**SQL Server Analysis Services (SSAS) Tutorial**

## Overview

SQL Server Analysis Services (SSAS) is the technology from the Microsoft Business Intelligence stack, to develop Online Analytical Processing (OLAP) solutions. In simple terms, you can use SSAS to create cubes using data from data marts / data warehouse for deeper and faster data analysis.   
  
Cubes are multi-dimensional data sources which have dimensions and facts (also known as measures) as its basic constituents. From a relational perspective dimensions can be thought of as master tables and facts can be thought of as measureable details. These details are generally stored in a pre-aggregated proprietary format and users can analyze huge amounts of data and slice this data by dimensions very easily. Multi-dimensional expression (MDX) is the query language used to query a cube, similar to the way T-SQL is used to query a table in SQL Server.  
  
Simple examples of dimensions can be product / geography / time / customer, and similar simple examples of facts can be orders / sales. A typical analysis could be to analyze sales in Asia-pacific geography during the past 5 years. You can think of this data as a pivot table where geography is the column-axis and years is the row axis, and sales can be seen as the values. Geography can also have its own hierarchy like Country->City->State.  Time can also have its own hierarchy like Year->Semester->Quarter. Sales could then be analyzed using any of these hierarchies for effective data analysis.  
  
A typical higher level cube development process using SSAS involves the following steps:  
  
1) Reading data from a dimensional model  
2) Configuring a schema in BIDS (Business Intelligence Development Studio)  
3) Creating dimensions, measures and cubes from this schema  
4) Fine tuning the cube as per the requirements  
5) Deploying the cube  
  
In this tutorial we will step through a number of topics that you need to understand in order to successfully create a basic cube. Our high level outline is as follows:

* Design and develop a star-schema
* Create dimensions, hierarchies, and cubes
* Process and deploy a cube
* Develop calculated measures and named sets using MDX
* Browse the cube data using Excel as the client tool

When you start learning SSAS, you should have a reasonable relational database background. But when you start working in a multi-dimensional environment, you need to stop thinking from a two-dimensional (relational database) perspective, which will develop over time.

# Creating a Sample SSAS Project and Cube

## Overview

Data in Online Transaction Processing (OLTP) systems is suited to support convenient data storage for user-facing applications. The data model in such systems is highly normalized. For data warehousing environments, data is required to be in a schema that supports a dimensional model. Data is therefore transformed from the OLTP storage systems to a data warehouse using ETL, so that data can be aligned in a suitable format to create data marts from the data warehouse.

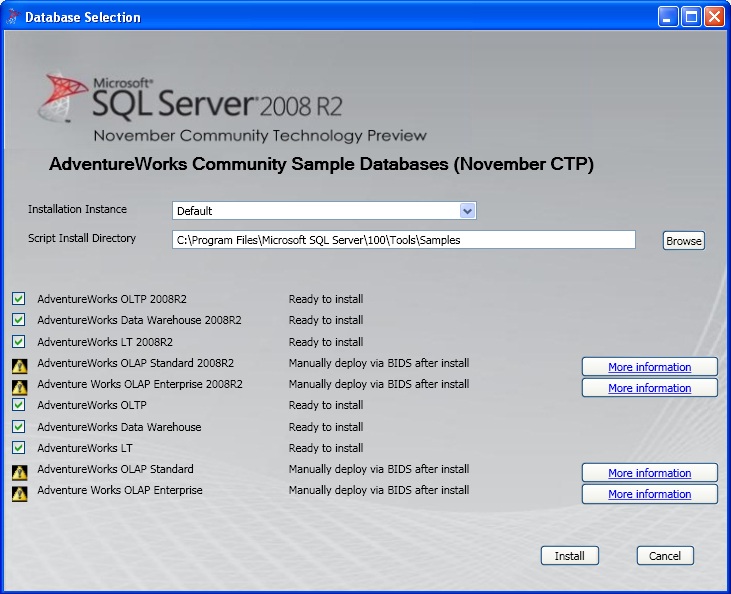
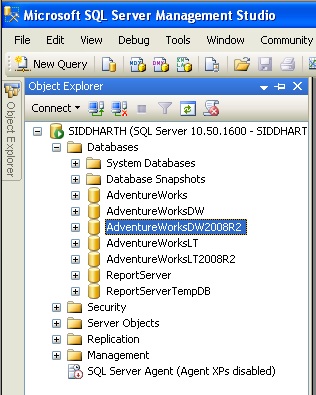
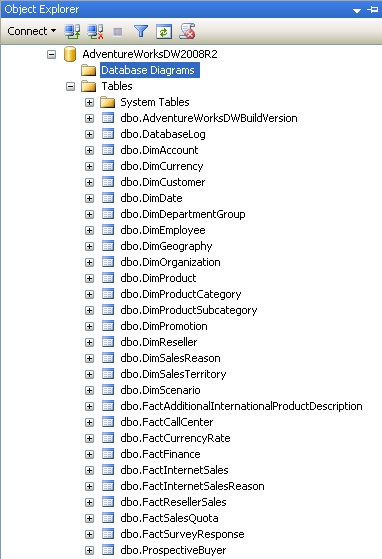
Two major theories driving the design of a data warehouse and data marts are from Ralph Kimball and Bill Inmon which are mostly practiced in real time environments.  Generally data is gathered from OLTP systems and brought to the data warehouse. From the data warehouse, context / requirement specific data marts are created, which can be perceived as a subset of the data warehouse. Cube source data from these data marts, and client applications connect to the cube. The schema for a cube falls into two categories: Star and Snowflake. In simple terms, Star Schema can be considered a more denormalized form of schema compared to Snowflake.  
  
Designing and developing a data warehouse is out scope for this tutorial. For the purpose of development, we will install and use the AdventureWorks DW database. We will then create a SSAS project and create a data source which will connect to this database. Finally we will create a star schema using a Data Source View.

# Installing AdventureWorks Sample Database

## Overview

AdventureWorks is the sample database available from Microsoft for different purposes as well as different SQL Server versions. We need to use the AdventureWorks DW 2008 R2 database for our cube design and development. This database contains dimension and fact tables with prepopulated data. We can use this database as a launchpad to start our SSAS project. Developing a data mart is out of the scope of this tutorial, so we will use this sample database.

## Explanation

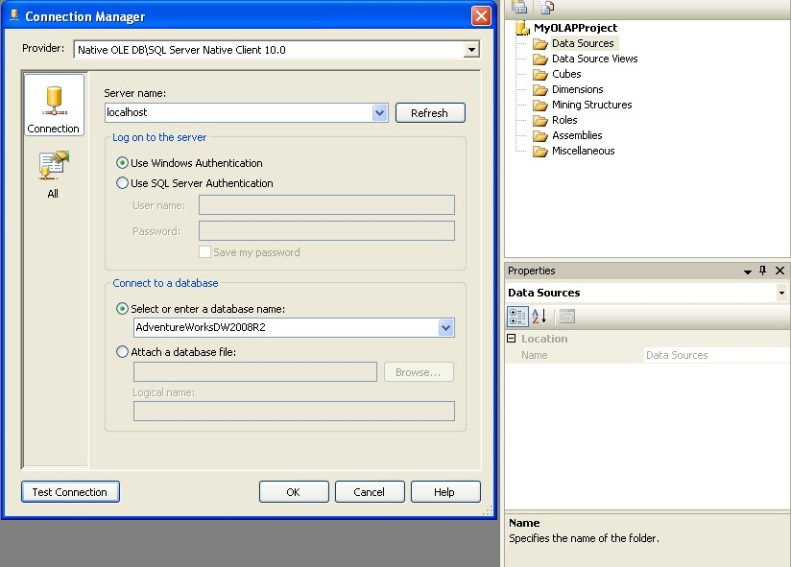
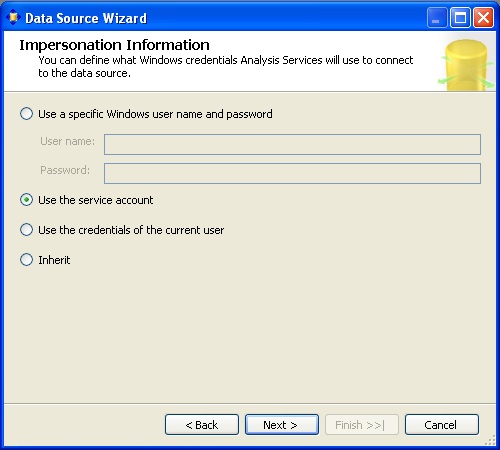
To install the AdventureWorks database, navigate to the codeplex (http://msftdbprodsamples.codeplex.com/) site and download the MSI for the version of SQL Server you are using. This tutorial expects that the reader is using SQL Server 2008 R2, and all the exercises will be using this version of SQL Server.   
  
After downloading, start the installer and you should get a screen similar to the one below.  
  
  
  
**AdventureWorks Data Warehouse 2008R2** is the database we need for our exercises. Point the installer to the SQL Server instance that you are using, and install the database. After the database in installed, open SQL Server Management Studio to verify the databases that were installed.  You should find something similar to the below screenshot.  
  
   
  
Expand the database higlighted above and check out the different Dim and Fact tables in this database. The tables having the prefix Dim are suited to be used as Dimension tables, and tables having prefix Fact are suited to be used as Fact tables.  
  


# Creating a SSAS Project

## Overview

To start development, we need to create a new SSAS project using Business Intelligence Development Studio. After creating the new project, we need to create a data source that points to the AdventureWorks DW 2008 R2 database.

