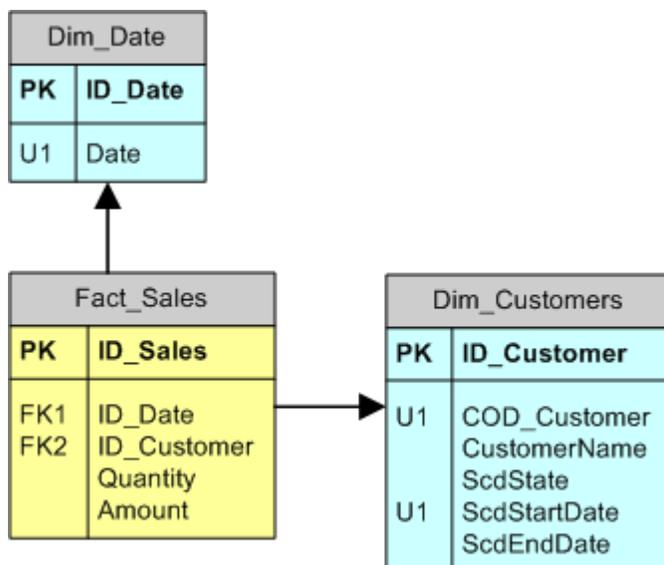


Figure 35 – Relational model with slowly changing dimension (SCD) Type II

We need a distinct count of customers applied to the COD_Customer attribute and not to the ID_Customer surrogate key. We will analyze several possible implementations that provide the desired results, considering both performance and impact on the relational and multidimensional models.

Implementation

I would like to introduce a simpler model than the one based on the Customers SCD, because it is important to understand how a many-to-many relationship works when we use it to obtain a value equivalent to a distinct count measure. To do that, we will consider the simpler relational model illustrated in Figure 36: Dim_Customers is a Type I SCD (in the Kimball terminology a Type I SCD does not preserve attribute history, holding only a row for each logical member and overriding old attributes with the new values on the member with the same application key)..

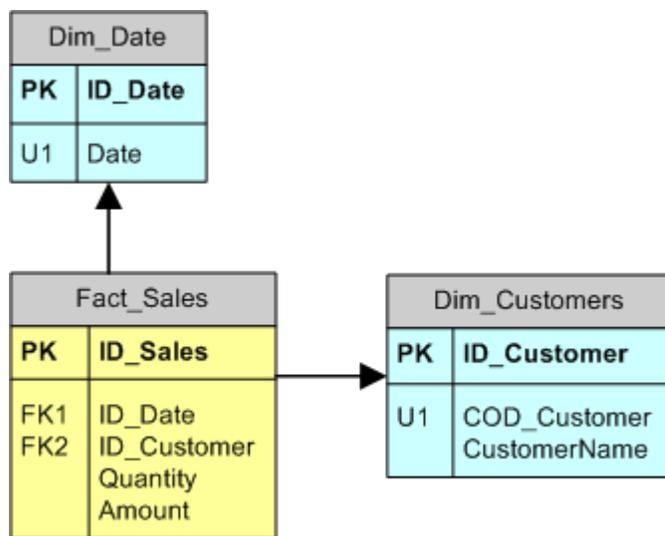


Figure 36 – Relational model without SCD (or SCD Type I)

We can easily build a cube with the two dimensions and standard measures (Sum of Quantity, Sum of Amount and Fact Sales Count). As you can see in Figure 37, I added a Year attribute to Date dimension (calculated as `YEAR(Date)`) and a Distinct Count of `ID_Customer` (it is named Customers Distinct Count into the Distinct Customers measure group).

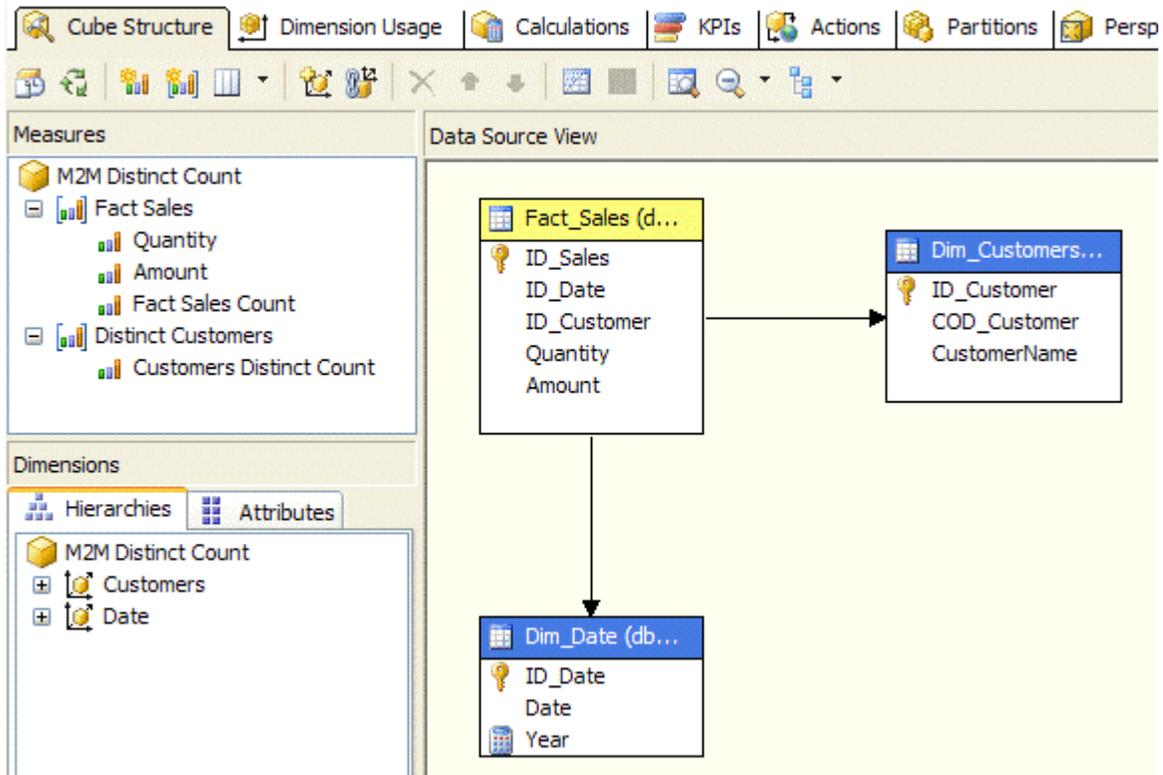


Figure 37 – Regular distinct count cube model

In Table 5 you can look at sample data I loaded into data mart. Note that Dim_Customers has nine customers, numbered from Customer 1 to Customer 9.

Table 5 – Fact_Sales sample data

Date	Customer	Quantity	Amount
01/01/2006	Customer 5	20	495.67
01/01/2006	Customer 5	3	6458.27
01/01/2006	Customer 6	7	7330.54
02/01/2006	Customer 3	28	2201.90
02/01/2006	Customer 5	25	911.05
06/01/2006	Customer 9	5	6342.61
07/01/2006	Customer 6	20	5437.42
10/01/2006	Customer 1	1	1084.56
10/01/2006	Customer 6	2	1000.29
10/01/2006	Customer 9	20	9319.23

Figure 38 shows you the pivot table results. We have only 5 distinct customers who made 10 sale transactions. The numbers are also shown at the day level (lowest grain) of the date dimension.

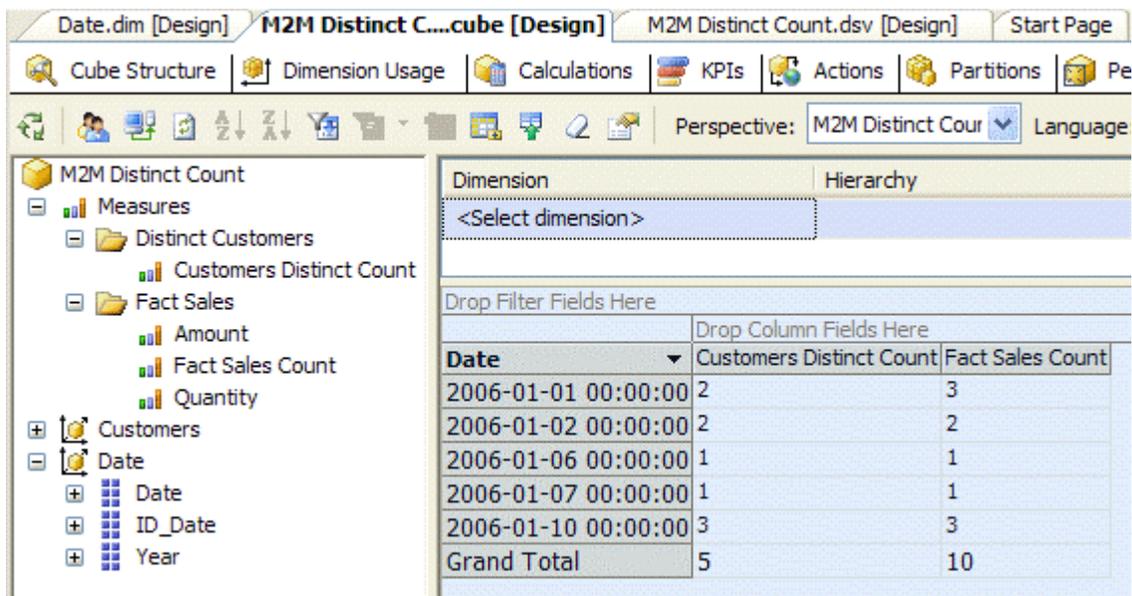


Figure 38 – Regular distinct count results

Now we can add a measure that counts the number of rows in Dim_Customers and compare the results. We configure the New Measure dialog box as shown in Figure 39.

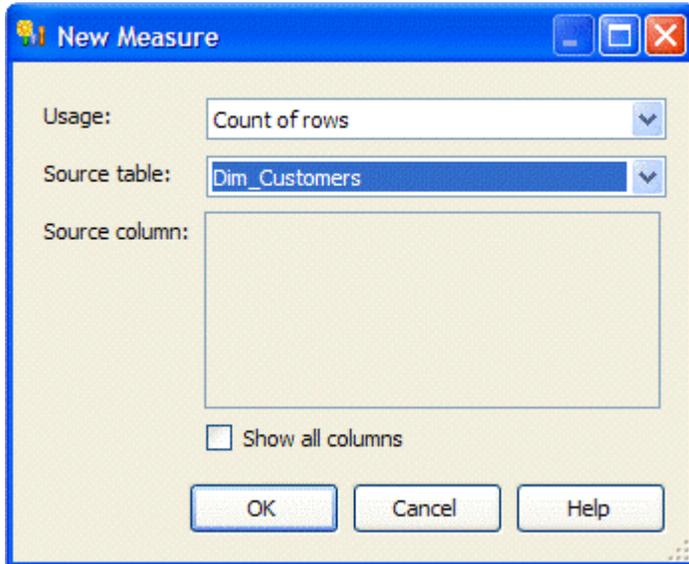


Figure 39 – New Measure based on Count of Rows of Dim_Customers

Figure 40 shows the updated cube structure after renaming the measure.

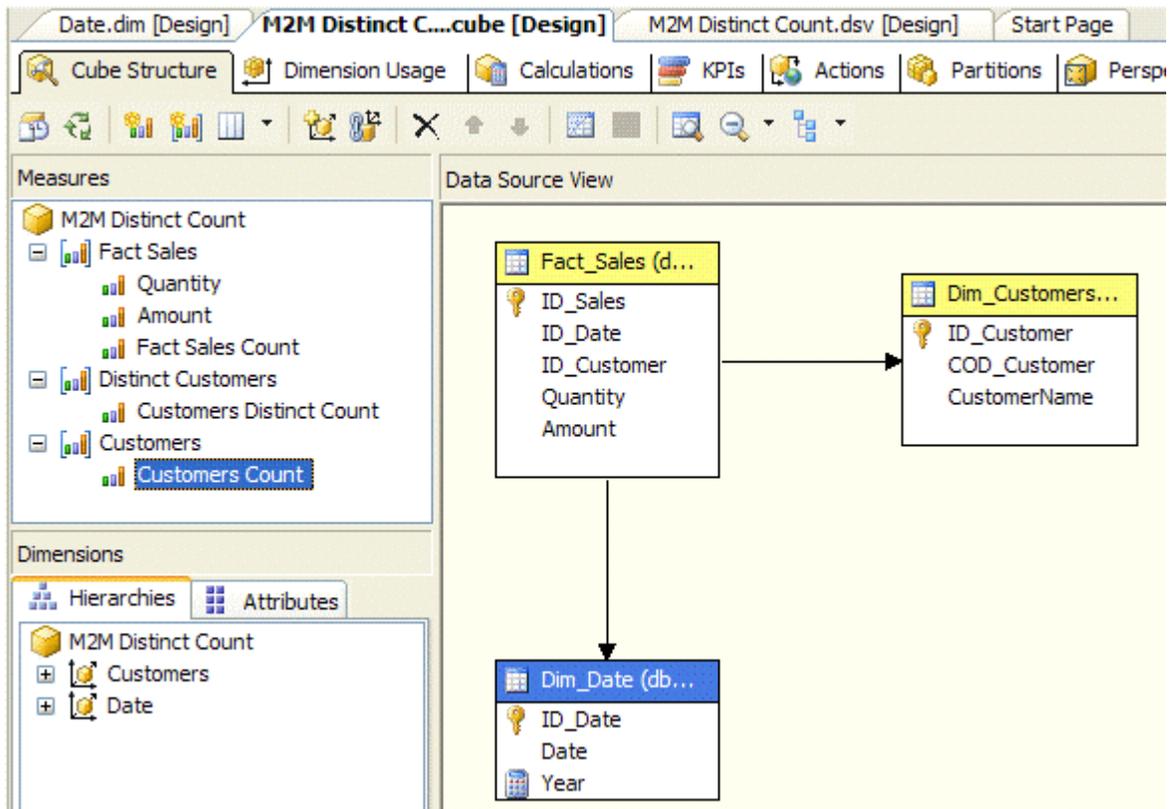


Figure 40 – Customers Count added to cube model

At this point, we need to define a relationship between the Customers measure group and the cube dimensions: if we did not, the report would show the total row count of all

rows in Dim_Customers for any query we will do. To avoid this, we use the Dimension Usage dialog to set up a many-to-many relationship with the Date dimension using Fact Sales as intermediate measure group (see Figure 41).

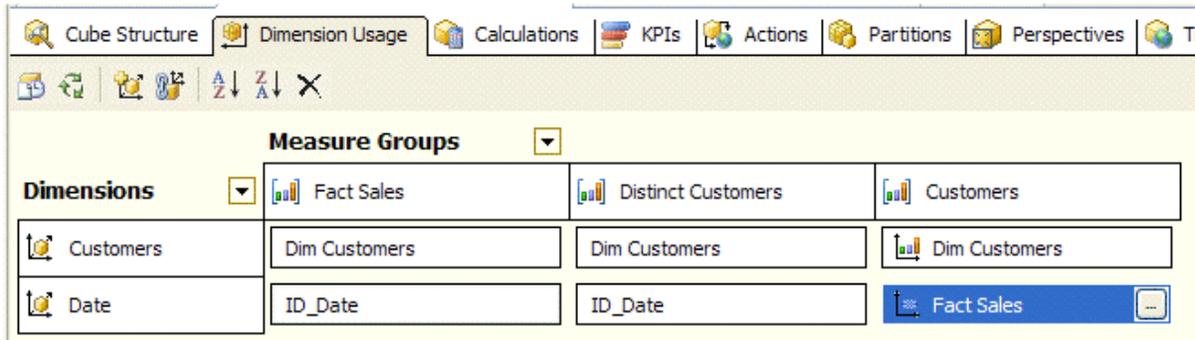


Figure 41 – Many-to-Many relationship between Customers and Date

Now we can compare the Customer Count produced by the many-to-many relationship with the Customers Distinct Count obtained with the regular Distinct Count aggregate function. As Figure 42 shows, the numbers are the same regardless of the selected date, but the Grand Total are different. The reason is that in absence of a Date selection there is no need to apply a filter on Customers based on the many-to-many relationship with the Date dimension; therefore, we have a value of 5 for Customers Distinct Count and 9 for Customers Count.

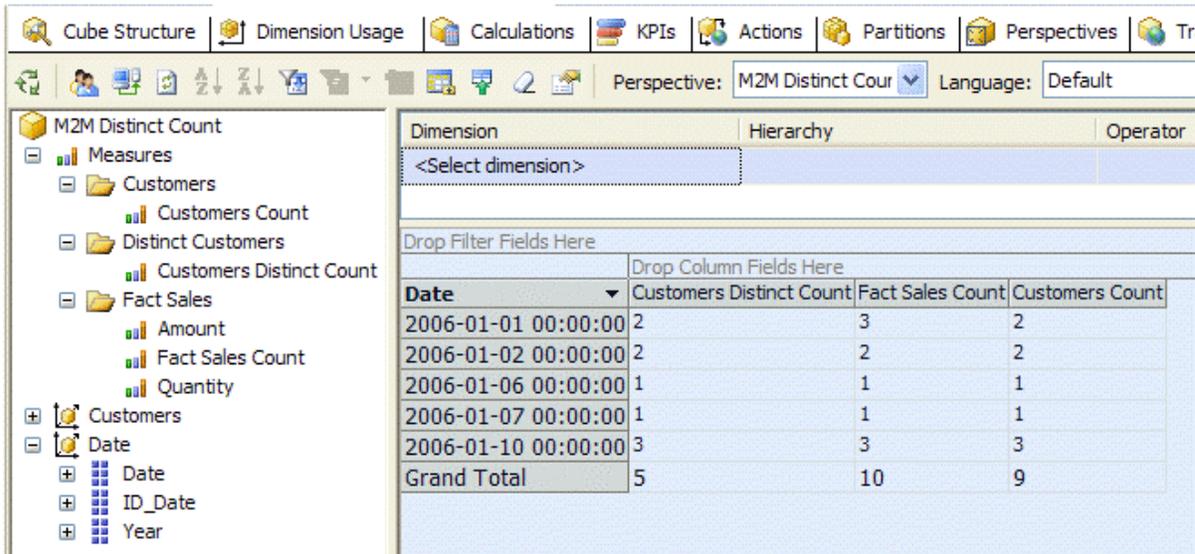


Figure 42 – Customers Count compared to Customers Distinct Count

You may think that the Customer Count column is useless because it is not consistent with the Customers Distinct Count measure. However, in many cases, a query contains a selection on an involved dimension. If we use the Year attribute instead of the Date attribute, we see the interesting data in Figure 43.

