Module 02: Introduction to Multidimensional Data Expressions (MDX)

Course Number: HSQL-1018
Level: 200
Duration: 24 hours
Conditions and Terms of Use

HDS Training Portals LLC and Affiliates

The contents of this student manual and all associated videos are for informational and training purposes only and are provided "as is" without warranty of any kind. Training content, including URL's and other Internet Web site references, is subject to change without notice. A registered student is anyone who is a registered student. The term “registered student” is defined below.

Copyright and Trademarks

@ [2008-1010] HDS Training Portals LLC, and Affiliates - all rights reserved

No part of this training material may be recorded or reproduced without permission. Registered students may print a single copy of the courseware, labs, and outlines as needed. It is illegal to re-use any part of this training or associated videos outside of this course.

Registered Trademarks of the Microsoft Corporation used in this course


Registered Student Definition

A registered student is someone who obtained these course materials by purchasing them through HDS Training Portals, or an affiliate. Further, a registered student is anyone who works for, or is associated with, an organization which obtained these course materials by purchasing them through HDS Training Portals, or an affiliate. It is illegal to use these materials if you are not a registered student. Furthermore, use of these materials to learn or train others is illegal without purchasing the training through HDS Training Portals or an affiliate.

Authorization If you have any questions about whether you are authorized to use these training materials, or to report illegal usage of these materials, email: customersupport@hdstraining.com

Copyright Responsibility and Liability

Complying with all applicable copyright laws is the responsibility of the user.

With the exception of the printing authorization given to registered students via the rights under copyright, no part of this document may be reproduced without the express written permission of HDS Training Portals, LLC and affiliates.
# Table of Contents

Module 2: Introduction to Multidimensional Data Expressions (MDX) ................................................. 4  
  MDX Query Basics (Tuples, Sets, Select) .......................................................................................... 5  
  The MDX Data Framework (Tuples and Sets) .................................................................................. 5  
  MDX Axis Framework – Names and Numbering ............................................................................. 10  
  MDX Axes framework ...................................................................................................................... 10  
  Key Concepts in SSAS/MDX Environments ..................................................................................... 14  
  Introduction to SSAS Components .................................................................................................... 14  
  MDX Syntax Elements ....................................................................................................................... 18  
  MDX Syntax Elements ....................................................................................................................... 18  
  MDX Expressions (introduction only) ............................................................................................... 21  
  Introduction to Member, Tuple and Set Functions ........................................................................... 22  
  MDX Operators ................................................................................................................................ 30  
  MDX Operators ................................................................................................................................ 30  
  Using MDX Operators in Queries ..................................................................................................... 34  
  MDX Operators in Queries ................................................................................................................ 34  
  Introduction to SET Functions ........................................................................................................... 40  
  Overview of MDX SET Functions ...................................................................................................... 40  
  .Children and .Members Functions ................................................................................................... 43  
  The Members Function ...................................................................................................................... 43  
  Descendants ....................................................................................................................................... 46  
  Introduction to Dimension Hierarchies ............................................................................................. 46  
  The Descendants Function ................................................................................................................. 47  
  Module 1 Lab .................................................................................................................................... 51  
  Exercise 1: Working with Axes, Tuples, and Sets ........................................................................... 51
Module 2: Introduction to Multidimensional Data Expressions (MDX)

MDX stands for Multidimensional Data Expressions. It’s used to query OLAP (online analytical Processing) cubes.

In this module, you’ll be introduced to the MDX language and some of the common language elements and functions.

Objectives

Understand and be able to implement the follow MDX concepts and functions

- Introduction to MDX
- Axis Framework – Names and Numbering
- Query Basics
- Simple MDX Query Construction
- .Members
- .Children
- .Descendants
- .Self_and_Before
- The Query Slicer
- Removing Empty Slices
- Tuples
- CrossJoin()
- Order()
- Querying Member Properties
MDX Query Basics (Tuples, Sets, Select)

Student Activities Required for this Lesson:

- Watch the following recording from the class portal:

  ![Movie: Mod 2.1 MDX Tuples and Sets](Image)

- Read this lesson.
- Read the following links:
  
  

Multidimensional Data Expressions (MDX)

Before you can dive head first into MDX, you need to know a little bit about its intended purpose. MDX is a query language which is used to retrieve and update (in some cases) data in a multi-dimensional structure like a SQL Server Analysis Services cube or a Hyperion cube.

Unlike Transact SQL which can only return two axes of data (rows and columns), MDX can return sets of data based on up to 128 dimensional axes. MDX supports expressions, functions, conditional statements, data types (variant), and operators.

The MDX Data Framework (Tuples and Sets)

The MDX query is based on an MDX SELECT statement. Before you can think about querying multidimensional data, you need to have a basic understanding of **sets** and **tuples**. The whole MDX data model is built around sets and tuples. We’ll be covering tuples and sets later in detail, so we’ll just introduce the two objects now so that you can understand how a basic MDX query works.
MDX Tuples

A **tuple** is a combination of members from one or more dimensions. A tuple returns a single cell of data. Tuples can be combined into sets. The job of the tuple is to completely define the cell of data. A tuple can be simple, containing one member from a single dimension. A tuple can also be more complex, containing members from multiple dimensions.

In MDX, and mathematics, tuples are delimited with parenthesis ( ). When a tuple has more than one dimension referenced, it can have only one member from each dimension. Tuples cannot contain other tuples as members. Below is a set containing two tuples. The { } marks the set, the two sets of ( ), marks each tuple.

```
{ -- Set open
  --Two tuples delimited by parenthesis
  ([Measures].[Internet Sales Amount], [Date].[Fiscal].[Fiscal Year].&[2007]),
  ([Measures].[Internet Tax Amount],[Date].[Fiscal].[Fiscal Year].&[2007])
} -- Set close
```

A Tuple Example

Below you'll see an example of a query that uses two complex tuples on the columns axis (0) because each tuple contains more than one dimension. They're called complex tuples when more than one dimension is used. The tuples are delimited by parenthesis ( ), and separated by commas. One of the most important things to learn as an MDX coder is the ability to convert a business requirement into MDX. The construction of the correct tuples for each report is a good place for you start. Once you have the correct tuples, you can save them as calculated members for re-use.

**First tuple:** Internet Sales for 2007

This tuple stores a cell value that represents all internet sales (SUM) for 2007.

```
([Measures].[Internet Sales Amount], [Date].[Fiscal].[Fiscal Year].&[2007])
```

**Second Tuple:** Internet Tax for 2007

This tuple stores a cell value that represents all internet sales (SUM) for 2007.

```
([Measures].[Internet Tax Amount],[Date].[Fiscal].[Fiscal Year].&[2007])
```
Select
{
  ([Measures].[Internet Sales Amount], [Date].[Fiscal].[Fiscal Year].&[2007]),
  ([Measures].[Internet Tax Amount],[Date].[Fiscal].[Fiscal Year].&[2007])
} on 0,
{[Product].[Category].&[1]} on 1
from [Adventure Works]

Results: (product category 1 is bikes).

<table>
<thead>
<tr>
<th></th>
<th>Internet Sales Amount</th>
<th>Internet Tax Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bikes</td>
<td>$5,762,134.30</td>
<td>$460,370.78</td>
</tr>
</tbody>
</table>

**Note:** This will serve as an introduction to tuples so that you can be introduced to select statements. You’ll be working with tuples in lots of ways during this class.

**MDX Sets**

A set is an ordered collection of tuples. A set can have one tuple, hundreds of tuples, or thousands of tuples in it. A set can also be empty. Mathematical sets are not allowed to contain the same tuple more than once. That limitation does not exist with MDX. You can have the same tuple more than once, possibly for comparison reasons.

Most languages (VB, C#...) would refer to a set a “sequence” or a “collection”. In MDX, they’re called sets.

Sets can be query scoped, session scoped, or permanently named and included in the cube schema for all to see and use. The most common use of sets is to include them as the rows or columns axis in the query.
A Set Example

This is the same example you looked at for the tuple example. This time we'll focus on the sets.

This query uses two sets, one on columns (axis 0), and the other on rows (axis 1).

**First Set:**

```
{
([Measures].[Internet Sales Amount], [Date].[Fiscal].[Fiscal Year].&[2007]),
([Measures].[Internet Tax Amount],[Date].[Fiscal].[Fiscal Year].&[2007])
}
```

**Second Set:**

```
{[Product].[Category].&[1]}
```

**Query Example**

```
Select
{
([Measures].[Internet Sales Amount], [Date].[Fiscal].[Fiscal Year].&[2007]),
([Measures].[Internet Tax Amount],[Date].[Fiscal].[Fiscal Year].&[2007])
} on 0,
{[Product].[Category].&[1]} on 1
from [Adventure Works]
```

**Results:** (product category 1 is bikes).

<table>
<thead>
<tr>
<th></th>
<th>Internet Sales Amount</th>
<th>Internet Tax Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>FY 2007</td>
<td>$5,762,134.30</td>
<td>$460,370.78</td>
</tr>
</tbody>
</table>

**Note:**

This will serve as an introduction to sets so that you can be introduced to select statements. You'll be working with sets in lots of ways during this class.
MDX Select to SSAS

In MDX, the SELECT statement is sent to the cube for processing. SSAS handles the query by forming a logical sub-cube of multi-dimensional data that’s based on the axes requested. Then, the query is satisfied by returning data from the subcube.

SSAS has two distinct query engine components. The formula engine builds the subcube and handles the query. The storage engine fetches data from disk. If the subcube required for the SELECT statement is already cached, then the formula engine isn’t used in the query processing. This is the key to fast cubes.

Separating the SSAS engines allows for efficient caching of sets for subsequent queries of the same data. The exact process of how an MDX query is handled on the cube side may vary from vendor to vendor, however that’s the way SSAS handles the request.

The SELECT statement contains:

- The sets and tuples on each query axis.
- The sets and tuples on the slicer axis.
- The number of query axes returned by the query. In MDX you have up to 128 axes available to the query to aggregate on. In TSQL, its only two axes: rows and columns.
- The name of the cube or perspective that the query is solved from.

MDX SELECT Example

The following query uses two query axes (rows (1), and columns (0). On rows you have two measures, Sales Amount and Tax Amount. On rows, you have product categories 3 and 4. The slicer axis restricts the aggregations to only sales that were delivered in 2008.

```sql
SELECT
    {[Measures].[Sales Amount],
     [Measures].[Tax Amount]} ON 0,
    {[Product].[Category].&[3],[Product].[Category].&[4]} ON 1
FROM [Adventure Works]
WHERE ([Delivery Date].[Fiscal].[Fiscal Year].&[2008])
```

Results:

<table>
<thead>
<tr>
<th></th>
<th>Sales Amount</th>
<th>Tax Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clothing</td>
<td>$1,269,045.12</td>
<td>$101,523.60</td>
</tr>
<tr>
<td>Accessories</td>
<td>$1,050,729.86</td>
<td>$84,058.39</td>
</tr>
</tbody>
</table>
MDX Axis Framework – Names and Numbering

Student Activities Required for this Lesson:

- Watch the following recording from the class portal:
  
  Movie: Mod 2.2 Axis Framework and Numbering.

- Read this lesson.
  
  Read the following links:
  

MDX Axes framework

The MDX query engine takes two types of statements; queries and expressions. The MDX query is broken down into two parts, the query axes, and the slicer axis.

Query axes

Query axes are used in the MDX SELECT statement to return a set of members to the client application. An MDX query can contain up to 128 query axes. The query axes are numbered 0-127. Query axis 0 is columns, and query axis 1 is rows. Read the first link above to learn about rules for axis numbering and to see some examples.

The following statement returns the Internet Sales Amount measure on columns (axis 0) and each member of the date dimension on rows (axis 1).