## Explanation

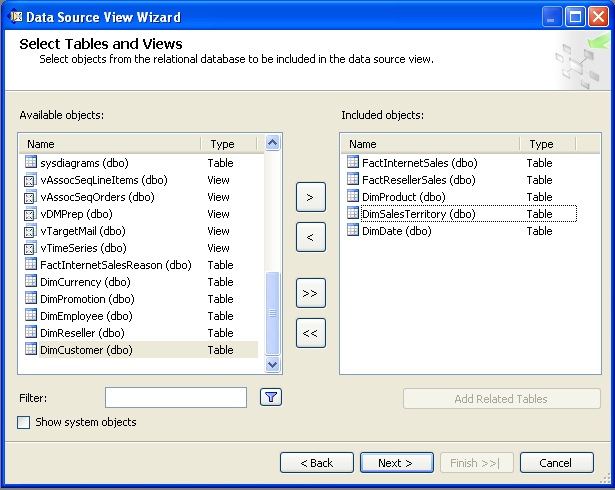
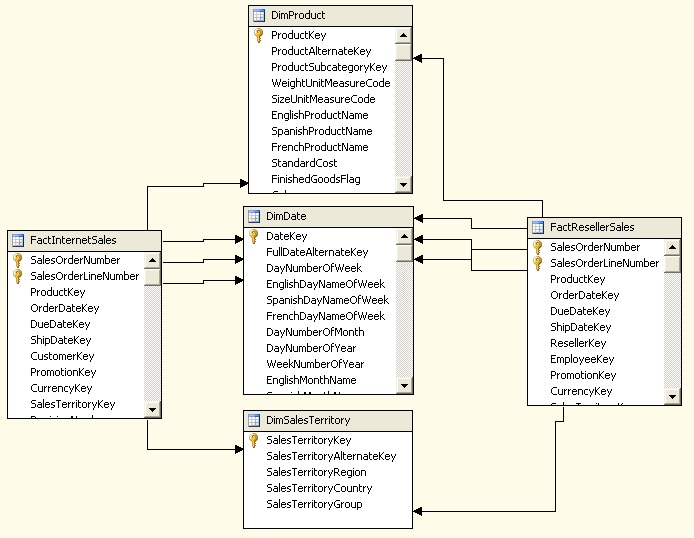
Open Business Intelligence Development Studio (BIDS). Create a new SSAS Project, by selecting New Project from the File menu. Name this project “MyOLAPProject”. As soon as the new project opens up, you should find a list of folders in the explorer tab. Right-click on the data sources folder and select New DataSource. A Data Source wizard will open with a Welcome screen, select Next and you should find a screen to define your connection. We need to define a new connection, so select “New” and a screen should appear as shown below. Point the connection to the **AdventureWorksDW2008R2** database and click OK.   
  
  
  
After this, you need to specify the impersonation information for the data source. This information is used to specify how the solution will connect to the SSAS instance using the credentials specified. Every time you deploy or process the solution, this connection information will be used. So keep in mind that the account you use should have sufficient privileges. If you are not sure which account to use, it is suggested that you use an account with administrator privileges on your development machine. Please keep in mind that this is not recommended and should not be done in production environments. This is just suggested to quickly get you started with cube design and development.  
  
  
  
After specifying this information, click “Next”. This should take you to the final screen where you need to name the data source. Name it something appropriate and click OK, which should create your data source.

# Creating a Star Schema Using a Data Source View

## Overview

A data warehouse or data mart from where we would source our data could contain ten to hundreds of tables. Also one would not have the liberty to change the schema of these tables to suit the requirements of the cube design. The Data Source View is an insulation layer between the actual data source and the solution. We can create and modify the schema we need in this layer and this is used as the data source for the different objects we create in the solution. A Star Schema is a schema structure where different dimension tables are directly connected to the fact table. If you imagine a fact table in the center and different dimensions attached to it, you would find the figure similar to a star and hence the name star schema. It’s the simplest form of the schema and hence we will use this in our exercise.

## Explanation

Right-click on the Data Source View and select New Data Source View and a wizard should pop-up with a Welcome screen. Select “Next”, and the next screen should prompt you to select a relational data source. Select the data source we just created and click “Next”, the next screen should prompt you to select tables that we intend to use in our solution. Select the tables as shown in the below screenshot. The below fact and dimension tables are chosen as they are interlinked with each other and also suits the requirements of the exercises to follow.  
  
  
  
Select “Next”, name the DSV to something appropriate and this should finally create your Data Source View. After arranging the tables in the DSV, your schema should look similar to the below screenshot.  
  
  
  
In the above figure, you can see that both the fact tables are related to all three dimensions in the same manner. This is a typical case of a star schema. You can also browse the data, create calculated fields, assign primary keys and carry out other similar function in this designer to modify the schema without modifying the actual schema in the database.

# Designing a Cube

## Overview

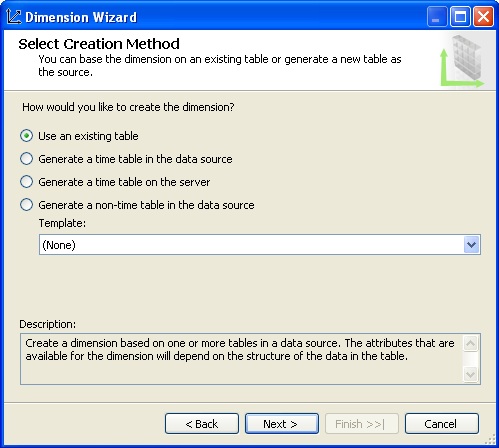
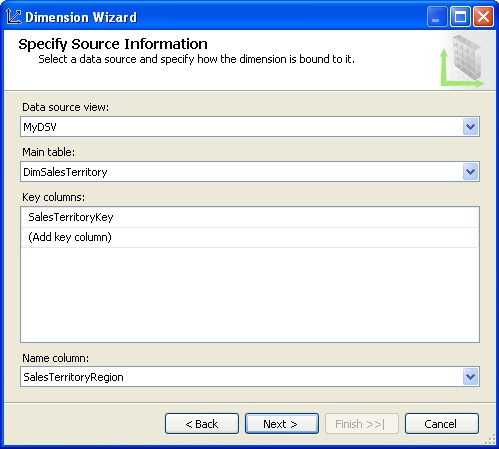
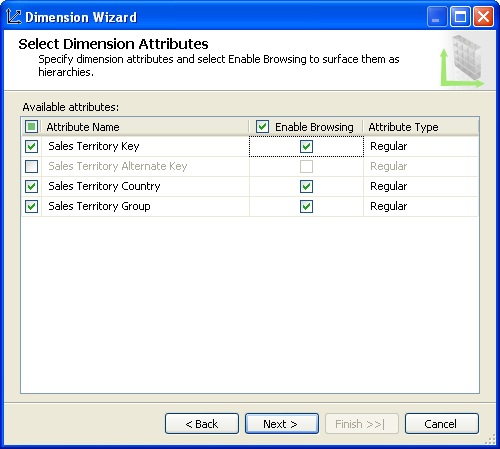
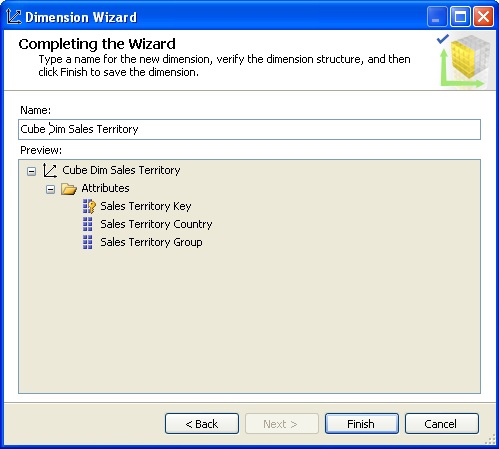
Using BIDS, after the DSV is developed, the next step is to create dimensions. Dimensions are of two types: Database Dimensions and Cube Dimensions. Database dimensions can be perceived as a master template, and Cube dimensions can be perceived as instances / children of this master template.  
  
We will start our development with the creation of database dimensions. If you consider a dimension as a table, all the fields in this table can be perceived as attributes. Hierarchy in a dimension is a group of attributes logically related to each other with a defined cardinality. Finally we will create a cube using the dimensions we just developed, and fact tables to create dimensions (cube dimensions) and measure groups (from fact tables).

# Creating a Dimension

## Overview

Dimensions are of two types: database dimension and cube dimension. The dimensions that are defined at the solution level can be termed as a database dimension and the ones defined inside the cube are termed as a cube dimension. Dimension Wizard is the primary means of creating a dimension. We will create a dimension using the three dimension tables which we have included in our schema.

