ESSBASE ASO TUNING AND OPTIMIZATION FOR MERE MORTALS

Tracy McMullen, interRel Consulting

Essbase aggregate storage databases are fast. Really fast. That is until you build a 25+ dimension database with millions of members. In that case, you might see some slow performance. So how can you get the best performance out of your aggregate storage databases? In this whitepaper, we’ll provide a number of tips and tricks for tuning ASO so that you can achieve the best performance possible.

Dimension and Hierarchy Settings

In our block storage databases, we take careful time to consider which dimensions should be dense and sparse and when dimensions should be tagged “Account” or “Time”. While dense and sparse concepts do not apply in ASO, the assignment of a stored vs. dynamic hierarchy is important to the performance of your ASO database. Choosing which dimensions should be Account or Time or attribute type is another important consideration in your ASO design.

Stored versus Dynamic versus Multiple Hierarchies

Which dimensions should be stored versus dynamic versus enabled for multiple hierarchies? Does it matter? Yes! If you tagged every dimension dynamic, you would probably have a very slow database. Tag dimensions as stored whenever possible. Definitely use stored hierarchies for your really big dimensions (the ones with thousands and millions of members). Stored dimensions will perform a straight aggregation of members. Only the plus consolidation tag is allowed and no member formulas. Also note dimensions tagged “Accounts” cannot be stored.

Use the dynamic hierarchy setting for dimensions and hierarchies that need to be calculated and not aggregated. Dynamic hierarchies allow multiple consolidation symbols and member formulas written in MDX. All calculations are performed at retrieval time and are not part of any aggregate view selection.

Dimensions enabled for multi-hierarchy can contain hierarchies that are either stored or dynamic. Multi-hierarchy enabled dimensions allow you to build alternate rollups with shared members. Non shared members must occur in the outline before the shared members and the first hierarchy in a multi-hierarchy enabled dimension cannot contain shared members. By enabling multiple hierarchies, you can effectively use stored members and dynamic calculated members when needed.

Accounts

Why should a dimension be tagged “Accounts” in aggregate storage databases? The answer to this question differs depending on your Essbase version. If you use an Essbase version prior to 9.3, the Accounts dimension is the dimension used for compression. (Beginning in 9.3.1, you can choose other dimensions as the compression dimension.) The Accounts dimension is the compression dimension by default in all versions. The Accounts dimension is a dynamic dimension that allows non additive unary operators.

If you are using Essbase 11, the main reason to tag a dimension Accounts in ASO is to take advantage of time balance features (time balance first, last or average for an account). If you choose to make another dimension Accounts, you can still achieve “Time Balance” functionality via analytic dimensions.

Accounts dimensions in block storage databases allow a feature called expense reporting tags. This feature is not available for aggregate storage databases. If you want to build this logic into aggregate storage databases, simply use UDAs and member formulas to achieve the same result.
If your accounts/measures dimension in your aggregate storage database contains a large number of members, you may want to consider not tagging the dimension as Accounts. By making the accounts dimension just a regular stored dimension that is aggregated, you might find some big performance improvements. But you don’t want to add expenses to revenue! The way around this is to use UDAs to flip signs during data loads (under Options >> Data Load Settings within a rules file):

The caveat is the user will have to understand some signage differences. The resulting outline would look like the following:

Another alternative for large account dimensions would be to use multiple hierarchies, allowing for more aggregates to be stored. Place the aggregate members in a stored hierarchy and an alternate rollup with shared member(s) to use non + consolidation tags:

**Time**

Time is a good candidate for compression (more on compression in just a moment). If you don’t use the Time dimension as the compression dimension, make it stored or use multiple hierarchies if member formulas are required.