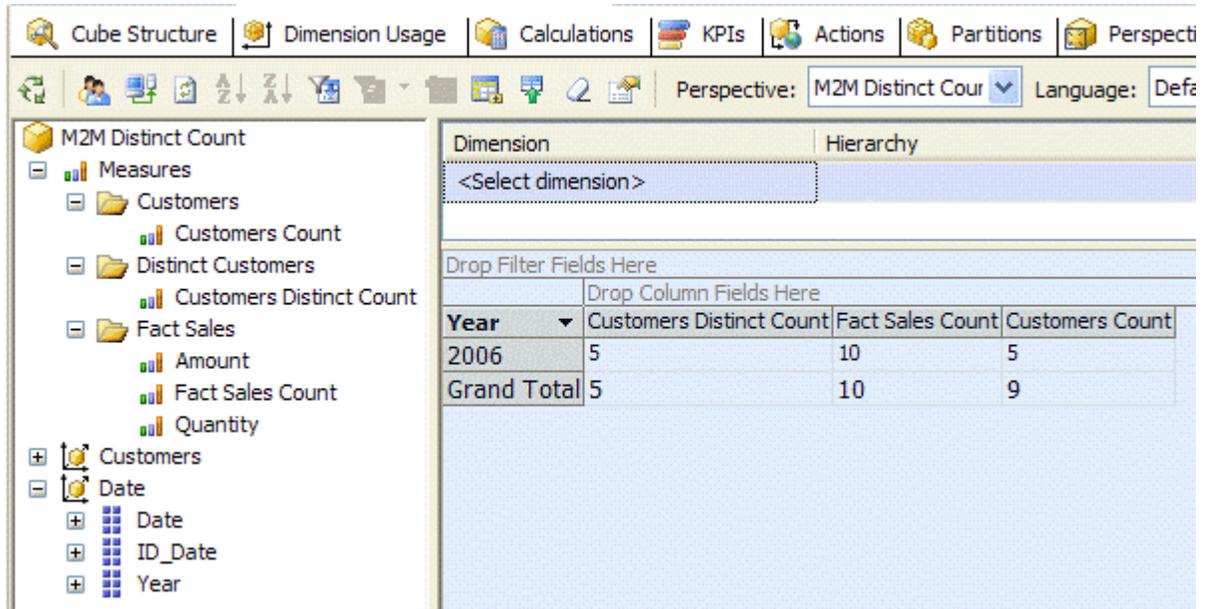


Figure 43 – Use of Year attribute instead of Date attribute

The Year 2006 is exactly what we are interested in. If you consider that usually need to count the customer only if she did at least one sale transaction overall (we assume that a customer is not a prospect), then it should be reasonable to expect that the Customers Count measure is practically the same as the Customers Distinct Count measure.

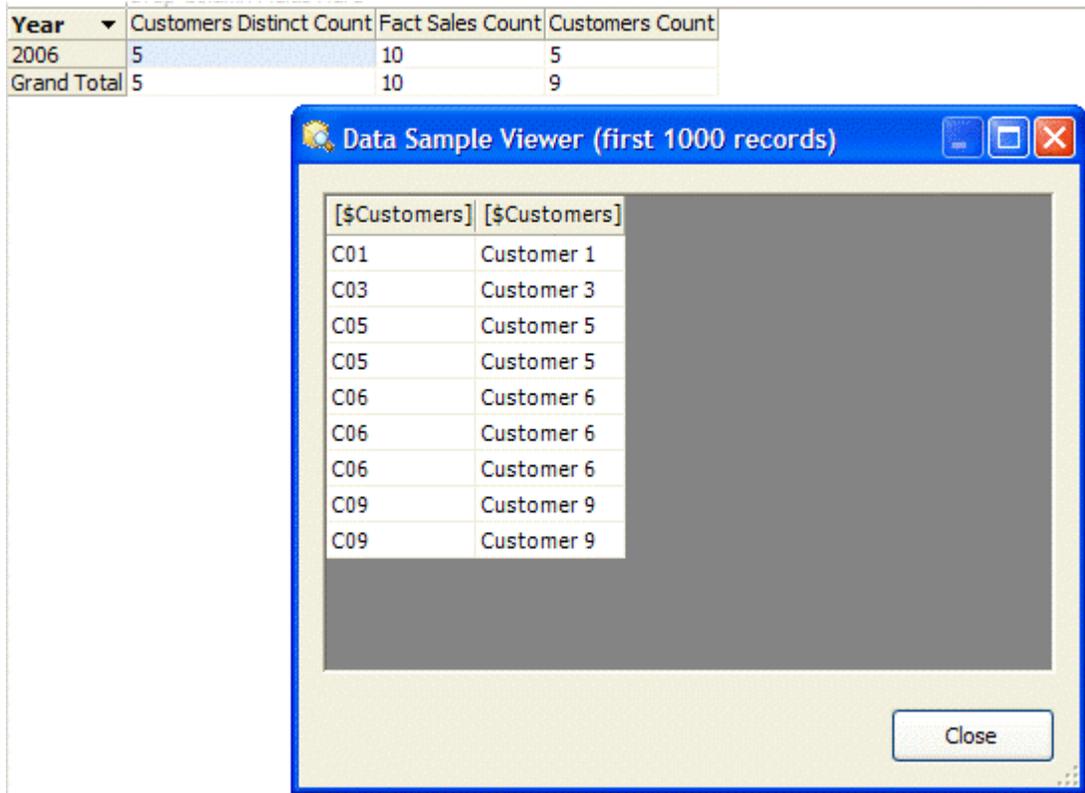
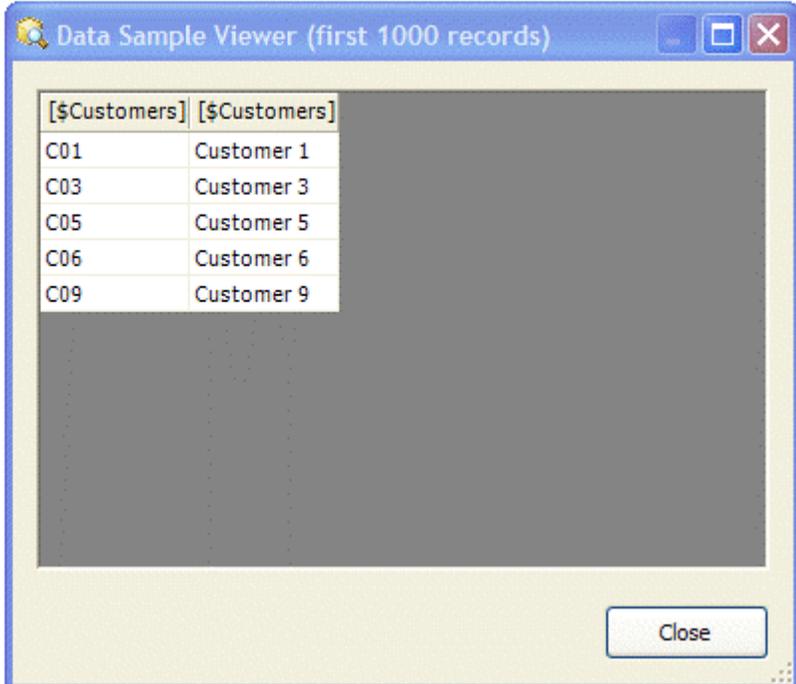


Figure 44 – Customers drillthrough for standard distinct count measure

Some modelers may favor the use of many-to-many relationships to define a distinct count measure just for a simple feature you obtain as a side effect. If you define a drillthrough action (named Customers in our case) to get the list of customers behind a given cell, you get the results shown in Figure 44 after drilling through the Customers Distinct Count measure for 2006. In comparison, Figure 45 shows the drillthrough results for the Customers Count measure for 2006. Here, we obtain the list of distinct customers while this is not the case with the Customers Distinct Count measure (see Figure 44 again). If you use a distinct count measure, consider a distinct filter on the drillthrough results to eliminate duplicated customers. This is not necessary with a many-to-many relationship.

Year	Customers Distinct Count	Fact Sales Count	Customers Count
2006	5	10	5
Grand Total	5	10	9



[\$Customers]	[\$Customers]
C01	Customer 1
C03	Customer 3
C05	Customer 5
C06	Customer 6
C09	Customer 9

Figure 45 – Customers drillthrough for Customers Count (obtained by many-to-many relationship)

We are ready to introduce the slowly changing dimension in this scenario. When evaluating the distinct count of customers in a Type II SCD who have made a transaction, we cannot rely on the distinct count of the customer surrogate key in the fact dimension.

We have several choices:

- A. Create a unique customer dimension: this means duplicating the customer dimension, at least for the most recent version of each member
- B. Create a Distinct Count measure on the application key of customer dimension: the measure is defined into a measure group that has similar relationship to the one we just used to evaluate the customer count measure through a many-to-many relationship

C. Define a solution that is similar to solution B, substituting the distinct count measure with another many-to-many relationship derived count measure.

Each one of these solutions has its positive and negative aspects. To test all of these cases, we need to modify our data. Table 6 shows that Customer 6 has two versions (it was changed on 05/01/2006). For this reason, we have still 9 customers but 10 different rows in Dim_Customers, and we have 5 different customers who made transactions but 6 different customer surrogate keys referenced in the fact table.

Table 6 – Fact_Sales SCD sample data

Date	Customer	Quantity	Amount
01/01/2006	Customer 5	20	495.67
01/01/2006	Customer 5	3	6458.27
01/01/2006	Customer 6 v1	7	7330.54
02/01/2006	Customer 3	28	2201.90
02/01/2006	Customer 5	25	911.05
06/01/2006	Customer 9	5	6342.61
07/01/2006	Customer 6 v2	20	5437.42
10/01/2006	Customer 1	1	1084.56
10/01/2006	Customer 6 v2	2	1000.29
10/01/2006	Customer 9	20	9319.23

In Figure 46 shows the new Data Source View. It uses additional views that simulate what we could have achieved by modifying the relational schema of our Data Mart. When to use views against materialized tables is another topic by itself, which has to be evaluated considering processing time, number of distinct count measures and complexity of existing ETL processes (they should be modified if data mart schema would be changed). The view `vFact_Sales_Unique` adds the `COD_Customer` at the fact table level, which is necessary to implement case A. The case B does not need any new elements. To implement case C we have to add two views: `vDim_CustomersUnique` simulates a customer dimension containing only a unique row for customers (without changing attributes), `vCustomersScd` simulates a bridge table that joins each unique customer member (`vDim_CustomersUnique`) with its versions (`Dim_Customers`).

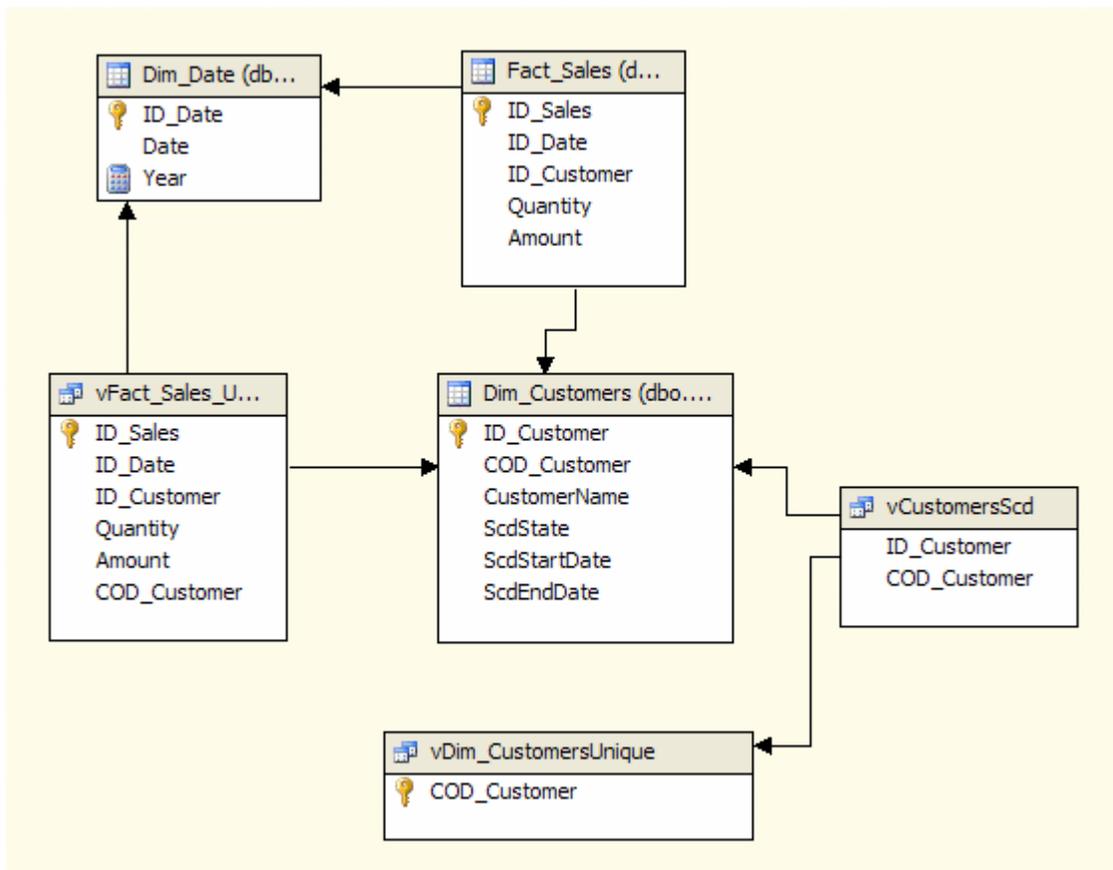


Figure 46 – Data Source View to implement different distinct count strategies

In the simplest scenario (case A), we create a unique customer id at the fact table level and then define a distinct count measure on it. Figure 47 shows that we could have used `vFact_Sales_Unique` view to build both Fact Sales measure group measures and A Count measure on A Customers measure group. However, there is no benefit doing so, because a distinct count measure needs a dedicated measure group (A Customers) that is processed with a separated query to the fact table. In this case, we want to limit the join between `Fact_Sales` and `Dim_Customers` only for the `COD_Customer` distinct count evaluation. From this point of view, I could eliminate the other measures (`Quantity` and `Amount`) from `vFact_Sales_Unique`. This is only an aesthetic touch without improvements on the performance side, but it makes a lot of sense from the

maintenance viewpoint and in order to not confuse people with two copies of the same table.

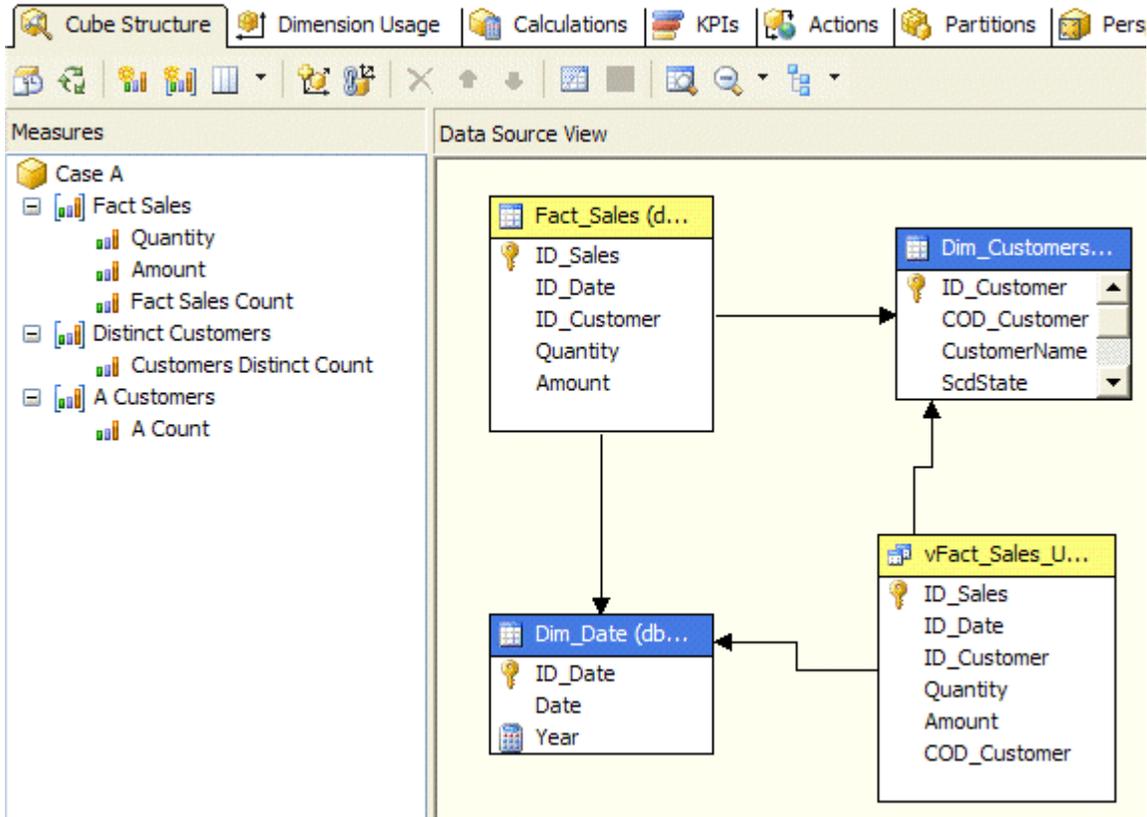


Figure 47 – Case A with standard distinct count measure on fact table

Once we created the A Customers measure group, we need to relate it to cube dimensions, as shown in Figure 48. Relationships are very simple and equals to those of other measure groups.

