**Introduction to 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.

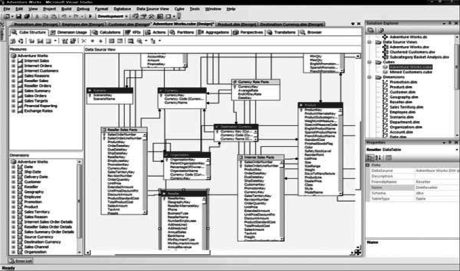
The Intellipaat [SSAS online training](https://intellipaat.com/ssas-training-online/) is all you need to get ahead in your career.

MDX is not a proprietary language; it is a standards – based query language used to retrieve data from [OLAP](https://intellipaat.com/tutorial/msbi-tutorial/ssas/) 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.

**Working with Data Sources and Data Source Views**

[« Previous](https://intellipaat.com/tutorial/ssas-tutorial/introduction-to-mdx/)

An Analysis Services database contains a collection of Data Sources, which stores all the data sources used to build dimensions and cubes within that database. Analysis Services will be able to retrieve source data from data sources via the native OLE DB interface or the managed .NET provider interface.

Data source that contains one fact table with some number of dimensions linked to it by joins; that data source is populated by data from an OLTP database and is called an *Operational Data Store* (ODS).

The ODS is a single entity storing data from various sources so that it can serve as a single source of data for your data warehouse.

Check the Intellipaat [SSAS online training](https://intellipaat.com/ssas-training-online/#course-content) course content now.

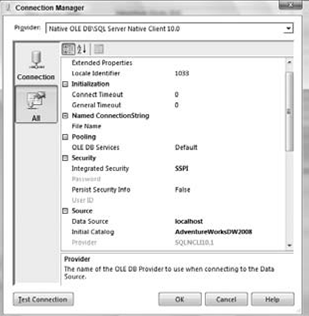
**Data Sources Supported by Analysis Services:**

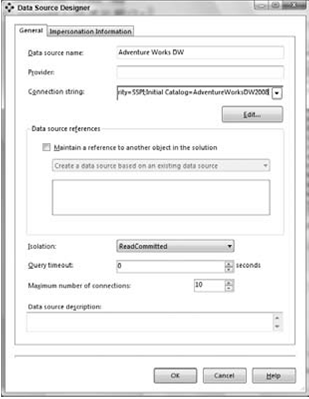
Analysis Services uses a cartridge mechanism that allows it to use the appropriate SQL language and extensions to talk to different relational database systems.

Analysis Services 2008 officially supports specific relational data sources. The major relational datasources for Analysis Services databases include Microsoft SQL Server, IBM ’ s DB2, Teradata, Oracle,Sybase, and Informix.

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/210.png)

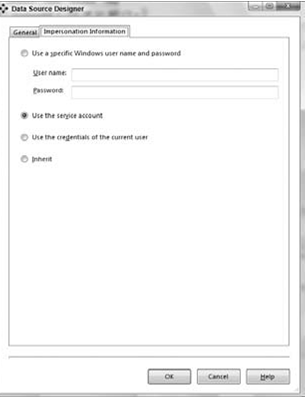
The Data Source Wizard also allows you to create data sources based on an existing data source connection already created so that a single connection can be shared by Analysis Services for multiple databases. The wizard also allows you to establish connections to objects within the current Analysis Services project, such as establishing an OLE DB connection to the cube being created in the project. Such a connection is typically useful while creating mining models.

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/410.png)

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/51.png)

The Impersonation Information dialog in the Data Source Wizard has four options to choose from. The following lists more details on the four options and when each option is likely to be used:

1. **Use a specific Windows user name and password:** we typically would choose this optionwhen the SSAS instance service startup account does not have permissions to access thebackend. When we select this option you need to specify a Windows username and password that [SSAS](https://intellipaat.com/product/ssas-self-paced-training/) will use to connect to the relational backend. Due to security reasons theusername and password are encrypted and stored. Only the encrypted password is sent tothe SSAS instance when the project is deployed.
2. **Use the service account:** This is the option typically selected by most users. We need to makesure the service startup account of the SSAS instance has access to the relational backend.
3. **Use the credentials of the current user:** This option is typically selected for data mining. Thisoption can be used for out – of – line bindings, DMX OPENQUERY statements, local cubes, andmining models. Do not select this option when we are connecting to a relational backend forprocessing, ROLAP queries, remote partitions, linked objects, and synchronization from targetto source.
4. **Inherit:** This option instructs Analysis Services to use the impersonation information specified for the database connection. This option used to be called “ Default ” in SQL Server 2005 edition.

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/61.png)

**. NET versus OLE DB Data Providers:**

There are two types of data providers that most data sources support. OLE DB defines a set of COM interfaces that let you access data from data sources. There is also a .NET managed code interface similarto OLE DB. Providers implementing that interface are called .NET providers.

