



Work with Relational Data Models



This document supports Pentaho Business Analytics Suite 5.0 GA and Pentaho Data Integration 5.0 GA, documentation revision August 22, 2013, copyright © 2013 Pentaho Corporation. No part may be reprinted without written permission from Pentaho Corporation. All trademarks are the property of their respective owners.

## Help and Support Resources

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](mailto:sales@pentaho.com).

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

## Liability Limits and Warranty Disclaimer

The author(s) of this document have used their best efforts in preparing the content and the programs contained in it. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, express or implied, with regard to these programs or the documentation contained in this book.

The author(s) and Pentaho shall not be liable in the event of incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of the programs, associated instructions, and/or claims.

## Trademarks

Pentaho (TM) and the Pentaho logo are registered trademarks of Pentaho Corporation. All other trademarks are the property of their respective owners. Trademarked names may appear throughout this document. Rather than list the names and entities that own the trademarks or insert a trademark symbol with each mention of the trademarked name, Pentaho states that it is using the names for editorial purposes only and to the benefit of the trademark owner, with no intention of infringing upon that trademark.

## 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 340  
5950 Hazelhine 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](mailto:sales@pentaho.com)

# Contents

About this Section.....	5
Knowledge Prerequisites.....	6
A Conceptual Overview of the Pentaho Metadata Editor.....	7
The Physical Layer.....	7
The Business View.....	8
The Abstract Business Layer.....	8
Incorporating Metadata.....	8
Examining the Sample Metadata Model.....	10
Creating a New Domain.....	12
Setting Up a Database Connection.....	12
Importing Physical Tables and Columns.....	14
Importing Tables Inside a Schema.....	14
Creating a Business Model.....	15
Creating Business Tables and Columns.....	16
Removing Unnecessary Business Columns.....	18
Creating Relationships between Business Tables.....	18
Creating Relationships Using the Editor Graph.....	20
Hadoop Hive-Specific SQL Limitations.....	21
Building a Business View.....	21
Building a Business View Using Manage Categories.....	21
Enriching Your Data — Understanding Metadata Concepts and Properties.....	24
Exceptions to the Rules.....	24
Required Properties.....	24
Applying Concepts and Properties in the Pentaho Metadata Editor.....	24
Applying Self Concepts: Managing Properties on the Model.....	25
The Properties List.....	25
The Properties Editor (Settings).....	25
Adding New Properties.....	26
Removing a Property.....	26
The Concept Editor.....	26
Building Concepts.....	27
Applying Concepts to Business Objects.....	27
Removing the Parent Concept.....	28
Metadata Security.....	29
Configuring the Security Service.....	29
Adding Column-Level Security Constraints.....	29
Adding Global Row-Level Security Constraints.....	30
MQL Formula Syntax For Global Constraints.....	30
Adding User or Role Row-Level Security Constraints.....	32
MQL Formula Syntax For User and Role Row-Level Constraints.....	32
Removing Security from Metadata Domain Repository.....	33
Adding Security to Metadata Business Objects.....	34
Defining Security Settings.....	34
Getting Security Settings Offline.....	35
Changing Security Constraints.....	35
Adding Row Level Security to a Pentaho Metadata Model.....	36
Applying Data Constraints.....	36
Configuring and Managing Internation Locales.....	38
Setting Up Locales.....	38
Importing and Exporting Domains.....	39
Importing a Domain.....	39
Exporting a Domain.....	39
Domain Backup and Recovery.....	40

Publishing a Domain to a Pentaho BA Server.....	41
Before you Publish Your Domain.....	41
Making a Model Available as a Data Source.....	41
Metadata Properties Reference.....	43
Out-of-the-Box Properties.....	43
Custom Properties.....	44
Required Properties per Business Object.....	45
Pentaho Metadata Formulas.....	46
Supported Functions.....	48
Supported Operators.....	50
Supported Aggregate Functions.....	50
Troubleshooting.....	51
Publishing.....	51
Managing Multiple Outer-Joins.....	51
Using the Delay Outer Join Conditions Property.....	52

## About this Section

---

This section provides you with information and instructions to help you use the **Pentaho Metadata Editor**, a tool that allows you to build metadata models and domains. There is also an exercise included that walks you through creating a metadata model using a small sample database.

There is sample data available if you want to test Metadata Editor before importing your own data. This sample data is inside of the Pentaho download. This process will be covered later in this section.

# Knowledge Prerequisites

---

To use Pentaho Metadata Editor, you must have database administrator skills (such as knowing how to import tables, create relationships between tables, assign aggregations, add categories, assign security, etc.) and extensive knowledge of your database(s). You must also know what type of data interests your business users so you can determine whether a column should be a measure or a dimension, for example. These skills allow you to map a business user model (logical model) to a complex relational database. Ultimately, the metadata models you provide to business users allows them to create Pentaho reports without requiring DBA assistance.

See [Examining the Default Metadata Model](#) for information about how metadata models are implemented in the Pentaho BA Server.

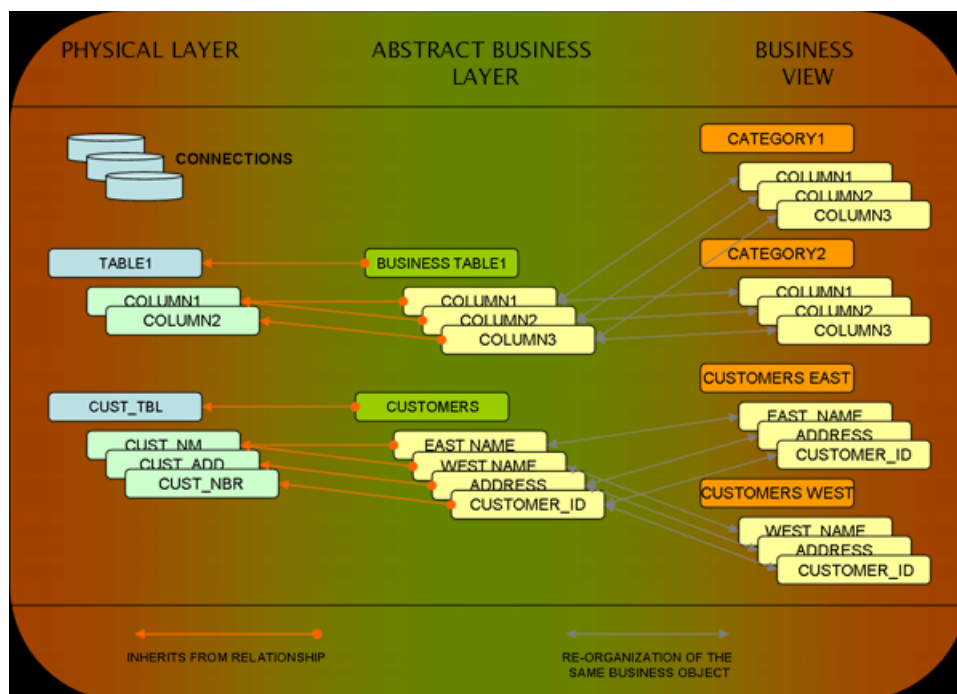
# A Conceptual Overview of the Pentaho Metadata Editor

Pentaho Metadata Editor (PME) is a tool that allows you to build Pentaho metadata domains and relational data models. A Pentaho Metadata Model maps the physical structure of your database into a logical business model. These mappings are stored in a centralized metadata repository and allow administrators to:


- Create business-language definitions for complex or cryptic database tables
- Decrease the cost and impact associated with low level database changes
- Set security parameters limiting user's report access to data
- Drive formatting on text, date, and numeric data improving report maintenance
- Localize the information to the user's regional settings

The goal of the Pentaho Metadata Editor is to simplify the experience of business users when they are creating reports.

The metadata business model is actually one major component in a Pentaho Metadata Domain. The domain encapsulates both the physical descriptions of your database objects and the logical model (the business model), the abstract representation of the database. The purpose of this page is to give a Pentaho metadata consumer a good idea of the relationships involved throughout a Pentaho domain, in order to fully leverage the power of the business models. This diagram depicts the business objects and relationships within the Pentaho metadata domain:



In this diagram, both the relationship arrows in the legend as well as the color of the business objects are significant. Each independent business object has its own color; thus, connections, physical tables, business tables, physical columns, business columns and categories are the main business objects in a Pentaho domain. The two types of relationships depicted in the diagram are key to understanding Pentaho metadata. The "inherits from" relationship is a relationship between two different business objects where one will inherit metadata from the other, with the ability to override any of the inherited metadata properties. An example of this is that a business table inherits metadata from its associated physical table. The second relationship, the "same object" relationship, is one where the two business objects are actually one and the same, just represented in a different organization or duplicated for usability. This is the relationship between business columns in the Abstract Business Layer, and business columns in the Business View.

 **Important:** Although the Pentaho Metadata Editor displays a checkbox for each of the permissions available in the Pentaho BA Server, only **Execute** is enforced.

## The Physical Layer

The **Physical Layer** of a Pentaho domain encompasses connections, physical tables and physical columns. These objects represent the database(s) you are trying to model and enrich with metadata. The Physical Layer is not

considered part of the business model, because not all connections defined in the Physical Layer will be used in every business model.

Multiple business models can be created in one domain, but those models must have one and only one connection reference. This means that you cannot mix and match physical tables from two different connections in the same model yet. We realize that this prevents the model from supporting multiple data sources, as well as severely limits the Pentaho Metadata Editor's ability to allow table changes across connections, a feature necessary for moving from dev to production databases, for example. Fortunately, you can get around the latter with a bit of hand-editing XML. Note that these features are important to us and are early inclusions in the metadata roadmap.

## The Business View

---

The Business View is the part of the business model that applications will operate against, and end-users will see. The Business View is nothing more than "buckets" (called categories) for you to re-arrange and re-organize your business columns in a fashion that makes sense to the consumers of the data.

In the Business View, you can create any number of categories and arrange your business columns in those categories however it best suits your business, mixing and matching columns that derived from different business tables, even adding business columns more than once to different categories. The only restriction to the Business View is you cannot add the same business column more than once to a single category.

## The Abstract Business Layer

---

The **Abstract Business Layer** is the heavy lifter in the metadata business model. The business model encompasses the Abstract Business Layer and the Business View. In the Abstract Business Layer, you have business tables, business columns, and business relationships.

You can create business tables for any physical table you have defined in the Physical Layer. You can also create more than one business table to reference the same physical table. The same rules apply for business columns. This can be useful in a multitude of scenarios, filtering security or even data at this level being one example. The business table keeps a reference to the physical table that it models, and this allows a metadata inheritance relationship between physical tables and business tables. If you define metadata on a physical table, the business table will inherit that metadata, unless and until the business table itself has overridden the inherited metadata. This concept operates on a metadata property to property basis, meaning that each property can be overridden individually.

The same relationship exists between physical columns and business columns. If you define metadata on a physical column, the business column will inherit that metadata, unless and until the business column itself has overridden the inherited metadata. Then there are relationships. Don't confuse the term relationships here with the relationships described in the domain diagram. The relationships described here are not represented in that diagram. These relationships are mappings that you define to describe the relational (or other) bonds between your business tables. One example may be a one to many relationship between a customer table and an orders table. The strength in metadata relationships is that you can identify relationships between columns or tables in the Abstract Business Layer that may not be obvious in the Physical Layer. An example would be to compare budget, actual and sales numbers, using a formula-derived business column that is specific to your business rules.

## Incorporating Metadata

---

Each business object in the domain can have metadata associated with it, with the exception of categories. In Pentaho terminology, a collection of metadata properties is called a concept.

Each business object can have three levels of **concepts**: its own (self or child concept), a parent concept and an inherited concept. All of the defined metadata properties from the three levels are available (to metadata-aware applications, end-users) on a business object. It is important to understand what the override hierarchy is, when more than one concept level has defined the same metadata property.

When a metadata property is set in the business object itself, and that same property is in play (either inherited or from a parent concept), the business object's self or child concept overrides all others. The next level in the hierarchy is the parent concept. So, for example, suppose the name property on a business object is *not* set on the object itself, but *is* inherited from its physical counterpart. In this instance, you set a parent concept on the business object with a name property defined in the parent concept. The parent concept's name property overrides the inherited name property.



Ultimately, the inherited metadata properties are used if, and only if, the same property is not defined somewhere down the hierarchical chain, as part of a self concept or parent concept.

# Examining the Sample Metadata Model

- The Pentaho Metadata Editor must be running.
- The Pentaho BA Server must be running.

Follow the instructions to see the example metadata model that is deployed with the BA Server.

1. In the main window of the Metadata Editor, go to **File > Import from XMI File**.
2. Navigate to your installation of the BA Server (for example, C:\Program Files\pentaho\server\bi-server-ee\pentaho-solutions\, if you used the installer).

There can be several root pentaho-solution folders and each of them may contain one metadata model repository.

3. Open the **steel-wheels** folder and click **[MyBusinessModel].xmi** where [myBusinessModel] is any name you have given the model.

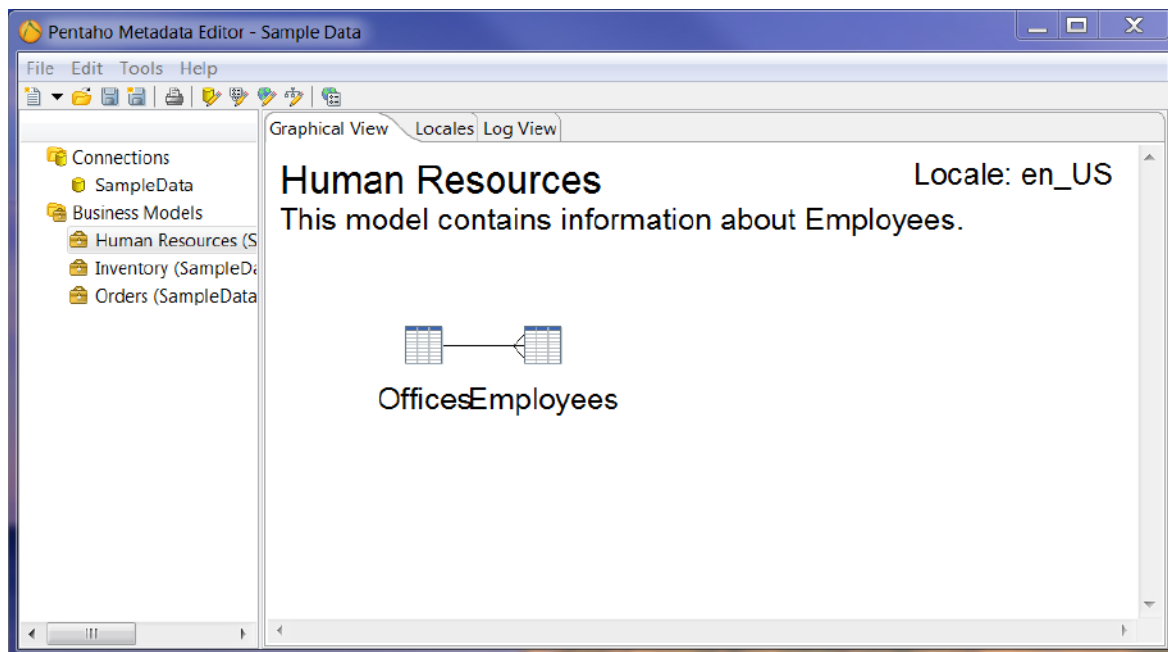
The .xmi is the repository for Pentaho-related metadata and business views.

