Querying Data Warehouses
(MDX language)

Helena Galhardas
DEI/IST

References

• A. Vaisman and E. Zimányi, Data Warehouse Systems: Design and Implementation, Springer, 2014 (chpt 6)
Outline

- Introduction to the MDX Language
  - MDX Basics
  - Slicing
  - Navigation
  - Cross Join
  - Calculated Members
  - Relative Navigation
- Advanced MDX
- Comparison of MDX and SQL

What is MDX?

- [https://en.wikipedia.org/wiki/MultiDimensional_eXpressions](https://en.wikipedia.org/wiki/MultiDimensional_eXpressions)
Introduction to the MDX Language

• Given a simple three-dimensional cube with one measure:

Two fundamental concepts in MDX: tuples and sets
– A tuple identifies a single cell in a multidimensional cube
– To identify the cell in the top left corner (value 21) we provide the coordinates of each dimension:

• Two fundamental concepts in MDX: tuples and sets
  – A tuple identifies a single cell in a multidimensional cube
  – To identify the cell in the top left corner (value 21) we provide the coordinates of each dimension:

Tuples

• A tuple is defined by specifying one member from one or several dimensions of the cube

• Specify the coordinate for each of the dimensions in the format:
  \( \text{Dimension.Level.Member} \)

• The order of the members is not significant; these two ways are equivalent:

• Since a tuple points to a single cell, each member in the tuple must belong to a different dimension

• A tuple does not need to specify a member from every dimension:
  \( (\text{Customer.City.Paris}) \)

• The tuple below points to the sales of beverages in Paris:

• If a member for a dimension is not specified, the default member for the dimension is implied (typically the All member)
Set

- A collection of tuples defined using the same dimensions:
  
  \[
  \]

- This set points to the cell with value 21, and the one behind it with value 12

Tuples and Hierarchies

- Suppose the hierarchy in the Customer dimension:
  
  - Customer -> City -> State -> Country

- The tuple:
  

points to the cell corresponding to total sales of beverages in France in the first quarter

- Uses the aggregated member France
Measures

• In MDX, measures act like dimensions
• If there are three measures in our cube: UnitPrice, Discount, and SalesAmount, then:
  – The Measures dimension (exists in every cube), contains three members
  – We can specify the measure we want as in the following tuple
• If a measure is not specified, a default measure is used

Basic MDX Queries

• The syntax of a typical MDX query is:
  
  SELECT <axis specification>  
  FROM <cube>  
  [ WHERE <slicer specification> ]

• MDX resembles SQL, but differ in many ways:
  – The axis specification allows to state the axes of a query as well as the members selected for each of these axis
  – Up to 128 axes in an MDX query: each axis has a number: 0 for the x-axis, 1 for the y-axis, 2 for the z-axis, ...
  – The first axes have predefined names: COLUMNS, ROWS, PAGES, CHAPTERS, and SECTIONS; query axes cannot be skipped, e.g., a query cannot have a ROWS axis without a COLUMNS axis
  – The slicer specification on the WHERE clause is optional
    • If not specified, the query returns the default measure for the cube
## Basic MDX Queries: simple form

- **Simplest form of an axis specification**: take the members of the required dimension, including those of the special Measures dimension.

- **Example**: Display all the measures for customers summarized at the country level.

  ```mdx
  SELECT [Measures].MEMBERS ON COLUMNS,
          [Customer].[Country].MEMBERS ON ROWS
  FROM Sales
  ```

- Square brackets optional except for a name with embedded spaces, numbers, MDX keyword.

### Example Table

<table>
<thead>
<tr>
<th></th>
<th>UnitPrice</th>
<th>Quantity</th>
<th>Discount</th>
<th>Sales Amount</th>
<th>Freight</th>
<th>Sales Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Austria</td>
<td>€84.77</td>
<td>4,644</td>
<td>21.71%</td>
<td>€115,328.31</td>
<td>€6,827.10</td>
<td>114</td>
</tr>
<tr>
<td>Belgium</td>
<td>€64.65</td>
<td>1,242</td>
<td>9.72%</td>
<td>€30,505.06</td>
<td>€1,179.53</td>
<td>49</td>
</tr>
<tr>
<td>Denmark</td>
<td>€70.28</td>
<td>1,156</td>
<td>17.94%</td>
<td>€32,428.94</td>
<td>€1,377.75</td>
<td>45</td>
</tr>
<tr>
<td>Finland</td>
<td>€54.41</td>
<td>848</td>
<td>9.09%</td>
<td>€17,530.05</td>
<td>€27.45</td>
<td>51</td>
</tr>
<tr>
<td>France</td>
<td>€64.51</td>
<td>3,052</td>
<td>11.76%</td>
<td>€77,056.01</td>
<td>€3,991.42</td>
<td>172</td>
</tr>
<tr>
<td>Germany</td>
<td>€79.54</td>
<td>8,670</td>
<td>19.26%</td>
<td>€219,356.08</td>
<td>€10,459.01</td>
<td>309</td>
</tr>
<tr>
<td>Ireland</td>
<td>(Null)</td>
<td>(Null)</td>
<td>(Null)</td>
<td>(Null)</td>
<td>(Null)</td>
<td>(Null)</td>
</tr>
</tbody>
</table>

