InfoSphere Data Architect: Best practices for modeling and model management

Chetan Bhatia
Joseph Yeh
Ajith Kumar Punnakula

November 14, 2013

This article explains how to create models with better maintainability and to change the models in a team environment using best practices. The article also details the product settings that you should set for the product to work at optimal level with the given resources.

Introduction
This article provides best practices for using InfoSphere Data Architect to achieve the following goals:

- Protect model integrity at all stages of development
- Manage logical and physical data models more efficiently
- Reduce memory and processor resource consumption
- Use version control systems (such as ClearCase) effectively

Protecting model integrity
InfoSphere Data Architect XML Metadata Interchange (XMI) file structures
InfoSphere Data Architect models use the XMI standard. Even though the model metadata is stored in XML format, models are not simple XML files. Each model contains complex interlinks between the XML nodes, and these links are not understood by tools that are not compatible with InfoSphere Data Architect model files. Therefore, if you want to perform any task that must be performed on the models (like comparing two models), you must use a version control system that is built into or compatible with InfoSphere Data Architect. The workbench is the only tool that you can use to complete data modeling tasks and create valid InfoSphere Data Architect models.

Note: The article "Comparing and merging UML models in IBM Rational Software Architect: Part 5" (see Related topics) discusses the XMI format used with Rational Software Architect. However, the XMI standard that is used in Rational Software Architect is shared with InfoSphere Data Architect, and the same concepts apply.

Model management
Model management is an important part in the data model design lifecycle. Proper model management improves performance and memory consumption when you use InfoSphere Data
Architect. When you maintain models well, the workbench can complete data modeling tasks that require less memory and processing power. You can even sometimes increase productivity when you manage models properly.

**Store model files together**

- When you work with models in InfoSphere Data Architect, store all of the files that are related to a model in the same data design project, as shown in Figure 1. If the files are stored in multiple projects, the model might become corrupted if there are references to the other models and the referenced models are not stored in the same workspace. If data models in a data design project use glossary models or domain models, store the glossary and domain model files in the same project.
- If you store glossary or domain models in separate projects in order to make them easier to maintain and reuse, store the related data model projects in the same workspace.
- Sometimes, models are stored in different design projects. If a model contains cross-referenced data objects in a model that is stored in another data design project, the other data design project should be loaded in the workspace, and the project should be open. If the project is not open, any changes that are made to cross-referenced data objects will not be propagated to the data models in the closed projects. The data model in the closed project, therefore, will be out of date, and you will have to manually correct the differences.

**Figure 1. Store model files in the same project**

Managing logical data models

**Managing packages**

In a logical data model, *packages* are *containers* that store related objects. The top-level package in a data model is the *root package*. Each logical data model has only one root package. You can create multiple *subpackages* under the root package, as shown in Figure 2. You can create entities in the root package and subpackages.
Using packages effectively

- If you do not want to create multiple model files, create subpackages in each logical data model. These packages can help you to structure the model into manageable units and keep related data objects together.

  If you create multiple subpackages in each logical data model, avoid creating dependencies (like a relationship) between packages when possible. Minimize the number of dependencies that you create. Performance is impacted when you create these dependencies.

  Each package should contain entities that do not depend on other parts of the data model. If you must share entities, create the entity at the root level (under the root package). When you create dependent entities at the root level, you can easily maintain this model in the future if the model must be split into multiple smaller models.
An example of this type of logical data model is shown in Figure 3. This logical data model has a root package (root) and two subpackages (Student and Teacher). The entities and related data objects in the Student and Teacher subpackages should preferably be independent from each other. If a relationship must exist between the Teacher and Student entities, these entities should exist in the root package, rather than in subpackages. The root package contains one shared entity, Classes.

- A data model file becomes difficult to manage when:
  - A VCS is used.
  - Packages are not independent.
  - Multiple data architects work on different packages of a single data model.

Consider separating packages into separate logical data model files, which allows each data architect to work with the data objects in a package without causing conflicts when the data architect commits or shares the changes.

- When you open a data model, the workbench loads the entire model into memory, even if you do not work with every package in the model. If a model contains several data objects and diagrams, the workbench consumes even more memory to load these objects, whether they are used or not. Performance can degrade if you open these large models and perform other
tasks that require a large amount of memory (such as comparison and synchronization or comparison and merge).