Measure Groups			
Dimensions	Fact Sales	Distinct Customers	A Customers
Customers	Dim Customers	Dim Customers	Dim Customers
Date	ID_Date	ID_Date	ID_Date

Figure 48 – Case A Dimension Usage

We can look at results obtained with A Count measure (Figure 49). The Customers Distinct Count measure is 6 for 2006 because it counts the number of rows in Dim_Customers; we have two versions for Customer 6 so it is counted twice here. The new A Count measure has the right number of 5 and it is the number we want to see.

Drop Filter Fields Here		Drop Column Fields Here			
Year ▼	Quantity	Amount	Fact Sales Count	Customers Distinct Count	A Count
2006	131	40581.54	10	6	5
Grand Total	131	40581.54	10	6	5

Figure 49 – Case A results

Although we have solved the business problem, we could face some performance issues, which I will discuss further in the Performance section. However, it is necessary to state something here:

- A Distinct Count measure is obtained through an ORDER BY query that uses the measure expression as the key to sort.
- The application key we are using to evaluate the distinct count could be a long string. It has to be copied into the cube, even if it is not interesting to end user.
- We used a view to avoid duplicating customer dimension data in a Customers Unique dimension, but this view contains a join and an ORDER BY clause. This could be very heavy on large fact tables and large dimensions.
- Distinct count measures on Analysis Services 2005 are not very performant or scalable.

Case B still uses a distinct measure, but this time we do not use a view. Instead, we rely on the UDM fact dimension feature. Figure 50 shows that the B Count measure is based on a distinct count of the COD_Customer field in Dim_Customers table (that is used both as a dimension and as a fact table).

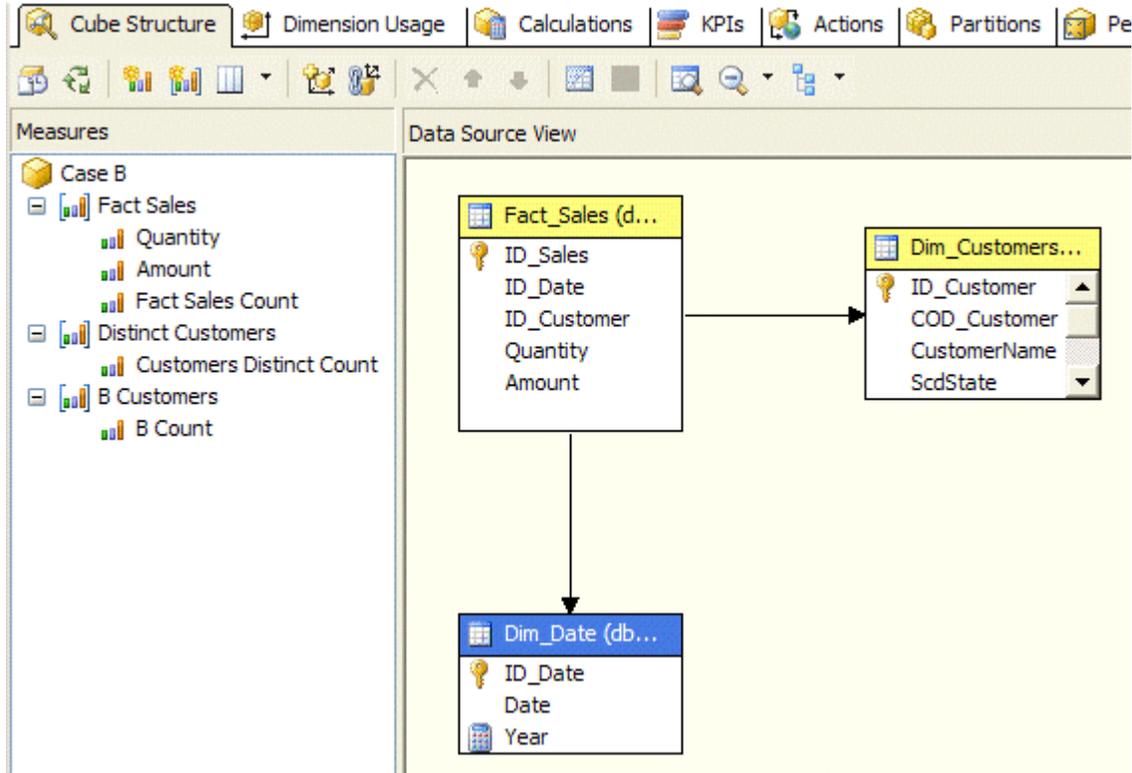


Figure 50 – Case B with distinct count measure on customer dimension

The B Customers measure group has a direct relationship with the Customers dimension (the relationship type is “Fact”) and a many-to-many relationship with Date dimension via the Fact Sales measure group. Apparently, this is a strange relationship because a row in Dim_Customers as fact table has a one-to-one relationship with Dim_Customers as Customers dimension (it is the same table!). However, the reality is that each customer can be related to many dates and each date can be related to many customers, and Fact_Sales defines exactly this relationship. Figure 51 shows the resulting Dimension Usage.

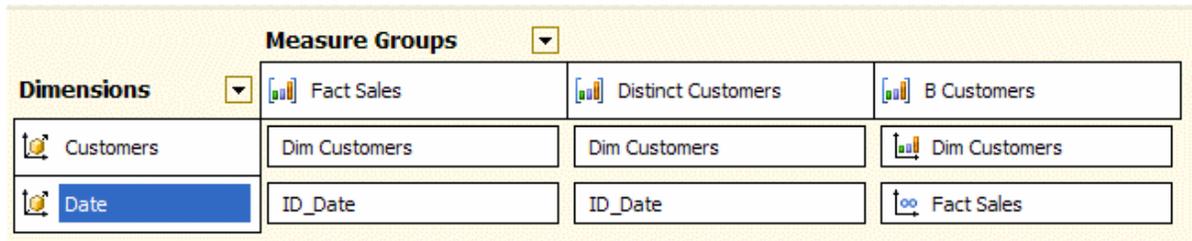


Figure 51 – Case B Dimension Usage

At the end, we have similar results to the case A: Figure 52 shows the B count results. The only difference is that when there is no filter on Date dimension (the Grand Total row) the B count shows the overall number of unique customers instead of considering

only the customers who made a transaction. We already discussed it with the previous scenario when we did not have a slowly changing dimension for Customers.

Drop Filter Fields Here		Drop Column Fields Here			
Year ▼	Quantity	Amount	Fact Sales Count	Customers Distinct Count	B Count
2006	131	40581.54	10	6	5
Grand Total	131	40581.54	10	6	9

Figure 52 – Case B results

What is the biggest difference between case A and case B? In case A, we had to build a view (or a persisted dimension table) to link the unique customers dimension to the Fact_Sales table. In case B, this is not needed. The processing query is made only against the cardinality of Dim_Customers table and not against the cardinality of the more populated Fact_Sales table. This may result in significantly better performance sometimes.

In case C, we apply the lesson we learned at the beginning of this chapter, when we used a many-to-many relationship to get the same results of a distinct count measure. In this way, we will remove the need for a distinct count measure and related implications.

Figure 53 shows that the model becomes relatively more complex. We need to build a fact dimension (vDim_CustomersUnique) where the number of rows equals the number of unique Customers we have. Unfortunately, we cannot extend the model we previously defined for case B because the fact dimension we used (Dim_Customers cannot serve as an intermediate measure group in a many-to-many relationship. For this reason, I created a view (vCustomersScd) that serves as a factless fact table between Dim_Customers and vDim_CustomersUnique.

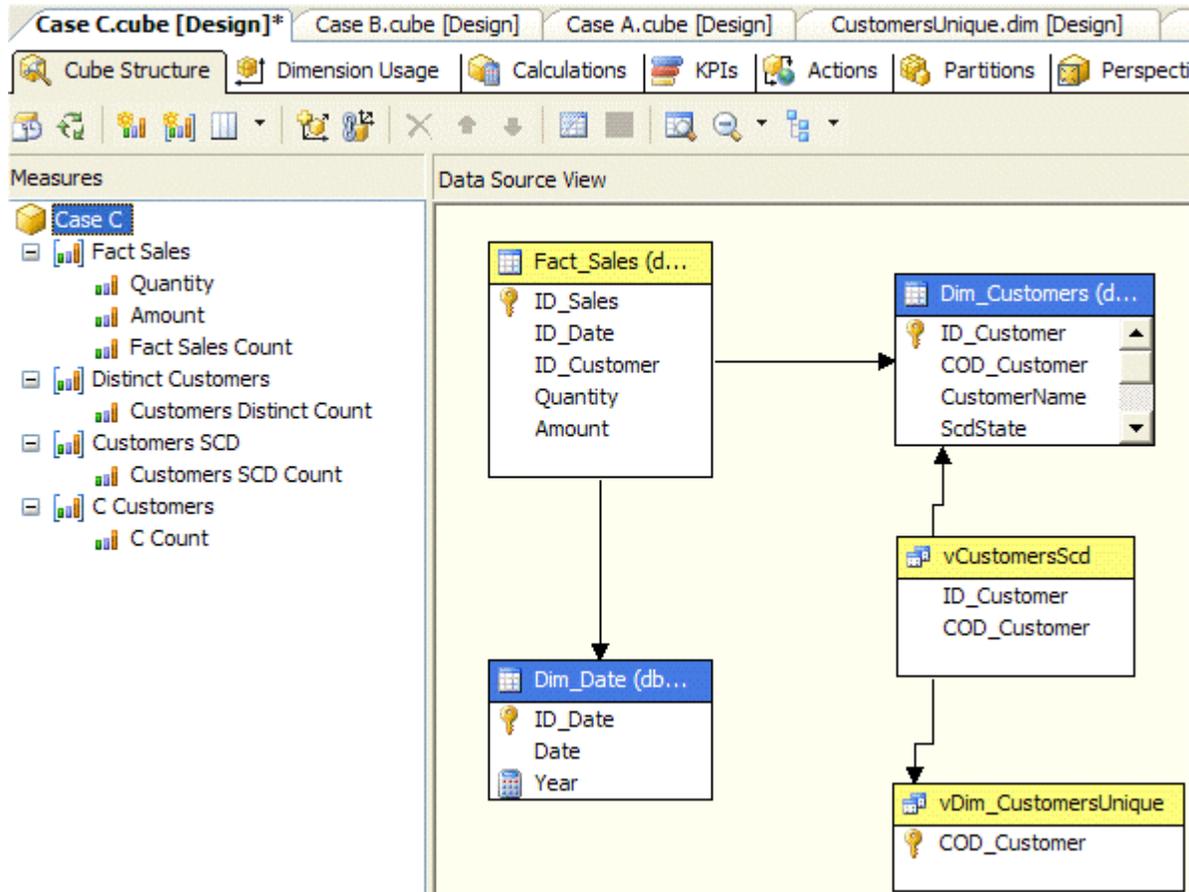


Figure 53 – Case C with distinct count measure by many-to-many relationship

The Customers SCD measure group has a row for each row in Dim_Customers, with a one-to-one relationship. The C Customers measure group has a row for each unique customer. To define a relationship between these two measure groups, it is necessary to have a dimension shared by both measure groups. This role is fulfilled by the CustomersUnique dimension, which has the same cardinality as C Customers. While we can identify a one-to-many relationship between Customers SCD and C Customers measure group, a better approach is to leverage with the UDM many-to-many relationship.

The Customers SCD measure group plays a very important role linking the C Customers measure group with all the other measure groups of a cube. Figure 54 shows the Dimension Usage setup required to implement case C.

Dimensions	Fact Sales	Distinct Customers	Customers SCD	C Customers
Customers	Dim Customers	Dim Customers	Dim Customers	Customers SCD
Date	ID_Date	ID_Date	Fact Sales	Customers SCD
CustomersUnique	Customers SCD	Customers SCD	Customer Code	Customer Code

Figure 54 – Case C Dimension Usage

The CustomersUnique dimension plays an important role to define the correct relationship between measure groups, but its content may not be useful for end user reporting. For this reason, often I usually hide this dimension to end users. Another interesting aspect is that the CustomersUnique dimension has the customer application key as a primary key of the dimension. If the application key (COD_Customer, in this case) is very long, it could become a potential performance bottleneck and it will consume more space for data storage. In real life, I have used a persistent dimension table instead of a view, just to get a surrogate key (of type int) instead of the large application key (more than 20 characters) we get from the operational data store.

Figure 55 shows the results.

Drop Filter Fields Here		Drop Column Fields Here				
Year	Quantity	Amount	Fact Sales Count	Customers Distinct Count	Customers SCD Count	C Count
2006	131	40581.54	10	6	6	5
Grand Total	131	40581.54	10	6	10	9

Figure 55 – Case C results

While the Customer SCD Count measure is not useful, the C Count behaves exactly as the B Count measure.

As I said before, you should consider implementing the distinct count measures with many-to-many relationships to gain performance improvements.

Performance

There are two main observable differences when we query a cube that has distinct count measures obtained with different methods: one is the capability to take advantage of multiple processors, the other is caching the query results. To understand performance impact, we have to understand how Analysis Services solves queries for these kinds of measures.

A “classical” distinct count measure works in this way:

- At processing time, an ORDER BY clause is added to the query sent to the relational engine for each cube partition to order data by the distinct count measure expression. In the best-case scenario, you do not have joins between the fact table and other tables, but when you have millions of rows the ORDER

BY clause could be very slow and it may require many resources (memory and disk for temporary table). Note that this affects the performance of the relational engine. The processing time of distinct count measures could be very long but can be improved using incremental updates rather than processing the entire cube (FullProcess option).

- This explains why the Distinct Count measure needs a separate measure group in UDM.
- My hypothesis is that a dedicated index is generated for a distinct count measure and the correct order of items is necessary to use a memory-efficient algorithm.
- At query time, a sequential scan of the distinct count partition is made for each query involving a distinct count measure. Query response time is related to both the number of rows of the fact table and the number of different distinct values of the measure. For some reason, the query cannot be entirely cached and a subsequent query containing a distinct count measure requires more or less the same time (probably the time improvement is only given by eliminating disk I/O with all necessary data already in server memory). Even a full measure group optimization (building 100% of possible aggregation) does not improve significantly this type of queries.

A measure involving a many-to-many relationship works as follows:

- At processing time, the factless fact table used by the many-to-many relationship is read in no particular order (like a regular fact table). There is no particular pressure on the relational engine even with millions of rows. Nevertheless, as a many-to-many relationship relates members of two different dimensions, it should be rare to have more than 10 million rows to process.
- At query time, the fact table is read in memory to evaluate the many-to-many relationship through the bridge measure group. This is done mainly to join the two measure groups at query time and the join has to be done on the lowest level of each dimension (common to both measure groups).

The engine does a hash join for this purpose (unlike SQL Server query engine, Analysis Services does not have multiple join algorithms to choose from). The hash join does a lookup on the factless measure group, builds a hash index on it, scans the fact measure group and combines the two results together. As you can imagine this operation requires enough virtual memory to load and evaluate the datasets. A fact table with 100 million rows can exhaust the 2Gb addressable memory space in a 32-bit system (I strongly suggest the use of a 64-bit system for Analysis Services). If the memory is sufficient, the first query may be very slow, while in a 2 GB user memory address space a fact table with 100 million rows could fail the query exhausting the address space of the Analysis Services process. It could be very slow the first time, but subsequent queries are very fast (immediate response) because Analysis Services caches well previous results.

Unfortunately, these two techniques to calculate a distinct count measure (the "classic" one and the one based on many-to-many dimension relationships) have both some shortcomings. If you could warm up the cache after cube processing (for example by executing a scheduled MDX query), users would probably favor the performance of a

distinct count measure based on many-to-many relationships. That is because each time the end user changes a selection or a filter with the “classic” model, the user will experience performance degradation. Consequently, interactive reports typically run faster with the many-to-many relationship technique. The performance degradation associated with the “classic” distinct count model is a less of an issue with static reports, especially with Reporting Services cached reports.

The real problem with using many-to-many relationship is the limit of fact table rows you can query. You should evaluate carefully the use of many-to-many relationships when you have measure groups getting data from fact table with more than 10 million of rows.

Multiple Groups

Sometime it is not easy to describe attributes related to a dimension member. In a typical cube dimension, attributes are defined at the data warehouse designing stage and adding an attribute is an operation that necessitate changes in all layers of the BI solution. While a rigid design is good for performance optimization, this is a limitation for end users like marketing analysts, who try to jump over these limits by extracting data from data warehouse and working with them offline, making custom groups of dimension elements based on some characteristics which was not known until a few days before.

There are many examples of this situation, but we can generalize it by assuming that a user may want to group some dimension members together, associating them to a group name. Moreover, a single dimension member may belong to more than one group: it can belong to N groups, where N is not defined beforehand.

The "Multiple Groups" model I am going to introduce has an interesting characteristic. It is based on a fixed relational and multidimensional schema, while loaded data may define new groups that are immediately available to all clients. Moreover, a new group can be added by only reprocessing a possibly small measure group (corresponding to the factless fact table for a many-to-many relationship), giving the opportunity to create custom solutions that enables a user to create custom groups on the fly, thus getting almost immediate results.

Business scenario

Typically, sales analysis involves the creation of custom groups of customer and product dimension members. These groups can be based on events (who has been included in a mail campaign), on profiling analysis (that could be the result of a manual segmentation or the end result of a data mining clustering model) or on other arbitrary data. I will discuss next a customer analysis scenario, but an equivalent model could be used for products analysis.

The classical approach for custom grouping is to define a table for each type of group, with a field for each group attribute and a field for customer key: that table will contain a row for each customer that belongs to each group. For example, For example, imagine that you need to segment customers in a certain way and track customers who received mailing offers for our products: Figure 56 shows a canonical solution that uses a separate table for each kind of group.

