NON UNIQUE MEMBER NAMES: AN EASIER WAY TO BUILD ANALYTIC APPLICATIONS

One of the prevailing business drivers that have helped propel MultiDimensional technologies as the preferred platform for delivering analytic applications is the ability to deliver self-service analysis to the end users. The ability to surface dimensions, hierarchies, members and natural drill paths in the format the business user can understand is key to the self-service paradigm.

INTRODUCTION

In the beginning, many analytic applications were focused in the finance organization, and contained more summarized data. This summarized data could be guaranteed to be represented via unique member names, and these unique identifiers made it easy for the end user to query data they were interested in, without need of writing complex query statements. Developers of these applications often employed various "transformations" for creating these unique identifiers, such as prefixing, suffixing or concatenating "key" fields. When more detail data was required as part of the analysis, the user was directed (either programmatically or manually) to the warehouses or marts for more detail level data, where non unique (or duplicate) member names were more likely to occur. However, over the years the amount of data needing to be analyzed in a company has exploded exponentially, and the requirement to deliver more granular analytics is paramount. This greater granularity is manifested in the need to support ever larger numbers of dimensions and members. The technology has also matured to efficiently handle this class of applications, and Hyperion Essbase 7X introduced the aggregate storage option (ASO) and outline paging to support larger sparse data cubes. As customers take advantage of the new advancements to deliver ever larger cubes, and move beyond traditional finance applications to deliver analytics on operational data, the chances for having member name conflicts are far greater. Hyperion® System™ 9 BI+™ Analytic Services™ introduces support for non unique member names that automatically handle member name conflicts and thereby removes the need for the user to explicitly define member name transformations.

INTENDED AUDIENCE

The paper is intended for administrators and developers who build analytic applications using Hyperion® System™ 9 BI+™ Analytic Services™. It will also touch on the effects that duplicate member names have on the end users. The data sample mentioned here are in a generic relational or flat file form not in an Essbase rule file or other specific format.

THE NEED FOR NON UNIQUE MEMBER NAMES

Building analytic applications where unique member names cannot be guaranteed takes some level of work to design. The challenge is to develop a strategy to uniquify member names while still delivering a business friendly experience for the end user. Once in production, these applications may still run into additional issues during their lifecycle. Because of the goal of these applications are to continually mature to reflect the state of the business, they must be continually updated in an automated fashion, without manual intervention. This means that new members, and possibly new dimensions, will continually be introduced into the application. The strategy used during design and development of the initial application needs to be review. These strategies would need to include considerations for the following:

- How to detect new members, especially members that will have name conflicts?
- When will conflicts have to be solved that require human intervention?
- What is the resolution time for these issues, and how does this impact the end users?
- Will new or modified transformations change the names of existing members which would effect end users and/or existing reports?
- As more analytic applications are built and deployed, and as these applications grow in scale, more time is spent in maintenance activities to insure continued uniqueness of member names. By supporting non unique (or duplicate) member names and aliases in Hyperion® System™ 9 BI+™ Analytic Services™, these maintenance activities can be largely eliminated. The benefits of supporting non unique names is many, including:
- Less work is spent designing and subsequently building and maintaining a single dimension. A dimension with duplicate member names, such as a Time dimension, can be automatically loaded without transformations.
- Member names and aliases can be more "business friendly", since no transformations are needed
- Less time is spent resolving cross-dimensional member conflicts, since the non unique naming feature is available both within and across dimensions.
- Ongoing maintenance is made easier because there is no need to modify transformations when new members or dimensions are added.

MEMBER NAME RULES FOR NON UNIQUE MEMBER OUTLINES

The rules required to build a uniquely named outline is simple. Every member name and alias in the outline has to be unique. This is even true for shared members if you view them as one member with more than one parent. That is, the "value" for any cross-dimensional intersection for a shared member is the same as the original member.

The rule for a non unique member outline is different, but also simple. Every member name and alias must be unique within its parent. This is a small logical change that has a big impact on how outlines are built and used.

BUILDING OUTLINES

This section will cover the specifics for building non unique member dimensions from a source expressed as a either a recursive hierarchy, or a generational hierarchy.