4. Type **Sample Data** in the **Save Model** dialog box.

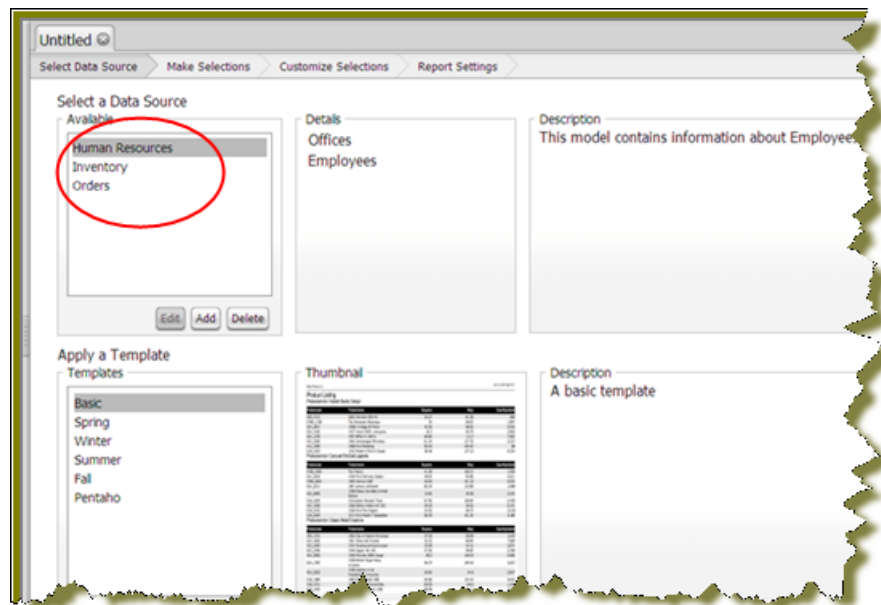


**Note:** If you see the message, "This model already exists....," click **Yes** to continue.

This step processes the file and displays the structure of the repository (connections and business models) on the navigation pane on the left.



5. Log into the User Console, and click **New Report**.



Notice that the business models (Human Resources, Inventory, and Orders) are displayed.

# Creating a New Domain

---

A metadata **domain** is a Pentaho term that represents all of the business objects created, stored and used in the metadata layer. A domain may consist of one or more connections, one or more models, security information, business tables, business views, categories, columns and concepts. You can create and save multiple metadata domains using the Metadata Editor.

A metadata domain is accessed by the BA Server by publishing or exporting the domain to an `.xmi` file, and placing the file in the Pentaho solutions repository.

In summary, a domain represents all of the associated modeled business entities. A domain can be viewed as metadata “document.” Each solution is restricted to have, at most, one domain. A solution repository can contain multiple solutions. A domain must be published as an `.xmi` file in the solution repository root. The Metadata Editor works with one domain at time; for example, a “Sales” domain that defines the relationships and entities used by sales team.

When you first launch the Metadata Editor, a new domain is automatically created. You can immediately begin adding connections, tables, columns, and more. If you want to start fresh with a new domain, select **File > New > Domain File** from the main menu.

There are several procedures associated with creating a new domain.

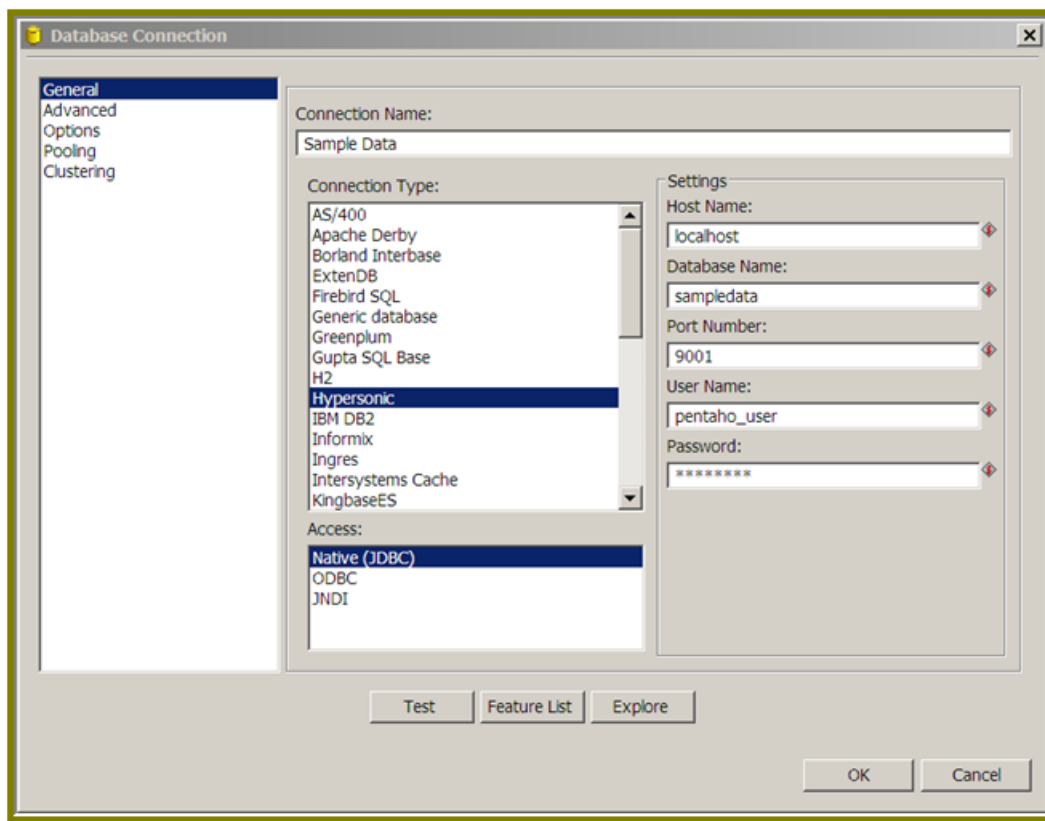
- Setting Up a Database Connection
- Importing Physical Tables and Columns
- Creating a Business Model
- Creating Business Tables and Columns
- Creating Relationships between Business Tables
- Building a Business View


## Setting Up a Database Connection

---


A connection represents connection information of a specific database, and acts as the parent in the hierarchy for all physical tables and physical columns that are defined for that database.

Pentaho metadata models can connect to most common relational databases using JDBC. The Pentaho Metadata Editor (and the Pentaho Metadata Architecture) supports a vast and rich set of data sources. Before you begin defining your business model, you must first describe the database or data source that you would like to model. You do this by defining one or more connections in the editor.



 **Note:** While the current implementation of Pentaho Metadata Editor supports multiple connections and multiple business models in the same domain, each business model must use physical objects (columns and tables) from a single connection only.

To create a new connection...

Step	Description
1	Right-click <b>Connections</b> in the navigation pane.
2	Select <b>New Connection</b> . The <b>Database Connection</b> dialog box appears.
3	Enter a <b>Connection Name</b> .
4	Define your database connection as needed.
5	Click <b>Test</b> to ensure that your entries are correct.
6	Click <b>OK</b> to save your connection. The list of tables in your database to be imported appears. If you want to find a specific table, enter the table name in the search box or use a regular expression. Select the tables you want to import and click <b>OK</b> or <b>Cancel</b> when you are done.   <b>Note:</b> You don't need to import tables at this point. See <a href="#">Importing Physical Tables and Columns</a> for more information.

### Connection Types - Adding a New Database Type

First, note the **Connection Type** list in the dialog box. This is the list of database connections that the Pentaho Metadata Editor and metadata models can support. The list is quite extensive. If you do not see your database in the Connection Type list, you may be able to add it.

To add a new database type, you must copy the JDBC driver archive for your database into the PME install directory `...\metadata-editor\lib`. Restart the Pentaho Metadata Editor, and you will see your database in the Connection Type list.

## Method of Access

Under Connection Type, you will see the **Access** list. Defining a JDBC or ODBC connection typically requires all of the remaining fields associated with the **General** tab to have the correct information; however, if you are into abstracting those details from your metadata domain, then use the JNDI method of access.

The JNDI access method keeps your server implementation cleaner as well; once you publish your domain to the server, as long as you have defined the JNDI connections with the same names, you still have a good implementation where your database information is only described to the JNDI layer. To take advantage of the JNDI method of access in the Pentaho Metadata Editor, you must define your database connection information in a properties file for the editor.

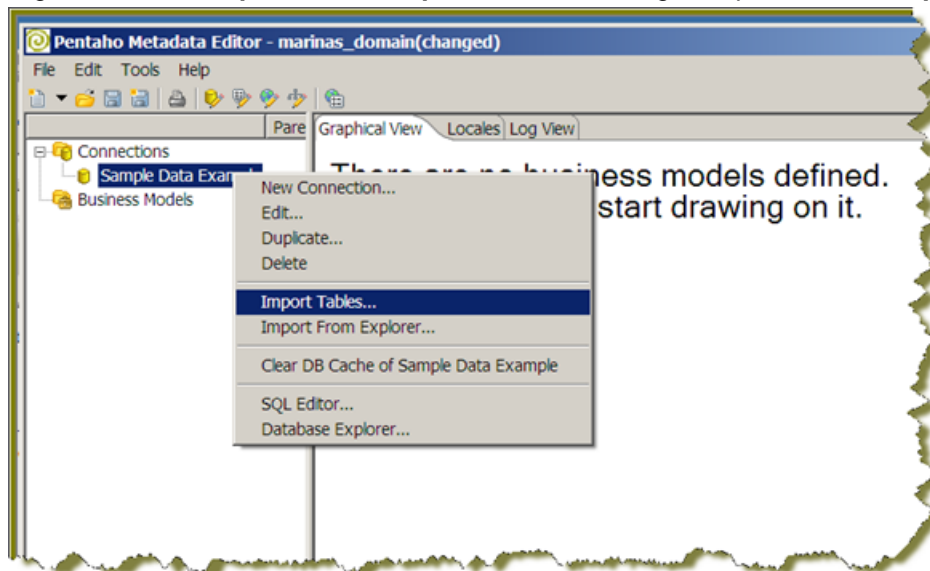
## Importing Physical Tables and Columns

After you define your database connections, the next step is to describe the physical database tables and columns that you want to include in your business model.

A **Business Model** contains all of the logical, abstract business objects and relationships that model your physical database objects in such a way that underlying changes to the physical database objects have minimal impact on your business, your business intelligence application, and your end users. There can be multiple business models in a single domain. A business model currently supports one database connection, and consists of business tables, relationships, and a multiple business views.

Fortunately, when you import a physical table, all of the table's columns come with it, so the import is a one step exercise instead of two. Later, you can remove the columns you do not want in the connection or the model. Below is an **example** of an import from HSQLDB (Hypersonic):

1. Right-click the **Sample Data Example** node in the navigation pane. Select **Import Tables** from the menu.



A dialog box that contains all physical tables available in the database appears.

2. Select the tables you want to include.



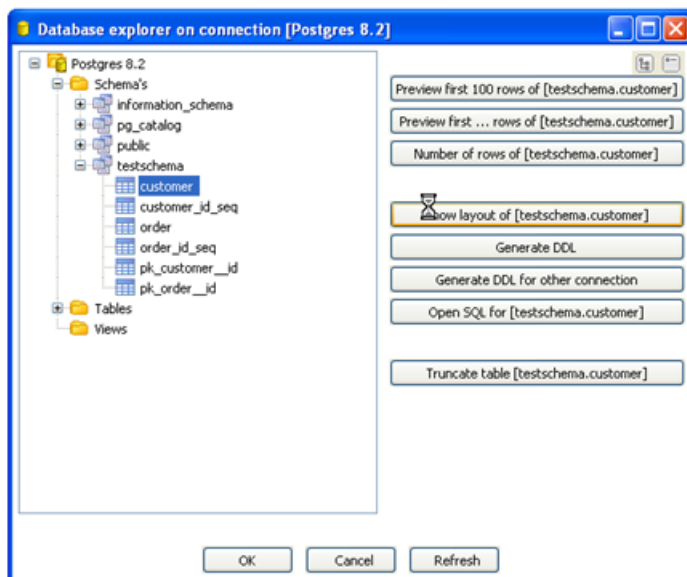
**Tip:** Press the <SHIFT> or <CTRL> key to select multiple tables.

3. Click **OK** when you are done.


The tables you selected appear in a list in the navigation pane in the left side of the workspace.

## Importing Tables Inside a Schema

If you are accessing a table inside a schema, use the **Import from Explorer** command.



1. Right-click on the connection and select **Import from Explorer**.  
A dialog box that contains all physical tables available in the database appears.
2. Navigate under **Schema's** to locate the correct table(s) to import.  
  