**. NET Providers:**

Microsoft has created the .NET Framework and programming languages that use the framework to run in the Common Language Runtime (CLR) environment. The .NET Framework itself is a huge class library that exposes tons of functionality and does so in the context of managed code.

**OLE DB Data Providers:**

OLE DB is an industry standard that defines a set of COM (Component Object Model) interfaces that allowclients to access data from various data stores. The OLE DB standard was created for client applications to have a uniform interface from which to access data. Such data can come from a wide variety of data sources such as Microsoft Access, Microsoft Project, and various database management systems.

**The Trade – Offs:**

Versions of Analysis Services prior to Analysis Services 2005 supported connecting to data sources through OLE DB providers only. SSAS 2008 and SSAS 2005 have much tighter integration with the .NET Framework and support connections via both OLE DB and .NET data providers.

**Data Source Views:**

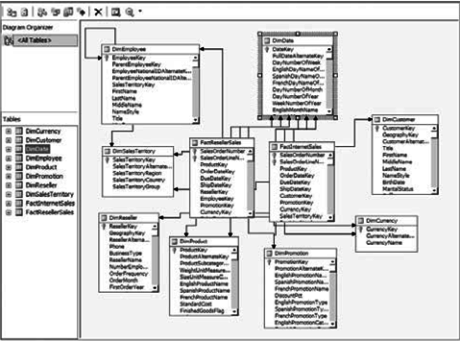
Data Source Views (DSVs) enable you to create a logical view of only the tables involved in your data warehouse design.

**DSV Wizard:**

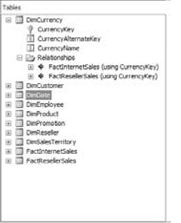
The DSV Wizard helped us to create a DSV by going through a few dialogs. We  need a data source to create a DSV. If we had not created a data source object in our database, the DSV Wizard allows us to create new data sources from the DSV Wizard’s Select a Data Source page by clicking the New Data Source button. In addition, the DSV Wizard also allows us to restrict specific schemas as well as filter certain tables, which helps us to work with only the tables we need to create our DSV.

**DSV Designer:**

The DSV Designer contains three panes. The center pane contains a graphical view of all the tables in the DSV and their primary keys. The relationships between tables are represented by lines with an arrow at the end. The top – left pane is called the *Diagram Organizer* , which is helpful in creating and saving concise views within large DSVs. When a DSV contains more than 20 tables it is difficult to visualize the complete DSV in the graphical view pane.

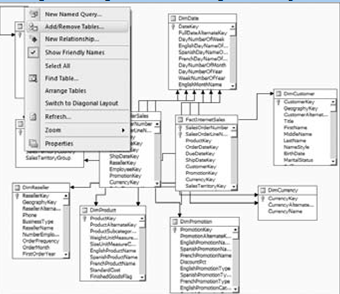
[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/111.png)

The lower – left pane of the DSV Designer is called Tables and is used to show the tree view of allthe tables of the DSV along with their relationships to other tables

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/112.png)

**Adding/Removing Tables in a DSV:**

The DSV Designer provides you with the capability to easily modify the DSV. To modify the existing tables, right – click the diagram view pane and select Add/ Remove Tables.

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/113.png)

Using this dialog we can add additional tables to the DSV by moving tables from the Available objects list to the Included objects list or remove existing tables by moving them from the Included objects list to the Available objects list. We  can also remove a table from the DSV in the DSV Designer in the graphical view pane or the table viewpane using the following steps:

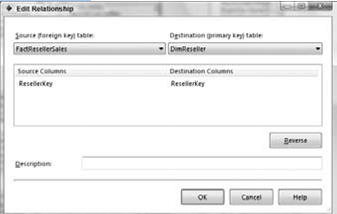
1. Select the table to be deleted.
2. Right – click the table and click Delete Table from DSV.
3. Click OK in the confirmation dialog that appears.

**Specifying Primary Keys and Relationships in the DSV:**

To specify the primary key(s) for a table, you need to do the following in the DSV Designer:

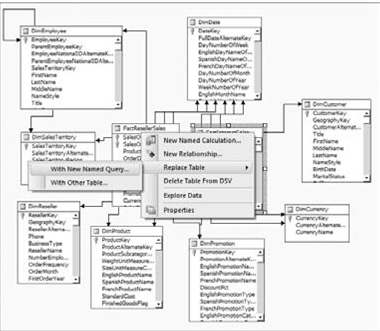
1. Select the column in the table that you want to specify as a primary key.
2. Right – click and select Set Logical Primary Key.
3. Select column A in Table1 that is involved in the join to Table2.
4. With column A selected, drag and drop it to column B in Table2. This forms a relationship between Table1 and Table2.

We  can also create a new relationship by right – clicking a table and selecting New Relationship.