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

Results (partial result set in the screen shot)

<table>
<thead>
<tr>
<th>Period</th>
<th>Internet Sales Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Periods</td>
<td>$29,359,677.22</td>
</tr>
<tr>
<td>CY 2005</td>
<td>$3,266,373.65</td>
</tr>
<tr>
<td>H2 CY 2005</td>
<td>$3,266,373.65</td>
</tr>
<tr>
<td>Q3 CY 2005</td>
<td>$1,453,522.89</td>
</tr>
</tbody>
</table>
Slicer axis

The slicer axis is used to filter the data that gets returned by the MDX query. Members are only returned that intersect the slicer. The slicer axis is optional, and is defined by the tuple or set expression in the WHERE clause of the SELECT statement.

The slicer below restricts results of the other axes to sales of product category 1 in Germany and Australia.

```sql
WHERE
( [
  [Geography].[Country].&[Australia],
  [Geography].[Country].&[Germany]
  ,
  [Product].[Category].&[1]
]
)
```

The slicer can take
- A tuple expression (a single member)
- A set expression (a list of members)

TSQL Where clause versus MDX slicers

- The where clause in MDX does not filter what is returned on the rows axis.
- MDX employs the FILTER function on the rows axis to restrict the rows that are returned.
Slicer Example 1

The following example uses a tuple to define the slicer. The tuple consists of Germany and Australia from the geography dimension, and product category 1 from the product dimension. Any sales outside of that tuple won’t be aggregated in the results.

```sql
SELECT 
  {[Measures].[Internet Tax Amount]} ON COLUMNS, 
  [Date].[Calendar Year].MEMBERS ON ROWS
FROM [Adventure Works]
WHERE 
  ( 
    {[Geography].[Country].&[Australia],[Geography].[Country].&[Germany] }, 
    [Product].[Category].&[1] 
  )
```

Results:

<table>
<thead>
<tr>
<th>Year</th>
<th>Internet Tax Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Periods</td>
<td>$2,265,451.62</td>
</tr>
<tr>
<td>CY 2005</td>
<td>$261,309.90</td>
</tr>
<tr>
<td>CY 2006</td>
<td>$522,427.50</td>
</tr>
<tr>
<td>CY 2007</td>
<td>$748,728.24</td>
</tr>
<tr>
<td>CY 2008</td>
<td>$732,385.99</td>
</tr>
<tr>
<td>CY 2010</td>
<td>(null)</td>
</tr>
</tbody>
</table>
Slicer Example 2

The following example uses a slightly different slicer from the previous example. Now the slicer shows the aggregated **Internet Tax Amount** for Canadian customers, who bought product category 1, and are home owners. Notice the number of rows is the same because you still have the same number of members returned on rows.

You'll learn about functions like FILTER and TOPCOUNT that can be used on the rows axis to restrict the number or rows returned.

```mdx
SELECT
    {[Measures].[Internet Tax Amount]} ON COLUMNS,
    [Date].[Calendar Year].MEMBERS ON ROWS
FROM [Adventure Works]
WHERE
    ([Geography].[Country].&[Canada],
    [Product].[Category].&[1],
    [Customer].[Home Owner].&[1])
```

Results:

<table>
<thead>
<tr>
<th>All Periods</th>
<th>Internet Tax Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>CY 2005</td>
<td>$172,920.33</td>
</tr>
<tr>
<td>CY 2006</td>
<td>$398,070.45</td>
</tr>
<tr>
<td>CY 2007</td>
<td>$526,851.83</td>
</tr>
<tr>
<td>CY 2008</td>
<td>$515,208.06</td>
</tr>
<tr>
<td>CY 2010</td>
<td>(null)</td>
</tr>
</tbody>
</table>
Key Concepts in SSAS/MDX Environments

Student Activities Required for this Lesson:

- Watch the following recording from the class portal:

  Movie: Mod 2.3 Key Concepts in SSAS/MDX Environments

- Read this lesson.

- Read the following links: http://msdn.microsoft.com/en-us/library/ms144884.aspx

Introduction to SSAS/MDX Components

If you are familiar with SQL Server Analysis Services (SSAS), then this first topic will be a review. It’s intended for students who are not very familiar with SSAS. To gain a good knowledge and hands-on expertise with SSAS 2008, you can take the HSQL-1007 Course: Designing and Implementing SQL Server 2008 Analysis Services Solutions.

For now, take a few moments and read the above link to get introduced to some of the terms and definitions that you’ll come across in this class.
Key MDX/SSAS Terms

SSAS cubes contain measures, dimensions, and dimension attributes. The terms below help define the SSAS terms and concepts that you should understand for this course.

More Information: For more information on Key MDX Terms: see the link at the beginning of the topic: http://msdn.microsoft.com/en-us/library/ms144884.aspx

**Dimensional Modeling:** Dimensional Modeling is a database modeling technique which uses fact and dimension tables. The model is based on a star, snowflake, or hybrid join model. The fact tables only contain the measures that are being aggregated by the cube, and the join keys for the cube dimensions. The cube dimensions are based on all the different business entities that you would like to slice the fact table measures with. For example, a data mart with a **FactInternetSales** table would probably have dimensions of **DimProduct** (the products sold), **DimTime** (order data, ship date), **DimCustomer** (the customer), **DimPromotion** (the sales promotion). No measures are kept in the dimensions. The **AdventureWorksDW** database is an example of a dimensionally modeled database. The reason this is called a star schema is because the PK-FK relationships start at the fact table in the middle and point out to the ring of dimension tables. The topology resembles a star.

**SSAS Database:** *An SSAS Database is the product of an SSAS Project which has been deployed to an SSAS Server.*

**SSAS Cube:** An SSAS cube is a collection of measures, measure groups, dimensions, dimension attributes, actions, calculated members, scripts, named sets, and much more. The cube is the object which is consumed by an OLAP reporting clients for high speed AD-HOC query performance.

**Attribute hierarchy:** An attribute hierarchy is a reporting drill down feature of Unified Dimensional Models (UDM’s).

**Attribute relationship:** In Analysis Services, all dimensional attributes need to be related in some way to the key attribute of the dimension. Reporting hierarchies can be optimized for speed by configuring various parent-child relationships which better match the way the data is aggregated. Relationships can be 1: Many, Many: 1, or Many: Many.

**Balanced hierarchy:** A balanced hierarchy is a hierarchy in which the same number of levels exists between the top level and any leaf member.

**Ragged Hierarchy:** A ragged hierarchy is a hierarchy which may contain a different number of levels for different leaf members. For example, an organization chart may contain a different number of management levels for various jobs.

**Cube Measure:** In SSAS cubes, the measures are numbers that are based on numeric values from the fact tables. After processing, the measures are aggregated based on the cube dimension attributes and attribute hierarchies and stored in a measure group partition.
**Calculated Member:** A calculated member is an OLAP developer-defined calculation which is based on a cube measure or another calculated member. A simple calculated member would be the subtraction of cost from sales price to produce profit. Calculated members are calculated at run time and therefore increase the query time when working with large sets. However, their flexibility usually out-weighs the performance hit that may arise. You’ll learn about how to optimize your MDX later.

**Cube dimension:** A cube dimension is an instance of an SSAS database dimension which has been included in a cube definition. Not all cubes use all dimensions. So when the dimension is used in a cube it’s called a cube dimension.

**Cube space:** Cube space is the final dimensional structure which is used to handle the query. This can be formulated at query time by the SSAS formula engine, or it can be explicitly defined in you MDX code.

**Data member:** A data member is the unique child values in a parent child hierarchy. For example a data member in the month level a data hierarchy would be December.

**Database dimension:** A database dimension is a dimension that has been defined in the OLAP database, not related to any measure groups in the cube.

**Dimension attribute:** A dimension attribute is a facet of the dimensional data. For example, a customer dimension might have the following attributes:

- CustomerKey (used for joining to the fact table and usually hidden from the report user)
- CustomerName
- CustomerAltKey
- CustomerRegion
- CustomerCode

**Granularity attribute:** The granularity attribute represents the grain of detail that can be reported on for the associated fact or measure. It’s always in regard to a particular dimension. For example, a shipping cost measure from a sales fact table might be linked to the time dimension with a granularity attribute of day. The time dimension has a hierarchy of Year ➔ Semester ➔ Quarter ➔ Month ➔ Day. This means that you can report and aggregate shipping costs by the day attribute, and all levels above.

If the granularity attribute was set to month, you’d be able to total by month and higher, but not day. Generally, you would want to select the lowest granularity which has reporting value for the granularity attribute. This gives you the reporting levels of detail that you need without creating excessive aggregations at processing time.

**Key Attribute:** The key attribute of a dimension is the attribute which is used to join to the fact table rows. All other non-key attributes have to be related either directly or indirectly to the key attribute in order to be used to slice fact measures.
**Leaf member:** A leaf member is the actual attribute which represents the lowest level of the hierarchy.

**Measure:** A measure is numeric value from a fact table which is aggregated by the cube dimensions.

**Measure Group:** A measure group is a collection of measures. A measure group is usually based on a fact table, but can also be based on a view.

**Member:** A member is a value of a dimension attribute. A member of the month attribute would be December and there would be 12 members.

**Parent member:** Every parent-child hierarchy (P-C hierarchy) has two members; the parent and the child. The parent member holds the aggregated values for all the children below it. A child member is a direct descendant of the parent member. For example year is parent to semester.
MDX Syntax Elements

Student Activities Required for this Lesson:

- Watch the following recording from the class portal:

  Movie: Mod 2.4 MDX Syntax Elements

- Read this lesson.


  Spend about 2-3 minutes on each link in the table. You just want to get an introduction to each main element.

MDX Syntax Elements

MDX is the language used to query multi-dimensional OLAP structures and cubes. In the following paragraphs, you’ll be introduced to the primary groups of language elements. Each element has many different MDX items associated with it. It's fair to say that most OLAP solutions that are popular today don’t require a heavy understanding of MDX to query the data. However, the more you know about MDX, the more you can customize and optimize your queries. It’s equally fair to say that without a thorough understanding of MDX, you’ll be at a disadvantage working with SSAS and most OLAP clients.

Tip:

If you’re serious about your MDX, this is a must have book:


MDX Identifiers

Whenever you reference an SSAS object with MDX, you have to reference it by its identifier. Think of an identifier as an object name for a dimension, member, attribute, measure, etc. If you’ve done any development in an object-based language like TSQL, VB, C#, you’ve come across object names and ID’s. MDX is no different. Each object has a unique object name in MDX. That name is an identifier.
Regular Identifiers naming rules

When the OLAP developer creates the SSAS cube and objects, it's important to follow legal MDX identifier rules when naming dimensions, hierarchy levels, attributes, measures and any other object that can be called through the MDX query.

Formatting Rules for Regular Identifiers

1. The first character must be one of the following:
   - A Unicode V2 letter (Any letter in the installed locale)
   - an underscore (_)

2. Subsequent characters can be:
   - A Unicode V2 letter
   - an underscore (_)
   - A number

3. An identifier cannot be an MDX reserved Keyword (reserved for metadata)


4. The identifier cannot contain spaces

Regular Identifier Examples

The query example below can be run without MDX delimiters. MDX delimiters are brackets []. Brackets are not required around the identifiers that are named within MDX rules. Notice that [Adventure Works] (the cube name) is encased in brackets because of the space in the name. Sometimes you have no control over names that you inherit. However, it's a best practice not to use spaces in object names when developing your MDX solution.

```
SELECT Measures.MEMBERS ON COLUMNS,
Product.Style.CHILDREN ON ROWS
FROM [Adventure Works]
```
Optional Delimited Identifiers

The example below, the Measures, Product, and Style regular identifiers have been delimited by using brackets. There’s nothing wrong with using delimiters with all identifiers. This happens automatically when you generate scripts or drag/drop items from the metadata browser in the SSMS query tool. The [Adventure Works] identifier still needs brackets, the others are optionally. When you dynamically generate MDX, it’s best to use delimiters because you can’t be sure what the values of the data will be.

```mdx
SELECT [Measures].MEMBERS ON COLUMNS,
[Product].[Style].CHILDREN ON ROWS
FROM [Adventure Works]
```

**Tip:** In Analysis Services, the name of the item is directly exposed to the business user, so the identifiers need to make sense to them. For that reason, you’ll come across spaces in names more often than not. You should get in the habit of delimiting your identifiers in MDX.