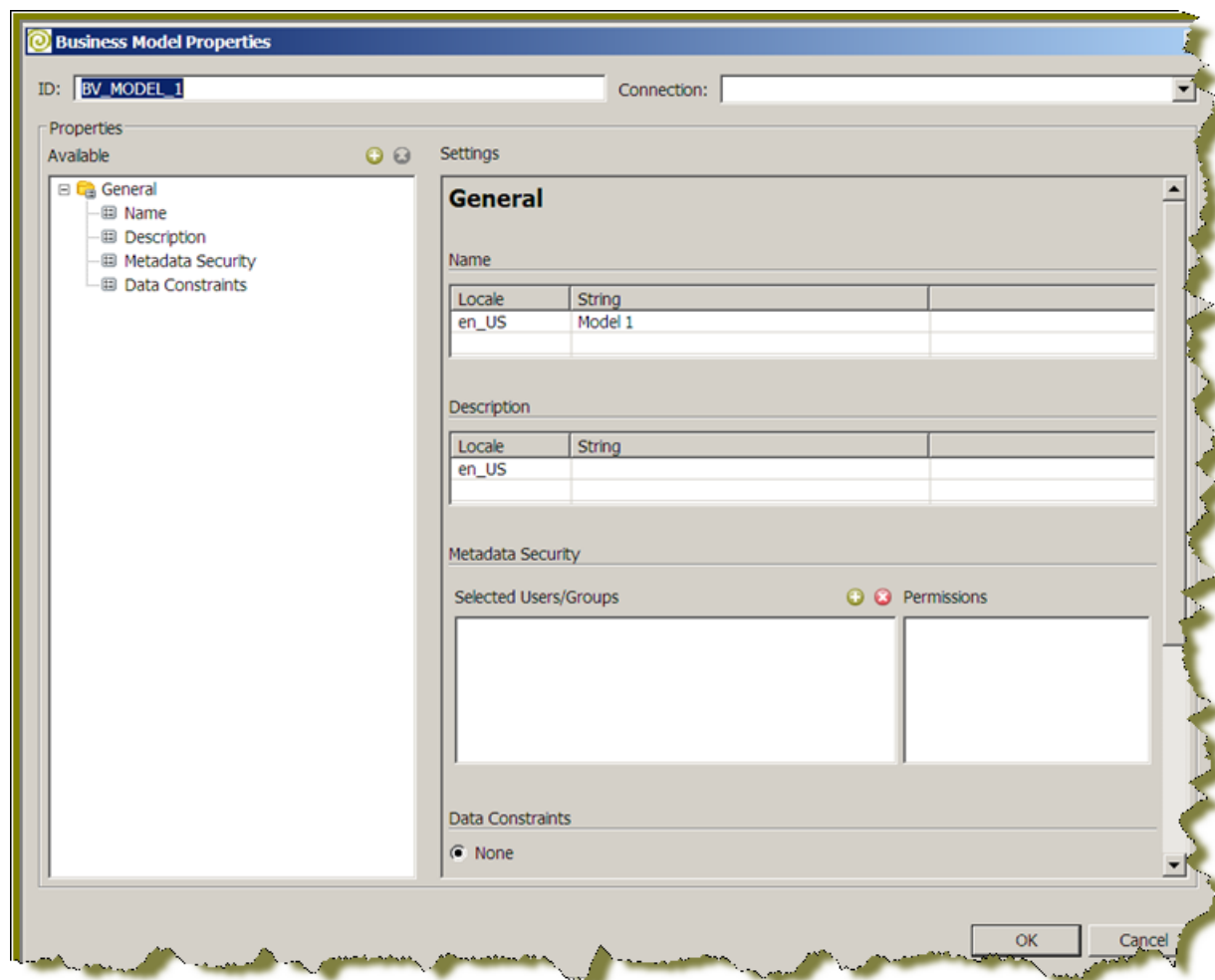
**Important:** You must select the individual tables. Selecting the top level schema exclusively results in an error message.
3. Click **OK**.  
The tables you selected appear in a list in the navigation pane in the left side of the workspace.

## Creating a Business Model

---

The business model is where the logical mapping of business objects to the physical tables and columns you set up takes place. The model is also the place where you define the relationships between your business tables, and organize the business view.

1. Right-click **Business Models** in the navigation pane and select **New Business Model**.



The Business Model Properties dialog box appears as shown above.

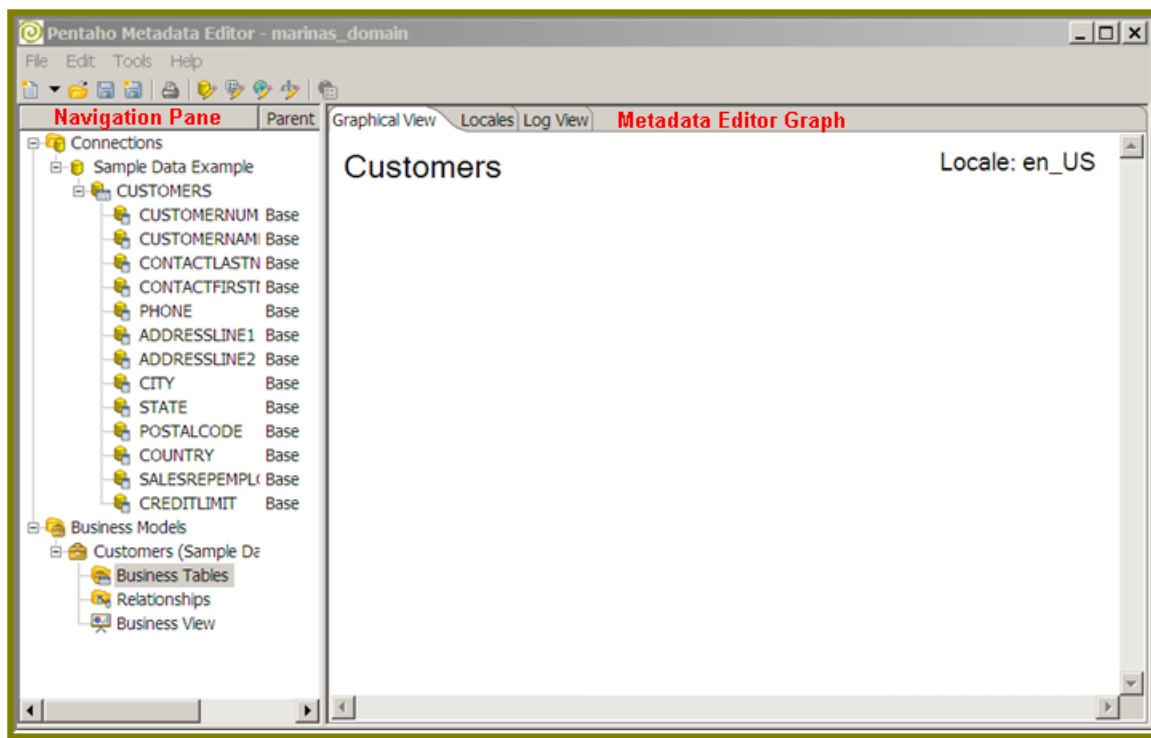
2. At the top of the dialog box, there is an ID text field, pre-populated with a value. Pentaho recommends you accept the pre-populated value as this value *must* be unique across all models that you define.
3. To name your new model, enter a new value under the **Name** property text box on the right.
4. Click **OK** to save your changes and exit the dialog box.

## Creating Business Tables and Columns

After you create a business model you must add the business tables and business columns; then, you create the relationships between your business tables.


Up to this point, you have used the navigation pane to build the business objects. The workspace on the right side of the page is called the **Metadata Editor Graph**. Use the Graph to lay out your business tables and show the relationships between them. While you can accomplish the rest of your tasks by manipulating the objects in the navigation pane, you can also use the Graph.





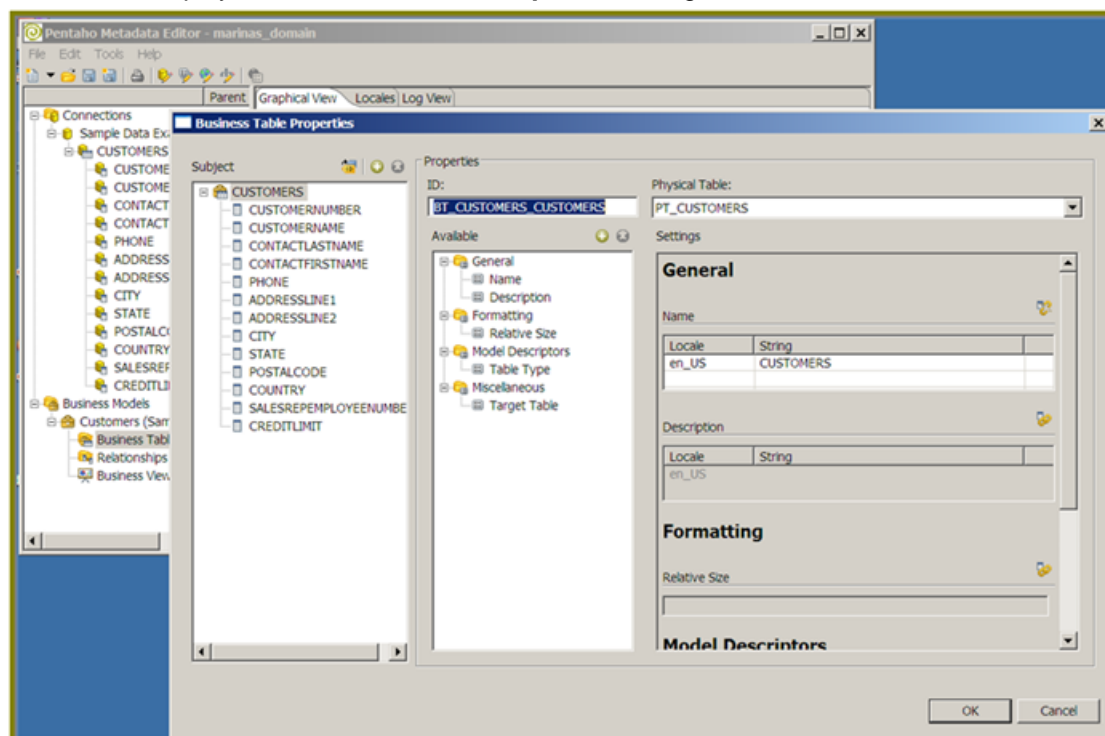
To create business tables and columns...

1. Right-click **Business Tables** in the navigation pane. Alternatively, you can right-click inside the **Metadata Editor Graph** workspace and select **Business Tables**.

 **Tip:** Drag the physical table from the navigation pane to the Graph workspace to create a new business table based on that physical table. Another shortcut is to drag the physical table in the navigation pane to the **Business Tables** node, which will also create a new business table based on that physical table.

A list of physical tables, each prefixed by the connection name to which they belong appears.

2. Select the physical table you want associated with the new business table.
3. Click **OK** to display the **Business Table Properties** dialog box.



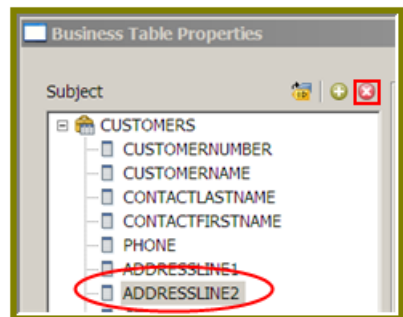
In the **Business Table Properties** dialog box, the **ID** field is pre-populated with a value. This ID identifies a specific business table, and must be unique. Pentaho recommends that you accept the default value for this field.


The table you chose in Step 3 is listed next, then below the fields is another navigation pane, with the name of your business table and all the columns inherited from the physical table included as business columns. The business columns are created for you when the new business table is created, so no need for the extra steps of creating them separately.

## Removing Unnecessary Business Columns

To delete unnecessary business columns...

1. In the **Business Table Properties** dialog box navigation pane, select the column you want to remove.



2. Click  (Delete Column).  
The column you selected is removed from the list.
3. Repeat Steps 1 and 2 with the remaining columns you want to delete.
4. Click **OK** when you are done.

## Creating Relationships between Business Tables

Once you have all of your business tables created, you must create relationships between the tables so that the query generators and SQL generators that work with Pentaho Metadata can create the data queries correctly. This is similar to drawing a relational diagram to show primary and foreign key relationships; however, relational links are not the only relationships that can be modeled. You can create a relationship between any two tables, link any two columns between them and dictate what the relationship is (one to many, many to many, and so on). The important pieces of information to know before you try to create a relationship are:

- What two business tables would you like to associate with this relationship?
- What columns in the business tables identify the relationship?
- What type of relationship is it — one to one, one to many, many to one, and so on?

To create a new relationship between business tables in the navigation pane, first make sure that the model you want to add this relationship to is selected, and that the **Relationships** node is visible.

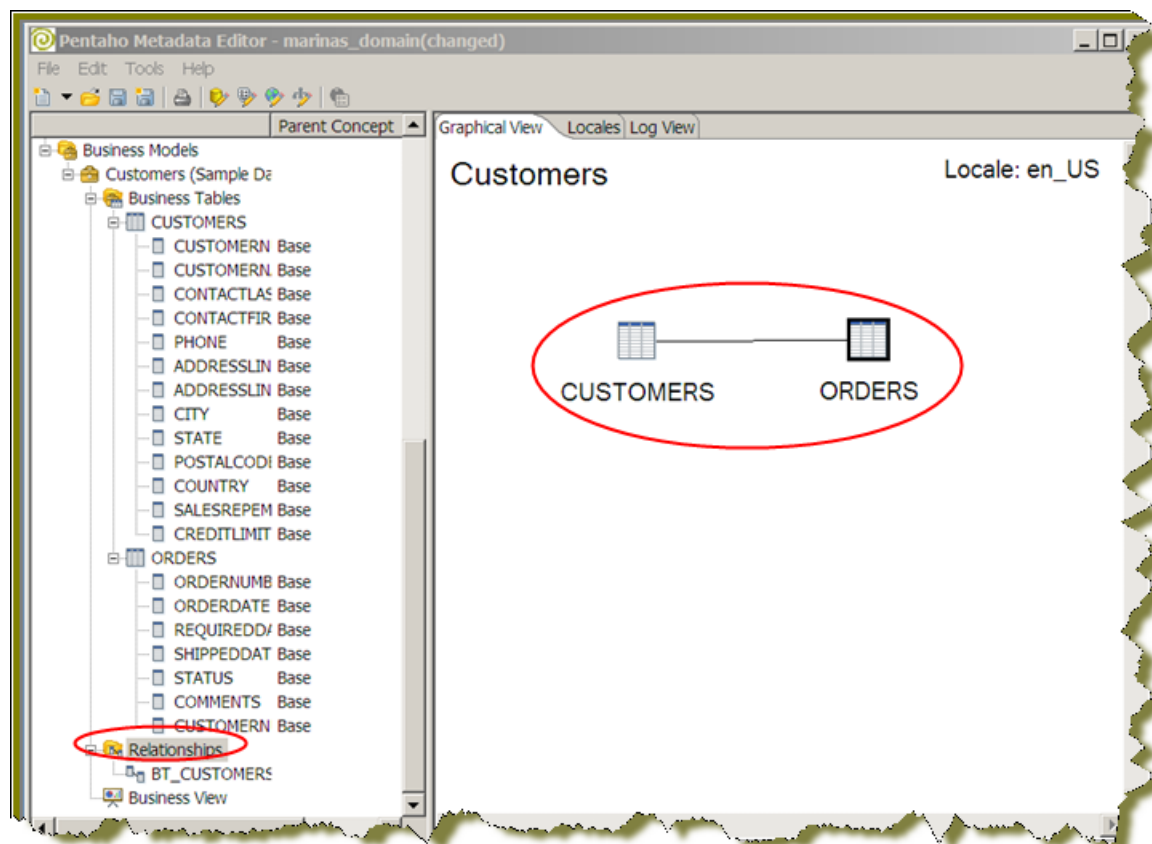
1. Right-click **Relationships** in the navigation pane.
2. Select **New Relationship**  
The **Relationship Properties** dialog box appears.
3. Select a business table from the **From Table/Field** list. This is the first relationship
4. Select a business table from the **To Table/Field** list. This steps sets up a relationship between two tables.
5. Now, specify the business columns (from the adjacent lists) from each business table that identify this relationship.  
If the business column names are similar, click **Guess Matching Fields** and let Pentaho Metadata Editor attempt to determine the columns for you.
6. Define the relationship from the **Relationship** drop down list. The relationships that can be chosen are defined with examples:

Relationship	Description
<b>1:N</b>	A <b>one-to-many</b> mandatory relationship is the most common relationship in databases. The primary key table contains only one record that relates to none, one, or many records in the related table. This relationship is similar to the one between you and one of your

Relationship	Description
	parents. You have one mother, but your mother may have several children.
<b>N:1</b>	A <b>many-to-one</b> is opposite of one to many (1:N) relationship.
<b>1:1</b>	In a <b>one-to-one</b> relationship, both tables are limited to one record only on either side of the relationship. Each primary key value relates to a single record, or no record, in the associated table. They are like spouses — you may be married, or not; however, if you are married, both you and your spouse can have only one partner. Most one-to-one relationships are forced by business rules. If you do not have a business rule, you can, in most cases, combine both tables into one table without breaking normalization rules.
<b>0:N</b>	A <b>zero to many</b> optional relationship indicates that a person may have no phone, one phone, or many phones, and that the phone may not be "owned," but can only be owned by a maximum of one person.
<b>N:0</b>	Opposite of a zero to many relationship
<b>0:1</b>	A <b>zero to one</b> relationship might indicate that a person may be a programmer, but a programmer must be a person. It is assumed that the mandatory side of the relationship is the dominant.
<b>1:0</b>	Opposite of a zero to one relationship
<b>N:N</b>	In a many to many relationship each record in both tables can relate to an unlimited number of records (or no records) in the other table. For example, if you have many siblings, your siblings also have many siblings. Many-to-many relationships must have a third table, referred to as an associate or linking table, because relational systems cannot accommodate the relationship directly.
<b>0:0</b>	A <b>zero to zero</b> optional relationship indicates that a person may occupy one parking space, but that a person is not necessary to have a space and a space does not need to have a person.

7. Click **OK** when you are done.

You should see a new relationship line drawn between the two tables on the Editor Graph, and the relationship represented in the navigation pane.

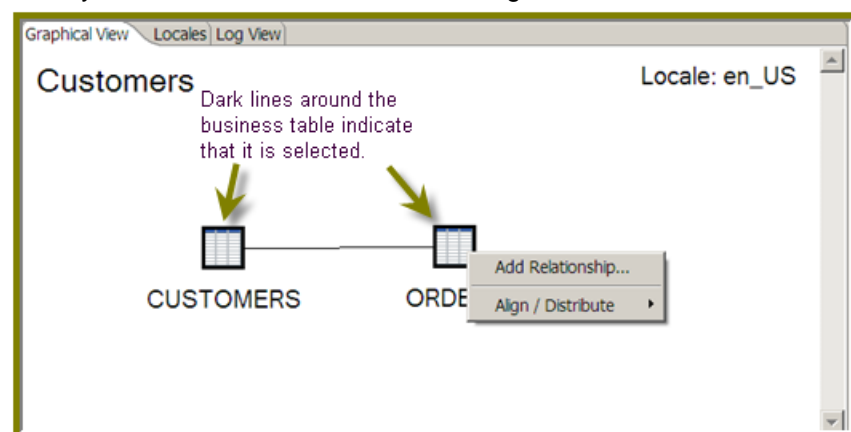


**Important:** Note that complex joins appear in the WHERE clause of the SQL statement, so currently any joining that takes place in the FROM clause of the SQL statement is not supported. An example of a complex join might be `TABLE_A.COL_A=TABLE_B.COL_A AND TABLE_A.COL_B=TABLE_B.COL_B`. This represents a join of two tables based on two key columns versus a single join column. Also note, the complex join expression provided must use the names of the physical tables and physical columns, not business tables and business column names.

## Creating Relationships Using the Editor Graph

In the Editor Graph, creating a new relationship is somewhat simplified because you select the two business tables on the canvas, and the **Relationship Properties** dialog box is pre-populated with your selections. Before you start, make sure that the model you want to add a relationship to is selected, and that the business tables are displayed in the Editor Graph.

1. Select the two business tables you want to include in the new relationship, either by clicking and dragging a marquee around the tables or by holding the **<SHIFT+CTRL>** keys, then clicking on the tables.
2. Once your business tables are selected, right-click on the selection. Click **Add Relationship...** in the popup menu.



The **Relationship Properties** dialog box appears.

3. Continue following steps 3 through 7 in [Creating Relationships between Business Tables](#) to finish creating the relationship.

## Hadoop Hive-Specific SQL Limitations

There are a few key limitations in Hive that prevent some regular Metadata Editor features from working as intended, and limit the structure of your SQL queries in Report Designer:

- **Outer joins are not supported.**
- **Each column can only be used once in a SELECT clause.** Duplicate columns in SELECT statements cause errors.
- **Conditional joins can only use the = conditional unless you use a WHERE clause.** Any non-equal conditional in a FROM statement forces the Metadata Editor to use a cartesian join and a WHERE clause conditional to limit it. This is not much of a limitation, but it may seem unusual to experienced Metadata Editor users who are accustomed to working with SQL databases.

## Building a Business View

---

The last step in creating a business domain is building your business view. A **Business View** is a collection of business categories that represents the "view" of your model, typically consumed by your end users. Each model can have one and only one business view. Business views are made up of logically (logically relevant to your organization or end-users) organized business categories and business columns.

A business category is like a bucket where you group and re-group your business columns. Business categories can mimic your business table names or be named after your favorite rock stars. Categories do not have metadata associated with them, have no tie back to any business table (although the Editor Graph gives you the impression this relationship exists; don't be fooled), and have the simple purpose of allowing you to bucket the business columns in your model as intuitively as possible for your data consumers.

Building a business view consists of creating your categories, then moving your business columns from the business tables into the categories. You can move columns from different business tables into the same category, and even duplicate the same business column into two different categories.

The Editor Graph represents the business tables portion of the business model only, so you'll use the navigation pane and the Category Editor to create a business view.

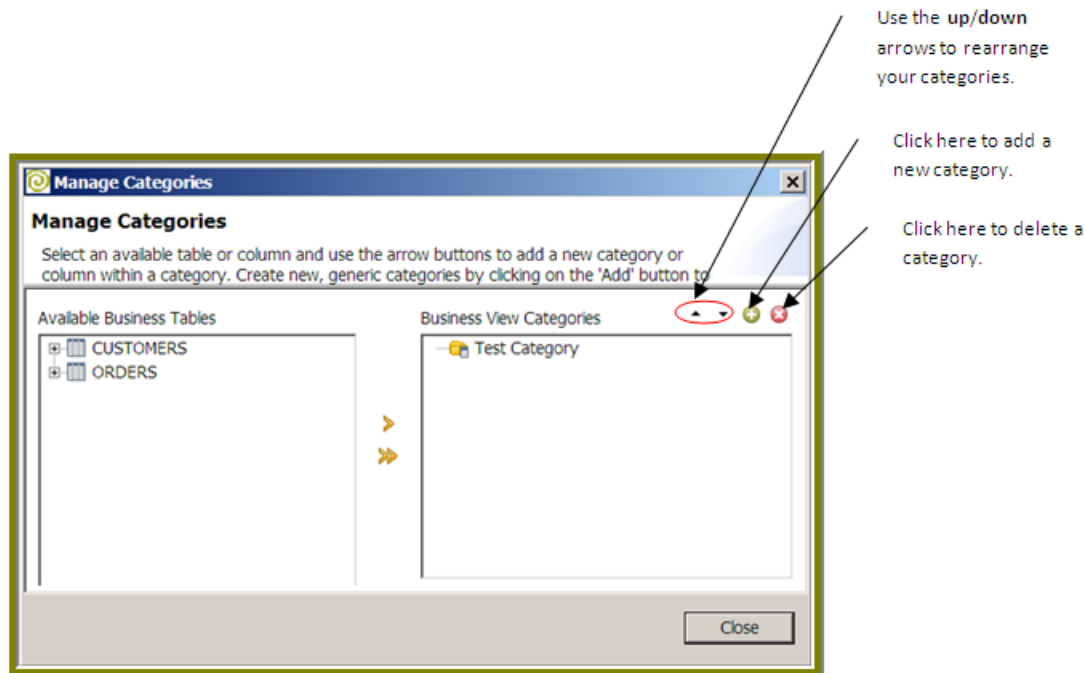
To create a new category using the navigation pane, first make sure that the model you want to add this category to is selected, and the **Business View** node is visible.

1. In the navigation pane, right-click **Business View**.
2. Select **New Category**.  
The **Business Category Properties** dialog box appears.
3. Type a name for the category in the **ID** field.
4. Click **OK**.

## Building a Business View Using Manage Categories

Follow the instructions below to build a business view using the **Manage Categories** dialog box. Before you begin, make sure that the model whose business view you want to create is selected, and the Business View node is visible in the navigation pane.

1. Double-click on Business View in the navigation pane. Alternatively, go to **Tools -> Manage Categories**.  
The Manage Categories dialog box appears.
2. Create categories directly from your business tables by moving columns into categories, add new categories, and remove categories and columns.



3. Click **Close** when you are done.

### Create a New Category from a Business Table

The navigation tree in the Manage Categories dialog box lists all of the business tables in your view. Follow the instructions below to create a new category from a business table in the Manage Categories dialog box:

1. First select the table in the navigation tree on the left.
2. Click ➤ (Add).

Notice that your category has the same name as your business table, and all of your business columns were added to the new category by default.

3. Click **Close** if you are done.

### Moving Columns into Categories

Follow the instructions below to move columns into categories in the Manage Categories dialog box:

1. Select the table from the list of available business tables. To view the columns under a business table, click the plus sign (+) to the left of each table name.
2. Select the destination category from the list on the right.
3. Click ➤ (Add).
4. Click **Close** if you are done.

### Adding a New Category


Follow the instructions below to add a new category in the Manage Categories dialog box.

1. Click + (Add) in the upper right corner of the Business View Categories list.  
The Business Category Properties dialog box appears.
2. Enter an ID (name) for your category, and click **OK**.  
The new category name appears in the **Manage Categories** dialog box.
3. Click **Close** if you are done.

### Remove Categories and Columns

Follow the instructions below to remove categories and/or columns in the Manage Categories dialog box:

1. Select the items you want to remove in the **Business View Categories** list on the right.

2. Click  (Delete).
3. Click **Close** when you are done.

# Enriching Your Data — Understanding Metadata Concepts and Properties

---

Presumably the end goal when creating a metadata domain is to have a model that is available to end-users and applications that allows them to consistently and richly enhance the data with which they are working. Whether that be through formatting, security metadata, or some other custom metadata, it all should result in a more efficient, consistent, clearer and smarter view of the data.

So far, you have walked through modeling your database in a manner that is more intuitive than its physical representation. The second half of the metadata work is defining the metadata. The Pentaho metadata paradigm uses the term **concept** to mean a collection of metadata properties that can be applied to a given business object (business table or column, for example).

A "property" has an identifier (a key into a map actually) and a value. This collection of properties is the map of attributes that you want to apply to a particular business object, such as a business column or business table. The concept IS the metadata you define for your business objects. Each business object (physical table, business table, columns, and so on) has its own concept whose properties override all other InheritedConcept or ParentConcept concepts.

Concepts can also be defined independent of any business object and can be structured in an inheritance hierarchy for better organization and management of your metadata. These independent concepts can then be applied to one or more business objects as a ParentConcept.

Refer to [Introducing the Pentaho Metadata Editor](#) for details regarding concepts and how they are used.

## Exceptions to the Rules

---

Like all exceptions, the exceptions to the metadata inheritance logic is complex, but well-grounded. There are two sets of properties that change how the typical inheritance is accomplished — **required properties** and the **default concept**.

## Required Properties

---

All physical and some business objects in the metadata model have a set of required properties. These properties are set automatically on creation of the object, and are not removable (although you can change their value). The purpose for required properties is disallow users from getting into a predicament where they have removed a property that is integral to the SQL generation process. For example, if a physical table did not have a Target Table property set, the SQL generator will cause errors, because it cannot access the correct physical table to query.

Using the previous example, when you set a parent concept on the physical table, and the parent concept has a Target Table property, the physical table does not recognize the parent concept's value for Target Table. This is because the physical table already has a Target Table property as part of it's self concept, and the self concept always overrides the parent concept. And since you cannot remove a required property, the parent concept's Target Table will never be recognized at the physical level.

So how do you override a physical object's required property? You set a parent concept at the business model or business view level. Since the inherited properties are override-able, the parent concept at the business level wins.

All physical metadata objects, the business categories and the business model have required properties. You can see what's required for each by referring to [Required Properties per Business Object](#) on page 45.

## Applying Concepts and Properties in the Pentaho Metadata Editor

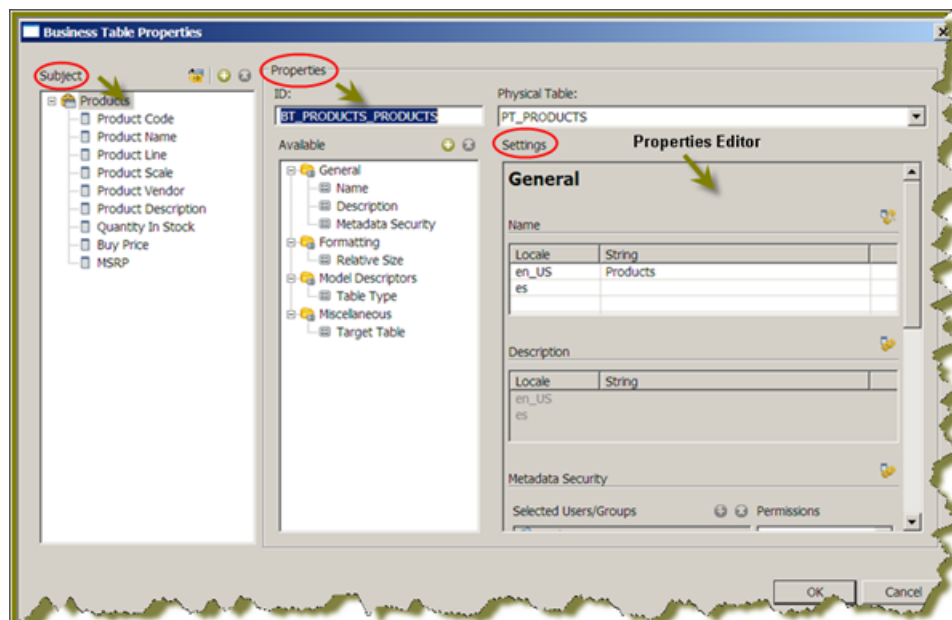
---

As explained previously, there are three levels of concepts for the business objects in the business model, which provides great power and flexibility in how you define your metadata. Since one of the levels is an inherited level, there are really only two concept application processes you need to learn to work with — applying **self concepts**, and applying **parent concepts**.





