Creating Hybrid Relational-Multidimensional Data Models using OBIEE and Essbase
by Mark Rittman and Venkatakrishnan J

ODTUG Kaleidoscope Conference June 2009, Monterey, USA

Oracle Business Intelligence Enterprise Edition Plus (OBIEE+) is Oracle's suite of business intelligence reporting tools and servers, providing access to data across the organization through web-based dashboards, alerts and reports. One of the key new features introduced in OBIEE+ version 10.1.3.3.2 was support for Oracle Essbase as a data source, allowing users to report on data taken from multi-dimensional databases along with their existing data in relational databases.

Whilst data sourced from Oracle Essbase can be reporting on in isolation, a key benefit of this new feature is the ability it provides to create hybrid data models including data from both multidimensional and relational data sources. This feature allows developers, for example, to create logical fact tables that includes historic data source from a relational data source, and forward-looking forecast data sourced from Oracle Essbase. An Oracle Essbase cube containing aggregated data could be used as a summary layer for detail-level relational data, or a relational database could be used to add textual attribute data to an Oracle Essbase cube containing just financial data.

In this paper, we will set out how OBIEE+ accesses Oracle Essbase data, and the features and limitations in the current 10gR3 implementation of this feature. We will then set out several scenarios where relational and Essbase data is combined into a single logical business model, using data sourced from the Sales History sample schema that ships with recent versions of the Oracle Database.

How Oracle Business Intelligence Enterprise Edition Supports Essbase as a Data Source

Oracle BI Enterprise Edition Plus comprises of a number of server applications that provide access to heterogeneous data through a number of interfaces. A central part of OBIEE+ is the Oracle BI Server, a component that provides connectivity to a number of data sources, a metadata model over the various data sources, calculation and aggregation capabilities together with a security model that can work with corporate directories such as LDAP and Microsoft Active Directory. The metadata layer provided by the Oracle BI Server creates a logical dimensional model over all of the organization's data sources, allowing users to access their data as facts and dimensions regardless of the structure of their underlying data.

As well as supporting relational and XML data sources, the Oracle BI Server supports a number of multi-dimensional Online Analytical Processing (OLAP) servers such as Microsoft Analysis Services, Oracle OLAP, SAP B/W and Oracle Essbase. Connectivity to these OLAP data sources is provided through native protocols and languages such as XML/A, SQL and MDX, with the Oracle BI Server then combining this data with relational data sources to provide a single view of all the organization's data.
Access to Oracle Essbase data is provided through the Essbase Client C API, which needs to be installed on the server(s) running Oracle BI Server. The Oracle BI Server then connects through this API to the Essbase server, initially reading the Essbase database outline into the Oracle BI Server's metadata layer, and then taking queries from users against the Oracle BI Server's logical dimensional model and turning them into MDX queries against Essbase where appropriate.

As such, Essbase data can be accessed through Oracle BI Enterprise Edition in three main ways:

- Standalone logical dimensional models that source their data solely from Essbase can be created in the Oracle BI Server
- Hybrid models that take their data from both Essbase and relational sources can also be created, or
- Essbase can be accessed directly, without going through the Oracle BI Server, if users still wish to use tools such as Oracle Hyperion WebAnalysis, Oracle Hyperion Financial Reporting and the like.

Oracle Essbase can also, through the Essbase Studio utility included with Oracle EPM Suite 11.1, access data in the Oracle BI Server metadata layer in order to build and populate cubes. This new feature makes it possible for Essbase developers to load their cubes from the pre-integrated and transformed data provided by the Oracle BI Server metadata layer, rather than having to go back to individual data sources to find the data they need.

Oracle Essbase can therefore be both a data provider and data consumer for the BI Server, and we will investigate both scenarios in this paper.

Although the metadata layer created by the Oracle BI Server generally supports queries using the Oracle BI Server's own dialect of SQL, it is possible to pass-through function calls to the Essbase server using the `EVALUATE` and `EVALUATE_AGGR` Oracle BI Server functions. This feature allows the developer to make use of more complex and powerful MDX functions when working against an Essbase data source, and this paper will set out an example of how these features can be used. The Oracle BI Server also automatically translates it's own `AGO` and `TODATE` "time-series" functions into the correct Essbase MDX syntax, so that the full functionality of the Essbase server can be used to resolve complex calculations such as these.