RECURSIVE DIMENSION BUILDS

Building dimensions from a recursive source for unique outlines is fairly straight forward. The only requirement that unique outlines have is that you have two columns that have the parent-child relationship. There is little reason to apply transformations since the input data guarantees uniqueness in the dimension but there could be conflicts with other dimensions in the outline. Take the data shown at the top of the next page:

Table 1

Employee ID	First Name	Last Name	Start date	Department	Phone Ext	Manager ID
10123	Bill	Fence	1/19/2002	Development	5512	Null
10124	Steve	Storm	12/8/2003	Sales	5522	10123
10125	Scott	Knife	2/26/2004	Sales	5554	10124
10126	Bill	Fence	3/7/2004	Sales	5564	10124

BUILDING OUTLINES

This is an employee hierarchy and as a recursive hierarchies it has

- A unique identifier of the member (Employee ID)
- All other columns are attributes of the member identifier
- One columns has a logical parent relationship with identifier (Manager ID)
- All parent identifier values should exist in the member identifier column.

If building a unique outline, your only choice in the above example is to use the Employee ID (Child) and Manager ID (Parent) columns. If the data can be relied on to be clean, we should have no conflicts, nor need transformations to build the employee hierarchy. However, there will be problems for the end user in navigating or understanding this dimension.

The problem is that Employee ID is difficult to remember. It will be difficult for end users to use a cube with only Employee ID. We therefore add 'First Name + Last name' as an alias but we run into trouble. 'First Name + Last name' is not unique, and we would get a rejected alias (record 4). We solve this in unique outlines by adding Employee ID to the alias string, either as a prefix or suffix. Whichever method is used, we would now get a unique alias 'Bill Fence 10126' but the employee ID is not adding much to end user ease of use.

Building unique outlines from a recursive source can be summarized as:

- Relatively easy to build
- Little need for transformation on name/ID
- Name/ID is difficult for end users to understand, and use of an aliasis probably needed
- Name and Alias have the same constraint. E.g. values have to be unique in the whole cubes
- Transformations to make the alias unique is most likely needed
- As Aliases are transformed, the name becomes longer and can also be difficult for the end user to understand.

Building non unique outlines gives more flexibility in building the dimension, and differs in the following aspects:

- Input columns are of two types: keys and names
- Names need only to be unique within a parent
- Keys need only to be unique within a dimension

Let's start with that columns are of two types. Look at the table 1 data above and look at all the columns. There are three columns where each value uniquely identifies a logical member (Employee ID, Manager ID and Phone Ext) all other columns have no uniqueness guaranteed.

Duplicate recursive hierarchies have three columns: Member key, Parent key and an optional Member name.

You do not have any choice on which two columns have the parent-child relationship. You do have some choices with names, but we will stick with the first name and last name columns, as this will present the information to the user in a logical format. What is different with the duplicate outlines if we use Employee ID as the member key, Manager ID as parent key, and 'First name + Last name' as member name or alias.

BUILDING OUTLINES

First of all we do not have to worry about other dimensions when in comes to uniqueness of the member and parent keys. No other dimension can have a name conflict, so it is guaranteed that no transformations are needed. The member name cannot be guaranteed to be unique since there is two employees named "Bill Fence", but that is not an issue since they have different parents. This outline will build fine with no transformations regardless of any new data in any other dimension. The member name will not have any extra characters since there is no need to added Employee ID to the member name.

Duplicate recursive hierarchies can be summarized as:

- Easy to build and are unaffected by other dimensions
- Guaranteed to not need transformation on keys
- Name is optional but probably needed
- Names do not need any extra characters to make them unique

You can, of course, add aliases to duplicate outlines and they have the same requirements as names (unique within parent). We did not discuss shared members in the example above but there are not any differences in capability between unique and duplicate outlines when it comes to recursive hierarchies with shared members. Both cases would use the column Employee ID for deciding of a member is shared. See generational build for more information on shared members.

GENERATIONAL OUTLINE BUILDS

Generational build are more work to build than recursive hierarchies because the input data does not guarantee uniqueness within the dimensions, but there is more flexibility in designing transformations since you can have different transformations for each generation of the hierarchy. Let's take a look at bulding a unique outline from the data below (Table 2).