## Applying Self Concepts: Managing Properties on the Model

You may have noticed when you were creating your tables and columns, that the properties dialog box for every business object has two additional lists to the right of the **Subject** list. These lists are related to **concept editing**. The two additional lists are the **Properties** list, and the **Property Editor** (Settings), and they operate in the same way regardless of the object to which you are applying metadata.



## The Properties List

The middle panel in the **Properties** dialog box is the **Properties** list. This is the list of currently applied metadata properties for the object that is selected in the **Subject** list. The properties are sorted into their appropriate categories

for organization. At the top of the Properties list, there are two icons. The  (add icon) is for adding new metadata properties to this business table's self concept; the  (remove icon) is for removing properties from the self concept. Note that the remove button is disabled until a property is selected in the Property list.

### Helpful Icon Color Coding

There are four ways a property can show up in the Property List:

1. The property is a default property for the associated business object, and thus cannot be removed.
2. The property was inherited from the business object's physical ancestor.
3. The property has been set as part of the self concept.
4. The property has been set as part of the object's parent concept.

It may be difficult to set properties without understanding the override hierarchy, so to help make the hierarchy a bit more apparent in the Properties list, the color code alerts you to what concept level is in play.


- **Yellow Icon** = This property is inherited from the object's physical ancestor.
- **Blue Icon** = Pentaho set the property on the object's own concept. This could be overriding an inherited property.
- **Orange Icon** = This property is set as a result of the parent concept applied to the object.
- **Purple Icon** = This is a special icon reserved for security properties.

## The Properties Editor (Settings)

The **Properties Editor** (Settings) shows the property name and the associated value for that property in a scrolling list. The properties are displayed in the Properties Editor in the same order as they appear in the Properties list. If you click

on a property in the Property List, the Property Editor scrolls to locate the property in the editor. In the Properties Editor, you can set or modify the values of the properties applied to the business object.


### Override

There are icons in the upper right corner next to any property whose value can be overridden. If you want to modify a property by overriding its inherited value, you must first click  (the override button). To cancel an override, click the override button again.

## Adding New Properties

---


Before you add a new property to a business object, make sure that the object you want to apply the new property to is selected in the **Subject** list.

1. In the Properties dialog box, click  (Add) next to **Available**.  
The **Add New Property** dialog box appears. You are prompted with a list of *property choices* to apply.
2. Select a property.
3. Click **OK**.  
The property is now available for you to modify in the Property Editor.

## Removing a Property

---

Follow the instructions below to remove a property:

1. In the Properties list, select the property you want to remove.
2. Click  (Remove)



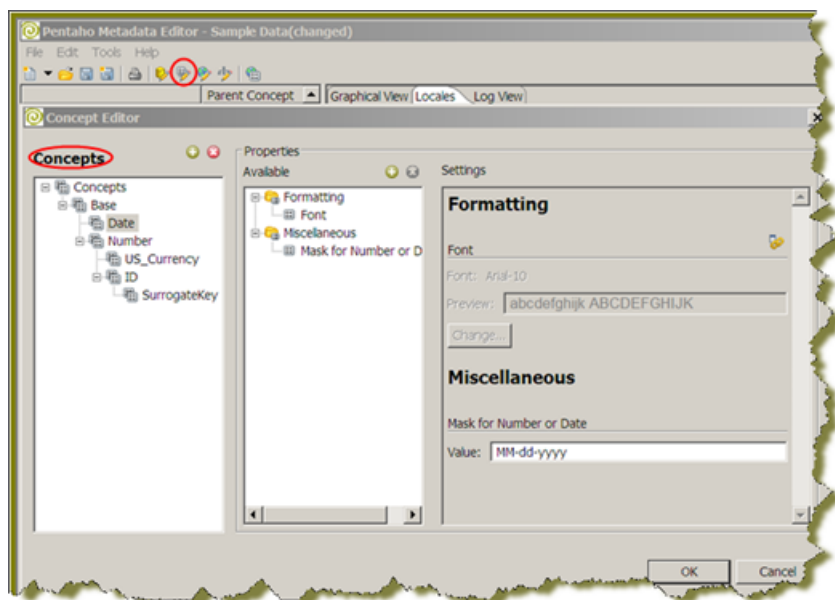
**Note:** If you want to remove inherited or parent concept properties, you must edit the inherited business object or the parent concept.

## The Concept Editor

---

Parent concepts are independent hierarchies of concepts that can be assigned to one or more business objects through the navigation tree. Before you can assign a parent concept you must first learn how to create one.

The **Concept** editor allows you to build concepts to be used as parent concepts. The ability to isolate the concepts, name them, then associate the named concept with one or more business objects gives you flexibility and good concept management. Another feature of the Concept editor is that you can define concepts that build upon other concepts. By nesting concepts this way, you minimize the number of properties you must repeatedly define and create a good inheritance hierarchy.



### The Default Concept

The default Concept is the default set of properties assigned to objects, mainly used for formatting. For example, the mask for all currencies could be set to \$#,##0.00;(\$#,##0.00) by default. Every model contains a default concept called **Base**. This concept is applied as the parent concept to all physical columns created under a connection. The default concept includes metadata properties that are commonly defined and sets a base line set of metadata for all physical columns in the metadata model. The purpose of the default concept is to provide legitimate values for common metadata properties that may not be set in other places in your hierarchy. You can update, add, and delete metadata properties on the default concept. You can also remove the default concept as the parent concept for any and all physical columns. You cannot, however, delete the default concept from the list of concepts in the Concept editor.

## Building Concepts

The set of instructions below is an example of how to build a concept.

1. On the main page of the Pentaho Metadata Editor toolbar, click (Concept Editor).
2. With the **Base** concept selected in the **Concepts** list, click (Add nested concept. Name this concept, **Number**).
3. Make sure **Number** is selected in the **Concepts** list. Note that the properties from Base has been inherited by Number.
4. Add a **Mask** property to **Number**, with a value of **####,##0.00**.
5. With **NUMBER** selected in the **Concept** list, add another nested concept called **ID**.
6. Select **ID** in the **Concepts** list, find the inherited **Mask** property (under **Settings**).
7. Click (Override) and enter an override value of **0** (zero) for the mask.

**Note:** You now have three defined concepts that can be used anywhere in your business model. Each concept serves a different type of business object; **Base** for default, generic columns, **Number** for those columns you know contain numeric financial data, and **ID** for ID columns.

8. Click **OK** to save the concepts.

## Applying Concepts to Business Objects

Now that you have some defined concepts, you can designate some of them as parent concepts in our business model.

1. Select any physical table in the navigation tree. In this sample walkthrough exercises the **Customers** physical table is used.
2. Expand the columns under the table in navigation tree. Apply the **ID** concept to the column **Customernumber**.
3. Right-click (or <CTRL> + click) on the column, and select **Set Parent Concept** from the menu.

4. Choose the **ID** concept and click **OK**.

You should see the concept in the right hand column of the navigation tree on all objects to which it was applied.

## Removing the Parent Concept

---

To remove a parent concept from an object in the business model, you clear the parent concept. 1. 2. 3. This removes the parent concept and its metadata properties from the object.

1. Select any physical table in the navigation tree.
2. Expand the columns under the table in navigation tree and select the column that contains the concept you want to remove.
3. Right-click (or <CTRL> + Click) on the column and select **Clear Parent Concept** from the menu.  
The parent concept and its metadata properties are removed from the object.

# Metadata Security

---

Most of the security configuration explained thus far has involved using features that are built into the User Console or involve configuration changes of some kind. However, if you need to restrict access to certain portions of a metadata model that you are using as a data source, the only way to do it is to edit the model with Metadata Editor and add restrictions. The Pentaho metadata model offers table-, column-, and row-level authorization control, so if you need to prevent certain users or roles from accessing it, you can change (and republish) the model accordingly.

## Configuring the Security Service

---

You must know the base URL for the Pentaho BA Server (the default URL is **http://localhost:8080/pentaho** as well as the name of the service to execute security information retrieval (the service is **ServiceAction**).

The Pentaho Metadata Editor must be configured to connect to your BA Server so that it can retrieve usernames, roles, and access control lists. Follow the below directions to set up Metadata Editor.

1. Start the Pentaho Metadata Editor.

```
sh /pentaho/design-tools/metadata-editor/metaeditor.sh
```

2. Go to the **Tools** menu, then select **Security....**

The Security Service dialogue will appear.

3. In the **Service URL** field, type in the base URL for the BA Server plus the security service.

```
http://localhost:8080/pentaho/ServiceAction
```

4. Next, select the level of detailed security information you want: **All**, **Users** or **Roles**.

If you have hundreds of users in your system, you probably only want to return the roles, and use roles for security information properties. The access control lists are returned with all three options.

5. In the **Username** and **Password** fields, type in the login credentials for an administrator-level account.

6. Click **Test**.

A popup window with the returned XML should appear. If it does not, check that the information you typed into the fields mentioned above is correct.

## Adding Column-Level Security Constraints

---

You need to have established a connection to a data source in Metadata Editor and selected one or more tables to create metadata for.

Follow the below instructions to add user- or role-based restrictions to your data source.

1. In the left pane, right-click the table or column you want to modify, then click **Edit..** from the context menu.

The **Physical Table Properties** dialogue will appear.

2. Click the green **+** icon above the **Available** field in the middle of the screen.

The **Add New Property** dialogue will appear.

3. Select **Metadata Security**, then click **OK**.

4. Click the new **Metadata Security** item in the **General** category.

5. Click the green **+** icon next to the **Selected Users/Groups** field in the right pane.

A list of users and/or roles (depending on what you selected when configuring the security service earlier) will appear.

6. Click the user or role in the left pane that you want to assign permissions to, then click the right arrow button in the middle of the window.

The user or role will move from the **Available** list on the left to the **Assigned** list on the right.

7. Click the checkboxes for the permissions that you want to assign to the selected user or role.

8. Repeat this process for other users or roles you want to assign metadata permissions to, then click **OK**.

9. Change any other relevant metadata options, then click **OK** to return to the Metadata Editor main window.

10. When you are finished, save the metadata configuration as a domain using the **Save As** button, then publish it to the BA Server as an XML schema by selecting **Publish** from the **File** menu.

## Adding Global Row-Level Security Constraints

You need to have established a connection to a data source in Metadata Editor and selected one or more tables to create metadata for.

A global constraint is an MQL Formula statement that institutes a global restriction on the data you specify, down to the row level. Follow the below instructions to add custom global user- or role-based restrictions to your data source.

1. In the left pane, right-click the table or column you want to modify, then click **Edit..** from the context menu.  
The **Physical Table Properties** dialogue will appear.
2. Click the green **+** icon above the **Available** field in the middle of the screen.  
The **Add New Property** dialogue will appear.
3. Select **Data Constraints**, then click **OK**.
4. Click the new **Data Constraints** item in the **General** category.
5. Select the **Global Constraint** option in the right pane.
6. Type in your constraint in the provided text box.
7. Change any other relevant metadata options, then click **OK** to return to the Metadata Editor main window.
8. When you are finished, save the metadata configuration as a domain using the **Save As** button, then publish it to the BA Server as an XML schema by selecting **Publish** from the **File** menu.

## MQL Formula Syntax For Global Constraints

You can use all of the standard operators, and any of the following functions when defining a global constraint:

Function Name	Parameters	Description
OR	Two or more boolean expressions	Returns <b>true</b> if one or more parameters are true
AND	Two or more boolean expressions	Returns <b>true</b> if all parameters are true
LIKE	Two	Compares a column to a regular expression, using % as a wild card
IN	Two or more	Checks to see if the first parameter is in the following list of parameters
NOW	N/A	The current date
DATE	Three numeric parameters: Year, month, and day	The specified date
DATEVALUE	One text parameter: year-month-day	The specified date
CASE	Two or more	Evaluates the odd-numbered parameters, and returns the even numbered parameter values. If there are an odd number of parameters, the last parameter is returned if no other parameter evaluates to true.
COALESCE	One or more	Returns the first non-null parameter. If all are null, the message in the last parameter is returned.
DATEMATH	One expression	Returns a date value based on a DATEMATH expression (see related links below for a link to the DATEMATH Javadoc)

**OR**

```
OR( [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMENAME] = "EuroCars";
    [BT_CUSTOMERS.BC_CUSTOMERS_CREDITLIMIT] > 1000 )
```

**AND**

```
AND( [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMENAME] = "EuroCars";
      [BT_CUSTOMERS.BC_CUSTOMERS_CREDITLIMIT] > 1000 )
```

**LIKE**

```
LIKE( [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMENAME]; "%SMITH%")
```

**IN**

```
IN( [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMENAME]; "Adam Smith"; "Brian
      Jones")
```

**NOW**

```
NOW( )
```

**DATE**

```
DATE(2008;4;15)
```

**DATEVALUE**

```
DATEVALUE( "2008-04-15" )
```

**CASE**

```
CASE( [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMENAME] = "EuroCars"; "European
      Cars";
      [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMENAME] = "AsiaCars";
      "Asian Cars"; "Unknown Cars"
      )
```

**COALESCE**

```
COALESCE( [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMENAME];
           [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMERID]; "Customer is
      Null" )
```

**DATEMATH**

```
DATEMATH( "0:ME -1:DS" )
```

This expression represents 00:00:00.000 on the day before the last day of the current month.

## Adding User or Role Row-Level Security Constraints

You need to have established a connection to a data source in Metadata Editor and selected one or more tables to create metadata for.

A role-based constraint is an MQL Formula statement that restricts access (on the row level) only to certain users or roles. Follow the below instructions to add fine-grained user- or role-based restrictions to your data source.