Required Delimited Identifiers

In the following MDX statement, All Measures, Test Cube, and Select identifiers require brackets (delimiters). The All Measures and Test Cube identifiers require delimiters because of the spaces. The select identifier is using a reserved keyword (select).

```mdx
SELECT Measures.[All Measures]
FROM [Test Cube]
WHERE Customers.[select]
```

**Tip:** To summarize, whenever you have SSAS object names that don’t meet MDX naming conventions, surround the identifiers with brackets to avoid MDX syntax errors.
MDX Expressions (introduction only)

Note: We’ll be building MDX Expressions throughout the class. This topic is only to introduce MDX expressions and an MDX language element. You’ll get plenty of MDX expression practice, so don’t try to learn how to develop them now. That’s coming.

Below is an example of an MDX Expression which is being used as calculation later in the query. This simple expression shows a calculated discount amount (1.8 * the current discount level). The slicer axis (the where clause) is limiting the aggregation scope to only category 3 products (clothing).

WITH
    MEMBER [Measures].[Special Discount] AS
    [Measures].[Discount Amount] * 1.8
SELECT
    [Measures].[Special Discount] on COLUMNS,
    NON EMPTY [Product].[Product].MEMBERS ON Rows
FROM [Adventure Works]
WHERE [Product].[Product Categories].[Category].&[3]

MDX Expressions can be complex or simple. Below is a list of different types of expressions that can be created in MDX. We’ll cover creating complex expressions as we move through the course.

- Cube and Subcube Expressions
- Dimension Expressions
- Member Expressions
- Tuple Expressions
- Set Expressions
- Scalar Expressions

Note: At this point in the class, we’re just introducing MDX language elements. MDX expressions are a language element. Members, Tuples, Dimensions, Scalar functions, and subcubes are coming up in details shortly.
Introduction to Member, Tuple and Set Functions

Student Activities Required for this Lesson:

- Watch the following recording from the class portal:

  Movie: Mod 2.5 Introduction to Member, Tuple, and Set Functions

- Read this lesson.

- Read the following links:

Just have a quick browse of the different types of functions supported by SSAS. We’ll cover most of the functions later in detail.

Tuples and Sets

Since MDX is such a mathematically-oriented language, it just makes sense that its most basic concepts (members, sets, and tuples) are derived from mathematical theory.

A **tuple** is a sequence (or ordered list) of elements. It can be one element or several. The list is ordered with integers starting at 0. In MDX, the **Item(0)** function is used to return the first tuple in the set.

```sql
SELECT
    -- Curly Brace denotes the start of a set
    {  
        -- Tuple member 1
        ([Measures].[Internet Freight Cost], [Date].[Calendar Year].[CY 2006])
        -- Tuple member 2
        , ([Measures].[Internet Freight Cost], [Date].[Calendar Year].[CY 2007])
    }[Item(0)]
    -- Closing the curly brace denotes the end of a set
    -- The Item(0) function returns the first item in the set
ON COLUMNS
FROM [Adventure Works]
```
Tuples as Set Members

Each member in a set is a **tuple**. A tuple is the intersection of one or more items from the cube dimensions (including measures from the measures dimension). The product of that intersection is a single value. That value may be selected directly or included in a set.

- If a set is made up of all the products from the products dimension, then each tuple in the set will be a unique product from the products dimension (all members)

\{
  \{\[Product\].Members\}\}

  o  **Product Dimension**

- If the set is made up freight cost for 2006. The aggregation total will be less than all members because you’re only returning the freight costs for orders shipped in calendar year 2006.

\{
  \{\[Measures\].[Internet Freight Cost],[Date].[Calendar Year].[CY 2006]\}\}

  o  **FactInternetSales** (Internet Freight Cost)

  o  **Date dimension** (2006)

Example of two tuples forming a set:

```sql
Select 
[Measures].[Internet Sales Amount] on 0,
{ 
  ([Product].[Product Categories].[Category].&[1],[Date].[Fiscal Year].&[2008]),
  ([Product].[Product Categories].[Category].&[1],[Date].[Fiscal Year].&[2007])
} on 1
from [Adventure Works]
```

<table>
<thead>
<tr>
<th>Bikes</th>
<th>FY 2008</th>
<th>$15,483,926.11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bikes</td>
<td>FY 2007</td>
<td>$5,762,134.30</td>
</tr>
</tbody>
</table>
Calculating Values in MDX versus TSQL

Calculating aggregations in relational databases with TSQL is possible if all items in the set exist in the same database or data mart. However, it will take more resources and time (considerably more in some cases). Both TSQL and MDX have their strong points with regard to returning data.

- TSQL is great at returning tabular data on normalized tables. However it’s less efficient for aggregating numbers across large data sets than SSAS and MDX. The reason is that SSAS pre-aggregates the totals based on cube structure and then stores the answers in a cube partition.

- MDX is great at aggregating measures across huge, complex data sets, however when drilling down to row detail, it can be less efficient than TSQL unless the OLAP storage mode is set to MOLAP. MOLAP stores detail data in the cube partition, which is very fast. This way, the data mart does not need to be queried for drill down data because it’s stored in the cube.

MDX Tuple Example

The following tuple would be the SUM of all Internet sales for clothing sold in the year 2008. The reason that the sum is calculated (versus average, count or another aggregation), is because SSAS has the aggregation type for this measure set to SUM.

```
Select
{
    [Measures].[Internet Sales Amount],
    [Date].[Calendar].[Calendar Year].&[2008],
    [Product].[Product Categories].[Category].&[3]
} on 0
from [Adventure Works]
```

| Internet Sales Amount | CY 2008 | Clothing | $201,524.64 |

Note: Notice that the tuple is a single value that’s the intersection of sales amount, 2008, clothing (category 3).
Also notice that the tuple is enclosed in ( ).
Using the Members Set Function

By simply applying the members function to calendar year, you retrieve the same tuple for all years. In other words, you get one tuple in the set for each year. Each tuple is the intersection of Internet Sales Amount, Clothing, and the calendar year. This tuple is then laid over the columns axis.

```
SELECT
  
  {[Measures].[Internet Sales Amount],
   [Date].[Calendar].[Calendar Year].members,
   [Product].[Product Categories].[Category].&[@3]
  )
  on 0
  from [Adventure Works]
```

In the example below, two tuples are put into a set. Sets are enclosed in curly braces `{Set...}`, and can be simple or very complex.

The first tuple in the set is the total freight costs for 2006. The second tuple in the set is total freight costs for 2007. The set is ordered. The first tuple in the set is read by requesting `{Set...}.Item (0), Item (1), etc.

```
SELECT
  
  --Curly Brace denotes the start of a set
  
  --Tuple member 1
  
  ([Measures].[Internet Freight Cost]
  ,[Date].[Calendar Year].[CY 2006]
  )

  --Tuple member 2
  
  ([Measures].[Internet Freight Cost]
  ,[Date].[Calendar Year].[CY 2007]
  )

  .Item(0)

-- Closing the curly brace denotes the end of a set
-- The Item(0) function returns the first item in the set
ON COLUMNS
FROM [Adventure Works]
```
Member Functions

MDX provides several functions that return or act on members. The link below is a list of the MDX functions (including member functions), along with a brief description of what each function does. You’ll be spending a lot of time in this link during the class, so save yourself for the long haul. The more time you spend in here looking at syntax and testing queries, the better you’ll get at MDX functions. It’s the key to learning the language.

Note: This link is the MDX Function Reference. You’ll notice that there are a lot of functions. The best thing you can do is spend time practicing with them. Start with the samples. Then customize to meet your needs. There’s no way you’ll learn them all during this class. So it’s very important to know where to look them up.


Member Function example (.firstchild)

The example below uses the firstchild function to return the first year in reported business. The calendar year was 2005 (CY 2005).

--Example 2 calls the first child of the calendar year by function
SELECT [Date].[Calendar Year].FirstChild on 0
FROM [Adventure Works]
Tuple Functions

The following example displays one calculated value for each tuple in a set of tuples.

The *Internet Sales Amount* for each *State-Provence* in *United States* and its percent of the total *Internet Sales Amount* for the *United States* will be displayed. Tuples are delimited by parenthesis. States with no sales will show as (null). The states with 0% had sales but, the sales totaled less than .5% causing the value to be rounded down to 0%.

```plaintext
WITH MEMBER Measures.Percent_of_Total AS
    [Measures].[Internet Sales Amount] / 
    ( [Measures].[Internet Sales Amount],
      Ancestors
      ( [Customer].[Customer Geography].CurrentMember,
        [Customer].[Customer Geography].[Country]
      ).Item (0)
    ), FORMAT_STRING = '0%'
SELECT {{[Measures].[Internet Sales Amount],Measures.Percent_of_Total} ON 0,
    { Descendants
      ([Customer].[Customer Geography].[Country].&[United States],
        [Customer].[Customer Geography].[State-Provence], SELF
      )
    } ON 1
FROM [Adventure Works]
```

<table>
<thead>
<tr>
<th>State</th>
<th>Internet Sales Amount</th>
<th>Percent of Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>$37,29</td>
<td>0%</td>
</tr>
<tr>
<td>Arizona</td>
<td>$2,104.02</td>
<td>0%</td>
</tr>
<tr>
<td>California</td>
<td>$5,714,257.69</td>
<td>61%</td>
</tr>
<tr>
<td>Colorado</td>
<td>(null)</td>
<td>(null)</td>
</tr>
<tr>
<td>Connecticut</td>
<td>(null)</td>
<td>(null)</td>
</tr>
<tr>
<td>Florida</td>
<td>$7,760.91</td>
<td>0%</td>
</tr>
<tr>
<td>Georgia</td>
<td>$1,858.32</td>
<td>0%</td>
</tr>
<tr>
<td>Idaho</td>
<td>(null)</td>
<td>(null)</td>
</tr>
<tr>
<td>Illinois</td>
<td>$2,828.09</td>
<td>0%</td>
</tr>
<tr>
<td>Indiana</td>
<td>(null)</td>
<td>(null)</td>
</tr>
<tr>
<td>Kentucky</td>
<td>$216.96</td>
<td>0%</td>
</tr>
<tr>
<td>Maine</td>
<td>(null)</td>
<td>(null)</td>
</tr>
</tbody>
</table>
The MDX Tuple Item Function

The **item** function can be used to extract the first tuple from set. The following example returns the first tuple from a set of tuples on the column axis. The set of tuples consists of the sum of all shipping cost for 2006. The second tuple is sum of all shipping costs for 2007. The **Item (0)** function calls the first tuple for display in the results or to pass it into another function.

```sql
SELECT {
    ([Measures].[Internet Freight Cost]
    ,[Date].[Calendar Year].CY 2006
    ),
    ([Measures].[Internet Freight Cost]
    ,[Date].[Calendar Year].CY 2007
    )
}.Item(0)
ON COLUMNS
FROM [Adventure Works]
```

Set Functions

In the example below, you see how to create a set explicitly by typing tuples and enclosing them in braces. However for large sets, you’ll want to create them using MDX expressions. Remember that sets are enclosed in `{ }`.

```sql
SELECT {
    [Calendar Quarter].[Q1 CY 2006],
    [Calendar Quarter].[Q2 CY 2006],
    [Calendar Quarter].[Q3 CY 2006],
    [Calendar Quarter].[Q4 CY 2006]
} ON 0
FROM [Adventure Works]
```

<table>
<thead>
<tr>
<th>Q1 CY 2006</th>
<th>Q2 CY 2006</th>
<th>Q3 CY 2006</th>
<th>Q4 CY 2006</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4,069,186.04</td>
<td>$4,153,820.42</td>
<td>$8,880,239.44</td>
<td>$7,341,183.75</td>
</tr>
</tbody>
</table>
The colon operator lets you use the natural order of members to create a set. The following example uses the colon operator to create a set from a range. As you can see in the previous example, it's possible to build a set statically. However, in more complex sets, you'll use expressions like the colon operator.

```sql
SELECT
  ([Calendar Quarter].[Q1 CY 2006]:[Calendar Quarter].[Q4 CY 2006])
ON 0
FROM [Adventure Works]
```

**Note:** The colon operator is an inclusive function.
MDX Operators

Student Activities Required for this Lesson:

- Watch the following recording from the class portal:

  Movie: Mod 2.6 MDX Operators

- Read this lesson.
- Read the following link: http://msdn.microsoft.com/en-us/library/ms144825.aspx

MDX Operators

More Information: For a complete list of all MDX operators, definitions, and examples, see:

MDX operators are common in MDX expressions. They allow developers to perform various operations, check for conditions, search, and much more. In the following pages, you'll learn about some of the most common arithmetic and comparison operators used in MDX expressions. For a complete list of operators and their purposes, see the above link.
Arithmetic Operators

See the table below for a list of common arithmetic operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Sums two numbers</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>^</td>
<td>Raise to power</td>
</tr>
</tbody>
</table>

Example

The calculated member **Arith_Operator_Example** subtracts one tuple from another.

