pentaho

Version 5.3

Work with Multidimensional Data Models

Copyright Page

Help and Support Resources

To view the most up-to-date help content, visit https://help.pentaho.com.

If you do not find answers to your questions here, please contact your Pentaho technical support representative.

Support-related questions should be submitted through the Pentaho Customer Support Portal at http://support.pentaho.com.

For information about how to purchase support or enable an additional named support contact, please contact your sales representative, or send an email to sales@pentaho.com.

For information about instructor-led training, visit http://www.pentaho.com/training.

Liability Limits and Warranty Disclaimer

Trademarks

Third-Party Open Source Software

For a listing of open source software used by each Pentaho component, navigate to the folder that contains the Pentaho component. Within that folder, locate a folder named licenses. The licenses folder contains HTML.files that list the names of open source software, their licenses, and required attributions.

Contact Us

Global Headquarters Pentaho Corporation Citadel International, Suite 460

5950 Hazeltine National Drive Orlando, FL 32822

Phone: +1 407 812-OPEN (6736)

Fax: +1 407 517-4575

http://www.pentaho.com

Sales Inquiries: sales@pentaho.com

Introduction

This section helps BA administrators prepare their data for use with the Pentaho Analyzer and Pentaho Report Designer. The collection of analysis components in Pentaho Business Analytics enables you to visualize data trends and reveal useful information about your business. You can do this by creating static reports from an analysis data source, traversing an analysis cube through an Analyzer report, showing how data points compare by using charts, and monitoring the status of certain trends and thresholds with dashboards.

Before you can begin using any client tools, you must consolidate data from disparate sources into one canonical source and optimize it for the metrics you want to analyze; create an analysis schema to describe the data; iteratively improve that schema so that it meets your users' needs; and create aggregation tables for frequently computed views.

OLAP Defined

Pentaho Analysis is built on the Mondrian online analytical processing (**OLAP**) engine. OLAP relies on a multidimensional data model that, when queried, returns a dataset that resembles a grid. The rows and columns that describe and bring meaning to the data in that grid are **dimensions**, and the hard numerical values in each cell are the **measures** or **facts**. In Pentaho Analyzer, dimensions are shown in yellow and measures are in blue.

OLAP requires a properly prepared data source in the form of a star or snowflake schema that defines a logical multi-dimensional database and maps it to a physical database model. Once you have your initial data structure in place, you must design a descriptive layer for it in the form of a Mondrian schema, which consists of one or more cubes, hierarchies, and members. Only when you have a tested and optimized Mondrian schema is your data prepared on a basic level for end-user tools like Pentaho Analyzer. See Workflow Overview for a more comprehensive overview of the Pentaho Analysis data preparation workflow, including which Pentaho tools you will need to execute this process.

For concise definitions of OLAP terms, refer to Mondrian Schema Element Quick Reference and the individual element pages it references.

Pentaho Analysis Enterprise Edition Features

Pentaho offers expanded functionality for Pentaho Analysis Enterprise Edition customers:

- The Pentaho Analyzer visualization tool
- A pluggable Enterprise Cache with support for highly scalable, distributable cache implementations including Infinispan, Memcached, and Terracotta BigMemory

Use of these features requires a Pentaho Analysis Enterprise Edition license installed on the BA Server and workstations that have Schema Workbench and Metadata Editor. A special BA Server package must also be installed; this process is covered in the BA installation documentation.

All relevant configuration options for these features are covered in this section.

Workflow Overview

To prepare data for use with the Pentaho Analysis (and Reporting, to a certain extent) client tools, you should follow this basic workflow:

Design a Star or Snowflake Schema

The entire process starts with a data warehouse. This section will not attempt to explain how to build this structure -- there are entire books on the subject, and an entire consulting industry dedicated to it already. The end result should be data model in the star or snowflake schema pattern. You don't have to worry too much about getting the model exactly right on your first try. Just cover all of your anticipated business needs; part of the process is coming back to the data warehouse design step and making changes to your initial data model after you've discovered what your operational needs are.

Populate the Star/Snowflake Schema

Once your data model is designed, the next step is to populate it with actual data, thereby creating your data warehouse. The best tool for this job is Pentaho Data Integration, an enterprise-grade extract, transform, and load (ETL) application.

Build a Mondrian Schema

Now that your initial data warehouse project is complete, you must build a Mondrian schema to organize and describe it in terms that Pentaho Analysis can understand. This is also accomplished through Pentaho Data Integration by using the Agile BI plugin. Just connect to your data warehouse and auto-populate your schema with the Modeler graphical interface. Alternatively, you can use Pentaho Schema Workbench to create an analysis schema through a manual process.

Initial Testing

At this point you should have a multi-dimensional data structure with an appropriate metadata layer. You can now start using Pentaho Analyzer to drill down into your data and see if your first attempt at data modelling was successful. In all likelihood, it will need some adjustment, so take note of all of the schema limitations that you're unhappy with during this initial testing phase.

Do not be concerned with performance issues at this time -- just concentrate on the completeness and comprehensiveness of the data model.

Adjust and Repeat Until Satisfied

Use the notes you took during the testing phase to redesign your data warehouse and Mondrian schema appropriately. Adjust hierarchies and relational measure aggregation methods. Create virtual cubes for analyzing multiple fact tables by conforming dimensions. Re-test the new implementation and continue to refine the data model until it matches your business needs perfectly.

Test for Performance

Once you're satisfied with the design and implementation of your data model, you should try to find performance problems and address them by tuning your data warehouse database, and by creating aggregation tables. The testing can only be reasonably done by hand, using Pentaho Analyzer. Take note of all of the measures that take an unreasonably long time to calculate. Also, enable SQL logging and locate slow-performing queries, and build indexes for optimizing query performance.

Create Aggregation Tables

Using your notes as a guide, create aggregation tables in Pentaho Aggregation Designer to store frequently computed Analyzer reports. Re-test and create new aggregation tables as necessary.

If you are working with a relatively small data warehouse or a limited number of dimensions, you may not have a real need for aggregation tables. However, be aware of the possibility that performance issues may come up in the future. Check in with your users occasionally to see if they have any particular concerns about the speed of their BI content.

Deploy to Production

Your data warehouse and Mondrian schema have been created, tested, and refined. You're now ready to put it all into production. You may need to train or purchase Pentaho training for those in your organization who use Pentaho client tools.

Dimensional Modeling

Dimensional modeling is the process of transforming data from multiple sources in non-human-friendly formats into a single data source that is organized to support business analytics. Below is a typical workflow for developing a dimensional model:

1. Collect user requirements for business logic and processes
2. Considering the entirety of your data, break it down into subjects
3. Isolate groups of facts into one or more fact tables
4. Design dimensional tables that draw relationships between levels (fact groups)
5. Determine which members of each level are useful for each dimensional table
6. Build and publish a Mondrian (Pentaho Analysis) schema and collect feedback from users
7. Refine your model based on user feedback, continue iterating through this list until users are productive

Or, expressed as a series of questions:

1. What topics or subjects are important to the users who are analyzing the data? What do your users need to learn from the data?
2. What are the important details your users will need to examine in the data?
3. How should each data column relate to other data columns?
4. How should datasets be grouped and organized?
5. What are some useful short descriptions for each dimensional level in a hierarchy (for each element, decide what is useful within that element; for instance, in a dimensional table representing time, your levels might be year, month, and day, and your members for the year level might be 2003, 2004, 2005).
6. How effective is this dimensional model for the intended userbase? How can it improve?

The Agile BI tools in Pentaho Data Integration make dimensional modeling much easier than the traditional methods. Through PDI, you can quickly adjust your business logic, the granularity of your fact tables, and the attributes of your dimension tables, then generate a new model and push it out to a test environment for evaluation.

- Virtual OLAP Cubes

Virtual OLAP Cubes

Another name for a dimensional model is a **cube**. Each cube represents one fact table and several dimensional tables. This model should be useful for reporting and analysis on the subject of the data in the fact table. However, if you want to cross-reference this data with another cube -- if you need to analyze data across two or more cubes, or need to combine information from two fact tables on the same subject but with different granularity -- then you must create a **virtual cube**. The XML elements that compose a virtual cube are explained in detail below.

Note: Virtual cubes cannot presently be created through Pentaho Data Integration's model perspective; you must use Schema Workbench instead.

The **<CubeUsages>** element specifies the cubes that are imported into the virtual cube. It holds <CubeUsage> elements.

The **<CubeUsage>** element specifies the base cube that is imported into the virtual cube. Alternatively you can define a **<VirtualCubeMeasure>** and use similar imports from the base cube without defining a <CubeUsage>.The **cubeName** attribute specifies the name of the base cube. The **ignoreUnrelatedDimensions** attribute determines whether or not the measures from this base cube will have non-joining dimension members pushed to the top level member. This attribute is false by default because it is still experimental.

The **<VirtualCubeDimension>** element imports a dimension from one of the constituent cubes. If you do not specify the cubeName attribute, this means you are importing a shared dimension.

Note: If a shared dimension is used more than once in a cube, there is no way to determine which usage of the shared dimension you intend to import.

The **<VirtualCubeMeasure>** element imports a measure from one of the constituent cubes. It is imported with the same name. If you want to create a formula or rename a measure as you import it, use the **<CalculatedMember>** element instead.

Virtual cubes are useful for situations where there are fact tables of different granularities (for instance, one Time fact table might be configured on a Day level, another at the Month level), or fact tables of different dimensionalities (for instance one on Products, Time and Customer, another on Products, Time and Warehouse), and you need to present the results to users who don't know how the data is structured.

Any common dimensions -- shared dimensions which are used by both constituent cubes -- are automatically synchronized. In this example, [Time] and [Products] are common dimensions. So if the context is ([Time].[2005].[Q2], [Products].[Productname].[P-51-D Mustang]), measures from either cube will relate to this context.

Dimensions which only belong to one cube are called **non-conforming dimensions**. The [Gender] dimension is an example of this; it exists in the Sales cube, but not Warehouse. If the context is ([Gender].[F], [Time].[2005].[Q1]), it makes sense to ask the value of the [Unit Sales] measure (which comes from the [Sales] cube) but not the [Units Ordered] measure (from [Warehouse]). In the context of [Gender].[F], [Units Ordered] has value NULL.

```
<VirtualCube name="Warehouse and Sales">
    <CubeUsages>
        <CubeUsage cubeName="Sales" ignoreUnrelatedDimensions="true"/>
        <CubeUsage cubeName="Warehouse"/>
    </CubeUsages>
    <VirtualCubeDimension cubeName="Sales" name="Customers"/>
    <VirtualCubeDimension cubeName="Sales" name="Education Level"/>
    <VirtualCubeDimension cubeName="Sales" name="Gender"/>
    <VirtualCubeDimension cubeName="Sales" name="Marital Status"/>
    <VirtualCubeDimension name="Product"/>
    <VirtualCubeDimension cubeName="Sales" name="Promotion Media"/>
    <VirtualCubeDimension cubeName="Sales" name="Promotions"/>
    <VirtualCubeDimension name="Store"/>
    <VirtualCubeDimension name="Time"/>
    <VirtualCubeDimension cubeName="Sales" name="Yearly Income"/>
    <VirtualCubeDimension cubeName="Warehouse" name="Warehouse"/>
    <VirtualCubeMeasure cubeName="Sales" name="[Measures].[Sales Count]"/>
    <VirtualCubeMeasure cubeName="Sales" name="[Measures].[Store Cost]"/>
    <VirtualCubeMeasure cubeName="Sales" name="[Measures].[Store Sales]"/>
    <VirtualCubeMeasure cubeName="Sales" name="[Measures].[Unit Sales]"/>
    <VirtualCubeMeasure cubeName="Sales" name="[Measures].[Profit Growth]"/>
    <VirtualCubeMeasure cubeName="Warehouse" name="[Measures].[Store Invoice]"/>
    <VirtualCubeMeasure cubeName="Warehouse" name="[Measures].[Supply Time]"/>
    <VirtualCubeMeasure cubeName="Warehouse" name="[Measures].[Units Ordered]"/>
    <VirtualCubeMeasure cubeName="Warehouse" name="[Measures].[Units Shipped]"/>
    <VirtualCubeMeasure cubeName="Warehouse" name="[Measures].[Warehouse Cost]"/>
    <VirtualCubeMeasure cubeName="Warehouse" name="[Measures].[Warehouse
Profit]"/>
    <VirtualCubeMeasure cubeName="Warehouse" name="[Measures].[Warehouse Sales]"/>
    <VirtualCubeMeasure cubeName="Warehouse" name="[Measures].[Average Warehouse
Sale]"/>
    <CalculatedMember name="Profit Per Unit Shipped" dimension="Measures">
        <Formula>[Measures].[Profit] / [Measures].[Units Shipped]</Formula>
    </CalculatedMember>
</VirtualCube>
```

Testing With Pentaho Analyzer and Report Wizard

You must have an analysis schema with at least one measure and one dimension, and it must be currently open and focused on the Model perfective in Spoon.

This section explains how to use the embedded Analyzer and Report Design Wizard to test a prototype analysis schema.

1. While in the Model perspective, select your visualization method from the drop-down box above the Data pane (it has a **New:** to its left), then click **Go**. The two possible choices are: **Pentaho Analyzer** and **Report Wizard**. You do not need to have license keys for Pentaho Analysis or Pentaho Reporting in order to use these preview tools.
2. Either the Report Design Wizard will launch in a new sub-window, or Pentaho Analyzer will launch in a new tab. Use it as you would in Report Designer or the Pentaho User Console.
3. When you have explored your new schema, return to the Model perspective by clicking **Model** in the upper right corner of the Spoon toolbar, where all of the perspective buttons are. Do not close the tab; this will close the file, and you will have to reopen it in order to adjust your schema.
4. If you continue to refine your schema in the Model perspective, you must click the **Go** button again each time you want to view it in Analyzer or Report Wizard; the Visualize perspective does not automatically update according to the changes you make within the Model perspective.

You now have a preview of what your model will look like in production. Continue to refine it through the Model perspective, and test it through the Visualize perspective, until you meet your initial requirements.

Prototype With Data Integration

Data Integration offers rapid prototyping of analysis schemas through a mix of processes and tools known as **Agile BI**. The Agile BI functions of Pentaho Data Integration are explained in this section, but there is no further instruction here regarding PDI installation, configuration, or use beyond OLAP schema creation. If you need information related to PDI in general, consult the Install PDI and Create DI Solutions articles.

Note: Agile BI is for **prototyping only**. It is extremely useful as an aid in developing OLAP schemas that meet the needs of BI developers, business users, and database administrators. However, **it should not be used for production**. Once your Agile BI schema has been refined, you will have to either hand-edit it in Schema Workbench to optimize it for performance, or completely re-implement the entire model with Schema Workbench.

- Creating a Prototype Schema With a Non-PDI Data Source
- Creating a Prototype Schema With a PDI Data Source
- Prototypes in Production

Creating a Prototype Schema With a Non-PDI Data Source

Your data sources must be configured, running, and available before you can proceed with this step.

Follow the below procedure to create a OLAP schema prototype from an existing database, file, or data warehouse.

Note: If you are already using PDI to create your data source, skip these instructions and refer to Creating a Prototype Schema With a PDI Data Source instead.

1. Start Spoon and connect to your repository, if you are using one.

   ```
   cd ~/pentaho/design-tools/data-integration/ && ./spoon.sh
   ```

2. Go to the **File** menu, then select the **New** sub-menu, then click on **Model**. The interface will switch over to the **Model** perspective.
3. In the **Properties** pane on the right, click **Select**. A data source selection window will appear.
4. Click the round green **+** icon in the upper right corner of the window. The **Database Connection** dialogue will appear.
5. Enter in and select the connection details for your data source, then click **Test** to ensure that everything is correct. Click **OK** when you're done.
6. Select your newly-added data source, then click **OK**. The **Database Explorer** will appear.
7. Traverse the database hierarchy until you get to the table you want to create a model for. Right-click the table, then select **Model** from the context menu. The Database Explorer will close and bring you back to the Model perspective.
8. Drag items from the **Data** pane on the left and drop them into either the **Measures** or **Dimensions** groups in the **Model** pane in the center. The Measures and Dimensions groups will expand to include the items you drag into them.
9. Select each new measure and dimension item, and modify its details accordingly in the **Properties** pane on the right.
10. Save your model through the **File** menu, or publish it to the BA Server using the **Publish** icon above the Model pane.

You now have a basic OLAP schema. You should test it yourself before putting it into production. To do this, continue on to Testing With Pentaho Analyzer and Report Wizard.

Creating a Prototype Schema With a PDI Data Source

1. Start Spoon and connect to your repository, if you are using one.

   ```
   cd ~/pentaho/design-tools/data-integration/ && ./spoon.sh
   ```

2. Open the transformation that produces the data source you want to create a OLAP schema for.
3. Right-click your output step, then select **Model** from the context menu.
4. Drag items from the **Data** pane on the left and drop them into either the **Measures** or **Dimensions** groups in the **Model** pane in the center. The Measures and Dimensions groups will expand to include the items you drag into them.
5. Select each new measure and dimension item, and modify its details accordingly in the **Properties** pane on the right.
6. Save your model through the **File** menu, or publish it to the BA Server using the **Publish** icon above the Model pane.

You now have a basic OLAP schema. You should test it yourself before putting it into production. To do this, continue on to Testing With Pentaho Analyzer and Report Wizard.

pentaho

Prototypes in Production

Once you're ready to test your OLAP schema on a wider scale, use the **Publish** button above the Model pane in the Model perspective, and use it to connect to your test or development BA Server.