### Slicing (1)

- **Example**: show all measures by year.

  ```mdx
  SELECT Measures.MEMBERS ON COLUMNS,
         [Order Date].[Year].MEMBERS ON ROWS
  FROM Sales
  ```

<table>
<thead>
<tr>
<th></th>
<th>UnitPrice</th>
<th>Quantity</th>
<th>Discount</th>
<th>Sales Amount</th>
<th>Freight</th>
<th>Sales Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>€134.14</td>
<td>46,388</td>
<td>27.64%</td>
<td>€1,145,155.86</td>
<td>€58,587.49</td>
<td>1,931</td>
</tr>
<tr>
<td>1996</td>
<td>€99.55</td>
<td>8,775</td>
<td>21.95%</td>
<td>€191,849.87</td>
<td>€9,475.00</td>
<td>371</td>
</tr>
<tr>
<td>1997</td>
<td>€116.63</td>
<td>23,461</td>
<td>25.89%</td>
<td>€570,199.61</td>
<td>€29,880.49</td>
<td>982</td>
</tr>
<tr>
<td>1998</td>
<td>€205.38</td>
<td>14,152</td>
<td>35.74%</td>
<td>€383,106.38</td>
<td>€19,232.00</td>
<td>578</td>
</tr>
</tbody>
</table>

- To restrict the result to Belgium, we can write.

  ```mdx
  SELECT Measures.MEMBERS ON COLUMNS,
         [Order Date].[Year].MEMBERS ON ROWS
  FROM Sales
  WHERE (Customer.Country.Belgium)
  ```

- The added condition (WHERE clause) only changes the values returned for each cell; not what is returned on the axes.
Slicing (2)

- Multiple members from different hierarchies can be added to the WHERE clause
- **Example:** All measures for all years for customers from Belgium who bought products in the category beverages
  
  ```sql
  SELECT Measures.MEMBERS ON COLUMNS,
             [Order Date].Year.MEMBERS ON ROWS
  FROM Sales
  ```

- For multiple members from the same hierarchy we need to include a set
- **Example:** All measures for all years for customers who bought Beverages and live in either Belgium or France
  
  ```sql
  SELECT Measures.MEMBERS ON COLUMNS,
             [Order Date].Year.MEMBERS ON ROWS
  FROM Sales
            Product.Categories.Beverages)
  ```

- A set in the WHERE clause implicitly aggregates values for all members in the set

---

Slicing (3)

- Specifying in the WHERE clause the measure to be displayed
- **Example:** The sales amount of customers by country and by year
  
  ```sql
  SELECT [Order Date].Year.MEMBERS ON COLUMNS,
             Customer.Country.MEMBERS ON ROWS
  FROM Sales
  WHERE Measures.[Sales Amount]
  ```

<table>
<thead>
<tr>
<th></th>
<th>All</th>
<th>1996</th>
<th>1997</th>
<th>1998</th>
</tr>
</thead>
<tbody>
<tr>
<td>Austria</td>
<td>€115,328.31</td>
<td>€24,467.52</td>
<td>€55,759.04</td>
<td>€35101.75</td>
</tr>
<tr>
<td>Belgium</td>
<td>€30,505.06</td>
<td>€5,865.10</td>
<td>€9,075.48</td>
<td>€15,564.48</td>
</tr>
<tr>
<td>Danmark</td>
<td>€32,428.93</td>
<td>€2,952.40</td>
<td>€25,192.53</td>
<td>€4,284.00</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Measures and dimensions in the WHERE clause
- **Example:** The sales amount of customers by country and by year for the category beverages
  
  ```sql
  SELECT [Order Date].Year.MEMBERS ON COLUMNS
             Customer.Country.MEMBERS ON ROWS
  FROM Sales
  WHERE (Measures.[Sales Amount], Product.Category.[Beverages])
  ```
Navigation (1)