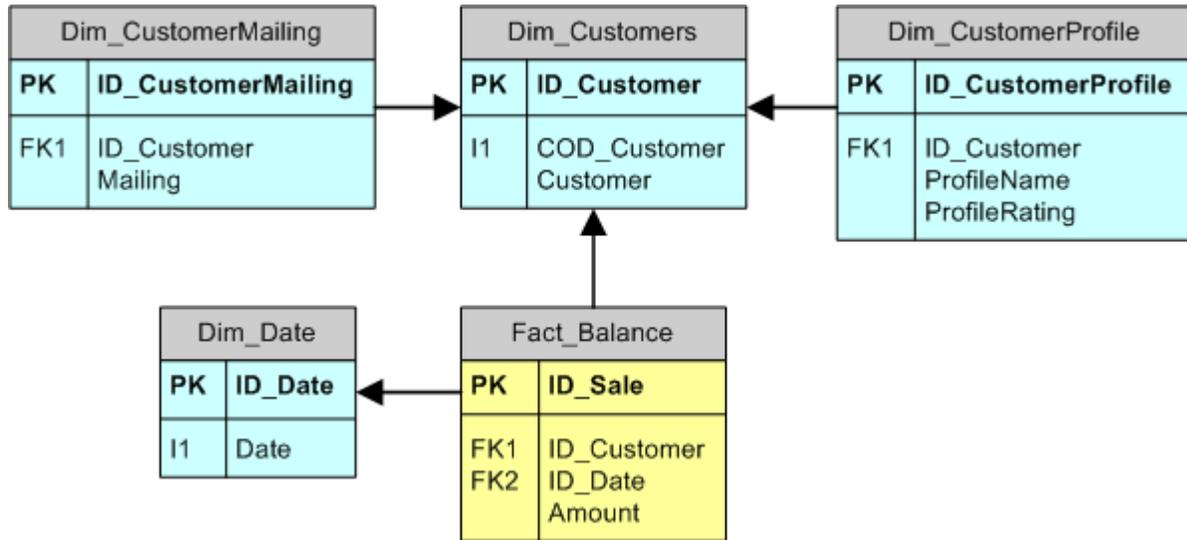


Figure 56 – Multiple grouping made with a table for each kind of group

We could implement a corresponding UDM with Customer dimension related to CustomerProfile and CustomerMailing dimensions with two different many-to-many relationships. The key point here is that if a customer could belong to more than one group, you have to go for many-to-many relationships. At this point, a more normalized and UDM-friendly way to handle this scenario is shown in Figure 57.

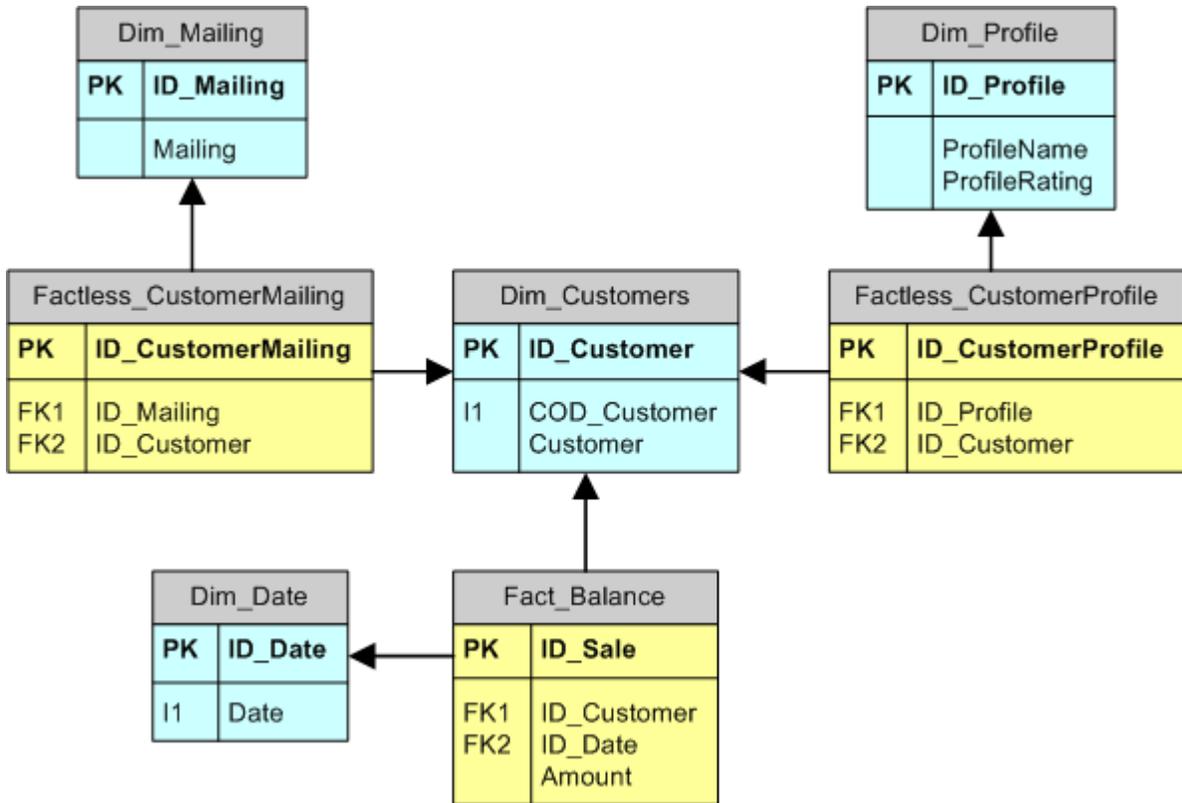


Figure 57 – Multiple grouping with explicit many-to-many relationships

This design may not give us enough flexibility. If a new group requires a new table in the data warehouse, it will also require changes to the ETL processes and UDM. However, generally speaking, this model allows us to use a single “group” dimension table for any kind of grouping.

Implementation

A potential weakness of the model (see Figure 57) is related to the customer-profiling requirement: in the real world, we may have many profiles, but for each profile a customer can have only one rating (or no rating at all). Unfortunately, we cannot implement this requirement with a constraint in the relational database. One way to implement this level of control on model shown in Figure 56 would be a unique index on the ProfileName and ID_Customer fields. Anyway, data integrity is out of our scope. After all, a data mart has to be loaded with correct data and we will delegate this check responsibility to the ETL pipeline. This note will be important in our final considerations for this scenario.

If we consider the whole scenario, we can identify these requirements: a customer can belong to many groups, a group can have many customers and a group can have a characteristic name and a “value” textual attribute (see Figure 58). Please note that

COD_GroupName and COD_GroupValue fields are application keys of grouping structure.

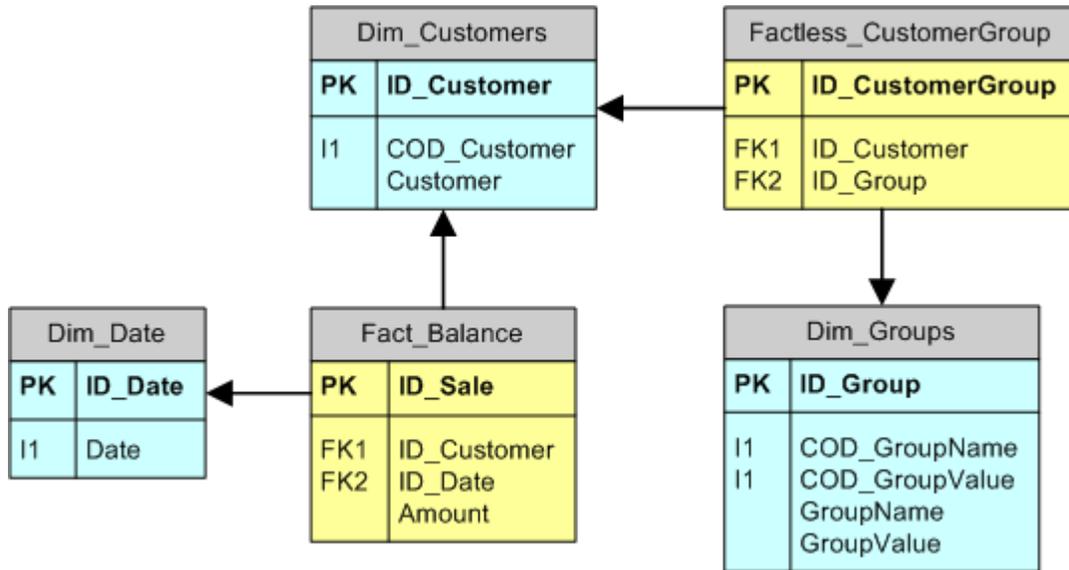


Figure 58 – Multiple grouping with a generic flexible model

Sometimes we can use the group name as a sort of group category and the group value as the real group name. Other times, we can use the group value for segmenting the group population. Table 7 shows both variants: the Mailing group name identifies a category of mailings and a customer could belong to any (even all) of the possible groups defined by Group Value (in this case, Promo Spring and Promo Fall are two mailings we have made to two different and partially overlapping groups of customers). The Profile group name identifies a single group where each customer must belong to only one of the possible group values: Retail, Affluent, Private or Corporate.

Table 7 – Groups dimension sample data

Group Name	Group Value
Mailing	Promo Spring
Mailing	Promo Fall
Profile	Retail
Profile	Affluent
Profile	Private
Profile	Corporate

The interesting part is that adding a new group does not require a structural operation. For example, a new Promo Winter mailing needs only a new record in Dim_Groups table and a correct population of the Factless_CustomerGroup table: given a new ID_Group, it is only necessary to get a list of ID_Customer to do this population. We can imagine a simple web application to do this. It may take the group name, group value, and a list of customer application keys (eventually uploadable as an excel file) to incrementally update Factless_CustomerGroup and Dim_Groups tables. There is plenty of room to get

creative here, but the point is that we have come up with a way to integrate user-defined groups in UDM with a minimum development effort.

We can create the cube with the auto build feature of the Cube Wizard. The resulting model would correctly identify dimension and fact tables but, as have seen before, we have to define manually some of the missing relationships between dimensions and measure groups.

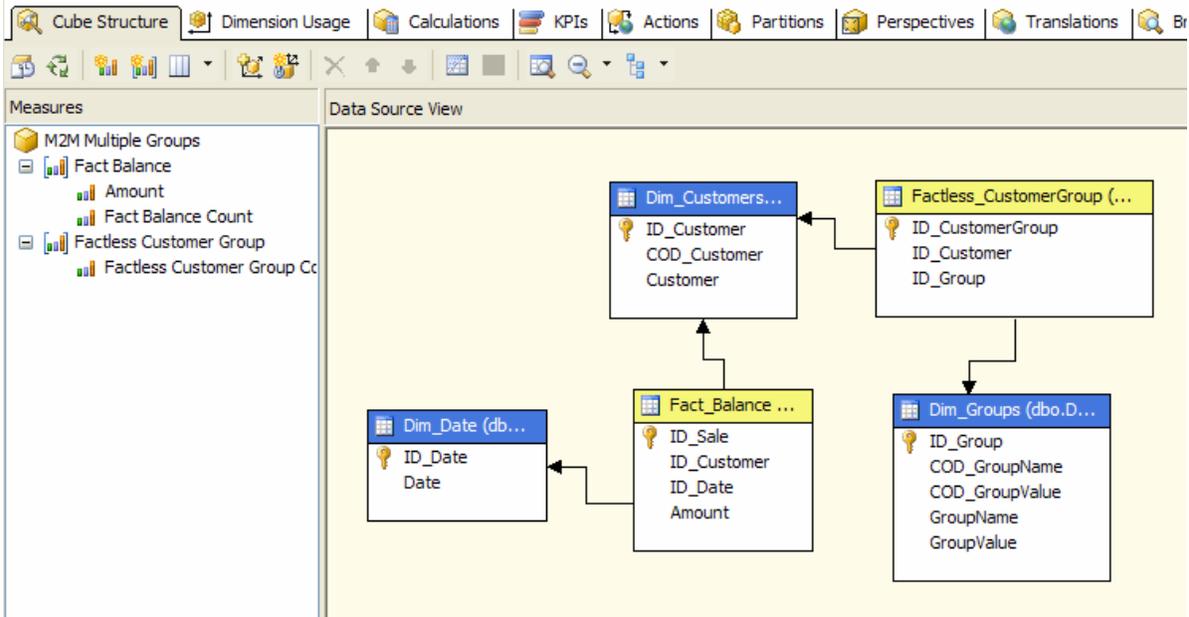


Figure 59 – Cube structure for multiple grouping

As we can see in Figure 59, we have two measure groups for a total of three measures (I decided to keep the original wizard-generated names). Fact Balance Count is the number of rows for Fact Balance table. Factless Customer Group Count is the number of customer for selected group(s): from another point of view, it is also the number of groups that a customer belongs to (it may not be that useful). If users do not need to analyze a group population, you can hide Factless Customer Group Count measure to them, otherwise renaming it to a more meaningful name would be a good idea.

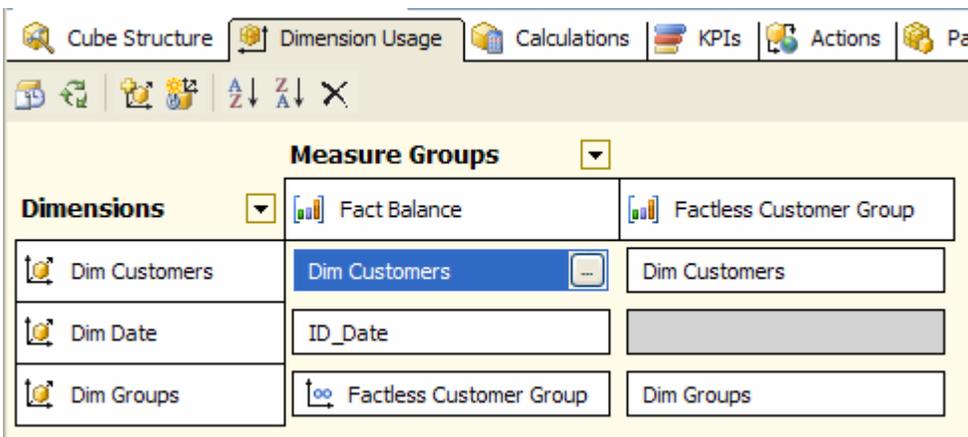


Figure 60 – Cube wizard dimension usage results for multiple grouping

Figure 60 shows that in this case only the Date dimension has to be related to Factless Customer Group. We can fill the gray cell with another many-to-many relationship (the first one was created by the wizard), as shown in Figure 61.

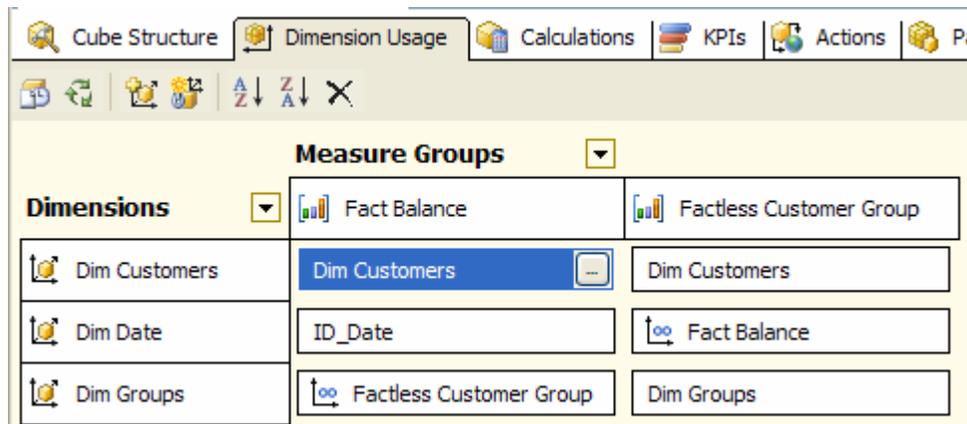


Figure 61 – Completed dimension usage for multiple grouping

Figure 62 shows a sample report using this model. The filter is set on a specific date (remember we have balance as measure which cannot be summed over time), while the Group Name and Group Value dimensions are placed on the rows. The two mailing groups (Promo Fall and Promo Spring) partially contain same members. In fact, Total row for Mailing group name is less than the sum of each single group value row. We have a different situation for Profile group name. A customer should belong to only one of the possible child group values, which is the case with our sample data.

Date		Drop Column Fields Here		
2005-12-31 00:00:00		Amount	Fact Balance Count	Factless Customer Group Count
Mailing	Promo Fall	300	3	3
	Promo Spring	200	2	2
	Total	400	4	5
Profile	Corporate	200	2	2
	Private	200	2	2
	Total	400	4	4
Grand Total		400	4	9

Figure 62 – Sample query for multiple groups

Having analyzed Balance Amount measure, we can apply the same considerations for Fact Balance Count measure. Typically, it is used as denominator to get an average amount balance instead of the total balance (Sum aggregation). It is important to note that Fact Balance Count could be lower than Factless Customer Group Count even for a single Group Value row. This happens when at least one customer associated with the group has no registered balances for the date chosen.

I want to make one last consideration about Factless Customer Group Count measure. It is aggregated as a regular measure and it is not to be confused with the number of *different* customers belonging to a group. This is particularly important when you are considering the total for a Group Name, grouping all his Group Value children: it is another good reason to hide this measure to end users.

While it could be possible to add other attributes to the Groups dimension, you have to be very careful doing so. If you want a generic way to group items of a dimension, it is important to leave the group dimension design as generic as possible. Adding an attribute that is used only with some specific groups would be a bad way to make things easy to use and to read.

A final consideration is about the overall performance. From a query standpoint, it is not possible to have aggregates at a group level for Fact Balance measure group (like any other many-to-many relationship, it has to be evaluated at query time), but in my experience the query response time could be still acceptable for many real-world scenarios. However, this query-time calculation has a very positive impact on the processing-time. If you need to add data to form a new group, it is necessary to process only Dim Groups dimension and Factless Customer Group measure group and these processes can be done incrementally! For this reason, I suggest you to consider this scenario even for on-the-fly modifications of custom groups made by end users, without relying on client-based solutions.

Remember that you need ETL processes to update and process group-related structures. The end user should not be able to manipulate the Dim_Groups dimension because this may lead to inconsistent data. Instead, the preferred approach is to implement UDM.