InfoSphere Data Architect can run out of memory if your workstation does not have enough memory to complete a task. For this reason, you should create smaller models. When you create smaller models, more memory is available for memory-intensive tasks.

Managing physical data models

The root of a physical data model is a database. Unlike a logical data model (which can contain subpackages), only one database object can exist. In other words, you cannot create database objects under a root database object.

Managing databases and schemas

Database objects are the root objects of a physical data model. The hierarchy below each database object is different, depending on the data server type that is being modeled, but the database object and schemas are common across all database vendors.

Even though multiple schemas can be created under a database object, it is easier to manage a single schema under a database. It becomes more difficult to manage the model when multiple users work on multiple schemas. The size of the model also increases as you add schemas, which causes both load times and time required to work with version control systems to increase. If you must work with multiple schemas, create separate physical data model files for each schema.

You can create a physical data model in several ways:

- Create a physical data model from scratch.
- Transform a logical data model into a physical data model.
- Reverse-engineer from an existing data source.

Managing changes and memory usage

- Some physical data models are created when they are transformed from a logical data model. You should make any changes in the logical data model, and then transform the logical data model into the existing physical data model. When you use this method, the models are consistent and synchronized.
- As with logical data models, when you open a physical data model, every schema and data object of that model is loaded into memory. To reduce the amount of memory consumed, move any schemas that can function independently into separate physical data models.
- When you reverse-engineer a physical data model from a database, only reverse-engineer the data objects that are necessary for your current task. For example, you might not want to work with table spaces, buffer pools, or stored procedures, but you do want to work with tables and columns. When you want to work with the storage objects, you can reverse-engineer them into a separate model or the existing model.

Cross-referenced models

If a project contains multiple models, and one model refers to another model, both models are cross-referenced models. The referenced object could be part of a diagram or relationship.
between two objects. Some data modelers prefer to share objects between two models if the models are stored in a centralized repository.

Even though you can cross-reference models, each model should be as independent as possible. Minimize the number of cross-referenced models. If possible, avoid copying data objects from one model into another model.

Cross references occur in two circumstances:

- You copy and paste or drag-and-drop a data object that has a relationship to another data object from one model to another model.
- You drag-and-drop a data object into a diagram that exists in another model.

The following list describes some disadvantages of cross-referenced models:

- All cross-referenced models must be stored in the same workspace, and more specifically, the same data design project. You cannot share between projects and workspaces.
- When you open a cross-referenced model, the workbench opens the other referenced model. Opening cross-referenced models could quickly consume valuable system resources.
- If a referenced object is modified, all data models that contain the referenced data object must also be open so that the change is propagated to all of those data models.
- Cross-referenced models can easily become corrupted if they are not loaded in the workspace when updates are made to referenced objects. It is hard to maintain cross-referenced models when all models are not open, and system memory is consumed quickly when multiple models are open.

**How to identify cross-referenced data objects in diagrams**

When you add a cross-referenced data object to a diagram, an arrow icon is added in the name compartment of the data object. You can also show the cross-referenced object in each of the open data models by right-clicking in the diagram and selecting **Navigate > Show In > Data Project Explorer**. Both of these methods are shown in Figure 4:
When you drag-and-drop an object from one model to another, the models is considered cross-referenced. The following tips help you to manage your cross-referenced models efficiently:

- When you cross-reference data objects, a message opens to verify that you want to create this cross reference. If you choose to create the cross reference, you can use the Compare editor to duplicate the child objects and properties of the copied data object.

The comparison and synchronization task is discussed further in the subsequent sections of this article.

- Create several small models so that fewer merges are required.
- All data objects in a package or schema should be independent of other packages and schemas. If a model becomes large (contains several packages or schemas), determine which packages or schemas should be added to a new model file.
- All models that cross-reference each other should be stored in the same workspace, in a single data design project. If cross-referenced models are stored in separate workspaces or projects, the workbench will not be able to process the information in the models. Models can become corrupted, invalid, or out of sync.

You can close an unnecessary cross-referenced model while you are working on unrelated data objects, but the referenced model should be available in the workspace.

**How to identify cross-referenced data objects with relationships across models**
To identify the cross-referenced data objects that are related to other data objects, run the impact analysis tool. You can locate impactor objects (objects that impact the selected data objects) or dependent objects (objects that depend on the selected data objects). In either case, be sure to include contained objects (data objects that xxxxx).