- The result of the query above contains aggregated values from all the years, including the All column
- To omit the All member we must use the CHILDREN function

$$\text{SELECT [Order Date].Year.CHILDREN ON COLUMNS, ...}$$

- However, rows do not include All because Customer.Country.MEMBERS is shorthand for Customer.Geography.Country.MEMBERS
  - All member is the topmost member above Continent
  - Thus, it selects the members of the Country level of the Geography hierarchy of the Customer dimension
  - All is the topmost member of the hierarchy, not a member of the Country level therefore it does not appear in the result
- Customer has an attribute hierarchy Company Name; thus, using the expression:

  $$\text{Customer.[Company Name].MEMBERS}$$
  - The result will contain the All member, plus the names of all the customers

Relational Representation of the Northwind Data Warehouse (recap.)
Navigation (2)

• Example: Sales amount of customers by year and by state, in France and Italy

```sql
SELECT [Order Date].Year.MEMBERS ON COLUMNS,
    NON EMPTY { Customer.France, Customer.Italy } ON ROWS
FROM Sales
WHERE Measures.[Sales Amount]
```

<table>
<thead>
<tr>
<th></th>
<th>All</th>
<th>1996</th>
<th>1997</th>
<th>1998</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bas-Rhin</td>
<td>€18,534.07</td>
<td>€9,986.20</td>
<td>€7,817.87</td>
<td>€730.00</td>
</tr>
<tr>
<td>Bouches-du-Rhône</td>
<td>€19,373.10</td>
<td>€2,675.88</td>
<td>€10,809.36</td>
<td>€5,887.86</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Reggio Emilia</td>
<td>€6,641.83</td>
<td>€80.10</td>
<td>€3,000.84</td>
<td>€3,560.89</td>
</tr>
<tr>
<td>Torino</td>
<td>(null)</td>
<td>(null)</td>
<td>€249.70</td>
<td>€1,296.00</td>
</tr>
</tbody>
</table>

Navigation (3)

• To drill-down we need the DESCENDANTS function
• Example: Sales amount for German cities

```sql
SELECT [Order Date].Year.MEMBERS ON COLUMNS,
    NON EMPTY DESCENDANTS(Customer.Germany, Customer.City) ON ROWS
FROM Sales
WHERE Measures.[Sales Amount]
```

<table>
<thead>
<tr>
<th></th>
<th>All</th>
<th>1996</th>
<th>1997</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mannheim</td>
<td>€2,381.80</td>
<td>(null)</td>
<td>€1,079.80</td>
</tr>
<tr>
<td>Stuttgart</td>
<td>€8,705.23</td>
<td>€2,956.60</td>
<td>€4,262.83</td>
</tr>
<tr>
<td>München</td>
<td>€26,656.56</td>
<td>€9,748.04</td>
<td>€11,829.78</td>
</tr>
</tbody>
</table>

• By default, the function DESCENDANTS displays only members at the level specified as its second attribute
  – An optional flag can be specified as third argument; it states whether to include or exclude descendants or children before and after the specified level
Navigation: Function Qualifiers

DESCENDANTS(Customer.Germany, Customer.City, <flag>)

- **SELF**: default, displays values for the City level
- **BEFORE**: displays values from the State level up to the Country level
- **SELF AND BEFORE**: displays values from the City level up to the Country level
- **AFTER**: displays values from the Customer level, since it is the only level after City
- **SELF AND AFTER**: displays values from the City and Customer levels
- **BEFORE AND AFTER**: displays values from the Country level to the Customer level, excluding the former
- **SELF BEFORE AFTER**: displays values from the Country level to the Customer level
- **LEAVES**: displays values from the City level, since it is the only leaf level between Country and City
  - If LEAVES is used without specifying the level, as in DESCENDANTS(Customer.Geography.Germany, LEAVES) the leaf level, i.e., Customer will be displayed

---

Navigation (4)

- **ASCENDANTS** returns a set that includes all the ancestors of a member and the member itself
- **Example**: Sales amount measure for a particular customer and all its ancestors

```
SELECT Measures.[Sales Amount] ON COLUMNS,
       ASCENDANTS(Customer.Geography.[Du monde entier]) ON ROWS
FROM Sales
```