Some features of Oracle Essbase however, as you will see in this paper, are not yet fully supported when accessed through the Oracle BI Server. Complex hierarchy types such as ragged hierarchies and unbalanced hierarchies are only partially supported, and filters created against Essbase data have to be replaced with Oracle BI Server filters operating against the overall logical dimensional model. Most
Essbase functionality can be used though, and the federated query feature of the Oracle BI Server provides many new opportunities to supplement the largely numeric and statistical data in an Essbase cube with richer, more descriptive data from relational sources. It should also be noted that as the Oracle BI Server communicates with Oracle Essbase via the Essbase C API, it does not therefore support Oracle Providers Services, and customers who wish to set up Oracle Essbase in a clustered, high-availability architecture will need to use alternative methods to achieve this goal.

In the following examples, we will initially look at creating a new business model within the OBIEE+ metadata using data sourced from an Oracle Essbase multi-dimensional database. Subsequent examples will take this model and enhance it the addition of data sourced from a relational database, to add new measures and to provide drill-through capabilities to transaction-level data.

**Importing Essbase Cube Metadata and Creating a Simple Logical Business Model**

When working with mixed multi-dimensional/relational business models, you can either start by importing relational data into the Oracle BI Administrator tool and then adding Essbase data afterwards, or start by importing Essbase data and then adding relationally sourced data to it. As you will see in this example, it is considerable easier to start with Essbase data and create the initial business model from this, whilst manually adding Essbase-derived metadata to an existing business model is relatively difficult and not recommended by Oracle Corporation. As such, in this first example, we will start by importing the Essbase database and add our additional relational data to it in later stages.

The first step in creating a business model using Oracle Essbase data is to load your repository, and then select **File > Import > From Multi-dimensional** from the application menu to start the import process.

A dialog will then be shown that prompts you to enter the server name, user name and password for an Essbase connection. These details will then be passed to the Essbase client software that you will also need to have installed on the PC running the Oracle BI Administrator application, as well as the server running Oracle BI Server.
Once the client software connects to the Essbase server, a list of Essbase applications and databases are displayed. Press **Import** to start the process of reading from the Essbase database **Outline**.

When the import process completes, you will see that a new database will have been added to your physical databases in the Oracle BI Administrator tool. If you expand this new database, you will see a level-based representation of the dimensions, hierarchies and measures detailed in your Essbase database.
There are some points to note at this stage. Firstly, even if your Essbase database's account dimension contained a nested hierarchy, all of the measures within the equivalent database within the BI Administration tool will be flattened. It is possible to retain the hierarchy within an account dimension, however this involves designating one of the other dimensions within your database as being the account dimension that may, or may not be a suitable solution. In addition, whilst it is possible to import ragged and unbalanced hierarchies from Essbase databases into the Administration tool, you will need to manually edit the dimension to set it as ragged/unbalanced. We will look at the effect of unbalanced and ragged hierarchies later in this paper, particularly in respect of federated queries.

Once you have created the physical model of your Essbase database, you can then drag the database into the Business Model and Mapping Layer of your metadata, and then drag this into the Presentation layer to create a new subject area for users to access.
You now have some basic data, based on data in an Oracle Essbase database, which you can use to report against. Note that if you view the properties of the measures in your new fact table, they are set to `AGGR_EXTERNAL` as their aggregation method, as the Oracle BI Server will obtain aggregates as needed from the Essbase server, rather than try and calculate them itself.

So now that you have your basic Essbase cube contained within the Oracle Business Intelligence metadata layer, you can start to create more complex models using data taken from both relational and multi-dimensional sources.

Creating a Federated Logical Model with Data from Essbase and Relational Sources

One of the most powerful capabilities of the Oracle BI Server is its ability to create federated business models. Federated models take data from two or more physical data sources but present them to the user as a single, combined model with joins taking place in the background to bring the data together. This capability is also extended to non-relational data sources such as the Oracle Essbase multi-dimensional database, as the import process that brings Essbase data into the Oracle BI Administrator physical database layer creates a relational view of the Essbase dimensions that can be joined, in the Oracle BI Server's memory, to the relational data source.

Typically, when working with combined (or hybrid) multi-dimensional and relational data sources, there are two main use cases where this feature is useful:

1. You have a single measure that is stored, in aggregated form, in an Essbase database, and you want to provide drill-through capabilities in Oracle BI Answers in order that users can report on data at more detailed level. This requires you to configure the business model and mapping layer so that queries for the measure are sent to either the Essbase database, or a relational database, depending on the level of detail in the report.

2. You have a logical fact table that needs to show both data from an Essbase database, for example forecast data or budget data sourced from a planning application, alongside historical data held in a relational database, with the Oracle BI Server making in-memory joins between the two data sources based on conforming common dimensions.