**Figure 5. Analyze impact to find cross-referenced objects**

The method that is described here helps you to identify objects that are cross-referenced and have relationships between them. You will not be able to locate objects that are added to a diagram via the drag-and-drop method. You can learn how to create diagrams of cross referenced objects in *How to identify cross-referenced data objects in diagrams*.

To find the cross-referenced objects, you can use the analyze impact functionality of InfoSphere Data Architect. You can perform impact analysis on the root object of the model. For a logical model, analyze the root package. For a physical data model, analyze the database object.

To identify cross-referenced objects:
1. Right-click the root package or database, then select **Analyze Impact**. The Impact Analysis window opens.
2. Specify that you want to locate **impactor objects** and **contained objects**, then click **OK**. A diagram opens to show the dependencies, and the Model Report view opens. Keep both of these views open.

Next, you should analyze the impact to locate dependent objects.

1. Right-click the root package or database, then select **Analyze Impact**.
2. In the Impact Analysis window, select **dependent objects** and **contained objects**, then click **OK**. A new diagram opens to show the dependencies, and the Model Report view displays a list of the dependencies.

You can use the Model Report view or diagram editor to show the objects in the Data Project Explorer.

**Note:** You cannot locate data objects in diagrams with the impact analysis functionality. The impact analysis functionality locates the objects as part of the model, but not within the diagram itself.

**Keep models small**

**Note:** The following sections are only guidelines to help you increase your productivity. You should understand that small models make models easier to manage, while minimizing memory usage. IBM does not recommend a concrete model size, because requirements differ between projects, companies, and enterprises.

When you begin work in a workspace, create a new model file or package, unless you are updating an existing project. When you use small models, you minimize the risk of creating large models with unrelated data objects. If you must reuse objects from other packages or models, use the Compare editor to clone the object. Do not copy and paste the object.

**Data modeling in a team environment**

Even though InfoSphere Data Architect is a data modeling tool, you can use the workbench throughout the complete data development and design lifecycle to help manage your projects. Version your data model files; multiple users can work on the same model file in parallel, while synchronizing their work by merging changes.

You can integrate versioning tools into InfoSphere Data Architect by installing compatible Eclipse plug-ins. When you use versioning tools in InfoSphere Data Architect, the tools also use memory and processing power. The advantage of integrating these tools is that the plug-ins seamlessly help the user to complete the tasks within the workbench without using other tools or products outside of the workbench. Rational ClearCase is one such tool that can help your team to version your model files. Because ClearCase and other versioning tools also use memory, they can require a large amount of memory to complete the task at hand, sometimes up to three to four times the size of the model.

When you create and work with small models with minimal cross references, the system has more resources available to it to perform other tasks in the data development and design life cycle. If
you work with large models that consume large amounts of memory, system resources can be impacted, and the workbench can run out of available memory.

You can increase the amount of memory available to InfoSphere Data Architect. See Increase the JVM max heap size.

Advantages of small models

- Smaller models are easier to manage in conjunction with a change management or version control system, because you and your team members require fewer merges. The models are less complex and cause far fewer conflicts when you merge changes.
- The workbench uses less memory and processing power when you open and work with smaller models that do not use cross-referenced data objects.

Copying, pasting or comparing and synchronizing to duplicate data objects between models

You might want to reuse an existing data object, but modify it slightly to use it for a different purpose. For example, you might want to reuse an attribute in a new entity, but you only want to change the name. There are a few ways that you can achieve this within the workbench:

- Copying and pasting the data object to its desired location
- Comparing and synchronizing the data object

When you clone a data object within a single model, you can use the method that works best for you. Since you are cloning an object within a single data model, no cross references are created.

When you clone a data object from one data model into another data model, avoid cross references whenever possible. As stated in section 3.2.4, you risk creating cross references when you copy objects between data models, and these cross references impact performance. Instead, you should use the Compare editor to clone the data object and its properties into another data model. When you use the Compare editor, the related data objects are also cloned, causing fewer inconsistencies.

Preferred method: Compare and synchronize to clone data objects between two models

Use the Compare editor to clone data objects.