<table>
<thead>
<tr>
<th>Sales</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Du monde</td>
<td>1,548.70</td>
</tr>
<tr>
<td>enter</td>
<td></td>
</tr>
<tr>
<td>Nantes</td>
<td>4,720.86</td>
</tr>
<tr>
<td>Loire-Atlantique</td>
<td>4,720.86</td>
</tr>
<tr>
<td>France</td>
<td>77,056.01</td>
</tr>
<tr>
<td>Europe</td>
<td>683,523.76</td>
</tr>
<tr>
<td>All Customers</td>
<td>1,145,155.86</td>
</tr>
</tbody>
</table>
Navigation (5)

- To obtain the result for an ancestor at a specified level, the function ANCESTOR can be used:

```
SELECT Measures.[Sales Amount] ON COLUMNS,
    ANCESTOR(Customer.Geography.[Du monde entier],
        Customer.Geography.State) ON ROWS
FROM Sales
```

Cross Join

- Combines several dimensions in a single axis to display more than 2 axes
- **Example:** Sales amount by product category, country and quarter (customer and time combined in same axis)

```
SELECT Product.Category.MEMBERS ON COLUMNS,
    CROSSJOIN(Customer.Country.MEMBERS,
        [Order Date].Calendar.Quarter.MEMBERS) ON ROWS
FROM Sales
WHERE Measures.[Sales Amount]
```

- Can also use the cross join operator `*`

```
SELECT Product.Category.MEMBERS ON COLUMNS,
    Customer.Country.MEMBERS *
    [Order Date].Calendar.Quarter.MEMBERS) ON ROWS
FROM Sales
WHERE Measures.[Sales Amount]
```
Example: result

<table>
<thead>
<tr>
<th></th>
<th>Beverages</th>
<th>Condiments</th>
<th>Confections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Austria</td>
<td>Q3 1996</td>
<td>€708.80</td>
<td>€884.00</td>
</tr>
<tr>
<td>Austria</td>
<td>Q4 1996</td>
<td>€12,955.60</td>
<td>€703.60</td>
</tr>
<tr>
<td>Austria</td>
<td>Q1 1997</td>
<td>(null)</td>
<td>€3,097.50</td>
</tr>
<tr>
<td>Austria</td>
<td>Q2 1997</td>
<td>€1,287.50</td>
<td>€1,390.95</td>
</tr>
</tbody>
</table>

... ... ... ...

Cross Join

- More than two crossjoins:

  ```sql
  SELECT Product.Category.MEMBERS ON COLUMNS,
         Customer.Country.MEMBERS *
         [Order Date].Calendar.Quarter.MEMBERS *
         Shipper.[Company Name].MEMBERS ON ROWS
  FROM Sales
  WHERE Measures.[Sales Amount]
  ```

<table>
<thead>
<tr>
<th></th>
<th>Beverages</th>
<th>Condiments</th>
<th>Confections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Austria</td>
<td>Q3 1996</td>
<td>All</td>
<td>€708.80</td>
</tr>
<tr>
<td>Austria</td>
<td>Q4 1996</td>
<td>Federal Shipping</td>
<td>€100.80</td>
</tr>
<tr>
<td>Austria</td>
<td>Q3 1996</td>
<td>Speedy Express</td>
<td>(null)</td>
</tr>
<tr>
<td>Austria</td>
<td>Q3 1996</td>
<td>United Package</td>
<td>(null)</td>
</tr>
<tr>
<td>Austria</td>
<td>Q4 1996</td>
<td>All</td>
<td>€12,995.60</td>
</tr>
</tbody>
</table>
Calculated Members and Named Sets

- Calculated members define new members in a dimension, or new measures, computed at runtime
  WITH MEMBER Parent.MemberName AS < expression >
- Named sets define new sets
  WITH SET SetName AS < expression >
- **Example**: A measure calculating the percentage profit of sales

```plaintext
WITH MEMBER Measures.Profit% AS
  (Measures.[Sales Amount] - Measures.[Freight]) / Measures.[Sales Amount], FORMAT STRING = '#0.00%'
SELECT { [Sales Amount], Freight, Profit% } ON COLUMNS,
         Customer.Country ON ROWS
FROM Sales
```

**Example: result**

<table>
<thead>
<tr>
<th></th>
<th>Sales Amount</th>
<th>Freight</th>
<th>Profit%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Austria</td>
<td>€115,328.31</td>
<td>6,827.10</td>
<td>94.08%</td>
</tr>
<tr>
<td>Belgium</td>
<td>€30,505.06</td>
<td>1,179.53</td>
<td>96.13%</td>
</tr>
<tr>
<td>Denmark</td>
<td>€32,428.94</td>
<td>1,377.75</td>
<td>95.75%</td>
</tr>
<tr>
<td>Finland</td>
<td>€17,530.05</td>
<td>827.45</td>
<td>95.28%</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>


