**SSAS (SQL Server Analysis Services)**

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.

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.

**SSAS Terminology**

**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 contains the multi-dimensional data and the environment hosting the data.

**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** - A 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, if at all it, it is used as an attribute. Typical examples of dimensions are Geography, Organization, Employee, Time, etc.

**Fact** - A fact is known as measures in a cube and 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. A typical example of facts are Sales, Performance, Tax, etc.

**Hierarchy** - A hierarchy is a collection of nested attributes associated in a parent-child fashion with a defined cardinality. A dimension is formed of attributes, and a 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. The value is derived based on the definition of the 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.

**Cube** - A cube is a multidimensional 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.

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

## Tabular Data Model

**Table** - A table is a set of columns that can be imported from a data source or can be added manually as calculated columns.

**Table metadata** - All the database objects like relationships, measures, perspectives, etc. are all metadata objects within the context of a table.

**Data** - In a tabular model, data is populated either by importing data or by creating data manually in calculated fields called calculated columns.

**Relationships** - In a tabular model, a relationship is a connection between two tables which correlates data between two tables.

**DAX** - Data Analysis Expressions is an expression language that can be used to query tabular data models as well as define calculations in the data model.

**Calculations** - In a tabular model, DAX formulas include functions, operators, and other constructs to create advance calculations.

**Measures** - In a tabular model, a measure is a calculation created using a DAX formula.

**Calculated columns** - A calculated column is a column that is added to an existing table using a DAX formula for the values stored in that column.

**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.

**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.

**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.

**Storage Types in SSAS:**

There are three standard storage modes (MOLAP, ROLAP and HOLAP) in OLAP applications which affect the performance of OLAP queries and cube processing, storage requirements and also determine storage locations.

A big advantage of a BI solution is the existence of a cube. Data and aggregations are stored in a optimized format to offer very fast query performance.

Sometimes, a big disadvantage of storing data and aggregations in a cube is the latency that it implies. SSAS processes data from the underlying relational database into the cube. After this is done the cube is no longer connected to the relational database so changes to this database will not be reflected in the cube. Only when the cube is processed again, the data in the cube will be refreshed.

SQL Server Analysis services gives you the possibility to choose different storage types for the following objects:

Ø  Cubes

Ø  Partitions

Ø  Dimensions

**MOLAP (Multi-dimensional Online Analytical Processing)**

MOLAP is the most used storage type. It’s designed to offer maximum query performance to the users. Data AND aggregations are stored in optimized format in the cube. The data inside the cube will refresh only when the cube is processed, so latency is high.

**ROLAP (Relational Online Analytical Processing)**

ROLAP does not have the high latency disadvantage of MOLAP. With ROLAP, the data and aggregations are stored in relational format. This means that there will be zero latency between the relational source database and the cube.

Disadvantage of this mode is the performance, this type gives the poorest query performance because no objects benefit from multi-dimensional storage.

**HOLAP (Hybrid Online Analytical Processing)**

HOLAP is a storage type between MOLAP and ROLAP. Data will be stored in relational format (ROLAP), so there will also be zero latency with this storage type.

Aggregations, on the other hand, are stored in multi-dimensional format (MOLAP) in the cube to give better query performance. SSAS will listen to notifications from the source relational database, when changes are made, SSAS will get a notification and will process the aggregations again.

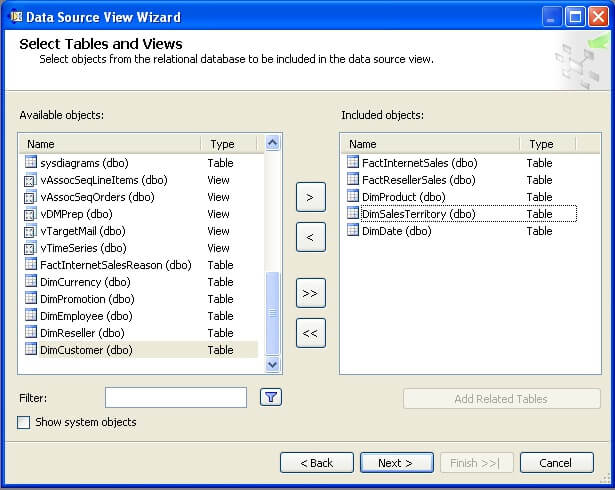
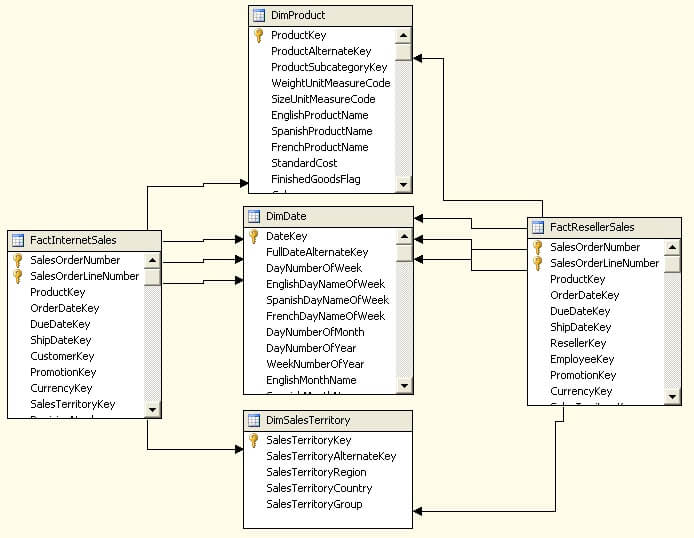
With this mode it’s possible to offer zero latency to the users but with medium query performance compared to MOLAP and ROLAP.

**SSAS Project – Example**

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:

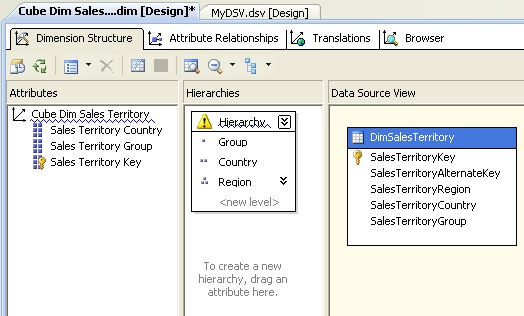
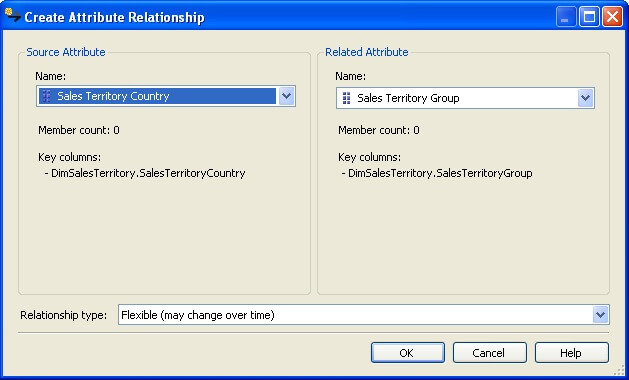
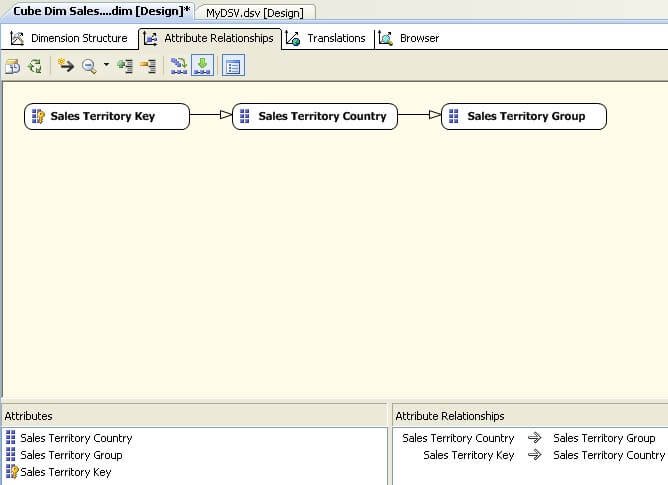
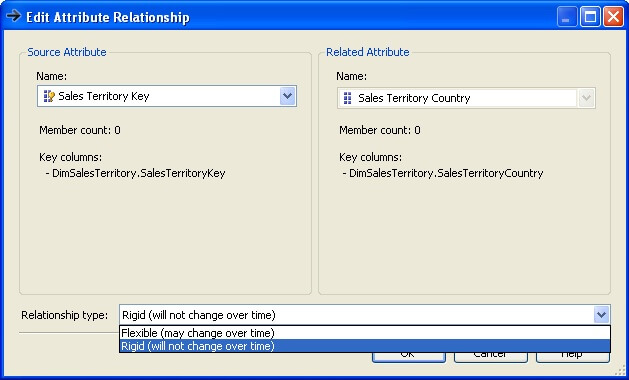
* Design and develop a star-schema
* Create dimensions, hierarchies, and cubes
* Process and deploy a cube
* Develop calculated measures and named sets using MDX
* Browse the cube data using Excel as the client tool

## 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.

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.

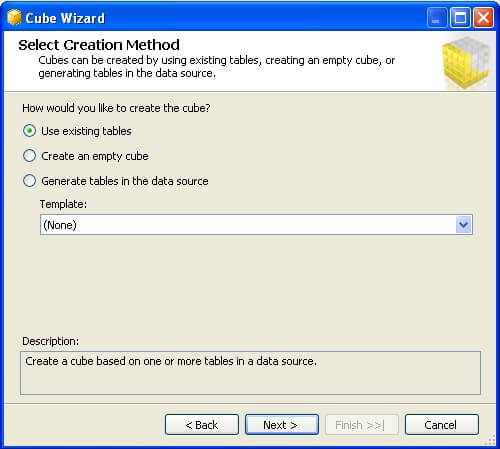
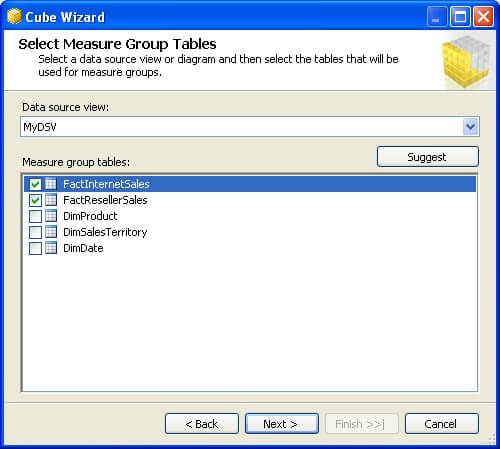
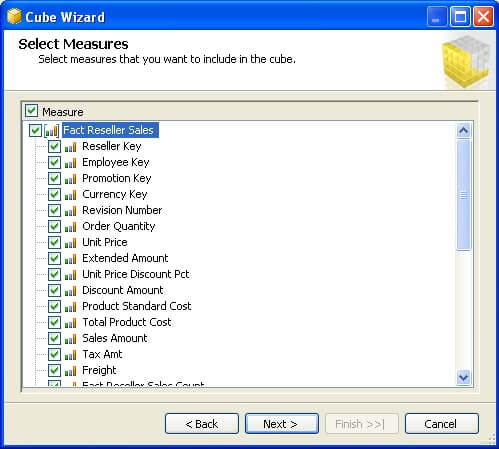
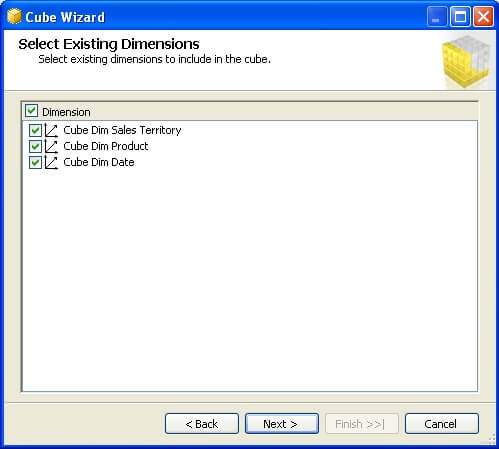
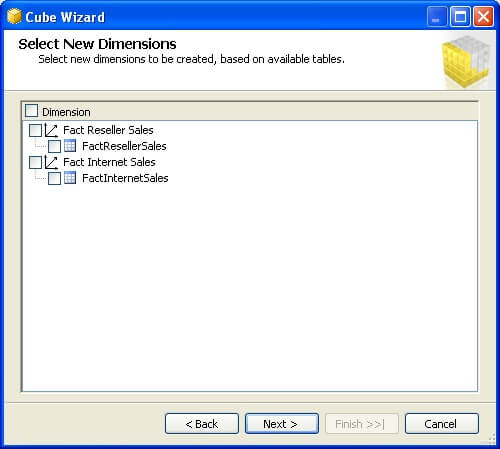
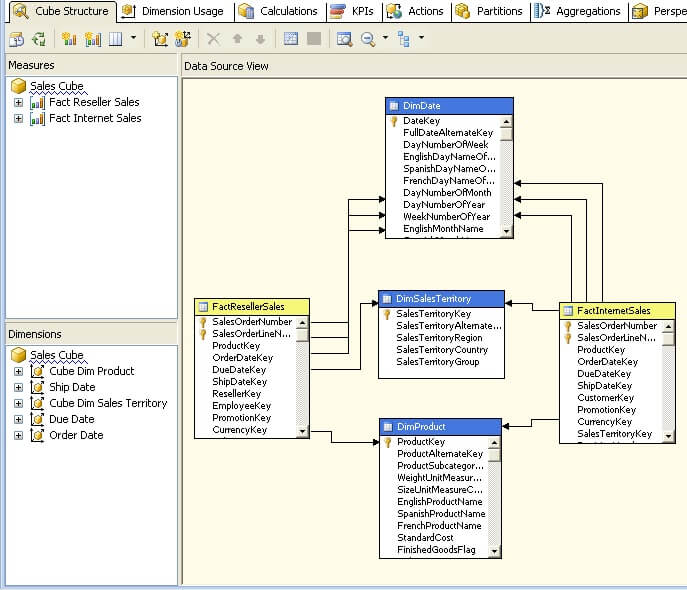
## Explanation