Cloning a single data object

To clone a single data object:

1. Create a data object in the target model. The data object should be the same type of data object that you want to clone.
2. Use the Compare editor to create the cloned object:
   a. Select the data object that you want to clone, then select the target data object in the target model.
b. Right-click one of the data objects, then select **Compare With > Each Other**. The Compare editor opens.

c. In the Compare editor, select the required properties of the source data object, then copy those properties to the target data object. Whether you copy from right to left or left to right, be sure to copy from the source to the target.

An example of this is shown in Figure 6. The user, Tom, is cloning the Classes entity, which exists in two separate data models. Tom selects the properties that he wants to copy. Because Tom is cloning a single object and not any related objects, he does not copy the relationship objects. If Tom did copy the relationships, the relationship and the related data object would also be cloned and added to the target model.

**Figure 6. Using the Compare editor to copy a single data object**

<table>
<thead>
<tr>
<th>Item</th>
<th>BestPracticeLogicalDataModelId</th>
<th>BestPracticeLogicalDataModelCo...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity</td>
<td>&lt;Entity&gt; Classes</td>
<td>&lt;Entity&gt; Classes</td>
</tr>
<tr>
<td>Attribute</td>
<td>&lt;Attribute&gt; class_id</td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>&lt;Attribute&gt; subject</td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>&lt;Attribute&gt; teacher_id</td>
<td></td>
</tr>
<tr>
<td>PrimaryKey</td>
<td>&lt;PrimaryKey&gt; PrimaryKey</td>
<td></td>
</tr>
<tr>
<td>Relationship</td>
<td>&lt;Relationship&gt; Classes_Student_FK</td>
<td></td>
</tr>
<tr>
<td>Relationship</td>
<td>&lt;Relationship&gt; Classes_Teacher_FK</td>
<td></td>
</tr>
</tbody>
</table>

**Cloning a data object and its relationships**

You can also clone the relationships of a target data object by using the Compare editor. The steps are the same as outlined in section 3.2.6.1.1, but you can copy the relationships. When you copy the relationships into the target data object, the related data objects are also added to the target model.

**Copy and paste to clone data objects across models**

You can also copy a data object from a source data model and paste it into another data model.

The copy-and-paste method is not recommended, because it can create cross-references, and these cross-references impact performance and can cause inconsistencies.

**Renaming objects**

When you rename a data object, you should consider the impact of your changes. You can invalidate the model if you cross-reference data objects, and those models and projects are not open or available in the workspace. If you rename a cross-referenced object, you may have to manually update the corresponding data objects later.

When you rename a data object, keep the models in the workspace. If the models are spread across different data design projects, make sure that all of the projects are loaded into the workspace.

You should only rename objects in a domain or glossary model when all of the models that use these models are in the same workspace. If the models are not in the same workspace, the data
objects will not be renamed across data models, and you must manually update the affected data objects.

**Diagram management**

Diagrams are personalized views of data objects in a data model, and they help you to visualize the relationship between those data objects. Multiple diagrams can contain the same data object, but it is not required to show these data objects in a diagram. The actual data object is shown in the Data Project Explorer.

You can modify data objects in a diagram or from the Data Project Explorer.

In the workbench, diagrams are a visualization of your actual data objects. Therefore, if the diagrams are deleted, the data objects remain in the data model and are accessible from the Data Project Explorer. To delete a data object from a diagram, you have to right-click the object, then select **Delete From Model**. That is the only circumstance in which a data object is deleted from a model from the diagram. Any other operation performed on a diagram modifies the actual data object.

**Figure 7. Delete data object through diagram**

Diagrams consume large amounts of memory. Manage your diagrams by integrating the following tips into your diagramming tasks:

- If you are not using a diagram to modify a data model, you can move the diagram into a separate model that contains only diagrams. You can move the diagram back into the data model when you need it again. If you never plan to use the diagram again, delete it from the data model.
- Diagrams can be convenient to visualize the relationships between objects and easily create new data objects for a data model. You can create a diagram to use temporarily while you perform these data modeling tasks. When you're finished with the tasks, delete the diagram from the model.
• You can use a diagram to modify parts of a data model that you are actively developing. These diagrams should be stored in the same model and same package or schema as the data objects that you are editing. You should store them in the same locations because the diagram creates data objects in that package or schema whenever you use the diagram to add an entity or table.

• Separate diagrams that you are not currently using to modify data models into a different data model. You can move diagrams in and out of models, depending on the models you are modifying.