In this next example, we are going to enhance the Essbase-derived business model created in the previous example, and initially map in data from another, relational data source so that users can drill down to more detail than is held in the original data source. Later on, we will use additional measures only present in the relational data source to add to the existing Essbase-derived measures in the
business model.

Before you try to set a model like this up, you should however be aware of certain rules that you (and the data sources) should follow:

1. Firstly, for the dimensions that are conforming (i.e. the ones you wish to join the data sources on), the dimension hierarchies on the Essbase side should not be ragged or unbalanced. To understand why this is the case, you need to consider how the Oracle BI Administrator tool creates a relational representation of the hierarchies that are imported into the physical model layer of the OBIEE+ metadata; when the import process happens, physical cube columns are created to represent each of the Essbase generations that come across as part of the Essbase outline, and if your dimension hierarchy is balanced, all of the customer names for example, will be stored in the same generation and will therefore be held in the same physical cube column. If your dimension hierarchy is unbalanced, however, customer names may well be found in more than one dimension hierarchy generation, which will mean that they will be stored in more than one physical cube column. Because of this, there will not be a single, consistent physical cube column to join your relational column to, and federation will not therefore be possible.

2. Secondly, common, conforming dimensions should exist between the Essbase source and the relational source, and as noted above, these common dimensions should have a common level at which a join can be made between Essbase and the relational source.

**Use Case #1: Providing Drill-Through To Transactional Level Via A Relational Data Source**

Before we start combining these data sources, first take a look at how the Oracle BI Administrator tool has set up the logical table source for the logical fact table. As you can see in the screenshot below, our Essbase-derived logical table source has been assigned to specific levels in the Essbase dimensional hierarchy.

![Logical Table Source](image)

The Oracle BI Server assumes that for all the levels above, the same logical table source would be used for bringing the data back.

You should also ensure that, since a relational and a multi-dimensional data source are being used for federation, the measures in the logical fact table should have an explicit aggregation rule that is supported by both the sources, such as the **SUM** rule shown in the screenshot below.
In this example, we have a relational data source that is also dimensioned by Channel, Customer, Promotion, Time and Product, but the Customer data is stored at individual customer level rather than the city level that the Essbase data is held at. In addition, the relational source contains two measures, Unit Cost and Unit Price, that are only dimensioned by Channel, Promotion, Time and Product (i.e. not Customer) and that are not present in the Essbase database. As such, the process for incorporating the relational data into the Essbase-derived business model and mapping layer is as follows:

1. Import the relational data into a new Physical Layer database model
2. Map in the relational dimension tables into the existing business model logical tables, creating new logical table sources, where there is an exact match between the source (i.e. all dimensions except the customer dimension)
3. Then map in the relational customer dimension, adding new logical columns for the detail-level customer information
4. Adjust the logical fact table logical table source mappings to take advantage of the new detail-level fact data coming from the relational source, and
5. Add the two new measures from the relational source, connecting them to all of the existing logical dimensions except the customer one

To start this process, begin by importing the relational database into the physical layer of the OBIEE+ metadata, and then start adding a new relational logical table source to each of the dimensions. Then, starting with the Channel physical table, right-click on it and select New Object > Logical Table Source, name it, then select the Column Mapping tab and map the physical relational columns to the corresponding logical dimension columns, which will have been named after the generations found in the original Essbase database, as shown in the screenshot below.
Once this step is completed, there is no need to change the logical keys or the dimension hierarchy for this logical dimension table as both the relational and Essbase dimensions are held at the same level of detail. Note that in Oracle Essbase, the topmost Generation always contains a single member that has the same name as the actual dimension; since relational sources do not have this level, use the Expression Editor to set the relevant entry for the relational local table source topmost level.

Then, repeat this same procedure above for the *Product* logical dimension table, like this:

Following this, create corresponding entries for the *Promotion* dimension.

Now that you have configured the “regular” dimensions, the next step is to map the *Time* dimension.

Time dimensions in Essbase usually differ from those stored in relational databases because Essbase normally requires that all member names in a dimension be unique, which causes an issue for month and quarter names, for example, which are the repeated for each year. To address this issue, Essbase time dimension member names are usually prefixed with year and quarter identifiers making each
member name unique, an example of this being is shown in the screenshot below.