## Explanation

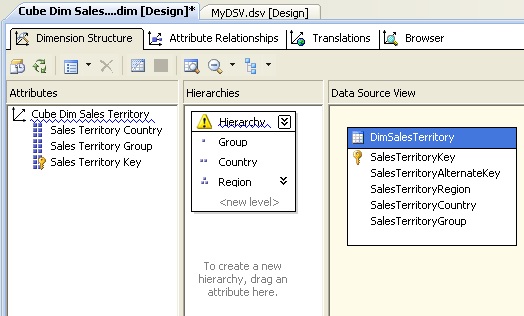
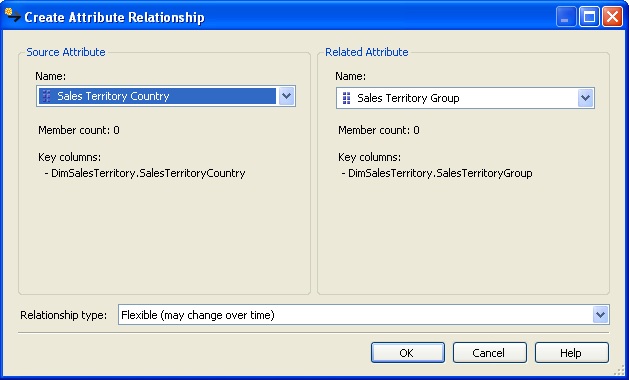
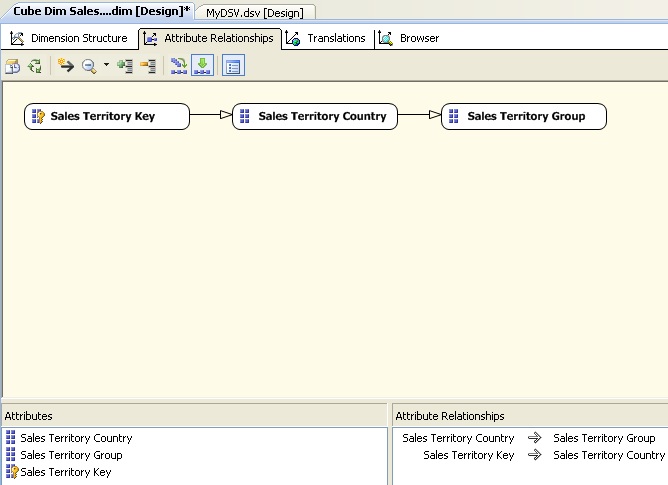
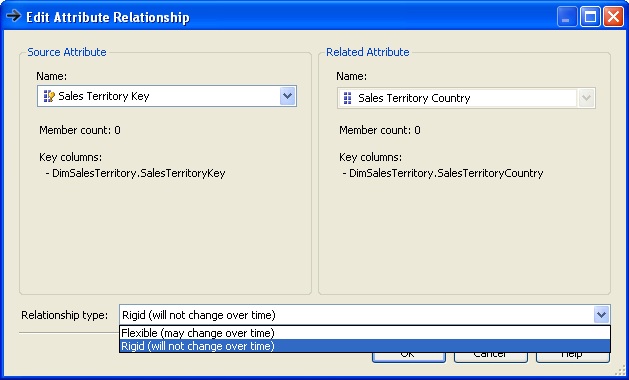
Right-click the Dimensions folder and select “New Dimension”, this will invoke the Dimension Wizard. The first screen should look like the below screenshot. You have the options of using an existing table, creating a table in the data source and using a template. We already have the dimension table in our schema and we will use this, so select “Use an existing table” and click “Next”.  
  
  
  
Select the DSV we created earlier in the DSV selection. We intend to create a dimension from the DimSalesTerritory table, so select the same table. Every dimension table needs to have a key attribute, and in this table SaleTerritoryKey is the primary key column which is guaranteed to identify each record uniquely. It would not make sense to browse this attribute using the Key, instead SalesTerritoryRegion field has unique values. We can also use this field as the key as well as name column. But for the purpose of our exercise, we will use the SaleTerritoryKey field as the key column and SalesTerritoryRegion as the name column. Though it looks inappropriate to use the key field, but when you are starting to develop an understanding of dimensions, this will help to set a rule in your mind that the key field is always required, mostly a surrogate key and you can set a name column to any field to facilitate a convenient browsing mechanism.  
  
  
  
In the next screen, you need to make a selection of the attributes that will be present in the dimension. If you uncheck the “Enable Browsing” button, they won’t be visible to client applications when they browse the dimension. Attributes can be of different types and you can specify the type in the Attribute Type field. The Dimension Wizard removes the Name column you set from the key column as that is available due to the key column. So you won’t find that field in this list of available attributes.  
  
  
  
Now the next step is to give a name to the dimension, name it “Cube Dim Sales Territory” or anything appropriate. After this step you have completed creating your first dimension.  
  
  
  
In a similar manner create Product and Date dimension using the Dimension Wizard.

# Creating a Hierarchy

## Overview

A Hierarchy is a set of logically related attributes with a fixed cardinality. While browsing the data, a hierarchy exposes the top level attribute which can be broken down into lower level attributes. For example, Year -> Semester – Quarter – Month is a hierarchy. While analyzing the data, it might be required to drill down from a higher level to a detail level, and exposing data as a hierarchy is one of the best solutions for this.

## Explanation

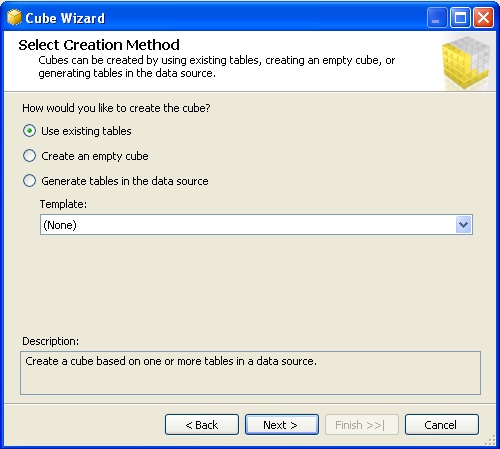
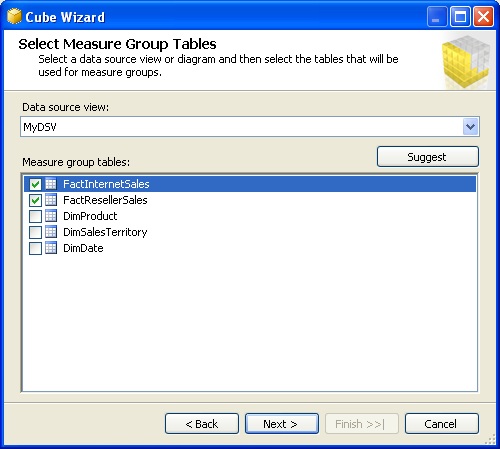
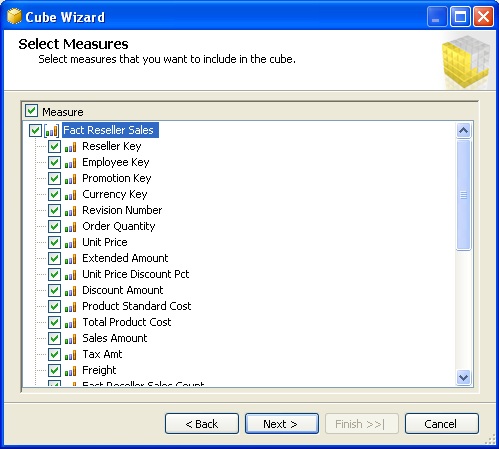
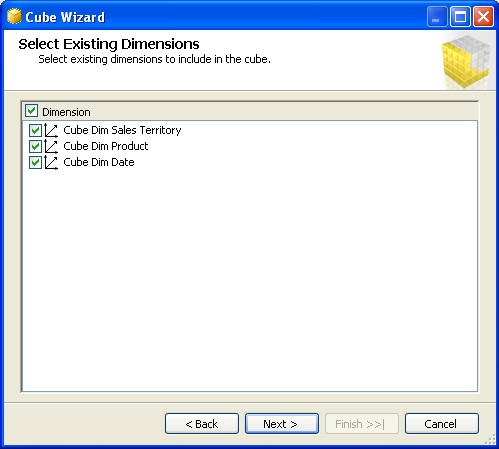
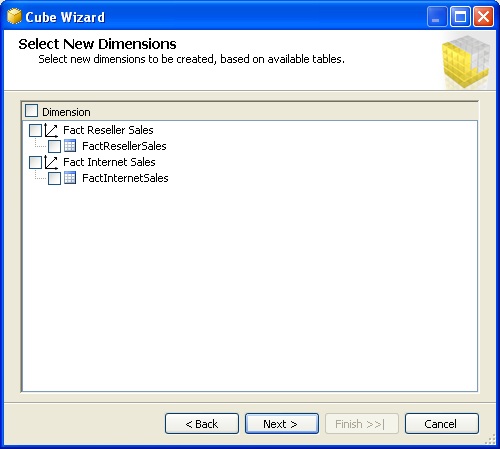
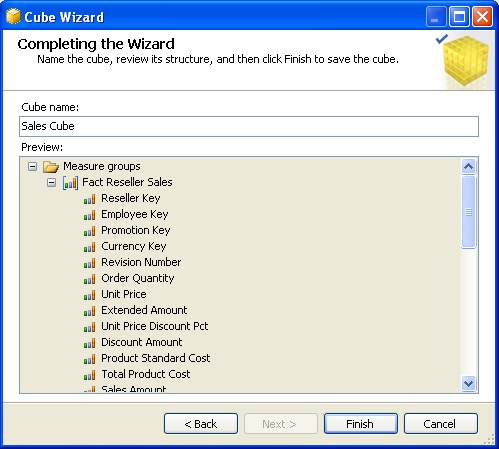
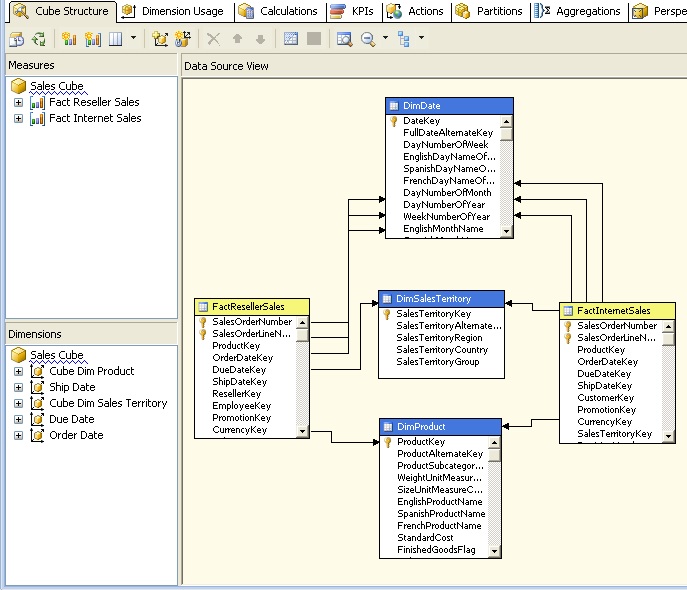
Creating a hierarchy is as easy as dragging and dropping attributes in the hierarchy pane of the dimension editor. We want to create a hierarchy in the Sales Territory dimension. Open Sales Territory dimension in the dimension editor, drag and drop attributes in the hierarchy pane, click on each of them and rename them to something appropriate. After completing this, your hierarchy should look similar to the below screenshot.  
  