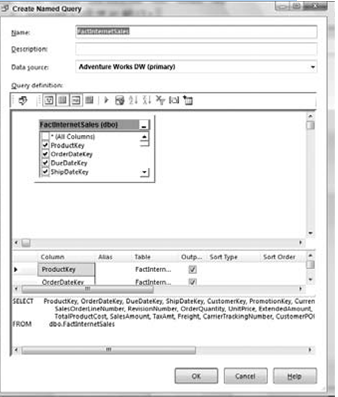
[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/114.png)

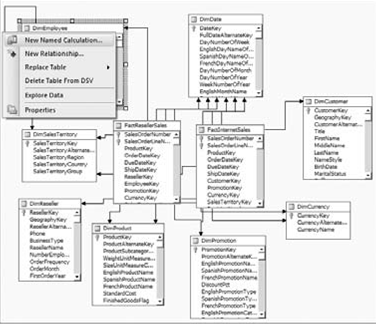
**Customizing Your Tables in the DSV:**

SSAS 2008 provides the functionality of performing all these particular operations within the DSV using a *Named Query. We* can invoke the Named Query editor by right – clicking a table and selecting Replace Table With New Named Query. If we want to add a specific table twice in Our DSV or add some columns of a new table, we can launch the query designer by right – clicking the DSV Designer and selecting With New Named Query.

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/115.png)

Execute the query to make sure the query is correct. The results from the underlying relational database will then be visible in a new pane beneath the SQL pane. Click OK once you have formed and validated your query. The table is now replaced with results from the Named Query you have specified in the DSV.

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/116.png)

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/117.png)

A new column is added to the Employee table. The data type of this calculated column will be determined based on the data types of the actual columns involved in the calculation or data used within the expression. If the expression results in a number, the data type for this column will be an integer.

The DSV maintains the calculated column of a table as a computed column in the metadata; it does not write it out to the underlying tables. When we want to view the data of this table the expression must be added to the SQL query so that we can see the data of this computed column.

**Data Source Views in Depth:**

Data warehouse designs consist of several fact tables and all the associated dimension tables. Small datawarehouses are usually comprised of 10 to 20 tables, whereas larger data warehouses can have morethan a hundred tables.

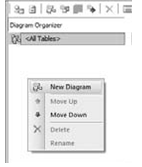
To create a new diagram, you need to do the following:

1. Right – click the Diagram Organizer pane and select New Diagram,

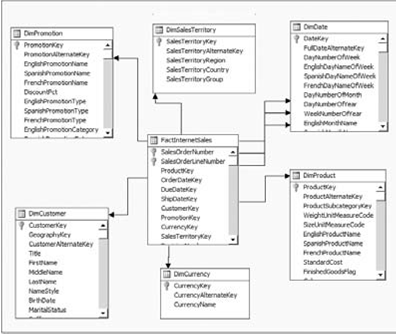
[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/118.png)

2. Name the new diagram “ Internet Sales. ”

3. We now have an empty diagram view. Right – click the diagram view and select Show Table. We  are presented with a dialog where we can choose the table(s) we want to include in this diagram.

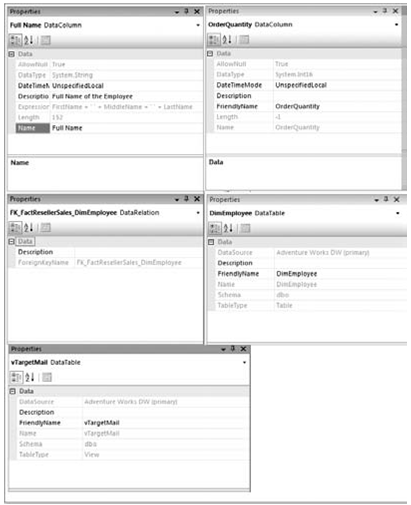
[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/119.png)

4. Select all the tables that are part of the InternetSales fact table and click OK. This gives you a diagram view of Internet Sales that contains the Internet Sales fact table and the related dimension tables.

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/120.png)

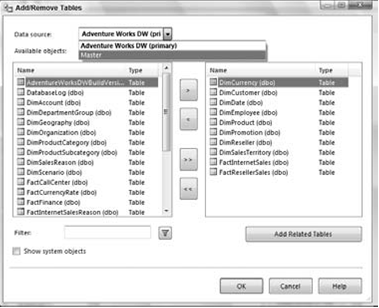
**Data Source View Properties|:**

Maintain properties. Within the DSV you can view the properties of the objects in the DSV such as tables, views, columns, and relationships. Properties of these objects are shown in the Properties window within BIDS.

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/121.png)

**Multiple Data Sources within a DSV:**

Once we have added the tables from multiple data sources to our DSV, we can start creating our cubes and dimensions as if these came from a single data source. The limitation that the primary data source needs to be a SQL Server is due to the fact that Analysis Services instance uses a SQL Server –specific feature called OPENROWSET to retrieve data from other data sources.

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/122.png)