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

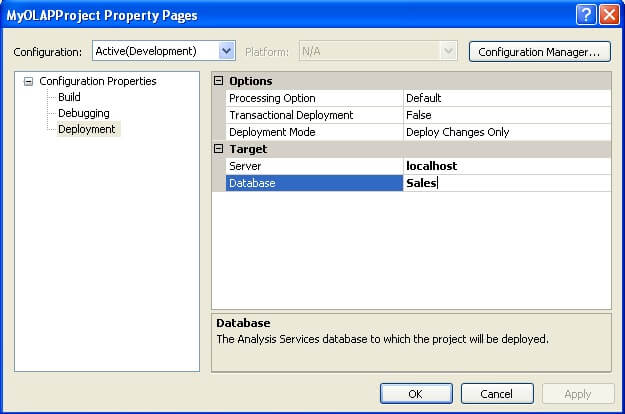
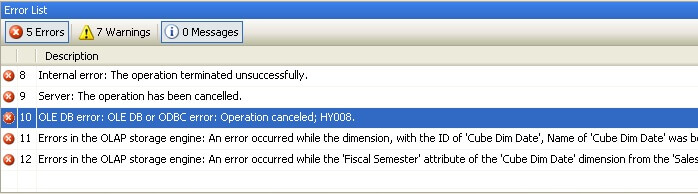
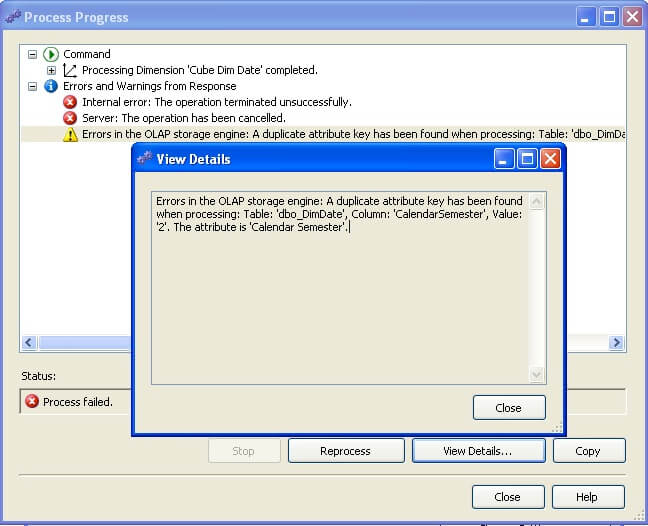
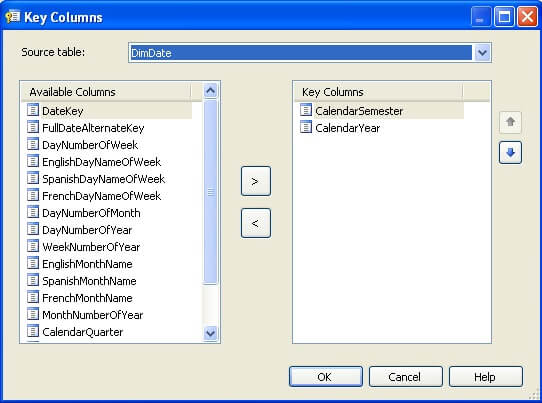
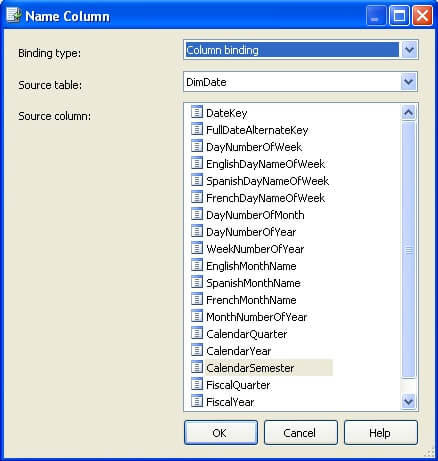
A Cube acts as an OLAP database to the subscribers who need to query data from an OLAP data store. A Cube is the main object of a SSAS solution where the majority of fine tuning, calculations, aggregation design, storage design, defining relationship and a lot of other configurations are developed. We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

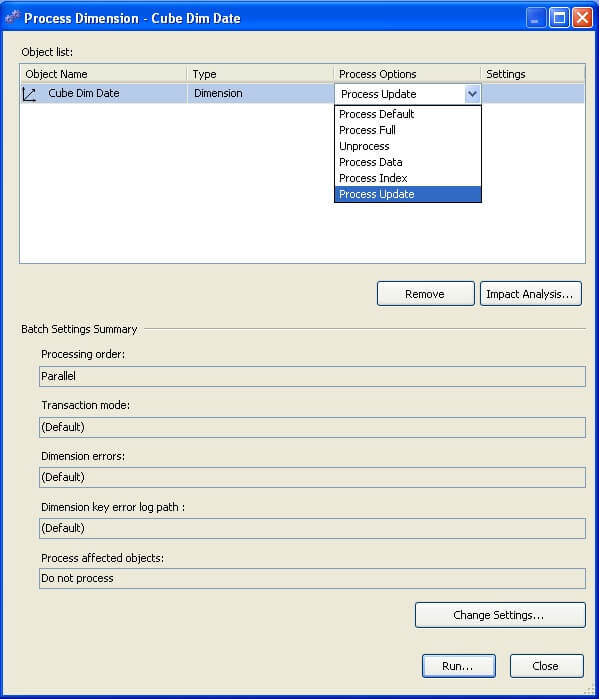
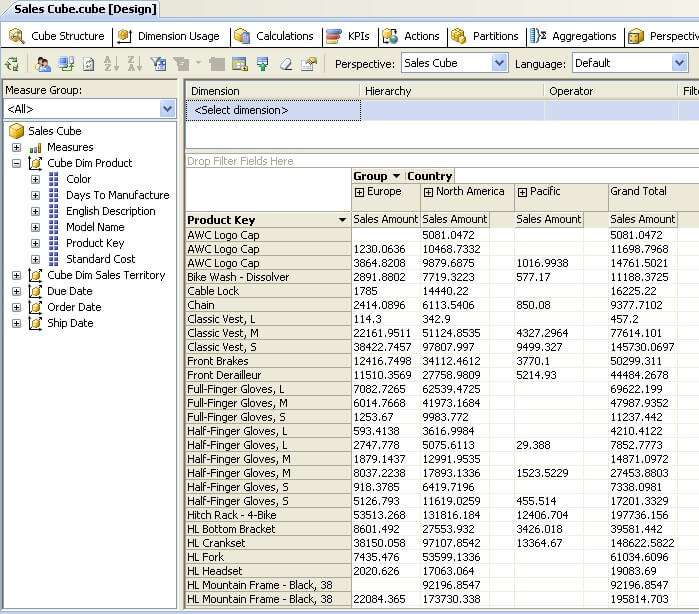
## Explanation