Contrast this with a time dimension stored in a relational table, which does have this requirement for uniqueness:

<table>
<thead>
<tr>
<th>CALENDAR_YEAR</th>
<th>CALENDAR_QUARTER_DESC</th>
<th>CALENDAR_MONTH_NAME</th>
<th>DAY_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1998 1998-01</td>
<td>January</td>
<td>Thursday</td>
</tr>
<tr>
<td>2</td>
<td>1998 1998-01</td>
<td>January</td>
<td>Friday</td>
</tr>
<tr>
<td>3</td>
<td>1998 1998-01</td>
<td>January</td>
<td>Saturday</td>
</tr>
<tr>
<td>4</td>
<td>1998 1998-01</td>
<td>January</td>
<td>Sunday</td>
</tr>
<tr>
<td>5</td>
<td>1998 1998-01</td>
<td>January</td>
<td>Monday</td>
</tr>
<tr>
<td>6</td>
<td>1998 1998-01</td>
<td>January</td>
<td>Tuesday</td>
</tr>
<tr>
<td>7</td>
<td>1998 1998-01</td>
<td>January</td>
<td>Wednesday</td>
</tr>
<tr>
<td>8</td>
<td>1998 1998-01</td>
<td>January</td>
<td>Thursday</td>
</tr>
<tr>
<td>9</td>
<td>1998 1998-01</td>
<td>January</td>
<td>Friday</td>
</tr>
<tr>
<td>10</td>
<td>1998 1998-01</td>
<td>January</td>
<td>Saturday</td>
</tr>
<tr>
<td>11</td>
<td>1998 1998-01</td>
<td>January</td>
<td>Sunday</td>
</tr>
<tr>
<td>12</td>
<td>1998 1998-01</td>
<td>January</td>
<td>Monday</td>
</tr>
</tbody>
</table>

We will therefore have to take an extra step to make the time dimension entries match across our relational and Essbase sources. To accomplish this, map your logical table source columns as before, but use the Expression Editor to add the necessary year, quarter and month prefixes so that member names match the relevant formats in the Essbase data source, as shown in the screenshot below:

The last dimension that is left is the Customer Dimension, as bringing in the more detailed relational logical table source involves changes to the hierarchy as well as logical keys.

In the first of the use cases detailed earlier in this paper, we wish to configure the business model to support drilling, via Oracle BI Answers or any tool that uses the Oracle BI Server as a data source, from the aggregated data provided by the Essbase source to more detailed-level data provided by the
To make this possible, we first have to identify the granularity of each customer dimension source, which on the Essbase side goes down to city level whilst on the relational side goes down to *Individual Customer* level. As such, the Essbase data source contains sales data aggregated at city level whilst the relational source contains the individual transactions, and in addition the more detailed customer dimension information provided by the relational source also provides additional customer attributes such as the *Customer Name*, *Customer Email Address*, *Customer Credit Limit* and so forth. We would want these additional attributes to be available for users to include in reports, and to be displayed when users drill-down in their reports beyond the city level.

To enable this drill, we first add a new logical table source to the logical *customer* dimension table, and then select the **General** tab in the **Logical Table Source** dialog to map the new source to two joined relational tables that contain details on the *customer* and the *country* they are located in, respectively.

![Logical Table Source - CUSTOMERS](image)

Next, to hold the new dimension level attributes that will come from the relational source, we create four new logical columns to hold these attributes:

1. Customer ID
2. Customer Name
3. Customer Email,
4. Customer Credit Limit

These new columns are then mapped to the relevant physical columns in the relational physical table.

![Logical Table Source - CUSTOMERS](image)

After you have closed the **Logical Table Source** dialog, double-click on the *Customer* logical dimension table and select the **Keys** tab, and then include the new *Customer ID* column in the key so that we can use the new relational columns as the lowest level in the hierarchy, as shown in the screenshot below.
Once the logical key has been modified, we then create a new leaf level in the Customer dimension hierarchy, in to which we will drag and drop the new customer attributes, as shown in the screenshot below.

This new leaf level will have two logical keys, as shown in the screenshot below:
One of these keys will be the actual *level key* which is unique to the level and will use the *Customer ID* column; the other, shown in the screenshot below, will be the actual *drill key* and will be used for displaying in reports when the user drills to detail, and will be used to show customer attributes such as *Customer Name, Customer Email* and *Customer Credit Limit*.

Once this new dimension level has been created, assign it as the logical level for the relational logical table source.
We are now at the point where the measures that were initially provided by the Oracle Essbase data source, and are also provided (at a finer level of granularity) by the relational source, are correctly mapped to their respective dimensions at the correct level of detail. Users can now report on data held in aggregate form in an Essbase database, with the Oracle BI Server switching to the relational data source when reports need to be displayed at transactional level.