Simple Case

Assume the dimension will be called Region. We can add Country as generation 2 and St/Province as generation 3 with no issues. Try to add City as generation 4 and we begin to run into issues. The second and fourth records are rejected because of duplicate names. Adding columns name City to the value would solve second record but not the fourth. Adding parent names will solve both issues but the names would not be intuitive for the end user. Note, that in this simple example, we only explored resolving name conflicts within the dimension. However, since unique outlines require member names/aliases to be unique not only within a dimension but across dimensions as well, we would need to address any interdimensional conflicts as well.

Duplicate (non unique) outlines will behave differently in one major way:

- Names need only to be unique within parent

Return to the Table 2 data below, and let's build it as a non unique dimension. We add Country as generation 2 and St/Province as generation 3 with no issues. Try to add City as generation 4 and we will still not run into any issues. The Second and fourth records are OK because the names are unique within their parents.

Table 2

Country	State/Province	City
US	New York	Albany
US	New York	New York
US	Washington	Vancouver
Canada	British Columbia	Vancouver

Table 3

Patient ID	Patient Name	Doctor ID	Doctor Name	Dr. Specialty	Insurance Type	Insurance Co
10123	Bill Fence	MD-111	M Babcock	Internal Med	HMO	Blue Cross
10123	Bill Fence	MD-222	E Mergency	Trauma	HMO	Blue Cross
10124	Steve Storm	MD-111	M Babcock	Cardiovascular	PPO	Blue Cross
10125	Scott Knife	MD-333	E Mergency	Immunology	FFS	Blue Cross
10126	Bill Fence	MD-333	E Mergency	Hematology	PPO	Blue Cross

GENERATIONAL OUTLINE BUILDS

Alternate Hierarchies

Now, let's take a look at creating alternate hierarchies that require the introduction of shared members. Table 3 above will be used for these examples.

We want two hierarchies

1. Doctor Specialty – Doctor – Patient
2. Insurance Type – Insurance – Patient

Some of the issues we will run into with a unique outline on the first hierarchy

- Doctor name is not unique (need transformations)
- Doctors have more than one specialty, Doctors must be shared members or be transformed in a some way
- Patient name is not unique (need transformations)
- A patient has more than one doctor, Patients must be shared members or be transformed in a some way

What we will run into on the second hierarchy

- A single insurance company offers multiple types of insurance (need transformations)

We can build the dimension, but it will require a lot of transformations and the names will not be intuitive for the end user.

Utilizing duplicate outlines, we can benefit from the following behavior:

- Names need only to be unique within parent

Assume we want the same two hierarchies as presented above

1. Doctor Specialty – Doctor – Patient
2. Insurance Type – Insurance – Patient

As can be seen, we will not encounter any of the issues in building these two hierarchies in a non unique outline that we did when it was built in a unique outline. Both the first (or primary) hierarchy as well as any secondary (or alternate) hierarchies will be handled without any special considerations.

SHARED MEMBERS

The member name rules for unique and non unique outline has the biggest effect on shared members.

Unique Outline:

- Every member name has to be unique within the cube.
- A member will be rejected if the name is not unique (with exception of a shared member)

Non Unique outline

- Every member name has to be unique within it parent.

A shared member can be view as one member that has more than one parent. What this means to a non unique outline is that it will always treat shared members as two different members with the same name but different parents unless we have extra information. In other words, name and parent is not enough information to decide if a member is a regular or a shared member.

What we need is a key that identifies members regardless of their parent. Luckily, keys are common in the input data, especially when coming from relational sources. Looking again at table 3, if we use 'Patient ID' as a key we would have enough information to make patient shared members between hierarchy 1 and 2.

GENERATIONAL OUTLINE BUILDS

Key must exists in recursive hierarchies (it would not be a recursive hierarchy otherwise), but they are optional for generational built hierarchies. It would be unusual to have relational generational data with logical shared members and no keys.