1. In the left pane, right-click the table or column you want to modify, then click **Edit..** from the context menu.  
The **Physical Table Properties** dialogue will appear.
2. Click the green **+** icon above the **Available** field in the middle of the screen.  
The **Add New Property** dialogue will appear.
3. Select **Data Constraints**, then click **OK**.
4. Click the new **Data Constraints** item in the **General** category.
5. Select **Role-Based Constraints** option in the right pane.
6. Click the green **+** icon next to the **Selected Users/Groups** field in the right pane.  
A list of users and/or roles (depending on what you selected when configuring the security service earlier) will appear.
7. Click the user or role in the left pane that you want to assign permissions to, then click the right arrow button in the middle of the window.  
The user or role will move from the **Available** list on the left to the **Assigned** list on the right.
8. Click the checkboxes for the permissions that you want to assign to the selected user or role.
9. Repeat this process for other users or roles you want to assign metadata permissions to, then click **OK**.
10. Change any other relevant metadata options, then click **OK** to return to the Metadata Editor main window.
11. When you are finished, save the metadata configuration as a domain using the **Save As** button, then publish it to the BA Server as an XML schema by selecting **Publish** from the **File** menu.

## MQL Formula Syntax For User and Role Row-Level Constraints

The MQL Formula syntax for defining a user or role row constraint is:

```
[table.column] = "row"
```

The table and column are defined as part of a metadata business model. Here is an example that isolates access to data from the Sales department:

```
[BT_OFFICE.BC_DEPARTMENT]="Sales"
```

It's also possible to give or deny access to an entire role, or a single user, by selecting that user or role, then using a boolean for a constraint:

```
TRUE( )
```

Or:

```
FALSE( )
```



## Removing Security from Metadata Domain Repository

---

The BA Server and console must be stopped before executing these instructions.

This procedure changes your default metadata domain repository so that it is no longer security-aware. It is a necessary step in completely removing security from the BA Platform; however, this procedure does not, in itself, remove **all** security. To do that, start with [Removing Security](#).

1. Edit the `/pentaho-solutions/system/pentahoObjects.spring.xml` file.
2. Comment out the **IMetadataDomainRepository** line, and uncomment the similar line below it.

Alternatively, you can switch the value of **IMetadataDomainRepository** from **org.pentaho.platform.plugin.services.metadata.SecurityAwareMetadataDomainRepository** to **org.pentaho.platform.plugin.services.metadata.MetadataDomainRepository**.

```
<!-- <bean id="IMetadataDomainRepository"
      class="org.pentaho.platform.plugin.services.metadata.
        SecurityAwareMetadataDomainRepository"
      scope="singleton"/> -->
<!-- Use this schema factory to disable PMD security -->
<bean id="IMetadataDomainRepository"
      class="org.pentaho.platform.plugin.services.metadata.MetadataDomainRepository"
      scope="singleton"/>
```

3. Save and close the file.

You've now switched the metadata domain repository to one that is not security-aware. Access controls will no longer be enforced on various metadata objects.

# Adding Security to Metadata Business Objects

Pentaho metadata provides a **Security Information** property that allows you to define table or column level security used by the Pentaho BA Server. Before you can use this property, you must define settings in the Pentaho Metadata Editor that allow the Pentaho Metadata Editor to communicate with the BA Server. The Pentaho Metadata Editor must be able to retrieve the list of users, roles and access control lists (ACLs) needed.

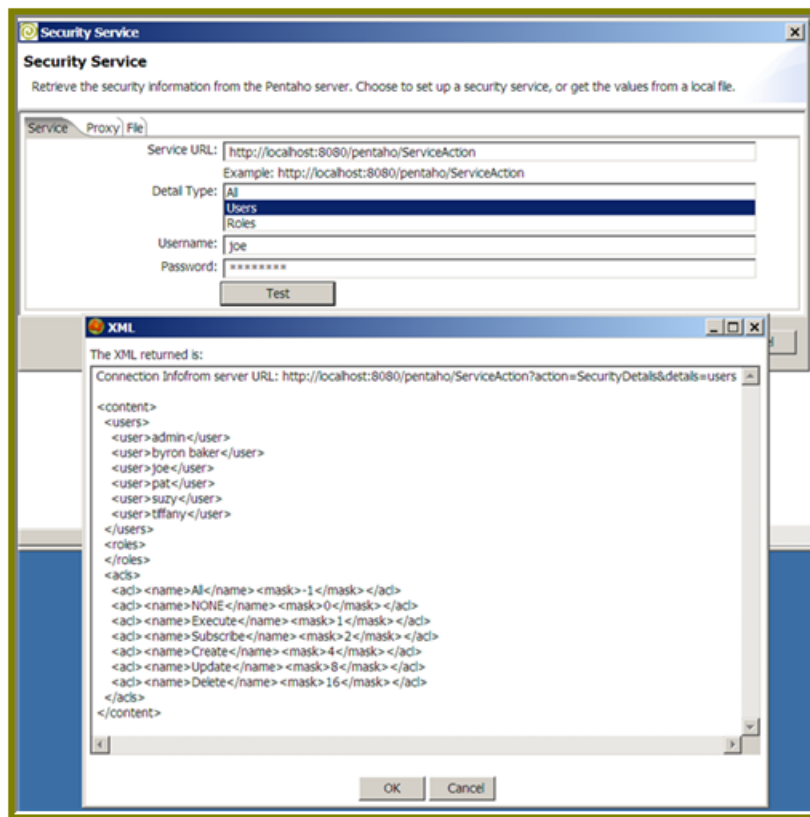
Because the security information used in the business model must be retrieved from a Pentaho BA Server, you must have the following information available before you configure security:

- The base URL of the Pentaho BA Server and the name of the service that executes security information retrieval. In the demo server, the base URL is **http://localhost:18080/pentaho**, (or **http://localhost:8080/pentaho**), and the default name for the service is **ServiceAction**.
- The logon credentials for the BA Server.

## Defining Security Settings

Follow the instructions below to define your security settings:

1. In the Pentaho Metadata Editor main window, go to **Tools -> Security**. The **Security Service** dialog box appears.
2. Type the correct URL in the **Service URL** text box.
3. Select **Users** from the **Detail Type** list.
4. Type **admin** in the **User Name** text field. This is the default admin user for the Pentaho User Console.
5. Type **password** in the **Password** text field. This password is associated with the default admin user.
6. Click **Test**.



If the information you entered is correct, a listing of users and roles appear.

7. Click **OK** to exit.


## Getting Security Settings Offline

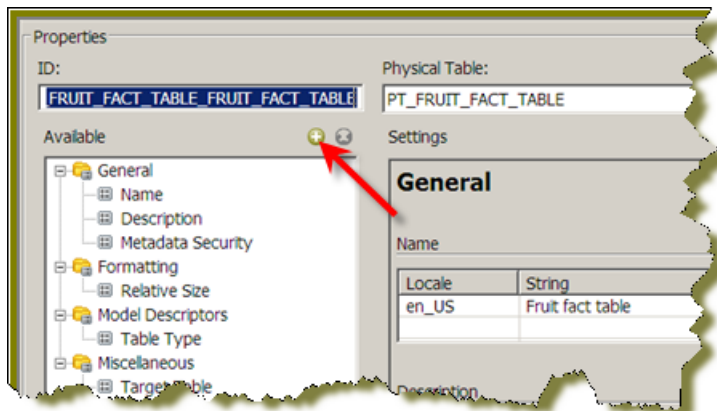
In instances when you want to work on your model and may not have access to the Pentaho BA Server, you can save your security information in a file. The Pentaho Metadata Editor will retrieve your settings from the file instead of accessing the server every time you open your domain.

1. Follow the instructions in .
2. After you click **Test**, all of the XML between the <content></content> tags, including content the tags themselves.
3. Paste the XML code into your favorite text editor, and save the file as **metadata\_security.xml**, in a location of your choice.
4. Click the **File** tab in the **Security Service** dialog box.
5. Browse to the file that you just saved.
6. Click **OK** to exit the dialog box.

## Changing Security Constraints

Follow the instructions below to change security constraints on a specific business table or column.

1. Open the **Properties** dialog box that contains the table or column you want to change.
2. Click  (Add Property).




The **Add New Property** dialog box appears.

3. Select the **Security Information** property and click **OK**.  
The Properties Editor (Settings) scrolls down to the **Metadata Security** section.
4. Add the individual role or user permissions to the business model, table, or column.  
These permissions be enforced in the Pentaho BA Platform after publishing the new metadata model.

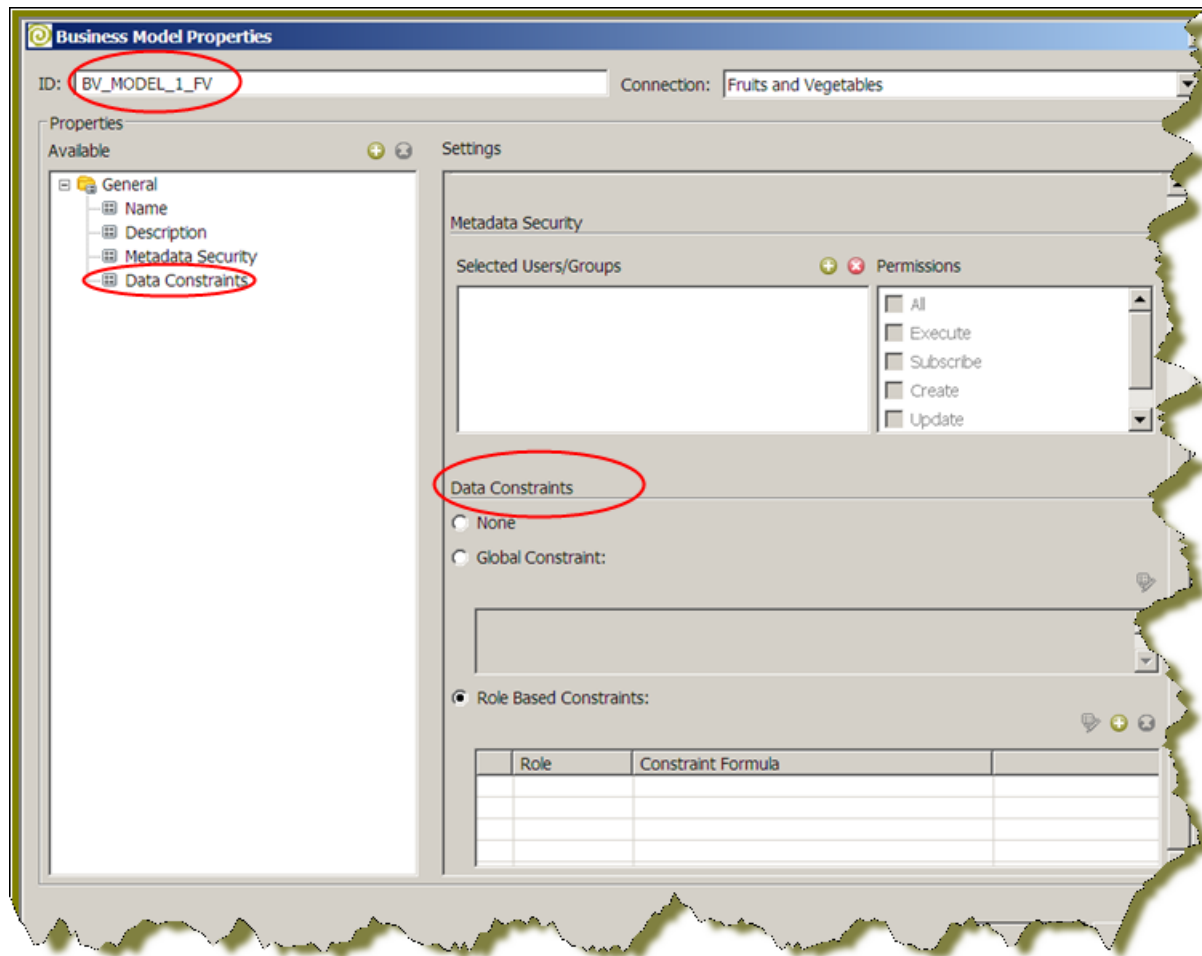
# Adding Row Level Security to a Pentaho Metadata Model

Row Level Security allows you to control the results that are returned in a query based on a user's security level. You can specify which rows of data each User Role or User ID is allowed to retrieve from the database, based on a column of data or combination of columns of data.

 **Note:** The only location that this will actually produce a result is on the Business Model Level.

## Applying Data Constraints

In the Pentaho Metadata Editor, select the model to which you want to add Row Level Security, right click on the Model, and select **Edit**.



Row Level Security is only in effect at the Model level. Any data constraints defined below the Model Level, such as in a Business Table or Business Column, is ignored and not used. In the Model Properties dialog box, select the **General** -> **Data Constraints**.

### Global Constraint

If you are using the **Global Constraint**, a single MQL Formula is used to define security for all users. In addition to the standard MQL Functions available, there are also two additional functions:

- **USER()** - returns the name of the current user
- **ROLES()** - returns a list of roles associated with the current user

The example below defines an MQL formula that allows administrators full access; all other users have no access,

```
IN("Admin"; ROLES())
```

## Role-Based Constraints

If you are using **Role-Based Constraints**, the Metadata engine determines which MQL constraints are appropriate for the current user and applies them to the current query. Constraints may be added for each Role and User in a system. If zero constraints match a user and his or her roles, no data is returned by the MQL query. If more than one constraint applies to a user, the constraints are OR'ed together to determine row visibility.

This example below defines an MQL Formula for three different roles. The Admin role has full row visibility, the Sales and Engineering roles can access data that joins to rows associated with their specific department only.

Role	Constraint
Admin	TRUE()
Sales	[BC_DEPARTMENT]="Sales"
Engineering	[BC_DEPARTMENT]="Engineering"



**Important:** Row Level Security Constraints are applied at the MQL Layer. The Business Columns referenced in the MQL Security Constraints will be resolved down to SQL Table Columns. The Tables which contain column references included in security constraints will be joined to your query, based on the relationships defined in the Business Model. It is recommended that you do not use outer joined business columns for the purposes of security constraints.