Use Case #2: Combining Essbase and Relational Measures in a Single Business Model

Once this initial dimension design is complete, the next step is to configure the measures in the business model to obtain their data from the appropriate relational or multi-dimensional source, bearing in mind the level of aggregation requested by the user. Within a business model, the number of fact tables you need to create is determined by the granularity of the measures you wish to present to users; if all the measures have the same grain then only one fact table is required, but as we will see in the next example, if you introduce new measures that do not have the same granularity as the existing ones within your model, you will need to do add a new fact table to hold them.

In the initial business model, based off of data sourced from our Essbase database, we had two measures that were dimensioned by all of the five dimensions in the database:

1. Quantity – This measure is present in both the Essbase, and the new relational source, albeit stored at a greater level of customer granularity in the relational source
2. Amount – Again, stored in both the relational and Essbase source, with the relational source storing data down to the individual customer (transaction) level

However the new relational source that we are working with has two additional measures not found in the Essbase data source:

3. Unit Cost - This measure has its source only from the relational database and is dimensioned by all the dimensions except the customer dimension.
4. Unit Price - This measure has its source only from the relational database and is dimensioned by all the dimensions except the customer dimension.

This difference in granularity between the two sets of measures leads us to a situation where two separate fact tables will need to be created in the business model, with one dimensioned by five dimensions and one by four. Quantity and Amount would be grouped under the Sales Fact table. Unit Cost and Unit Price would be grouped under the Costs fact table, as shown in the screenshot below.
Once the fact tables have been created, the next step is to ensure that each of the metrics are mapped to their corresponding fact sources. Since *Quantity* and *Amount* derive their values from both Essbase and the relational database, we bring in both the sources as logical table sources to this fact table. Then, we start the process of mapping the fact values to their corresponding source columns in the logical tables sources.

The measures that are dimensioned by the *Customer* dimension need mapping in a special way. As we are looking to configure the business model so that queries against the *Quantity* and *Amount* measures use the (aggregated) Essbase data source where possible, but switch to the detail-level relational source when users drill-down to the individual customer level in their report, we do this by specifying the level of *Customer* dimension detail that each logical table source provides.

In the case of the Essbase logical table source for the fact table containing these measures, we map the levels as shown in the screenshot below:

![Essbase Logical Table Source Screenshot](image1.png)

Then, for the relational logical table source for this fact table, we map the levels as shown below:

![Relational Logical Table Source Screenshot](image2.png)

As you see, the primary difference in the level mappings of both the logical table sources lie in the *Customers* dimension. The Essbase source is mapped to Level 5, which is the *City* level, whilst the relational source is mapped to the *Customer* (transaction) level.
The above set of steps will ensure that the sales fact table is set up correctly for relational and multi-dimensional reporting. Configuring the other fact table, the one that is dimensioned by just four of the dimensions and is only available in the relational data sources, is a similar process and just requires mapping to the relational source in the normal manner, as shown in the screenshot below.

Next, we ensure that complex (logical) joins are created properly in the business model and mapping layer, joining the *Costs* fact table to four of the dimensions and the *Sales* one to five.
Then as our final task in the business model and mapping layer, we ensure the aggregation rule for the measures are set to *Sum*.

Moving on to the Presentation Layer, we create a custom subject area and rename all the auto generated “Gen”-format column names to logical names as shown in the screenshot below:
Finally, to test our hybrid repository we save it and open Oracle BI Answers, and create a very simple report that includes **Customer Country**, **Customer City**, **Channel Class** and **Amount Sold**.
Looking at the entries in the NQSQuery.log file after this query executes, you would notice that since the report runs at the **Customer City** level, due to the fragmentation settings the Oracle BI Server obtains its data from Essbase alone, as shown in the log entry excerpt below:

```sql
With
    set [Channel3] as '[Channel].Generations(3).members'
    set [Customers5] as '[Customers].Generations(5).members'

select
    { [Measures].[Amount] }
    on columns,
    NON EMPTY (crossjoin {[Channel3],[Customers5]}) properties ANCESTOR_NAMES, GEN_NUMBER
from [SH.SH]
```