This example takes all months in 2007 (descendants of calendar year 2007) and lays them across the columns axis. All product lines are on axis 1 (rows). The slicer calculates and displays a negative or positive profit percentage based on how each month is doing compared to the previous month.

In this case, the **PrevMember** function is referring to the month's level because it's the selected time period in the query. If you lay years across axis 0, then you’ll get a grain of year.

```
WITH MEMBER [Measures].[Arith_Operator_Example] AS 
    ( 
        (Measures.[Gross Profit Margin]) - 
        ([Date].[Calendar].CurrentMember.PrevMember, Measures.[Gross Profit Margin]) 
    ), FORMAT_STRING = 'Percent'

SELECT 
    DESCENDANTS([Date].[Calendar].[Calendar Year].&[2007], [Date].[Calendar].[Month]) ON 0, 
    [Product].[Product Model Lines].Members ON 1 
FROM 
    [Adventure Works] 
WHERE 
    ([Measures].[Arith_Operator_Example])
```

Results:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>All Products</td>
<td>0.54%</td>
<td>-3.51%</td>
<td>3.38%</td>
<td>0.73%</td>
<td>2.19%</td>
<td>2.92%</td>
<td>-3.12%</td>
</tr>
<tr>
<td>Accessory</td>
<td>1.87%</td>
<td>-1.02%</td>
<td>0.24%</td>
<td>0.05%</td>
<td>0.45%</td>
<td>-0.02%</td>
<td>-16.37%</td>
</tr>
<tr>
<td>Bike Wash</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>32.55%</td>
</tr>
<tr>
<td>Cable Lock</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>-0.46%</td>
<td>-30.77%</td>
</tr>
<tr>
<td>Classic Vest</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>32.93%</td>
</tr>
</tbody>
</table>
Comparison Operators

You can use comparison operators in any Multidimensional Expressions (MDX) expression to compare the values of two MDX expressions. Comparison operators work with scalar data. In computing, a scalar value is single value like the MIN or MAX aggregation across a set. So the comparison operators are used in expressions and calculated members to compare two values. There are equality and inequality operators.

MDX Supported Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>(Equal To)</td>
</tr>
<tr>
<td></td>
<td>✷ Returns TRUE if the left argument is equal to the right argument; otherwise, FALSE is returned.</td>
</tr>
<tr>
<td></td>
<td>✷ If either of the arguments are NULL, then NULL is returned</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>(Not Equal To)</td>
</tr>
<tr>
<td></td>
<td>✷ Returns TRUE if the left argument is not equal to the right argument; otherwise, FALSE is returned.</td>
</tr>
<tr>
<td></td>
<td>✷ If either of the arguments are NULL, then NULL is returned</td>
</tr>
<tr>
<td>&gt;</td>
<td>(Greater Than)</td>
</tr>
<tr>
<td></td>
<td>✷ Returns true if left argument is a higher value than the right argument; otherwise, FALSE is returned.</td>
</tr>
<tr>
<td></td>
<td>✷ If either of the arguments are NULL, then NULL is returned</td>
</tr>
<tr>
<td>&gt;=</td>
<td>(Greater Than or Equal To)</td>
</tr>
<tr>
<td></td>
<td>✷ Returns TRUE if the left argument has a higher value than the right; otherwise, FALSE is returned.</td>
</tr>
<tr>
<td></td>
<td>✷ If either of the arguments are NULL, then NULL is returned</td>
</tr>
<tr>
<td>&lt;</td>
<td>(Less Than)</td>
</tr>
<tr>
<td></td>
<td>✷ Returns TRUE if the left argument is less than the right; otherwise, FALSE is returned.</td>
</tr>
<tr>
<td></td>
<td>✷ If either of the arguments are NULL, then NULL is returned</td>
</tr>
<tr>
<td>&lt;=</td>
<td>(Less Than or Equal To)</td>
</tr>
<tr>
<td></td>
<td>✷ Returns TRUE if the left argument is less than the right; otherwise, FALSE is returned.</td>
</tr>
<tr>
<td></td>
<td>✷ If either of the arguments are NULL, then NULL is returned</td>
</tr>
</tbody>
</table>
Comparison functions, such as the MDX **IIF** function. For the scope of this topic, we’ll key in on the comparison operators. However, if you want to get started reading about more complex conditional comparison expressions, see the link below.

**More Information:**


With member measures. [Sample IIF example] as

\[
\text{IIf} ([\text{Measures}].[\text{Internet Order Count}] > 2000, \text{"orders count is high"}, \text{"order count is low"})
\]

Select {\([\text{Measures}].[\text{Internet Order Count}], \text{measures.[Sample IIF example]}\)} on 0,

[Product].[Product Categories].members on 1

from [Adventure Works]

**Results:**

<table>
<thead>
<tr>
<th>All Products</th>
<th>Internet Order Count</th>
<th>Sample IIF example</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Products</td>
<td>27,659</td>
<td>orders count is high</td>
</tr>
<tr>
<td>Accessories</td>
<td>18,208</td>
<td>orders count is high</td>
</tr>
<tr>
<td>Bike Racks</td>
<td>328</td>
<td>order count is low</td>
</tr>
<tr>
<td>Hitch Rack - 4-Bike</td>
<td>320</td>
<td>order count is low</td>
</tr>
<tr>
<td>Bike Stands</td>
<td>249</td>
<td>order count is low</td>
</tr>
<tr>
<td>All-Purpose Bike Stand</td>
<td>249</td>
<td>order count is low</td>
</tr>
<tr>
<td>Bottles and Cages</td>
<td>4,768</td>
<td>orders count is high</td>
</tr>
<tr>
<td>Mountain Bottle Cage</td>
<td>2,025</td>
<td>orders count is high</td>
</tr>
</tbody>
</table>
Using MDX Operators in Queries

Student Activities Required for this Lesson:

- Watch the following recording from the class portal:
  
  Movie: Mod 2.7 Using MDX Operators in Queries

- Read this lesson.

MDX Operators in Queries

Arithmetic Operators

The following query implements an arithmetic operator. The arithmetic operator (subtraction) is part of the calculated member `Arith_Operator_Example`.

- The calculated member subtracts the gross profit margin for the previous data member (month in this case) from that of the current month, and displays the difference in percent of current value.
- Notice that the slicer axis (where clause) is used to show the calculated member as the data values in the grid.
- The descendants function shows all the month descendants from the year 2007 on axis 0 (rows axis).
- All product lines are displayed on axis 1 (rows axis)

```sql
WITH MEMBER [Measures].[Arith_Operator_Example] AS
    (
        (Measures.[Gross Profit Margin]) -
        ([Date].[Calendar].CurrentMember.PrevMember,
        Measures.[Gross Profit Margin])
    ), FORMAT_STRING = 'Percent'

SELECT
    DESCENDANTS(
        [Date].[Calendar].[Calendar Year].&[2007],
        [Date].[Calendar].[Month]) ON 0,
    [Product].[Product Model Lines].Members ON 1
FROM
    [Adventure Works]
WHERE
    ([Measures].[Arith_Operator_Example])
```
Results:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>All Products</td>
<td>0.54%</td>
<td>-3.51%</td>
<td>3.38%</td>
<td>-0.73%</td>
<td>-2.19%</td>
<td>2.92%</td>
<td>-8.12%</td>
</tr>
<tr>
<td>Accessory</td>
<td>1.67%</td>
<td>-1.20%</td>
<td>0.24%</td>
<td>0.05%</td>
<td>0.45%</td>
<td>-0.02%</td>
<td>-16.87%</td>
</tr>
<tr>
<td>Bike/Wash</td>
<td>[null]</td>
<td>[null]</td>
<td>[null]</td>
<td>[null]</td>
<td>[null]</td>
<td>[null]</td>
<td>37.55%</td>
</tr>
<tr>
<td>Cable Lock</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.48%</td>
<td>-0.77%</td>
<td></td>
</tr>
</tbody>
</table>

**Union Operator (+)**

The following query is used to union two MDX sets together with the (+) operator.

```sql
SELECT [Date].[Calendar].[Calendar Year].Members ON 0,
       {{[Product].[Product Categories].[Category].&[4]} +
         {{[Product].[Product Categories].[Category].&[1]}} ON 1
FROM [Adventure Works]
WHERE ([Measures].[Gross Profit Margin])
```

Results:
Cross Join Operator (*)

The following query is used to cross-join (*) two sets. A cross join is the product of all items in one set, intersected by all items in another set. It can be very CPU and memory intensive.

```
SELECT [Date].[Calendar].[Calendar Year].Members * 
    [Reseller].[Reseller Type].Children ON 0, 
    [Product].[Category].[Category].Members ON 1 
FROM [Adventure Works] 
WHERE ([Measures].[Gross Profit Margin])
```

Results:

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specialty Bike Shop</td>
<td>Value Added Reseller</td>
<td>Warehouse</td>
<td>Specialty Bike Shop</td>
<td>Value Added Reseller</td>
</tr>
<tr>
<td>Accessories</td>
<td>40.36%</td>
<td>40.36%</td>
<td>40.36%</td>
<td>30.57%</td>
<td>30.57%</td>
</tr>
<tr>
<td>Bikes</td>
<td>14.82%</td>
<td>14.82%</td>
<td>14.62%</td>
<td>9.11%</td>
<td>9.11%</td>
</tr>
<tr>
<td>Clothing</td>
<td>-5.56%</td>
<td>-5.56%</td>
<td>-5.56%</td>
<td>21.03%</td>
<td>21.03%</td>
</tr>
<tr>
<td>Components</td>
<td>8.78%</td>
<td>8.73%</td>
<td>8.78%</td>
<td>11.80%</td>
<td>11.80%</td>
</tr>
</tbody>
</table>
```
Boolean Operators (evaluates to TRUE or FALSE)

The following query is used to see how MDX Boolean (true/false) operators work. This query has five calculated members that use the equals (=) operator to evaluate the equality of two values. Review the results.

Boolean operators return true or false. Even though you’re not actually pulling any data from the Adventureworks cube, you still have to include a cube space in the query. Notice that 0=Null evaluates to true.

With
Member [Measures].bool1 as 1=1
Member [Measures].bool2 as 1=0
Member [Measures].bool3 as null=null
Member [Measures].bool4 as 0=null
Member [Measures].bool5 as 1=null
Select
{[Measures].bool1,[Measures].bool2,[Measures].bool3,[Measures].bool4,[Measures].bool5}
On 0
From [Adventure Works]

Results:

<table>
<thead>
<tr>
<th></th>
<th>bool1</th>
<th>bool2</th>
<th>bool3</th>
<th>bool4</th>
<th>bool5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

IS Operator

This IS operator is used to compare two cube object references to see if they point to the same object. If they do, the IS operator evaluates to true, otherwise the expression evaluates to false.

This is a simple calculated member that returns true if the product category is clothing [3].

With
Member [Measures].[Is_Clothing] AS
[Product].[Category].CurrentMember IS [Product].[Category].&[3]
SELECT
{[Measures].[Is_Clothing]} ON 0,
[Product].[Category].[Category].Members ON 1
FROM
[Adventure Works]
Results:

<table>
<thead>
<tr>
<th>Messages</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is_Clothing</td>
<td>True</td>
</tr>
<tr>
<td>Accessories</td>
<td>False</td>
</tr>
<tr>
<td>Bikes</td>
<td>False</td>
</tr>
<tr>
<td>Clothing</td>
<td>True</td>
</tr>
<tr>
<td>Components</td>
<td>False</td>
</tr>
</tbody>
</table>

Range Operator (:) 

The following query uses a range operator. This query shows aggregated Internet sales amounts for each product category from January 2008 – March 2008. The set of dates is expressed as a range. Range operators are inclusive.