You can continue to refine your schema if you like, but it must be republished each time you want to redeploy it.

Note: Agile BI is for **prototyping only**. It is extremely useful for developing OLAP schemas that meet the needs of business analytics developers, business users, and database administrators. However, **it should not be used for production**. Rather, once your Agile BI schema has been refined, you will have to either hand-edit it in Schema Workbench to optimize it for performance, or completely re-implement the entire model with Schema Workbench.

Create and Modify Mondrian Schemas

Now that you have a physical data model in place, you must create a logical model that maps to it. A Mondrian schema is essentially an XML file that performs this mapping, thereby defining a multidimensional database structure.

In a very basic scenario, you will create a Mondrian schema with one cube that consists of a single fact table and a few dimensions, each with a single hierarchy consisting of a handful of levels. More complex schemas may involve multiple virtual cubes, and instead of mapping directly to the single fact table at the center of a star schema, they might map to views or inline tables instead.

All of the Mondrian XML elements are documented in this section in both a single quick reference list and a full individual reference piece for each element. Primarily you will be using Pentaho Schema Workbench to create Mondrian schemas graphically, though you can do advanced schema design through the Data Source Wizard in the User Console or through Schema Workbench later on.

- Schema Workbench Notes
- Add a Data Source in Schema Workbench
- Create a Mondrian Schema
- Adapt Mondrian Schemas to Work with Pentaho Analyzer
- Localization/Internationalization of Analysis Schemas

Schema Workbench Notes

Before you start using Schema Workbench, you should be aware of the following points:

- You start Schema Workbench by executing the `/pentaho/design-tools/schema-workbench/workbench` script. On Linux and OS X, this is a **.sh** file; on Windows it's **.bat**.
- You must be familiar with your physical data model before you use Schema Workbench. If you don't know which are your fact tables and how your dimensions relate to them, you will not be able to make significant progress in developing a Mondrian schema.
- When you make a change to any field in Schema Workbench, the change will not be applied until you click out of that field such that it loses the cursor focus.
- Schema Workbench is designed to accommodate multiple sub-windows. By default they are arranged in a cascading fashion. However, you may find more value in a tiled format, especially if you put the JDBC Explorer window next to your Schema window so that you can see the database structure at a glance. Simply resize and move the sub-windows until they are in agreeable positions.

pentaho

Add a Data Source in Schema Workbench

Your data source must be available, its database driver JAR must be present in the `/pentaho/design-tools/schema-workbench/drivers/` directory, and you should know or be able to obtain the database connection information and user account credentials for it.

Follow the below process to connect to a data source in Schema Workbench.

1. Establish a connection to your data source by going to the **Options** menu and selecting **Connection**. The **Database Connection** dialog appears.
2. Select your database type, then enter in the necessary database connection information, then click **Test**. When you've verified that the connection settings work, click **OK**. The database connection information includes the database name, port number, and user credentials. If you don't know what to type into any of these fields, consult your database administrator or database vendor's documentation.
   Note: The **Require Schema** check box, when selected in the Options menu, puts Schema Workbench into a mode where unpopulated elements appear in the schema.
   Note: **If you are using an Oracle data source**, selecting Require Schema will dramatically improve your Analysis schema load time.
3. If you required a database schema in the previous step, you must now define it by going to the **Options** section of the database dialogue, and creating a parameter called **FILTER_SCHEMA_LIST** with a value of the schema name you want to use.

Your data is now available to Schema Workbench, and you can proceed with creating a Mondrian schema.

Remove Mondrian Data Sources

As you phase out old analysis schemas, you will have to manually remove their data source entries in the Data Source Wizard in the User Console.

1. Login to the User Console with administrator credentials.
2. On the Home page of the User Console, click **Manage Data Sources**. The **Data Source Wizard** appears.
3. Click to highlight the data source to be deleted, and click **Remove**.

The data source is removed and is no longer available for use.

Create a Mondrian Schema

In order to complete this process, you should have already connected to your data source in Schema Workbench.

This section explains the basic procedure for creating a barebones Mondrian schema using Schema Workbench. If you are confused about the definition or application of any of the elements discussed below, refer to the Mondrian Schema Element Quick Reference and the individual reference pieces it links to.

1. To create a new Mondrian schema, click the New button, or go to the **File** menu, then select **New**, then **Schema**. A new schema sub-window will appear. Resize it to fit your preference.
2. It's easier to visualize your physical data model if you have it in front of you. Turn on the JDBC Explorer from the **New** section of the **File** menu and position it according to your preference. If you have a third-party database visualization tool that you are more familiar with, use that instead. The JDBC Explorer is not interactive; it only shows the table structure of your data source so that you can see at a glance what the names of the columns and rows in it.
3. Typically your first action when creating a schema is to add a cube. Right-click the Schema icon in the schema window, then select **Add cube** from the context menu. Alternatively you can click the New Cube button in the toolbar. A new default cube will show up in your schema.
4. Give your cube a name.
5. Add a table by clicking the New Table button, or by right-clicking your cube, then selecting **Add Table**. This will be your fact table. Alternatively you can select **View** or **Inline Table** if these are the data types you need for your fact table.
6. Click the **Table** entry in the **name** field of your new table, and select or type in the name of the table in your physical model that you want to use for this cube's fact table.
7. Add a dimension by right-clicking the cube, then selecting **Add Dimension**, or by clicking the New Dimension button.
8. Type in a friendly name for this dimension in the **name** field.
9. Select a foreign key for this dimension from the **foreignKey** drop-down box, or just type it into the field.
10. When you add a dimension, a new hierarchy is automatically created for it. To configure the hierarchy, expand the dimension by clicking the lever icon on the left side of the dimension's tree entry, then click on **New Hierarchy 0**. Choose a **primaryKey** or **primaryKey Table**.
11. Add a table to the hierarchy by right-clicking the hierarchy, then selecting **Add Table** from the context menu.
12. Choose a column for the **name** attribute.
13. Add a level to the hierarchy by right-clicking the hierarchy, then selecting **Add Level** from the context menu.
14. Give the level a **name** and choose a **column** for it.
15. Add a member property to the level by right-clicking the level, then selecting **Add Property** from the context menu.
16. Give the property a **name** and choose a **column** for it.
17. Add a measure to the cube by right-clicking the cube and selecting **Add Measure** from the context menu.

18. Choose a **column** that you want to provide values for, then select an **aggregator** to determine how the values should be calculated.

These instructions have shown you how to use Schema Workbench's interface to add and configure basic Mondrian schema elements.

When your schema is finished, you should test it with a basic MDX query such as:

```
select {[Dim1].[All Dim1s]} on rows, {[Measures].[Meas1]} on columns from
[CubeName]
```

In order to use your schema as a data source in any Pentaho Business Analytics client tools, you must publish it to the BA Server. To do this, select **Publish** from the **File** menu, then enter in your BA Server connection information and credentials when requested.

- Use MDX Mode and Edit a Schema
- Add Business Groups
- Add Field Descriptions
- Build a Schema and Detecting Errors

Use MDX Mode and Edit a Schema

There are two advanced tools in Schema Workbench that enable you to work with raw MDX and XML. The first is the MDX query editor, which can query your logical data model in real time. To open this view, go to the **File** menu, select **New**, then click **MDX Query**.

The second is XML viewing mode, which you can get to by clicking the rightmost icon (the pencil) in the toolbar. This replaces the name/value fields with the resultant XML for each selected element. To see the entire schema, select the top-level schema entry in the element list on the left of the Schema Workbench interface. Unfortunately you won't be able to edit the XML in this view; if you want to edit it by hand, you'll have to open the schema in an XML-aware text editor.

Add Business Groups

The available fields list in Analyzer organizes fields in folders according to the **AnalyzerBusinessGroup** annotation. To implement business groups, add these annotations to your member definitions appropriately. If no annotation is specified, then the group defaults to "Measures" for measures and the hierarchy name/ caption for attributes.

Below is an example that puts Years, Quarters and Months into a "Time Periods" business group:

```
...
<Level name="Years" ... >
   <Annotations><Annotation name="AnalyzerBusinessGroup">Time
Periods</Annotation></Annotations>
</Level>
<Level name="Quarters" ... >
   <Annotations><Annotation name="AnalyzerBusinessGroup">Time
Periods</Annotation></Annotations>
</Level>
<Level name="Months" ... >
   <Annotations><Annotation name="AnalyzerBusinessGroup">Time
Periods</Annotation></Annotations>
</Level>
...
```

The AnalyzerBusinessGroup annotation is supported on the following schema elements:

- Level
- Measure
- CalculatedMember
- VirtualCubeMeasure

Add Field Descriptions

By adding **description** attributes to your Mondrian schema elements, you can enable tooltip (mouse-over) field descriptions in Analyzer reports.

```
<Level name="Store Country" column="store_country" uniqueMembers="true"
caption="%{foodmart.dimension.store.country.caption}"
            description="%{foodmart.dimension.store.country.description}"/>
```

CAUTION:
Remove the line-wrap or this may not work. These variables will not work unless you localize schemas.

This attribute can be set on the following schema elements:

- Level
- Measure
- CalculatedMember

Build a Schema and Detecting Errors

Analysis schemas are built by publishing them to the BA Server. Each schema is validated to make sure that there are no errors before it is built; if there are any, they'll be shown to you and the schema will fail to publish. If you want to see the errors marked in Schema Workbench before you publish, go to the **Options** menu and select **Require Schema**. When this option is checked, schema validation will happen as new elements are added, and any errors will show as a red **x** next to the offending element.

Adapt Mondrian Schemas to Work with Pentaho Analyzer

The following Mondrian features are not yet functional in Pentaho Analyzer, but are scheduled to be added in the future:

- **Parent-child hierarchies** will render the entire set of members instead of showing them as parents and children.
- **Ragged hierarchies** currently throw an exception when using them in Analyzer
- Additional Mondrian features not yet available: **cube captions**, **drill-through**, **parameterization**

To adjust for these limitations and to enable some Analyzer functions to work properly, you must make the changes explained in the subsections below.

Enable Relative Date Filters in Mondrian

Pentaho Analyzer supports many types of relative date filters, but in order to apply them for a given level, you need to define the format string used to construct MDX members for that level. This is because each data warehouse implementation may have a different date format and set of hierarchy levels.

Common Relative Date Filters

In the Steel Wheels sample data cube provided by Pentaho for evaluation and testing, the Month level uses abbreviated three-letter month names. Furthermore, the Month level sits under the Quarter level. In Steel Wheels, the format string for an MDX member from the Month level would look like this:

```
[yyyy].['QTR'q].[MMM]
```

Some other common date formats:

- **[yyyy]** (Year)
- **[yyyy].[q]** (Quarter)
- **[yyyy].[q].[M]** (Month)
- **[yyyy].[q].[M].[w]** (Week)
- **[yyyy].[q].[M].[w].[yyyy-MM-dd]** (Day)

The **Day** line above also specifies a format to represent the entire date. Without this format, a simple **[d]** parameter would be difficult to put into context. For more information on date format strings, refer to the SimpleDateFormat page on the ICU Project site.

To setup relative date filtering, for each level, you need to do the following:

- In your Mondrian schema file, set the **levelType** XML attribute to TimeYears, TimeMonths, TimeQuarters, TimeWeeks or TimeDate
- Define the MDX date member format as an annotation with the name **AnalyzerDateFormat**.

Here is an example from the Pentaho sample data (Steel Wheels) Time dimension:

```
<Level name="Years" levelType="TimeYears" ... >
   <Annotations><Annotation
name="AnalyzerDateFormat">[yyyy]</Annotation></Annotations>
</Level>
<Level name="Quarters" levelType="TimeQuarters" ... >
   <Annotations><Annotation name="AnalyzerDateFormat">[yyyy].
['QTR'q]</Annotation></Annotations>
```

```
</Level>
<Level name="Months" levelType="TimeMonths" ... >
   <Annotations><Annotation name="AnalyzerDateFormat">[yyyy].['QTR'q].
[MMM]</Annotation></Annotations>
</Level>
```

Other Relative Date Filters

Other types of relative date filters are often used, especially the fiscal year in the business sector. A fiscal year will vary with each business and is based on how that business calculates its annual financial statements. You can define a Fiscal Calendar dimension in your Mondrian schema to accommodate this, and Analyzer will then use the current date to look up fiscal time periods in the fiscal time dimension.

For example, suppose a business has defined their fiscal year to always start on the first of May. Their fiscal time dimension table would look like this:

Date	Fiscal Week	Fiscal Month	Fiscal Quarter	Fiscal Year
2014-04-30	2014-W53	2014-M12	2014-Q4	2014
2014-05-01	2015-W1	2015-M1	2015-Q1	2015
2014-05-02	2015-W1	2015-M1	2015-Q1	2015

Looking at the table and using a date such as **2014-05-01**, we can find which **Fiscal Week**, **Month**, **Quarter**, or **Year** that it belongs to. Just look for the date in the table, then look further up the hierarchy to find **2015-M1**. If you need to get the **Current Month** and **Previous Month**, you can first find **2015-M1** and then look back on the hierarchy to find **2014-M12**, which is a sibling of **2015-M1** in the hierarchy.

There are a few key points to keep in mind about this dimension, before you get started.

- The bottommost level must be a **Date**, which will be used to look up a parent-level member based on the current date.
- The **Date** level must specify a new **AnalzyerFiscalDateFormat** annotation. This annotation value should specifiy a Java format string, which when evaluated with the current date, yields the MDX name of the **Date** level member. This format string should not include the format string for any parents above the **Date** level. This is different from the **AnalyzerDateFormat** annotation in which parents are also included in the format string.
- The **Date** level members must be unique within the level, so **uniquemembers** is set to `true`. This doesn't need to be the same for parent levels, but it is a good practice to do so since this is a time dimension.
- All levels in this hierarchy need to specify the **levelType** attribute.
- Levels above the **Date** level should not specify the **AnalyzerDateFormat** annotations.

Here is an example of a **Fiscal Calendar** dimension defined within a Mondrian schema:

```
<Dimension name="Fiscal Calendar" type="TimeDimension">
    <Hierarchy hasAll="true" primaryKey="DATE_KEY">
        <Table schema="FOODMART" name="CALENDAR"/>
        <Level name="Fiscal Year" levelType="TimeYears" column="FSC_YEAR_STR"
uniqueMembers="true" type="String" ordinalColumn="FSC_YEAR" />
        <Level name="Fiscal Quarter" levelType="TimeQuarters" column="FSC_QUARTER_
YEAR_STR" uniqueMembers="true" type="String" ordinalColumn="FSC_DIM_QUARTER_NUM"
/>
        <Level name="Fiscal Month" levelType="TimeMonths" column="FSC_MONTH_YEAR_
STR" uniqueMembers="true" type="String" ordinalColumn="FSC_DIM_MONTH_NUM" />
        <Level name="Fiscal Week" levelType="TimeWeeks" column="FSC_WEEK_YEAR_
STR" uniqueMembers="false" type="String" ordinalColumn="FSC_DIM_WEEK_NUM" />
        <Level name="Date" levelType="TimeDays" column="CAL_DATE"
uniqueMembers="true" type="Date" ordinalColumn="DATE_KEY" >
            <Annotations><Annotation name="AnalyzerFiscalDateFormat">[yyyy-MM-
dd]</Annotation></Annotations>
        </Level>
    </Hierarchy>
</Dimension>
```

With this set up, Analyzer will be able to generate the MDX to turn a filter like **Current Month** into the correct **Fiscal Month** member:

```
Ancestor([Fiscal Calendar].[Date].[1997-06-28],[Fiscal Calendar].[Fiscal Month])
```

This MDX references a specific date member in the **Date** level, and then uses the **Ancestor** function to locate the parent month. Finding the **Previous Month** would be as simple as using the **Lag** MDX function:

```
Ancestor([Fiscal Calendar].[Date].[1997-06-28],[Fiscal Calendar].[Fiscal Month]).
Lag(1)
```

Once you have these set up, your users will be able to apply this filter by selecting **Choose a commonly used time period** in the **Filter on Fiscal Month** dialog box.

Adjust for Calculated Members and Named Sets

If your Mondrian schema defines calculated members or named sets that reference MDX members without the dimension prefix, then you must set **mondrian.olap.elements.NeedDimensionPrefix** to **false** in your **mondrian.properties** file. Under all other conditions you would want to set this property to **true** because it increases Mondrian performance, as well as the readability of the schema XML file.

Add Geo Map Support to a Mondrian Schema

The Geo Map visualization in Analyzer requires both a Geo Service that provides coordinate data (delivered through a BA Server pentaho-geo plugin), and special Mondrian schema annotations and member properties.

Only the levels marked with Geographical roles (via annotations) can be added to a Geo Map visualization in Analyzer. During the rendering process, the visualization will call the pentaho-geo plugin in the BA Server to look up coordinates that correspond with the level.

Annotations

First, find all levels that describe location data, then add the appropriate annotations as shown and explained below:

```
<Level name="CITY" column="CITY" type="String" uniqueMembers="false">
  <Annotations>
    <Annotation name="Data.Role">Geography</Annotation>
    <Annotation name="Geo.Role">city</Annotation>
  </Annotations>
</Level>
```

Data.Role: Indicates the type of level. Presently, the only valid data role type is **Geography**.

Geo.Role: Specifies the geographical classification (city, state, zip, etc.). While there are built-in types used in the Data Source Wizard and PDI modelers, the values are arbitrary. You could specify a "city" type and, as long as the service provdes data for this classification, it will work.
Note: In the above example, the **city** role retrieves centroid coordinates.