Static Named Sets

- **Example**: Nordic Countries composed of Denmark, Finland, Norway, and Sweden

  ```sql
  WITH SET [Nordic Countries] AS
  SELECT Measures.MEMBERS ON COLUMNS,
  [Nordic Countries] ON ROWS
  FROM Sales
  ```

<table>
<thead>
<tr>
<th>Country</th>
<th>UnitPrice</th>
<th>Quantity</th>
<th>Discount</th>
<th>Sales Amount</th>
<th>Freight</th>
<th>Sales Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denmark</td>
<td>€70.28</td>
<td>1,156</td>
<td>17.94%</td>
<td>€32,428.94</td>
<td>€1,377.75</td>
<td>45</td>
</tr>
<tr>
<td>Finland</td>
<td>€54.41</td>
<td>848</td>
<td>9.09%</td>
<td>€17,530.05</td>
<td>€827.45</td>
<td>51</td>
</tr>
<tr>
<td>Norway</td>
<td>€97.95</td>
<td>152</td>
<td>0.00%</td>
<td>€5,321.15</td>
<td>€257.45</td>
<td>15</td>
</tr>
<tr>
<td>Sweden</td>
<td>€68.73</td>
<td>2,149</td>
<td>19.57%</td>
<td>€51,292.64</td>
<td>€3,032.12</td>
<td>94</td>
</tr>
</tbody>
</table>

- Static Named Set because it is defined by enumerating its members

Dynamic Named Sets

- **Top five selling products**

  ```sql
  WITH SET TopFiveProducts AS
  TOPCOUNT ( Product.Categories.Product.MEMBERS, 5,
  Measures.[Sales Amount] )
  SELECT { [Unit Price], Quantity, Discount, [Sales Amount] } ON COLUMNS,
  TopFiveProducts ON ROWS
  FROM Sales
  ```

<table>
<thead>
<tr>
<th>Product</th>
<th>UnitPrice</th>
<th>Quantity</th>
<th>Discount</th>
<th>Sales Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cte de Blaye</td>
<td>€256.63</td>
<td>623</td>
<td>4.78%</td>
<td>€141,396.74</td>
</tr>
<tr>
<td>Raclette Courdavault</td>
<td>€53.17</td>
<td>1,369</td>
<td>3.96%</td>
<td>€65,658.45</td>
</tr>
<tr>
<td>Thringen Rostbratwurst</td>
<td>€115.24</td>
<td>596</td>
<td>6.21%</td>
<td>€63,657.02</td>
</tr>
<tr>
<td>Tarte au Sucre</td>
<td>€46.56</td>
<td>1,068</td>
<td>5.53%</td>
<td>€46,643.97</td>
</tr>
<tr>
<td>Camembert Pierrot</td>
<td>€34.32</td>
<td>1,498</td>
<td>7.21%</td>
<td>€44,200.68</td>
</tr>
</tbody>
</table>
Relative Navigation

- Methods to traverse a hierarchy: CURRENTMEMBER, PREVMEMBER, NEXTMEMBER, PARENT, FIRSTCHILD, LASTCHILD
- **Example:** Sales of a member of the Geography hierarchy as a percentage of the sales of its parent

```plaintext
WITH MEMBER Measures.[Percentage Sales] AS
    (Measures.[Sales Amount],
     Customer.Geography.CURRENTMEMBER) / 
    (Measures.[Sales Amount],
     Customer.Geography.CURRENTMEMBER.PARENT),
FORMAT STRING = '#0.00%
SELECT { Measures.[Sales Amount], Measures.[Percentage Sales] } ON COLUMNS,
DESCENDANTS(Customer.Europe, Customer.Country, SELF AND BEFORE) ON ROWS
FROM Sales
```

- **CURRENTMEMBER** returns the current member along a dimension during an iteration; can be abbreviated to:

```
(Measures.[Sales Amount]) / (Measures.[Sales Amount],
Customer.Geography.CURRENTMEMBER.PARENT)
```