However if the user drills down from one of the cities, for example **Darwin** as shown in the screenshot below:
the log entries now show the relational source being used, as only it holds data down to this level of detail.

```sql
select D1.c1 as c1,
       D1.c2 as c2,
       D1.c3 as c3,
       D1.c4 as c4,
       D1.c5 as c5,
       D1.c6 as c6,
       D1.c7 as c7
from
    (select T1915.COUNTRY_NAME as c1,
         T1927.CUST_CITY as c2,
         T1927.CUST_EMAIL as c3,
         T1927.CUST_CREDIT_LIMIT as c4,
         concat(concat(T1927.CUST_FIRST_NAME, ' - '), T1927.CUST_LAST_NAME) as c5,
         T203.CHANNEL_CLASS as c6,
         sum(T1960.AMOUNT_SOLD) as c7,
         T1927.CUST_ID as c8
      from
        COUNTRIES T1915,
        CUSTOMERS T1927,
        CHANNELS T203,
        SALES T1960
      where  ( T203.CHANNEL_ID = T1960.CHANNEL_ID 
               and T1915.COUNTRY_ID = T1927.COUNTRY_ID
```
As you see in the query generated and the results, for the user the drills appears “seamless”. The Oracle BI Server automatically generates the query against the relational database as it is “level aware”, and for an end-user it looks as though everything is obtained from a single source.

**Using ****EVALUATE** and **EVALUATE_AGGR** **to provide Essbase-Specific Capabilities to Oracle BI Enterprise Edition Plus**

One of the major advantages of Essbase is that it provides around 100 different MDX (“Multi-Dimensional Expression” Language, the OLAP equivalent of SQL) functions that can be used to analyze your data, going beyond the analytic functions normally found in a relational database. As such, users may well wish to take advantage of this functionality then working with Essbase data sources within an Oracle BI Enterprise Edition report.

Oracle Business Intelligence Enterprise Edition Plus makes it possible to leverage Essbase-specific calculations and functionality through its own EVALUATE and EVALUATE_AGGR functions, that pass back Essbase-specific functions to the Essbase server without trying to translate them into generic Oracle BI Server functions. These two functions provide similar features but are used under different circumstances.

1. **EVALUATE** - This function is used for passing back Essbase function names that return dimension members, or their corresponding attributes such as rank.
2. **EVALUATE_AGGR** - This function is used for passing back Essbase function names that return calculated values such as variance.

Note also that at EVALUATE_AGGR cannot be used within Oracle BI Answers, EVALUATE provides the features of this function from within this tool.

As of the 10.1.3.4 release of OBIEE+, there are also certain caveats that you need to be aware of while using these functions:

1. No two columns using an EVALUATE function from the same dimension can be used in a report.
2. Try to define the grain of the MDX query as part of the function itself.
3. If any dimension is neglected in the function then it assumes the topmost node for those dimensions.

Therefore, to avoid surprises and possibly the wrong data being returned, make sure that you qualify the function with the members from all the dimensions if possible.

To understand the capabilities of these functions, we will now go through a number of examples of their use. Each of these examples either uses EVALUATE or EVALUATE_AGGR, and demonstrates the analytical capabilities within Essbase that can be leveraged from OBIEE+

1. Finding the rank of any Product member within the outline (Rank)
2. Finding the number of immediate children any member has in within Customer dimension
3. Calculating Derived Unit Price – As only Sales data is available within the Essbase data source, this metric derives the Unit price using the Sales amount and the Quantity sold, so that Essbase can also provide this metric through its data source.
4. Calculating Customer Spread Amount, using a simple planning calculation that calculates the actual sales data for any Customer member when spread equally across all its children.
5. Calculating the *Maximum Product Amount* that was sold

**MDX Example#1: Product Rank**

In this first example, we will use the MDX **RANK** function to calculate the ranking for a particular product originally sourced from an Essbase database. To create this calculated column, we first create a new logical column within the Product logical dimension table, and then edit the logical table source that points to the Essbase database to specify how it is populated. Using the Expression Editor, we define the column’s data source using the **EVALUATE** function, and example of which is shown below:

```
EVALUATE('Rank(%1.dimension.currentmember,%2.dimension.members)',
    "vejanaki-lap.in.oracle.com"."SH"."SH"."Gen5,Product",
    "vejanaki-lap.in.oracle.com"."SH"."SH"."Gen5,Product")
```

When created, the updated Essbase logical table source would look like the screenshot below:

![Logical Table Source - SH](image)

Looking more closely at the MDX function used within the **EVALUATE** function, you will noticed that it is qualified by the **Product** dimension alone, which means that it would be attributed by Product and any other dimension member chosen in the report. If a dimension does not exist in the report then the top most member of that dimension is assumed inherently by the MDX function.