Cross-Time

Almost all measures in a data warehouse are time-dependent. The classical star schema has a fact table that contains numeric measures and many dimension tables that define the grain of any single measure. This is a good model (especially if you build an OLAP cube on it) for analyzing sales over a given period. Nevertheless, it does not show how distribution of dimension attributes changes over time. For this reason, Kimball's advice is to define a separate fact table that take "snapshots" of dimension state over time.

However, snapshot fact tables may not satisfy all user reporting needs: for example, it is hard to query for the change of an attribute distribution between two dates. We can do more leveraging on new SSAS features, as the many-to-many relationship. We will call "cross-time" the technique that combines "time snapshots" and many-to-many relationship to enhance analysis capabilities inside client tools like a pivot table.

Business scenario

While the cross-time technique can be applied to any slowly changing dimension (SCD), a typical scenario involves a Customer dimension. Customer attributes changes over time and SCD tracks the history changes. However, usually it is not so easy to analyze the SCD changes without a two-step operation that will require first selecting a set of customer with certain attributes at a specific date and then using this selection to query data and see measures or attributes values on a different date.

Typically, the existing star schema may look like the one illustrated in Figure 63. Here, we have a fact table with meaningful measures (in this case Balance is a non-additive measure over Time), a date dimension and a Customer SCD Type II dimension. Please note that here we have a snowflake schema for customers because application key has already been normalized in Dim_CustomerUnique table. This model also makes it easier to model distinct count measures as we have seen before.

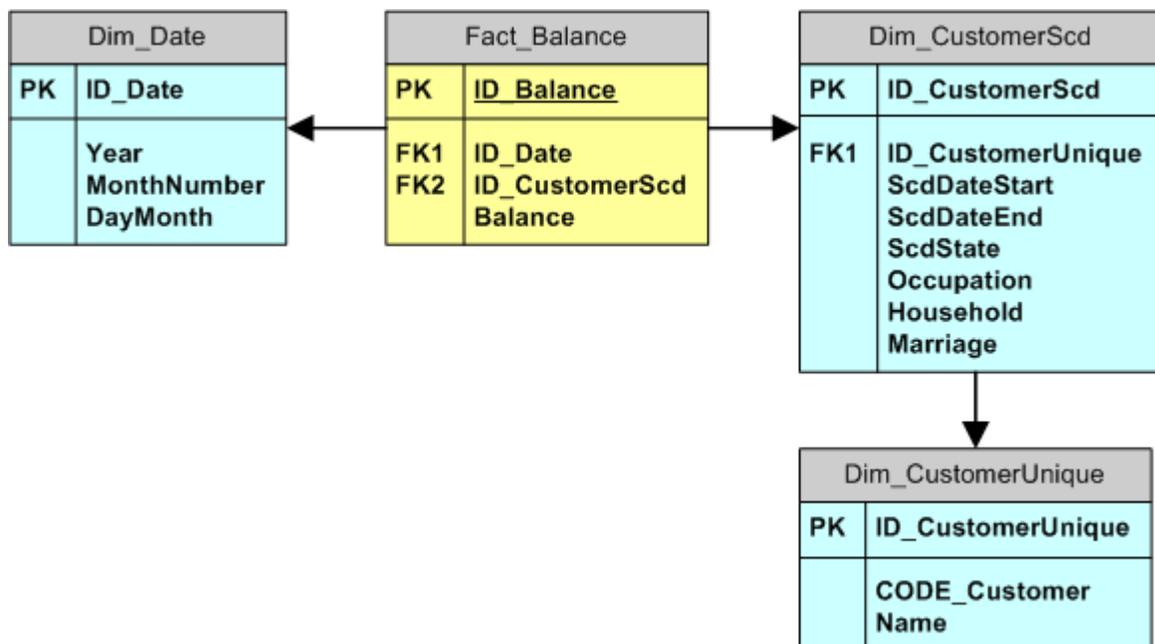


Figure 63 – Relational star schema with unique dimension

Our users may need to understand how customers have changed occupation from January to December 2005.

This first question can be answered by the SQL query shown beside, which is not so easy to build with a query builder. Moreover, changing the analyzed attribute (e.g.) requires a different syntax since it could not be parameterized.

```
SELECT
  c1.Occupation AS JanuaryOccupation,
  c2.Occupation AS DecemberOccupation,
  COUNT(*) AS Customers
FROM Dim_CustomerScd c1
INNER JOIN Dim_CustomerScd c2
  ON c2.ID_CustomerUnique = c1.ID_CustomerUnique
  AND '20051201' >= c2.ScdDateStart
  AND ('20051201' <= c2.ScdDateEnd OR c2.ScdDateEnd IS NULL)
WHERE '20050101' >= c1.ScdDateStart
  AND ('20050101' <= c1.ScdDateEnd OR c1.ScdDateEnd IS NULL)
GROUP BY c1.Occupation, c2.Occupation
```

What will happen if the end user wants to analyze balances for year 2005 for customers who had a Student occupation in January, irrespective of their current occupation? To answer to this second question, we can use SQL again, but this may not be practical to analyze if considering the fact that the user may favor OLAP browsers, such as pivot table of pivot char. Worse, the end user may mismatch one of the joins getting wrong results.

Ideally, we would like to track differences in attributes between different snapshots and relate standard measures (like balance in our sample) behavior over time for a group of customers at a given point of time.

```
SELECT b.ID_Date, SUM( b.Balance )
FROM Fact_Balance b
INNER JOIN Dim_CustomerScd c
  ON c.ID_CustomerScd = b.ID_CustomerScd
INNER JOIN Dim_CustomerUnique u
  ON u.ID_CustomerUnique = c.ID_CustomerUnique
INNER JOIN Dim_CustomerScd cj
  ON cj.ID_CustomerUnique = u.ID_CustomerUnique
WHERE cj.Occupation = 'Student'
  AND '20050101' >= cj.ScdDateStart
  AND ('20050101' <= cj.ScdDateEnd
  OR cj.ScdDateEnd IS NULL)
GROUP BY b.ID_Date
```

Implementation

As we have seen in Figure 63, ScdDateStart and ScdDateEnd do not give us an easy way to get the set of valid customers at a certain date period, especially if use a we want to do it from a pivot table report. The classical way is to denormalize the schema.

We could get a complete denormalization of customer attributes by making a snapshot table for them. Since we already have a SCD Type II customer dimension, we can shortcut the implementation process by making a snapshot table that only store the relationship between a date period and a customer version. In this way, we can obtain the previously discussed complete attribute snapshot table by only joining two tables (it saves space and probably execution time).

The tricky part is to allow the user able to select all the customer versions that had a particular attribute value for a particular date period. We will leverage many-to-many dimensions to do that. To relate easily different versions for the same customer, we store the unique customer identification in snapshot table too. Figure 64 illustrates the resulting relational model. I named the snapshot table as Factless_CustomerSnapshot. It will be the bridge table that relates different Date and Customer dimensions.

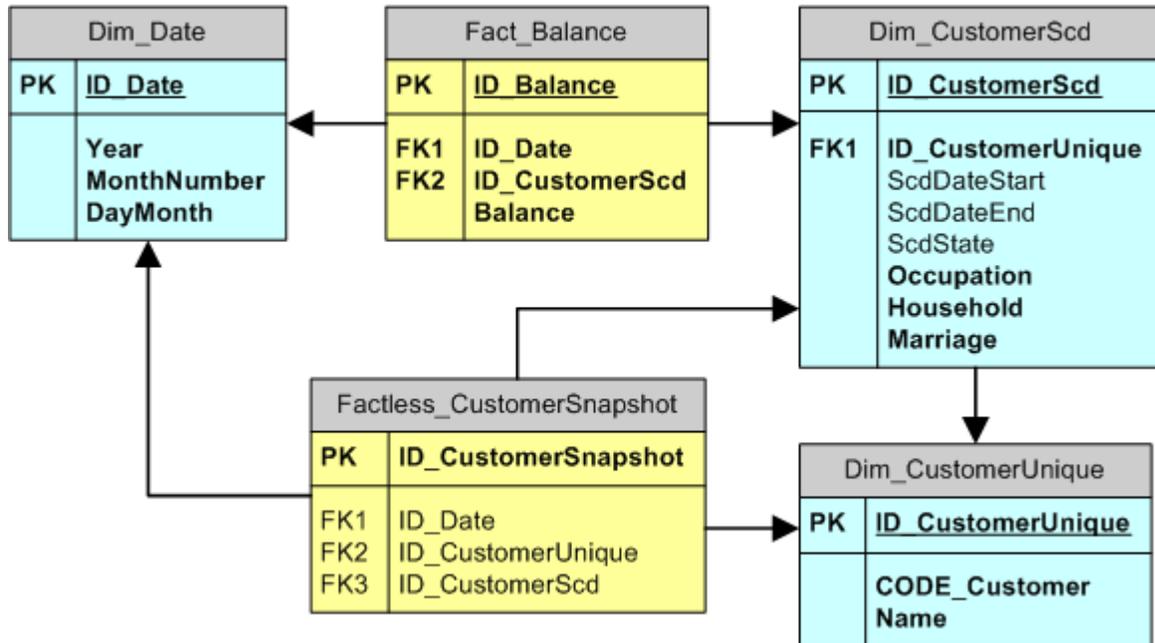


Figure 64 – Relational schema enable to cross-time analysis

We need to make a minor change to the data source view (Figure 65), by adding the vFactless_CustomerScd view. As we have already seen for Distinct Count scenario we cannot use a fact dimension as an intermediate measure group in a many-to-many relationship.

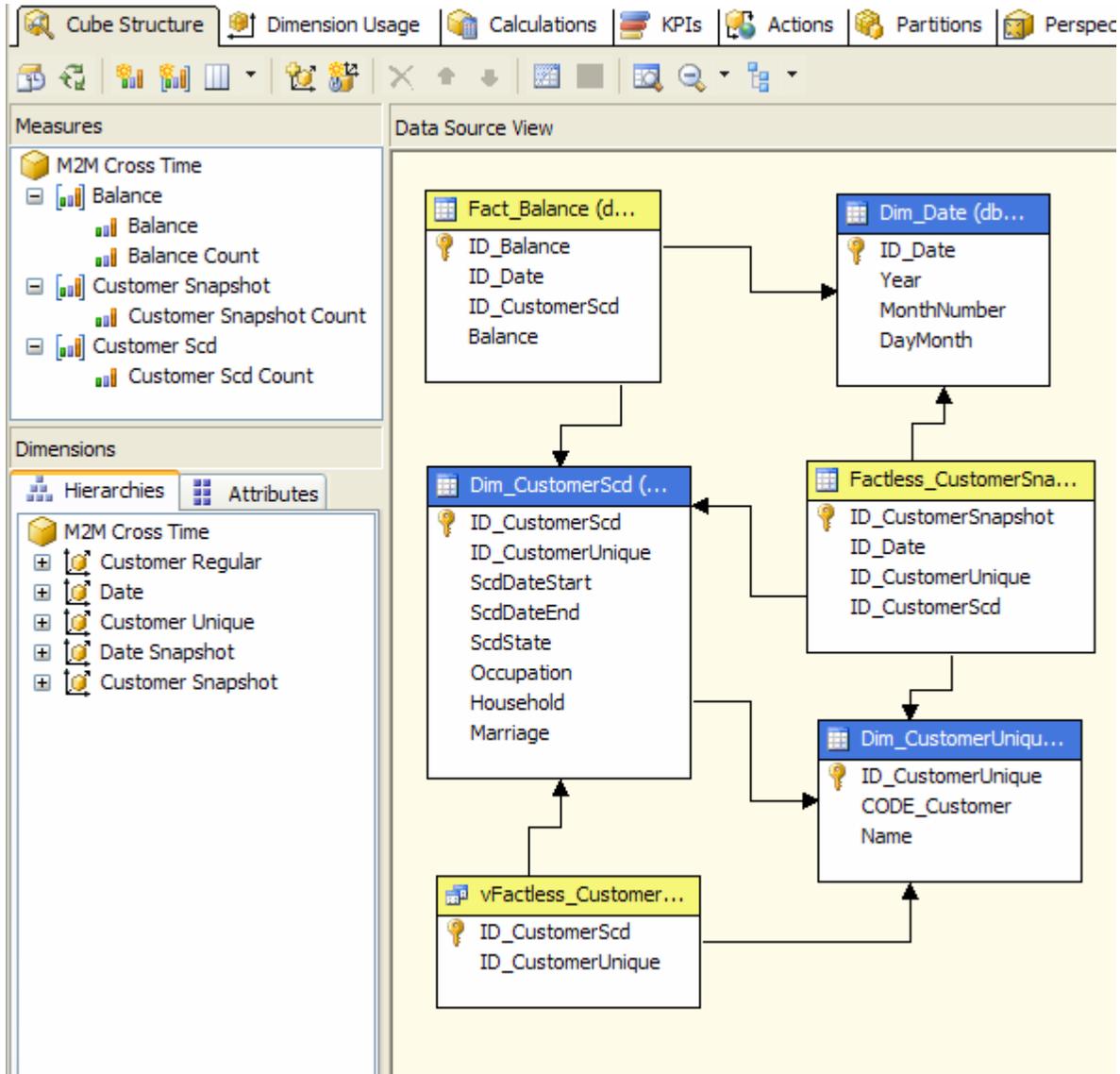


Figure 65 – Cross-Time Data Source View and cube structure

This first cube is built using role-playing dimensions. The Date dimension represents the date for balance measure group and the Date Snapshot dimension represents the date for a particular snapshot (in the sample UDM, I used a monthly snapshot, but you can choose a different granularity). We also have two Customer dimensions. Customer Regular references to the Balance measure group (it is the “regular” dimension for transactional measures) and thus is indirectly related to the Date dimension. Customer Snapshot is related to Date Snapshot and it is used to select a group of customers for a particular snapshot.

The Dim_CustomerUnique dimension becomes the bridge between Customer Snapshot and Customer Scd measure groups. The former mirrors the snapshot information, while

the later supports the many-to-many relationship between Customer Regular and Customer Scd dimensions. Note that Customer Unique dimension can also be used to query data for a particular customer given its application key.

Figure 66 shows the resulting dimension usage matrix. If you try to use the cube wizard, the dimension usage matrix will be very different. Once auto build completes, you need to add role-playing dimensions and you need also to change many dimension usage settings. Instead, I suggest you define the dimension usage manually, so you do not forget to adjust some relationships.

Dimensions	Balance	Customer Snapshot	Customer Scd
Customer Scd (Customer Regular)	ID Customer Scd	Customer Scd	ID Customer Scd
Date	ID_Date	Customer Scd	Balance
Customer Unique	Customer Regular	Dim Customer Unique	Dim Customer Unique
Date (Date Snapshot)	Customer Snapshot	ID_Date	Customer Snapshot
Customer Scd (Customer Snaps...)	Customer Snapshot	ID Customer Scd	Customer Snapshot

Figure 66 – Dimension Usage for Cross-Time cube

Before querying the cube, it is important to clarify the purpose of the measures used. Balance and Balance Count are regular measures coming from the Fact_Balance table. If we need to get an average balance, we should divide Balance by Balance Count. Customer Snapshot Count tells us how many customers are present in the snapshot for the current date selection. When you select a single snapshot date, Customer Snapshot Count gives you the number of customers for that snapshot for that date. Customer Scd Count may not be that useful to end users because it gives the number of customer versions for the current selection. For example, if you choose a single customer, it will show the number of different versions stored in Dim_CustomerScd (a SCD Type II Customer Dimension) for that customer.

To avoid confusion, I suggest you to hide Customer Scd Count and Customer Snapshot Count measures .

Figure 67 show a sample report. I filtered data by January 2005 (dimension Date Snapshot in filter area), I placed Year/Month of Date dimension on rows and Customer Snapshot Occupation attribute on columns. as you can see, we had three customers in January snapshot, one was a student and two were teachers.

Date Snapshot.Year - Month Number		Occupation					
1		Student		Teacher		Grand Total	
Year	Month Number	Balance	Balance Count	Balance	Balance Count	Balance	Balance Count
2005	1	110	1	220	2	330	3
	2	120	1	240	2	360	3
	3	130	1	260	2	390	3
	4	140	1	280	2	420	3
	5	150	1	300	2	450	3
	6	160	1	320	2	480	3
	7	170	1	340	2	510	3
	8	180	1	360	2	540	3
	9	190	1	380	2	570	3
	10	200	1	400	2	600	3
	11	210	1	420	2	630	3
	12	220	1	440	2	660	3
		Total	1980	12	3960	24	5940
Grand Total		1980	12	3960	24	5940	36

Figure 67 – Cross-Time results for snapshot customer occupation attribute

If we now place Customer Regular Occupation attribute on columns, we get the results shown in Figure 68. We can deduce that the student we had in January became an employee in March. If we had filtered by Student we would have obtained the same result without the Teacher column.

Date Snapshot.Year - Month Number		Occupation							
1		Employee		Student		Teacher		Grand Total	
Year	Month Number	Balance	Balance Count	Balance	Balance Count	Balance	Balance Count	Balance	Balance Count
2005	1			110	1	220	2	330	3
	2			120	1	240	2	360	3
	3	130	1			260	2	390	3
	4	140	1			280	2	420	3
	5	150	1			300	2	450	3
	6	160	1			320	2	480	3
	7	170	1			340	2	510	3
	8	180	1			360	2	540	3
	9	190	1			380	2	570	3
	10	200	1			400	2	600	3
	11	210	1			420	2	630	3
	12	220	1			440	2	660	3
		Total	1750	10	230	2	3960	24	5940
Grand Total		1750	10	230	2	3960	24	5940	36

Figure 68 – Cross-Time results for regular customer occupation attribute

Now we have a powerful tool to produce reports that cross snapshots, “transactional” dimensions and attributes. This is very useful to satisfy our second objective (occupation change between January and December) previously described in the business scenario, but still does not completely satisfy the first one (balances for

customers who were students in January). In fact, if we are to compare the January snapshot with the December customer data, we need to make a prior assumption that all customers we had in January also had balances in December. If this is correct, we can author the report shown in Figure 69.