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

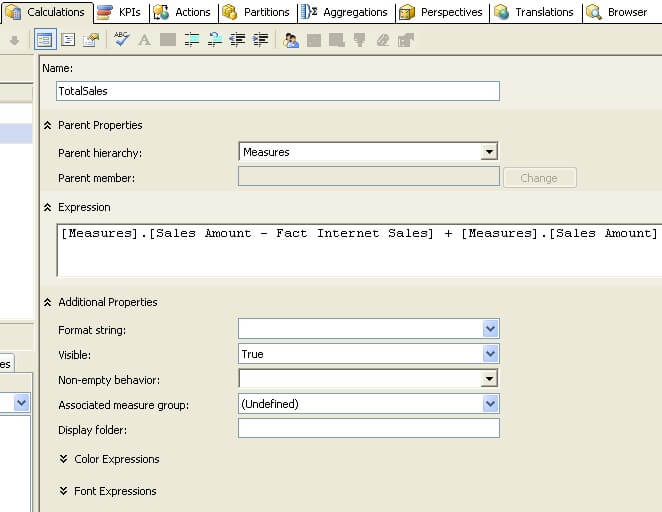
## Explanation

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.  
  


# Developing a Calculated Measure

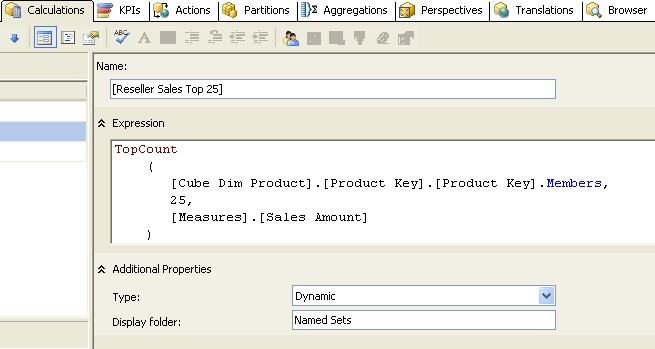
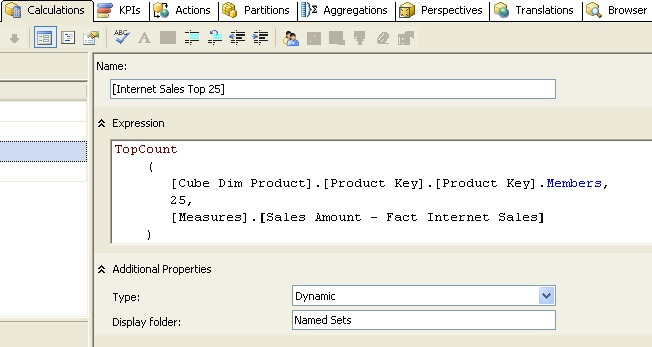
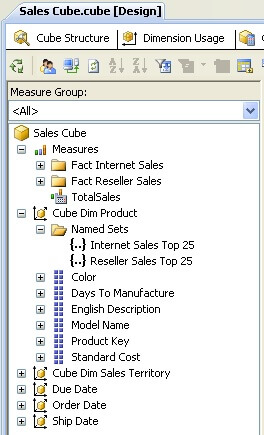
## Explanation

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.

# Developing Named Sets

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.  
  


## ****What is MDX?****

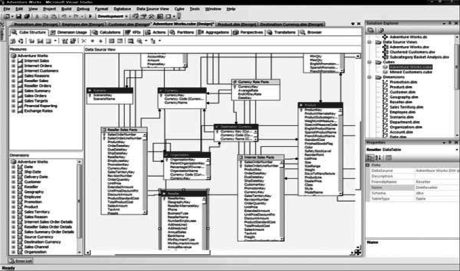
MDX (Multi – Dimensional eXpressions) is a query language used to retrieve data from multidimensional databases. More specifically, MDX is used for querying multidimensional data from Analysis Services and supports two distinct modes.

MDX is not a proprietary language; it is a standards – based query language used to retrieve data from OLAP databases. MDX is part of the OLE DB for OLAP specification sponsored by Microsoft. Many other OLAP providers support MDX, including Microstrategy’s Intelligence Server, Hyperion’s Essbase Server, and SAS’s Enterprise BI Server.

When one refers to MDX they might be referring either to the MDX query language or to MDX expressions. Even though the MDX query language has similar syntax as that of SQL, it is significantly different.

**Fundamental Concepts:**

A multidimensional database is typically referred to as a cube. The cube is the foundation of a multidimensional database, and each cube typically contains more than two dimensions.

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/110.png)

A set can contain zero, one, or more tuples. A set with zero tuples is referred to as an empty set. An empty set is represented as :

<pre>{   }

{Customer.Country.Australia, Customer.Country.Canada,Customer.Country.Australia}</pre>

This set contains two instances of the tuple Customer.Country.Australia. Because a member of a dimension by itself forms a tuple, it can be used as such in MDX queries. Similarly, if there is a tuple that is specified by only one hierarchy, we do not need the parentheses to specify it as a set. When there is a single tuple specified in a query we do not need curly braces to indicate.

**MDX Queries:**

The syntax for an MDX query is as follows:

<pre>[WITH <formula\_expression> [, <formula\_expression> …]]

SELECT [ <axis\_expression> , [ <axis\_expression> …]]

FROM [ <cube\_expression> ]

[WHERE [slicer\_expression]]</pre>

The keywords WITH ,SELECT , FROM , and WHERE along with the expressions following them are referred to as a *clauses*. In the preceding MDX query template, anything specified within square brackets means it isoptional; that is, that section of the query is not mandatory in an MDX query. We can see that the WITH and WHERE clauses are optional because they are enclosed within square brackets.

The WITH clause is typically used for custom calculations and operations.

**The SELECT Statement and Axis Specification**

The MDX SELECT statement is used to retrieve a subset of the multidimensional data in an OLAP cube. In SQL, the SELECT statement allows us to specify which columns will be included in the row data we retrieve, which is viewed as two – dimensional data. If you consider a two – dimensional coordinate system, you have the X and Y axes. The Y axis is used for the COLUMNS and the X axis is used for ROWS. In MDX, the SELECT statement is specified in a way that allows retrieving data with more than just two dimensions. Indeed, MDX provides you with the capability of retrieving data on one, two, or many axes.