**MDX Example#2: Number of Immediate Children**

To create a custom member attribute called **Number of Immediate Children** on the custom dimension, again add a new logical column to the Customer logical dimension table, in the same way as the previous example. In this instance though, whilst we will still use the **EVALUATE** function to get data from Essbase, we will use the **COUNT** MDX function to return the number of “children” for a particular dimension member, using the syntax shown below:

```
EVALUATE('Count(%1.dimension.currentmember.children)',
    "vejanaki-lap.in.oracle.com"."SH"."SH"."Gen5,Customers")
```

Again, the screenshot below shows how this function would be defined within the Essbase logical table source.
**MDX Example#3: Maximum Product Sold**

*Maximum Product Sold* is a metric measure that is used for analysis across multiple dimension attributes, and as such you would need to include this measure as part of the fact logical table source. When you create custom aggregate measures in a fact logical table source you should ensure that the `EVALUATE_AGGR` function is used instead of `EVALUATE`, as the former is used for returning aggregated values whilst the latter is used for returning member attributes.

The MDX function (along with the `EVALUATE_AGGR` OBIEE function used to call it) is shown below:

```
EVALUATE_AGGR('Max(%1.members,[Quantity])', "vejanaki-lap.in.oracle.com"."SH"."SH"."Gen5,Product")
```

In the Essbase logical table source, the function would look as in the screenshot below:
**MDX Example#4: Derived Unit Price:**

Derived unit price is an example of an MDX calculation that derives the unit price based on the number of products sold and amount sold, and again as it generates a number that can be aggregated within the Oracle BI Server it needs to be called via the `EVALUATE_AGGR` function. The MDX function (along with the calling `EVALUATE_AGGR` function) is as follows:

```mdx
evaluate_aggr('%1.value/%2.value',
  "vejanaki-lap.in.oracle.com"."SH"."SH"."Amount",
  "vejanaki-lap.in.oracle.com"."SH"."SH"."Quantity")
```

The screenshot below shows this function used with the Essbase fact table logical table sources:
MDX Example#5: Amount Customer Spread:

One of the key benefits of the `EVALUATE` function used against an Essbase data source is that it provides access to the forecasting, allocation and budgeting capabilities of the Essbase server. In our final example, we will create a new measure in the logical fact table that uses Essbase’s allocation functionality to create a new Customer Spread measure that derives the allocated amount that has been sold at the top levels of the Customer dimension. The MDX Function is given below:

```
EVALUATE_AGGR('(%1,Customers.dimension.currentmember).value/
    Count(Customers.dimension.currentmember.children)',
    "vejanaki-lap.in.oracle.com"."SH"."SH"."Amount")
```

Taking a final look at the Essbase logical table source used by our fact table, you can see the three new columns that make use of the `EVALUATE` and `EVALUATE_AGGR` function to leverage Essbase functionality.

### Essbase Data Sources and Time-Series Calculations

Oracle Business Intelligence Enterprise Edition Plus supports time-series queries against data in its metadata layer. Time-series queries typically return values compared to a particular date in the past, or aggregate data from a time in the past to the current date, and are supported in OBIEE+ plus through the AGO and TODATE functions, which perform the following calculations:

1. **AGO** – This function calculates the value of a given measure time shifted by an index.
2. **TODATE** – This function calculates the Period To Date values of a given time member in a time generation.

Using these functions against a relational data source does however often lead to complex and expensive SQL queries.

One of the benefits of working with a multi-dimensional database such as Essbase is that it handles queries such as these “natively” within the OLAP engine. OBIEE+ is aware of this ability and “function-ships” AGO and TODATE calculations to the equivalent Essbase MDX functions. To see how this feature works, we will work through another example, first setting **Time Dimension** property for the **Times** dimension.
Then we include the Level 5 dimension member as the *Chronological Key*. 

Once the Time Dimension property is set and the Chronological Key is created, we can then start creating custom measures using **AGO** and **TODATE**. We create a new custom measure called *Quantity – Ago Level 5* in the *Sales* logical fact table, which calculates the value of the *Quantity* measure time.
shifted by 1 day, using the formula shown below.

\[ \text{AGO}(\text{SH.Sales.Quantity}, \text{SH.Times."Gen5,Times"}, 1) \]

In the same way, we then create a custom measure called \textit{Quantity – ToDate Level 5}. This measure would calculate the Period To Date value of the \textit{Quantity} measure, using the formula shown below

\[ \text{TODATE}(\text{SH.Sales.Quantity}, \text{SH.Times."Gen4,Times"}) \]
To validate the functions created above, we then start Oracle BI Answers and create a report containing the *Day* column, *Quantity* measure and the 2 new measures that we created above. We then also add a filter on the *Day* column to show only 5 consecutive days, as shown below.