---

Example: result

<table>
<thead>
<tr>
<th></th>
<th>Sales Amount</th>
<th>Percentage Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Europe</td>
<td>€683,523.76</td>
<td>59.69%</td>
</tr>
<tr>
<td>Austria</td>
<td>€115,328.31</td>
<td>16.87%</td>
</tr>
<tr>
<td>Belgium</td>
<td>€30,505.06</td>
<td>4.46%</td>
</tr>
<tr>
<td>Denmark</td>
<td>€32,428.94</td>
<td>4.74%</td>
</tr>
<tr>
<td>Finland</td>
<td>€17,530.05</td>
<td>2.56%</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Outline

• Introduction to the MDX Language

> Advanced MDX
  – Time Series Functions
  – Sorting and Filtering
  – Top and Bottom Analysis
  – Aggregation

• Comparison of MDX and SQL

Time Series Functions

• \textbf{PARALLELPERIOD}: compares values of a specified member with those of a member in the same relative position in a prior period

\begin{verbatim}
WITH MEMBER Measures.[Previous Year] AS
  (Measures.[Net Sales], PARALLELPERIOD([Order Date].Calendar.Quarter, 4)), FORMAT STRING = '€ ###,##0.00'
MEMBER Measures.[Net Sales Growth] AS
Measures.[Net Sales] - Measures.[Previous Year],
FORMAT STRING = ' € ###,##0.00; € -###,##0.00'
SELECT { [Net Sales], [Previous Year], [Net Sales Growth] } ON COLUMNS,
[Order Date].Calendar.Quarter ON ROWS
FROM Sales
\end{verbatim}
Example: Result

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Net Sales</th>
<th>Previous Year</th>
<th>Sales Growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q3 1996</td>
<td>€67,531.59</td>
<td>(null)</td>
<td>€67,531.59</td>
</tr>
<tr>
<td>Q3 1996</td>
<td>€114,843.27</td>
<td>(null)</td>
<td>€114,843.27</td>
</tr>
<tr>
<td>Q1 1997</td>
<td>€125,174.40</td>
<td>(null)</td>
<td>€125,174.40</td>
</tr>
<tr>
<td>Q2 1997</td>
<td>€121,518.78</td>
<td>(null)</td>
<td>€121,518.78</td>
</tr>
<tr>
<td>Q3 1997</td>
<td>€133,636.32</td>
<td>€67,531.59</td>
<td>€66,104.73</td>
</tr>
<tr>
<td>Q4 1997</td>
<td>€159,989.61</td>
<td>€114,843.27</td>
<td>€45,146.34</td>
</tr>
<tr>
<td>Q1 1998</td>
<td>€259,322.36</td>
<td>€125,174.40</td>
<td>€134,147.95</td>
</tr>
<tr>
<td>Q2 1998</td>
<td>€104,552.03</td>
<td>€121,518.78</td>
<td>€-16,966.75</td>
</tr>
</tbody>
</table>

Moving Average

- The LAG function, combined with the Range operator ‘:’ help us to write moving averages in MDX
  - Range operator returns a set of members made of two given members and all the members in between

- Example: Three-month moving average of the number of orders
  ```mdx
  WITH MEMBER Measures.MovAvg3Months AS
  AVG([Order Date].Calendar.CURRENTMEMBER.LAG(2): [Order Date].Calendar.CURRENTMEMBER, Measures.[OrderNo]),
  FORMAT STRING = '###,##0.00'
  SELECT { Measures.[Order No], MovAvg3Months } ON COLUMNS,
  [Order Date].Calendar.Month.MEMBERS ON ROWS
  FROM Sales
  WHERE (Measures.MovAvg3Months)
  ```
  - LAG(2) function obtains the month that is two months before the current one
  - Range returns the set containing the three months over which the average is computed
Example: result

<table>
<thead>
<tr>
<th>Date</th>
<th>Order No</th>
<th>MovAvg3Months</th>
</tr>
</thead>
<tbody>
<tr>
<td>July 1996</td>
<td>21</td>
<td>21.00</td>
</tr>
<tr>
<td>August 1996</td>
<td>25</td>
<td>23.00</td>
</tr>
<tr>
<td>September 1996</td>
<td>21</td>
<td>22.33</td>
</tr>
<tr>
<td>October 1996</td>
<td>25</td>
<td>23.67</td>
</tr>
<tr>
<td>November 1996</td>
<td>25</td>
<td>23.67</td>
</tr>
<tr>
<td>December 1996</td>
<td>29</td>
<td>26.33</td>
</tr>
</tbody>
</table>