Member properties

In addition to retrieving coordinates from the Geo Service, the **location** Geo.Role value defines a level with member properties supplying **Latitude** and **Longitude** values. Levels that are tagged with **location** must also provide two member properties with the exact names of **latitude** and **longitude** that point to the column in the database which contains these values for the level.

```
<Level name="LatTest" column="CUSTOMERNUMBER" type="Numeric"
uniqueMembers="false">
  <Annotations>
    <Annotation name="Data.Role">Geography</Annotation>
    <Annotation name="Geo.Role">location</Annotation>
  </Annotations>
  <Property name="Latitude" column="CUSTLAT" type="Numeric" />
  <Property name="Longitude" column="CUSTLON" type="Numeric"/>
</Level>
```

Disable Drill-Through Links

You can permanently disable drill-through links for all analysis reports by editing the **analyzer.properties** file, or you can disable drill-through links on a cube-by-cube basis. Both methods are described in these steps.

1. **Global Disable**: Follow these steps to disable drill-through links for all cubes and analysis data sources.
    a. Open your **analyzer.properties** file with a text editor and look for the property called `report.drill.links.disabled`.
    b. Set the value from **false** to **true** as shown here.

    From:

    ```
    report.drill.links.disabled=false
    ```

    To:

    ```
    report.drill.links.disabled=true
    ```

    Drill-through links are now disabled for all cubes and analysis data sources. The option for drill-through links is no longer visible on the Report Options dialog box.

2. **Individual Cubes**: Follow these steps to disable drill-through links on a cube-by-cube basis.
    a. Open the schema for the particular cube on which you want to disable drill-through links.
    b. Find the Cube element in the schema file and add this annotation to disable the links:

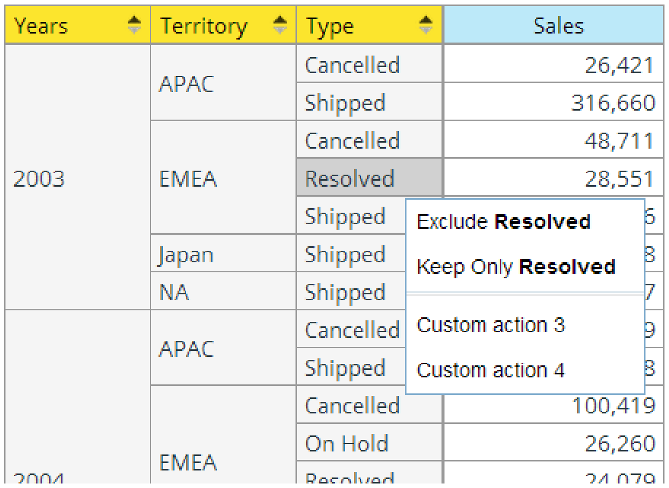
    ```
    <Annotations><Annotation
    name="AnalyzerDisableDrillLinks">true</Annotation></Annotations>
    ```

    c. Save and close the schema.

    Drill-through links are now completely disabled for that cube, and the option to show drill-through links is no longer visible on the Report Options dialog box for the cube.

Configuring Custom Analyzer Actions

Analyzer can be configured with custom action links that call out to JavaScript functions. These action links are available in a context menu by right-clicking on level members or measure cells.



Years	Territory	Type	Sales
2003	APAC	Cancelled	26,421
		Shipped	316,660
	EMEA	Cancelled	48,711
		Resolved	28,551
		Shipped	6
	Japan	Shipped	8
	NA	Shipped	7
2004	APAC	Cancelled	9
		Shipped	8
	EMEA	Cancelled	100,419
		On Hold	26,260
		Resolved	24,079

Define Custom Actions in Mondrian

Action links can be defined in your Mondrian schema as **annotations**. These can be defined under a **Level** or a **Measure**. There is no limit to the number of custom action links that you can define, but they need to be named in ascending order, such as **AnalyzerCustomAction**, **AnalyzerCustomAction2**, **AnalyzerCustomAction3**. The annotation value is just a link-label and JavaScript function, separated by a comma. Analyzer will automatically try to add custom action links on a **Type** level or a **Sales** measure whenever they are used in a report.

Annotation defined on a **Type** level:

```
<Dimension foreignKey="STATUS" name="Order Status">
     <Hierarchy hasAll="true" allMemberName="All Status Types"
primaryKey="STATUS">
        <Level name="Type" column="STATUS" type="String" uniqueMembers="true"
levelType="Regular" hideMemberIf="Never">
                <Annotations>
                    <Annotation name="AnalyzerCustomAction">Custom action
3,customHandlerThree</Annotation>
                    <Annotation name="AnalyzerCustomAction2">Custom action
4,customHandlerFour</Annotation>
```

```
                </Annotations>
        </Level>
    </Hierarchy>
</Dimension>
```

Annotation defined on a **Sales** measure:

```
<Measure name="Sales" column="TOTALPRICE" formatString="#,###" aggregator="sum"
description="Foo">
    <Annotations>
        <Annotation name="AnalyzerBusinessGroup">Measures</Annotation>
        <Annotation name="AnalyzerCustomAction">Custom action
1,customHandlerOne</Annotation>
        <Annotation name="AnalyzerCustomAction2">Custom action
2,customHandlerTwo</Annotation>    </Annotations>
    <CalculatedMemberProperty name="CHART_SERIES_COLOR" value="#0d8ecf" />
</Measure>
```

Implement a Custom Action JavaScript Function

In order to implement the JavaScript function, you need to create a new Pentaho plugin that injects your JavaScript functions into Analyzer. Here is an example of an **analyzer_extension_plugin.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin title="analyzer-extension">
   <static-paths>
        <static-path url="/analyzer-extension/resources" localFolder="resources"/>
   </static-paths>
   <external-resources>
         <file context="analyzer">content/analyzer-extension/resources/analyzer_
extension_plugin.js</file>
   </external-resources>
</plugin>
```

This basically tells the BA Server to inject the **analyzer_extension_plugin.js** file into Analyzer so that those functions are now available to Analyzer to call when a user clicks on a custom action link.

Here is an example **analyzer_extension_plugin.js**.

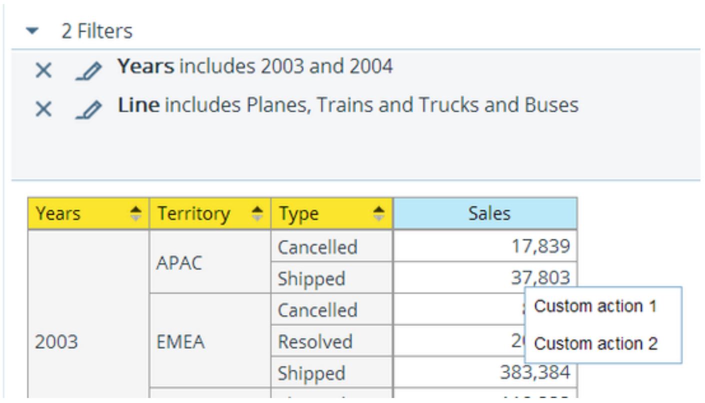
```
cv.extension = cv.extension || {};

/**
```

```
 * report - Analyzer report definition.
 * formula - The level or measure that was clicked on.
 * ctx - All levels that intersect on the clicked on level or cell.
 * filter - Filters applied on the report. Only includes filters which includes
members.
 */

cv.extension.customHandlerOne = function (report, formula, ctx, filter) {
    var year = ctx['[Time].[Years]']; // Returns the member unique name
    if (year)
        year = cv.util.parseMDXExpression(year); // Extract the name of the member
    var url = window.CONTEXT_PATH + "api/
repos/:public:Steel%20Wheels:Country%20Performance%20(heat%20grid).xanalyzer/
viewer?yearParameter=" + year;
    if (window.parent && window.parent.parent && window.parent.parent.mantle_
openTab) {
        window.parent.parent.mantle_openTab("Custom One", "Custom One", url);
    }
    window.open(url);
}

cv.extension.customHandlerOne_validate = function (report, formula, ctx, filter) {
    var territory = ctx['[Markets].[Territory]'];
    if (territory == "[Markets].[Japan]")
        return false;
    return true;
}
```

You must define your custom action JavaScript function under the **cv.extension** namespace. The name of the JavaScript function must exactly match the name you used in the **AnalyzerCustomAction** annotation. The function takes four parameters:

- **report**- This is the Analyzer report object. You normally will not use this, but if you want to access the report XML definition to inspect the state of the current report definition, you can access **report.reportDoc**.
- **formula**- This is either the level MDX unique name or the measure unique name, depending on what the user clicked on.
- **ctx**- This is a map of all the levels on the row or column zone and their corresponding MDX members. When clicking on a cell, this map will contain all row and column levels on the report. When clicking on a level member, this map will only contain outer levels which are usually to the left or above the clicked-on level.
- **filter**- This is a level map-to-filter operator-to-member array of all report filters with the exception of numeric filters like **Top10** or **Greater than**.

As an example, assuming the user clicks on this cell:


2 Filters
Years includes 2003 and 2004
Line includes Planes, Trains and Trucks and Buses
Custom action 1
Custom action 2

Years	Territory	Type	Sales
2003	APAC	Cancelled	17,839
		Shipped	37,803
	EMEA	Cancelled	
		Resolved	2
		Shipped	383,384

Then the **member**, **ctx**, and **filter** arguments will look like:

```
ctx: Object
      [Markets].[Territory]: "[Markets].[APAC]"
      [Measures].[MeasuresLevels]: "[Measures].[Sales]"
      [Order Status].[Type]: "[Order Status].[Shipped]"
      [Time].[Years]: "[Time].[2003]"
      __proto__: Object
filter: Object
     [Product].[Line]: Array[3]
       0: "[Product].[Trucks and Buses]"
       1: "[Product].[Trains]"
       2: "[Product].[Planes]"
       length: 3
       __proto__: Array[0]
     [Time].[Years]: Array[2]
     __proto__: Object
formula: "[Measures].[Sales]"
```

Here are a couple of helpful tips for implementing the JavaScript functions:

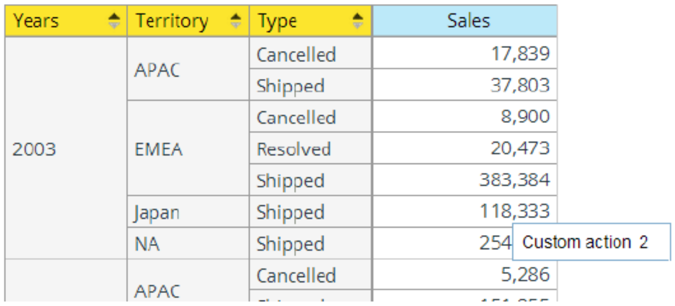
- You can use **cv.util.parseMDXExpression** to extract the name of the member. For example, **[Year].[2003]** would return: 2003.
- You can construct your own URL and then open the URL in a new PUC tab, assuming Analyzer is running within PUC with the function: **window.parent.parent.mantle_openTab**.

Determine When to Show a Custom Action

There is also another feature to validate whether a custom action link should be included in the context menu or not. You can implement a validation function which returns false to hide the link in the UI. If this validation function is not implemented, then the link will always be shown. This validation function must be named by suffixing the custom action JavaScript function name with **_validate**.

In this example, the **Custom action 1** menu item will not be included if the user right-clicks on a **Measure** cell where the current context includes **Territory: Japan**:

```
cv.extension.customHandlerOne_validate = function (report, formula, ctx, filter) {
var territory = ctx['[Markets].[Territory]'];
if (territory == "[Markets].[Japan]")
return false;
return true;
}
```



Years	Territory	Type	Sales
2003	APAC	Cancelled	17,839
		Shipped	37,803
	EMEA	Cancelled	8,900
		Resolved	20,473
		Shipped	383,384
	Japan	Shipped	118,333
	NA	Shipped	254
	APAC	Cancelled	5,286

Notice how **Custom action 1** was not included in the above menu.

Localization/Internationalization of Analysis Schemas

You can create internationalized message bundles for your analysis schemas and deploy them with the Pentaho Web application. This enables Pentaho Analyzer to access localized schemas.

1. Edit your analysis schema and tokenize all values that you want to localize. Typically you would create variables for all caption and description values.

```
<Schema measuresCaption="%{foodmart.measures.caption}">
    <Dimension name="Store" caption="%{foodmart.dimension.store.caption}"
description="%{foodmart.dimension.store.description}">
        <Hierarchy hasAll="true" allMemberName="All Stores"
allMemberCaption="%{foodmart.dimension.store.allmember.caption =All Stores}"
primaryKey="store_id"  caption="%{foodmart.hierarchy.store.country.caption}"
description="%{foodmart.hierararchy.store.country.description}>
            <Table name="store"/>
            <Level name="Store Country" column="store_country"
uniqueMembers="true" caption="%{foodmart.dimension.store.country.caption}"
description="%{foodmart.dimension.store.country.description}"/>
```

2. Create localized **MondrianMessages.properties** files in the `/WEB-INF/classes/com/pentaho/messages/` directory inside of the Pentaho WAR, and define each token you used in the analysis schema.

NOTE:

JBoss users will have to delete the unpacked Pentaho WAR directory if it exists, then unpack the pentaho.war file with an archive utility, create the message bundles in the proper location, then repack it into a WAR again.

If you need further assistance in creating localized message bundles, refer to BA Server and Thin Client Message Bundles.

```
foodmart.measures.caption=Measures
foodmart.dimension.store.country.caption=Store Country
foodmart.dimension.store.name.property_type.column=store_type
foodmart.dimension.store.country.member.caption=store_country
foodmart.dimension.store.name.property_type.caption=Store Type
foodmart.dimension.store.name.caption=Store Name
foodmart.dimension.store.state.caption=Store State
foodmart.dimension.store.name.property_manager.caption=Store Manager
foodmart.dimension.store.name.property_storesqft.caption=Store Sq. Ft.
foodmart.dimension.store.allmember.caption=All Stores
foodmart.dimension.store.caption=Store
foodmart.cube.sales.caption=Sales
foodmart.dimension.store.city.caption=Store City
foodmart.cube.sales.measure.unitsales=Unit Sales
```

3. Edit the **mondrian.properties** file in the `/pentaho/server/biserver-ee/pentaho-solutions/system/mondrian/` directory and add this line (or modify it if it's already there):

```
mondrian.rolap.localePropFile=com.pentaho.messages.MondrianMessages
```

4. Save and close the file.
5. Restart the BA Server.
6. Login to the User Console with administration permissions, then click **Manage Data Sources**, then the **Add** button. Choose **Analysis** from the menu. Browse to import your file.
7. Edit your Analysis data source by checking the option next to **Manually enter data source parameters**.
   h. A. If absent from the list of parameters, add one parameter called **DataSource** whose value is the name of the JDBC data source to use.
   i. B. Create a new parameter called **Locale** and enter the value for the language that you want to make available.
   j. C. Create a new parameter called **DynamicSchemaProcessor** whose value is `mondrian.i18n.LocalizingDynamicSchemaProcessor`.
   k. D. Create a new parameter called **UseContentChecksum** whose value is `true`.
8. In the User Console, go to **Tools** > **Refresh** > **Mondrian Schema Cache**.

Your analysis schemas will now be localized to whatever language is currently selected in the Pentaho User Console, if a message bundle for that locale was copied to the proper directory as explained above.

- BA Server and Thin Client Message Bundles
- Pentaho Analyzer Localization
- Set a Default Font for PDF Exports

BA Server and Thin Client Message Bundles

Note: Stop the Pentaho application server before editing anything inside of the Pentaho WAR file.

You can localize the Pentaho User Console, Pentaho Analyzer, and Dashboard Designer by creating locale- and language-specific message bundles within the Pentaho Web application. Message bundles are dynamically adjusted according to browser locale, so you can create localized message bundles for every language you want to support, and let each individual user's system language settings determine which one is loaded.

For brevity's sake, only the default Pentaho User Console files will be explained in detail. The file naming convention is identical among all message bundles in Pentaho Business Analytics. The following files are located in the `/mantle/messages/` directory:

- **mantleMessages.properties**: The default message bundle for the Pentaho User Console. In its initial condition, it is a copy of **mantleMessages_en.properties**. If you want to change the default language and dialect, copy your preferred message bundle file over this one.
- **mantleMessages_en.properties**: The English-language version of the standard message bundle. This is an identical copy of **mantleMessages.properties**.
- **mantleMessages_fr.properties**: The French-language version of the standard message bundle.
- **mantleMessages_de.properties**: The German-language version of the standard message bundle.
- **mantleMessages_supported_languages.properties**: Contains a list of localized message bundles and the native language names they correspond to; this relieves the BA Server of the burden of having to discover them on its own. A supported_languages.properties file should be created for every message bundle that you intend to localize.

Example of mantleMessages_supported_languages.properties