• Do not create duplicate data objects that are also in another model in a diagram. Any data objects that you create in a diagram are created in the parent data model. Do not create data objects in the diagrams that are in another model. Any data objects created in a diagram get created in the same model in which the diagram exists. To create a new data object, use a temporary diagram method if required.

• Use diagrams to help visualize or explain a design, data structure, or business case. You should not create a diagram that contains all of the objects in a data model. When you create a diagram, only include the data objects that are in context to help explain a part of your model, or communicate a business model.

• You should not create diagrams that contain more than 100 objects. If you have already created a diagram with more than 100 objects, modify it, move it out of the data model, or delete it if it is no longer required. In most cases, diagrams with more than 100 objects are harder to understand, and these diagrams consume large amounts of memory.

Note on deleting diagrams or objects within diagrams

• When you delete a diagram, the data objects that are in the diagram are not deleted from the model.

• If you select an object in the diagram and then press the Delete key, the data object is deleted from the diagram only.

• When you delete an object from a diagram by right-clicking in the diagram editor, you can delete the object from the diagram or model.

InfoSphere Data Architect and version control systems (VCS)

Working with models that are in a VCS

When you work in data design projects, relationships can exist between multiple files in these projects. For example, you might include glossary or domain models that are linked to the data models. To preserve these relationships, be sure to load all files from the project, and do not move them out of the project or workspace.

Which VCS to use

Versioning capabilities are not explicitly built into InfoSphere Data Architect. However, the workbench can still work with different supported version control systems.

You can use any VCS with the workbench, but only certain VCSes are explicitly supported. Try to use a VCS that the workbench supports (called an integrated VCS), so that the workbench
can help resolve the differences and conflicts that occur with an integrated VCS. When a conflict occurs with an integrated VCS, you can use the comparison and merge functionality to resolve the conflicts.

If you use a VCS that the workbench does not support, the comparison and merge functionality that is built into the workbench may not be able to resolve conflicts, and your models could become corrupted. These problems occur because unsupported VCSes typically store data model files as simple XML files and not files built on an XMI specification.

**Using IDA with Rational ClearCase**

Rational ClearCase is a VCS that you can integrate with InfoSphere Data Architect and all ClearCase operations can then be controlled from within InfoSphere Data Architect. To understand more on ClearCase usage, you can read Kim Letkeman's article (see Related topics).

**Note:** Kim Letkeman's article discusses a use case, and it might refer to Rational Software Architect. However, most of the section about Rational ClearCase applies to InfoSphere Data Architect, too, as the same concepts apply.

**How to increase memory for InfoSphere Data Architect**

You can edit the `eclipse.ini` file (in the folder where InfoSphere Data Architect was installed) to increase the amount of memory that the workbench can use.

**Figure 8. Eclipse.ini file location**

To increase the amount of memory that the workbench can use:
1. Create a backup copy of the `eclipse.ini` file.
2. Open `eclipse.ini` in a plain text editor, such as Notepad.
3. Add or modify an argument to increase the amount of memory the workbench can use:
   - If it does not already exist under the `-vmargs` section, add a new argument, `-Xmx`. Then specify the amount of memory that the workbench can use by appending the amount (in megabytes) to the `-Xmx` argument. The following example shows how to specify that the workbench should use 1024 megabytes of memory:
     ```
     -Xmx1024m
     ```
   - If the `-Xmx` argument exists, modify the current value to a higher value. The highest value that you can specify depends on how much memory is installed on the computer and how much memory the operating system can let the workbench use.
     The value that you specify for `-Xmx` should be multiples of 128. You can start by specifying 128, then increase the value by 128. So you would start with 128, then specify 256, then specify 384, and higher until you find a value that allows you to work efficiently.
4. Start the workbench.

If your operating system allows you to use this value, the workbench will open. If the workbench does not open, you must change the value. If the product does not start, try to reduce the `-Xmx` value to find the best value the operating system will accept.

**Preventing instability: Studying use cases**

**Use case #1: Large models**

One of the main ways that a model can become corrupted is to encounter an *out of memory* error. When you run out of memory, the result is unpredictable, and the model could become corrupted. The only way to recover from this type of problem would be to regularly back up your work. You can create a personal backup or use a version control system like Rational ClearCase.

