MDX Query

Let's start by outlining one of the simplest forms of an MDX expression, bearing in mind this is for an outline of an expression returning two cube dimensions:

```
SELECT axis specification ON COLUMNS, axis specification ON ROWS
FROM cube_name
WHERE slicer_specification
```

The axis specification can also be thought of as the member selection for the axis. If a single dimension is required, using this notation, COLUMNS must be returned. For more dimensions, the named axes would be PAGES, CHAPTERS and, finally, SECTIONS. If you desire more generic axis terms over the named terms, you can use the AXIS(index) naming convention. The index will be a zero-based reference to the axis.
The slicer specification on the WHERE clause is actually optional. If not specified, the returned measure will be the default for the cube. Unless you actually query the Measures dimension (as will the first few expressions), you should always use a slicer specification.

The simplest form of an axis specification or member selection involves taking the MEMBERS of the required dimension, including those of the special Measures dimension:

```
SELECT Measures.MEMBERS ON COLUMNS,
[Store].MEMBERS ON ROWS
FROM [Sales]
```

This expression satisfies the requirement to query the recorded measures for each store along with a summary at every defined

summary level. Alternatively, it displays the measures for the stores hierarchy. In running this expression, you will see a row member named "All Stores." The "All" member is generated by default and becomes the default member for the dimension. The square brackets are optional, except for identifiers with embedded spaces, where they are required. The axis definition can be enclosed in braces, which are used to denote sets. (They are needed only when enumerating sets.)

In addition to taking the MEMBERS of a dimension, a single member of a dimension can be selected. If an enumeration of the required members is desired, it can be returned on a single axis:

```
SELECT Measures.MEMBERS ON COLUMNS,
{[Store].[Store State].[CA], [Store].[Store State].[WA]} ON ROWS
FROM [Sales]
```

This expression queries the measures for the stores summarized for the states of California and Washington. To actually query the measures for the members making up both these states, one would query the CHILDREN of the required members:

```
SELECT Measures.MEMBERS ON COLUMNS,
{[Store].[Store State].[CA].CHILDREN,
[Store].[Store State].[WA].CHILDREN} ON ROWS
FROM [Sales]
```

The following simple SELECT statement returns the measure Internet Sales Amount on the **Columns axis**, and uses the MDX MEMBERS function to return all the members from the Calendar hierarchy on the Date dimension on the **Rows axis**:

```
SELECT {[Measures].[Internet Sales Amount]} ON COLUMNS,
{[Date].[Calendar].MEMBERS} ON ROWS
FROM [Adventure Works]
```

The two following queries return exactly the same results but demonstrate the use of axis numbers rather than aliases:

```
SELECT {[Measures].[Internet Sales Amount]} ON 0,
{[Date].[Calendar].MEMBERS} ON 1
FROM [Adventure Works]
```

```
SELECT {[Measures].[Internet Sales Amount]} ON AXIS(0),
{[Date].[Calendar].MEMBERS} ON AXIS(1)
FROM [Adventure Works]
```

The simple example presented in this topic illustrates the basics of specifying and using query and slicer axes.

Examples

The following query does not include a WHERE clause, and returns the value of the Internet Sales Amount measure for **all** Calendar Years:

```
SELECT {[Measures].[Internet Sales Amount]} ON COLUMNS,
[Date].[Calendar Year].MEMBERS ON ROWS
FROM [Adventure Works]
```

Adding a WHERE clause, as follows:

```
SELECT {[Measures].[Internet Sales Amount]} ON COLUMNS,
[Date].[Calendar Year].MEMBERS ON ROWS
FROM [Adventure Works]
WHERE ([Customer].[Customer Geography].[Country].&[United States])
```

does not change what is returned on Rows or Columns in the query; it changes the values returned for each cell. In this example, the query is sliced so that it returns the value of Internet Sales Amount for all Calendar Years but only for Customers who live in the United States.

"Multiple members" from "different hierarchies" can be added to the WHERE clause. The following query shows the value of Internet Sales Amount for all Calendar Years for Customers who live in the United States and who bought Products in the Category bikes (=1)

```
SELECT {[Measures].[Internet Sales Amount]} ON COLUMNS,
[Date].[Calendar Year].MEMBERS ON ROWS
FROM [Adventure Works]
WHERE ([Customer].[CustomerGeography].[Country].&[United States],
[Product].[Category].&[1])
```

If you want to use "multiple members" from "the same hierarchy", you need to include a set in the WHERE clause. For example, the following query shows the value of Internet Sales Amount for all Calendar Years for Customers who bought Products in the Category Bikes and live in either the United States or the United Kingdom:

```
SELECT {[Measures].[Internet Sales Amount]} ON COLUMNS,
[Date].[Calendar Year].MEMBERS ON ROWS
FROM [Adventure Works]
WHERE (
{[Customer].[Customer Geography].[Country].&[United States]
, [Customer].[Customer Geography].[Country].&[United Kingdom]}
, [Product].[Category].&[1] )
```

As mentioned above, using a set in the WHERE clause will implicitly aggregate values for all members in the set. In this case, the query shows aggregated values for the United States and the United Kingdom in each cell.

You define the slicer specification with the WHERE clause, outlining the slice of the cube to be viewed. Usually, the WHERE clause is used to define the measure that is being queried. Because the cube's measures are just another dimension, selecting the desired measure is achieved by selecting the appropriate slice of the cube.