```
en=English
de=Deutsch
fr=Français
```

New files are created in the following format: **mantleMessages_xx_YY.properties** where **xx** represents a lowercase two-letter language code, and **YY** represents a two-letter locale code, where applicable. So, for instance, U.S. and British English could have two separate message bundles if you wanted to draw a distinction between the two dialects:

- mantleMessages_en_US.properties
- mantleMessages_en_GB.properties

The language and country codes must be in standard ISO format. You can look up both sets of codes on these pages:

- http://en.wikipedia.org/wiki/List_of_ISO_639-1_codes
- http://www.iso.org/iso/country_codes/iso_3166_code_lists.htm

You can edit the default message bundle directly if you need to make surgical changes to the content inside of it. If you plan to translate it into another language, it makes more sense to copy the file and change the name appropriately, then translate it line by line. Be sure to update supported_languages.properties with the new country and dialect code and the native language name that it corresponds to.
Note: All message bundles must be UTF-8 encoded.

- Pentaho User Console and Dashboard Designer Message Bundles
- Pentaho Analyzer Localization
- Pentaho Interactive Reporting Localization
- Geographic Service (pentaho-geo) Localization
- Report Viewer Localization

Pentaho Analyzer Localization

Follow the directions below to create localized message bundles for Pentaho Analyzer.

Note: This covers only the Analyzer interface, not the OLAP data sources you use with Analyzer. For schema localization, refer to Localization/Internationalization of Analysis Schemas.

1. If the BA Server is currently running, shut it down.
2. Make a copy of the **messages.properties** file in `/pentaho-solutions/system/analyzer/resources/`; name the copy according to the standard locale naming scheme defined earlier in this section.

   ```
   cp messages.properties messages_fr.properties
   ```

3. Translate the content of the new message bundle into the locale defined in its file name.
4. Edit the **messages_supported_languages.properties** file in `/pentaho-solutions/system/analyzer/resources/` and add the new locale.

   ```
   fr=Francais
   ```

You now have a translated Analyzer message bundle available in your BA Server.

- Localization/Internationalization of Analysis Schemas

Set a Default Font for PDF Exports

Before following this procedure, stop the BA Server and the User Console.

When displaying data in Analyzer, your reports will use the default browser fonts. However, the PDF export function may not have the same fonts available to it when creating a PDF from your Analyzer report, resulting in output that doesn't look the same way in PDF format as it does in the browser. The default font for PDFs is Helvetica, but you can specify any TrueType font or collection to replace it. Follow the instructions below to specify a different font for PDF exports.

Note: If you have localized your schema in a language that uses a multi-byte character set (most Asian languages fit into this category), this process is required to make PDF output appear without errors.

1. Edit the **analyzer.properties** file in the `/pentaho/server/biserver-ee/pentaho-solutions/system/analyzer/` directory.
2. Uncomment the **renderer.pdf.font.path** line.

   ```
   renderer.pdf.font.path=C:/WINDOWS/Fonts/MSGOTHIC.TTC,1
   ```

3. Replace the value of this line with the TrueType font or collection that you want to use as the default. If you are specifying a collection, you must put a **,1** after the font name, as shown in the above example. This does not apply to individual fonts (TTF files).

   ```
   renderer.pdf.font.path=/usr/share/fonts/truetype/freefont/FreeSans.ttf
   ```

4. Save and close the file, and start the BA Server.

Your PDF exports from Analyzer should have the font you specified.

Analysis Schema Security

You can restrict portions of your OLAP schema from being viewed by certain user roles defined within your schema definition, and then map those role restrictions to the BA Server so that when you publish the schema, its security restrictions are obeyed in Analyzer and Dashboard Designer (or any other BA Server-based dashboard). If you do this, only the permissible parts of the schema will be available to the specified roles. Pentaho offers several unique paths to accomplish role-based security, all of which are explained later in this section.

Refer to the subsections below that apply to your situation.
Note: Changes to a OLAP schema should be done with Schema Workbench. If you change a schema, you must republish it to the BA Server in order for the modifications to take effect. Changes to XML configuration files (such as those that contain core BA Server and Mondrian engine settings or properties) can be done with any text editor.

- Restrict Access to Specific Members
- Mondrian Role Mapping in the BA Server

Restrict Access to Specific Members

You can restrict access to parts of a schema by implementing the **<HierarchyGrant>** element to define access to a hierarchy:

```
<Role name="California manager">
    <SchemaGrant access="none">
        <CubeGrant cube="Sales" access="all">
            <HierarchyGrant hierarchy="[Store]" access="custom" topLevel="[Store].
[Store Country]">
                <MemberGrant member="[Store].[USA].[CA]" access="all"/>
                <MemberGrant member="[Store].[USA].[CA].[Los Angeles]"
access="none"/>
            </HierarchyGrant>
            <HierarchyGrant hierarchy="[Customers]" access="custom"
topLevel="[Customers].[State Province]" bottomLevel="[Customers].[City]">
                <MemberGrant member="[Customers].[USA].[CA]" access="all"/>
                <MemberGrant member="[Customers].[USA].[CA].[Los Angeles]"
access="none"/>
            </HierarchyGrant>
            <HierarchyGrant hierarchy="[Gender]" access="none"/>
        </CubeGrant>
    </SchemaGrant>
</Role>
```

The **access** attribute can be **"all"**, meaning all members are visible; **"none"**, meaning the hierarchy's very existence is hidden from the user; and **"custom"**. With custom access, you can use the **topLevel** attribute to define the highest visible level (preventing users from seeing too much of the 'big picture,' such as viewing revenues rolled up to the Store or Country level); or use the **bottomLevel** attribute to define the lowest visible level (preventing users from looking at an individual customer's details); or control which sets of members a user can see by defining nested **<MemberGrant>** elements.

You can only define a **<MemberGrant>** element if its enclosing **<HierarchyGrant>** has **access="custom"**. Member grants give (or remove) access to a given member and all of its children. Here are the rules:

1. **Members inherit access from their parents.** If you deny access to California, you won't be able to see San Francisco.

2. **Grants are order-dependent.** If you grant access to USA, then deny access to Oregon, then you won't be able to see Oregon or Portland. But if you were to deny access to Oregon, then grant access to USA, you can effectively see everything.
3. **A member is visible if any of its children are visible.** Suppose you deny access to USA, then grant access to California. You will be able to see USA, and California, but none of the other states. The totals against USA will still reflect all states, however.
4. **Member grants don't override the hierarchy grant's top- and bottom-levels.** If you set topLevel="[Store].[Store State]", and grant access to California, you won't be able to see USA.

Mondrian Role Mapping in the BA Server

The role mapper connects user role restrictions defined in a Mondrian schema to user roles defined in the Pentaho BA Server. This enables BI developers to set schema access controls in a single place, rather than in many places across different parts of Business Analytics. If you do not configure the role mapper, then none of the roles defined in your schema will restrict access in the BA Server.

The role mapper is configured through the BA Server in the `/pentaho-solutions/system/pentahoObjects.spring.xml` file. There are three mapper implementations available, each with disabled example configurations in pentahoObjects.spring.xml. All three are explained in the below subsections.

- The Mondrian One-To-One UserRoleMapper
- The Mondrian SampleLookupMap UserRoleMapper
- The Mondrian SampleUserSession UserRoleMapper

The Mondrian One-To-One UserRoleMapper

The **Mondrian-One-To-One-UserRoleMapper** maps each role name in the BA Server to roles defined in the OLAP schema. Therefore, the mapper assumes that the roles defined in your OLAP schema are mirrored in the BA Server. For example, if you have a role called "CTO" in your schema, and a role called "CTO" in the BA Server, this role mapper would be appropriate.

```
<bean id="Mondrian-UserRoleMapper"
name="Mondrian-One-To-One-UserRoleMapper"
class="org.pentaho.platform.plugin.action.mondrian.mapper.
MondrianOneToOneUserRoleListMapper"
scope="singleton" />
```

The Mondrian SampleLookupMap UserRoleMapper

This mapper provides a translation table (in the form of a **<map>** element) to associate BA Server roles with OLAP schema roles. The lookups take the form of key/value pairs where the key is the BA Server role, and the value is the OLAP schema role. In the example below, the "ceo" role in the BA Server maps to the "California manager" role in the schema.

```
<bean id="Mondrian-UserRoleMapper"
        name="Mondrian-SampleLookupMap-UserRoleMapper"
        class="org.pentaho.platform.plugin.action.mondrian.mapper.
        MondrianLookupMapUserRoleListMapper"
        scope="singleton">
    <property name="lookupMap">
        <map>
            <entry key="ceo" value="California manager" />
            <entry key="cto" value="M_CTO" />
            <entry key="dev" value="M_DEV" />
        </map>
    </property>
</bean>
```

The Mondrian SampleUserSession UserRoleMapper

This mapper retrieves OLAP schema roles from a named HTTP session variable. In the below example, the session is stored in a variable called **MondrianUserRoles**.

```
<bean id="Mondrian-UserRoleMapper"
    name="Mondrian-SampleUserSession-UserRoleMapper"
    class="org.pentaho.platform.plugin.action.mondrian.mapper.
    MondrianUserSessionUserRoleListMapper"
    scope="singleton">
        <property name="sessionProperty" value="MondrianUserRoles" />
</bean>
```

Restrict OLAP Schemas Per User

To enable user-level access restrictions for each published analysis schema in the Pentaho User Console, follow the instructions below.

Note: There is a different process for restricting OLAP schemas per role. See Mondrian Role Mapping in the BA Server for instructions on role mapping.

1. Log into the Pentaho User Console as the administrator user.
2. Create new a solution directory for the user you want to provide a private analysis schema for.
3. Change the access permissions on the new user directory so that only the specified user has access to it.
4. Copy the target schema to the private directory that you just created.
5. Rename the copied schema such that it reflects the user account that now owns it.
6. Refresh the solution repository.
7. Add data source entries for each personalized schema copy you created, using the Data Source Wizard to edit the data source.

   ```
   <Catalog name="Suzy">
       <DataSourceInfo>Provider=mondrian;DataSource=SampleData</DataSourceInfo>
       <Definition>solution:suzy/suzy.mondrian.xml</Definition>
   </Catalog>
   <Catalog name="Tiffany">
       <DataSourceInfo>Provider=mondrian;DataSource=SampleData</DataSourceInfo>
       <Definition>solution:tiffany/tiffany.mondrian.xml</Definition>
   </Catalog>
   ```

8. Edit the `/pentaho/server/biserver-ee/pentaho-solutions/system/pentaho.xml` file and add a **,xml** to the end of the **acl-files** list.

   ```
   <acl-files>xaction,url,prpt,xdash,xcdf,xanalyzer,xanalyzer,xml</acl-files>
   ```

9. Restart the BA Server.

The schema copies you created are now only available to the users you specified.

Configure Analysis Options

The Pentaho Analysis (Mondrian) engine is configurable through a properties file. Mondrian options allow for enhanced engine and data source performance and functionality under certain conditions. Pentaho Analyzer is also configurable, but more in terms of adjusting the functionality of the Analyzer client tool itself.

- Manage Analysis Data Sources
- Configure the Mondrian Engine (mondrian.properties)
- Change Analyzer Default Behavior
- Assign Analyzer Chart Colors
- Customizing Analyzer Sort Options

Manage Analysis Data Sources

In order to use a properly prepared data source with Pentaho Analysis, it must be established in either the User Console as a Native (JDBC) data source, or in your Web application server as a JNDI data source. Whichever method you choose, you will have to know the name of the data source later on when you're asked to supply it in Pentaho Schema Workbench.

Schema Workbench can only work with one data source for each Mondrian schema; this is a structural limitation in the software that will be remedied at a later time. You can have multiple JNDI or Native (JDBC) data sources established, but only one at a time can be used with Schema Workbench.

Configure the Mondrian Engine (mondrian.properties)

Purpose: mondrian.properties is a configuration file referenced by the Pentaho Analysis (Mondrian) engine to determine performance and usability settings.

Location: `/pentaho/server/biserver-ee/pentaho-solutions/system/mondrian/mondrian.properties`

Note: This section is specific to maximizing Mondrian performance with Pentaho Analyzer. For all available settings, see http://mondrian.pentaho.com/documentation/configuration.php.

Performance

One of the key performance goals for Analyzer is to push as many operations to the database as possible. This is particularly important when querying many high-cardinality dimensions because you want Mondrian to only process the combinations that make sense and not a huge cartesian product which is sparsely populated. Pay particular attention to the following engine properties:

```
mondrian.expCache.enable=true
```

Improves the performance of summary numbers such as ranks and running sums.

```
mondrian.native.crossjoin.enable=true
```

This is an essential property which enabes pushdown to the database so that Mondrian does not perform large cross joins in memory.

```
mondrian.native.filter.enable=true
```

Particularly important for pushdown of Not In and Excludes filters.

```
mondrian.native.nonempty.enable=true
```

This should be set to false to optimize Mondrian SQL generation given the MDX patterns generated by Analyzer. Analyzer uses non-empty cross joins to push down joins between high cardinality attributes to the database. If this property is set to false, there are many non-empty cross joins which would get evaluated by Mondrian instead because Mondrian pushes down the individual arguments such as **dimension.members**.

```
mondrian.olap.maxConstraints=1000
```

Used in conjunction with the **mondrian.native.ExpandNonNative** property. This should be set to as large as the underlying DB can support.

```
mondrian.native.ExpandNonNative=true
```

Allows pushdown of even more crossjoins by materializing the crossjoin inputs into IN lists. For example, a crossjoin whose inputs are MDX functions can first have their inputs evaluated to members and then convert the crossjoin into native evaluation.

```
mondrian.olap.elements.NeedDimensionPrefix=true
```

Analyzer always generates dimension qualified members so no need to spend time searching for members on different dimensions.

Usability

```
mondrian.result.limit=5000000
```

In the event that pushdown cannot occur, this is the largest crossjoin size that Mondrian should try to process. Anything that exceeds this will likely send the CPU for a toss and result in long server hangs. If this limit is hit, Analyzer will present the user a nice warning suggesting options to simplify the report.

```
mondrian.olap.case.sensitive=true
```

This is important for "Equals" filters because the UI will preserve the casing of filter values. If the user defines a filter on "John Doe", then the filter should only applies to "John Doe" and not "john doe"

```
mondrian.olap.ignoreInvalidMembers=true
```

This is important for saved reports because the user may build a report with a filter on 10 sales rep and after the next ETL, one of them is gone. The report should continue to run and return just the 9 remaining sales reps.

```
mondrian.olap.ignoreInvalidMembersDuringQuery=true
```

See mondrian.olap.ignoreInvalidMembers

```
mondrian.olap.iterationLimit=5000000
```

Similar to mondrian.result.limit except for controlling limits on aggregate evaluation.

```
mondrian.olap.compareSiblingsByOrderKey=true
```

This is required for sorting members in a dimension A->Z or Z->A. This property fixes a bug in Mondrian but was added for backward compatibility.

```
mondrian.olap.NullDenominatorProducesNull=true
```

The best way to understand this property is via an example. Suppose you want to see quota attainment for your sales reps which is computed as Booked Deals / Quota. Some reps may not have quotas and so quota attainment is either infinity or null. By treating divide by null as infinity, Mondrian will return these sales reps in the report. Otherwise, if divide by null evaluates to null, then Mondrian will filter those reps out (due to the NON EMPTY).

Change Analyzer Default Behavior

Giving Analyzer reports a configuration that fits your needs involves editing the analyzer properties file. Here are some examples of settings that you can change.

- Enabling your logo to appear in PDF output
- Specifying the maximum number of rows when drilling-down in a report
- Defining a specific value for blank cells
- Changing the default chart options

The complete list of properties that you can change are in the file comments, and you can find the file here: `/pentaho/server/biserver-ee/pentaho-solutions/system/analyzer/analyzer.properties`. Note: Changes take effect after you restart the BA Server.

Assign Analyzer Chart Colors

Sometimes visualizations are more clear if you assign specific chart colors to data objects. For instance, you might want to assign colors to sales volume in different regions: red for the Northern region and blue for the Southern region. You can define colors like this using a member property in the Mondrian schema or using a text-based file inside the resources folder.
The color property works differently in various situations.

- **If a series contains one level and one measure,** then the color depends on the level member.
- **If a series contains multiple levels and one measure,** then the color depends on the level member that is farthest to the right.
- **If a series contains zero or more levels, or multiple measures,** then the color depends on the measure member. If there are multiple measures, each measure has a unique color.

If a member has a color mapping defined in both the text-based .json file and in the Mondrian schema using the CHART_SERIES_COLOR property, the JSON file takes precedence over the CHART_SERIES_COLOR property.

Assigning Colors Using a Text-based File

You can create member-to-color mappings in a text-based .json file if you do not want to define colors in the database or do not want to change the data structure, . The file is located in `system/common-ui/resources/chartseriescolor/`. You must edit the `mdx.json` file to define color information for OLAP models such as Mondrian. Analyzer then uses this information to define colors. Define the color information using a web API such as Json Parser Online to ensure you are using the correct JSON syntax.

When running a report, color mapping runs for all members in the report. If the ,json file has an invalid syntax, an error posts in the log file and no color information applies.

1. Using a JSON editor, open `biserver-ee\penatho-solutions\system\common-ui\resources\chartseriescolor\mdx.json`
2. Edit the sample so that the elements and colors are defined appropriately. The format is Hierarchy Level > Member > Color. Specify color using hexadecimal or decimal values. In this example, the colors are set for the regions, which are Central, Eastern, Southern, and Western. The colors are then set for the measures, which are Actual, Budget, and Variance.