GENERATION BUILD SUMMARY

Unique dimensions built generationally can be summarized as:

- Require understanding of the complete outline data to build
- Requires transformations
- Names are transformed and becomes longer and not that intuitive for the end user

Using non unique naming functionality behaves differently in three aspects

- Names need only to be unique within parent
- Keys are needed for shared members
- Keys need only to be unique within generation/level

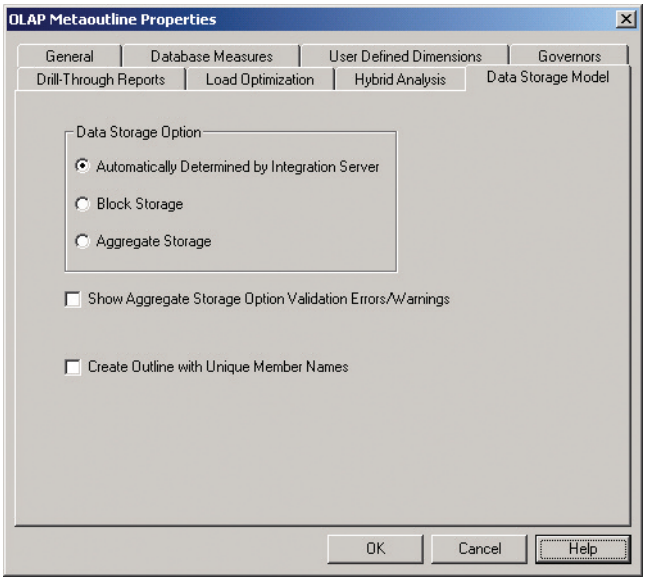
Non unique dimensions built generationally can be summarized as:

- Easier to build
- No need for transformations since names do not need any extra characters to make them unique
- Keys are needed for shared members

PERFORMANCE AND SCALABILITY OF OUTLINE BUILDS

Non unique outlines are much easier to build than unique outlines, especially for generational builds. What impact does this have on outline build performance and scalability? Performance depends on what tool is used to build the outline. But generally a duplicate recursive hierarchy will not be much different from a unique recursive hierarchy with an alias.

The generational build is a little different. In the example of using Hyperion® System™ 9 BI+™ Analytic Integration Services™ it will spend more time on member load to simplify and improve data load speed. The effort to load a member is similar to a unique generational build with an alias. Speed is depended on the size of the member names. Hand tuned transformation in a unique outline can often be smaller than the automatic transformation. An 2-3 minute / million member increase in elapsed time for duplicate outlines are expected. To turn duplicate outline on for Integration Services set the Metaoutline properties dialog.



Member load: 2 million plus members

Model 1 ASO

Member Load	
Unique	Duplicate
11:30	15:07

Member load: 2 million plus members

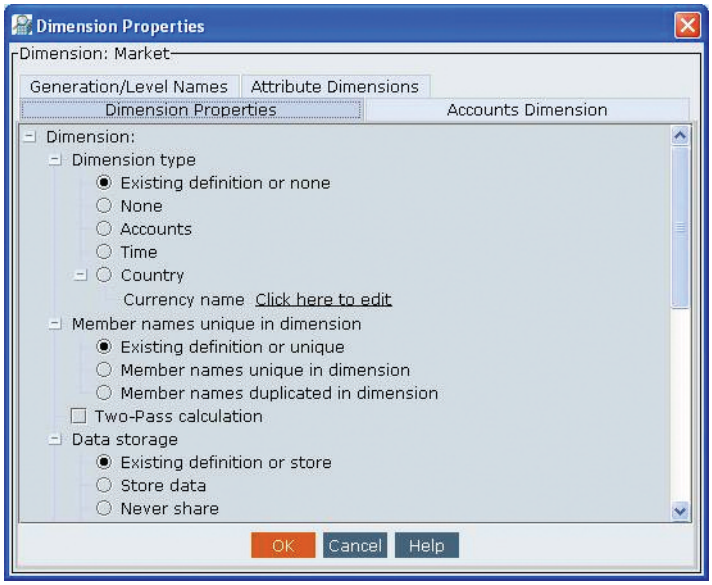
Model 2 BSO

Member Load	
Unique	Duplicate
10:28	14:05

PERFORMANCE AND SCALABILITY OF OUTLINE BUILDS

Scalability has to do with memory usage e.g. how big outline can be built or how many outline can be built at the same time. Duplicate recursive hierarchies are about the same size as unique hierarchies if the optional name is added then the size is similar with a unique hierarchy with an alias. Generational builds with no alias are similar to unique outlines with an alias.