If one were required to query the sales average for the stores summarized at the state level, cross referenced against the store type, one would have to define a slicer specification. This requirement can be expressed by the following expression:

```
SELECT {[Store Type].[Store Type].MEMBERS} ON COLUMNS,
{[Store].[Store State].MEMBERS} ON ROWS
FROM [Sales]
WHERE (Measures.[Sales Average])
```

As stated, the slicer specification in the WHERE clause is actually a dimensional slice of the cube. Thus the WHERE clause can, and often does, extend to other dimensions. If one were only interested in the sales averages for the year 1997, the WHERE clause would be written as:

WHERE (Measures.[Sales Average], [Time].[Year].[1997])

It is important to note that slicing is not the same as filtering. Slicing does not affect selection of the axis members, but rather the values that go into them. This is different from filtering, because filtering reduces the number of axis members.

==========

The Cube

A cube, named TestCube, has two simple dimensions named Route and Time. Each dimension has only one user hierarchy, named Route and Time respectively. Because the measures of the cube are part of the Measures dimension, this cube has three dimensions in all.

The Query

The query is to provide a matrix in which the Packages measure can be compared across routes and times.

In the following MDX query example, the Route and Time hierarchies are the query axes, and the Measures dimension is the slicer axis. The Members function indicates that MDX will use the members of the hierarchy or level to construct a set. The use of the Members function

means that you do not have to explicitly state each member of a specific hierarchy or level in an MDX query.

```
SELECT { Route.nonground.Members } ON COLUMNS,
  { Time.[1st half].Members } ON ROWS
FROM TestCube
WHERE ( [Measures].[Packages] )
```

The result is a grid that identifies the value of the Packages measure at each intersection of the COLUMNS and ROWS axis dimensions. The following table shows how this grid would look.

	air	sea
1st quarter	60	50
2nd quarter	45	45

1) NON EMPTY keyword

The NON EMPTY keyword, used before the set definition, is an easy way to remove all empty tuples from an axis. For example, in the examples we've seen so far there is no data in the cube from August 2004 onwards. To remove all rows from the cellset that have no data in any column, simply add NON EMPTY before the set on the Rows axis definition as follows:

```
SELECT {[Measures].[Internet Sales Amount]} ON COLUMNS,
NON EMPTY {[Date].[Calendar].MEMBERS} ON ROWS
FROM [Adventure Works]
```

NON EMPTY can be used on all axes in a query. Compare the results of the following two queries, the first of which does not use NON EMPTY and the second of which does on both axes:

```
SELECT {[Measures].[Internet Sales Amount]} *
[Promotion].[Promotion].[Promotion].MEMBERS ON COLUMNS,
{[Date].[Calendar].[Calendar Year].MEMBERS} ON ROWS
FROM [Adventure Works]
WHERE([Product].[Subcategory].&[19])
```

```
SELECT NON EMPTY {[Measures].[Internet Sales Amount]} *
[Promotion].[Promotion].[Promotion].MEMBERS  ON COLUMNS,
NON EMPTY {[Date].[Calendar].[Calendar Year].MEMBERS} ON ROWS
FROM [Adventure Works]
WHERE([Product].[Subcategory].&[19])
```

2) HAVING CLAUSE

The HAVING clause can be used to filter the contents of an axis based on a specific criteria; it is less flexible than other methods that can achieve the same results, such as the FILTER function, but it is simpler to use.

Here is an example that returns only those dates where Internet Sales Amount is greater than $15,000:

```
SELECT [Date].[Calendar].[Date].MEMBERS
HAVING [Measures].[Internet Sales Amount]>15000   ON 0
FROM [Adventure Works]
```

Another solution:

```
SELECT {[Measures].[Internet Sales Amount]}   ON COLUMNS,
```

```
NON EMPTY {[Date].[Calendar].[Date].MEMBERS}
HAVING [Measures].[Internet Sales Amount]>15000   ON ROWS
FROM [Adventure Works]
```

3) FILTER function

FILTER function can be used to filter the contents of an axis based on a specific criteria, it is flexible than other methods that can achieve the same results, such as the HAVING clause, but it is simpler to use

```
SELECT {[Measures].[Internet Sales Amount]}   ON COLUMNS,
NON EMPTY FILTER ( {[Date].[Calendar].[Date].MEMBERS} ,
[Measures].[Internet Sales Amount]>15000 )  ON ROWS
FROM [Adventure Works]
```

4) TOPCOUNT function

Tìm 2 quyển sách có tổng số lượng lớn nhất.

```
Select topcount( [Dim Book].[Title].children,2,[Measures].[Quantity]) on 0
From [Bookstore WH];
```

Lưu ý, giả sử có 3 quyển sách với quyển đầu có tổng số lượng cao nhất, hai quyển kế tiếp có tổng số lượng như nhau thì khi chọn top 2 -> hàm Topcount chọn sách đầu tiên và quyển thứ 1 trong hai quyển kế tiếp mà không chọn tất cả 3 quyển. Mở rộng n quyển như nhau cũng cho cùng 1 kết quả.

5) GENERATE FUNCTION and CURRENTMEMBER method

Sử dụng generate và currentmember, cả 2 cách đều ra cùng kết quả.

vd: Tìm 3 quyển sách có số lượng lớn nhất của từng năm.

```
select [Measures].[Quantity] on 0,
non empty generate(
        [Dim Time].[year Sale].children,
        topcount([Dim Time].[year Sale].currentmember*[Dim
Book].[Title].children,3,[Measures].[Quantity])
        ) on 1
from[Bookstore WH];
```

CÁCH KHÁC

```
select [Measures].[Quantity] on 0,
non empty generate(
        [Dim Time].[year Sale].children,
        [Dim Time].[year Sale].currentmember*topcount([Dim
Book].[Title].children,3,[Measures].[Quantity])
        ) on 1
from[Bookstore WH];
```

Có thể bỏ dấu [ ].[ ]. giữa các Dimension và các Fields, Measures, chỉ dùng khi tên các bảng, field và tên độ đo có khoảng trắng ở giữa.