```
"SampleData" : {
    "[Region].[Region]": {
        "[Region].[Central]": "#0000cc",
        "[Region].[Eastern]": "#0d8ecf",
        "[Region].[Southern]": "#b0de09",
        "[Region].[Western]": "#fcd202"
    },
     "[Measures].[MeasuresLevel]": {
         "[Measures].[Actual]": "#0000cc",
         "[Measures].[Budget]": "#0d8ecf",
         "[Measures].[Variance]": "#b0de09"
      }
    }
}
```

3. Save and close the .json file.

Assigning Colors Using a Member Property

You can change the colors of the members in an Analyzer report by defining a member property in the Mondrian schema. There are two different methods for assigning colors to levels or assigning colors to measures. The database column that stores color information is TERRITORY_COLOR. This column is mapped to the CHART_SERIES_COLOR.

Assigning Colors to Levels

1. Find the level for which you want to specify colors. In this example, the level name is `Territory`, coming from a column named TERRITORY.
2. Define the member property name as CHART_SERIES_COLOR, the mapping to the database column as TERRITORY_COLOR, and the data type for the color values as Integer.

```
<Level name="Territory" column="TERRITORY" type="String"
uniqueMembers="true"
levelType="Regular" hideMemberIf="Never"
<Property name="CHART_SERIES_COLOR" column="TERRITORY_COLOR"
type=Integer></Property>
```

Assigning Colors to Measures

1. Find the measure for which you want to specify colors.
2. Define the CalculatedMemberProperty name as `CHART_SERIES_COLOR`.
3. Assign the color value using either a hexidecimal or decimal value.

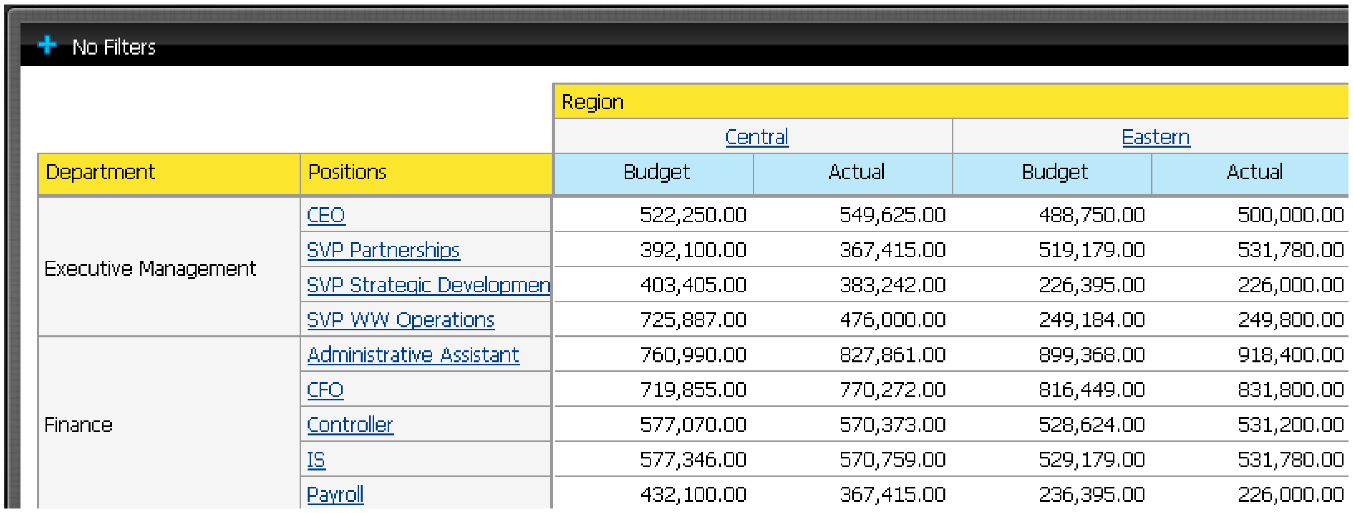
```
Measure name="Quantity" column="QUANTITYORDERED" formatString="#,###"
aggregator="sum"
<CalculatedMemberProperty name="CHART_SERIES_COLOR" value="13369344"/>
</Measure>
```

Define Hyperlinks

Presenting too much information in one report can overwhelm readers with distracting details, causing them to miss information that is important to them. You can manage the amount of information displayed in a report by hyperlinking from one report to other related reports, charts, dashboards, and URLs. For example, you can present basic information in an easy-to-comprehend report with hyperlinks to reports that contain details.

For charts, hyperlinks take precedence over the drill-down chart feature. For example, when readers click a bar in a chart, it displays data related to the hyperlink you define, not the drill-down chart.

For reports, you can define a hyperlink on any row label or column header. When you define a hyperlink, the link is applied to all members within the row or column. In this source report, hyperlinks have been defined for the Positions row label and the Region column header. Notice how each of the row and column members have a blue underlined hyperlink.

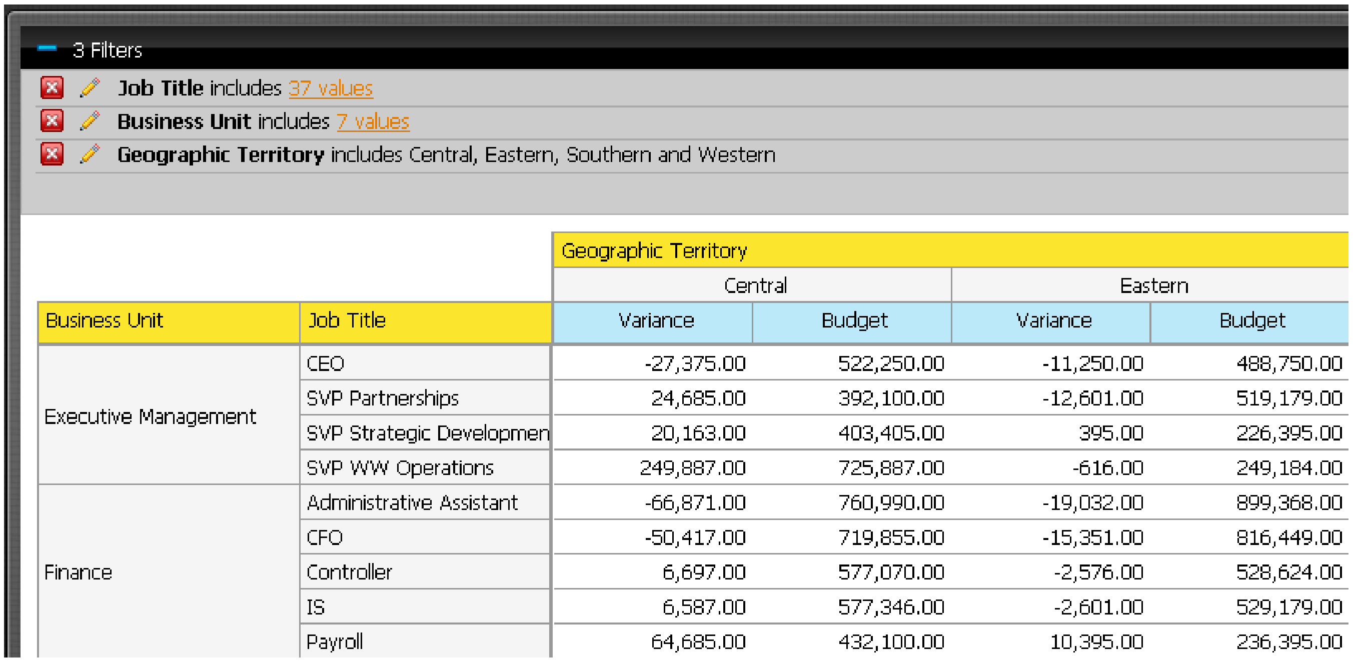


No Filters

		Region			
		Central		Eastern	
Department	Positions	Budget	Actual	Budget	Actual
Executive Management	CEO	522,250.00	549,625.00	488,750.00	500,000.00
	SVP Partnerships	392,100.00	367,415.00	519,179.00	531,780.00
	SVP Strategic Developmen	403,405.00	383,242.00	226,395.00	226,000.00
	SVP WW Operations	725,887.00	476,000.00	249,184.00	249,800.00
Finance	Administrative Assistant	760,990.00	827,861.00	899,368.00	918,400.00
	CFO	719,855.00	770,272.00	816,449.00	831,800.00
	Controller	577,070.00	570,373.00	528,624.00	531,200.00
	IS	577,346.00	570,759.00	529,179.00	531,780.00
	Payroll	432,100.00	367,415.00	236,395.00	226,000.00

When defining hyperlinks to a destination report that has parameters, you can map row labels and column headers in the source report to parameters in the destination report. This enables you to constrain the hyperlink result to display *only* data for the mapped parameters. If you do not restrain the results, all of the data appears and no filter applies.

For example, you can create a hyperlink in the source report for all the members in the **Position** row, and constrain the displayed data to only that related to each position and its department. To do this, you map the **Department** and **Position** row labels in the source report to the **Business Unit** and **Job Title** parameters in this destination report.

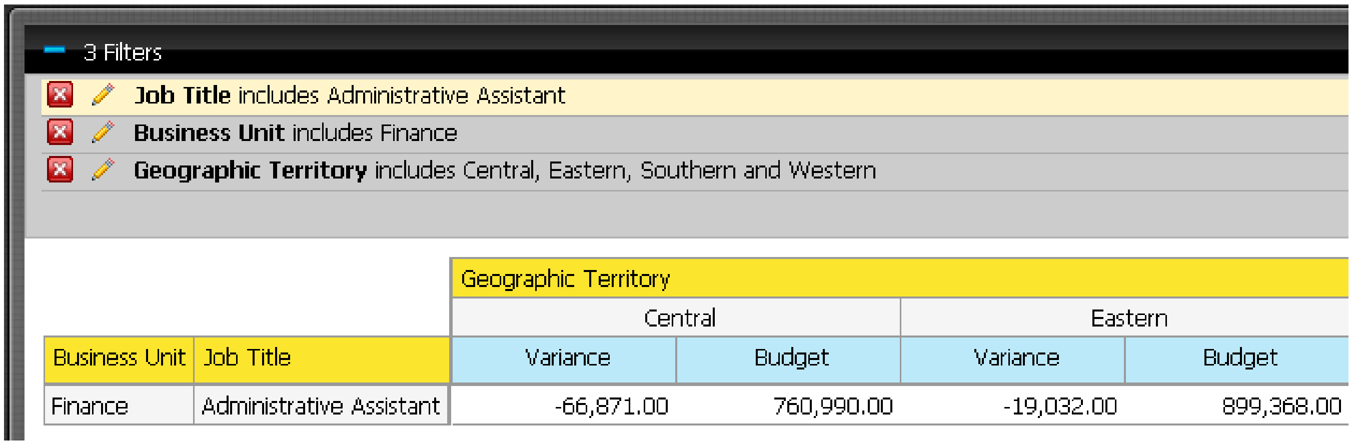


3 Filters

- **Job Title** includes 37 values
- **Business Unit** includes 7 values
- **Geographic Territory** includes Central, Eastern, Southern and Western

		Geographic Territory			
		Central		Eastern	
Business Unit	Job Title	Variance	Budget	Variance	Budget
Executive Management	CEO	-27,375.00	522,250.00	-11,250.00	488,750.00
	SVP Partnerships	24,685.00	392,100.00	-12,601.00	519,179.00
	SVP Strategic Developmen	20,163.00	403,405.00	395.00	226,395.00
	SVP WW Operations	249,887.00	725,887.00	-616.00	249,184.00
Finance	Administrative Assistant	-66,871.00	760,990.00	-19,032.00	899,368.00
	CFO	-50,417.00	719,855.00	-15,351.00	816,449.00
	Controller	6,697.00	577,070.00	-2,576.00	528,624.00
	IS	6,587.00	577,346.00	-2,601.00	529,179.00
	Payroll	64,685.00	432,100.00	10,395.00	236,395.00

This is the result when the reader clicks on the Administrative Assistant position within the Finance Department in the source report.



3 Filters

- **Job Title** includes Administrative Assistant
- **Business Unit** includes Finance
- **Geographic Territory** includes Central, Eastern, Southern and Western

		Geographic Territory			
		Central		Eastern	
Business Unit	Job Title	Variance	Budget	Variance	Budget
Finance	Administrative Assistant	-66,871.00	760,990.00	-19,032.00	899,368.00

Each parameter added to the mapping constrains the data further. You can map any row labels that appear to the left, and column headers that appear above the member data.
Note: If you did not constrain the data with parameters, readers would see data for all Administrative Assistant Positions in all Departments.

Create Hyperlinks to a Report in the BA Repository

1. Create an Analyzer report or open an existing one.
2. Right-click a row label or column header and select **Hyperlink**. The **Link on** dialog box appears.

3. Click **Enable Link** to activate the hyperlink feature. You can disable linking by clearing the **Enable Link** check box.
4. In the **Link To** drop-down menu, select **Pentaho Repository File**.
5. Click **Browse** to locate a report, chart, or dashboard in the BA repository and click **Open**.
    a. If the destination report has parameters, they automatically appear in the **Destination Parameter** list on the left. Map parameters to related row labels or column headers by selecting the check box for each parameter you want to use to constrain the resulting data. Enter the related names of the row labels or column headers within curly brackets.
    b. If the destination report does not have parameters, the Destination Parameter list does not appear. Go to the next step.
6. Specify how hyperlink content displays by clicking the **Open in:** drop-down menu and selecting **New Tab**, **New Window**, or **Current Window**.
7. Enter a **Tool Tip** to be displayed when you hover over hyperlinks and click **OK.** Hyperlinks appear in the Analyzer report.
8. Click the links to ensure the content associated with them appears correctly and save the report.

Hyperlinking to a URL

1. Create an Analyzer report or open an existing one.
2. Right-click a row label or column header and select **Hyperlink**. The **Link on** dialog box appears.
3. Click **Enable Link** to activate the hyperlink feature. You can disable linking by clearing the **Enable Link** check box.
4. In the **Link To** drop-down menu, choose **URL** from the dropdown menu.
5. In the **URL** field, enter the full web address you want the hyperlink to launch. For example, `http://www.yahoo.com`.
6. Choose how the URL displays by clicking on the **Open in:**drop-down menu and selecting **New Tab**, **New Window**, or **Current Window**.
7. Enter a **Tool Tip** to be displayed when you hover over hyperlinks and click **OK.** The new hyperlinks appear in the Analyzer report.
8. Click the links to ensure the website associated with them appears correctly and save the report.

Number Formatting

Analyzer can format number fields in a few different ways:

- **Default**: Uses the cell formatter specified in the Mondrian schema
- **General Number**: A number that can be separated by commas and decimal places, but has no sign. Pound symbols represent number places if they exist, and zeroes represent potential holders for places that do not exist. So for a definition of **00#,###.00**, the number 11223.4 would be represented in Analyzer as 011,223.40.
- **Percentage**: A number that can be separated by commas and decimal places, followed by a percent sign. Pound symbols represent number places if they exist, and zeroes represent potential holders for places that do not exist. So for a definition of **00#,###.00%**, the number 11223.4 would be represented in Analyzer as 011,223.40%.
- **Currency**: A number that can be separated by commas and decimal places, preceded by a currency sign. Pound symbols represent number places if they exist, and zeroes represent potential holders for places that do not exist. So for a definition of **$00#,###.00**, the number 11223.4 would be represented in Analyzer as $011,223.40.
- **Expression**: A custom format defined by some user-supplied logic. See Advanced Conditional Formatting With MDX Expressions for more information.

To change the formatting of a measure, right-click on it in the grid, then select **Column Name and Format...**

from the context menu.
Note: Formatting options apply to both the report viewer and exports (PDF, XLS).

Conditional Formatting

Conditional formatting in the Analyzer data grid means that cells will be physically affected by the data they contain. The most common form of conditional formatting is stoplight reporting, where cell backgrounds are colored red, green, or yellow depending on user-defined thresholds. Analyzer offers some simple pre-defined methods of conditionally altering visual cues for numeric data in a variety of formats, and the ability to use a custom MDX expression to implement fine-grained conditions.

- Simple Conditional Formatting of Measures
- Advanced Conditional Formatting With MDX Expressions

Simple Conditional Formatting of Measures

Conditional formatting in the Analyzer data grid means that cells will be physically affected by the data they contain. The most common form of conditional formatting is stoplight reporting, where cell backgrounds are colored red, green, or yellow depending on user-defined thresholds. Analyzer offers some simple pre-defined methods of conditionally formatting numeric data. Follow the directions below to implement conditional cell formatting.

1. Right-click a measure in the grid, then select **Conditional Formatting** from the context menu. A sub-menu with conditional formatting types will appear.
2. Select your preferred number format from the list.

The analyzer report will refresh and apply the formatting choice you specified.

- Conditional Formatting Types

Conditional Formatting Types

Indicator Type	Description
Color scale	The background cell color will be shaded according to the value of the cell relative to the highest and lowest recorded values in that measure. There are several color progressions to choose from.
Data bar	The cell background is partially filled with a solid color proportional to the scale of the cell's value relative to the highest and lowest recorded values in that measure.
Trend arrow	An upward or downward arrow is displayed to the right of the cell value depending on whether it contains a positive or negative value.

Advanced Conditional Formatting With MDX Expressions

If the premade conditional formatting options in Analyzer are not precise enough for your needs, you can apply custom formatting to a measure by using an MDX expression, as explained below.

1. Right-click a measure in the grid, then select **Column Name and Format...** from the context menu. The **Edit Column** dialogue will appear.
2. Select **Expression** from the **Format** drop-down list. A default MDX expression that prints green or red arrows in cells if their values are greater than or less than zero, respectively, will appear in the **MDX Format Expression** field:

```
Case
When [Measures].CurrentMember > 0
Then '|#,##0|arrow=up'
When [Measures].CurrentMember < 0
Then '|#,##0|arrow=down'
Else '|#,##0'
End
```

3. Modify the expression to suit your needs. Consult Conditional Formatting Expressions for more information on conditional formatting syntax and options.
   Note: If the MDX expression is invalid, an **invalid report definition** error will appear at the top of the dialogue.
4. Click **OK** to commit the change.

The analyzer report will refresh and apply the formatting choices you specified.

- **Conditional Formatting Expressions**

Conditional Formatting Expressions

Valid MDX format strings contain properties that apply special rendering for the HTML pivot tables. A format string follows this syntax:
| # , # # # | style = red

The leading | (pipe) tells Analyzer that this format string contains properties. The **#,###** is the measure's value format, which in this example is one number separated from three further places or decimal places by a comma (for instance: 3,667). **style=red** is a key/value pair; all possible keys and values are explained in the table below.
The following properties are supported by Analyzer:

Indicator Type	Description	Values
style	Changes the cell background color.	red, yellow, green
arrow	Shows a trend arrow that points up or down. A value of **none** will not render the arrow; this is useful in situations in which you only want to show one directional arrow instead of two.	up, down, none
link	Creates an HTML link which will open in a new window when clicked in Analyzer.	Any browser-renderable URL Must be a fully-qualified URL URL must be enclosed in quotes
image	Renders a custom image that you specify. This image file must be stored in the `/pentaho-solutions/system/analyzer/resource/image/report/` directory.	An image file name, with extension

Customizing Analyzer Sort Options

Available Fields for Sorting in Analyzer

The **Available fields** list in Analyzer can be sorted using the **View** toggle with the following sorting options:

UI Label	Description	Sort Option
By Category	Sorts by folder names.	cmdViewCategory
Measure - Level - Time	Sorts by the type of field.	cmdViewType
A > Z	Sorts by field names without any folders.	cmdViewName
Schema	Sorts in the same order thant is defined in the Mondrian schema file.	cmdViewSchema

The default sort used in an Analyzer report is based on the following priority:

1. A sort specified in an URL takes the highest priority.
2. The last sort option is automatically remembered when a report is reopened.
3. Next comes the annotation value specified on the report's cube in the Mondrian schema file.
4. A system wide setting in analyzer.properties file.
5. Last, the default value of **cmdViewCategory**.

Specify the Sort Option with an URL

The sort option can be set in the URL by adding a "fieldListView" query parameter at the end of the URL.

For example, you can do something like this in a codeblock in HTML or XML.