  
  
You will find a warning icon on the hierarchy pane, which says that attribute relationships are missing between these attributes. Country has a one-to-many relationship with Region, and Group has a one-to-many relationship with Country. But these relationships need to be defined explicitly in the dimension. Click on Attribute Relationships tab, right-click the region attribute and select “New Attribute Relationship”. Set the values as shown in the below screenshot to correct the relationships between these attributes.  
  
  
  
After you have applied the above changes, your attribute relationship tab should look like the below screenshot.  
  
  
  
If you have observer carefully, relationship types are of two types: Rigid and Flexible. This has an effect on the processing of the cube. Rigid means that you do not expect the relationship to change and Flexible means that relationship values can change. In our dataset, Group is a logical way to categorize countries and it can change, while regions within country have limited or no change. So the relationship type between country and group should be flexible and relationship type between region (sales territory key) and country should be rigid. Double click on the arrow joining Key attribute and Country, and change the relationship type as shown below.  
  
  
  
Check out the Hierarchy pane, and you should find that the warning icon is no longer visible. You can change the name of the hierarchy to something appropriate. In the interest of beginners who might get confused with the distinction between attributes and hierarchy, we will keep the name as “Hierarchy”.  
  
Edit the Date dimension, and create a Year – Semester – Quarter – Month hierarchy in the date dimension.

# Creating a Cube using the Cube Wizard

## Overview

A Cube acts as an OLAP database to the subscribers who need to query data from an OLAP data store. A Cube is the main object of a SSAS solution where the majority of fine tuning, calculations, aggregation design, storage design, defining relationship and a lot of other configurations are developed. We will create a cube using our dimension and fact tables.

## Explanation

Right-click the Cube folder and select “New Cube”, and it will invoke the Cube Wizard. In the first screen you need to select one of the methods of creating a Cube. We already have our dimensions ready, and schema is already designed to contain dimension and fact tables. So we will select the option of “Use existing tables”.  
  
  
  
In the next screen, we need to select the tables which will be used to create measure groups. We already have a DSV which has fact tables in the schema. So we will use this as shown in the below screenshot.  
  
  
  
In the next screen, we need to select the measures that we want to create from the fact tables we just selected in the previous screen. For now, select all the fields as shown below and move to the next screen.  
  
  
  
In this screen you need to select any existing dimensions. We have created three dimensions and we will include all of these dimensions as shown below.  
  
  
  
In the next screen, we can select if we want to create any additional new dimensions from the tables available in the DSV. We do not want to create any more dimensions, so unselect any selected tables as shown below and move to the next screen.  
  
  
  
Finally you need to name your cube, which is the last step of the wizard before your cube is created. Name it something appropriate like “Sales Cube” as shown below.  
  
  
  
Now your cube should have been created and if your cube editor is open you should find different tabs to configure and design various features and aspects of the cube. If you look carefully in the below screenshot, you will find FactInternetSales and FactResellerSales measure groups. Also you will find Sales Territory and Product dimension, but Date dimension is missing. Both fact tables have multiple fields referencing the DateKey from the Date dimension. BIDS intelligently creates three dimensions from the Date dimension and names them to the name of the field which is referenced from the Date dimension. So you will find three compounds of Date dimension – Ship Date, Due Date and Order Date dimensions. These are known as role-playing dimensions.  
  


# Processing and Deploying a Cube

## Overview

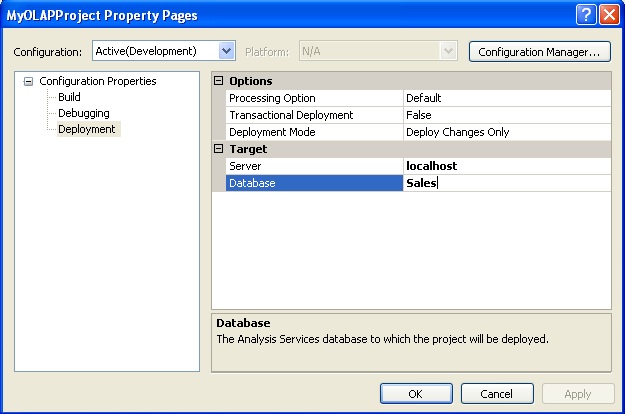
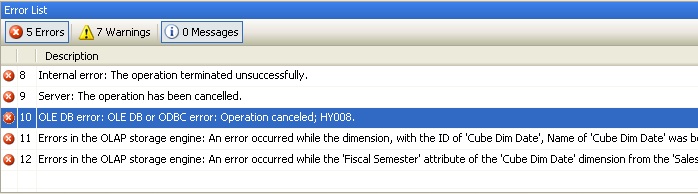
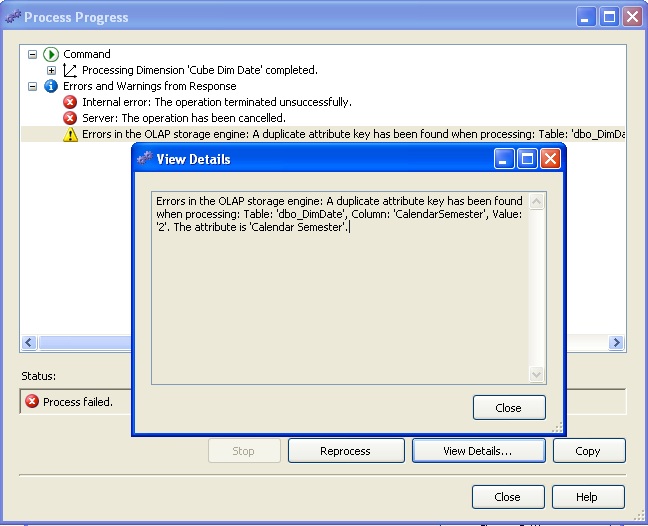
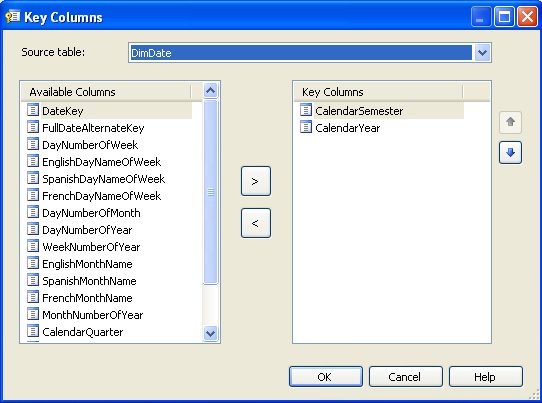
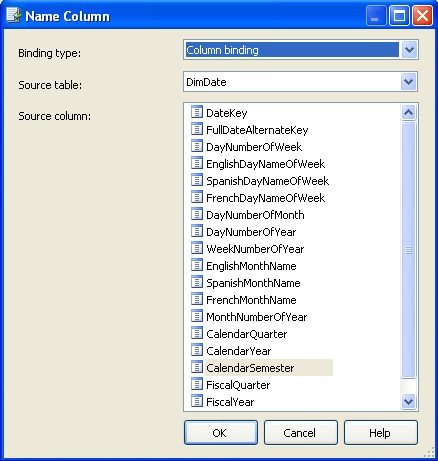
Once the cube design and development is complete, the next step is to deploy the cube. When the cube is deployed, a database for the solution is created in the SSAS instance, if not already present. Each of the dimensions and measure group definitions are read, and data is calculated and stored as per the design and configuration of these objects. Once the cube is successfully deployed, client applications can connect to the cube and browse the cube data. We will deploy the cube we have developed and test connecting to the cube. We might also face errors during deployment, and we will attempt debugging and resolving these errors.

# Debugging Deployment Errors

## Overview

In a development environment, ideally you would come across errors during deployment and processing of the cube. Debugging errors is an essential part of the cube development life cycle. We will configure the deployment properties and we should face some errors during the deployment. We will then analyze and resolve these errors.

## Explanation

Right-click the solution and select Properties, this would bring up a pop-up window. Select the deployment tab and it will bring up the deployment properties. Mention the SSAS server name and the database name that was created for your solution in the SSAS instance. Since SSAS in installed on my local / development machine, I have chosen server as “localhost” and name of the database as “Sales”. We will keep the rest of the options as default for now.  
  