The syntax of the SELECT statement is :

**SELECT [ <axis\_expression> , [ <axis\_expression> …]]**

The axis\_expressions specified after the SELECT refer to the dimension data we are interested in retrieving. These dimensions are referred to as axis dimensions because the data from these dimensions are projected onto the corresponding axes.

The syntax for axis\_expressionis :

**<pre><axis\_expression> := < set > ON (axis | AXIS(axis number) | axis number)</pre>**

Axis dimensions are used to retrieve multidimensional result sets. The set, a collection of tuples, is defined to form an axis dimension. MDX provides you with the capability of specifying up to 128 axes in the SELECT statement. The first five axes have aliases. They are COLUMNS, ROWS, PAGES, SECTIONS, and CHAPTERS. Axes can also be specified as a number, which allows you to specify more than five dimensions in your SELECT statement.

Lets take the following example:

SELECT Measures.[Internet Sales Amount] ON COLUMNS,

[Customer].[Country].MEMBERS ON ROWS,

[Product].[Product Line].MEMBERS ON PAGES

FROM [Adventure Works]

Three axes are specified in the SELECT statement. Data from dimensions Measures, Customers, and Product are mapped on to the three axes to form the axis dimensions.

This statement could equivalently be written as :

SELECT Measures.[Internet Sales Amount] ON 0,

[Customer].[Country].MEMBERS ON 1,

[Product].[Product Line].MEMBERS ON 2

FROM [Adventure Works]

**Axis Dimensions:**

The axis dimensions are what we build when we define a SELECT statement. A SELECT statement specifies a set for each dimension; COLUMNS, ROWS, and additional axes. Unlike the slicer dimension, axis dimensions retrieve and retain data for multiple members, not just single members.

**The FROM Clause and Cube Specification:**

The FROM clause in an MDX query determines the cube from which you retrieve and analyze data. It’s similar to the FROM clause in a SQL query where you specify a table name. The FROM clause is a necessity for any MDX query.

The syntax of the FROM clause is :

**FROM <cube\_expression>**

The cube\_expression denotes the name of a cube or a subsection of a cube from which we want to retrieve data. In SQL’s FROM clausewe  can specify more than one table, but in an MDX FROM clause we can define just one cube name.

The cube specified in the FROM clause is called the ***cube context***and the query is executed within this cube context. That is, every part of axisexpressions are retrieved from the cube context specified in the FROM clause:

SELECT [Measures].[Internet Sales Amount] ON COLUMNS

FROM [Adventure Works]

This is a valid MDX query that retrieves data from the [Internet Sales Amount] measure on the X – axis. The measure data is retrieved from the cube context [Adventure Works]. Even though the FROM clause restricts us to working with only one cube or section of a cube, we can retrieve data from other cubes using the **MDX LookupCubefunction**.

When there are two ore more cubes having common dimension members, the **LookupCube**function retrieves measures outside the current cube’s context using the common dimension members.

**The WHERE Clause and Slicer Specification:**

In  any relational database work that we do, we issue queries that return only portions of the total data available in a given table, set of joined tables, and/or joined databases. This is accomplished using SQL statements that specify what data we do and do not want returned as a result of running your query.

Here is an example of an unrestricted SQL query on a table named Product that contains sales information for products:

SELECT \*

FROM Product

Assume the preceding query results in five columns being retrieved with the following four rows

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Product ID** | **Product Line** | **Color** | **Weight** | **Sales** |
| 1 | Accessories | Silver | 5.00 | 200.00 |
| 2 | Mountain | Grey | 40.35 | 250.00 |
| 3 | Road | Silver | 50.23 | 2500 |
| 4 | Touring | Red | 45.11 | 2000.00 |

The \* represents “ all, ” meaning that query will dump the entire contents of the table. If we want to know only the Color and Product Line for each row, we can restrict the query so that it returns just the we want. The following simple example demonstrates a query constructed to return just two columns from the table:

SELECT ProductLine, Color

FROM Product

This query returns the following:

|  |  |
| --- | --- |
| Product Line | Color |
| Accessories | Silver |
| Mountain | Grey |
| Road | Silver |
| Touring | Red |

The MDX SELECT statement is used to identify the dimensions and members a query will return and the WHERE statement limits the result set by some criteria. The preceding SQL example restricts the returned data to records where Color = ‘ Silver ’ . Note that in MDX members are the elements that make up a dimension ’ s hierarchy. The Product table, when modeled as a cube, will contain two measures, Sales and Weight, and a Product dimension with the hierarchies ProductID, ProductLine, and Color. In this example the Product table is used as a fact as well as a dimension table. An MDX query against the cube that produces the same results as that of the SQL query is :

SELECT Measures.[Sales] ON COLUMNS,

[Product].[Product Line].MEMBERS on ROWS

FROM [ProductsCube]

WHERE ([Product].[Color].[Silver])

This query returns the following:

|  |  |
| --- | --- |
| ProductLine | Sales |
| Accessories | 200 |
| Road | 2500.00 |

The two columns selected in SQL are now on the axes COLUMNS and ROWS. The condition in the SQL WHERE clause, which is a string comparison, is transformed to an MDX WHERE clause, which refers to a slice on the cube that contains products that have silver color.

**The Slicer Dimension:**

The *slicer dimension*is what you build when you define the WHERE statement. It is a filter that removes unwanted dimensions and members.

The slicer dimension includes any axis in the cube including those that are not explicitly included in any of the queried axes. The default members of hierarchies not included in the query axes are used in the slicer axis. When there are tuples specified for the slicer axis, MDX will evaluate those tuples as a set and the results of the tuples are aggregated based on the measures included in the query and the aggregation function of that specific measure.

**The WITH Clause and Calculated Members:**

Often business needs involve calculations that must be formulated within the scope of a specific query.

The MDX WITH clause provides you with the ability to create such calculations and use them within the context of the query. In addition, we can also retrieve data from outside the context of the current cube using the LookupCubeMDX function.

Typical calculations that are created using the WITH clause are named sets and calculated members. In addition to these, the WITH clause provides us with functionality to define cell calculations, load a cube into an Analysis Server cache for improving query performance, alter the contents of cells by calling functions in external libraries, and additional advanced capabilities such as solve order and pass order.

The syntax of the WITH clause is :

[WITH <formula\_expression> [, <formula\_expression> …]]

The formula\_expression will vary depending upon the type of calculations. Calculations are separated by commas.

**Named Sets:**

A named set is nothing but an alias for an MDX set expression that can be used anywhere within the query as an alternative to specifying the actual set expression.

**Calculated Members:**

Calculated members are calculations specified by MDX expressions. They are resolved as a result of MDX expression evaluation rather than just by the retrieval of the original fact data.