**Attributes**

Don’t forget about attribute dimensions in aggregate storage databases. Even though we can build a database with 20+ dimensions doesn’t mean that some of those dimensions wouldn’t be better served as attribute dimensions. Retrieval time is similar to other hierarchy members and internally they are treated as a shared roll-up. Attribute dimensions can be used as part of Query Tracking. One big benefit (depending on your requirements) is that attribute dimensions are not displayed by default in adhoc queries using Smart View, Web Analysis or other tool. So from an end user perspective, their view isn’t “cluttered” with a long list of dimensions. Here is the default query for ASOSamp.Sample database in Smart View (notice no attribute dimensions):
The easiest way for a user to query an attribute dimension is using the Smart View Query Designer:

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Years</td>
<td></td>
</tr>
<tr>
<td>Measures</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

POV Sheet1 - Query_1

Columns
- Years
- Measures
- POV
- Age
- Geography
- Income Level
- Payment Type
- Products
- Promotions
- Stores
- Time
- Transaction Type

Attributes
- Store Manager - Stores
- Square Footage - Stores
- Area Code - Geography
```

Remember varying attribute dimensions (new feature in Essbase 11) are not available for aggregate storage databases so if an attribute needs to vary over Time or other dimension, make it a regular dimension.
Compression

Now that we’ve covered some design tips and tricks for dimensions and hierarchies, let us turn our attention to compressing ASO databases.

You can decrease the size of the aggregate storage database which improves performance (smaller databases, faster queries) using a feature called compression.

To compress an ASO database, you simply tag a dimension to be the “compression dimension”. In previous releases, compression was tied to the Accounts dimension and is the default compression dimension in current versions. Beginning in version 9.3, any dynamic dimension can be used for compression. The compression dimension influences the size of the compressed database. It is not mandatory but helps overall performance. The goal is to optimize data compression but maintain end user retrievals.

Compression Dimension Rules

- Must be a dynamic dimension
- Must contain a single dynamic hierarchy
- No attributes
- No associated base members for attribute dimension

Evaluate and Set the Compression Dimension

The cool thing about ASO compression is that Essbase helps you choose a compression dimension by estimating what the database size would be depending on which dimension is tagged for compression.

To evaluate and or select a compression dimension,

1. Right click on the database and select Select compression dimension:
2. The available dimensions will display with the expected level zero database size.
3. Note Measures is selected as the compression dimension and the expected level 0 size is significantly smaller than the other dimension alternatives:
4. If you’d like to change the compression dimension, double click on the desired dimension.
5. Click OK.

Another way to view compression settings and get more detailed information is under the database properties.

1. Right click on the database and select Edit Properties.
2. Select the Compression tab:

Notice we see the same level 0 database size displayed in MB but we also see a number of other metrics that help us in our evaluation of the best dimension for compression. The Stored Level 0 Members displays the number of stored level zero members for the dimension (imagine that). You want this number to be low. Dimensions with many level zero members do not perform well when tagged as Compression.
Compression is more effective if values are grouped together consecutively versus spread throughout the outline with missing in between. Essbase creates a bundle of 16 data values. The *Average Bundle Fill* is the average number of values stored in a group or bundle. The number will be between 1 and 16 where 16 is the best average bundle fill value.

*Average Value Length* is the average storage size in bytes required for the stored values in cells. The average storage size will be between 2-8 bytes, where 2 is the targeted value. Without compression, 8 bytes is required to store a value in a cell. With compression, it takes few bytes and the smaller the amount the better. Rounding data values can help compression as it reduces the average value length.

The compression dimension can also be specified in the outline editor.

1. Open the outline editor.
2. Right click on a dimension and select *Edit member properties*.
3. Select the Compression type to be *True*.

Changing compression requires a full restructure which could take some time. Perform this change during off hours when users are not in the system.

**Compression Tips**

If you are using compression, make sure to consider these tips. Ideally the compression dimension should be the column headers of your data source file. Time is an excellent candidate for compression dimension, especially if you have fiscal year as a separate dimension. The best compression is achieved when the leaf level member count is evenly divided by 16. And finally watch out for a compression dimension with many level zero members. This could negatively affect performance.

**Faster Data Loads**

ASO and BSO databases share some commonalities in the data load process: data sources include both text files and relational databases; the rules files creation process is same; and one or more data sources can be loaded at the same time and those sources can be loaded with or without rules files depending on the data layout. While these concepts hold true for both BSO and ASO, the way the data is stored in ASO is very different than BSO (tablespaces versus page and index files). And more importantly, your aggregate storage databases are going to be much bigger than BSO databases with millions of members and 20+ dimensions. This means you are going to be loading some really big files. Because we have this requirement of loading large volumes of records, the ASO data load process provides buffering and parallel capabilities not available for block storage databases.

**Load Buffer for Multiple ASO Sources**

Because the data files are potentially very large for aggregate storage databases, a temporary load buffer is used for loading multiple files (loading a single load file does not involve the buffer).

In Administration Services, when you load multiple sources in a single step, the load buffer is automatically used. Two additional options to define include *Aggregate Use Last* and *Resource Usage*:
ASO Tuning for Mere Mortals

Aggregate Use Last option tells Essbase what to do with multiple values for the same cell. By default Essbase will sum up the values in the load buffer for each cell. However, if you requirements dictate, you can check Aggregate Use Last to load the value of the cell that was loaded last into the load buffer.

The Resource Usage option controls how much cache a data load buffer can use. This setting is defined in a percentage between .01 and 1. The total of all data load buffers created on a database cannot exceed 1 (so if you are performing multiple loads concurrently don’t set all of them to use 100% of the cache).

You can also use the load buffer through MaxL statements. When you load multiple files, the first step in the process will be to initialize the load buffer to accumulate the data. Next, Essbase reads in the data sources into the load buffer. Finally the data will be loaded from the load buffer into the database.