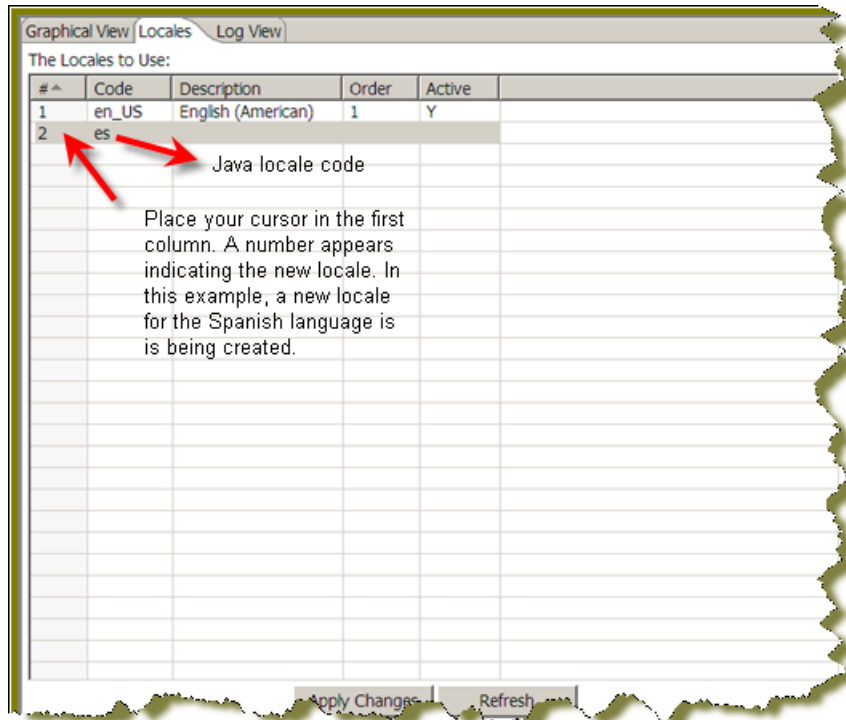
# Configuring and Managing International Locales

Pentaho Metadata Editor supports multiple locale entries for any text or string-based metadata property. But first you must specify the locales that you want to include in your model. To do this, use the Pentaho Metadata Editor **Locales Editor**.

## Setting Up Locales

Follow the instructions below to set up locales:

1. In the Pentaho Metadata Editor, go to **Tools -> Locales Editor**.
2. Place your cursor in the first column of the first empty row in the Locales Editor main table



3. In the next column and type the *Java Locale code* of the language you want to add.
4. Specify the order you want this locale to be used. In the example screen capture, **English (American)** is designated as the first language (Order = 1).
5. Type a **Y** in the last column to activate the locale, or an **N** to de-activate it.
6. Click **Apply Changes** when you are done.

# Importing and Exporting Domains

---

When you save a domain, the domain is stored in a metadata repository. The Pentaho BA server does not use the metadata repository, instead it accesses an XML file exported from the Pentaho Metadata Editor. Exporting your domain is a good way to ensure safe backups of your domains.

## Importing a Domain

---

1. In the Pentaho Metadata Editor, go to **File > Import From XMI File**.



**Note:** You may be prompted to save the currently active model if you have any unsaved changes pending. When you import a new domain, that domain becomes the active domain in the Pentaho Metadata Editor.

2. In the file browser, navigate to your domain file, select it, and click **OK**.
3. In the **Save Model** dialog box, type a name for the domain.



**Important:** If you enter the name of an existing domain, that domain is overwritten by the import.

## Exporting a Domain

---

1. In the Metadata Editor, go to **File > Export to XMI File**.



**Note:** You may be prompted to save the currently active model if you have any unsaved changes pending.

2. Type a file name and select a location to save your file.



**Important:** The default extension for a Metadata domain XML file is `.xmi`.

3. Click **Save**.

Once you have entered a name for your export file, the domain is exported to that file. You can inspect the export file using a text editor if you are curious about the underlying XML code.

# Domain Backup and Recovery

---

Each domain can be saved to the Common Warehouse Metadata (CWM) repository with any name you like. The Save and Save As options are available from both the Pentaho Metadata Editor File menu and the toolbar. Each time a domain is saved to the repository, a recovery export file of the domain is saved to the file system, under the .pentaho-meta directory. This directory is typically located in the your home directory. The recovery file contains the last successfully saved state of the domain. The files are named **recovery\_[studio:domainname].xmi**.

You can restore a domain to the last saved state by *importing the recovery file* importing the recovery file from the file system. Domain restoration may be necessary because you want to revert to the last known good state of your domain in the event of repository errors or corruption.



# Publishing a Domain to a Pentaho BA Server

---

You can share an XML representation of your domain with a Pentaho solution that your BA server recognizes, so that the server can access the metadata domain and its contents. The implementation of metadata in the Pentaho BA Server requires that you follow some basic rules.

- There can only be one metadata domain per Pentaho solution.
- The metadata domain is associated with the solution by placing the XML format of the domain in the root directory of the solution.
- The domain XML file must be `[myBusinessModel].xmi`, where `[myBusinessModel]` is any name you want to give the model.

You can accomplish this by using the Pentaho Metadata Editor **Publish** feature, or you can [export your domain file](#) and copy it manually to the solution repository folder associated with the solution.

Once you have published a model, it become available as a public data source, unless you add [security](#) to the metadata business objects. Published models appear as data sources for these Pentaho design tools and components.

- Report Designer
- Interactive Reports
- Dashboards
- Pentaho Agile BI using the Report Wizard
- Pentaho Data Integration

## Before you Publish Your Domain

---

Before you publish a domain make sure you have this information available.

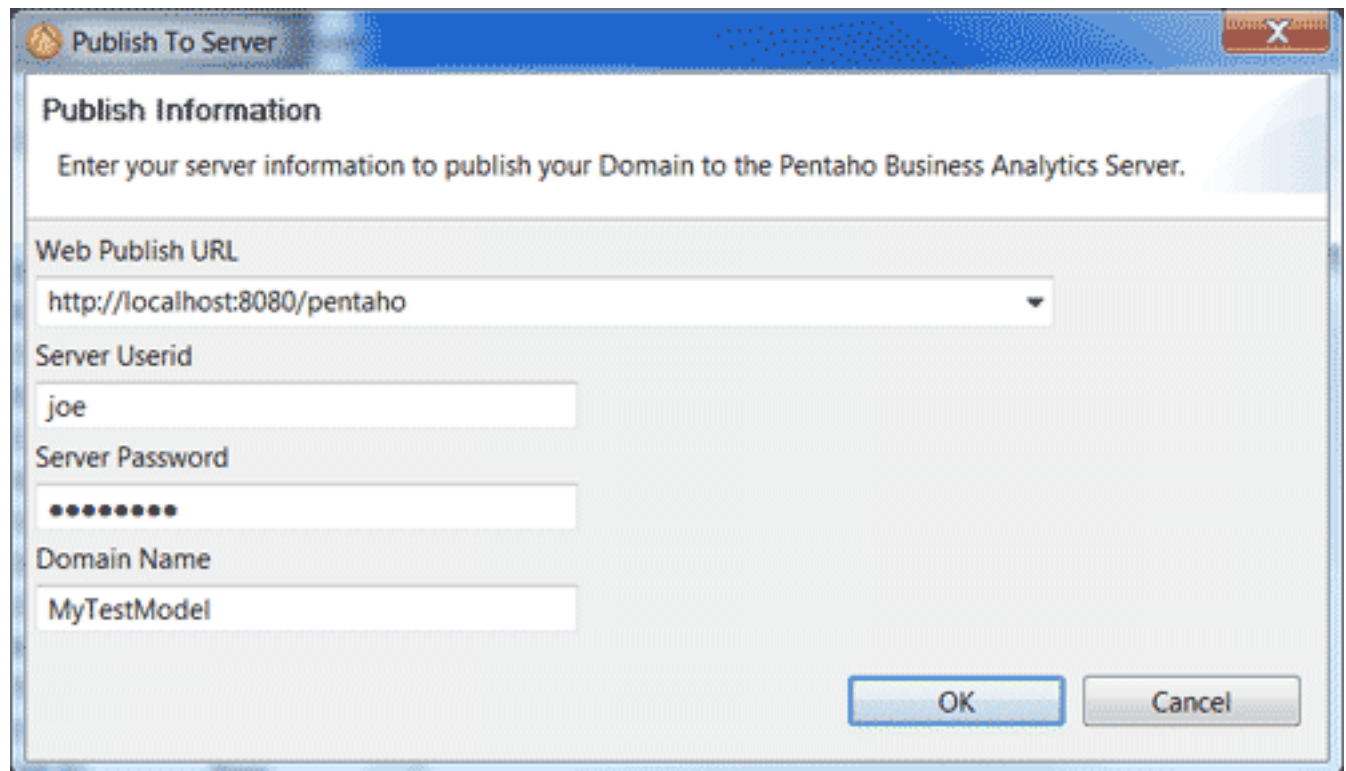
- The user name and password for the Pentaho BA Server. The default user name is **admin**.
- The base URL for your Pentaho BA Server. This consists of all URL information up to and including the Web application context. For example, the base URL of the Pentaho demo server is **http://localhost:8080/pentaho**. For publishing, you must append the base URL the to name of the publish service. The default is **RepositoryFilePublisher**. This is the value you must use, unless your server administrator has changed the service name.
- The solution to which you want to publish the domain. For example, the Pentaho demo server ships with two different solutions, the **Samples** solution and the **Steel Wheels** solution. No slashes are required.

## Making a Model Available as a Data Source

---

After you have created a domain with atleast one model using the Metadata Editor, you can make it appear in the data source lists for the User Console and client tools by publishing the model to the BA Server. Save the model before you try to publish it.

1. In the Pentaho Metadata Editor go to **File > Publish to Server**.  
The **Publish to Server** dialog box appears.



**Publish To Server**

**Publish Information**

Enter your server information to publish your Domain to the Pentaho Business Analytics Server.

Web Publish URL  
http://localhost:8080/pentaho

Server Userid  
joe

Server Password  
••••••••

Domain Name  
MyTestModel

OK Cancel

2. Enter the **Web Publish URL**—the Pentaho BA Server base URL followed by the publish service name. **RepositoryFilePublisher** is the default.
3. Enter the **Server Userid**—The default is `admin`.
4. Enter the **Server Password**—The default is `password`.
5. Enter the **Domain Name**—the domain name of the solution to which you want to publish.
6. Click **OK** to save and exit the dialog box.  
The model is saved to the BA Server in the solution repository as an `.xml` file. If this file already exists, you are prompted to overwrite it.

# Metadata Properties Reference

This section contains the Metadata Properties Reference.

## Out-of-the-Box Properties

The table below shows the properties that are provided with Pentaho Metadata Editor.

\*Localized Properties

ID	Description	Category	Values
Name*	This property describes the display name for the business object	General	Alphanumeric
Description*	A descriptive text entry describing the business object	General	Alphanumeric
Comments*	Additional comments regarding the business object.	General	Alphanumeric
Security Information	Security rules for granting/restricting access to the business object	General	Determined by security widget; see Adding Security
Font	The font properties to apply to this business object	Formatting	Determined by selections in the font dialog box
Color of Text	The foreground or text color for the business object	Formatting	Determined by selections in the color dialog box
Text Alignment	The text alignment for the business object	Formatting	Left   Right   Centered   Justified
Background color	The background color for the business object	Formatting	Determined by selections in the color dialog box
Relative Size	This property is normally associated with business tables and is used to calculate join paths. The sum of all table relative sizes in a path are calculated when deciding on a multi-table join and the multi-table join with the smallest summed value is used for the join path.	Formatting	Numeric
Aggregation Rule	Determines the method of aggregating the data from this business object	Model Descriptors	None   Sum   Count   Distinct Count   Minimum   Maximum
Data Type	Data type for this business object	Model Descriptors	Unknown   String   Date   Boolean   Numeric   Binary   Image   URL + Length and Precision (Integers)
Field Type	The type or relationship purpose this field serves	Model Descriptors	Other   Dimension   Fact   Key   Attribute

ID	Description	Category	Values
Table Type	Table type is used to automatically determine relationship types with other tables. For instance, if a Fact table is joined with a Dimension table, this is normally an N to 1 relationship	Model Descriptors	Other   Dimension   Fact
Formula	This property allows you to create a calculation defining the business object. See <a href="#">Pentaho Metadata Formulas</a> for more information.	Calculation	Alphanumeric
Is the Formula Exact?	Determines if the formula must be parsed as a Metadata Formula. If false, this field is treated as a database column name .	Calculation	Boolean
Column Width	The width of the column as represented for display	Miscellaneous	Pixels   Percent of Page Width   Inches   Centimeters   Points ) + Integer
Hidden For the User?	Hides the object from being displayed in the Business View of the model	Miscellaneous	Boolean
Mask for Number or Date	The format mask to use when the data for this object is displayed. For dates, the format follows the date/ time patterns for a Java <a href="#">SimpleDateFormat</a> format mask. For numbers, the pattern should follow the Java <a href="#">DecimalFormat</a> format mask.	Miscellaneous	Alphanumeric
Target Schema	Defines the database schema to use when querying the business object	Miscellaneous	Alphanumeric
Target Table	Defines the physical database table from which the business object is defined	Miscellaneous	Alphanumeric

## Custom Properties

You can define any number of custom properties. You must give your property an ID to set its type.





- String
- Date
- Numeric Value
- Color
- Font

- Type of Field
- Type of Aggregation
- Boolean
- Field Data Type
- Localized String
- Type of Table
- URL
- Security
- Text Alignmen
- Column Width

## Required Properties per Business Object

The table below contains the required properties for given business objects. Required properties cannot be deleted.

Required =  Not Required = 

ID	Physical Table	Physical Column	Business Category	Business Model
Name				
Description				
Security Information				
Table Type				
Relative Size				
Formula				
Field Type				
Data Type				
Aggregation Rule				
Is the Formula Exact?				
Hidden for the User?				
Font				
Mask for Number or Date				
Color of Text				
Color of Background				

# Pentaho Metadata Formulas

Formulas have multiple uses in Pentaho Metadata. The first use of formulas in Pentaho Metadata is in the constraint definition of a Metadata Query, also known as MQL. A constraint function references business table columns and uses various comparison operators to determine which subset of data the business user is interested in.

The second use is in the definition of Physical Table Columns. In addition to Physical table columns mapping directly to a database table column, physical table columns defined in Pentaho Metadata may also be defined as a formula. This allows for combining of multiple columns into a single column, and also for doing more advanced aggregate calculations within aggregate table definitions.

The third use is in the definition of complex joins within business model relationships. This allows for multiple key joins as well as other logic when joining tables.

The fourth use is row level security. Under the covers, Pentaho Metadata uses JFreeReport's libFormula package for interpreting formulas. The goal is to support OpenFormula syntax in the Metadata environment. Formulas are first interpreted by libFormula, and then in the Metadata system are converted to native SQL depending on the type of database used.