To prevent this situation:

- Create and work with smaller models.
- Increase the memory that is available to InfoSphere Data Architect to make it easier work on larger models. However, limitations can arise, depending on the operating system that you use and the amount of memory that is installed in your system. In most cases, though, increasing the memory does help when you aren't as restricted by limitations. Read the Increase the JVM Max Heap Size section, to learn how to increase the amount of memory that is reserved by the system for InfoSphere Data Architect.

**Use case #2: Cross-referenced models**

Sometimes, referenced model objects are changed, and changes aren't propagated correctly because the other cross-referenced models are not available, or the models are not updated and saved correctly.
More concrete use cases are provided in the following sections. For these use cases, assume that there are two models: M1 and M2. Objects in the M2 model reference objects in the M1 model.

**Use case #2a: Diagrams**

Sometimes, data objects are cross-referenced in diagrams. For example, a diagram in M2 references objects from M1.

Even though diagrams may not be updated properly, or diagrams get corrupted, you can simply delete the diagram and recreate it if necessary. In most cases, diagrams update automatically when you open them. When diagrams get corrupted, the parent data model should not become corrupted.

**Use case #2b: Relationships**

Problems can arise when your data objects are related across different data models.

- **Primary key or foreign key relationships:** A child table or entity in M2 references a parent table or entity in M1. Someone updates one of the data objects. The relationship becomes corrupted or out of sync.
- **Generalization relationships:** When someone changes the primary key for a table in M1, the corresponding foreign key in M2 is not updated correctly.

**Use case #2c: Domain models**

You cannot avoid cross references in domain models, as these models are designed to be separate from the logical data models and physical models that use them. All of the models that refer to the domain model should be in the workspace when you update the domain model.

For example, a table in M2 is cross-referenced with a table in M1. A column in that table in M1 contains a domain reference. When the domain is updated, if the M2 model is not also open, the domain reference in M2 is not updated correctly.

**Preventing problems with cross references**

To prevent these issues:

- Create and work with smaller models.
- Minimize the amount of cross references that you create.
- Make sure that all related objects and models are in the same workspace before you proceed.

**Working with large data models**

The base recommendation in model management is to keep models small. However, there are cases where this cannot be put into practice. When dealing with large models, the following are some tips that may prevent out of memory errors. These include: turning on the heap monitor, backing up changes frequently, and increasing the JVM maximum heap size.
Turn on the heap monitor

It is recommended that you turn on the heap monitor (from Preferences -> General -> Show Heap Status) and keep an eye on the heap usage. When turned on, the heap monitor appears at the bottom of the IDA workbench window.

Every once in a while, or after doing a memory intensive operation, click the garbage can in the heap monitor several times. This attempts to assist the garbage collection. If the heap monitor gets too close to the maximum heap used, do not do any more memory intensive operations such as save and check-in. The save operation might run out of memory and cause issues with the model you are trying to save. Simply exit and restart IDA. This is true of ALL versions of IDA.

Back up your changes frequently

The local history facility (Preferences -> General -> Workspace -> Local History) provides the ability to keep several backups on the local file system. Using this system as the primary means of model backup or change recovery could clutter up the file system because multiple versions of the models are kept and loaded into IDA. The recommendation is to rely on a supported version control system (CCRC, RTC, or CVS) by checking in changes frequently instead of using local history. Local history can be used in-between check-ins. The time between check-ins should not span more than a day or two. The preference setting for saving local history should be reduced from 31 days to 1-2 days.

Although the version control systems will remind you to save the changed file while you check in, it is better to save the changed file before checking in the file.

Increase the JVM max heap size

You can edit the eclipse.ini file (in the folder where InfoSphere Data Architect was installed) to increase the amount of memory that the workbench can use.

Conclusion

This article covered the following:

- Managing models in IDA
- Splitting the models into smaller models
- Managing cross referenced models
- IDA performance tuning
- Working with larger models
- Managing models with a version control system
Related topics

- You can read more about using Rational ClearCase in the article "Comparing and merging UML models in IBM Rational Software Architect: Part 5 Review the section" (developerWorks, Kim Letkeman, July 2007). Specifically, read the section Introduction to ClearCase.
- Refer to the IBM InfoSphere MDM Version11.0 information center.
- Refer to the IBM Cognos Business Intelligence 10.1.1 information center.