Date.Year - Month Number ▼		Date Snapshot.Year - Month Number ▼				
12		1				
	Occupation ▼					
	Student	Teacher		Grand Total		
Occupation ▼	Balance	Balance Count	Balance	Balance Count	Balance	Balance Count
Employee	220	1			220	1
Teacher			440	2	440	2
Grand Total	220	1	440	2	660	3

Figure 69 – Cross-Time results comparing a snapshot with a date

It seems all good but it does not work well when we choose different months for the Date dimensions. In Figure 70, I chose Jan/Feb for Date Snapshot dimension and Nov/Dec for Date dimension. Now, it is difficult to understand how many customers we really have. We would have run in the same problem before if a customer had multiple balances for the same date. Therefore, dividing by the number of months selected is not a valid solution.

Date.Year - Month Number ▼		Date Snapshot.Year - Month Number ▼				
(Multiple Items)		(Multiple Items)				
	Occupation ▼					
	Student	Teacher		Grand Total		
Occupation ▼	Balance	Balance Count	Balance	Balance Count	Balance	Balance Count
Employee	430	2			430	2
Teacher			860	4	860	4
Grand Total	430	2	860	4	1290	6

Figure 70 - Cross-Time results comparing multiple periods

The last step is the longest one. We have to add a distinct count measure to get the right number of customers under any conditions. This is often the case when we have a SCD Type II dimension in our cube.

Figure 71 shows the new cube structure. While the Data Source View is unchanged, we have a new measure group (Customer Unique) that contains a new measure (Customers Distinct Count), which should be made visible to end users. Dim_CustomerUnique assumed also a fact table role.

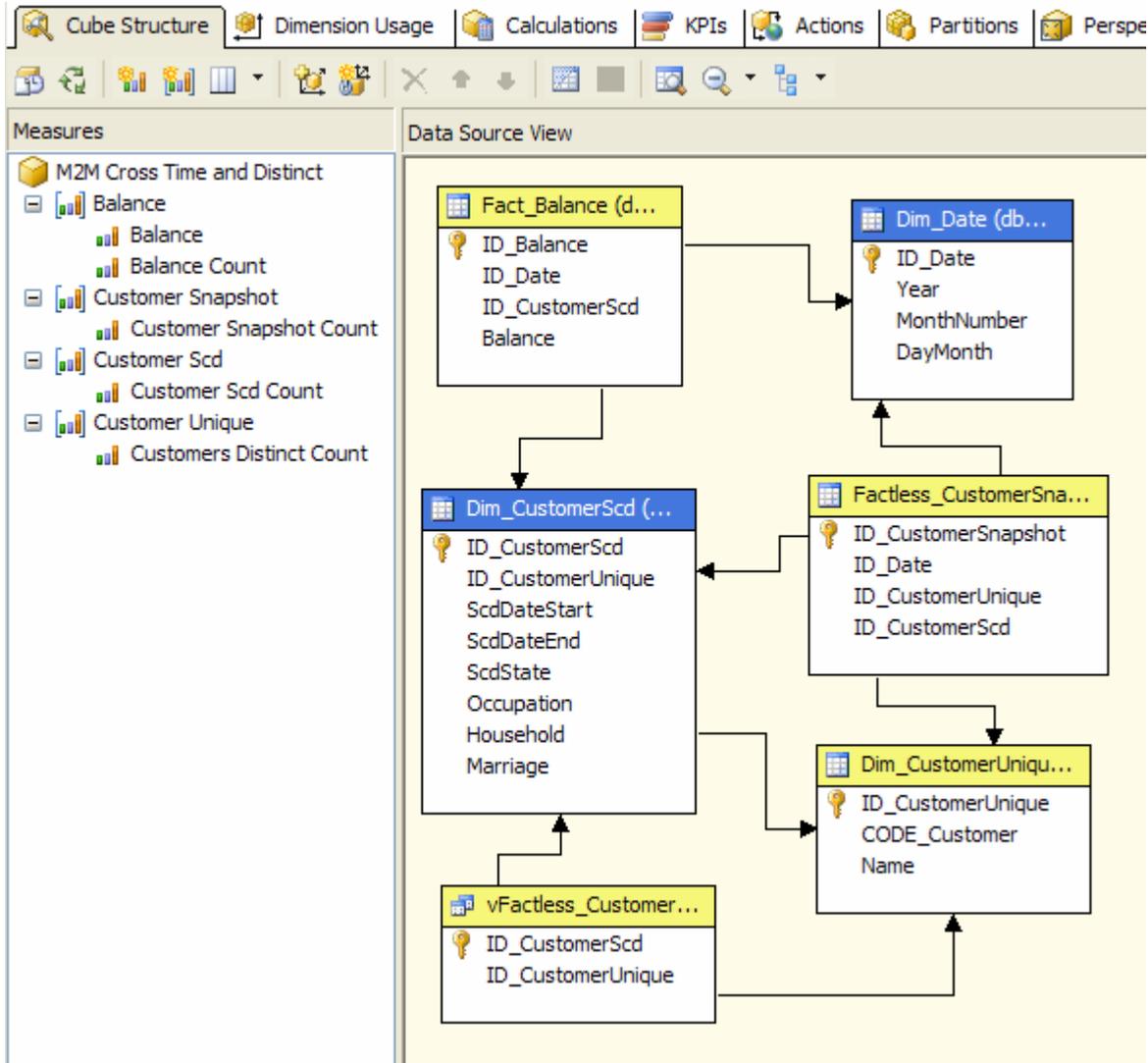


Figure 71 – Cross-Time Data Source View and cube structure (with distinct)

The new measure group has to be related correctly to existing dimensions (see Figure 72). It is important to define all relationships between Customer Unique and all cube dimensions other than Customer Unique dimension as many-to-many relationships.

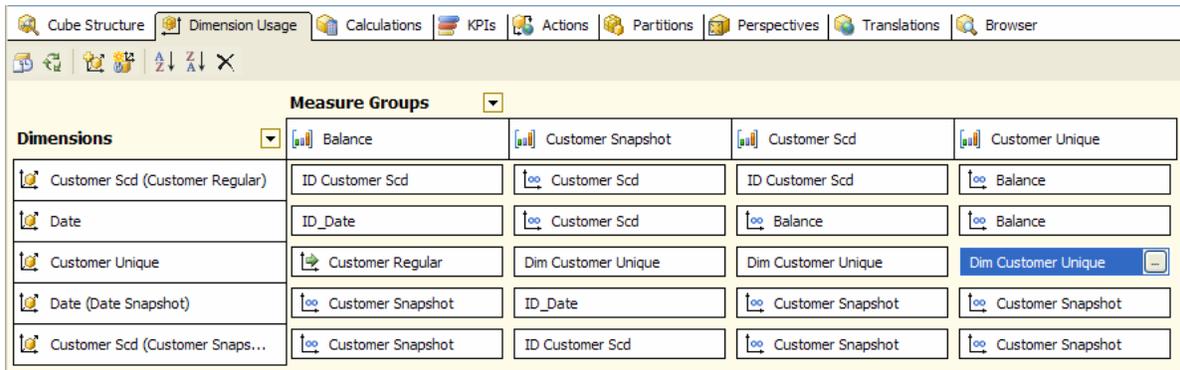


Figure 72 – Dimension Usage for Cross Time cube (with distinct customers)

Once this is done, we can get the correct results for our first objective (see Figure 73), that is how customers have changed occupation from January to December 2005.

Date Snapshot.Year - Month Number (Multiple Items)	Date.Year - Month Number (Multiple Items)		
	Occupation		
	Student	Teacher	Grand Total
Occupation	Customers Distinct Count	Customers Distinct Count	Customers Distinct Count
Employee	1		1
Teacher		2	2

Figure 73 – Cross-Time results comparing multiple periods with distinct count

To be honest, this report does not allow us to compare different snapshots because we assume that the regular “date” dimension can be treated as yet another snapshot dimension. This could be a valid assumption because the balance fact table is a different kind of snapshot table, but this would not be the case if we had a sales or movements fact table instead that one with balances. A better model that meets these requirements is presented in Transition Matrix section that follows next.

However, the use of a distinct count measure is often required by end users to ease the analysis of data when grouping more periods both in snapshot and/or in regular Date dimensions, especially when we have a more complex cube with other dimensions and relationships with other measure groups.

Transition Matrix

Have you ever been asked how many times a given attribute has changed between two dates? A typical question with customer segmentation could be “How many customers classified as type A in 2004 have converted to type B in 2005?”.

In such cases, Kimball suggests we consider a snapshot fact table. This is a good approach, but in the UDM we cannot relate the same time dimension twice to a fact table without having two different date columns. One possible solution could be to generate the Cartesian product of the time dimension but this could be very expensive in terms of storage space and processing time.

The many-to-many relationship allows us to duplicate the time dimension in the UDM without duplication of storage. By the term “transition matrix”, we will refer to a model that makes it possible to analyze the state transition of analyzed elements between two dates even when results are displayed in a pivot table report.

Business scenario

In general, every time we have a fact that is repeated over time against the same dimension members, we could analyze how related attributes change over time for that dimension member. A good example of it is the credit rating of a customer. In this scenario, we have a monthly update of the customer ratings and we are interested in analyzing their changes. Figure 74 shows the relational schema for this data.

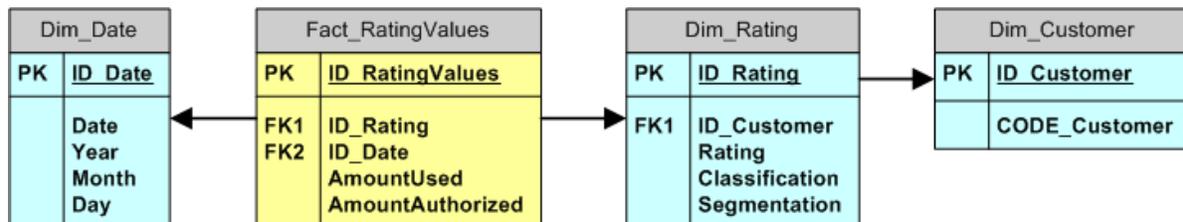


Figure 74 – Relational rating star schema

This star schema allows us creating a simple UDM, with Rating, Customer and Date dimensions. The fact table has two measures: the credit amount authorized and used for each customer on a specific date. One potential source of confusion is that a regular star schema is expected to have two completely independent dimensions for Rating and Customer, while our Customer dimension is a referenced dimension (a sort of a snowflake schema design) that is related to the fact table through the Rating dimension. However, there are good reasons to go for this design, as I will explain next.

Fact_RatingValues is a sort of snapshot fact table with periodic (probably monthly) updates. For each snapshot, we have current amounts (AmountUsed and AmountAuthorized) and a related rating. Chances are that the customer rating may not be updated frequently and regularly. We could have an original rating table with two date fields (DateStart and DateEnd) indicating the validity period of a certain rating (it ends when rating changes for any reason). Similarly, Dim_Rating could be created with the same criteria, becoming a Type II SCD. For the sake of simplicity, we do not have SCD canonical fields (DateStart, DateEnd, Current flag) because they are not relevant for us, but they could be added if needed! It is important to understand that the consistency between those SCD dates and the Fact_RatingValues snapshot must be granted by the process that populates the fact table, because it contains the only date

used by our analysis. Table 8 shows the history of rating changes used in our scenario. For the sake of simplicity again, AmountUsed and AmountAuthorized values will be respectively 50 and 100 for all Fact_RatingValues rows.

Table 8 – Ratings used for sample data in transition matrix model

DateStart	DateEnd	Customer	Rating	Classification	Segmentation
01/01/2005	15/04/2005	Frank	AAB	Class A	Retail
16/04/2005	21/06/2005	Frank	AAB	Class A	Affluent
22/06/2005		Frank	AAC	Class A	Affluent
01/01/2005	14/03/2005	Mark	AAA	Class A	Private
15/03/2005	08/05/2005	Mark	AAA	Class A	Affluent
09/05/2005		Mark	AAB	Class A	Affluent
01/01/2005	11/02/2005	Paul	AAA	Class A	Private
12/02/2005	17/05/2005	Paul	AAB	Class B	Private
18/05/2005		Paul	AAB	Class C	Private

Rating, Classification and Segmentation are independent attributes that are customer-related and could change over time. Our goal is to analyze the customer changes from one attribute state to another over time. The typical question we would like to answer is “How many customers with rating AAA have been downgraded to AAB from January to June?”

This scenario is very similar to the survey model we have discussed before. The main difference is that we have two class of dimension to “duplicate” in query (Date and Rating) as we can see in the side box with a possible SQL solution.

```
SELECT COUNT(*)
FROM Fact_RatingValues rv1
INNER JOIN Dim_Rating r1
  ON r1.ID_Rating = rv1.ID_Rating
INNER JOIN Dim_Date d1
  ON d1.ID_Date = rv1.ID_Date
INNER JOIN Dim_Rating r2
  ON r1.ID_Customer = r2.ID_Customer
INNER JOIN Fact_RatingValues rv2
  ON r2.ID_Rating = rv2.ID_Rating
INNER JOIN Dim_Date d2
  ON d2.ID_Date = rv2.ID_Date
WHERE d1.ID_Date = '20050131'
AND d2.ID_Date = '20050630'
AND r1.Rating = 'AAA'
AND r2.Rating = 'AAB'
```

```

SELECT
  RatingJanuary AS Rating,
  [AAA], [AAB], [AAC]
FROM (
SELECT
  r1.Rating AS RatingJanuary,
  r2.Rating AS RatingJune
FROM Fact_RatingValues rv1
INNER JOIN Dim_Rating r1
  ON r1.ID_Rating = rv1.ID_Rating
INNER JOIN Dim_Date d1
  ON d1.ID_Date = rv1.ID_Date
INNER JOIN Dim_Rating r2
  ON r1.ID_Customer = r2.ID_Customer
INNER JOIN Fact_RatingValues rv2
  ON r2.ID_Rating = rv2.ID_Rating
INNER JOIN Dim_Date d2
  ON d2.ID_Date = rv2.ID_Date
WHERE d1.ID_Date = '20050131'
  AND d2.ID_Date = '20050630'
) AS Ratings
PIVOT (
  COUNT(RatingJune)
  FOR RatingJune IN ( AAA, AAB, AAC)
) AS PivotTable
    
```

Another common request could be to produce a transition matrix of ratings from January to June: the box beside shows the PIVOT SQL query that generates the results in Table 9.

Table 9 – Transition matrix with SQL

Rating	AAA	AAB	AAC
AAA	0	2	0
AAB	0	0	1

Each row represents a rating in January and each column shows a rating in June. We can say that two customers who were rated AAA in January (Mark and Paul) have been declassified to AAB in June (even if the date of the rating change could be anywhere between these two

dates). Moreover, the customer who was rated AAB in January (Frank) has been declassified to AAC in June. Tough times for credit ratings!

As before, SQL solutions are not very flexible or easy to build for an end user. A pivot table that can generate the results shown in Table 9 would be greatly appreciated.

Implementation

We will implement a UDM with 3 dimensions: Date, Customer and Rating. These dimensions corresponds to the Dim_* tables shown in Figure 74.

The cube needs to duplicate some dimensions: we need two Date dimensions and two Rating dimensions. The end user would select two dates to see two different ratings, one for each selected date. We will reuse the technique discussed in the Survey model, where we used role-playing dimensions. Then we will change relationships with the measure groups by using views representing different and independent factless fact tables, although they duplicate the same original data.

As in the Survey model, we need to maintain the relationship between the selected attributes (ratings in this case) of the same customer. For this reason, we create two named views that we will use as factless fact tables between Customer and Rating dimension: these views (see side box) are based on Dim_Rating, which is already a sort of bridge table, since it relates rating and customers. As we have two independent measure groups (fact tables) between Customer and Rating dimensions, we also need two independent measure groups between Rating and Date dimensions. Before continuing further, look at Figure 75 to get a complete picture of the model we are going to build.

```

SELECT
  ID_Rating,
  ID_Customer
FROM Dim_Rating
    
```

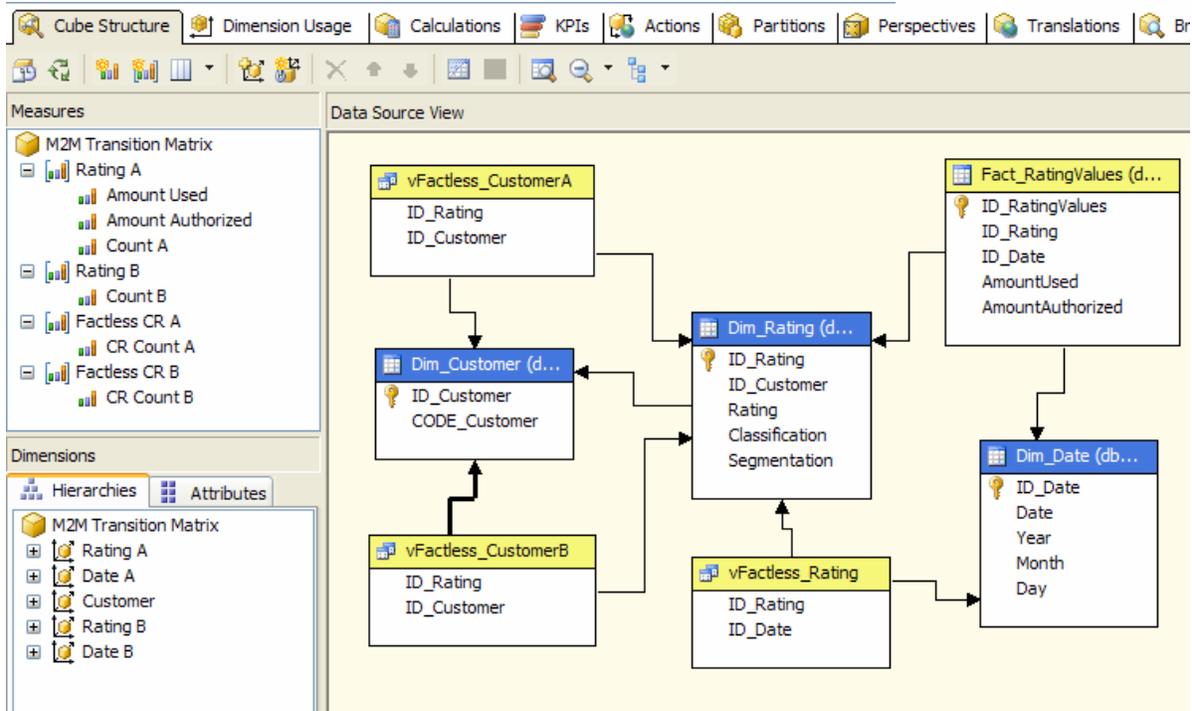


Figure 75 – Transition Matrix cube structure

While we already have Fact_RatingValues fact table to relate Rating and Date, we need another named view (vFactless_Rating, defined in side box) to get an independent alternative relationship between these dimensions. We do not need to duplicate the other Fact_RatingValues measures (AmountUsed and AmountAuthorized).

```
SELECT
  ID_Rating,
  ID_Date
FROM Fact_RatingValues
```

The two views discussed above are named vFactless_CustomerA and vFactless_CustomerB. They serve as a source for the Factless CR A and Factless CR B measure groups. These measure groups contains only one measure (row count) that will be useful to define the required many-to-many relationships. Count A and Count B measures, used in Rating A and Rating B measure groups, are needed to get many-to-many relationships working.