  
  
Right-click the solution and select “Deploy”, this will start deploying the solution. If you have not specified an appropriate account in the impersonation information, your deployment might fail as the account might not have sufficient privileges.  
  
If you have followed all the previous steps as explained, you should face errors as shown below. From the error message you can make out that cube processing failed due to the Date dimension.  
  
Right-click the Cube Dim Date dimension and select “Process”, and you would find the following error.  
  
  
  
If you recall we have defined a hierarchy in the Date dimension, Year -> Semester -> Quarter -> Month, and the attribute relation expected is one to many. If you browse the data, you will find that the same set of semester values exist in each year, so how do you make them unique for each Quarter? When the Quarter is processed, it will find duplicate Semester as the key columns for the Semester is Semester itself by default which is not unique. So we need to make each attribute unique by changing its key columns.  
  
  
  
Edit the Date dimension in the dimension editor, select the Semester attribute and edit the Key Columns property. This should bring up a pop-up window as shown below. To make the Semester attribute unique, we need to make the key column a composite key Year + Semester to make it unique. So select key columns as shown below.  
  
  
  
When you select multiple columns in the key column, the name column property becomes blank and it’s a mandatory property. So select this property and set it again to Semester as we want to display semesters when this is browsed.  
  
  
  
This should solve the error we were facing on the date dimension. Duplicate keys are one of the most common errors during dimension processing and we just learned how to resolve this issue.

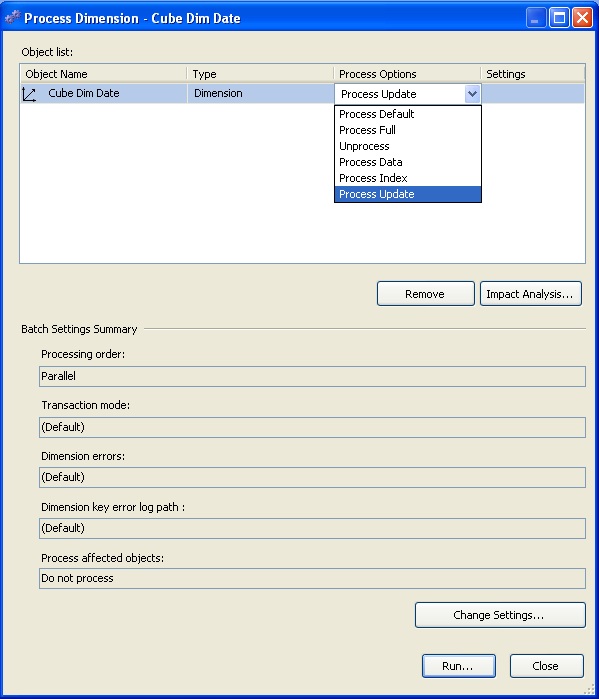
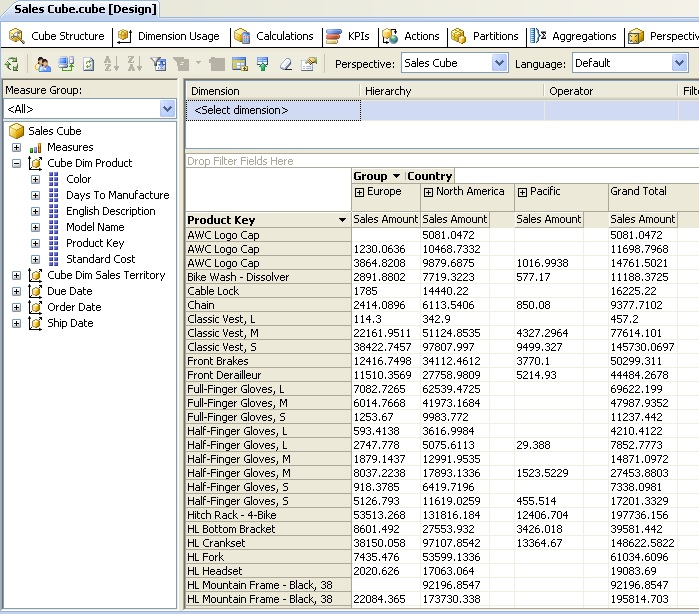
# Processing Dimensions and Cube

## Overview

SSAS provides various cube processing methods and options to configure error logging as well as impact on processing when errors are encountered. We will briefly look at these options, understand what processing of the cube means, deploy our cube and try to access data from the cube.

## Explanation

Right-click on the dimension or cube and select “Process”, and this should bring up a similar screen with processing options as shown in the below screenshot. Various processing options are visible in the dropdown. Unprocess would remove all the aggregation created by the processing of the object. Process Full would also do the same operation, but also create all the aggregations again. More reference about these options can be found in MSDN BOL.

In the "Change Settings" and "Impact Analysis" options you will find more error configuration and other options related to processing.  
  
  
  
Deploy the cube and the cube should be deployed successfully. Go to the Browser pane after successful deployment, and try to connect to the cube and browse data by dragging and dropping dimension attributes and measures on the browsing area. Below is an example.  
  


# Calculated Measures and Named Sets

## Overview

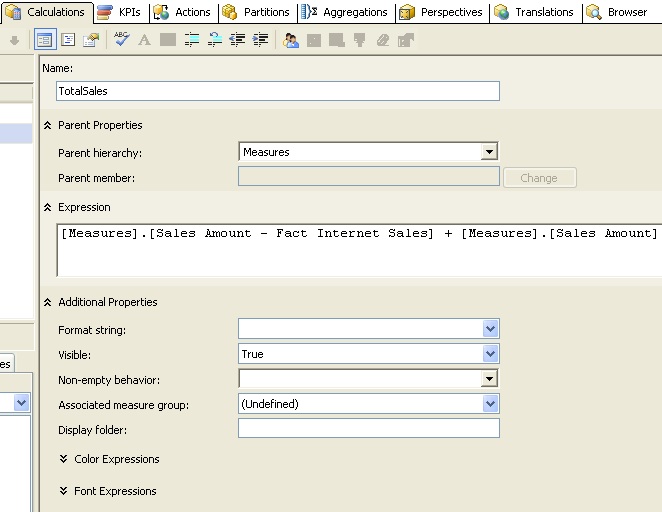
Fields from fact tables get converted into measures in measuregroups in a cube. When measuregroups are created in a cube, one measuregroup is created per fact table. Often in production systems, developing calculated measures is a regular requirement. Multi-Dimensional Expressions (MDX) is the query language for a cube and is synonymous to what T-SQL is to SQL Server. Often queries that are frequently used are required to be in some ready format in a cube, so that the users do not need to develop them over and over again. One of the solutions for this is named sets, which can be perceived as a query already defined in the cube, similar to views in SQL Server. We will develop a calculated measure and a few named sets in this section.

# Developing a Calculated Measure

## Overview

Measures created directly from the fields of a fact table are called base measures. But often we require measures based on custom requirements, so we apply some logic and/or formula to these base measures and create calculated measures. We will add two measures from two measure groups and create a calculated measure.

## Explanation

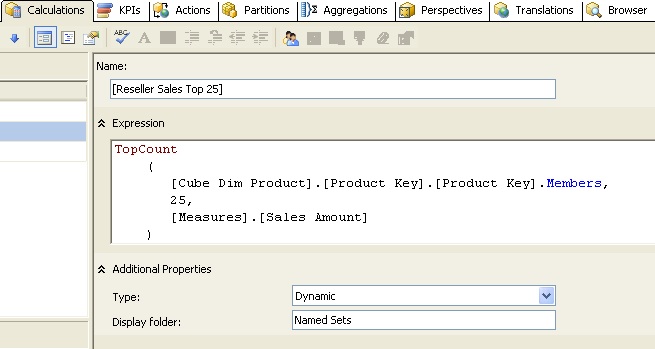
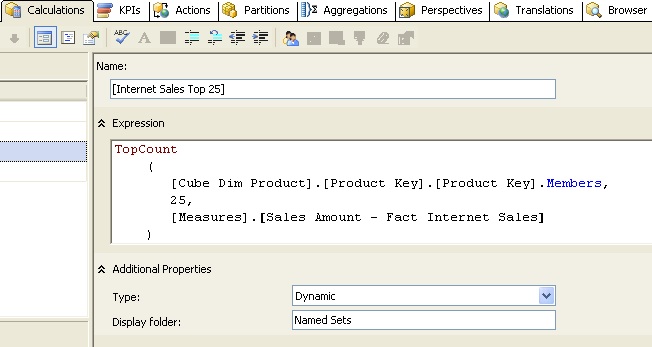
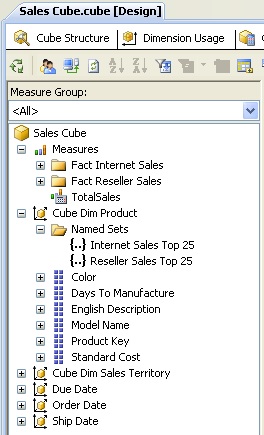
Open the cube designer, and click on the Calculations tab. Click on “New Calculated Measure” from the toolbar, and key in the values as shown in the below screenshot.  
  
  
  
We have named this new calculated measure “TotalSales”. The "Parent hierarchy" specifies which parent hierarchy the measure will be part and in this case it will be “Measures”. It’s a built-in hierarchy and all measures normally fall under this.  
  
In the Expression, we can specify any MDX expression. Here we are adding Internet Sales Amount from FactInternetSales and Reseller Sales Amount from FactResellerSales measure groups. You do not need to type the values you can just drag and drop values from the panes on the left-hand side of the window.  
  