## First Use — MQL Constraints

Below is an example of an MQL Constraint formula:

```
OR ( [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMERNAME] = "EuroCars";
      ( ([BT_CUSTOMERS.BC_CUSTOMERS_CREDITLIMIT] * 2) / 3 > 1000 ) )
```

Note the **OR** function. This is a boolean function that has two parameters, separated by semicolons. These parameters are boolean expressions. The first boolean expression references a business column from the Metadata model. All references appear with brackets around them []. This reference refers to the business table, and then to the business column. This boolean expression does some arithmetic and checks to see if the final value is larger than 1000. In the second expression, business column **BT\_CUSTOMERS.BC\_CUSTOMERS\_CUSTOMERNAME** is compared to **EuroCars**. Notice that double quotes are used when referring to text; double quotes are required.

## Second Use — Physical Table Column Formulas

Below is an example of a Physical Table Column Formula:

```
[QUANTITYORDERED] * [PRICEEACH]
```

The references in this example specifically refer to the database column, not derived the physical column definitions. All operators and functions may be used in the definition of the physical table column. For this formula to be recognized, the **isExact** property of the physical table column must be set to **true**. The referenced physical column must be explicitly defined in the metadata model.



**Note:** In earlier versions of Pentaho Metadata Editor, (prior to the 3.5 release), aggregation functions had to be specified explicitly and the aggregation rule had to be selected. This is no longer necessary; the query that is generated will use the selected aggregation rule during execution.

## Multi-table expressions — Formulas can use any business column in the model

Since the latest versions (after 2008/03/14) it is possible to define formulas that use business columns from anywhere in the business model. For example suppose there are two business tables:

- Orders (fact table), ID=BT\_ORDER\_FACT
- Product (dimension), ID=BT\_PRODUCT

Suppose you want to calculate the turnover based on:

- Orders (fact table), ID=BT\_ORDER\_FACT
- Product (dimension), ID=BT\_PRODUCT
- The number of products sold, from the Orders table, ID=BC\_FACT\_ORDER\_NRPRODUCTS
- The price of the product, from the Product table, ID=BC\_DIM\_PRODUCT\_PRICE

To get there, you must define a new business column, say in the Orders business table (although you could take Product too):

- Table: Orders (BT\_ORDER\_FACT)
- ID = BC\_FACT\_ORDER\_TURNOVER
- Name = Turnover
- Formula = [BT\_ORDER\_FACT.BC\_FACT\_ORDER\_NRPRODUCTS]
- Exact = Yes
- Aggregation Rule = SUM

The SQL generator is now going to replace the business columns by their respective SQL variants. As such, you must make sure that the business columns are resolving correctly. In this specific case, this means you want the two columns to be non-aggregated. If you now select the single business column BT\_FACT\_ORDER\_TURNOVER, below is the SQL that is generated:

```
SELECT
    SUM( BT_ORDER_FACT.NRPRODUCTS * BT_PRODUCT.PRICE ) AS COL0
FROM
    FACT_ORDER BT_ORDER_FACT ,DIM_PRODUCT BT_PRODUCT
WHERE
    ( BT_ORDER_FACT.PRODUCT_TK =
      BT_PRODUCT.PRODUCT_TK )
```

Now, suppose you want to generate the multiplication of the two sums (different use-case). You define the formula as "[BT\_ORDER\_FACT.BC\_FACT\_ORDER\_NRPRODUCTS] \* [BT\_PRODUCT.BC\_DIM\_PRODUCT\_PRICE]" (without the SUM) and specify an aggregation for the two business columns in use. The generated SQL will be as follows:

```
SELECT
    SUM( BT_ORDER_FACT.NRPRODUCTS ) * SUM( BT_PRODUCT.PRICE ) AS COL0
FROM
    FACT_ORDER BT_ORDER_FACT ,DIM_PRODUCT BT_PRODUCT
WHERE
    ( BT_ORDER_FACT.PRODUCT_TK =
      BT_PRODUCT.PRODUCT_TK )
```

It is possible to create two versions of the used business columns, one aggregated (exposed to the users) and one non-aggregated (hidden from the users) for example.

The SQL generator works recursively. That means that it is possible to create a formula that calculates 7% (taxes for example) of the turnover:

- ID = BC\_FACT\_ORDER\_TURNOVER\_TAXES
- Name = Turnover Taxes
- Formula = [BT\_ORDER\_FACT.BC\_FACT\_ORDER\_TURNOVER] \* 7 / 100
- Exact = Yes

If you add that column to the selection, you get one extra column as shown below:

```
( SUM( BT_ORDER_FACT.NRPRODUCTS * BT_PRODUCT.PRICE ) * 7 / 100 ) AS COL1
```

## Formula Syntax

*Function syntax:*

```
FUNCTION_NAME ( PARAM ; PARAM )
```

*Text (requires double quotes):*

```
"TEXT"
```

*Parenthesis are used for formula precedence:*

```
( 1 + 2 ) * 3
```

## Metadata References

*Business Column References:*

```
[<BUSINESS_TABLE_ID> . <BUSINESS_COLUMN_ID> ]
```

*Physical Column References (only used in physical column formula definitions):*

[ <PHYSICAL\_COLUMN\_NAME> ]

## Supported Functions

The table below contains a listing of supported functions. Examples are show below each supported function.

Function Name	Parameters	Description
OR	Two or more boolean expression parameters	Returns true if one or more parameters are true

```
OR( [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMENAME] = "EuroCars";
    [BT_CUSTOMERS.BC_CUSTOMERS_CREDITLIMIT] > 1000 )
```

Function Name	Parameters	Description
AND	Two or more boolean expression parameters	Returns true if all parameters are true

```
AND( [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMENAME] = "EuroCars";
      [BT_CUSTOMERS.BC_CUSTOMERS_CREDITLIMIT] > 1000 )
```

Function Name	Parameters	Description
LIKE	Two parameters	Compares a column to a regular expression, using "%" as wild cards

```
LIKE( [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMENAME]; "%SMITH%" )
```

Function Name	Parameters	Description
IN	Two or more parameters	Checks to determine if the first parameter is in the following list of parameters

```
IN( [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMENAME]; "Adam Smith"; "Brian Jones" )
```

Function Name	Parameters	Description
NOW	none	The current date

```
NOW( )
```



Function Name	Parameters	Description
DATE	Three numeric parameters, year, month, and day	A specified date

```
DATE(2008;4;15)
```

Function Name	Parameters	Description
DATEVALUE	One text parameter "year-month-day"	A specified date

```
DATEVALUE("2008-04-15")
```

Function Name	Parameters	Description
CASE	Two or more parameters	Evaluates the first, third, etc. parameter, and returns the second, fourth, etc parameter value if there are an odd number of parameters, the last parameter is returned if no other parameter evaluates to true. Note that when using this function, the formula must be set on a new column as shown below.

```
CASE( [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMENAME] = "EuroCars"; "European Cars";
      [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMENAME] = "AsiaCars";
      "Asian Cars"; "Unknown Cars"
    )
```

Function Name	Parameters	Description
COALESCE	One or more parameters	Returns the first non null parameter

```
CASE( [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMENAME] = "EuroCars"; "European Cars";
      [BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMENAME] = "AsiaCars";
      "Asian Cars"; "Unknown Cars"
    )
```

Function Name	Parameters	Description
DATEMATH	One or more parameters; <a href="#">DateMath Javadoc for full syntax</a>	Returns a date based on an expression

```
DATEMATH("0:ME -1:DS") - 00:00:00.000 of the day before the last day of the current month
DATEMATH("0:MS 0:WE") - 23:59:59.999 the last day of the first week of the month
DATEMATH("0:ME") - 23:59:59.999 of the last day of the current month
DATEMATH("5:Y") - the current month, day and time 5 years in the future
```

```
DATEMATH("5:YS") - 00:00:00.000 of the first day of the years 5 years in the future
```

## Supported Operators

The table below contains a listing of supported operators.

Operator	Description
=	Returns true if two expressions are equal
>	Returns true if first expression is larger than the second
<	Returns true if first expression is smaller than the second
>=	Returns true if first expression is larger than or equal to the second
<=	Returns true if first expression is smaller than or equal to the second
<>	Returns true if two expressions are not equal
+	Adds two values
-	Subtracts two values
*	Multiplies two values
/	Divides two values

## Supported Aggregate Functions

The table below contains a listing of supported aggregate functions.

Function Name	Description
SUM	Sums a specific columns values determined by grouping
COUNT	Counts a specific columns values determined by grouping
AVG	Averages a specific columns values determined by grouping
MIN	Selects the minimum column value determined by grouping
MAX	Selects the maximum column value determined by grouping

# Troubleshooting

This section contains reported or expected problem descriptions and solutions.

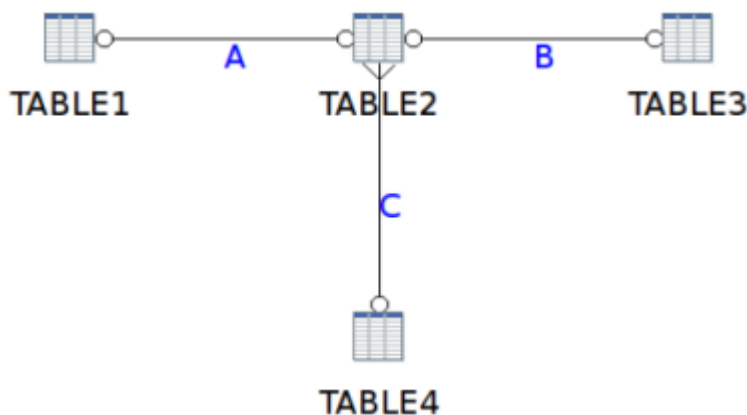
## Publishing

If publishing to the BA Server fails, here are a few things you may want to verify.

- Ensure that the BA Server is running.
- Verify that your security access is defined as the Pentaho administrator role. The default is **admin**.
- Check that you have not used trailing or leading slashes in the Web Publish URL or Domain Name

## Managing Multiple Outer-Joins

When you have three or more tables that require outer joins, the *order* in which the tables are joined is critical. Consider the example below:



In the sample preview below, the entries, 1, 2, 3, and 4, in Table4 are taken and outer-joined with the records in the two other tables. The three other tables contain fewer records. The relationships are defined but now the order of execution critical. Relationship A is executed first, followed by B, then C.

Examine preview data					
Rows of step: The first 5000 rows from the generated query					
#	BC_TABLE1_T1_PK	BC_TABLE2_T2_PK	BC_TABLE3_T3_PK	BC_TABLE4_T4_PK	
1	1	1	1	1	
2	2	2		2	
3				3	
4				4	

Below is the query that is generated:

```

SELECT DISTINCT
    TABLE1.PRIMARYKEY AS COL0
    ,TABLE2.PRIMARYKEY AS COL1
    ,TABLE3.PRIMARYKEY AS COL2
    ,TABLE4.PRIMARYKEY AS COL3

FROM TABLE4 LEFT OUTER JOIN
    (
        TABLE3 FULL OUTER JOIN
        (
            TABLE1 FULL OUTER JOIN TABLE2
            ON ( TABLE1.PRIMARYKEY = TABLE2.FOREIGNKEY )
        )
        ON ( TABLE2.PRIMARYKEY = TABLE3.FOREIGNKEY )
    )
ON ( TABLE2.FOREIGNKEY = TABLE4.PRIMARYKEY )

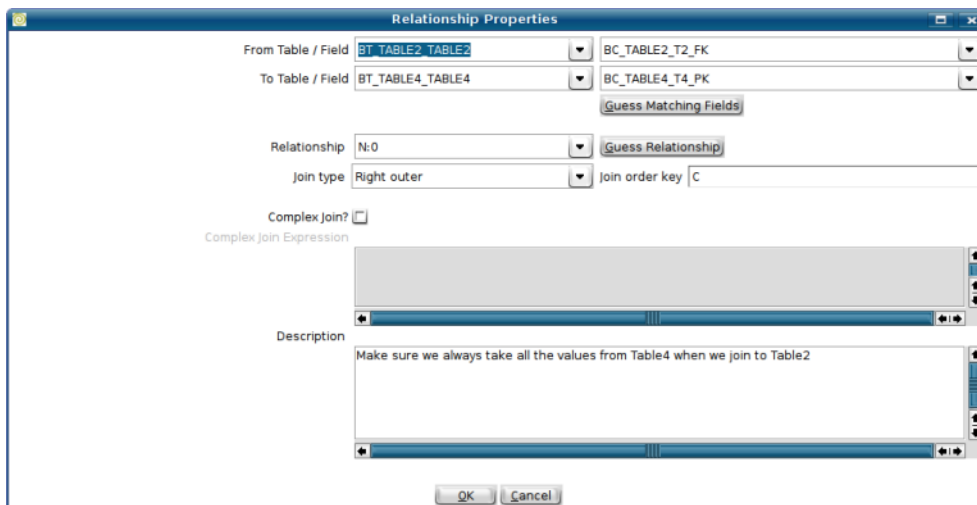
ORDER BY
    TABLE4.PRIMARYKEY

```

The nested join syntax that is generated forces the order of execution:

- Join Table1 and Table2 (shown in red)
- Join Table3 and A = B (shown in blue)
- Join Table4 with B = Result

Other orders of execution are just as valid depending on the business context to which they are applied. Another order of execution will generate a different result. To allow business model designers to ensure that user selections are executed in a specific way, a **Join Order Key** is added to the **Relationship Properties** dialog box.



The join order key is relevant only in instances in which outer joins are deployed in business models. To make the importance of the execution order apparent, this information is displayed in the graphical view of the model, as shown below.



**Note:** It is not mandatory to use uppercase letters, (A, B, C, as shown in the first image), to set the order in which tables are executed. Any alphanumeric characters (0-9, A-Z) can be used. The system will calculate the ASCII values of each character; the values are then used to determine the order of execution. In the example, A, B, C, AA, AB, Pentaho Metadata Editor will execute the table relationships in the following order: A, AA, AB, B, C.

## Using the Delay Outer Join Conditions Property

To force conditions that would ordinarily be processed in the JOIN condition to be processed in the WHERE clause, follow the directions below to create a **delay\_outer\_join\_conditions** custom property.

1. Right-click on a business model and select **Edit**.
2. Add a property by clicking the green **+** icon.
3. Select **Add a Custom Property** and set its ID to **delay\_outer\_join\_conditions** and select **boolean** for the **Type**, then click **OK**.
4. Select the newly-created **delay\_outer\_join\_conditions** property, then click the checkbox for **delay\_outer\_join\_conditions** under the **Custom** heading on the right side of the window, then click **OK**.

Instead of the conditions being rolled into the JOIN clause, they will be allowed to roll down into the WHERE clause.