The formula\_expression of the WITH clause for calculated members is :

Formula\_expression := MEMBER <MemberName> AS [‘] <MDX\_Expression> [‘],

[ , SOLVE\_ORDER = < integer > ]

[ ,<CellProperty> = <PropertyExpression> ]

MDX uses the keywords MEMBER and AS in the WITH clause for creating calculated members.

The following are examples of calculated member statements:

**Example 1**:

WITH MEMBER MEASURES.[Profit] AS [Measures].[Internet Sales Amount]-

[Measures].[Internet Standard Product Cost]

SELECT measures.profit ON COLUMNS,

[Customer].[Country].MEMBERS ON ROWS

FROM [Adventure Works]

In Example a calculated member, Profit, has been defined as the difference of the measures [Internet Sales Amount] and [Internet Standard Product Cost]. When the query is executed, the Profit value will be calculated for every country based on the MDX expression.

**MDX Expressions**

MDX expressions are partial MDX statements that evaluate to a value. They are typically used in calculations or in defining values for objects such as default members and default measures, or for defining security expressions to allow or deny access. MDX expressions typically take a member, a tuple, or a set as a parameter and return a value. If the result of the MDX expression evaluation is no value, a Null value is returned.

Following are some examples of MDX expressions:

**Example 1**

Customer.[Customer Geography].DEFAULTMEMBER

It will returns the default member specified for the Customer Geography hierarchy of

the Customer dimension.

**Operators:**

An operator is a function that is used to perform a specific action, takes arguments, and returns a result. MDX has several types of operators including arithmetic operators, logical operators, and special MDX operators.

**Arithmetic Operators:**

Regular arithmetic operators such as +, – , \*, and / are available in MDX. Just as with other programming languages, these operators can be applied on two numbers. The + and – operators can also be used as unary operators on numbers. Unary operators, as the name indicates, are used with a single operand (single number) in MDX expressions such as + 100 or – 100.

**Set Operators:**

The +, – , and \* operators, in addition to being arithmetic operators, are also used to perform operations on the MDX sets. The + operator returns the union of two sets, the – operator returns the difference of two sets, and the \* operator returns the cross product of two sets. The cross product of two sets results inall possible combinations of the tuples in each set and helps in retrieving data in a matrix format.

**Example 1:**The result of the MDX expression

{[Customer].[Country].[Australia]} + {[Customer].[Country].[Canada]}

The union of the two sets as shown here:

{[Customer].[Country].[Australia], [Customer].[Country].[Canada]}

**Example 2:**The result of the MDX expression

{[Customer].[Country].[Australia],[Customer].[Country].[Canada]}\*

{[Product].[Product Line].[Mountain],[Product].[Product Line].[Road]}

The cross product of the sets as shown here:

{([Customer].[Country].[Australia],[Product].[Product Line].[Mountain])

([Customer].[Country].[Australia],[Product].[Product Line].[Road])

([Customer].[Country].[Canada],[Product].[Product Line].[Mountain])

([Customer].[Country].[Canada],[Product].[Product Line].[Road])}

**Comparison Operators:**

MDX supports the comparison operators < ,< =, > , > =, =, and <> . These operators take two MDX expressions as arguments and return TRUE or FALSE based on the result of comparing the values of each expression.

**Example:**

The following MDX expression uses the greater than comparison operator, > :

Count (Customer.[Country].members) > 3

In the above example Count is an MDX function that is used to count the number of members in Country hierarchy of the Customer dimension. Because there are more than three members, the result of the MDX expression is TRUE.

**Logical Operators:**

The logical operators that are part of MDX are AND, OR, XOR, NOT, and IS, which are used for logical conjunction, logical disjunction, logical exclusion, logical negation, and comparison, respectively. These operators take two MDX expressions as arguments and return TRUE or FALSE based on the logical operation. Logical operators are typically used in MDX expressions for cell and dimension security.

**Special MDX Operators — Curly Braces, Commas,and Colons:**

The curly braces, represented by the characters { and }, are used to enclose a tuple or a set of tuples to form an MDX set. Whenever we have a set with a single tuple, the curly brace is optional because Analysis Services implicitly converts a single tuple to a set when needed. When there is more than one tuple to be represented as a set or when there is an empty set, we need to use the curly braces.

The comma character is used to form a tuple that contains more than one member. By doing this you are creating a slice of data on the cube. In addition, the comma character is used to separate multiple tuples specified to define a set. In the set {(Male,2003), (Male,2004), (Male,2005),(Female,2003),(Female,2004),(Female,2005)} the comma character is not only used to form tuples but also to form the set of tuples.

The colon character is used to define a range of members within a set. It is used between two non -consecutive members in a set to indicate inclusion of all the members between them, based on the setordering (key – based or name – based).

For example:

{[Customer].[Country].[Australia], [Customer].[Country].[Canada],

[Customer].[Country].[France], [Customer].[Country].[Germany],

[Customer].[Country].[United Kingdom], [Customer].[Country].[United States]}

The following MDX expression

{[Customer].[Country].[Canada] : [Customer].[Country].[United Kingdom]}

Results in the following set:

{[Customer].[Country].[Canada], [Customer].[Country].[France],

[Customer].[Country].[Germany], [Customer].[Country].[United Kingdom]}

**MDX Functions:**

MDX functions can be used in MDX expressions or in MDX queries. MDX expressions or queries including ordering tuples in a set, counting the number of members in a dimension, and string manipulation required to transform user input into corresponding MDX objects.

**MDX Function Categories:**

MDX functions can be called in several ways:

1. **Function (read *dot*function)**

**Example:**Dimension.Name returns the name of the object being referenced (could be a hierarchy or level/member expression). Perhaps this reminds us of the dot operator in VB.NET

WITH MEMBER measures.LocationName AS [Customer].[Country].CurrentMember.Name

SELECT measures.LocationName ON COLUMNS,

Customer.Country.members on ROWS

FROM [Adventure Works]

1. **Function:**

**Example:**Username is used to acquire the username of the logged – in user. It returns a string in the following format: domain – name\user – name. Most often this is used in dimension or cell

security related MDX expressions. The following is an example of how username can be used in an MDX expression:

WITH MEMBER Measures.User AS USERNAME

SELECT Measures.User ON 0 FROM [Adventure Works]

1. **Function ( )**

**Example:**The function CalculationCurrentPass ( ) requires parentheses, but takes no arguments.

1. **Function (arguments):**

**Example:**OpeningPeriod( [Level\_Expression [ , Member\_Expression] ] ) is an MDX function that takes an argument that can specify both level\_expressionwitHmember\_expression or just the member\_expression itself. This function is most often used with Time dimensions, but will work with other dimension types. It returns the first member at the level of the member\_ expression. For example, the following returns the first member of the Day level of the April member of the default time dimension:

OpeningPeriod (Day, [April])

**Set Functions:**

Set functions,  operate on sets. They take sets as arguments and often return a set. Some of the widely used set functions are Crossjoinand Filter .