```
http://localhost:8080/pentaho/api/re...ue&debug=true&fieldListView=cmdViewName
```

Specify Sort Option with Annotation

To specify the sort option using an annotation, add a Cube-level annotation called "**AnalyzerFieldListView**" in your Mondrian schema file. This annotation must be the first child element under a cube as shown here.

```
<Cube name="Quadrant Analysis">
   <Annotations>
          <Annotation name="AnlalyzerFieldListView">cmdViewName</Annotation>
   </Annotations>
   <Table name="Quadrant_Acuals" />
   <DimensionUsage name="Region" source="Region" />
   <DimensionUsage name="Department" source="Department" />
   <DimensionUsage name="Positions" source="Positions" />
```

Specify the Sort Option through the analyzer.properties file

This sort option is specified by setting the cmdViewType for the "report.field.list.view" property.

```
# Default field list view mode used to sort the available field
# list in the editor.  Possible values include: cmdViewCategory,
# cmdViewType, cmdViewSchema and cmdViewName
# This can also be overriden on a cube level with the annotation
# AnalyzerFieldListView
report.field.list.view=cmdViewType
```

Mondrian Cache Control

This section contains instructions for configuring and controlling the cache infrastructure that the Pentaho Analysis engine uses for OLAP data. This information is useful for properly updating your OLAP cubes when your data warehouse is refreshed, and for performance-tuning.

Restriction: Most of the advanced cache features explained in this section are for Enterprise Edition deployments only. Within that, most of the Enterprise Edition features of the Analysis engine are only beneficial to large, multi-node OLAP deployments that are performing poorly.

The Analysis engine does not ship with a segment cache, but it does have the ability to use third-party cache systems. If you've installed Pentaho Analysis Enterprise Edition, then you have a default configuration for the JBoss **Infinispan** distributed cache, though the actual Infinispan software is not included and must be downloaded separately. Infinispan supports a wide variety of sub-configurations and can be adapted to cache in memory, to the disk, to a relational database, or (the default setting) to a distributed cache cluster.

The Infinispan distributed cache is a highly scalable solution that distributes cached data across a self-managed cluster of Mondrian instances. Every Mondrian instance running the Analysis Enterprise Edition plugin on a local network will automatically discover each other using UDP multicast. An arbitrary number of segment data copies are stored across all available nodes. The total size of the cache will be the sum of all of the nodes' capacities, divided by the number of copies to maintain. This is all fully configurable; options are explained later in this section.

Other supported segment cache configurations include, but are not limited to:

- **Memcached**, which uses an established (extant) Memcached infrastructure to cache and share the segment data among Mondrian peers.
- **Pentaho Platform Delegating Cache**, which relies on the Pentaho BA Server to delegate segment data storage to the BA Server's native caching capabilities, thus leveraging the existing caching configuration. Some people may prefer this configuration because it keeps the BA Server and Analysis engine manageable as a single entity.

Note: The Pentaho Platform Delegating Cache is not yet feature-complete. It will be available in a future Business Analytics release, but it is not yet ready for production use.

- Segment Cache Architecture
- Cache Configuration Files
- Modify the JGroups Configuration
- Switch to Another Cache Framework

Segment Cache Architecture

Restriction: The segment cache features explained in this section are for very large OLAP deployments, and require a Pentaho Analysis Enterprise Edition license.

How the Analysis Engine Uses Memory

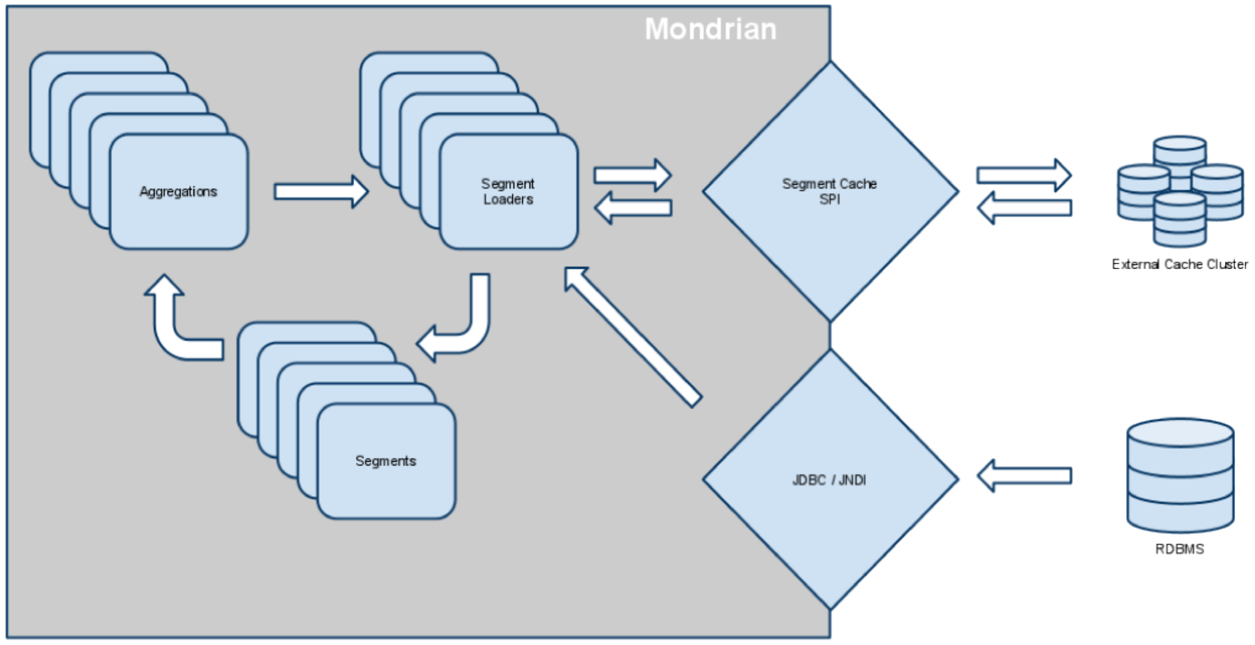
Each Mondrian segment cache node, regardless of which configuration it uses, loads the segments required to answer a given query into system memory. This cache space is called the **query cache**, and it is composed of hard Java references to the segment objects. Each individual node must have enough memory space available to answer any given query. This might seem like a big limitation, but Mondrian uses deeply optimized data structures which usually take no more than a few megabytes, even for queries returning thousands of rows.

Once the query finishes, Mondrian will usually try to keep the data locally, using a weak reference to the segment data object. A weak reference is a special type of Java object reference which doesn't force the JVM to keep this object in memory. As the Mondrian node keeps answering queries, the JVM might decide to free up that space for something more important, like answering a particularly big query. This cache is referred to as the **local cache**.

The local cache can be switched on or off by editing the Pentaho Analysis EE configuration file and modifying the value (set it to **true** or **false**) of the **DISABLE_LOCAL_SEGMENT_CACHE** property. Setting this property will not affect the query cache.

This is the order in which Mondrian will try to obtain data for a required segment once a query is received:

1. The node will parse the query and figure out which segments it must load to answer that particular query
2. It checks into the local cache, if enabled.
3. If the data could not be loaded from the local cache, it checks into the external segment cache, provided by the Pentaho Analysis plugin, and it places a copy inside the query cache.
4. If the data is not available from the external cache, it loads the data form SQL and places it into the query cache.
5. If the data was loaded form SQL, it places a copy in the query cache and it sends it to the external cache to be immediately shared with the other Mondrian nodes.
6. The node can now answer the query.
7. Once the query is answered, Mondrian will release the data from the query cache.
8. If the local cache is enabled, a weak reference to the data is kept there.



Cache Control and Propagation

All cache control operations are performed through Mondrian's CacheControl API, which is documented in the Mondrian project documentation at http://mondrian.pentaho.com. The CacheControl API allows you to modify the contents of the cache of a particular node. It controls both the data cache and the OLAP schema member cache.

When flushing a segment region on a node, that node will propagate the change to the external cache by using the SegmentCache SPI. If the nodes are not using the local cache space, then the next node to pick up a query requiring that segment data will likely fetch it again through SQL. Once the data is loaded from SQL, it will again be stored in the external segment cache.

You should not use the local cache space when you are using the external cache. For this reason, it is disabled by default in Pentaho Analysis Enterprise Edition.

Using the local cache space on a node can improve performance with increased data locality, but it also means that all the nodes have to be notified of that change. Mondrian nodes don't propagate the cache control operations among the members of a cluster. If you deploy a cluster of Mondrian nodes and don't propagate the change manually across all of them, then some nodes will answer queries with stale data.

Cache Configuration Files

The following files contain configuration settings for Pentaho Analysis cache frameworks. All of them are in the same directory inside of the deployed pentaho.war: `/WEB-INF/classes/` .

- **pentaho-analysis-config.xml** Defines the global behavior of the Pentaho Analysis Enterprise Edition plugin. Settings in this file enable you to define which segment cache configuration to use, and to turn off the segment cache altogether.
- **infinispan-config.xml** The InfinispanSegmentCache settings file. It configures the Infinispan system.
- **jgroups-udp.xml** Configures the cluster backing the Infinispan cache. It defines how the nodes find each other and how they communicate. By default, Pentaho uses UDP and multicast discovery, which enables you to run many instances on a single machine or many instances on many machines. (There are examples of other communication setups included in the JAR archive.) This file is referenced by infinispan as specified in the infinispan-config.xml configuration file.
- **memcached-config.xml** Configures the Memcached-based segment cache. It is not used by default. To enable it, modify **SEGMENT_CACHE_IMPL** in pentaho-analysis-config.xml.

Modify the JGroups Configuration

Restriction: The segment cache features explained in this section are for very large OLAP deployments, and require a Pentaho Analysis Enterprise Edition license.

The default Infinispan configuration uses JGroups to distribute the cache across all Mondrian instances it finds on the local network. If you want to modify how those communications are done, you must edit the JGroups configuration file.

Note: Fine-grained JGroups configuration is covered in the JGroups documentation; you should read through it before making changes.

Each node might require a different configuration, so although the default configuration is highly portable, it might not work for you.

If you are deploying this plugin on Amazon EC2, JGroups has a special configuration file that you copied to your `/WEB-INF/classes/` directory when you installed the Analysis Enterprise Edition package. Additionally, default JGroups configuration files are inside of the JAR archive. To switch implementations, edit **infinispan-config.xml** and make the modification appropriate to your communication method:

Comm. type	Config entry
UDP communication	`<property name="configurationFile" value="jgroups-udp.xml"/>`
TCP communication	`<property name="configurationFile" value="jgroups-tcp.xml"/>`
Amazon EC2	`<property name="configurationFile" value="jgroups-ec2.xml"/>`

Switch to Another Cache Framework

Restriction: The segment cache features explained in this section are for very large OLAP deployments, and require a Pentaho Analysis Enterprise Edition license.

Pentaho Analysis Enterprise Edition ships with configuration files that assume a JBoss Infinispan deployment. Instructions are provided below for switching to the Pentaho Platform Delegating Cache or Memcached. However, **Pentaho strongly recommends Infinispan over Memcached for maximum OLAP performance.**

Also in this section is a brief overview of how to create a Java class to implement your own custom cache system.

- Switch to Memcached
- Switch to Pentaho Platform Delegating Cache
- Use a Custom SegmentCache SPI

Switch to Memcached

In order to complete this procedure, you must have your own pre-configured Memcached instance. You should have also installed the Analysis Enterprise Edition package to your BA Server or standalone Mondrian engine.

If you already use the Memcached cache framework in your organization and would like to hook it up to the Pentaho Analysis OLAP engine, follow the directions below to switch from the default Infinispan cache framework configuration.

CAUTION:
Pentaho and Mondrian developers recommend against using Memcached. You are almost certain to have better performance with Infinispan.

1. If the BA Server or standalone Mondrian engine are running, shut them down now.
2. If you performed a default install of the Pentaho Analysis Enterprise Edition package, then you should have all of the required JARs installed to the BA or Mondrian server. If you aren't sure, verify now that the following JARs are present in the `/WEB-INF/lib/` directory inside of your deployed pentaho.war or Mondrian engine:
   - pentaho-analysis-ee
   - commons-lang
   - commons-io
   - commons-codec
   - pentaho-ee-dsc-core
   - memcached

9. Edit the **pentaho-analysis-config.xml** in the `/WEB-INF/classes/` directory inside the deployed pentaho.war or Mondrian engine, and change the value of **SEGMENT_CACHE_IMPL** to match the class name referenced below:

```
<entry key="SEGMENT_CACHE_IMPL">com.pentaho.analysis.segmentcache.impl.
memcached.MemcachedSegmentCache</entry>
```

10. Edit the **memcached-config.xml** in the `/WEB-INF/classes/` directory inside the deployed pentaho.war or Mondrian engine, and change the values of **SALT**, **SERVERS**, and **WEIGHT** to match your preference:

```
                    <entry key="SALT">YOUR SECRET SALT VALUE HERE</entry>
                    <entry key="SERVERS">192.168.0.1:1642,192.168.0.
2:1642</entry>
                    <entry key="WEIGHTS">1,1</entry>
```

Your Pentaho Analysis Enterprise Edition instance is now configured to use Memcached for OLAP segment caching.

- Memcached Configuration Options

Memcached Configuration Options

These properties control Memcached settings, and are set in the **memcached-config.xml** file in the `/WEB-INF/classes/` directory inside of your deployed pentaho.war or Mondrian engine.

Note: This is not a comprehensive list of the potential Memcached settings; the options explained below are the ones most critical to Memcached configuration for Pentaho Analysis.

Property	Purpose
SERVERS	A comma-separated list of servers and port numbers representing the Memcached nodes usable by the plugin.
WEIGHTS	A comma-separated list of numbers representing the relative caching capacity of the servers defined in the SERVERS property. There must be exactly as many values of WEIGHTS as there are values of SERVERS. As an example, if the first server has a capacity of 128 megabytes, and the second has a capacity of 256 megabytes, the correct values for the WEIGHTS property should be "1,2", indicating that the first server has a relative size of half of the second one.
SALT	A secret key prefix to be used when saving and loading segment data from the Memcached nodes. This property must be the same for all Mondrian nodes that share their caches. If the SALT value is different from one node to the next, the nodes will not be able to share their cache data.

Switch to Pentaho Platform Delegating Cache

In order to complete this procedure, you must have installed the Analysis Enterprise Edition package to your BA Server.

If you would like to share the BA Server solution cache with the Pentaho Analysis segment cache, follow the directions below.

This cache system is still experimental and not fully implemented; it is not recommended for production use. Therefore, no public documentation is available at this time.

Use a Custom SegmentCache SPI

If you want to develop your own implementation of the SegmentCache SPI, you'll have to follow this basic plan:

1. Create a Java class that implements **mondrian.spi.SegmentCache**
2. Compile your class and make it available in Mondrian’s classpath
3. Edit **mondrian.properties** and set **mondrian.olap.SegmentCache** to your class name
4. Start the BA Server or Mondrian engine

This is only a high-level overview. If you need more specific advice, contact your Pentaho support representative and inquire about developer assistance.

Visualize Your Data

Pentaho provides an end-user client tool that is designed specifically for traversing an analysis schema: Pentaho Analyzer, a highly interactive and easy to use analysis slicer-dicer.

Once you have Pentaho Analyzer content, you can use Dashboard Designer to create a dashboard that includes your Analyzer report plus any other content in a variety of configurations and styles.

It is also possible to use your analysis schema as a data source in Pentaho Report Designer, though you will have to create your own MDX query to refine the data for reporting.

- Introduction to the Multidimensional Expression Language (MDX)
- Create Analyzer Reports
- Use Analysis Cubes in Report Designer

Introduction to the Multidimensional Expression Language (MDX)

The Multidimensional Expression Language (MDX) is an OLAP database query and calculation language similar to SQL. It is the method by which a dataset is retrieved from a OLAP database. While you do not necessarily have to know MDX in order to use Pentaho's client tools, you should be somewhat familiar with it for the purpose testing your Mondrian schema.

If you are already familiar with SQL, then much of the MDX syntax will look familiar, and the rest should be relatively easy to learn. You can edit the MDX using Analyzer or a text editor.

There are six MDX data types:

1. Dimension or hierarchy
2. Level
3. Member
4. Tuple
5. Scalar
6. Set

Below is an example of a very simple MDX query:

```
SELECT
   { [Measures].[Salesfact] } ON COLUMNS,
   { [Date].[2004], [Date].[2005] } ON ROWS
FROM Sales
```

- Further Reading

Further Reading

MDX was initially developed by Microsoft for its SQL Server analysis products, though it has since become an independent standard. It's been around long enough now that there are many MDX tutorials and references, most notably:

- http://msdn.microsoft.com/en-us/library/ms145506.aspx
- http://en.wikipedia.org/wiki/Multidimensional_Expressions

Note: MDX implementations vary, and many MDX documentation resources are specific to certain niche products or standards. Not all MDX functions and extensions are supported in Pentaho Analysis.

Create Analyzer Reports

Pentaho offers one primary slicer-dicer tool: **Pentaho Analyzer**.

Pentaho Analyzer is a powerful and extremely easy-to-use visualization tool that is delivered as a plugin to the Pentaho User Console. This is the preferred client tool for generating analysis content.

These sections explain how to use **Analyzer** to perform basic tasks.

- Create a New Analyzer Report
- Analyzer Visualizations

Use Analysis Cubes in Report Designer

Pentaho Report Designer is typically used with raw data sources, or with databases that have a metadata layer through a tool like Pentaho Metadata Editor. However, you can also use an MDX query to retrieve a data set from a multidimensional database. The sections below explain how to set up a Pentaho Analysis data source and add an MDX query in Pentaho Report Designer.
Note: For more information on Report Designer, consult the section Design Print-Quality Reports in the Pentaho InfoCenter, or through the **Documentation** link in the **Help** menu in Report Designer.

- Add an OLAP Data Source
- Create an MDX Query

Add an OLAP Data Source

You must have a report file open in order to proceed, and your data source must be accessible before you can connect to it in Report Designer. You may need to obtain database connection information from your system administrator, such as the URL, port number, JDBC connection string, database type, and user credentials. Follow this procedure to add a Pentaho Analysis (Mondrian) data source in Report Designer.

1. Select the **Data** tab in the upper right pane. By default, Report Designer starts in the **Structure** tab, which shares a pane with **Data**.
2. Click the yellow cylinder icon in the upper left part of the Data pane, or right-click **Data Sets**. A drop-down menu with a list of supported data source types will appear.
3. Select **OLAP** from the drop-down menu, then select one of the following: **Pentaho Analysis**, **Pentaho Analysis (Denormalized)**, or **Pentaho Analysis (Legacy)**. The **Mondrian Datasource Editor** window will appear.
4. If you want to provide parameters that contain different Mondrian connection authentication credentials, click the **Edit Security** button in the upper left corner of the window, then type in the fields or variables that contain the user credentials you want to store as a parameter with this connection. The role, username, and password will be available as a security parameter when you are creating your report.
5. Click **Browse**, navigate to your Mondrian schema XML file, then click **Open**.
6. Above the **Connections** pane on the left, click the round green **+** icon to add a new data source. If you installed the Pentaho sample data, several **SampleData** entries will appear in the list. You must have HSQLDB to view the sample data.
7. In the subsequent **Database Connection** dialogue, type in a concise but reasonably descriptive name for this connection in the **Connection Name** field; select your database brand from the **Connection Type** list; select the access type in the **Access** list at the bottom; then type in your database connection details into the fields in the **Settings** section on the right. The Access list will change according to the connection type you select; the settings section will change depending on which item in the access list you choose.
8. Click the **Test** button to ensure that the connection settings are correct. If they are not, the ensuing error message should give you some clues as to which settings need to be changed. If the test dialogue says that the connection to the database is OK, then click the **OK** button to complete the data source configuration.

Now that your data source is configured, you must enter an MDX query before you can finish adding the data source. You can also create a dynamic query through scripts; see Dynamic Query Scripting for more information.

Create an MDX Query

You must be in the Data Source Configuration window to follow this process. You should also have configured a JNDI data source connection.

Follow this process to add a Mondrian cube definition (multidimensional expression) as a data source.

1. At the bottom of the Data Source Configuration window, click **Use Mondrian Cube Definition (MDX)**. The **Browse** button will become active.
2. Click **Browse**, navigate to the location of your cube definition file, click to select it, then click **Open**.
3. Type an MDX query into the **Query** pane on the right.
4. Click **OK**.

Your data source is now properly configured and refined according to the Mondrian definition you specified.

Log Output

The BA Server and the standalone Mondrian engine both use **log4j** to record information about generated OLAP-generated SQL queries. It may be useful to examine this log output if you are having trouble with configuration or performance-tuning.

- Analysis SQL Output Logging
- Enabling Segment Cache Logging
- View Log Output in Analyzer

Analysis SQL Output Logging

Follow the directions below to turn on logging for SQL generation in the BA Server or Mondrian engine. If you are aware of other log4j classes that will assist you in configuration troubleshooting or performance-tuning, you can add them during this process.

1. Stop the BA Server or standalone Mondrian engine.
2. Edit the **log4j.xml** file in the `/WEB-INF/classes/` directory inside of the deployed pentaho.war or Mondrian engine.
   Note: In some Mondrian implementations, this file is **log4j.properties**. Substitute this filename accordingly if you do not see a log4j.xml file.
3. Add the following line anywhere inside of the **</log4j:configuration>** node (but not inside of other constituent node tags):

   ```
   log4j.logger.mondrian.sql=DEBUG
   ```

4. Save and close the file.
5. Start the BA Server or Mondrian engine.

A significant amount of Analysis SQL query debugging information will now be available in your application server's logs. Some data about the segment cache plugin will also appear, but there are some other steps to take beyond this if you want log data specific to the segment cache. Refer to Enabling Segment Cache Logging for more information.

Enabling Segment Cache Logging

When deployed inside or alongside of the Pentaho BA Server, you can view Analysis engine segment cache logs by opening the log window from within Analyzer.

If you are using Analyzer with a standalone Mondrian engine, cache log information is available through log4j. Edit your log4j configuration as explained in Analysis SQL Output Logging, and set the value of **com.pentaho.analysis.segmentcache** to DEBUG. Here are some other log4j categories that can help you diagnose configuration and performance problems:

Class name	Cache system
com.pentaho.analysis.segmentcache. impl.infinispan	Outputs information related to the Infinispan implementation of the segment cache.
com.pentaho.analysis.segmentcache. impl.memcached	Outputs information related to the Memcached implementation of the segment cache.
com.pentaho.analysis.segmentcache. impl.pentaho	Outputs information related to the Pentaho BI Platform Delegating Cache implementation.

View Log Output in Analyzer

Create a new Pentaho Analyzer report and drag a measure or a dimension in the report. Once your report is displaying numbers, click on the top link, identified as "Log". If there is no such link available within Pentaho Analyzer, it is because you have not used an administrator account to login to Pentaho User Console. You must log out and log back in using an administrator account credentials.

If your plugin was successfully configured, you should see log information relating to the segment cache. If there was an error in your configuration, all exception messages will be displayed here as well.

Troubleshooting

This section contains known problems and solutions relating to the procedures earlier.

- Jobs Scheduled on DI Server Cannot Execute Transformation on Remote Carte Server
- Sqoop Import into Hive Fails

Multi-Byte Characters Don't Appear In PDFs Exported From Analyzer

If you are using a multi-byte character set, such as would be used for languages like Japanese or Chinese, and find that you have missing or corrupted output when exporting Analyzer reports to PDF, you will have to specify a default TrueType font for PDF rendering that supports multi-byte characters. The default PDF font in Analyzer is Helvetica, which does not support multi-byte character sets. To change this setting, see Set a Default Font for PDF Exports.

Mondrian Schema Element Quick Reference

All of the possible Mondrian schema elements are defined in brief below, listed in the hierarchy in which they are used. To see a more detailed definition and a list of possible attributes and content, click on an element name.

Element	Definition
<Schema>	A complete Mondrian schema; a collection of cubes, virtual cubes, shared dimensions, and roles.
<Cube>	A collection of dimensions and measures, all centered on a fact table.
<VirtualCube>	A cube defined by combining the dimensions and measures of one or more cubes. A measure originating from another cube can be a <CalculatedMember>.
<CubeUsages>	Base cubes that are imported into a virtual cube.
<CubeUsage>	Usage of a base cube by a virtual cube.
<VirtualCubeDimension>	Usage of a dimension by a virtual cube.
<VirtualCubeMeasure>	Usage of a measure by a virtual cube.
<Dimension>	Defines a dimension -- a collection of hierarchies.
<DimensionUsage>	Usage of a shared dimension by a cube.
<Hierarchy>	Specifies a predefined drill-down.
<Level>	A level of a hierarchy.
<KeyExpression>	SQL expression used as key of the level, in lieu of a column.
<NameExpression>	SQL expression used to compute the name of a member, in lieu of Level.nameColumn.
<CaptionExpression>	SQL expression used to compute the caption of a member, in lieu of Level.captionColumn.
<OrdinalExpression>	SQL expression used to sort members of a level, in lieu of Level.ordinalColumn.
<ParentExpression>	SQL expression used to compute a measure, in lieu of Level.parentColumn.

<Property>	A member property. The definition is contained in a hierarchy or level, but the property will be available to all members.
<PropertyExpression>	SQL expression used to compute the value of a property, in lieu of Property.column.
<Measure>	Specifies an aggregated numeric value.
<CalculatedMember>	A member whose value is derived using a formula, defined as part of a cube.
<NamedSet>	A set whose value is derived using a formula, defined as part of a cube.
<Table>	A fact or dimension table.
<View>	Defines a table by using an SQL query, which can have different variants for different underlying databases.
<Join>	Defines a table by joining a set of queries.
<InlineTable>	Defines a table using an inline dataset.
<Closure>	Maps a parent-child hierarchy onto a closure table.
<AggExclude>	Exclude a candidate aggregate table by name or pattern matching.
<AggName>	Declares an aggregate table to be matched by name.
<AggPattern>	Declares a set of aggregate tables by regular expression pattern.
<AggFactCount>	Specifies name of the column in the candidate aggregate table which contains the number of fact table rows.
<AggIgnoreColumn>	Tells Mondrian to ignore a column in an aggregate table.
<AggForeignKey>	Maps a foreign key in the fact table to a foreign key column in the candidate aggregate table.
<AggMeasure>	Maps a measure to a column in the candidate aggregate table.
<AggLevel>	Maps a level to a column in the candidate aggregate table.
<Role>	An access-control profile.
<SchemaGrant>	A set of rights to a schema.
<CubeGrant>	A set of rights to a cube.
<HierarchyGrant>	A set of rights to both a hierarchy and levels within that hierarchy.
<MemberGrant>	A set of rights to a member and its children.
<Union>	Definition of a set of rights as the union of a set of roles.

<RoleUsage>	A reference to a role.
<UserDefinedFunction>	Imports a user-defined function.
<Parameter>	Part of the definition of a hierarchy; passed to a MemberReader, if present.
<CalculatedMemberProperty>	Property of a calculated member.
<Formula>	Holds the formula text within a <NamedSet> or <CalculatedMember>.
<ColumnDefs>	Holder for <ColumnDef> elements.
<ColumnDef>	Definition of a column in an <InlineTable> dataset.
<Rows>	Holder for <Row> elements.
<Row>	Row in an <InlineTable> dataset.
<Value>	Value of a column in an <InlineTable> dataset.
<MeasureExpression>	SQL expression used to compute a measure, in lieu of Measure.column.
<SQL>	The SQL expression for a particular database dialect.

- AggExclude
- AggFactCount
- AggForeignKey
- AggIgnoreColumn
- AggLevel
- AggMeasure
- AggName
- AggPattern
- AggTable
- CalculatedMember
- CalculatedMemberProperty
- CaptionExpression
- Closure
- ColumnDef
- ColumnDefs
- Cube
- CubeGrant
- CubeUsage
- CubeUsages
- Dimension
- DimensionGrant

- DimensionUsage
- Formula
- Hierarchy
- HierarchyGrant
- InlineTable
- Join
- KeyExpression
- Level
- Measure
- MeasureExpression
- MemberGrant
- NamedSet
- NameExpression
- OrdinalExpression
- Parameter
- ParentExpression
- Property
- PropertyExpression
- Role
- RoleUsage
- Row
- Rows
- Schema
- SchemaGrant
- SQL
- Table
- Union
- UserDefinedFunction
- Value
- View
- VirtualCube
- VirtualCubeDimension
- VirtualCubeMeasure

AggExclude

This element excludes a candidate aggregate table via name or pattern matching.

Attributes

Attribute	Data type	Definition
pattern	String	A Table pattern not to be matched.
name	String	The Table name not to be matched.
ignorecase	Boolean	Whether or not the match should ignore case.

AggFactCount

Specifies name of the column in the candidate aggregate table which contains the number of fact table rows.

Attributes

Attribute	Data type	Definition
N/A	N/A	N/A

AggForeignKey

Maps foreign key in the fact table to a foreign key column in the candidate aggregate table.

Attributes

Attribute	Data type	Definition
factColumn	String	The name of the base fact table foreign key.
aggColumn	String	The name of the aggregate table foreign key.

AggIgnoreColumn

Tells Mondrian to ignore a column in an aggregate table.

Attributes

Attribute	Data type	Definition
N/A	N/A	N/A

AggLevel

Maps a level to a column in the candidate aggregate table.

Attributes

Attribute	Data type	Definition
column	String	The name of the column mapping to the level name.
name	String	The name of the Dimension Hierarchy level.

AggMeasure

Maps a measure to a column in the candidate aggregate table.

Attributes

Attribute	Data type	Definition
column	String	The name of the column mapping to the measure name.
name	String	The name of the Cube measure.

AggName

Declares an aggregate table to be matched by name.

Attributes

Attribute	Data type	Definition
name	String	The table name of a specific aggregate table.

AggPattern

Declares a set of aggregate tables by regular expression pattern.

Attributes

Attribute	Data type	Definition
pattern	String	A Table pattern used to define a set of aggregate tables.

Constituent elements

Element	Definition
AggExclude	

AggTable

A definition of an aggregate table for a base fact table. This aggregate table must be in the same schema as the base fact table.

Attributes

Attribute	Data type	Definition
ignorecase	Boolean	Whether or not the match should ignore case.

Constituent elements

Element	Definition
AggFactCount	Describes what the fact_count column looks like.
AggIgnoreColumn	
AggForeignKey	
AggMeasure	
AggLevel	

CalculatedMember

A member whose value is derived using a formula, defined as part of a cube.

Attributes

Attribute	Data type	Definition
name	String	Name of this calculated member.
formatString	String	Format string with which to format cells of this member. For more details, see {@link mondrian.util.Format}.
caption	String	A string being displayed instead of the name. Can be localized from Properties file using #{propertyname}.
formula	String	MDX expression which gives the value of this member. Equivalent to the Formula sub-element.
dimension	String	Name of the dimension which this member belongs to.
visible	Boolean	Whether this member is visible in the user-interface. Default true.

Constituent elements

Element	Definition
Formula	MDX expression which gives the value of this member.
CalculatedMemberProperty	

CalculatedMemberProperty

Property of a calculated member defined against a cube. It must have either an expression or a value.

Attributes

Attribute	Data type	Definition
name	String	Name of this member property.
caption	String	A string being displayed instead of the Properties's name. Can be localized from Properties file using #{propertyname}.
expression	String	MDX expression which defines the value of this property. If the expression is a constant string, you could enclose it in quotes, or just specify the 'value' attribute instead.
value	String	Value of this property. If the value is not constant, specify the 'expression' attribute instead.

CaptionExpression

SQL expression used to compute the caption of a member, in lieu of **Level.captionColumn**.

Attributes

Attribute	Data type	Definition
N/A	N/A	N/A

Closure

Specifies the transitive closure of a parent-child hierarchy. Optional, but recommended for better performance. The closure is provided as a set of (parent/child) pairs: since it is the transitive closure these are actually (ancestor/descendant) pairs.

Attributes

Attribute	Data type	Definition
parentColumn	String	The parent column
childColumn	String	The child column

Constituent elements

Element	Definition
Table	

ColumnDef

Column definition for an inline table.

Attributes

Attribute	Data type	Definition
name	String	Name of the column.
type	String	Type of the column: String, Numeric, Integer, Boolean, Date, Time or Timestamp.

ColumnDefs

Holder for an array of ColumnDef elements.

Constituent elements

Element	Definition
ColumnDef	The columns to include in the array

Cube

A cube is a collection of dimensions and measures, all centered on a fact table.

Attributes

Attribute	Data type	Definition
name	String	Name of this cube.
caption	String	A string being displayed instead of the cube's name. Can be localized from Properties file using #{propertyname}.
defaultMeasure	String	The name of the measure that would be taken as the default measure of the cube.
cache	Boolean	Should the Fact table data for this Cube be cached by Mondrian or not. The default action is to cache the data.
enabled	Boolean	Whether element is enabled - if true, then the Cube is realized otherwise it is ignored.

Constituent elements

Element	Definition

Relation (as <Table>, <View>, or <InlineTable>)	The fact table is the source of all measures in this cube. If this is a Table and the schema name is not present, table name is left unqualified.
CubeDimension	
Measure	