```maxl
/*Initialize the load buffer*/
alter database ASOSamp.Sample initialize load_buffer with buffer_id 1
    resource_usage .5;

/*Read the data sources into the load buffer*/
import database ASOSamp.Sample data from server data_file 'file_1' to load_buffer
    with buffer_id 1 on error abort;
import database ASOSamp.Sample data from server data_file 'file_2' to load_buffer
    with buffer_id 1 on error abort;
import database ASOSamp.Sample data from server data_file 'file_3' to load_buffer
    with buffer_id 1 on error abort;

/*Load data from the buffer into the database*/
import database ASOSamp.Sample data from load_buffer with buffer_id 1;
```

The import statements do not need to be continuous. As long as the buffer exists, the database is locked from queries, aggregations and data loads by other means.

Presorting multiple data source loads will improve performance even though the buffer always sorts: A sort-sort-merge-sort will be faster than a nonsort-nonsort-merge-sort. You won’t see as dramatic a difference as when you apply sorting in BSO loads, but it is still important on very large loads and time critical incremental loads.

**Concurrent Loads**

Multiple data load buffers can co-exist for the same application which means you can load data into multiple buffers at the same time, giving us yet another way to speed up data load timing. This method is called concurrent loading. Using separate MaxL sessions, you load data into the Essbase database with different load buffers. Once the data loads are complete, you commit the multiple data load buffers in the same operation (which is faster than committing each buffer by itself).

Here’s an example:

MaxL Session 1:
**ASO Tuning for Mere Mortals**  
By McMullen

```
alter database ASOSamp.Sample
initialize load_buffer with buffer_id 1 resource_usage 0.5;
import database ASOSamp.Sample data
from data_file "dataload1.txt"
to load_buffer with buffer_id 1
on error abort;
```

MaxL Session 2:

```
alter database ASOSamp.Sample
initialize load_buffer with buffer_id 2 resource_usage 0.5;
import database ASOSamp.Sample data
from data_file "dataload2.txt"
to load_buffer with buffer_id 2
on error abort;
```

When data is fully loaded, use one commit statement to commit all of the load buffers:

```
import database ASOSamp.Sample data
from load_buffer with buffer_id 1, 2;
```

We’ve shared some tips for the full data loads for ASO databases, but what if only a small subset of your data changes? Do you have to reload everything? Let’s next discuss incremental data loads for ASO databases.

**Trickle Feeds & Slices**

In the most recent versions of Essbase, you’ll hear about the new feature “trickle feeds” or near real time data access. What exactly does this mean? Historically, users were not able to access an ASO database when it was being loaded. So in order to load the database, you would have to load in off hours or kick everyone out of the system.

Today is a different story. An aggregate database can store data in multiple “slices”. The main database slice is created when you perform the main data load. This often occurs on a nightly or weekly basis.

With multiple slices, you can load additional slices without impacting the main slice, allowing users to remain in the database. The idea is that you can provide near real time data updates to an ASO database, hence the saying “trickle feeds”. Imagine data is a stream that trickles into your Essbase ASO database. An incremental data load will perform much faster than reloading the entire record set. You can then merge all incremental slices into the main database slice or merge all incremental slices into a single data slice, leaving the main database slice unchanged.

In Administration Services you can load data to a new slice by checking the `Create New Slice` option:
On a periodic basis, possibly nightly or weekly, you will want to merge the slices into one main slice or merge all incremental slices together into one slice. To merge slices in Administration Services, right click on the database and select *Merge >> All slices into one* or *Merge >> Incremental slices only*.

Dynamic aggregations are performed across the necessary slices to provide accurate query results. Different materialized views might exist within a slice as compared to the primary slice of the database.

One other helpful tip: Using the load buffer during incremental data loads improves performance.

Now you know how to incrementally load data but what if you need to clear the slice before the incremental load? The annual budget is loaded and static while current month actuals are changing on an hourly basis. Your desired process is clear the current month actual data and keep the static budget data. Enter partial data clears.

**Partial Data Clears**

Essbase 11 now allows partial data clear for data sets in an ASO database. For example, our actual data is dynamic and needs to be refreshed often but the budget data is static (no need to reload). Two types of partial clears are available: physical and logical. Physical clears completely remove cells from a database while logical clears remove cells in the database by creating offsetting cells in a new slice.

See the illustrations below to help understand how this feature works:

A physical clear removes the cells from the database.
A logical clear creates offsetting cells in a new slice, a faster alternative to physical clears (in many cases, significantly faster).

What is the impact to upper levels? Essbase recognizes that values are changed and displays correct query results. Essbase doesn’t rematerialize the aggregation: you have to reprocess aggregations.

You will use a combination of MaxL and MDX to clear a defined set of data. The region of data to be cleared will be identified using MDX:

```
CLEAR {Jan, Feb, Mar}
```

A new MaxL command was introduced in version 11 to clear a region. The following statement logically clears a region identified by ‘MDX-SET-EXPRESSION’:

```
ALTER DATABASE DBS-NAME CLEAR DATA IN REGION ‘MDX-SET-EXPRESSION’;
```

The following statement physically clears a region identified by ‘MDX-SET-EXPRESSION’:

```
ALTER DATABASE DBS-NAME CLEAR DATA IN REGION ‘MDX-SET-EXPRESSION’ PHYSICAL;
```

There is a limit to the number of slices created for an ASO database (remember, logical clears create additional slices). The recommended process is to run the physical data clear at night to clean up the new slices and re-optimize the cube.

What is the impact to query performance when using partial clears? After the immediate clear, both options are a bit slower. Logical clears are slower because you have to combine two slices to view the correct data set. Physical clears result in slower performance too because you haven’t processed any aggregations. After the materialization of aggregation, physical clears will always result in faster query performance.

You now have the tools for faster performance in ASO. Get busy with your ASO cubes!

---

**About the Author**

Tracy McMullen, Oracle ACE Director, is the co-author of the best-selling Hyperion book “Look Smarter than You are with Essbase.” She is a Director at interRel Consulting, a Hyperion-specific consulting firm. She is a Hyperion Certified Consultant for Essbase, Certified Solutions Architect for Hyperion Planning, and a Certified Project Manager. Tracy has been leading implementations of EPM and Data Warehousing systems for over 10 years.