Why these views, measure groups and dimensions have suffixes A and B? To understand this, take a look at Figure 76.

Dimensions	Rating A	Rating B	Factless CR A	Factless CR B
Rating (Rating A)	Dim Rating	Factless CR B	Dim Rating	Factless CR A
Date (Date A)	ID_Date	Factless CR B	Rating A	Factless CR A
Customer	Rating A	Rating B	Customer Code	Customer Code
Rating (Rating B)	Factless CR A	Dim Rating	Factless CR B	Dim Rating
Date (Date B)	Factless CR A	ID_Date	Factless CR B	Rating B

Figure 76 – Transition Matrix dimension usage

We can think of A and B as two axes in our transition matrix: if A is the starting point (i.e. the rating on a certain date), then B is the ending one (the rating on another date). The same suffix is used in measure groups and role-playing dimensions. The Rating A measure group has a direct relationship with Rating A and Date A dimensions. It also has a referenced dimension relationship with Customer through Rating A dimension and has many-to-many relationships with Rating B and Date B dimensions. The opposite is true for the Rating B measure group (a regular relationship with B-suffixed dimensions and many-to-many relationships with A-suffixed dimensions).

The other two measure groups function as links between Rating and Customer dimension: it is very important to observe the relationships used in this case. The factless measure groups (Factless CR *) have to connect the rating of a customer with all the dates in which this rating exists for that customer. Thus Factless CR A has a direct relationship with Rating A and Customer dimension, while the many-to-many relationship with Date A has to be defined through the Rating A measure group; likewise Factless CR B goes direct to the Rating B and Customer dimensions and goes to Date B through the Rating B measure group.

All relationships between a measure group and dimensions with a different suffix must be many-to-many relationships using a Factless CR measure group as an intermediate one. The rating measure groups use a Factless CR with the same suffix, while Factless measure groups use the other Factless CR measure group as intermediate one. If you are confused, remember the rule of thumb I suggested for dealing with cascading many-to-many scenarios: you have to choose the intermediate measure group that is nearest to the measure group [you are starting from], considering all the possible measure groups that you can cross going from the measure group to the dimension of interest. Looking at data source view in Figure 75 and trying to follow the links between each dimension and each measure groups, you should be able to apply this rule easily.

As we have seen in Figure 75, only Rating A measure group has Amount-* measures: if you want to name measures more specifically than using A and B suffixes (e.g. with "Start" end "End"), then it could be a good idea to duplicate the Amount-* measures in both measure groups just to give more flexibility to the end user.

Here is a potential source of confusion when querying the cube. When we use role-playing dimensions, we can change the dimension name but not internal hierarchies

and/or attributes names. If the OLAP browser does not show the dimension name near the attribute, the end user will have to pay attention to which attributes are used to understand the meaning of results: this is the case for the Cube Browser, which is based on OWC components.

Figure 77 shows the resulting report with Rating A on columns, Customer and Date A (level Months) on rows and the Amount Authorized measure in the data area. This report shows the underlying data at the maximum granularity. We can see all snapshots available for all customers, where the amount is displayed in the column corresponding to the rating that customer has on the month for the matching row.

Drop Filter Fields Here				
		Rating ▾		
		AAA	AAB	AAC
Custc ▾	Month ▾	Amount	Amount	Amount
☐ Mark	1	100		
	2	100		
	3	100		
	4	100		
	5		100	
	6		100	
☐ Paul	1	100		
	2		100	
	3		100	
	4		100	
	5		100	
	6		100	
☐ Frank	1		100	
	2		100	
	3		100	
	4		100	
	5		100	
	6			100

Figure 77 – Amount Authorized by Customer, Month and Rating

So far, we have not used the B-suffixed dimensions. We will do that in the following example.

In Figure 78, we get all the answers for our initial business scenario. The filter area contains January (selected from the Date A dimension) and the columns contains Rating attribute from the Rating A dimension. The rows are defined as a Cartesian product of the months and ratings available from the Date B and Rating B dimensions. For each month, there is a row for each possible rating. The meaning of each data cell is “how many customers with [column rating] in January have [row rating] in [row month name]”. Obviously, January rows have the same rating as shown in the corresponding columns. We can see that June has two original AAA customers who turned into AAB and one AAB who turned into AAC: this is the same data that was presented in Table 9, except that the meaning of the rows and columns is inverted.

Date A.Month ▾		Rating ▾		
1		AAA	AAB	AAC
Month ▾	Rating ▾	Count A	Count A	Count A
☐ 1	AAA	2		
	AAB		1	
	AAC			
☐ 2	AAA	1		
	AAB	1	1	
	AAC			
☐ 3	AAA	1		
	AAB	1	1	
	AAC			
☐ 4	AAA	1		
	AAB	1	1	
	AAC			
☐ 5	AAA			
	AAB	2	1	
	AAC			
☐ 6	AAA			
	AAB	2		
	AAC		1	

Figure 78 – Transition Matrix of Ratings between January and other months

Besides allowing us to use a pivot table for transition matrix queries, our model also simplifies direct queries: the side box shows the MDX query that produces the same result shown in Table 9, but it has a syntax that is much easier to write and understand than the SQL one we used before. The only difference in the results is that null values are returned instead of zeros.

```
SELECT
  [Rating B].Rating.Rating ON 0,
  NON EMPTY [Rating A].Rating.Rating ON 1
FROM [M2M Transition Matrix]
WHERE (
  [Date A].[Month].[1],
  [Date B].[Month].[6],
  Measures.[Count A] )
```

Multiple Hierarchies

Parent-child dimensions are a useful feature of Analysis Services. They can be used to model hierarchical and fast changing dimensions like sales or employee organizations. A limitation of this feature is that you can define only one parent-child hierarchy in a dimension. In the real world, this may be an issue. For example, in the middle of a company reorganization, someone may need to analyze alternatively the present with the eyes of the past (actual data for previous organization hierarchy) and the past with the eyes of the present (past data for actual organization hierarchy).

The “multiple hierarchies” is a model that leverages the many-to-many relationship feature: it allows us to assign a single leaf member to many different logical parent-child hierarchies, by using a single physical parent-child hierarchy.

Business scenario

As usual, let us explore a business scenario that can be easily applied to many different situations. In this case, we have a branch organization that can be viewed under different hierarchies: one is the regional hierarchy, which has an impact on logistical and technical aspects; the other is the market hierarchy, responsible for customer market analysis. Table 10 shows us the hierarchies associated with each company branch.

Table 10 – Branch Hierarchies

Branch	Regional Hierarchy	Market Hierarchy
NY01	North West	Corporate / Enterprise
NY02	North West / New York	Corporate / Small
NY03	North West / New York	Private
BO01	North West	Corporate / Small
BO02	North West / Boston	Corporate / Small
BO03	North West / Boston	Private

Oftentimes, we may face the issue that the hierarchies change over time, both in the number of hierarchies and in the hierarchy’s data. As usual, we do not want to change the relational and dimensional schemas each time we need to modify these logical views.

The relational model to work around this issue is relatively simple: Figure 79 shows that we have a Dim_Branch dimension table that contains real branches and a Dim_BranchHierarchies dimension table that associates each Dim_Branch row to many different hierarchies. As always, the ETL process must maintain data integrity in those tables.

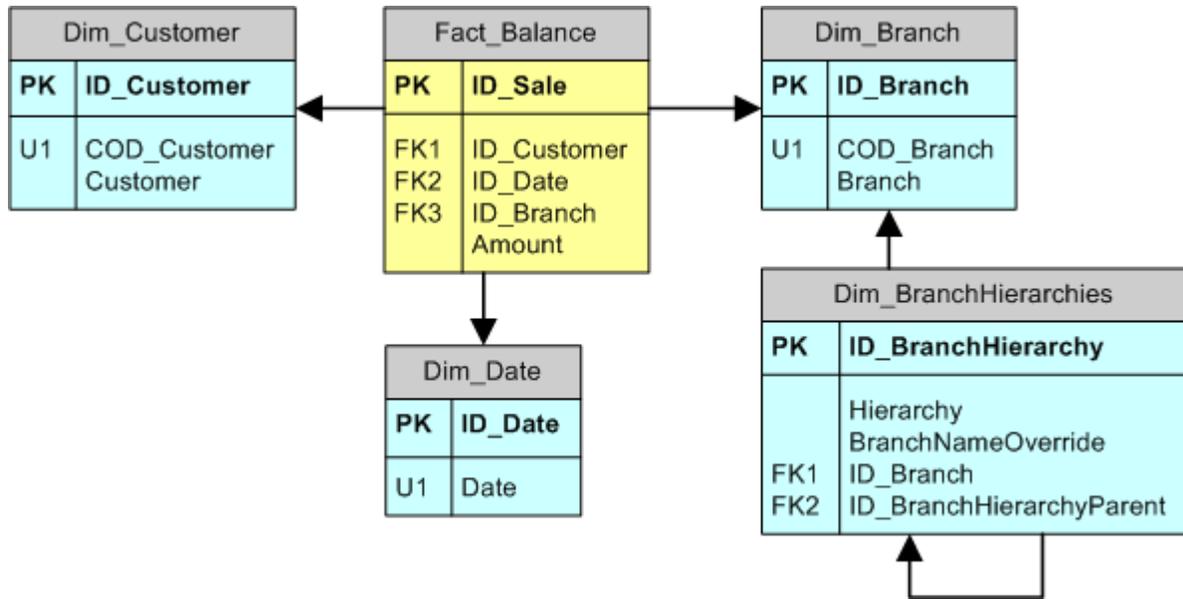


Figure 79 – Relational schema for Multiple Hierarchies

The Dim_BranchHierarchies table defines different hierarchies for the same branch. We can obtain a single parent-child hierarchy by filtering the rows in Dim_BranchHierarchies for a specific Hierarchy attribute value.

A Dim_Branch row should exist only once for each hierarchy and a row in Dim_BranchHierarchies must have a reference to an ID_Branch only for the leaves members of the hierarchy. Intermediate nodes may not have a corresponding ID_Branch: in this case, it is necessary to define a value for BranchNameOverride, which is the name to give to the node. If BranchNameOverride is not defined, the node/leaf name is the branch name referenced by ID_Branch.

To understand better how this model is populated, we can look at the dimensions members. Table 11 shows the Dim_Branch data. As explained, we have only branches that have corresponding measures in Fact_Balance and it is not necessary to have rows for the intermediate nodes of a hierarchy.

Table 11 – Dim_Branch content

ID_Branch	COD_Branch	Branch
1	NY01	NY01
2	NY02	NY02
3	NY03	NY03
4	BO01	BO01
5	BO02	BO02
6	BO03	BO03

The two hierarchies are described in Dim_BranchHierarchies, as shown in Table 12. The field BranchNameOverride is used only for intermediate hierarchy nodes. It could be used also to rename a leaf name for a particular hierarchy. When its value is NULL then we intend to use the branch name referenced by ID_Branch as the name of the item in the hierarchy.

Table 12 – Dim_BranchHierarchies content

ID_Branch-Hierarchies	Hierarchy	BranchName-Override	ID_Branch	ID_Branch-HierarchyParent
1	Regional		1	7
2	Regional		2	8
3	Regional		3	8
4	Regional		4	7
5	Regional		5	9
6	Regional		6	9
7	Regional	North West		
8	Regional	New York		7
9	Regional	Boston		7
10	Market		1	17
11	Market		1	18
12	Market		2	19
13	Market		3	18
14	Market		4	18
15	Market		5	19
16	Market	Corporate		
17	Market	Enterprise		16
18	Market	Small		16
19	Market	Private		

To keep things simple, each branch has a single customer and a single balance amount for each date and the balance amount is the same for all customers on that date (Table 12 shows the same balance amount of 100 for January).

Our goal is to query a UDM that has many parent-child hierarchies for the Branch dimension. The query result should automatically show only the members of the selected hierarchy.

Implementation

The relational model shown in Figure 79 is close to the envisioned UDM. However, Analysis Services does not directly support the relationship that exists between Dim_Branch and Dim_BranchHierarchies, because we have a one-to-many relationship while only many-to-one relationships can be implemented through referenced dimensions.

We already solved an analogous problem in the Multiple Groups scenario using a many-to-many relationship through the expanded model that is shown in Figure 80. We define a view (see side box) that corresponds to Factless_BranchHierarchies in our model. This query filters all rows that do not relate to a branch: these rows in our sample are all intermediate nodes in the hierarchies.

```
SELECT
  ID_BranchHierarchy,
  ID_Branch
FROM Dim_BranchHierarchies
WHERE ID_Branch IS NOT NULL
```

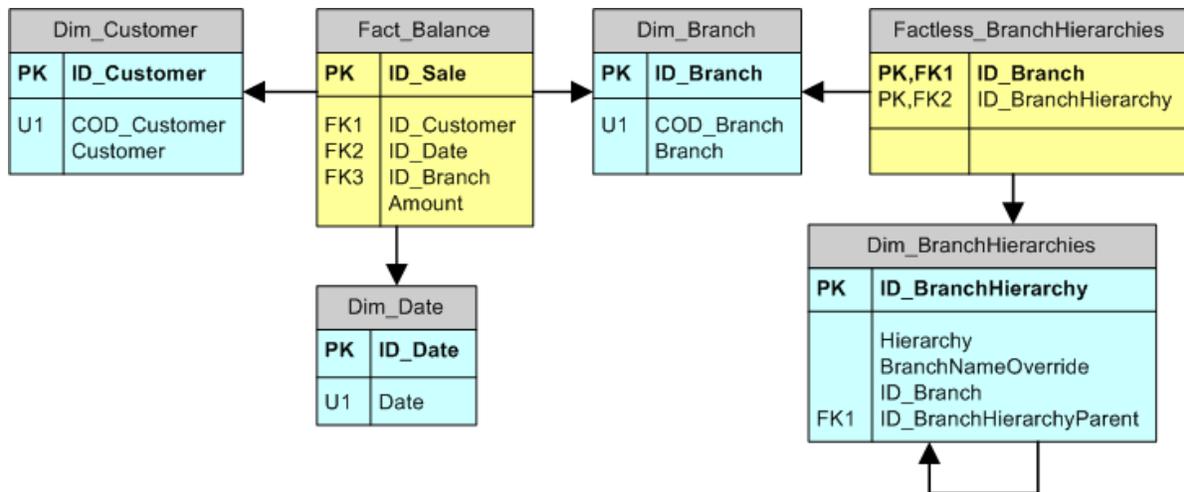


Figure 80 – Relational schema expanded for Multiple Hierarchies

We have 4 dimensions in our UDM: the only one that deserves more attention is the one based on Dim_BranchHierarchies.

First, we create a named view (see side box) called `vDim_BranchHierarchies`, which we will use as a source for the dimension. This is necessary because we have to define the name of all the nodes and leaves for all the hierarchies; our relational model defines an override mechanism for the node name that has to be resolved for Analysis Services.

```
SELECT
  h.ID_BranchHierarchy,
  h.ID_Branch,
  h.ID_BranchHierarchyParent,
  h.Hierarchy,
  COALESCE( h.BranchNameOverride,
            b.COD_Branch ) AS COD_Branch,
  COALESCE( h.BranchNameOverride,
            b.Branch ) AS Branch
FROM Dim_BranchHierarchies h
LEFT JOIN Dim_Branch b
ON h.ID_Branch = b.ID_Branch
```

Then, we create the dimension Branch Hierarchies shown in Figure 81. It has a single parent-child hierarchy (Hierarchized Branch) and an attribute (Hierarchy) containing the names of the available hierarchies. Filtering by this attribute, users will be able to see only nodes and leaves of the desired hierarchy.

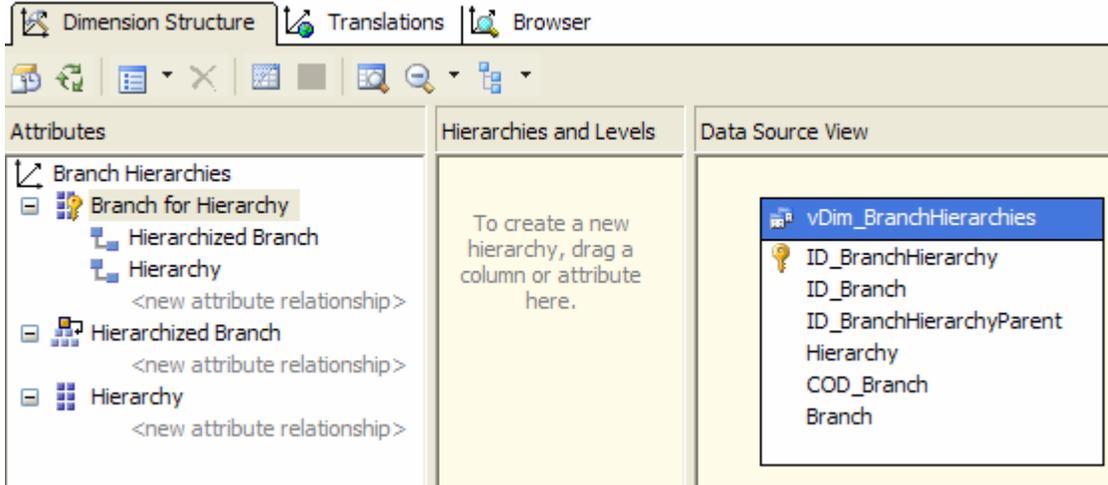


Figure 81 – Dimension Branch Hierarchies

It is important to understand what keys and names we use for the dimension attributes. Table 13 shows the most important properties used. Note that the InstanceSelection property for Hierarchy attribute should be set to MandatoryFilter (so that a client application that supports this property could suggest the selection of a member to end users).