```
SELECT
    { [Ship Date].[Calendar].[Month].[2008][1]: [Ship Date].[Calendar].[Month].[2008][3] } ON 0,
    [Product].[Category].[Category].Members ON 1
FROM
    [Adventure Works]
WHERE
    [Measures].[Internet Sales Amount]
```

Results:

<table>
<thead>
<tr>
<th>Messages</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>January 2008</td>
</tr>
<tr>
<td>Accessories</td>
<td>$57,923.18</td>
</tr>
<tr>
<td>Bikes</td>
<td>$1,334,494.11</td>
</tr>
<tr>
<td>Clothing</td>
<td>$30,096.07</td>
</tr>
<tr>
<td>Components</td>
<td>(null)</td>
</tr>
</tbody>
</table>
Exception Operator (-)

This query uses the exception (-) operator to show the reseller tax amount measure for all provinces in Canada except for Alberta. The expression is subtracting Alberta from the set which creates the set with all provinces except Alberta.

```sql
SELECT [Measures].[Reseller Tax Amount] ON COLUMNS,
      [Geography].[Geography].[Country].&[Canada].Children - [Geography].[Geography].[State-Province].&[AB]&[CA] ON ROWS
FROM   [Adventure Works]
```

Results:

<table>
<thead>
<tr>
<th>Province</th>
<th>Reseller Tax Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>British Columbia</td>
<td>$273,570.38</td>
</tr>
<tr>
<td>Brunswick</td>
<td>$26,001.83</td>
</tr>
<tr>
<td>Manitoba</td>
<td>$5,295.46</td>
</tr>
<tr>
<td>Ontario</td>
<td>$494,335.02</td>
</tr>
<tr>
<td>Quebec</td>
<td>$239,349.22</td>
</tr>
</tbody>
</table>
Introduction to SET Functions

Student Activities Required for this Lesson:

- Watch the following recording from the class portal:
  
  Movie: Mod 2.8 Introduction to SET functions

- Read this lesson.

- Read the following link: http://msdn.microsoft.com/en-us/library/ms144851.aspx

Overview of MDX SET Functions

As was introduced earlier, the SET is at the very heart of MDX. Understanding the basic SET functions will take you a long way towards mastering the language. In this topic, the goal is to get you curious as to what SET functions available to you as a developer.

Common SET Functions

More often than not, SET functions will be used to assemble sets from different levels in dimensional hierarchies. Some functions traverse up the hierarchy, some down the hierarchy, some are for reading member and member properties from the current level.

- .members
- .allmembers
- .descendants
- .ancestors
- .exists
- .children
- .siblings
MDX SET Functions

Use the following table to familiarize yourself with the set functions that are available to you in MDX. We’ll cover some of them in class; however, you’ll want to have reference to the list when developing. You don’t have to memorize what you can look up.

<table>
<thead>
<tr>
<th>SET Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddCalculatedMembers (MDX)</td>
<td>Hierarchize (MDX)</td>
</tr>
<tr>
<td>AllMembers (MDX)</td>
<td>Intersect (MDX)</td>
</tr>
<tr>
<td>Ancestors (MDX)</td>
<td>LastPeriods (MDX)</td>
</tr>
<tr>
<td>Ascendants (MDX)</td>
<td>Members (Set) (MDX)</td>
</tr>
<tr>
<td>Axis (MDX)</td>
<td>Mtd (MDX)</td>
</tr>
<tr>
<td>BottomCount (MDX)</td>
<td>NameToSet (MDX)</td>
</tr>
<tr>
<td>BottomPercent (MDX)</td>
<td>NonEmptyCrossjoin (MDX)</td>
</tr>
<tr>
<td>BottomSum (MDX)</td>
<td>Order (MDX)</td>
</tr>
<tr>
<td>Children (MDX)</td>
<td>PeriodsToDate (MDX)</td>
</tr>
<tr>
<td>Crossjoin (MDX)</td>
<td>Qtd (MDX)</td>
</tr>
<tr>
<td>CurrentOrdinal (MDX)</td>
<td>Siblings (MDX)</td>
</tr>
<tr>
<td>Descendants (MDX)</td>
<td>StripCalculatedMembers (MDX)</td>
</tr>
<tr>
<td>Distinct (MDX)</td>
<td>StrToSet (MDX)</td>
</tr>
<tr>
<td>DrilldownLevel (MDX)</td>
<td>Subset (MDX)</td>
</tr>
<tr>
<td>DrilldownLevelBottom (MDX)</td>
<td>Tail (MDX)</td>
</tr>
<tr>
<td>DrilldownLevelTop (MDX)</td>
<td>ToggleDrillState (MDX)</td>
</tr>
<tr>
<td>DrilldownMember (MDX)</td>
<td>TopCount (MDX)</td>
</tr>
<tr>
<td>DrilldownMemberBottom (MDX)</td>
<td>TopPercent (MDX)</td>
</tr>
<tr>
<td>DrilldownMemberTop (MDX)</td>
<td>TopSum (MDX)</td>
</tr>
<tr>
<td>DrillupLevel (MDX)</td>
<td>Union (MDX)</td>
</tr>
<tr>
<td>DrillupMember (MDX)</td>
<td>Unorder (MDX)</td>
</tr>
<tr>
<td>Except (MDX)</td>
<td>VisualTotals (MDX)</td>
</tr>
<tr>
<td>Exists (MDX)</td>
<td>Wtd (MDX)</td>
</tr>
<tr>
<td>Extract (MDX)</td>
<td>Ytd (MDX)</td>
</tr>
<tr>
<td>Filter (MDX)</td>
<td></td>
</tr>
<tr>
<td>SET Function</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---</td>
</tr>
<tr>
<td>Generate (MDX)</td>
<td></td>
</tr>
</tbody>
</table>
.Children and .Members Functions

Student Activities Required for this Lesson:

- Watch the following recording from the class portal:
  
  Movie: Mod 2.9 MDX Children and Member Functions

- Read this lesson.

- Read the following links:
  

The Members Function

In topic, you’ll learn to build sets with the .members function. We’ll get into more complex SET expression and functions as we move through the course. The .members function is used to return the set of members in a dimension, level, or hierarchy.

The .members function works by returning all dimension or hierarchy level members (excluding calculated members) for the referenced dimension or hierarchy level. To view calculated members, use the .allmembers function.

Notice that the .members function is used on the rows axis to display one row for product with an aggregation named All Products which is the sum for all categories (name automatically generated).

```sql
SELECT
  { [Measures].[Sales Amount],
    [Measures].[Tax Amount] } ON 0,
  {[Product].[Category].members } ON 1
FROM [Adventure Works]
WHERE ([Delivery Date].[Fiscal].[Fiscal Year].&[2008] )
```

Results:

<table>
<thead>
<tr>
<th></th>
<th>Sales Amount</th>
<th>Tax Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Products</td>
<td>$52,136,359.49</td>
<td>$4,170,908.70</td>
</tr>
<tr>
<td>Accessories</td>
<td>$1,050,729.86</td>
<td>$84,058.39</td>
</tr>
<tr>
<td>Bikes</td>
<td>$43,813,374.32</td>
<td>$3,505,069.91</td>
</tr>
<tr>
<td>Clothing</td>
<td>$1,269,045.12</td>
<td>$101,523.60</td>
</tr>
<tr>
<td>Components</td>
<td>$6,003,210.20</td>
<td>$480,256.80</td>
</tr>
</tbody>
</table>
All Members

In the following example, we’re querying all measures across the category 1 (Bikes) tuple set. When asking for the measures on the rows axis, we’re using the members function to return all measures in the measures dimension. The measures dimension is an SSAS dimension that stores the measure for the cube.

The Internet Sales measure group has both measures and calculated members.

You can see the difference between the two in the metadata browser of any OLAP client. In this case, we’ll view the measures from the metadata browser in SQL Server 2008 SSMS.

Notice that the calculated members are marked with a small calculator over the measure. This means that they have been added as customizations in the cube by the cube administrator or developer. More times than not, it’s easier to create a calculated member for a specific need than it is to customize the ETL and add it as a fact table column in the data mart.

```sql
select
    {{[Product].[Product Categories].[Category].&[1]} on 0,
    {[Measures].members} on 1
from [Adventure Works]
```

Notice that the calculated members are not returned using the `.members` function. In order to return calculated members, you need to use the `.allmembers` function.
To return calculated members along the bikes tuple set, you need to use the .allmembers function on the [Measures] dimension. The calculated members are appended to the set after all the members are displayed. You'll soon see how to order the set.