Load rules has a smaller overhead for member load but pay a price at data load time instead. You turn the duplicate outline on by editing the outline properties see screen shot below



DATA LOADS

How the dimensions are built (either generationally or recursively) has no affect on data loads, but the data load itself is affected by the change in naming rules.

In a unique outline, every member name has to be unique within the cube. This means that each data load records is uniquely identified by the intersection of a member name for each dimension and the data values

In a non unique outline every member name has to be unique within it parent. That means that a member name is no longer enough, and we need to fully qualify the name with its fully qualified path. For example, using the original table 2 data (repeated below) the members member New York is not uniquely defined. It could be State New York or City New York.

Table 2 (repeated)

Country	State/Province	City
US	New York	Albany
US	New York	New York
US	Washington	Vancouver
Canada	British Columbia	Vancouver

Market].[US].[New York].[New York] is a fully qualified path and uniquely defines the city New York. The brackets are the MDX standard way of specifying a qualified path.

The easy solution is to use fully qualified path for each member name at each intersection (each dimensional combination). That will work but can have performance impacts because

- Much more data is sent back and forth to Analytic Services
- The network card could be a bottle neck if data needs to be sent over the network
- It might be expensive to create fully qualified paths. E.g. a 200 million data load fact table with one or more joins for each of the dimensions

Analytic Integration Services will automatically generate the data load that will work with the corresponding Integration Services member loads. Performance is normally the same as unique data load. This is accomplished by analyzing the relational model and taking advantage of information that the relational joins have. It is possible for non unique outlines to be faster than unique outlines when it comes to data loads because of the ability to generate better SQL. This is made possible by using the new key column attribute information that was not available before. Specifying key input columns on level zero members will increase the changes of data load being fast.

Data Load: 6 million plus data load records

Model 1 ASO

Member Load	
Unique	Duplicate
11:10	12:32

Data Load: 6 million plus data load records

Model 2 BSO

Member Load	
Unique	Duplicate
2:31:33	2:30:10

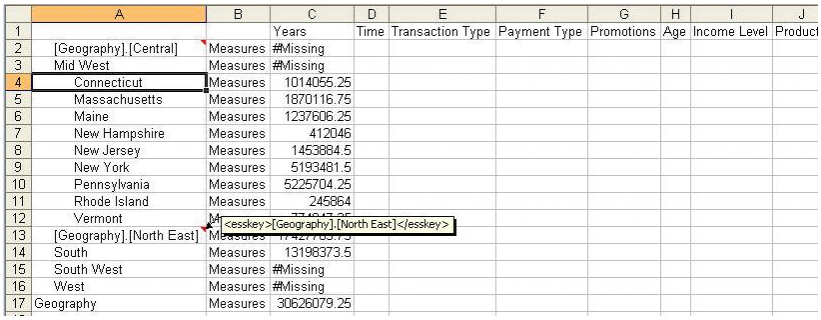
Load rules performance is relation the length of the qualified path. A two composite member name [Market].[US].[New York] is takes about twice the time of a unique New York.

EFFECT OF NON UNIQUE OUTLINES ON END USER ACTIONS

Now that we managed to build a non unique outline, it is important to understand how this will affect the end user.

Navigation outlines

The first thing users will encounter is the navigation of the outline itself in order to perform member selections or formulating queries. The actual behavior of a zoom in or zoom out is no different, however the member name itself does not give enough information to know which actual member the user wants. More information, like which parent, level



