topdown CONSULTING

Oracle Hyperion Data Relationship Management Best Practices, Tips and Tricks

WHITEPAPER

This edition published July 2012.

TABLE OF CONTENTS

Oracle Hyperion Data Relationship Management Best Practices, Tips and Tricks

Topdown Consulting

Executive Summary

Oracle Hyperion Data Relationship Management (DRM) is a hierarchy management tool. It pulls together all the data from disparate sources, allowing you to apply one set of rules and processes so that you are working with a complete, accurate, and consistent set of data. Oracle DRM is how enterprises manage operational and analytical master data. To be more precise, it helps you manage master data and metadata as well as data quality and integrity.

Oracle DRM is an open, transparent centralized data repository that has integrity enforced by business rules and validations. DRM provides the master records for all the information you need to hold. It can push the consistent set of master data out to different Oracle EPM systems like Hyperion Planning, HFM, FDR, and Essbase, or it can function like a traffic cop and collect data from the systems and roll it into one.

One of the key features in DRM is enhanced reporting integrity: DRM provides organizations with the ability to manage complex relationships between master data elements, business rules, and peripheral systems' requirements, ensuring information accuracy across enterprise systems. This white paper discusses how organizations can manage these complex relationships, providing insight for companies evaluating DRM for new implementations and/or with established systems.

Oracle Hyperion Data Relationship Management – Hierarchies and Nodes

DRM is a hierarchy management tool. A hierarchy is a structure of members (nodes) in a parent child relationship. Hierarchies are often used in accounting applications and reporting tools in order to report or manage data at different levels. For example, the General Ledger typically manages data at the bottom member level, whereas Hyperion Financial Management typically reports at a higher level in the hierarchy. DRM manages the relationships of the nodes as well as the properties (attributes) that go with the nodes within a company's hierarchies.

The typical applications that integrate with DRM are Hyperion Financial Management (HFM), Essbase, Hyperion Planning, OBIEE, and various General Ledgers. Typical hierarchies in these applications are Account, Entity, Department, Product, Project, Currency, Cost Center, Profit Center, Location, Scenario, and Year. Other hierarchies that may be used by various other applications include Employee, customer, vendor, brand, sales, marketing, and agency.

DRM does not allow nodes with the same name to exist in the same hierarchy. This rule may drastically affect the design of your application. If your downstream application needs to have nodes with the same name in the same hierarchy, you can either create shared nodes within the hierarchy, or create another hierarchy with the shared nodes, and then merge these hierarchies together during the export process.

A node must have the same children in all the hierarchies within a version. This rule may also affect how you design your application. With the shared node functionality, there are implicitly shared nodes (descendents of the shared member). During the export process you can either choose to include these implicitly shared nodes, or not to include them. If you have chosen to split the hierarchy into two hierarchies, the duplicate member will share the same descendants. If you don't want the descendents to be exported, you will have to suppress them from the export.

A version in DRM is a container of all the hierarchies. Versioning is a term that refers to a process of finalizing the current working version and then copying the version to a new version after the month end close is complete. Versioning allows you to keep a snapshot of the masterdata for each close. Naming the version with a date timestamp will organize the versions alphabetically within the status. For example, if you work for company ABC, you may want to name your versions ABC_20120102. This would indicate that the version was created on January 2nd. This should have all the metadata for the January close if the next version is named ABC_20120203. Then your versions will be sorted in order within each status section. There are four statuses within DRM and are sorted from top to bottom in this order: working, submitted, finalized, and expired. Users can edit working versions. Administrators can only edit submitted versions. Finalized and expired versions cannot be edited.

DRM was designed to manage hierarchies across multiple applications. In order to have one version of the truth that has application specific nodes in the hierarchies, there must be a way to filter out the nodes that do not apply. Typically this is done with a property that indicates if the node should be exported to the application. This seems pretty simple, but there are a couple of different methods to do this, and the way this is set up may affect your node types, validations, exports and many other key features.

Properties in DRM

Suppression property

The first technique that we will discuss is called a suppression property. This property will be set to false by default and then marked as true if it should not go to the downstream application. The advantage to creating a suppression property is that when the applications are very similar or you only have one or two downstream applications there should be very few nodes that would need to be set and therefore the maintenance should be relatively simple. The problem with this is when you start adding applications and the logic starts getting complex, the negative logic (if the node is set to false it is exported to the application) can be confusing. It is also confusing because it will be set to false in hierarchies that are not exported to the downstream application and therefore you will have to look at the export to know what hierarchies are exported and then look at the property to check if it is being exported.

Export property

The second technique is called an export property. This property will also default to false and then be set to True for all the nodes that go to the particular application (opposite of the former technique). This may ultimately have more values set in the database, but has positive logic and is typically easier to understand.

An export property should be created for each subscribing application even if the application should get the combination of two other applications. For example:

Cube1 gets all the OpEx accounts and Cube2 gets all the Gross Profit accounts. Cube3 gets all the accounts from Cube1 and Cube2. Instead of creating a query for Cube3 that says Cube2 + Cube3, go ahead and create a Cube3Export property and have it be RWDerived to look at Cube1 and Cube2.

