Data Analysis and Integration

Lab 10: Analytical queries with MDX

Note: In order to do this lab, you need to have successfully completed the previous lab.

In this lab we will be querying the OLAP cube that we defined in the previous lab using Pentaho Schema Workbench (PSW). The query language that we will be using is MDX. To see the results of the MDX queries, we will be running them on Saiku Business Analytics.

Inspecting the cube

1. Open the **steelwheels_dw.xml** file (from the previous lab) in a Web browser or in a text editor that can understand XML. You should see a structure similar to the following:

```
<Schema name="steelwheels_dw">
  <Cube name="Orders" cache="true" enabled="true">
    <Table name="fact_order">
    </Table>
    <Dimension type="StandardDimension" foreignKey="CUSTOMERNUMBER" name="Customer">
      <Hierarchy name="Customer Hierarchy" allMemberName="All Customers" primaryKey="CUSTOMERNUMBER">
        <Table name="dim_customer">
        </Table>
        <Level name="Country" column="COUNTRY" type="String" levelType="Regular">
        </Level>
        <Level name="City" column="CITY" type="String" levelType="Regular">
        </Level>
        <Level name="Customer Name" column="CUSTOMERNAME" type="String" levelType="Regular">
        </Level>
      </Hierarchy>
    </Dimension>
    <Dimension type="StandardDimension" foreignKey="PRODUCT_ID" name="Product">
      <Hierarchy name="Product Hierarchy" allMemberName="All Products" primaryKey="PRODUCT_ID">
        <Table name="dim_product">
        </Table>
        <Level name="Product Line" column="PRODUCTLINE" type="String" levelType="Regular">
        </Level>
        <Level name="Product Vendor" column="PRODUCTVENDOR" type="String" levelType="Regular">
        </Level>
        <Level name="Product Name" column="PRODUCTNAME" type="String" levelType="Regular">
        </Level>
      </Hierarchy>
    </Dimension>
    <Dimension type="TimeDimension" foreignKey="TIME_ID" name="Time">
      <Hierarchy name="Time Hierarchy" allMemberName="All Years" primaryKey="TIME_ID">
        <Table name="dim_time">
        </Table>
        <Level name="Year" column="YEAR_ID" type="Integer" levelType="TimeYears">
        </Level>
        <Level name="Quarter" column="QTR_NAME" ordinalColumn="QTR_ID" type="String" levelType="TimeQuarters">
        </Level>
        <Level name="Month" column="MONTH_NAME" ordinalColumn="MONTH_ID" type="String" levelType="TimeMonths">
        </Level>
      </Hierarchy>
    </Dimension>
    <Measure name="Sales" column="LINETOTAL" datatype="Numeric" formatString="$ #,###.00" aggregator="sum">
    </Measure>
    <Measure name="Quantity" column="QUANTITYORDERED" datatype="Integer" formatString="#,###" aggregator="sum">
    </Measure>
  </Cube>
</Schema>
```

2. Locate the following elements in the XML file:
   a) The name of the cube.
   b) The fact table.
   c) The dimensions and measures.
   d) The hierarchy for each dimension.
   e) The dimension table for each hierarchy.
   f) The levels of each hierarchy.

3. Have a look at the attributes of each element. This is exactly what you have configured in the previous lab using Pentaho Schema Workbench (PSW).

Querying the cube with MDX

4. Open a new terminal and navigate to the folder: **~/Pentaho/pentaho-server/**

5. Start the Pentaho Server with: **./start-pentaho.sh**
   Note: Pentaho Server will start running on the background, and it may take a few minutes for the startup to complete.

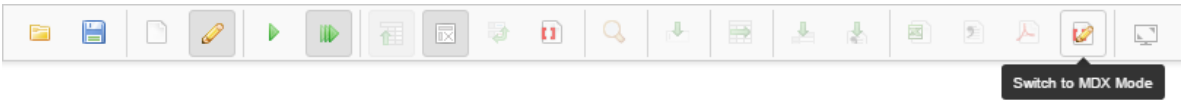
6. Open Firefox and navigate to: **http://localhost:8080/**

7. On the **Welcome** page, press **Log in as an evaluator** and **Log in** as **Administrator**.

8. On the **Home** page, at the top left corner, select **File > New > Saiku Analytics**.

9. In **Select a cube**, select the **Orders** cube under the **steelwheels_dw** data source.

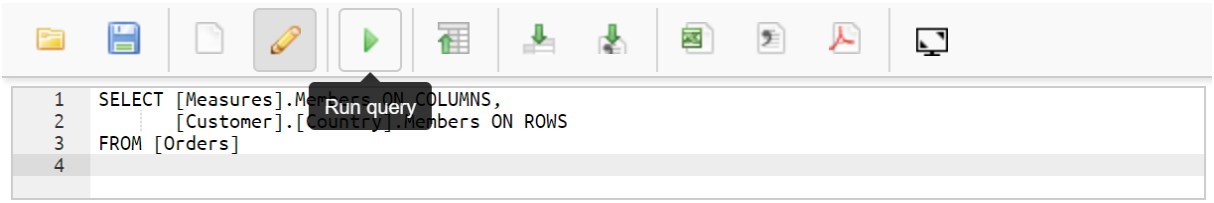
10. In the toolbar, click on the button **Switch to MDX Mode**.