Filtering

- Allows to reduce the number of axis members that are displayed
- **Example:** Sales amount in 1997 by city and product category, only for cities whose sales amount exceed €25,000

```
SELECT Product.Category.MEMBERS ON COLUMNS,
FILTER(Customer.City.MEMBERS, (Measures.[Sales Amount],[Order Date].Calendar.[1997])>25000) ON ROWS
FROM Sales
WHERE (Measures.[Net Sales Growth],
[Order Date].Calendar.[1997])
```
Example: result

<table>
<thead>
<tr>
<th>Graz</th>
<th>€2,370.58</th>
<th>€6,114.67</th>
<th>€8,581.51</th>
<th>€7,171.01</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>CuneWald</td>
<td>€6,966.40</td>
<td>€2,610.51</td>
<td>€8,821.85</td>
<td>€7,144.74</td>
<td>...</td>
</tr>
<tr>
<td>London</td>
<td>€2,088.23</td>
<td>€683.88</td>
<td>€1,942.56</td>
<td>€83.13</td>
<td>...</td>
</tr>
<tr>
<td>Montral</td>
<td>€9,142.78</td>
<td>€2,359.90</td>
<td>€213.93</td>
<td>€3,609.16</td>
<td>...</td>
</tr>
<tr>
<td>Boise</td>
<td>€1,871.10</td>
<td>€94.84</td>
<td>€4,411.46</td>
<td>€6,522.61</td>
<td>...</td>
</tr>
</tbody>
</table>

Sorting (1)

- All the members in a dimension have a hierarchical order, e.g.:
  ```sql
  SELECT Measures.MEMBERS ON COLUMNS,
         Customer.Geography.Country.MEMBERS ON ROWS
  FROM Sales
  ```

<table>
<thead>
<tr>
<th>UnitPrice</th>
<th>Quantity</th>
<th>Discount</th>
<th>Sales Amount</th>
<th>Freight</th>
<th>Sales Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Austria</td>
<td>€84.77</td>
<td>4,644</td>
<td>21.71%</td>
<td>€115,328.3</td>
<td>€6,827.10</td>
</tr>
<tr>
<td>Belgium</td>
<td>€64.65</td>
<td>1,242</td>
<td>9.72%</td>
<td>€30,505.06</td>
<td>€1,179.53</td>
</tr>
<tr>
<td>Denmark</td>
<td>€70.28</td>
<td>1,156</td>
<td>17.94%</td>
<td>€32,428.94</td>
<td>€1,377.75</td>
</tr>
<tr>
<td>Finland</td>
<td>€54.41</td>
<td>348</td>
<td>9.09%</td>
<td>€17,530.05</td>
<td>€827.45</td>
</tr>
<tr>
<td>France</td>
<td>€64.51</td>
<td>3,052</td>
<td>11.76%</td>
<td>€77,056.01</td>
<td>€3,991.42</td>
</tr>
</tbody>
</table>

- Countries are displayed according to the order of the hierarchy: first the European countries, then the North American countries, etc., i.e., according to the ordering of the parent level of country (Area)

- To sort countries by their name, we can use the ORDER function:
  ```sql
  ORDER(Set, Expression [, ASC | DESC | BASC | BDESC])
  ```
Sorting (2)

- **Example**: Sorting the set of countries in the previous query skipping the hierarchy
  
  ```sql
  SELECT Measures.MEMBERS ON COLUMNS,
       ORDER(Customer.Geography.Country.MEMBERS,
               Customer.Geography.CURRENTMEMBER.Name,BASC) ON ROWS
  FROM Sales
  ```

- **Property Name** returns the name of a level, dimension, member, or hierarchy

- **Another Example**: Sorting the query result based on the sales amount (a measure)
  
  ```sql
  SELECT Measures.MEMBERS ON COLUMNS,
       ORDER(Customer.Geography.Country.MEMBERS,
               Measures.[Sales Amount],BDESC) ON ROWS
  FROM Sales
  ```

Top and Bottom Analysis

- **HEAD** and **TAIL** functions return the first (last) members in the set based on a number that the user requests

- **Example**: Top three best-selling store cities
  
  ```sql
  SELECT Measures.MEMBERS ON COLUMNS,
       HEAD(ORDER(Customer.Geography.City.MEMBERS,
                   Measures.[Sales Amount],BDESC),3) ON ROWS
  FROM Sales
  ```