This may be cut and Dry for the Accounts hierarchy, but then when you get to the Departments or another hierarchy, you may find that this logic does not work. Then you have to add the Cube3Export property anyway and then you have one hierarchy that looks at the property and another hierarchy that looks at two other properties.

Creating export properties

Create all the export properties as local read write derived properties. Just because you have an alternate hierarchy in DRM does not mean the node in the alternate hierarchy is going to be exported to the application if it is set to export in the main hierarchy (and vice versa). For this reason, the node should be set to local. There may be logic that you will want to set later within the implementation, or after the implementation to have the properties derived their value. For instance, if you want all

limb nodes with descendents to be set to True. This way, your stranded parents will not be set to true unless they get children attached to them. When the children are added, the stranded parents are auto set to true.

When creating properties there are many choices you need to make. You have to decide the name, the description, the label, the datatype, what property level to use and what propertytype to use, the default value if it has one and whether it has a list of choices. The three most important choices to make are the name, the property type, and the property level. These options cannot be changed once the property is saved.

Property Prefixes

Prefix your property names according to the application that will be using the property. If there are multiple applications using the same property, you may consider having a common prefix (DRM for example) that is used for maintenance. If the individual applications need to have their own property, it can be derived from the maintenance property. Use names that clearly explain what the property was created for. Constants can be created in a property at the version level. Prefix the properties with "ver" so they are easily identified in the property definition list. Prefix properties that are used in the validations with "val".

Property types

The property type can be defined, derived, lookup or RWDerived. The properties that are going to be maintained manually and do not have derivation logic should be created as defined. If there is derivation logic that can be applied to the property and it's 100% valid, a derived property should be used. If the property derives the value based on another properties value and can be stored in a lookup table, the property should be set up as a lookup property. If the property can be derived, but there are exceptions to the rules, a RWDerived property should be created. If you are unsure if the property should be defined or RWDerived, make the property RWDerived and have the formula point at the properties default value. This way if you need to change the formula later you can, but for now you can maintain the property manually. If you are unsure if the property should be derived or RWDerived, make the property derived. It is easier to delete a derived property and replace it with a RWDerived property than the reverse. You can also create a defined property later that the derived property looks at to override the formula if you need to.

Property level

The property level has four choices:

1. Version
2. Hierarchy
3. Global
4. Local

The version properties are set at the version level. When exported, they will be the same value for every node in the version. Typically constant values are set up at the version level. Hierarchy properties are set up on the hierarchy and pertain to the hierarchy. When exported, each node in a specific hierarchy will have the same value. Global properties are set on the node and are constant across hierarchies. For example:

Consider a name or the description property. No matter what hierarchy the node is in, it has the same description. Local properties are also set on the node, but can have different values in different hierarchies. An example of this would be the parent property. Since a node can have a different parent in each hierarchy it's in, the parent property must be local. If you are unsure if a property should be local or global, you should ask yourself if the value can be different if the node is in a different hierarchy. If the answer is NO, then the value should be global.

Working with Properties

Descriptions

Use the description in the property definition to clearly describe what the property was created for. These descriptions will help you later if you need to research what the property was created for or if you need to change the property or document it.

Reusing properties

Reuse your properties when possible. Having multiple properties that do the same thing can cause confusion. A good example of this is the nodetype property. If you set up a different nodetype property for each application, it would cause confusion as to which nodetype property is relevant for the node. Sometimes this is unavoidable because different applications may have different values for the same property. An example of this would be the storage property in Essbase may be dynamic calc

in one cube, and store in another cube based on what level the cube receives. In this case a separate datastorage property would need to be created for each cube.

Prefixes

Prefix your nodes and come up with a naming convention for your prefixes early in your implementation. The benefit to prefixing a node is so the node will not duplicate with another node in a different hierarchy. A common rebuttal to this logic is that essbase doesn't allow nodes in different hierarchies to have duplicates so there is no reason to prefix because you cannot have these duplicates in essbase anyway. The problem with this premise is that you may have another application that has a hierarchy or a level in the hierarchy that is only common to that application. An example of this would be if you have Hyperion Planning nodes that do not exist in essbase, and they overlap names in Essbase. Then you will have to prefix these planning only nodes, and if you have already implemented essbase, then your implementation will be mismatched with some nodes that are prefixed and some that are not. This will be confusing to those who were not part of the implementation and you will have to build extra properties to be used to manage the inconsistencies.

Typically, implementations like to prefix based on the hierarchy however I would suggest taking this one step further and prefix based on node types. Other implementations will have a hierarchy prefix and then a secondary prefix that indicates the nodetype. Or there may be logic to derive the nodetype based on the level in the hierarchy. These are all good and common practices, however if you would like to simplify your maintenance later, you will spend time at the beginning of the project in order to identify the nodetypes and prefix the nodes accordingly.

Grouping

Group your exports in a book and prefix them according to their group. You may have a set of exports for HFM, another set for each essbase cube, and possibly a set for reporting purposes. Group these in a book and also having them prefixed according to the group will make your application better organized, hence making them easier to find later.