11. Write the following query:

```
SELECT [Measures].Members ON COLUMNS,
       [Customer].[Country].Members ON ROWS
FROM [Orders]
```

12. Execute the query by pressing the "play" button (**Run query**).



13. From the results, try to understand what the query is doing.

Slicing

14. Run the following query and, from the results, try to understand what the query is doing:

```
SELECT Measures.Members ON COLUMNS,
       Time.Year.Members ON ROWS
FROM Orders
WHERE Customer.Country.Italy
```

15. Run the following query and, from the results, try to understand what the query is doing:

```
SELECT Measures.Members ON COLUMNS,
       Time.Year.Members ON ROWS
FROM Orders
WHERE (Customer.Country.Italy, Product.[Product Line].[Classic Cars])
```

16. Run the following query and, from the results, try to understand what the query is doing:

```
SELECT Measures.Members ON COLUMNS,
       Time.Year.Members ON ROWS
FROM Orders
WHERE {(Customer.Country.Italy, Product.[Product Line].[Classic Cars]),
       (Customer.Country.France, Product.[Product Line].[Classic Cars])}
```

17. Run the following query and, from the results, try to understand what the query is doing:

```
SELECT Time.Year.Members ON COLUMNS,
       Customer.Country.Members ON ROWS
FROM Orders
WHERE (Measures.Sales, Product.[Product Line].[Classic Cars])
```

Navigation

18. Run the following query and, from the results, try to understand what the query is doing:

```
SELECT Time.[2003].Children ON COLUMNS,
       Customer.Country.Members ON ROWS
FROM Orders
WHERE Measures.Sales
```

19. Run the following query and, from the results, try to understand what the query is doing:

```
SELECT Time.[2003].Children ON COLUMNS,
       {Customer.Country.France, Customer.Country.Italy} ON ROWS
FROM Orders
WHERE Measures.Sales
```

20. Run the following query and, from the results, try to understand what the query is doing:

```
SELECT Time.[2003].Children ON COLUMNS,
       {Customer.Country.Germany.Children,
        Customer.Country.Italy.Children} ON ROWS
FROM Orders
WHERE Measures.Sales
```

21. Run the following query and, from the results, try to understand what the query is doing:

```
SELECT Time.[2003].Children ON COLUMNS,
       {DRILLDOWNLEVEL(Customer.Country.Germany),
        DRILLDOWNLEVEL(Customer.Country.Italy)} ON ROWS
FROM Orders
WHERE Measures.Sales
```

22. Run the following query and, from the results, try to understand what the query is doing:

```
SELECT Time.[2003].Children ON COLUMNS,
       DESCENDANTS(Customer.Italy,  Customer.City, SELF) ON ROWS
FROM Orders
WHERE Measures.Sales
```

23. In the previous query, try replacing SELF with:
    a) BEFORE
    b) SELF_AND_BEFORE
    c) AFTER
    d) SELF_AND_AFTER
    e) BEFORE_AND_AFTER
    f) SELF_BEFORE_AFTER

24. Run the following query and, from the results, try to understand what the query is doing:

```
SELECT Time.[2003].Children ON COLUMNS,
       ASCENDANTS(Customer.City.Milan) ON ROWS
FROM Orders
WHERE Measures.Sales
```

25. Run the following query and, from the results, try to understand what the query is doing:

```
SELECT Time.[2003].Children ON COLUMNS,
       ANCESTOR(Customer.City.Milan, Customer.Country) ON ROWS
FROM Orders
WHERE Measures.Sales
```

Cross join

26. Run the following query and, from the results, try to understand what the query is doing:

```
SELECT Product.[Product Line].Members ON COLUMNS,
       CROSSJOIN(Customer.Country.Members, Time.Year.Members) ON ROWS
FROM Orders
WHERE Measures.Sales
```

27. Run the following query and check that the results are the same:

```
SELECT Product.[Product Line].Members ON COLUMNS,
       Customer.Country.Members * Time.Year.Members ON ROWS
FROM Orders
WHERE Measures.Sales
```

Calculated members

28. Run the following query and, from the results, try to understand what the query is doing:

```
WITH MEMBER Measures.SalesPerUnit AS (Measures.Sales / Measures.Quantity)
SELECT Measures.SalesPerUnit ON COLUMNS,
       Customer.Country.Members ON ROWS
FROM Orders
```

29. Run the following query and, from the results, try to understand what the query is doing:

```
WITH MEMBER Measures.SalesPerUnit AS (Measures.Sales / Measures.Quantity)
SELECT Measures.AllMembers ON COLUMNS,
       Customer.Country.Members ON ROWS
FROM Orders
```

30. What is the difference between the two previous queries?

31. Run the following query and, from the results, try to understand what the query is doing:

```
WITH MEMBER Measures.Cars AS Product.[Product Line].[Classic Cars] +
                              Product.[Product Line].[Vintage Cars]
SELECT Time.Year.Members ON COLUMNS,
       Measures.Cars ON ROWS
FROM Orders
```

32. Repeat the query above twice:
    - One with Classic Cars only
    - Another with Vintage Cars only

    Check that the results agree with those of the previous query.