<table>
<thead>
<tr>
<th>Unit Price</th>
<th>Quantity</th>
<th>Discount</th>
<th>Sales Amount</th>
<th>Freight</th>
<th>Sales Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cunewalde</td>
<td>101.46</td>
<td>3,616</td>
<td>103,597.43</td>
<td>4,999.77</td>
<td>77</td>
</tr>
<tr>
<td>Boise</td>
<td>90.90</td>
<td>4,809</td>
<td>102,253.85</td>
<td>6,570.58</td>
<td>113</td>
</tr>
<tr>
<td>Graz</td>
<td>88.00</td>
<td>4,045</td>
<td>93,349.45</td>
<td>5,725.79</td>
<td>92</td>
</tr>
</tbody>
</table>

- **Alternatively**, **TOPCOUNT** can be used

  ```sql
  SELECT Measures.MEMBERS ON COLUMNS,
       TOPCOUNT(Customer.Geography.City.MEMBERS,5,
                 Measures.[Sales Amount]) ON ROWS
  FROM Sales
  ```
Aggregation Functions

- MDX provides many aggregation functions: SUM, AVG, MEDIAN, MAX, MIN, VAR, and STDDEV
- **Example:** Total, maximum, minimum, and average sales amount for a one-month period in 1997

```mdx
WITH MEMBER Measures.[Maximum Sales] AS
    MAX(DESCENDANTS([Order Date].Calendar.Year.[1997], [Order Date].Calendar.Month), Measures.[Sales Amount])

MEMBER Measures.[Minimum Sales] AS
    MIN(DESCENDANTS([Order Date].Calendar.Year.[1997], [Order Date].Calendar.Month), Measures.[Sales Amount])

MEMBER Measures.[Average Sales] AS
    AVG(DESCENDANTS([Order Date].Calendar.Year.[1997], [Order Date].Calendar.Month), Measures.[Sales Amount])
```

SELECT { [Sales Amount], [Maximum Sales], [Minimum Sales], [Average Sales] } ON COLUMNS,
      Product.Categories.Category.MEMBERS ON ROWS
FROM Sales

---

**Example: result**

<table>
<thead>
<tr>
<th></th>
<th>Sales Amount</th>
<th>Maximum Sales</th>
<th>Minimum Sales</th>
<th>Average Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beverages</td>
<td>€237,203.91</td>
<td>€21,817.76</td>
<td>€2,109.84</td>
<td>€7,652.65</td>
</tr>
<tr>
<td>Condiments</td>
<td>€91,528.81</td>
<td>€5,629.70</td>
<td>€1,252.33</td>
<td>€3,842.09</td>
</tr>
<tr>
<td>Confections</td>
<td>€162,443.91</td>
<td>€11,538.61</td>
<td>€2,174.89</td>
<td>€6,798.83</td>
</tr>
<tr>
<td>Dairy Products</td>
<td>€221,157.31</td>
<td>€12,992.48</td>
<td>€5,584.84</td>
<td>€9,119.26</td>
</tr>
<tr>
<td>Grains/Cereals</td>
<td>€80,870.58</td>
<td>€6,012.65</td>
<td>€1,891.00</td>
<td>€4,193.64</td>
</tr>
<tr>
<td>Meat/Poultry</td>
<td>€139,428.18</td>
<td>€14,110.16</td>
<td>€1,029.00</td>
<td>€6,217.45</td>
</tr>
<tr>
<td>Produce</td>
<td>€90,216.14</td>
<td>€12,157.90</td>
<td>€1,650.00</td>
<td>€4,429.52</td>
</tr>
<tr>
<td>Seafood</td>
<td>€122,307.02</td>
<td>€8,448.86</td>
<td>€1,587.11</td>
<td>€5,263.19</td>
</tr>
</tbody>
</table>
Outline

• Introduction to the MDX Language
• Advanced MDX
  ➢ Comparison of MDX and SQL

Comparison of MDX and SQL

• Main difference between SQL and MDX: Ability of MDX to reference multiple dimensions
  – Although possible to use SQL to query cubes, MDX’s commands are designed specifically for multidimensional data
  – SQL refers to only two dimensions, columns and rows
• OLAP tools: Cannot display a result set with more than two dimensions.
• SQL’s SELECT clause defines the column layout for a query
• MDX’s SELECT clause is used to define several axis dimensions
• SQL’s WHERE clause used to filter the data returned by a query
• MDX’s WHERE clause used to provide a slice of the data returned by a query
  – The concept of a slice implies a reduction in the number of dimensions
  – Unlike in SQL, the WHERE clause in MDX cannot filter what is returned on an axis of a query. To filter what appears on an axis of a query, we can use functions such as FILTER, NONEMPTY, and TOPCOUNT.
Next Lecture

• Data Analytics: Exploiting the Data Warehouse