Property queries

Create property queries for each application and re-use the queries for each export to the application. The property query should simply look at the export property that was created for that application. For example, the HFMExport query should look at the HFMExport property. This query can be used by all 13 exports that are used to create the app file for HFM. Name your property query according to the criteria. Many people may name their queries by the export that is going to use the query, but if you are going to re-use the query then you should name it according to the criteria within the query.

Other DRM Features

Rules

DRM includes rules that govern the way the tool acts and manages the hierarchies. Knowing these rules is important in order to understand how to best design your environment.

Automators

Imports and Actions Scripts (automators) are two features that DRM has available to bulk load masterdata.

An import loads a flat file into a new version. There are five different sections in the import file. The version section is not typically used, but can be used to populate the properties at the version level. The hierarchy section indicates what hierarchies will be in the version, and populates the properties at the hierarchy level. The relationship section holds the parent child relationships of the nodes and can also populate property values. The node section populates the global properties, and the hierarchy node section populates the local properties. Imports are much faster than action scripts, but they are loaded into a new version and then must use a blender to get the changes into the current version. Imports and blenders are typically used for implementations or projects that need to be moved into production.

The action script loads a flat file into an existing version. Each record in the action script has a specific update to a node. If you need to make 30 property changes to one node, there will need to be 30 records in the action script. Action scripts are used for day to day or month end processing.

Blenders merge two versions into one. There are many options on the blender that help you to merge just what you want from the source version to the target version. If you just want to move the structure, or if you just want to move one property value, the blender can help you. Blenders were created to help with initial implementations. They were meant to be used to merge versions together once imported. If you have 5 source systems in an implementation, you will have 5 imports and 4 blenders. Each of the blenders should have the same property values as it's corresponding import. For example, if you have HFM, Planning, and three Essbase cubes and you choose to have HFM be your main version, your planning blender will have the same properties as your planning import. In this scenario, planning would be your source version and HFM will be your target version.

Migration utility

Use the migration utility to create your properties. If you have 10 properties that are similar, you can create one property, export this to an xml file, modify the xml file and then import the changed property. CAVEAT: This is not a best practice due to the error potential editing the xml file, however this could save you time and effort during your implementation.

Staying organized

Keep a list of all the validations by application, hierarchy and by nodetype that are being discussed during the implementation. Many times the user will be talking about a leaf member but they will say that all the nodes will need to follow a rule. For example, the General Ledger users may say that all the accounts must be 10 digits when in reality, only the leaf member need to be 10 digits because only the leaf members are in the general ledger.

Final Thoughts

Managing the DRM repository is potentially a change in process, because it does require one owner to manage definitions and hierarchies, the way information gets rolled up, and if you want, pushed out to the different systems. However, you can distribute DRM management based on business units—basically each one can access DRM and do what they need to do. For example, when a new account is added to the master data, the Essbase admin fills in his Essbase properties for it. The Planning admin takes same account number and fills in planning associations. HFM takes same account number and completes for HFM. Multiple people may enter the data, but the work is all done in one place.

Instead of the constant struggle of trying to validate financial reports monthly, imagine having a centralized information platform that allows you to:

- Analyze aggregated data from multiple sources in one convenient location.
- Share results throughout your organization, locally or globally, via web-based portals and dashboards.
- Provide all business units the data they need, how they need it, by automating report generation and developing customized reports that let you tailor results to the needs of stakeholders and/or corporate decision-makers.
- Improve collaboration between departments by sharing data.
- Cut down on manual, repetitive data processing tasks by automating analysis while integrating third party tools and capturing business rules and best practices.
- Increase productivity, operate with higher efficiency, and make better-informed decisions by browsing, collating, reporting, and sharing local, corporate, and public information in an efficient, timesaving manner.

Achieving a 360-degree view of all your financial data doesn't have to be a pipe dream. To be successful, though, you must implement the right strategy and use the best technology.

About TopDown Consulting

About TopDown Consulting, Inc. Founded in 2000, TopDown Consulting is the acknowledged leader in designing, implementing, and deploying EPM solutions. TopDown has the experience, expertise, and proven approach to deliver successful implementations for Global 2000 clients. Our consultants average 20 years of Hyperion and industry experience with a complete range of skill sets, including: solution and system architects, CPAs, finance executives, MBAs, analysts, and project managers. TopDown's proven Project Success Methodology provides a customizable strategic framework for guiding and measuring project initiatives, enabling us to deliver solutions that meet our clients' current and future business and technology needs and their unique corporate culture.

TopDown Consulting, Inc. serves clients nationally and internationally from our San Francisco headquarters. For more information or to inquire about our services, please contact us.

TOPDOWN CONSULTING, INC.

530 Divisadero Street #310
San Francisco, CA 941177

Phone: (888) 644-8445 (Calls from within US and Canada)
Phone: (617) 576-8445 (Outside US and Canada)

Email: info@topdownconsulting.com
Web: www.topdownconsulting.com