CalculatedMember	Calculated members in this cube.
NamedSet	Named sets in this cube.

CubeGrant

Grants (or denies) this role access to a cube. access may be "all" or "none".

Attributes

Attribute	Data type	Definition
cube	String	The unique name of the cube

Constituent elements

Element	Definition
DimensionGrant	
HierarchyGrant	

CubeUsage

Usage of a base cube by a virtual cube.

Attributes

Attribute	Data type	Definition
cubeName	String	Name of the cube which the virtualCube uses.
ignoreUnrelatedDimensions	Boolean	Unrelated dimensions to measures in this cube will be pushed to top level member.

CubeUsages

List of base cubes used by the virtual cube.

Constituent elements

Element	Definition
CubeUsage	

Dimension

A Dimension is a collection of hierarchies. There are two kinds: a public dimension belongs to a Schema, and be used by several cubes; a private dimension belongs to a Cube. The foreignKey field is only applicable to private dimensions.

Attributes

Attribute	Data type	Definition
name	String	The name of this dimension.
type	String	The dimension's type may be one of "Standard" or "Time". A time dimension will allow the use of the MDX time functions (WTD, YTD, QTD, etc.). Use a standard dimension if the dimension is not a time-related dimension. The default value is "Standard".
caption	String	A string being displayed instead of the dimensions's name. Can be localized from Properties file using #{propertyname}.
usagePrefix	String	If present, then this is prepended to the Dimension column names during the building of collapse dimension aggregates

		allowing 1) different dimensions to be disambiguated during aggregate table recognition. This should only be set for private dimensions.

Constituent elements

Element	Definition
Hierarchy	The hierarchies (pre-defined drill-downs) for this dimension.

DimensionGrant

Grants (or denies) this role access to a dimension. access may be "all" or "none". Note that a role is implicitly given access to a dimension when it is given acess to a cube. See also the "all_dimensions" option of the "SchemaGrant" element.

Attributes

Attribute	Data type	Definition
dimension	String	The unique name of the dimension

DimensionUsage

A DimensionUsage is usage of a shared Dimension within the context of a cube.

Attributes

Attribute	Data type	Definition
source	String	Name of the source dimension. Must be a dimension in this schema. Case-sensitive.
level	String	Name of the level to join to. If not specified, joins to the lowest level of the dimension.
usagePrefix	String	If present, then this is prepended to the Dimension column names during the building of collapse dimension aggregates allowing 1) different dimension usages to be disambiguated during aggregate table recognition and 2) multiple shared dimensions that have common column names

		to be disambiguated.

Formula

Holds the formula text within a <NamedSet> or <CalculatedMember>.

Attributes

Attribute	Data type	Definition
N/A	N/A	N/A

Hierarchy

Defines a hierarchy, which is a pre-defined drill-down.

Note: You must specify at most one <Relation> or memberReaderClass. If you specify none, the hierarchy is assumed to come from the same fact table of the current cube.

Attributes

Attribute	Data type	Definition
name	String	Name of the hierarchy. If this is not specified, the hierarchy has the same name as its dimension.
hasAll	Boolean	Whether this hierarchy has an 'all' member.
allMemberName	String	Name of the 'all' member. If this attribute is not specified, the all member is named 'All hierarchyName', for example, 'All Store'.
allMemberCaption	String	A string being displayed instead as the all member's name. Can be localized from Properties file using #{propertyname}.
allLevelName	String	Name of the 'all' level. If this attribute is not specified, the all member is named '(All)'. Can be localized from Properties file using #{propertyname}.
primaryKey	String	The name of the column which identifies members, and which is referenced by rows in the fact table. If not specified, the key of the lowest level is used. See also CubeDimension.foreignKey.
primaryKeyTable	String	The name of the table which contains primaryKey. If the

		hierarchy has only one table, defaults to that; it is required.
defaultMember	String	
memberReaderClass	String	Name of the custom member reader class. Must implement the mondrian.olap.MemberReader interface.
caption	String	A string to be displayed in the user interface. If not specified, the hierarchy's name is used. Can be localized from Properties file using #{propertyname}.

Constituent elements

Element	Definition
RelationOrJoin (as <Table>, <View>, <Join>, or <InlineTable>)	The Table, Join, View, or Inline Table that populates this hierarchy.
Level	

HierarchyGrant

Grants (or denies) this role access to a hierarchy. access may be "all", "custom" or "none". If access is "custom", you may also specify the attributes topLevel, bottomLevel, and the member grants.

Attributes

Attribute	Data type	Definition
hierarchy	String	The unique name of the hierarchy
topLevel	String	Unique name of the highest level of the hierarchy from which this role is allowed to see members. May only be specified if the HierarchyGrant.access is "custom". If not specified, role can see members up to the top level.
bottomLevel	String	Unique name of the lowest level of the hierarchy from which this role is allowed to see members. May only be specified if the HierarchyGrant.access is "custom". If not specified, role can see members down to the leaf level.
rollupPolicy	String	Policy which determines how cell values are calculated if not all of the children of the current cell are visible to the current role. Allowable values are 'full' (the default), 'partial', and 'hidden'.

Constituent elements

Element	Definition
MemberGrant	

InlineTable

Defines a table using an inline dataset.

Attributes

Attribute	Data type	Definition
alias	String	Alias to be used with this table when it is used to form queries. If not specified, defaults to the table name, but in any case, must be unique within the schema. (You can use the same table in different hierarchies, but it must have different aliases.)

Constituent elements

Element	Definition
ColumnDefs	
Rows	

Join

Defines a 'table' by joining a set of queries.

Attributes

Attribute	Data type	Definition
leftAlias	String	Defaults to left's alias if left is a table, otherwise required.
leftKey	String	
rightAlias	String	Defaults to right's alias if right is a table, otherwise required.
rightKey	String	

Constituent elements

Element	Definition
RelationOrJoin (as <Table>, <View>, <Join>, or <InlineTable>)	There must be two defined; a **left**, and a **right**

KeyExpression

SQL expression used as key of the level, in lieu of a column.

Attributes

Attribute	Data type	Definition
N/A	N/A	N/A

Level

Level of a hierarchy.

Attributes

Attribute	Data type	Definition
approxRowCount	String	The estimated number of members in this level. Setting this property improves the performance of MDSCHEMA_LEVELS, MDSCHEMA_HIERARCHIES and MDSCHEMA_DIMENSIONS XMLA requests
name	String	
table	String	The name of the table that the column comes from. If this hierarchy is based upon just one table, defaults to the name of that table; otherwise, it is required. Can be localized from Properties file using #{propertyname}.
column	String	The name of the column which holds the unique identifier of this level.
nameColumn	String	The name of the column which holds the user identifier of this level.
ordinalColumn	String	The name of the column which holds member ordinals. If this column is not specified, the key column is used for ordering.
parentColumn	String	The name of the column which references the parent member in a parent-child hierarchy.
nullParentValue	String	Value which identifies null parents in a parent-child hierarchy. Typical values are 'NULL' and '0'.

type	String	Indicates the type of this level's key column: String, Numeric, Integer, Boolean, Date, Time or Timestamp. When generating SQL statements, Mondrian encloses values for String columns in quotation marks, but leaves values for Integer and Numeric columns un-quoted. Date, Time, and Timestamp values are quoted according to the SQL dialect. For a SQL-compliant dialect, the values appear prefixed by their typename, for example, "DATE '2006-06-01'".
uniqueMembers	Boolean	Whether members are unique across all parents. For example, zipcodes are unique across all states. The first level's members are always unique.
levelType	String	Whether this is a regular or a time-related level. The value makes a difference to time-related functions such as YTD (year-to-date).
hideMemberIf	String	Condition which determines whether a member of this level is hidden. If a hierarchy has one or more levels with hidden members, then it is possible that not all leaf members are the same distance from the root, and it is termed a ragged hierarchy.
formatter	String	Name of a formatter class for the member labels being displayed. The class must implement the mondrian.olap.MemberFormatter interface. Allowable values are: Never (a member always appears; the default); IfBlankName (a member doesn't appear if its name is null or empty); and IfParentsName (a member appears unless its name matches the parent's.
caption	String	A string being displayed instead of the level's name. Can be localized from Properties file using #{propertyname}.

captionColumn	String	The name of the column which holds the caption for members.

Constituent elements

Element	Definition
KeyExpression	The SQL expression used to populate this level's key.
NameExpression	The SQL expression used to populate this level's name. If not specified, the level's key is used.
OrdinalExpression	The SQL expression used to populate this level's ordinal.
ParentExpression	The SQL expression used to join to the parent member in a parent-child hierarchy.
Closure	
Property	

Measure

A measure is an aggregated numeric value. Typically it is a sum of selected numbers in a column, or a count of the number of items in a list.

Attributes

Attribute	Data type	Definition
name	String	Name of this measure.
column	String	Column which is source of this measure's values. If not specified, a measure expression must be specified.
datatype	String	The datatype of this measure: String, Numeric, Integer, Boolean, Date, Time or Timestamp. The default datatype of a measure is 'Integer' if the measure's aggregator is 'Count', otherwise it is 'Numeric'.
formatString	String	Format string with which to format cells of this measure. For more details, see the mondrian.util.Format class.
aggregator	String	Aggregation function. Allowed values are "sum", "count", "min", "max", "avg", and "distinct-count". ("distinct count" is allowed for backwards compatibility, but is deprecated because XML enumerated attributes in a DTD cannot legally contain spaces.)
formatter	String	Name of a formatter class for the appropriate cell being displayed. The class must implement the mondrian.olap.CellFormatter interface.

caption	String	A string being displayed instead of the name. Can be localized from Properties file using #{propertyname}.
visible	Boolean	Whether this member is visible in the user-interface. Default true.

Constituent elements

Element	Definition
MeasureExpression	The SQL expression used to calculate a measure. Must be specified if a source column is not specified.
CalculatedMemberProperty	

MeasureExpression

A schema is a collection of cubes and virtual cubes. It can also contain shared dimensions (for use by those cubes), named sets, roles, and declarations of user-defined functions.

Attributes

Attribute	Data type	Definition

Constituent elements

Element	Definition

MemberGrant

A schema is a collection of cubes and virtual cubes. It can also contain shared dimensions (for use by those cubes), named sets, roles, and declarations of user-defined functions.

Attributes

Attribute	Data type	Definition

Constituent elements

Element	Definition

NamedSet

A schema is a collection of cubes and virtual cubes. It can also contain shared dimensions (for use by those cubes), named sets, roles, and declarations of user-defined functions.

Attributes

Attribute	Data type	Definition

Constituent elements

Element	Definition

NameExpression

A schema is a collection of cubes and virtual cubes. It can also contain shared dimensions (for use by those cubes), named sets, roles, and declarations of user-defined functions.

Attributes

Attribute	Data type	Definition

Constituent elements

Element	Definition

OrdinalExpression

A schema is a collection of cubes and virtual cubes. It can also contain shared dimensions (for use by those cubes), named sets, roles, and declarations of user-defined functions.

Attributes

Attribute	Data type	Definition

Constituent elements

Element	Definition

Parameter

A schema is a collection of cubes and virtual cubes. It can also contain shared dimensions (for use by those cubes), named sets, roles, and declarations of user-defined functions.

Attributes

Attribute	Data type	Definition

Constituent elements

Element	Definition

ParentExpression

A schema is a collection of cubes and virtual cubes. It can also contain shared dimensions (for use by those cubes), named sets, roles, and declarations of user-defined functions.

Attributes

Attribute	Data type	Definition

Constituent elements

Element	Definition

Property

Member property that enables you to create subcategories for levels and hierarchies.

Attributes

Attribute	Data type	Definition
name	String	The display name of this property.
column	String	The data column that will determine this subcategory's content.
type	String	Data type of this property: String, Numeric, Integer, Boolean, Date, Time or Timestamp.
formatter	String	Name of a formatter class for the appropriate property value being displayed. The class must implement the mondrian.olap.PropertyFormatter interface.
caption	String	A string being displayed instead of the name. Can be localized from Properties file using #{propertyname}.

PropertyExpression

SQL expression used to compute the value of a property, in lieu of Property.column.

Attributes

Attribute	Data type	Definition
N/A	N/A	N/A

Role

A role defines an access-control profile. It has a series of grants (or denials) for schema elements.

Attributes

Attribute	Data type	Definition
name	String	The name of this role

Constituent elements

Element	Definition
SchemaGrant	
Union	

RoleUsage

Usage of a Role in a union Role.

Attributes

Attribute	Data type	Definition
roleName	String	The name of the role

Row

Row definition for an inline table. Must have one Column for each ColumnDef in the InlineTable.

Constituent elements

Element	Definition
Value	

Rows

Holder for an array of Row elements

Constituent elements

Element	Definition
Row	

Schema

A schema is a collection of cubes and virtual cubes. It can also contain shared dimensions (for use by those cubes), named sets, roles, and declarations of user-defined functions.

Attributes

Attribute	Data type	Definition
name	String	Name of this schema
measuresCaption	String	Label for the measures dimension. Can be localized from Properties file using #{propertyname}.
defaultRole	String	The name of the default role for connections to this schema

Constituent elements

Element	Definition
Parameter	This schema's parameter definitions.
Dimension	Shared dimensions in this schema.
Cube	Cubes in this schema.
VirtualCube	Virtual cubes in this schema.
NamedSet	Named sets in this schema.
Role	Roles in this schema.
UserDefinedFunction	Declarations of user-defined functions in this schema.

SchemaGrant

Grants (or denies) this role access to this schema. access may be "all", "all_dimensions", or "none". If access is "all_dimensions", the role has access to all dimensions but still needs explicit access to cubes.

Constituent elements

Element	Definition
CubeGrant	

pentaho

SQL

The SQL expression for a particular database dialect.

Attributes

Attribute	Data type	Definition
dialect	String	Dialect of SQL the view is intended for. Valid values include, but are not limited to: <ul><li>generic</li><li>access</li><li>db2</li><li>derby</li><li>firebird</li><li>hsqldb</li><li>mssql</li><li>mysql</li><li>oracle</li><li>postgres</li><li>sybase</li><li>teradata</li><li>ingres</li><li>infobright</li><li>luciddb</li></ul>

Table

A fact or dimension table.

Attributes

Attribute	Data type	Definition
name	String	The name of the table
schema	String	Optional qualifier for table.
alias	String	Alias to be used with this table when it is used to form queries. If not specified, defaults to the table name, but in any case, must be unique within the schema. (You can use the same table in different hierarchies, but it must have different aliases.)

Constituent elements

Element	Definition
SQL	The SQL WHERE clause expression to be appended to any select statement

AggExclude	
AggTable	

Union

Body of a Role definition which defines a Role to be the union of several Roles. The RoleUsage elements must refer to Roles that have been declared earlier in this schema file.

Constituent elements

Element	Definition
RoleUsage	

UserDefinedFunction

A UserDefinedFunction is a function which extends the MDX language. It must be implemented by a Java class which implements the interface mondrian.spi.UserDefinedFunction.

Attributes

Attribute	Data type	Definition
name	String	Name with which the user-defined function will be referenced in MDX expressions.
className	String	Name of the class which implemenets this user-defined function. Must implement the mondrian.spi.UserDefinedFunction interface.

Value

Column value for an inline table. The CDATA holds the value of the column.

Attributes

Attribute	Data type	Definition
column	String	Name of the column.

View

Defines a 'table' using a SQL query, which can have different variants for different underlying databases.

Attributes

Attribute	Data type	Definition
alias	String	

Constituent elements

Element	Definition
SQL	

VirtualCube

A cube defined by combining the dimensions and measures of one or more cubes. A measure originating from another cube can be a <CalculatedMember>.

Attributes

Attribute	Data type	Definition
enabled	Boolean	Whether this element is enabled - if true, then the Virtual Cube is realized otherwise it is ignored.
name	String	
defaultMeasure	String	The name of the measure that would be taken as the default measure of the cube.
caption	String	A string being displayed instead of the cube's name. Can be localized from Properties file using #{propertyname}.

Constituent elements

Element	Definition
CubeUsages	
VirtualCubeDimension	
VirtualCubeMeasure	

CalculatedMember	Calculated members that belong to this virtual cube. (Calculated members inherited from other cubes should not be in this list.)
NamedSet	Named sets in this cube.

VirtualCubeDimension

A VirtualCubeDimension is a usage of a Dimension in a VirtualCube.

Attributes

Attribute	Data type	Definition
cubeName	String	Name of the cube which the dimension belongs to, or unspecified if the dimension is shared.
name	String	Name of the dimension.

VirtualCubeMeasure

A VirtualCubeMeasure is a usage of a Measure in a VirtualCube.

Attributes

Attribute	Data type	Definition
cubeName	String	Name of the cube which the measure belongs to.
name	String	Unique name of the measure within its cube.
visible	Boolean	Whether this member is visible in the user-interface. Default true.