```
select
{[Product].[Product Categories].[Category].&[1]} on 0,
{[Measures].allmembers} on 1
from [Adventure Works]
```

Results:

<table>
<thead>
<tr>
<th>Internet Sales</th>
<th>Bikes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Rate</td>
<td>1.00</td>
</tr>
<tr>
<td>End of Day Rate</td>
<td>1.00</td>
</tr>
<tr>
<td>Internet Gross Profit</td>
<td>$11,505,796.50</td>
</tr>
<tr>
<td>Internet Gross Profit Margin</td>
<td>40.63%</td>
</tr>
<tr>
<td>Internet Average Unit Price</td>
<td>$1,862.42</td>
</tr>
<tr>
<td>Internet Average Sales Amount</td>
<td>$1,862.42</td>
</tr>
<tr>
<td>Internet Ratio to All Products</td>
<td>96.46%</td>
</tr>
<tr>
<td>Internet Ratio to Parent Product</td>
<td>96.46%</td>
</tr>
<tr>
<td>Internet Sales Amount</td>
<td>NA</td>
</tr>
<tr>
<td>Internet Standard Product Cost</td>
<td>($390,698.89)</td>
</tr>
<tr>
<td>Internet Tax Amount</td>
<td>-1.49%</td>
</tr>
<tr>
<td>Internet Total Product Cost</td>
<td>$882.72</td>
</tr>
</tbody>
</table>

Reseller Orders
Descendants

Student Activities Required for this Lesson:

- Watch the following recording from the class portal:
  
  Movie: Mod 2.10a MDX Hierarchies and Descendants

  Movie: Mod 2.10b MDX Descendants Function

- Read this lesson.
- Read the following link: http://msdn.microsoft.com/en-us/library/ms146075.aspx

Introduction to Dimension Hierarchies

For the scope of this topic, it’s important for you to understand the basics of dimensional hierarchies. A hierarchy is a set of 1: many or in certain cases many: many relationships between attributes in a dimension or dimensions. For example, the geography hierarchy has 4 levels to it. The lowest grain is country and the highest grain is postal code.

Grain or granularity is used to define the level of detail that can be described by the hierarchy. For example each county can have multiple state-province members. This makes the state province a higher grain than country. The key attribute is the highest grain because it represents a unique geography member. Each geography key is a unique combination of all the other levels in the hierarchy.

Below is a screen shot from the attributes relationships tab for the Geography hierarchy in BIDS.
From a PivotTable

Notice that the top of the hierarchy from the UI is the furthest to the right in the attribute relationships. The 1: Many relationships cascade downward from Country.

<table>
<thead>
<tr>
<th>Country</th>
<th>State-Province</th>
<th>City</th>
<th>Postal Code</th>
<th>Reseller Sales Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia</td>
<td></td>
<td></td>
<td></td>
<td>$1,594,335.38</td>
</tr>
<tr>
<td>Canada</td>
<td>Alberta</td>
<td>Burnaby</td>
<td></td>
<td>$1,304,163.69</td>
</tr>
<tr>
<td></td>
<td>British Columbia</td>
<td>Richmond</td>
<td></td>
<td>$1,250,920.00</td>
</tr>
<tr>
<td></td>
<td>Surrey</td>
<td>V3T 4W3</td>
<td></td>
<td>$61,809.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Total</td>
<td></td>
<td>$790,199.30</td>
</tr>
<tr>
<td></td>
<td>Vancouver</td>
<td></td>
<td></td>
<td>$4,537.64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Total</td>
<td></td>
<td>$41,410,629.76</td>
</tr>
</tbody>
</table>

The Descendants Function

- The `.descendants` function allows you to define a set that contains members based on each member's location in the hierarchy.

- The `.descendants` function returns the set of descendants of a member, at a specified level or distance from that level or member.

When requesting descendant members from a particular hierarchy level, the query needs to know how far down the hierarchy to look, and whether to include intermediate level members.

The `.descendants` function takes the following arguments:

- Member Expression
- Set Expression
- Level Expression
- Distance
- DESC Flag

In short, the Member Expression is a valid MDX expression which returns one or more members. It defines the starting point in the hierarchy for which the `.descendants` function will be based.

The Set Expression defines the set or is the named set. If a Level Expression is specified, the `.descendants` function returns a set that corresponds to that level in the hierarchy.

If Distance is specified, the `.descendants` function returns a set that contains the descendants of the specified number of levels down the hierarchy from the level defined by the Member Expression.

If the specified distance is zero (0), the set defined by the Member Expression is returned.
The Description Flag (DESC)

The **Description** flag (DESC) flag of the **descendents** function is used to include or exclude descendants at the specified level or distance. The description flag supports the following arguments:

- SELF
- AFTER
- BEFORE
- BEFORE_AND_AFTER
- SELF_AND_AFTER
- SELF_AND_BEFORE
- SELF_BEFORE_AFTER
- LEAVES

**Descendants Example 1**

```sql
SELECT [Measures].[Reseller Sales Amount] on 0,
       Descendants
       ([Geography].[Geography].[Country].&[United States],
        [Geography].[Geography].[City], BEFORE)
       ON 1
FROM [Adventure Works]
```

Results: Shows Reseller sales amount on columns, and all states on rows. The DESC flag of BEFORE (level city), tells the query to return everything after United States (Country Level) and before the city level. That leaves all members of the State-Province level. So the results show all U.S. states.

<table>
<thead>
<tr>
<th>Reseller Sales Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States</td>
</tr>
<tr>
<td>Alabama</td>
</tr>
<tr>
<td>Arizona</td>
</tr>
<tr>
<td>California</td>
</tr>
<tr>
<td>Colorado</td>
</tr>
<tr>
<td>Connecticut</td>
</tr>
<tr>
<td>Florida</td>
</tr>
<tr>
<td>Georgia</td>
</tr>
</tbody>
</table>
Descendants Example 2

In this example, the county of United States is replaced with the County level of the Geography hierarchy (all countries). This time all cities from all counties are calculated and displayed.

```sql
SELECT
[Measures].[Reseller Sales Amount] on 0,
Descendants
  ([Geography].[Geography].[Country],
   [Geography].[Geography].[City], BEFORE
 ) ON 1
FROM [Adventure Works]
```

Results: Shows State-Province members for all countries, and totals for each country.

<table>
<thead>
<tr>
<th>State-Province</th>
<th>Reseller Sales Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Niederrhein-Westfalen</td>
<td>€223,919.94</td>
</tr>
<tr>
<td>Saarland</td>
<td>€307,718.47</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>€4,273,000.83</td>
</tr>
<tr>
<td>England</td>
<td>€4,273,000.83</td>
</tr>
<tr>
<td>United States</td>
<td>€53,637,801.21</td>
</tr>
<tr>
<td>Alabama</td>
<td>€45,429.03</td>
</tr>
<tr>
<td>Arizona</td>
<td>€1,432,505.57</td>
</tr>
<tr>
<td>California</td>
<td>€3,764,270.71</td>
</tr>
<tr>
<td>Colorado</td>
<td>€2,395,513.84</td>
</tr>
<tr>
<td>Connecticut</td>
<td>€1,125,700.41</td>
</tr>
<tr>
<td>Florida</td>
<td>€2,299,000.07</td>
</tr>
<tr>
<td>Georgia</td>
<td>€1,044,577.80</td>
</tr>
<tr>
<td>Idaho</td>
<td>€227,301.75</td>
</tr>
</tbody>
</table>

© [2008-2010] HDS Training Portals. All rights reserved. www.hdstrainingstore.com / customersupport@hdstraining.com 49
Descendants Example 3

The following example uses the hierarchy level ordinal to specify what levels will be included. In this case, the set expression is starting at the country level (all members), and goes down to level 4 (Postal Code), and only displays members before the postal code because of the DESC Before flag.

```
SELECT [Measures].[Reseller Sales Amount] on 0, Descendants
     ([Geography].[Geography].[Country]
     , [Geography].[Geography].Levels (4)
     , BEFORE
     ) ON 1
FROM [Adventure Works]
```

Results: Shows all geography members except for postal codes (countries, state-provinces, and cities).

<table>
<thead>
<tr>
<th>Country</th>
<th>Reseller Sales Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia</td>
<td>$1,594,335.38</td>
</tr>
<tr>
<td>New South Wales</td>
<td>$1,129,270.79</td>
</tr>
<tr>
<td>Alexandria</td>
<td>(null)</td>
</tr>
<tr>
<td>Colfs Harbour</td>
<td>(null)</td>
</tr>
<tr>
<td>Darlinghurst</td>
<td>$7,337.60</td>
</tr>
<tr>
<td>Goulburn</td>
<td>(null)</td>
</tr>
<tr>
<td>Lane Cove</td>
<td>$148,996.51</td>
</tr>
<tr>
<td>Lavender Bay</td>
<td>$278,391.16</td>
</tr>
</tbody>
</table>

Descendants Example 4

This example used the DESC flag of SELF_AND_BEFORE to include level 4 members (Postal Codes).

```
SELECT [Measures].[Reseller Sales Amount] on 0, Descendants
     ([Geography].[Geography].[Country]
     , [Geography].[Geography].Levels (4)
     , SELF_AND_BEFORE
     ) ON 1
FROM [Adventure Works]
```

Results: Postal coded included now.

<table>
<thead>
<tr>
<th>Location</th>
<th>Reseller Sales Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Darlinghurst</td>
<td>$7,337.60</td>
</tr>
<tr>
<td>2010</td>
<td>$7,337.60</td>
</tr>
<tr>
<td>Goulburn</td>
<td>(null)</td>
</tr>
<tr>
<td>2580</td>
<td>(null)</td>
</tr>
</tbody>
</table>
Module 1 Lab

Exercise 1: Working with Axes, Tuples, and Sets

In this exercise you’ll work with the MDX language elements you learned about in this module.

Task 1: Working with MDX Axes

Note: SSMS code editor shortcuts:

1. Construct and execute the following query to show the Internet Sales Amount measure on the columns axis (0), and each member of the date dimension on the rows axis (1).

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

   The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-SimpleQuery.MDX

Results:

<table>
<thead>
<tr>
<th></th>
<th>Internet Sales Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Periods</td>
<td>$29,359,677.22</td>
</tr>
<tr>
<td>CY 2005</td>
<td>$3,266,373.66</td>
</tr>
<tr>
<td>H2 CY 2005</td>
<td>$3,266,373.66</td>
</tr>
<tr>
<td>Q3 CY 2005</td>
<td>$1,453,522.89</td>
</tr>
</tbody>
</table>
2. Modify the query used in step 1 to reference the columns axis as **axis 0**, and the rows axis as **axis 1**. The MDX query window in SSMS cannot display more than rows and columns, but certain front end applications can display more than two axes through the use of advanced graphics and report controls.

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

Results:

<table>
<thead>
<tr>
<th></th>
<th>Internet Sales Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Periods</td>
<td>$29,359,677.22</td>
</tr>
<tr>
<td>CY 2005</td>
<td>$3,266,373.66</td>
</tr>
<tr>
<td>H2 CY 2005</td>
<td>$3,266,373.66</td>
</tr>
<tr>
<td>Q3 CY 2005</td>
<td>$1,453,522.89</td>
</tr>
</tbody>
</table>

3. Construct and execute the following query to summarize Internet sales by each product category. Use the slicer axis to only return rows for 2008.

**The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-SimpleSlicer.MDX**

```plaintext
SELECT { [Measures].[Internet Sales Amount]} ON COLUMNS,
       [Product].[Category].[Category].MEMBERS ON ROWS
FROM [Adventure Works]
WHERE [Delivery Date].[Fiscal Year].&[2008]
```
4. Construct and execute the following query to apply a slicer which uses a set from the geography dimension (Countries of Germany and Australia), and the tuple of product category 1. The cells returned on the rows and columns axes will only show aggregated values that meet the restrictions of the slicer.

The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-SlicerSet-Tuple.MDX

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

([Geography].[Country].&[Australia],[Geography].[Country].&[Germany] ),
[Product].[Category].&[1]

```

Results:

<table>
<thead>
<tr>
<th>Period</th>
<th>Internet Tax Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Periods</td>
<td>$2,265,451.62</td>
</tr>
<tr>
<td>CY 2005</td>
<td>$261,309.99</td>
</tr>
<tr>
<td>CY 2006</td>
<td>$522,427.50</td>
</tr>
<tr>
<td>CY 2007</td>
<td>$748,728.24</td>
</tr>
<tr>
<td>CY 2008</td>
<td>$732,985.99</td>
</tr>
<tr>
<td>CY 2010</td>
<td>(null)</td>
</tr>
</tbody>
</table>
5. Construct and execute the following query. This query uses a slightly different slicer from the previous step. Now the slicer shows the aggregated Internet Tax Amount for Canadian customers, who bought product category 1, and are home owners. Notice the number of rows is the same because you still have the same number of members returned on rows.

You'll learn about functions like FILTER and TOPCOUNT that can be used on the rows axis to restrict the number or rows returned in the next module.

The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-SlicerSet-Tuple2.MDX

```mdx
SELECT
{[Measures].[Internet Tax Amount]} ON COLUMNS,
[Date].[Calendar Year].MEMBERS ON ROWS
FROM [Adventure Works]
WHERE
(
{[Geography].[Country].&[Canada] },
[Product].[Category].&[1],
[Customer].[Home Owner].&[1]
)
```

Results:

<table>
<thead>
<tr>
<th></th>
<th>Internet Tax Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Periods</td>
<td>$1,573,050.58</td>
</tr>
<tr>
<td>CY 2005</td>
<td>$172,920.33</td>
</tr>
<tr>
<td>CY 2006</td>
<td>$358,070.45</td>
</tr>
<tr>
<td>CY 2007</td>
<td>$526,851.83</td>
</tr>
<tr>
<td>CY 2008</td>
<td>$515,208.06</td>
</tr>
<tr>
<td>CY 2010</td>
<td>(null)</td>
</tr>
</tbody>
</table>
Task 2: Using Identifiers and MDX Expressions

Mod 2.12 Lab Demo: Using Identifiers and MDX Expressions

Tip: Watch the lab Demo before doing the lab. Refer to it for a demonstration of all steps. Steps in the lab exercise are abbreviated to make you think about what you’re doing instead of just following steps.