In the additional properties you can set additional options for this measure. Save your solution, in the next section we will create named sets and then deploy these at the same time.

# Developing Named Sets

## Overview

Named sets return a dataset based on defined logic. They are primarily useful to create datasets that are often requested from the cube. Named sets are of two types: Static and Dynamic. The difference between these two is that static named sets are calculated when they are requested the first time in a session and dynamic named sets are calculated each time a query references it. In this section we will look at how to create dynamic named sets. Note that dynamic named sets were not introduced until SQL Server 2008.

## Explanation

Open the cube designer, and click on the Calculations tab. Click on “New Named Set” from the toolbar and key in the values as shown in the below screenshots.  
  
  
  
  
  
Here we are creating two named sets, Internet Sales Top 25 and Reseller Sales Top 25. In these named sets, we are returning the Top 25 products based on Internet Sales and Reseller Sales. In this formula, TopCount, the MDX function returns top 25 records from the dataset.  
  
In the Type selection, we can select whether we want the named set to be static or dynamic. We have selected Dynamic as we want to create a dynamic named set.  
  
In the Display folder selection, we can specify where the named sets will appear. By default named sets appear in the last dimension that is used in the formula. Here we have used an attribute hierarchy from Product dimension, so the named sets should appear in the same dimension under “Named Sets” directory.  
  
Save and deploy the solution, and then re-connect to the cube in the “Browser” pane. You should be able to see the calculated measure and named sets as shown in the below screenshot.  
  


# Browsing a Cube Using Excel

## Overview

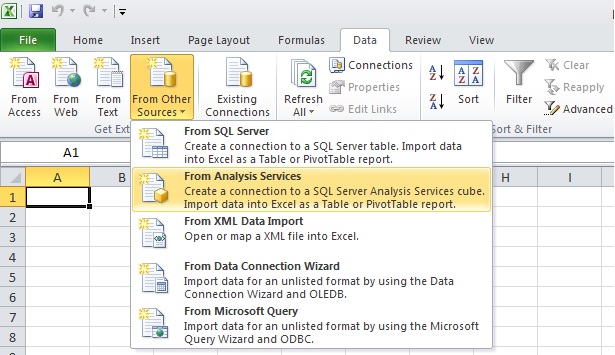
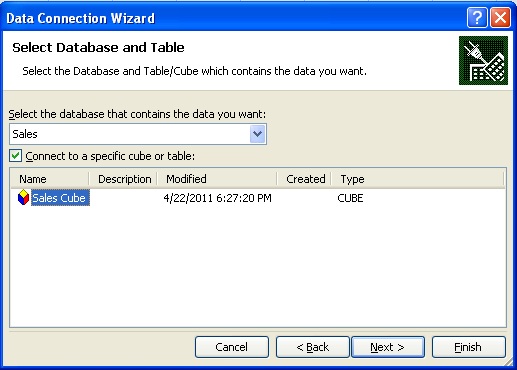
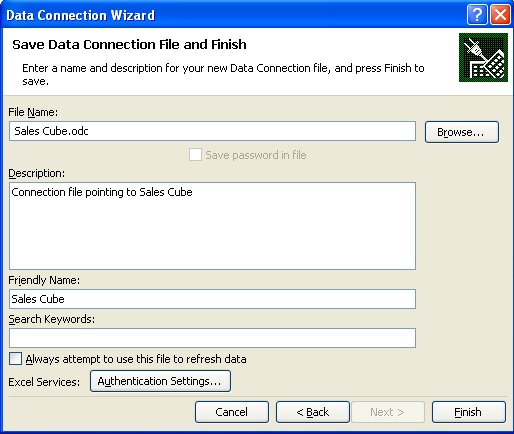
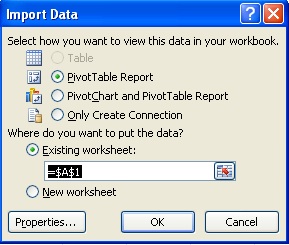
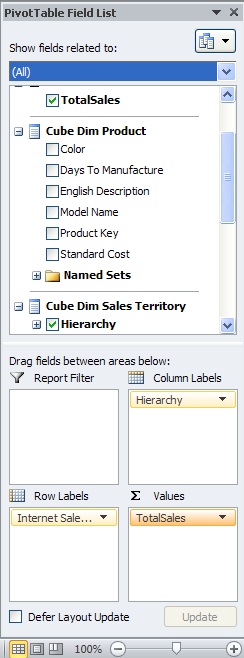
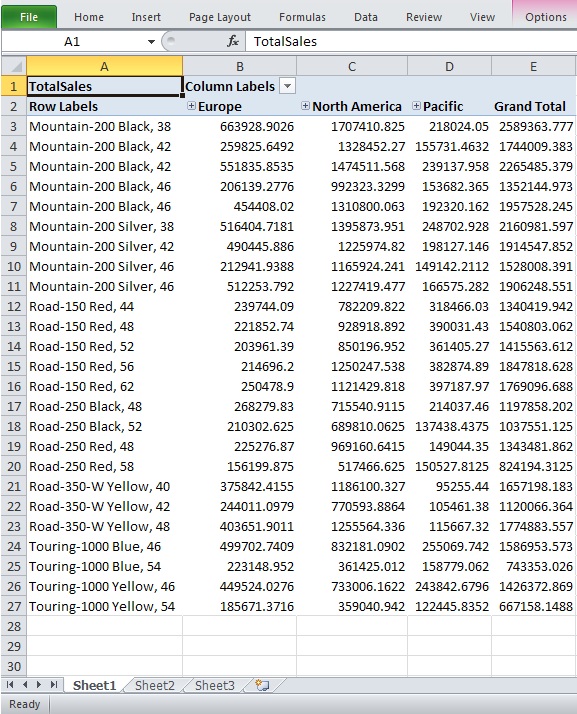
Once the cube is deployed and ready to host queries from the data store, client applications can start querying the cube. One of the most user friendly client tools for business users to query a cube is Microsoft Excel. It has a built-in interface and components to support GUI based connection, querying and formatting of data sourced from a cube. Business users can use the familiar interface of Excel and create ad-hoc pivot table reports by querying the cube without any detailed knowledge about querying a multi-dimensional data source. We will connect to the cube we just created using Excel and develop a very simple report using the cube data.

# Using Excel and Creating a Pivot Table Report

## Overview

We will first create a connection to the cube we have developed in the previous exercises. After connecting the cube we will use the calculated measures and a named set to create a very basic pivot table report. For the purpose of demonstration, Excel 2010 is used and is installed on the development machine, but you can also use Excel 2007 to connect to the cube.

## Explanation

Open Microsoft Excel and select the “Data” tab from the menu ribbon. Click on “From Other Sources” and select “From Analysis Services” option as shown in the below screenshot.  
  
  
  
In the next step specify the SSAS server name and logon credentials. If you have everything on the local machine, you can also use “localhost” as the server name.  
  
  
  
If you were able to successfully connect to the specified SSAS instance with the logon credentials specified, in the next step you should be able to select the SSAS “Sales” database and find the Sales Cube. Select the Sales Cube and proceed to the next step.  
  
  
  
In the next step, specify the name of the connection file to save. This file will be saved as an .ODC file and you can reuse this connection file when you want to use the same connection in other workbooks.  
  
  
  
After saving the file, you will be prompted with the option to select the kind of report you want to create. We will go with the default option and select “PivotTable Report”.  
  
  
  
After selecting “PivotTable Report”, a designer will open with options to select dimension, attributes and measures to populate your pivot table. Select the values as shown in the below screenshot. Our intention is to display the hierarchy we created in the Sales Territory dimension on the columns axis, Internet Sales Top 25 named set on the rows axis, and the Total Sales calculated measure in the values area.  
  
  
  
After making the above selections, your report should look like the below screenshot. Using the features available from the “Options” tab, you can format this report and give it a more professional look. You can try drilling down the hierarchy, but you will see that you need to develop the hierarchies. Users who frequently want to see sales of products to top customers, can pick up any named-set that we defined earlier. Instead of having users define formulas for adding internet sales and reseller sales, users can just select Total Sales.  
  


# SQL Server Analysis Services Glossary

Following is a list of common terms when working with SQL Server Analysis Services.

**Cube** - Cube is a multi dimensional data structure composed of dimensions and measure groups. The intersection of dimension and measure groups contained in a cube returns the dataset.

**Calculated Measure** - Each field in a measure group is known as a base measure. Measures created using MDX expressions with/without base measures are known as calculated measures.

**Data Source View** - It's an insulation layer that inherits the basic schema from the data source with the flexibility to manipulate the schema in this layer without modifying the actual schema in the data source.

**Dimension** - Dimension is an OLAP structure that is basically used to contain attributes related to an entity to categorize data on the row / column axis. A dimension almost never contains measurable numeric data, and if at all it contains, it is used as an attribute. Typical example of dimensions are Geography, Organization, Employee, Time etc.

**Fact** - Fact known as a Measure Group in a cube, is an OLAP structure that is basically used to contain measureable numeric data, for one or more entities. In cube parlance these entities are known as Dimensions. A dimension need not be necessarily associated directly with a fact, but a fact is always associated directly with at least one dimension. Typical example of facts are Sales, Performance, Tax etc.

**Hierarchy** - Hierarchy is collection of nested attributes associated in a parent-child fashion with a defined cardinality. Dimension is formed of attributes, and hierarchy contained in a dimension is formed of one or more attributes from the same dimension.