Crossjoinreturns all possible combinations of sets as specified by the arguments to the Crossjoinfunction. If there are N sets specified in the Crossjoinfunction, this will result in a combination of allthe possible members within that set on a single axis.

**Example:**

Crossjoin( Set\_Expression [ ,Set\_Expression ...] )

SELECT Measures.[Internet Sales Amount] ON COLUMNS,

CROSSJOIN( {Product.[Product Line].[Product Line].MEMBERS},

{[Customer].[Country].MEMBERS}) on ROWS

FROM [Adventure Works]

This query produces the cross product of each member in the Product dimension with each member of the Customer dimension along the sales amount measure. The following are the first few rows of results from executing this query:

**Sales Amount:**

Accessory All Customers $604,053.30

Accessory Australia $127,128.61

Accessory Canada$82,736.07

Accessory France$55,001.21

Accessory Germany $54,382.29

AccessoryUnited Kingdom$67,636.33

Accessory United States$217,168.79

Components All Customers (null)

**Member Functions:**

Member functions are used for operations on the members such as retrieving the current member,ancestor, parent, children, sibling, next member, and so on. All the member functions return a member.One of the most widely used member functions is called ParallelPeriod. The ParallelPeriodfunction helps you to retrieve a member in the Time dimension based on a given member and certain conditions.

The function definition for ParallelPeriodis :

ParallelPeriod( [ Level\_Expression [ ,Numeric\_Expression [ , Member\_Expression ] ] ] )

The ParallelPeriodfunction is used to compare measure values relative to various time periods.

**Numeric Functions:**

Numeric functions come in used when we are defining the parameters for an MDX query or creating any calculated measure.

The most common of the numeric functions is a simple one called Count along with its close cousin, DistinctCount. The Count function is used to count the number of items in the collection of a specific object like a Dimension, a Tuple, a Set, or a Level. The DistinctCountfunction, on the other hand, takes a Set\_Expression as an argument and returns a number that indicates the number of distinct items in the Set\_Expression, not the total count of all items.

Here are the function definitions for each:

Count ( Dimension | Tuples | Set| Level)

DistinctCount( Set\_Expression )

**Example:**

WITH MEMBER Measures.CustomerCount AS DistinctCount(

Exists([Customer].[Customer].MEMBERS,[Product].[Product Line].Mountain,

“Internet Sales”))

SELECT Measures.CustomerCount ON COLUMNS

FROM [Adventure Works]

The DistinctCountfunction counts the number of distinct members in the Customer dimension who have purchased products in the Mountain product line. If a customer has purchased multiple products from the specified product line, the DistinctCountfunction will count the customer just once. The MDX function Exists is used to filter customers who have only purchased product line Mountain through the Internet.

**Dimension Functions, Level Functions, and Hierarchy Functions:**

Functions in these groups are typically used for navigation and manipulation. Here is an example of just such a function, the “ Level ” function from the Level group:

SELECT [Date].[Calendar].[Calendar Quarter].[Q1 CY 2004].LEVEL ON COLUMNS

FROM [Adventure Works]

This query results in a list of all the quarters displayed in the results. The reason is because [Date].[Calendar].[Calendar Quarter].[Q1 CY 2004].LEVEL evaluates to [Date].[Calendar Year].[CalendarSemster].[Calender Quarter]. From this, you get the list of all quarters for all calendar years

**SQL Server Analytical Services (SSAS):** SSAS is a technology from Microsoft Business Intelligence stack, which is used to develop online Analytical Processing (OLAP) solutions. It can also be used to create cubes using data from data marts/ data warehouse for faster and efficient data analysis.

**Cubes:** Cubes are the multi-dimensional data sources which has two basic constituents named dimensions and facts (measures). Dimensions are referred to as Master Tables and facts are referred to as measurable details

**Multi-Dimensional expressions (MDX):** MDX is a query language that is used to query a cube

**Key features of SSAS:**

* **Speed:** It takes less time to respond to a query due to aggregation of the facts
* **Data Analysis:** Allows multi-dimensional analysis facilitated by the cubes
* **Automatic Link and display:** It provides the facility to automatically link and display the report
* **Good data model:** For better business reporting and analysis a good data model can be created

**MDX (Multi-dimensional expressions):** It is a query language used for retrieving data from multidimensional databases like OLAP databases

**Multi-dimensional database:** It is referred to as a cube which is a foundation of multi-dimensional databases and each cube typically contains more than two dimensions

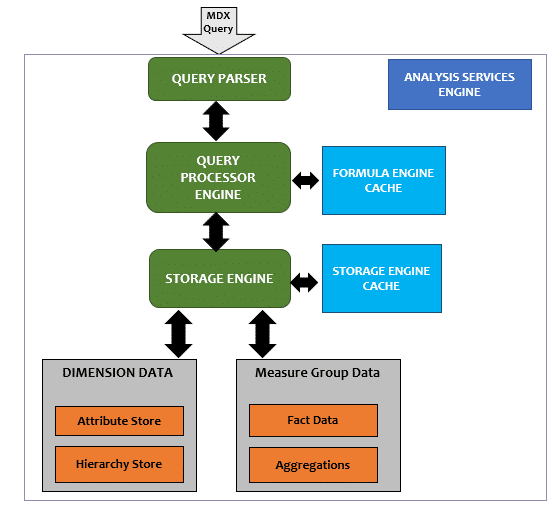
**OLAP cube:** OLAP cube is a technique which is used to hold the data in an optimized form and used to analyze the data with quick response

**Star schema:** A schema where every dimension present in the Data Source View (DSV) is directly linked or related to fact or measurable table. It consists of a DE normalized data and can be used in small companies with small databases

**Snowflake schema:** It is a schema where some dimensions are linked directly to a fact table and some are indirectly linked to fact tables. It consists of a normalized data and can be used in large companies with big databases

**Star Flake:** It is a hybrid structure which contains a combination or star (DE normalized data) and snowflake (normalized data) schema