Named sets

33. Run the following query and, from the results, try to understand what the query is doing:

```
WITH SET [Nordic Countries] AS {Customer.Country.Denmark,
                                Customer.Country.Finland,
                                Customer.Country.Norway,
                                Customer.Country.Sweden}
SELECT Time.Year.Members ON COLUMNS,
       [Nordic Countries] ON ROWS
FROM Orders
WHERE Measures.Sales
```

34. Run the following query and, from the results, try to understand what the query is doing:

```
WITH SET TopCountries AS TOPCOUNT(Customer.Country.Members, 3, Measures.Sales)
SELECT Measures.Members ON COLUMNS,
       TopCountries ON ROWS
FROM Orders
```

Relative navigation

35. Run the following query and, from the results, try to understand what the query is doing:

```
WITH MEMBER Measures.PercentSales AS
            (Measures.Sales, Customer.Country.CurrentMember) /
            (Measures.Sales, Customer.Country.CurrentMember.Parent),
            FORMAT_STRING = '#0.00%'
SELECT {Measures.Sales, Measures.PercentSales} ON COLUMNS,
       Customer.Country.Members ON ROWS
FROM Orders
```

36. Run the following query and, from the results, try to understand what the query is doing:

```
SELECT Time.Year.Members ON COLUMNS,
       GENERATE({Customer.Country.Belgium, Customer.Country.France},
                DESCENDANTS(Customer.CurrentMember, Customer.City)) ON ROWS
FROM Orders
WHERE Measures.Sales
```

37. Run the following query and, from the results, try to understand what the query is doing:

```
WITH MEMBER Measures.[Previous Month] AS Time.Month.CurrentMember.PrevMember
     MEMBER Measures.[Sales Growth] AS Measures.Sales -
                                       Measures.[Previous Month]
SELECT {Measures.Sales,
        Measures.[Previous Month],
        Measures.[Sales Growth]} ON COLUMNS,
        DESCENDANTS(Time.Year.[2004], Time.Month) ON ROWS
FROM Orders
```

38. Run the following query and, from the results, try to understand what the query is doing:

```
WITH MEMBER Measures.[Previous Year] AS PARALLELPERIOD(Time.Month, 12)
     MEMBER Measures.[Sales Growth] AS Measures.Sales -
                                       Measures.[Previous Year]
SELECT {Measures.Sales,
        Measures.[Previous Year],
        Measures.[Sales Growth]} ON COLUMNS,
        DESCENDANTS(Time.Year.[2004], Time.Month) ON ROWS
FROM Orders
```

39. Why are the results different in the two previous queries?

Filtering

40. Run the following query and, from the results, try to understand what the query is doing:

```
SELECT Measures.Members ON COLUMNS,
       FILTER(Customer.Country.Members, Measures.Sales > 1000000) ON ROWS
FROM Orders
```

41. Run the following query and, from the results, try to understand what the query is doing:

```
SELECT Time.Year.Members ON COLUMNS,
       FILTER(Customer.Country.Members,
              (Measures.Sales, Time.[2004]) > 250000) ON ROWS
FROM Orders
WHERE Measures.Sales
```

Sorting

42. Run the following query:

```
SELECT Measures.Members ON COLUMNS,
       Customer.Country.Members ON ROWS
FROM Orders
```

43. Now run the following query and, from the results, try to understand what the query is doing:

```
SELECT Measures.Members ON COLUMNS,
       ORDER(Customer.Country.Members, Measures.Sales, DESC) ON ROWS
FROM Orders
```

Top analysis

44. Run the following query and, from the results, try to understand what the query is doing:

```
SELECT Measures.Members ON COLUMNS,
       HEAD(ORDER(Customer.Country.Members, Measures.Sales, DESC), 3) ON ROWS
FROM Orders
```

45. The following query will give the same results. Why?

```
SELECT Measures.Members ON COLUMNS,
       TOPCOUNT(Customer.Country.Members, 3, Measures.Sales) ON ROWS
FROM Orders
```

46. The following query will give the same results. Why?

```
SELECT Measures.Members ON COLUMNS,
       TOPPERCENT(Customer.Country.Members, 50, Measures.Sales) ON ROWS
FROM Orders
```

47. Take a screenshot of the previous query and results.

Further reading

48. For more information about the MDX language and available functions, see: http://mondrian.pentaho.com/documentation/mdx.php