1. Logon to your cloud computing image as **Student** with password **Pa$$w0rd**.
2. Open **SQL Server Management Studio** (SSMS).
3. Attach the default instance of Analysis Services.
4. Expand **Databases** → **Adventure Works DW 2008R2**

5. Click **New Query**.
6. Type (or drag/drop to build) the following query. This query is an example of calling regular identifiers.

The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-RegularIdentifier.mdx.

```sql
SELECT Measures.MEMBERS ON COLUMNS,
       Product.Style.CHILDREN ON ROWS
FROM [Adventure Works]
```

7. Type (or drag/drop to build) the following query. This query is an example of calling delimited identifiers.

The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-DelimitedIdentifier.mdx.

```sql
SELECT [Measures].MEMBERS ON COLUMNS,
       [Product].[Product].[Front Brakes] ON ROWS
FROM [Adventure Works]
```
8. Type and execute the following query to see how to use an MDX expression in a query. Below is an example of an MDX Expression which is being used as calculation later in the query. The simple expression shows a calculated discount amount (1.8 * the current discount level). The slicer is for only category 3 products (clothing).

The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-SimpleMDXExpression.mdx.

```mdx
WITH
    MEMBER [Measures].[Special Discount] AS
        [Measures].[Discount Amount] * 1.8
SELECT
    [Measures].[Special Discount] on COLUMNS,
    NON EMPTY [Product].[Product].MEMBERS  ON Rows
FROM [Adventure Works]
WHERE [Product].[Product Categories].[Category].&[3]
```

9. Stay logged in with SSMS open for the next task.
Task 3: Simple and Complex Tuples

1. Construct and execute the following statement.

The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-TupleIntro.mdx.

Select
{ ([Measures].[Internet Sales Amount], [Date].[Fiscal].[Fiscal Year].&[2007]), ([Measures].[Internet Tax Amount],[Date].[Fiscal].[Fiscal Year].&[2007]) } on 0,
{ [Product].[Category].&[1] } on 1
from [Adventure Works]

Make sure you can identify both tuples in the rows axis set.

First tuple: Internet Sales for 2007

This tuple stores a cell value that represents all internet sales (SUM) for 2007.

Second Tuple: Internet Tax for 2007

This tuple stores a cell value that represents all internet sales (SUM) for 2007.

Results: (product category 1 is bikes).

<table>
<thead>
<tr>
<th></th>
<th>Internet Sales Amount</th>
<th>Internet Tax Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>FY 2007</td>
<td>$5,762,134.30</td>
<td>$460,370.78</td>
</tr>
<tr>
<td>Bikes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Change the query you build in step 1 to include all members of product categories hierarchy attribute by using the members function on the rows (1) axis. Bikes were the only sales in 2007.

Select
{ ([Measures].[Internet Sales Amount], [Date].[Fiscal].[Fiscal Year].&[2007]), ([Measures].[Internet Tax Amount],[Date].[Fiscal].[Fiscal Year].&[2007]) } on 0,
[Product].[Category].members on 1
from [Adventure Works]
3. Construct a simple tuple and return it on the columns axis (axis 0).

The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-SimpleTuple.mdx.

```csharp
Select
    ( [Measures].[Internet Sales Amount],
      [Date].[Calendar Year].&[2008],
      [Product].[Product Categories].[Category].&[3] )
on 0
from [Adventure Works]
```

Results:

```
Internet Sales Amount
----
CY 2008
Clothing
$201,524.64
```

4. Change the query from step one to include an orders list of tuples by changing the tuple from just 2008 to all years.

The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-SimpleTupleSet.mdx.

```csharp
Select
    ( [Measures].[Internet Sales Amount],
      [Date].[Calendar Year].members,
      [Product].[Product Categories].[Category].&[3] )
on 0
from [Adventure Works]
```

Results:
5. Execute the following MDX query to create a set of tuples and return the first tuple in the set. You can drag and drop from the metadata browser to create your query.

The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-ExplicitTupleSet.mdx.

```mdx
SELECT {
    [Calendar Quarter].[Q1 CY 2006],
    [Calendar Quarter].[Q2 CY 2006],
    [Calendar Quarter].[Q3 CY 2006],
    [Calendar Quarter].[Q4 CY 2006]
} ON 0
FROM [Adventure Works]
```

6. Type and execute a similar query to that of step 1, except this time use a colon operator to produce the same output by using a range to generate the tuple set. Verify the result match step 1.

```mdx
SELECT {
    [Calendar Quarter].[Q1 CY 2006]:[Calendar Quarter].[Q4 CY 2006]
} ON 0
FROM [Adventure Works]
```

The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-ExpressionTupleSet.mdx.
7. Execute the following code to return the first tuple in a set of tuples. The first tuple is the first country listed in the current geography. The last tuple is the last country listed in the current geography.

The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-ItemTupleSet.mdx.

WITH MEMBER Measures.Percent_of_Total
AS
[Measures].[Internet Sales Amount] / 
( [Measures].[Internet Sales Amount],
Ancestors
( [Customer].[Customer Geography].CurrentMember,
[Customer].[Customer Geography].[Country]
).Item (0)
), FORMAT_STRING = '0%
SELECT {[Measures].[Internet Sales Amount],Measures.Percent_of_Total} ON 0,
{ Descendants
( [Customer].[Customer Geography].[Country].&[United States],
[Customer].[Customer Geography].[State-Province], SELF
)
} ON 1
FROM [Adventure Works]

Results:

<table>
<thead>
<tr>
<th>Country</th>
<th>Internet Sales Amount</th>
<th>Percent_of_Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>$37.29</td>
<td>0%</td>
</tr>
<tr>
<td>Arizona</td>
<td>$2,104.02</td>
<td>0%</td>
</tr>
<tr>
<td>California</td>
<td>$5,714,257.69</td>
<td>61%</td>
</tr>
<tr>
<td>Colorado</td>
<td>(null)</td>
<td>(null)</td>
</tr>
<tr>
<td>Connecticut</td>
<td>(null)</td>
<td>(null)</td>
</tr>
<tr>
<td>Florida</td>
<td>$7,760.91</td>
<td>0%</td>
</tr>
<tr>
<td>Georgia</td>
<td>$1,658.92</td>
<td>0%</td>
</tr>
<tr>
<td>Idaho</td>
<td>(null)</td>
<td>(null)</td>
</tr>
<tr>
<td>Illinois</td>
<td>$2,828.09</td>
<td>0%</td>
</tr>
<tr>
<td>Indiana</td>
<td>(null)</td>
<td>(null)</td>
</tr>
<tr>
<td>Kentucky</td>
<td>$216.96</td>
<td>0%</td>
</tr>
<tr>
<td>Maine</td>
<td>(null)</td>
<td>(null)</td>
</tr>
</tbody>
</table>
Task 4: Working with MDX Operators

**Movie: Mod 2.14 MDX Operators**

1. Construct and execute the following query to check for order counts above 2000 for all products. If the order quantity is > 2000 return “order count is high”, otherwise return “order count is low”.

   The below code is in \C:\Labs-HSQL1018\Solution\Module2\MDX-IIF.MDX

   With member measures.[Sample Iff example] as
   IIf ([Measures].[Internet Order Count] > 2000, "orders count is high", "order count is low")

   Select {{Measures].[Internet Order Count], measures.[Sample Iff example]} on 0,
   [Product].[Product Categories].members on 1
   from [Adventure Works]

   Results:

<table>
<thead>
<tr>
<th>Product Category</th>
<th>Internet Order Count</th>
<th>Sample Iff example</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Products</td>
<td>27,659</td>
<td>orders count is high</td>
</tr>
<tr>
<td>Accessories</td>
<td>18,208</td>
<td>orders count is high</td>
</tr>
<tr>
<td>Bike Racks</td>
<td>328</td>
<td>order count is low</td>
</tr>
<tr>
<td>Hitch Rack - 4-Bike</td>
<td>320</td>
<td>order count is low</td>
</tr>
<tr>
<td>Bike Stands</td>
<td>249</td>
<td>order count is low</td>
</tr>
<tr>
<td>All-Purpose Bike Stand</td>
<td>249</td>
<td>order count is low</td>
</tr>
<tr>
<td>Bottles and Cages</td>
<td>4,768</td>
<td>orders count is high</td>
</tr>
<tr>
<td>Mountain Bottle Cage</td>
<td>2,025</td>
<td>orders count is high</td>
</tr>
</tbody>
</table>

2. Execute the following query to implement comment operators. The arithmetic operator (subtraction) is part of the calculated member `Arith_Operator_Example`.

   - The calculated member subtracts the gross profit margin for the previous data member (month in this case) from that of the current month, and displays the difference in percent of current value.
   - Notice that the slicer axis (where clause) is used to show the calculated member as the data values in the grid.
   - The descendants function shows all the month descendents from the year 2007 on axis 0 (rows axis).
   - All product lines are displayed on axis 1 (rows axis)
The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-Arithoperator.MDX.

WITH MEMBER [Measures].[Arith_Operator_Example] AS
  (Measures.[Gross Profit Margin]) -
  (([Date].[Calendar].CurrentMember.PrevMember,
  Measures.[Gross Profit Margin])
  ), FORMAT_STRING = 'Percent'

SELECT DESCENDANTS([Date].[Calendar].[Calendar Year].&[2007],
  [Date].[Calendar].[Month]) ON 0,
  [Product].[Product Model Lines].Members ON 1
FROM [Adventure Works]
WHERE ([Measures].[Arith_Operator_Example])

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>All Products</td>
<td>0.54%</td>
<td>-3.51%</td>
<td>3.38%</td>
<td>-0.73%</td>
<td>-2.19%</td>
<td>2.92%</td>
<td>-8.12%</td>
</tr>
<tr>
<td>Accessories</td>
<td>1.67%</td>
<td>-1.20%</td>
<td>0.24%</td>
<td>0.05%</td>
<td>0.45%</td>
<td>-0.02%</td>
<td>-16.87%</td>
</tr>
<tr>
<td>Bike/Wash</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>37.55%</td>
</tr>
<tr>
<td>Cable Lock</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>-0.48%</td>
<td>-30.77%</td>
</tr>
</tbody>
</table>

10. Execute the following query to union two MDX sets together with the + operator.

The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-UnionOperator.MDX.