**Data Source Views (DSV):** DSVs enable to create logical view of only the tables involved in the data warehouse design

**SSAS Architecture**

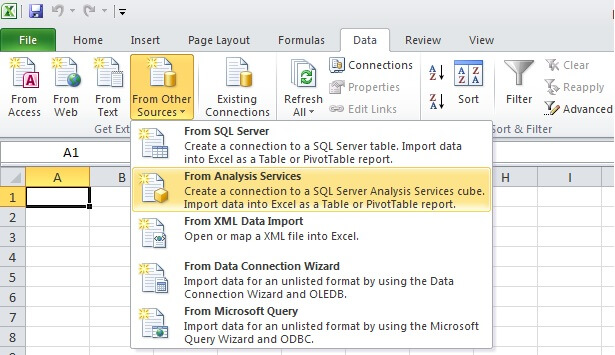
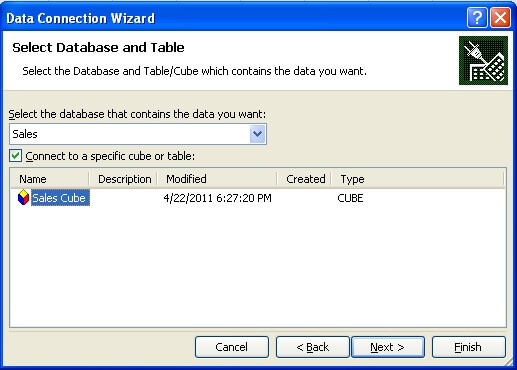
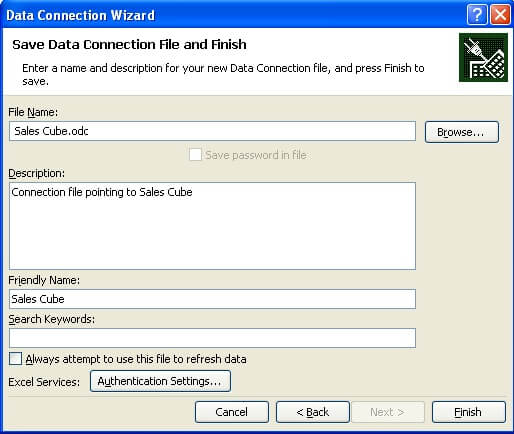
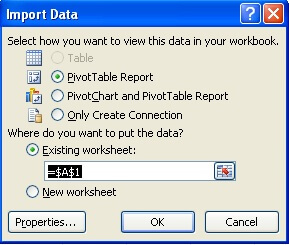
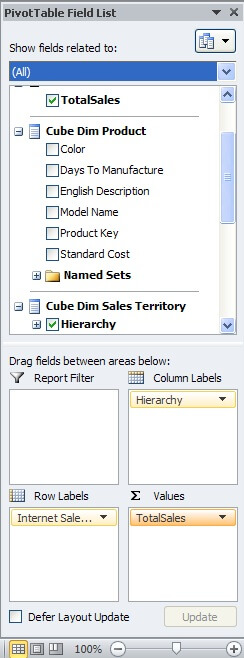
**MDX Expressions:**

|  |  |
| --- | --- |
| **Statement** | **Expression** |
| To calculate Simple Gross Profit | [Measures].[Sales Amount] – [Measures].[Total Product Cost] |
| To calculate sales in a particular country e.g. (Canada) | ([Measures].[Sales Amount], [Customer].[Country].&[Canada]) |
| To calculate year to date sales of any level of date hierarchy | AGGREGATE( PeriodsToDate( [Date].[Calendar Hierarchy].[Year], [Date].[Calendar Hierarchy].CurrentMember ), ([Measures].[Sales])) |
| Alternate year to date expression | AGGREGATE ( YTD ([Date Order].[Calendar].CurrentMember), Measures.[Sales Amount]) |
| To check the product ranking | IIF (Product.Product.CurrentMember IS Product.Product.[All],NULL, IIF (Measures.[Sales Amount] = 0, NULL, RANK(Product.Product.CurrentMember, ORDER (Product.Product.Members, Measures.[Sales Amount])))) |
| Sales from 365 days ago | (ParallelPeriod([Invoice Date].[Date Hierarchy].[Date], 365, [Invoice Date].[Date Hierarchy].CurrentMember), [Measures].[Sales Amount]) |
| Sales done in the previous period | (Measures.[Sales Amount], [Date Order].[Calendar].CurrentMember.PrevMember) |
| To view top 10 selling products | TopCount (Product.Product.Product.Members,10,Measures.[Sales Amount]) |
| To display the average sales from three years till date | Avg( {ParallelPeriod( [Date].[Date].[Year], 3, StrToMember(“[Date].[Date].&[” + Format(now(), “yyyyMMdd”) + “]”)) : StrToMember(“[Date].[Date].&[” + Format(now(), “yyyyMMdd”) + “]”)} , [Measures].[Sales Count]) |
| Drillthrough Action caption | ‘Get Sales Details for’ +[Product].[Product].CurrentMember.Member\_Caption |
| To change SSAS Calculation text color | IIF([Measures].[Profit Percentage] < .40, 255 , 0) |
| Changing a calculation using scope statement | SCOPE ([Measures].[Scope Profit]); THIS = ([Measures].[Sales Amount] – [Measures].[Standard Product Cost]); END SCOPE; |
| Clear ration value when at all levels | SCOPE ([Customer].[Customer Geography].[All], Measures.RatioOverParent); THIS = NULL |
| SSAS KPI Value Expression | [Measures].[Sales Amount] \* 1.2 |
| SSAS KPI Goal Expression | Case When IsEmpty (ParallelPeriod ([Date Order].[Fiscal].[Fiscal Year], 1,[Date Order].[Fiscal].CurrentMember)) Then [Measures].[Sales Amount] Else 1.10 \* ([Measures].[Sales Amount], ParallelPeriod([Date Order].[Fiscal].[Fiscal Year], 1,[Date Order].[Fiscal].CurrentMember)) End |
| SSAS KPI Status Expression | Case When KpiValue( “Sales Revenue YTD” ) / KpiGoal( “Sales RevenueYTD” ) > 1 Then 1 When KpiValue( “Sales Revenue YTD” ) / KpiGoal( “Sales Revenue YTD” ) <= 1 And KpiValue( “Sales Revenue YTD” ) / KpiGoal( “Sales Revenue YTD” ) >= .85 Then 0 Else -1 End |
| SSAS KPI Trend Expression | Case When IsEmpty (ParallelPeriod ([Date Order].[Fiscal].[Fiscal Year], 1,[Date Order].[Fiscal].CurrentMember)) Then 0 When VBA!Abs ((KpiValue( “Sales Revenue YTD” )- (KpiValue( “Sales Revenue YTD” ), ParallelPeriod ( [Date Order].[Fiscal].[Fiscal Year], 1, [Date Order].[Fiscal].CurrentMember)))<=.02 (KpiValue( “Sales Revenue YTD” ), ParallelPeriod ( [Date Order].[Fiscal].[Fiscal Year], 1,[Date Order].[Fiscal].CurrentMember))) <=.02 Then 0 When (KpiValue( “Sales Revenue YTD” )- (KpiValue( “Sales Revenue YTD” ), ParallelPeriod ( [Date Order].[Fiscal].[Fiscal Year], 1, [Date Order].[Fiscal].CurrentMember))) /(KpiValue( “Sales Revenue YTD” ), ParallelPeriod( [Date Order].[Fiscal].[Fiscal Year],1,[Date Order].[Fiscal].CurrentMember)) >.02 Then 1 Else -1 End |

# Using Excel and Creating a Pivot Table Report

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.  
  