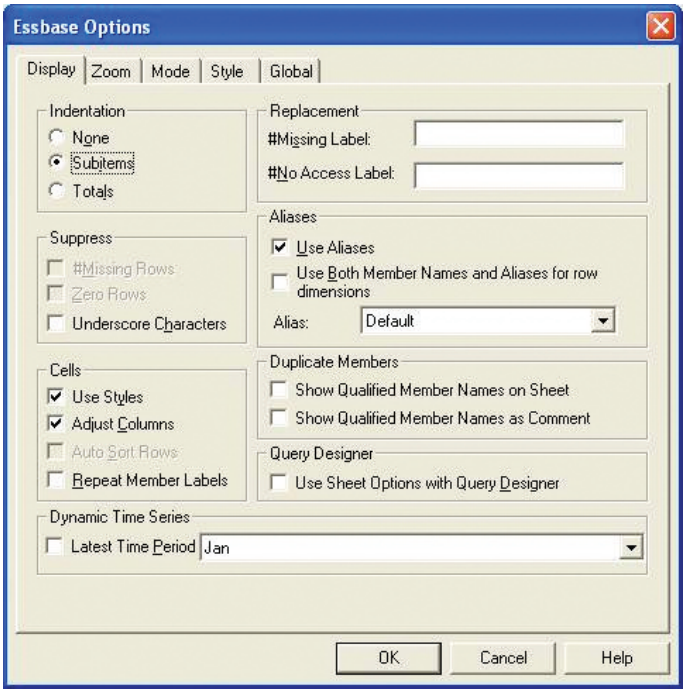
	A	B	C	D	E	F	G	H	I	J
1			Years	Time	Transaction Type	Payment Type	Promotions	Age	Income Level	Product
2	[Geography].[Central]	Measures	#Missing							
3	Mid West	Measures	#Missing							
4	Connecticut	Measures	1014055.25							
5	Massachusetts	Measures	1870116.75							
6	Maine	Measures	1237606.25							
7	New Hampshire	Measures	412046							
8	New Jersey	Measures	1453884.5							
9	New York	Measures	5193481.5							
10	Pennsylvania	Measures	5225704.25							
11	Rhode Island	Measures	245864							
12	Vermont	M								
13	[Geography].[North East]	Measures								
14	South	Measures	13198373.5							
15	South West	Measures	#Missing							
16	West	Measures	#Missing							
17	Geography	Measures	30626079.25							

<esskey>[Geography].[North East]</esskey>

You can control how the Spreadsheet Addin displays member anems in the Essbase Display Option dialog

EFFECT OF NON UNIQUE OUTLINES ON END USER ACTIONS

and dimension helps. Because the Spreadsheet Addin exposes the most robust methods for adhoc querying of an Analytic Services database, this interface exposes labels to help users more easily navigate. It can show you a qualified path for any member in the cell comments or on the sheet.



QUERY REQUESTS

Interactive query requests are not much different from outline navigation because the tools will generate the query request. Scripted queries have to have qualified member names to remove ambiguity. The good part is that the scripted query will never be broken by a transformation. Moving members will affect non unique outline scripts more since the qualified member name is changed. Calc scripts will behave like scripted queries.

PERFORMANCE OF QUERY REQUEST

Performance of finding a member is dependent on the length of the qualified path. Member lookup of composite member name [Market].[US].[New York] takes about twice the time of a unique New York. Note the member lookup is only a portion of the total query time.

There is APIs for tools to use to get an internal member ID. This member ID will be a faster way to lookup members. Note internal IDs are available for the API but they are not persistence across member loads they can change.

Performance of interactive queries should be similar to unique outlines if optimized internal member ids are used.

Scripted queries can not use internal member id since they are not guaranteed to be persistent and have a performance slowdown based on the number of levels

- Duplicate outline cubes can be summarized as:
- Names are short and look good
- Names are not unique more information needed in the GUI
- Script need to use qualified member names
- Scripted queries will have performance slowdown based on the number of levels

MISCELLANEOUS

Drill Through

Integration Services drill through is not changed for the end user, but the SQL generated is different to handle non unique member names. It will contain more columns in the where clause and will require more diligence in creating SQL override syntax.

PRODUCT INFORMATION TEL: 1.800.286.8000 (U.S. ONLY)
CONSULTING SERVICES E-MAIL: NORTHAMERICAN_CONSULTING@HYPERION.COM TEL: 1.203.703.3000
EDUCATION SERVICES E-MAIL: EDUCATION@HYPERION.COM TEL: 1.203.703.3535
WORLDWIDE SUPPORT E-MAIL: WORLDWIDE_SUPPORT@HYPERION.COM

PLEASE CONTACT US AT WWW.HYPERION.COM/CONTACTUS FOR MORE INFORMATION.

HYPERION SOLUTIONS CORPORATION WORLDWIDE HEADQUARTERS
5450 GREAT AMERICA PARKWAY, SANTA CLARA, CA 95054
TEL: 1.408.744.9500 FAX: 1.408.588.8500