SELECT [Date].[Calendar].[Calendar Year].Members ON 0,
  {{[Product].[Product Categories].[Category].&[4]} +
  {{[Product].[Product Categories].[Category].&[1]}} ON 1
FROM [Adventure Works]
WHERE ([Measures].[Gross Profit Margin])

Results:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessories</td>
<td>40.38%</td>
<td>30.57%</td>
<td>46.06%</td>
<td>55.25%</td>
<td>(null)</td>
</tr>
<tr>
<td>Bikes</td>
<td>14.82%</td>
<td>3.11%</td>
<td>8.74%</td>
<td>15.38%</td>
<td>(null)</td>
</tr>
</tbody>
</table>

11. Execute the following query to cross-join (*) two sets. A cross join is the product of all items in one set, intersected by all items in another set.
The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-CrossJoinOperator.MDX

```sql
SELECT [Date].[Calendar].[Calendar Year].Members *
[Reseller].[Reseller Type].Children ON 0,
[Product].[Category].[Category].Members ON 1
FROM [Adventure Works]
WHERE ([Measures].[Gross Profit Margin])
```

Results:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Specialty Bike Shop</td>
<td>40.38%</td>
<td>40.38%</td>
<td>40.38%</td>
<td>30.57%</td>
<td>30.57%</td>
</tr>
<tr>
<td>Value Added Reseller</td>
<td>14.82%</td>
<td>14.82%</td>
<td>14.82%</td>
<td>9.11%</td>
<td>9.11%</td>
</tr>
<tr>
<td>Warehouse</td>
<td>-5.56%</td>
<td>-5.56%</td>
<td>-5.56%</td>
<td>21.03%</td>
<td>21.03%</td>
</tr>
<tr>
<td>Specialty Bike Shop</td>
<td>8.78%</td>
<td>8.73%</td>
<td>8.78%</td>
<td>11.80%</td>
<td>11.80%</td>
</tr>
<tr>
<td>Value Added Reseller</td>
<td>-5.56%</td>
<td>-5.56%</td>
<td>-5.56%</td>
<td>21.03%</td>
<td>21.03%</td>
</tr>
</tbody>
</table>
12. Construct and execute the following query to see how Boolean operators work.
   Here, you'll create five calculated members to use the equals (=) operator. Review
   the results. Boolean operators return true or false. Even though you're not actually
   pulling and data from the Adventureworks cube, you still have to include a cube
   space in the query.

   **The below code is in C:\Labs-HSQL18\Solution\Module2\MDX-BooleanOperator.MDX**

   ```mdx
   With
   Member [Measures].bool1 as 1=1
   Member [Measures].bool2 as 1=0
   Member [Measures].bool3 as null=null
   Member [Measures].bool4 as 0=null
   Member [Measures].bool5 as 1=null
   Select
   {[Measures].bool1,[Measures].bool2,[Measures].bool3,[Measures].bool4,[Measures].bool5}
   On 0
   From [Adventure Works]
   ```

   **Results:**

<table>
<thead>
<tr>
<th>bool1</th>
<th>bool2</th>
<th>bool3</th>
<th>bool4</th>
<th>bool5</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

13. Construct and execute the following query to use the IS comparison operator. This
   is a simple calculated member that returns true if

   **The below code is in C:\Labs-HSQL18\Solution\Module2\MDX-Objec_Comparison.MDX**

   ```mdx
   With
   Member [Measures].[Is_Clothing] AS
   [Product].[Category].CurrentMember IS [Product].[Category].&[3]
   SELECT
   {[Measures].[Is_Clothing]} ON 0,
   [Product].[Category].[Category].Members ON 1
   FROM
   [Adventure Works]
   ```

   **Results:**
14. Construct and execute the following query to use a range operator. This query shows aggregated Internet sales amounts for each product category from January 2008 – March 2008. The date set is expressed as a range.

**The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-RangeOperator.MDX**

```mdx
SELECT
    {[Ship Date].[Calendar].[Month].&[2008]&[1] : [Ship Date].[Calendar].[Month].&[2008]&[3]} ON 0,
    [Product].[Category].[Category].Members ON 1
FROM [Adventure Works]
WHERE [Measures].[Internet Sales Amount]
```

Results:

<table>
<thead>
<tr>
<th></th>
<th>January 2008</th>
<th>February 2008</th>
<th>March 2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessories</td>
<td>$57,923.18</td>
<td>$58,139.48</td>
<td>$59,392.40</td>
</tr>
<tr>
<td>Bikes</td>
<td>$1,334,494.11</td>
<td>$1,359,229.71</td>
<td>$1,393,460.30</td>
</tr>
<tr>
<td>Clothing</td>
<td>$30,096.07</td>
<td>$28,137.79</td>
<td>$28,233.43</td>
</tr>
<tr>
<td>Components</td>
<td>[null]</td>
<td>[null]</td>
<td>[null]</td>
</tr>
</tbody>
</table>

15. Construct and execute the following query. This query uses the exception (-) operator to show the reseller tax amount measure for all provinces in Canada except for Alberta.

**The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-ExceptOperator.MDX**

```mdx
SELECT
    [Measures].[Reseller Tax Amount] ON COLUMNS,
    [Geography].[Geography].[Country].&[Canada].Children
    - [Geography].[Geography].[State-Province].&[AB]&[CA] ON ROWS
FROM [Adventure Works]
```

Results:
<table>
<thead>
<tr>
<th>Reseller Tax Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>British Columbia</td>
</tr>
<tr>
<td>Brunswick</td>
</tr>
<tr>
<td>Manitoba</td>
</tr>
<tr>
<td>Ontario</td>
</tr>
<tr>
<td>Quebec</td>
</tr>
</tbody>
</table>
Task 5: Working with .members and .allmembers functions

1. Construct and execute the following query to return the members of the measures dimension across the tuple set of bikes. Remember that calculated members will excluded from the set.

The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-Members.MDX

```mdx
select
    {[Product].[Product Categories].[Category].&[1]} on 0,
    {[Measures].members} on 1
from [Adventure Works]
```

Results:

<table>
<thead>
<tr>
<th></th>
<th>Bikes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Sales</td>
<td>$20,316,144.65</td>
</tr>
<tr>
<td>Internet Order</td>
<td>15,205</td>
</tr>
<tr>
<td>Internet Extended</td>
<td>$20,316,144.65</td>
</tr>
<tr>
<td>Internet Tax</td>
<td>$2,265,451.62</td>
</tr>
<tr>
<td>Internet Freight</td>
<td>$707,954.31</td>
</tr>
<tr>
<td>Internet Total</td>
<td>$16,812,348.15</td>
</tr>
<tr>
<td>Internet Standard</td>
<td>$16,812,348.15</td>
</tr>
<tr>
<td>Internet Order</td>
<td>15,205</td>
</tr>
<tr>
<td>Customer Count</td>
<td>9,132</td>
</tr>
<tr>
<td>Reseller Sales</td>
<td>$66,302,381.56</td>
</tr>
<tr>
<td>Reseller Order</td>
<td>75,015</td>
</tr>
</tbody>
</table>

1. Change the above query to .allmembers and re-run. You should now see calculated members. To view calculated members, look at the icon in the metadata browser.

```mdx
select
    {[Product].[Product Categories].[Category].&[1]} on 0,
    {[Measures].allmembers} on 1
from [Adventure Works]
```
Task 6: Working with the .Descendants Function

**Movie: Mod 2.16 MDX Operators**

1. Construct and execute the following query. This query shows reseller sales amounts on columns, and all US states on rows. The DESC flag of BEFORE (level city), tells the query to return everything after United States (Country Level) and before the city level. That leaves all members of the of the US State-Province level. So the results show all U.S. states.

   The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-Descendants1.MDX

   ```
   SELECT
   [Measures].[Reseller Sales Amount] on 0,
   Descendants
   ([Geography].[Geography].[Country].&[United States],
    [Geography].[Geography].[City], BEFORE
   ) ON 1
   FROM [Adventure Works]
   Results:
<table>
<thead>
<tr>
<th></th>
<th>Reseller Sales Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States</td>
<td>$53,607,801.21</td>
</tr>
<tr>
<td>Alabama</td>
<td>$45,429.03</td>
</tr>
<tr>
<td>Arizona</td>
<td>$1,432,585.57</td>
</tr>
<tr>
<td>California</td>
<td>$9,764,270.71</td>
</tr>
<tr>
<td>Colorado</td>
<td>$2,395,913.84</td>
</tr>
<tr>
<td>Connecticut</td>
<td>$1,125,700.41</td>
</tr>
<tr>
<td>Florida</td>
<td>$2,299,888.87</td>
</tr>
<tr>
<td>Georgia</td>
<td>$1,044,577.80</td>
</tr>
<tr>
<td>....</td>
<td></td>
</tr>
</tbody>
</table>
   ```

2. Construct and execute the following query. This time, the county of United States is replaced with the County level of the Geography hierarchy (all countries). Now all cities from all counties are calculated and displayed.

   The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-Descendants2.MDX

   ```
   SELECT
   [Measures].[Reseller Sales Amount] on 0,
   Descendants
   ([Geography].[Geography].[Country]  
   ) ON 1
   FROM [Adventure Works]
   ```
, [Geography].[Geography].[City], BEFORE
) ON 1
FROM [Adventure Works]
Results: Shows State-Province members for all countries, and totals for each country.

<table>
<thead>
<tr>
<th>Country</th>
<th>Reseller Sales Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Northern-Westalen</td>
<td>$223,919.84</td>
</tr>
<tr>
<td>Saaland</td>
<td>$807,718.47</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>$4,279,006.83</td>
</tr>
<tr>
<td>England</td>
<td>$4,279,006.83</td>
</tr>
<tr>
<td>United States</td>
<td>$53,537,801.21</td>
</tr>
<tr>
<td>Alabama</td>
<td>$45,423.03</td>
</tr>
<tr>
<td>Arizona</td>
<td>$1,432,585.57</td>
</tr>
<tr>
<td>California</td>
<td>$3,764,270.71</td>
</tr>
<tr>
<td>Colorado</td>
<td>$2,395,913.84</td>
</tr>
<tr>
<td>Connecticut</td>
<td>$1,125,700.41</td>
</tr>
<tr>
<td>Florida</td>
<td>$2,299,000.07</td>
</tr>
<tr>
<td>Georgia</td>
<td>$1,044,577.80</td>
</tr>
<tr>
<td>Idaho</td>
<td>$227,301.75</td>
</tr>
</tbody>
</table>

3. Construct and execute the following query. This query uses the hierarchy level ordinal, [Geography].[Geography].Levels (4), to specify what levels will be included. In this case, the set expression is starting at the country level (all members), and goes down to level 4 (Postal Code), and only displays members before the postal code because of the DESC BEFORE flag. Notice below that postal code is level 4 (numbered 1, 2, 3, 4 ...) from left to right.

The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-Descendants3.MDX

```
SELECT [Measures].[Reseller Sales Amount] on 0,
Descendants
    ([Geography].[Geography].[Country]
        , [Geography].[Geography].Levels (4)
    , BEFORE
    ) ON 1
FROM [Adventure Works]
```

Results on the next page:
Results: Shows all geography members except for postal codes (countries, state-provinces, and cities).

<table>
<thead>
<tr>
<th></th>
<th>Reseller Sales Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia</td>
<td>1,594,335.38</td>
</tr>
<tr>
<td>New South Wales</td>
<td>1,129,270.79</td>
</tr>
<tr>
<td>Alexandria</td>
<td>(null)</td>
</tr>
<tr>
<td>Coffs Harbour</td>
<td>(null)</td>
</tr>
<tr>
<td>Darlinghur</td>
<td>7,337.60</td>
</tr>
<tr>
<td>Gouburn</td>
<td>(null)</td>
</tr>
<tr>
<td>Lane Cove</td>
<td>148,995.51</td>
</tr>
<tr>
<td>Lavender Bay</td>
<td>278,391.16</td>
</tr>
</tbody>
</table>

4. Construct and execute the following query. This example uses the DESC flag of SELF_AND_BEFORE to include level 4 members (Postal Codes).

5. The below code is in C:\Labs-HSQL1018\Solution\Module2\MDX-Descendants4.MDX

```sql
SELECT [Measures].[Reseller Sales Amount] on 0,
Descendants
    ([Geography].[Geography].[Country]
    , [Geography].[Geography].Levels (4)
    , SELF_AND_BEFORE
 ) ON 1
FROM [Adventure Works]

Results: Postal coded included now.

<table>
<thead>
<tr>
<th></th>
<th>7,337.60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Darlinghur</td>
<td>7,337.60</td>
</tr>
<tr>
<td>2010</td>
<td>(null)</td>
</tr>
<tr>
<td>Gouburn</td>
<td>(null)</td>
</tr>
<tr>
<td>2580</td>
<td>(null)</td>
</tr>
</tbody>
</table>
```