**KPI** - Key Performance Indicators are logical structures defined using MDX expressions. Each KPI has a goal, status, value, trend, and indicator associated with it. Value is derived based on the definition of KPI, all the rest of these values vary based on this derived value. KPIs are the primary elements that makes up a scorecard in a dashboard.

**MDX** - Multi Dimensional Expressions is considered as the query language of multi dimensional data structures. This can be considered as the SQL of OLAP databases, with the major difference that MDX is mostly used for reading data only.

**Named Set** - Named Set is a pre-defined MDX query defined in the script of the cube. It can be thought of synonymous to Views in a SQL Server database. Named sets can be dynamic or static and this nature defines the time when this query gets evaluated.

**OLAP** - Online Analytical Processing is a term used to represent analytical data sources and analysis systems. The fundamental perception and expectation associated with the term OLAP is that it would contain multi dimensional data and the environment hosting the same.

**Snowflake Schema** - Snowflake schema is an OLAP schema, where one or more normalized dimension tables are associated with a fact table. For example, Product Sub Category -> Product Category -> Product can be three normalized dimension tables and Product table can be associated with a fact table like Sales. This is a very common example of a snowflake schema.

**Star Schema** - Star schema is an OLAP schema, where all dimension tables are directly associated with fact tables, and no normalized dimension tables are considered in the schema. For example, Time, Product, Geography dimension tables would be directly associated with a fact table like Sales. This is a very common example of star schema.

# Adding a KPI to an SQL Server Analysis Services Cube

Key Performance Indicators, which vary according to the application, are widely used as a measure of the performance of parts of an organisation. Analysis Services makes this KPI data easily available to your cube. All you have to do is to follow Rob Sheldon's simple instructions.

In SQL Server Analysis Services (SSAS), you can add key performance indicators (KPIs) to your database cube in order to evaluate business performance, as reflected in the cube data. A KPI is associated with a measure group and is made up of a set of calculations. Typically, the calculations are a combination of calculated members and Multidimensional Expressions (MDX) statements.

A KPI consists of four main properties that are important to evaluating business performance:

* **Value Expression.** An MDX expression that returns the KPI’s actual value.
* **Goal Expression.** An MDX expression that returns the KPI’s target value.
* **Status Expression.** An MDX expression that returns the KPI’s state at a specific point in time.
* **Trend Expression.** An MDX expression that returns the KPI’s value over time.

In addition to these components, there are other properties that you can configure, but these four components make up the heart of your KPI.

As you work through the process of creating a KPI, you’ll get a better sense of what each of these properties means and how they relate to one another. In this article, I show you how to add a KPI to a cube in an Analysis Services 2008 database. The example I demonstrate is based on the solution from an earlier article I wrote, Five Basic Steps for Implementing an Analysis Services Database”. For that solution, I created the following database components:

1. A data source that points to the AdventureWorksDW2008 database on a local instance of SQL Server 2008.
2. A data source view that includes the tables shown in Figure 1.
3. Database dimensions based on each dimension table in the data source view.
4. A cube based on the database dimensions as well as on the two fact tables in the data source view.

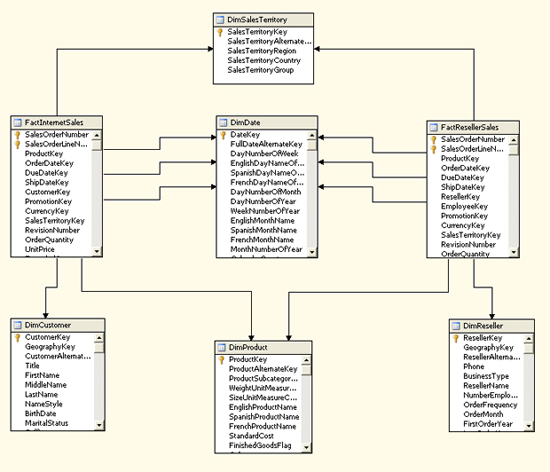


Figure 1: Default diagram from the Sales data source view

Be sure to refer to the article mentioned above for more details about the solution. In addition, if you don’t know how to implement a basic cube in Analysis Services, read that article first and refer to SQL Server Books Online for additional information. Once you know how to implement an Analysis Services database, you’re ready to add KPIs to your solution.

## Creating a Calculated Member

When you create a KPI, you base one or more of your expressions on members in a measure group or dimension. However, in some cases, the existing members don’t support the type of KPI you want to create, at least not in their current form. If that’s the case, you can create a calculated member, which is similar to creating a computed column in a SQL Server database.

To create a calculated member, open your Analysis Services project in SQL Server Business Intelligence Development Studio (BIDS), and then open the cube in which you want to create your KPI. (For this article, I’m adding the KPI to the Sales cube.) In Cube Designer, click the **Calculations** tab, and then click the **New Calculated Member** button. A new calculation form opens in the right pane, as shown in Figure 2.

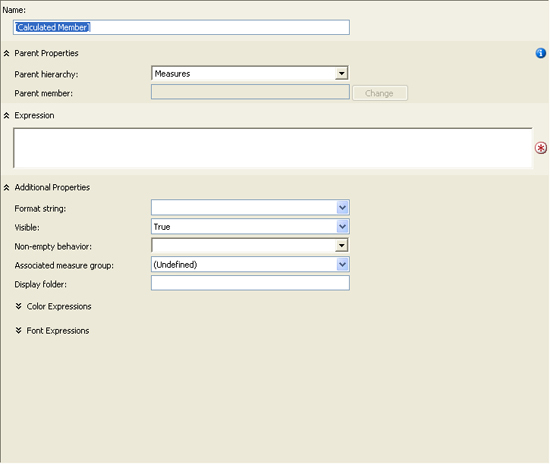


Figure 2: Form for a new calculated member

You should first name the calculated member by typing the name in the **Name** text box. For this example, I use the following name:

[Profit Margin]

Notice that I enclose the name in brackets. If your name includes a space, as mine does, you must use the brackets.

Next, you should verify the setting for the **Parent hierarchy** property. By default, the property is set to **Measures**. Because we’re creating a calculated measure, this is the hierarchy we want to use.

After you select the hierarchy, you must define an MDX expression that determines the value for your calculated measure. In this case, I want to create a measure that provides the profit margin for each sale. The following expression calculates the margin by dividing the net profit by the sales amount:

|  |  |
| --- | --- |
|  | ([Measures].[Sales Amount] -    ([Measures].[Total Product Cost] + [Measures].[Tax Amt] +      [Measures].[Freight])) /    [Measures].[Sales Amount] |

Notice that I calculate the net profit by subtracting the total product cost, tax amount, and freight from the sales amount. (You might decide on a different formula for you net profit.) I then divide that total by the sales amount.

**Note:** When opening the **Calculations** tab, you probably noticed the list of measure groups and dimensions in the lower-left pane. You can drag a member from any of these hierarchies to your expression text box to add the *fully qualified name* of that member to the expression. Also note, the MDX expression shown above is a relatively simple one. You can, of course, create far more complex expressions. However, an in-depth discussion about MDX is beyond the scope of this article. For more information about MDX, see SQL Server Books Online.

After you create your expression for the calculated member, you can set additional properties. For this example, I set the **Format string** property to **“Percent”** and then select **Fact Internet Sales** in the **Associated Measure Group** property because I want to associate the calculated member with that measure group. Figure 3 shows what the form should look like after you’ve configured all the properties.

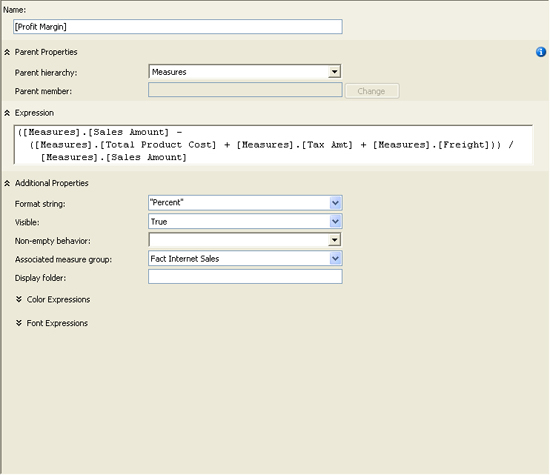


Figure 3: Creating the Profit Margin calculated member

That’s all there is to creating a calculated member. Be sure to save the project and then process the cube so the measure is available to your KPI. After you process your cube, you can verify that the measure has been successfully added by browsing the cube data and viewing the Profit Margin measure.

## Creating a Key Performance Indicator

Now that your calculated measure is set up, you’re ready to create your KPI. In Cube Designer, click the **KPIs** tab, and then click the **New KPI** button. A new KPI form opens in the right pane, as shown in Figure 4.

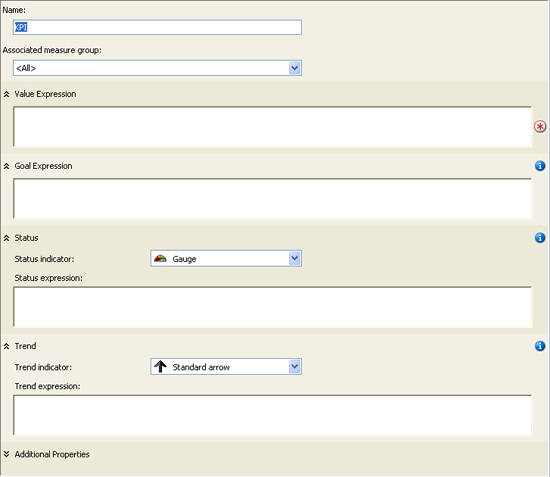


Figure 4: Form for a new key performance indicator (KPI)

To configure the KPI, first provide a name. (I use *Gross Profit* for our example KPI.) Then select a value for the **Associated measure group** property. (I use **Fact Internet Sales**.) You’re now ready to add the necessary expressions to your KPI.