Table 13 – Dim_BranchHierarchies attributes properties definition

Attribute Name	Hierarchy Visible	Usage	Key Column	Name Column
Branch for Hierarchy	False	Key	ID_BranchHierarchy	Branch
Hierarchized Branch	True	Parent	ID_BranchHierarchyParent	(none)
Hierarchy	True	Regular	Hierarchy	(none)

Figure 82 shows that by browsing the Hierarchized Branch hierarchy we can get all hierarchies: Corporate and Private are top-level nodes for the Market hierarchy, while North West is the only top-level node for the Regional hierarchy. If we filter by the Hierarchy attribute, then we can only browse the nodes of the selected hierarchy.

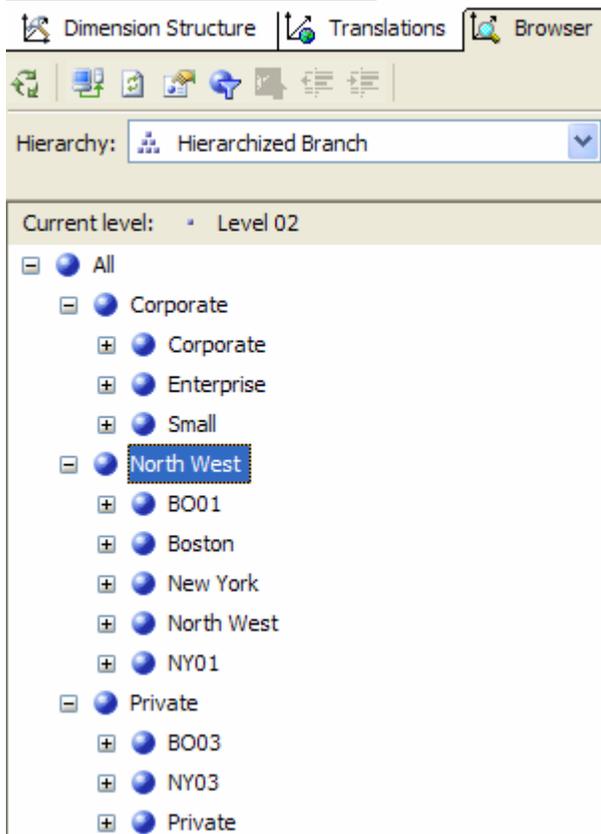


Figure 82 – Browse Branch Hierarchies dimension without filtering

The cube definition shown in Figure 83 has two measure groups, one for the facts (Fact Balance) and the other (Branch Hierarchies) for the many-to-many relationship between Branch and Branch Hierarchies dimensions.

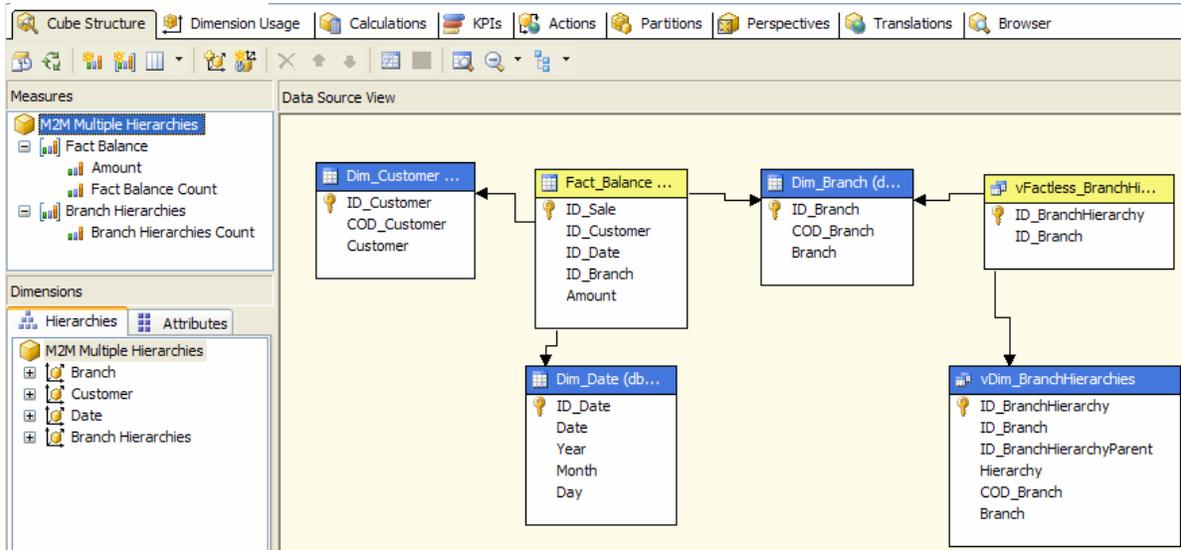


Figure 83 – Multiple Hierarchies Cube Structure

At this point, the dimension usage shown in Figure 84 should be easy to grasp. Since the Branch Hierarchies measure group only has a hidden measure, relationships between this measure group and the Customer and Date dimensions are not needed. This is why the Branch Hierarchies measure group only has two regular relationships with Branch and Branch Hierarchies dimension. The Fact Balance measure group has a many-to-many relationship with the Branch Hierarchies dimension using the Branch Hierarchies measure group as intermediate.

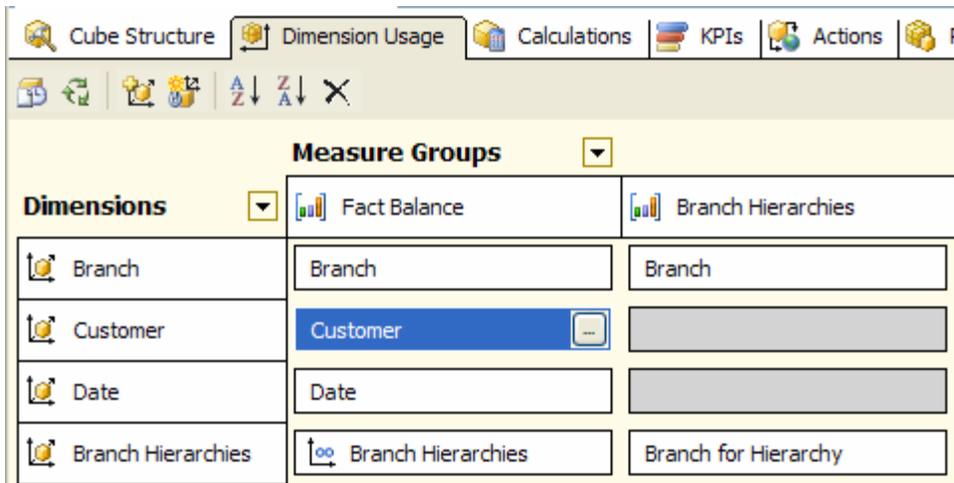


Figure 84 – Multiple Hierarchies Dimension Usage

Browsing the cube with a NON EMPTY clause give us the expected results. The pivot table shown in Figure 85 has the Hierarchized Branch attribute/hierarchy on the rows and two filters (Date is January 2005 and the Hierarchy attribute of the Branch

Hierarchies dimension is Regional). The result displays the same hierarchy shown in Table 10 for the Regional hierarchy.

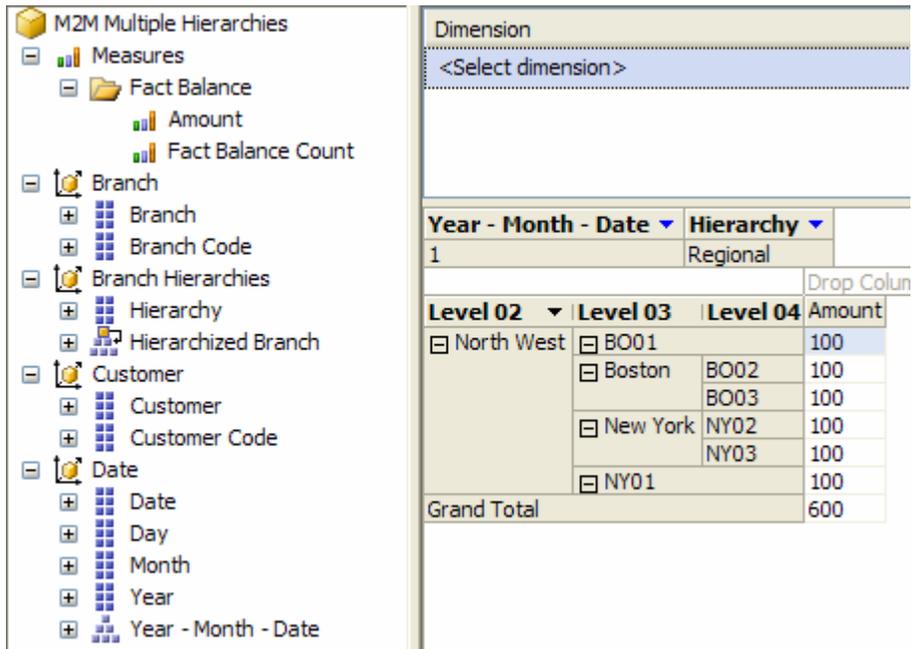


Figure 85 – Multiple Hierarchies query filtered by Regional hierarchy

The only change to the pivot table shown in Figure 86 is the filter value for the Hierarchy attribute, which is set to Market. This time the pivot table displays the hierarchy defined in Table 10 for the Market hierarchy.

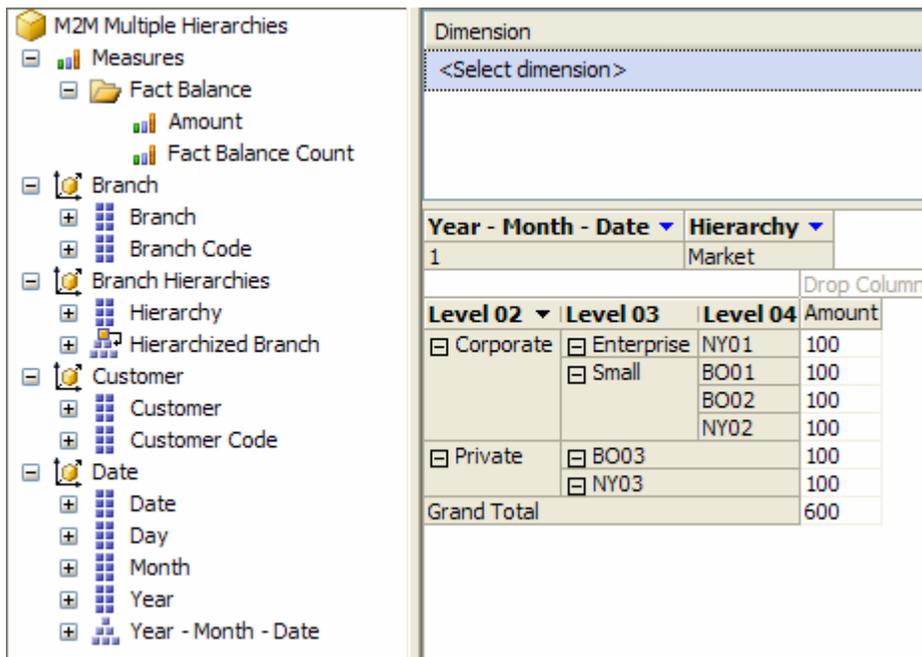


Figure 86 – Multiple Hierarchies query filtered by Market hierarchy

The only problem now is that the OLAP browser may not be smart enough to filter dimension browsing when you already put a filter on the table. If you define a subcube

by applying a dimension filter in the area above pivot table (see Figure 87), the Cube Browser will show the filtered members only, but this does not happen if you place the filter in the pivot table filter area (see Figure 88), because of a different filtering implementation.

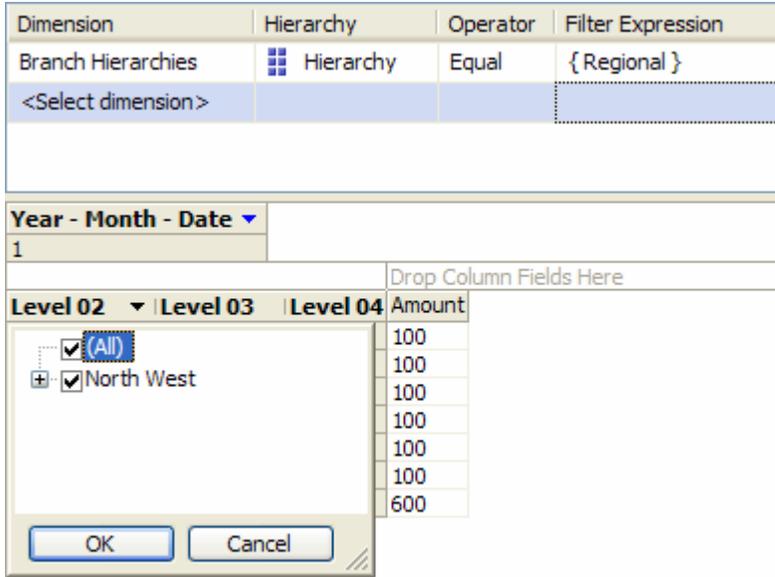


Figure 87 – Dimension filter lets filter dimension browsing too

In the case your OLAP browser does not support subcube filtering, or it may be too complex for the end user to use this feature, you could define a single top-level member for each hierarchy that corresponds to the name of the hierarchy. I use this strategy to facilitate the use of “legacy” clients like Excel 2003 with this model.

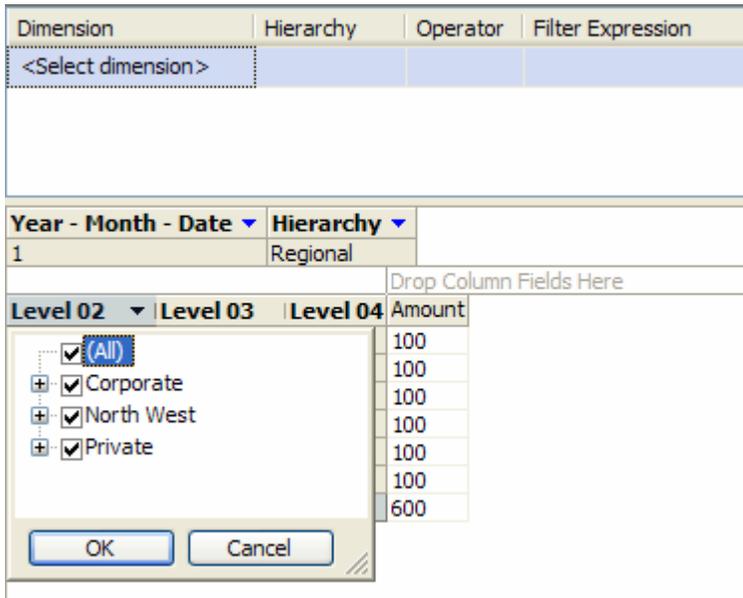


Figure 88 – Pivot table filter does not filter dimension browsing

The Multiple Hierarchies model allows the creation of a new hierarchy without the need to restructure neither the relational nor the dimensional schemas. In addition, changes

only need an incremental update of the Branch Hierarchies dimension and measure group. It is interesting for on-the-fly modifications as those suggested at the end of the Multiple Groups scenario: you could provide a user interface to create new hierarchies that become available on the server immediately for all users, without the need for a full process.

Conclusions

We examined several models that leverage on the UDM many-to-many relationships feature to solve real world business problems. There are many other uses and scenarios that we have not considered here. Many-to-many relationships open a wide (and unexplored) world of opportunities. I hope that the models presented in this paper will help you to approach this new revolutionary world of data analysis.

The most important technique to master with many-to-many relationships in Analysis Services 2005 is the use of cascading many-to-many relationships. The other skill you have to acquire is the ability to create a relational model that can easily be represented in a UDM, even if the relational model by itself cannot be easily queried by SQL. By leveraging the many-to-many relationships, you may need relatively simple MDX queries instead of rather complex SQL queries with one or more subqueries.

I hope that future releases of Analysis Services will assist further in creating a UDM with many-to-many relationships. Currently it can be very difficult to add a measure group or a dimension to a UDM with several many-to-many relationships: many of the choices that are offered when you select the intermediate measure group do not make sense, a better order or a simple hint suggesting the most probable right choice would be helpful.

Performance analysis and scalability are other areas that require a further study. Recommendations presented in this document are influenced by particular stress tests that I performed last year. In general, the flexibility provided by these models is much more important than possible degradation in performance. Without precise rules to anticipate performance issues as a result of using the many-to-many relationships, the most effective practice is to run tests measuring the response times with your own data.

Feedbacks are most welcome: a support forum is available on <http://www.sqlbi.com/forum.aspx>.

Links

<http://www.sqlbi.com/manytomany.aspx>: the project home page for this paper and related resources

<http://www.sqlbi.com/forum.aspx>: contains a forum dedicated to many-to-many dimensional modeling

<http://www.sqlbi.com>: community dedicated to Business Intelligence with SQL Server

http://sqlblog.com/blogs/marco_russo: blog of Marco Russo (author)

<http://www.sqlserveranalysiservices.com/OLAPPapers/IntroToMMDimensionsV2.htm>: introduction to Many-to-Many Dimensions by Richard Tkachuk (contains an explanation of Visual Total limitations)