### Adding the Value Expression

Your value expression should reflect the basic measure by which your KPI is gauged. The value returned by the expression serves as the foundation for your KPI. It’s the only one of the four expressions that’s required. For our example KPI, I use the following expression:

|  |  |
| --- | --- |
|  | [Measures].[Profit Margin] |

As you can see, I’m simply calling the Profit Margin calculated measure. Notice that, as you saw when creating your calculated member, I’m using the fully qualified member name. Also, as with the calculated member, you can drag the name from the hierarchies listed in the lower-left pane to the expression text box.

### Adding the Goal Expression

As the name suggestions, the goal expression indicates what your organization is trying to achieve. For example, your profit margin might currently be at 25%, but your goal might be to reach 30%. And you can also set your goal to match more specific criteria. For instance, in the example KPI, I set the goal to vary depending on the specific sales territory group, as shown in the follow MDX expression:

|  |  |
| --- | --- |
|  | Case    When [Territory].[Sales Territory Group]      Is [Territory].[Sales Territory Group].[Europe]        Then .34    When [Territory].[Sales Territory Group]      Is [Territory].[Sales Territory Group].[North America]        Then .36    When [Territory].[Sales Territory Group]      Is [Territory].[Sales Territory Group].[Pacific]        Then .32    Else .30  End |

Notice that I use a Case statement to define my criteria. The Case statement includes three When expressions, one for each territory group. Each When expression identifies the member on which to base the expression and the member value, following the Is keyword. This is followed by a Then expression which defines what action to take. For example, the first When expression states that if the sales territory group is Europe, then the profit margin goal is 34%. However, the goal for the North American group is 36%, and the goal for the Pacific group is 32%. The Else clause then specifies that all other groups have a 30% goal.

### Adding the Status Expression

Your status expression determines the current status of the KPI by comparing the goal expression to the value expression. For example, if your KPI value returns a 20% profit margin, but your goal is 30%, the status will indicate that you are below your goal. However, to arrive at the status, your status expression must return a value in the range of -1 to +1, where -1 indicates bad performance and +1 indicates good performance. For our example KPI, I use the following MDX expression to determine the status of performance:

|  |  |
| --- | --- |
|  | Case    When KpiValue("Gross Profit") / KpiGoal("Gross Profit") > .90      Then 1    When KpiValue("Gross Profit") / KpiGoal("Gross Profit") <= .90      And         KpiValue("Gross Profit") / KpiGoal("Gross Profit") > .80      Then 0    Else -1  End |

Once again, I create a Case statement. In the first When expression, I divide the KPI value by the KPI goal and compare it to .90. If the value is greater than 90% of the goal, I assign the status value a +1. However, in the second When expression, I specify that if the KPI value is less than or equal to 90% *and* is greater than 80%, the status should be 0. Otherwise, the KPI status should be -1.

Notice that I use the KpiValue function to retrieve the KPI’s value and I use the KpiGoal function to retrieve the KPI’s goal. These functions make it easy to retrieve the value and goal within your MDX expressions.

One other thing I want to point out is the status indicator. For this example, I use the traffic light. That means, when the status value is 1, the traffic light will be green. If the status value is 0, the light will be yellow. Otherwise, the light will be red. For example, if my KPI goal is 30% and my value is 20%, the traffic light will be red. That’s because a 20% profit margin represents only about 67% my goal, which would evaluate to a -1 in the status expression. However, a value of 25% would evaluate to about 83%, which would mean a yellow light, and a value of 28% would evaluate to about 93%, which would result in a green light.

After you add the status expression and set up the status indicator, your KPI form should look similar to the one shown in Figure 5. (You have to scroll down to see the trend expression, which we’ll work on next.)

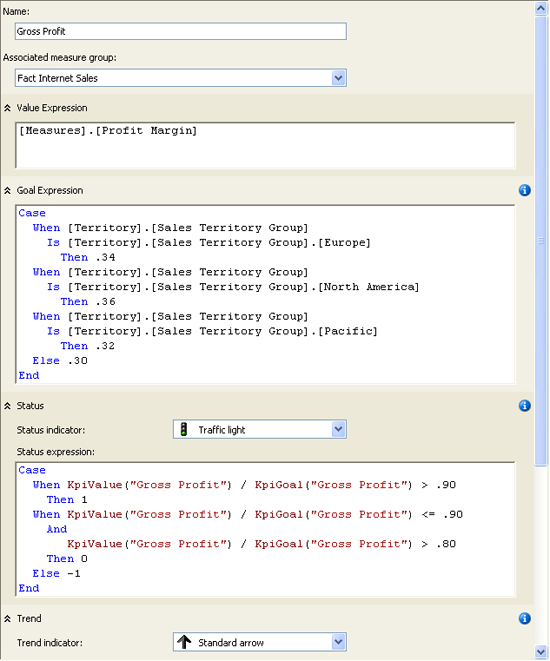


Figure 5: Creating the Gross Profit KPI

### Adding the Trend Expression

The trend expression lets you compare how your KPI is performing over time. Although the status value can tell you how well you’re achieving you goal at a fixed point in time, it doesn’t tell you how those achievements compare to another point in time. For example, your organization might have reached 92% of its goal this year, but reached 95% the year before. Although the performance looks good when just viewing this year’s total, that performance measure doesn’t reveal that this is actually a downward trend. And this is what the trend expression allows you to find out.

For the Gross Profit example KPI, I use a trend expression that compares this year’s totals to the previous year, as shown in the following Case statement:

|  |  |
| --- | --- |
|  | Case    When IsEmpty(ParallelPeriod([Order Date].[Calendar Year].[Calendar Year],        1, [Order Date].[Calendar Year]))      Then 0    When [Measures].[Profit Margin] >      (ParallelPeriod([Order Date].[Calendar Year].[Calendar Year],        1, [Order Date].[Calendar Year]), [Measures].[Profit Margin])      Then 1    When [Measures].[Profit Margin] =      (ParallelPeriod([Order Date].[Calendar Year].[Calendar Year],        1, [Order Date].[Calendar Year]), [Measures].[Profit Margin])      Then 0    Else -1  End |

Notice that the first When expression includes the IsEmpty function and the ParallelPeriod function. The IsEmpty returns a value of true if the evaluated expression an empty cell value. Otherwise the function returns false. I use this function in the first When expression to determine whether the year preceding the current one exists in the cube data. If not, then the trend expression returns a 0, which indicates a neutral trend. (A trend expression, like the status expression, should return a value from -1 to +1.)

The ParallelPeriod function returns a member value from a previous period that’s in the same relative position as the current member. Because I specify Calendar Year, the previous member will also be based on Calendar Year. The function includes three arguments. The first argument returns the level of the hierarchy that you want to target. In this case, it’s the Calendar Year level of the Calendar Year hierarchy, which is part of the Order Date hierarchy. The second argument, 1, indicates the number of units that you want to go back. Because Calendar Year is the current position, the function will go back one year. The final argument specifies that I am basing the calculation on Calendar year. As a result, the first When expression in the Case statement above will determine if the previous calendar year exists, and if it doesn’t the trend will be set to 0.

The second When expression determines whether the profit margin for the current year is greater than the profit margin for the previous year. Notice that the third argument in the ParallelPeriod function now specifies Profit Margin, rather than Calendar Year. If the profit margin is greater in the current year, the trend value is set to +1. However, the third When expression determines whether the profit margin for the current year *equals* that of the previous year. If so, the trend value is set to 0. Otherwise the trend value is set to -1.

Notice also that you can set the trend indicator. You have three different types of arrows from which to choose. Or you can pick the smiley face!

## Completing the Key Performance Indicator

A KPI supports properties in addition to the ones I’ve described so far. Although configuring those properties are beyond the scope of the article, you should have some idea how they work. If you click the **Additional Properties** down arrow at the bottom of the KPI form, you can view and configure the following properties:

* **Display folder.** The folder in which the KPI can be found when browsing the cube.
* **Parent KPI.**> A KPI that acts as the parent of the current KPI so the parent KPI can use the value of the child KPI.
* **Current time member.** An MDX expression that returns a member that identifies the KPI’s temporal context.
* **Weight.** An MDX expression that assigns a weight to a child KPI to indicate its relative importance in the parent KPI.
* **Description.** A description of the KPI.

After you’ve completed your KPI, you can then view its results, based on the current values in the cube data. To view the KPI, click the **Browser View** button on the **KPIs** tab. The browser view includes two panes. The top pane lets you define filters that determine what data the KPI uses, and the bottom pane displays the KPI. By default (before any filters are defined), the KPI calculates the KPI value for the entire data set. However, you can create filters that let you define the data for which you want to run the KPI.

For example, Figure 6 shows the filter I created for the Gross Profit KPI. I first selected Pacific as the sales territory group, and then selected 2004 as the calendar year. As you can see in the figure, the KPI value is 29.72%. Because the goal for the Pacific group is only 32%, the Pacific group reached nearly 93% of its goal for 2004, which means the status indicator is green. However, notice that the trend points downward. That’s because they Pacific group did better in 2003 (30.58%).