**Columns**
Click on column names in the selection pane to add them to the request. Once added, drag-and-drop columns.

<table>
<thead>
<tr>
<th>Times</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Day</td>
<td></td>
</tr>
<tr>
<td>Quantity</td>
<td></td>
</tr>
<tr>
<td>Quantity - Ago Level 5</td>
<td></td>
</tr>
<tr>
<td>Quantity - ToDate Level 5</td>
<td></td>
</tr>
</tbody>
</table>

Display Results  Remove All

**Filters**
Add filters to the request criteria by holding down the CTRL key and clicking on column names in the selection pane.


Save Filter... Remove Filters

Checking the results, the figures look correct.
Taking a look at the query log file to check the MDX that was generated, you would notice that there is a one to one correspondence between the \texttt{AGO} and \texttt{TODATE} functions of the RPD and the MDX functions \texttt{ParallelPeriod} and \texttt{PeriodsToDate}.

With
\begin{verbatim}

   set [Times5] as '{[Times].[01-Apr-1998], [Times].[02-Apr-1998],
   [Times].[03-Apr-1998], [Times].[04-Apr-1998], [Times].[05-Apr-1998]}

   member [Measures].[MS1] as
   'SUM({ParallelPeriod([Times].[Gen5,Times],1,[Times].currentmember)},Measures.[Quantity])'

   member [Measures].[MS2] as
   'SUM({PeriodsToDate([Times].[Gen4,Times],[Times].currentmember)},Measures.[Quantity])'

select
   { [Measures].[Quantity],
   [Measures].[MS1],
   [Measures].[MS2]
} on columns,
   NON EMPTY {{[Times5]}} properties ANCESTOR_NAMES, GEN_NUMBER on rows
from [SH.SH]
\end{verbatim}

This confirms that OBIEE+ is making full of the native time-series functionality in Essbase. You should note however that these functions have more capabilities that are used by OBIEE+ function shipping, and therefore if you wish to make use of these you should use the \texttt{EVALUATE} and \texttt{EVALUATE_AGGR} functions to use them directly.

\textbf{Using Essbase to provide Aggregations for Relational Data}

As well as Essbase being a potential data provider for Oracle BI Enterprise Edition, it can also use the Oracle BI Server metadata layer as a data source for building cubes. This feature became available with the Oracle EPM Suite 11.1 release of Essbase through a new utility, Oracle Essbase Studio, that can extract data from a number of data sources including Oracle Business Intelligence in order to build and populate cubes.

This new feature benefits both Essbase and Oracle BI Enterprise Edition developers in the following ways:

1. Essbase developers can take the pre-integrated data, already organized into logical star schemas, from the Oracle BI Enterprise Edition metadata layer and use it as the data source for a cube, rather than having to go to each individual source which may or may not be organized into a convenient star schema
2. OBIEE developers can use Essbase as an aggregation layer, using the Oracle BI Enterprise Edition metadata layer as the data source for an Essbase cube, then mapping the cube back into the metadata layer at the aggregate level as you would with regular summary tables, but with potentially faster performance and full access to MDX functions.
When extracting data out of Oracle BI Enterprise Edition to create an Essbase cube, data can either be loaded into the cube at the most detailed level that is held in Oracle BI Enterprise Edition (allowing you to create a cube that mirrors the level of detail in the Oracle BI Enterprise Edition logical model), or more realistically it would be loaded in slightly aggregated to reduce the size of the subsequent Essbase cube. In addition, if your Oracle Business Intelligence logical model features fact tables containing measures of differing dimensionality, perhaps because the data they contain is taken from data sources that are federated together, you may end up building multiple Essbase cubes to satisfy different aggregation scenarios.

In this last example, we will take a new Oracle Business Intelligence logical model and aggregate it into one or more Essbase cubes, in order to improve the performance of user queries. This logical model contains a fact table, Items, that is joined to three dimensions called Products, Customers and Times.

The majority of this logical model is mapped to a detail-level Oracle relational database source, so that we can analyze the Quantity measure that it provides by all three dimensions at any level of detail. At some point after the initial model definition, an additional data source was introduced into this logical model, providing a new Quotas measure held at the Product Category and Month dimension hierarchy levels. As such, it can only be analyzed alongside the Quantity measure when data is requested at the Product Category level or higher, and the Month level or higher, and it cannot be analyzed at all when the columns from the customer dimension are added to the request.

Therefore, if we are to extract this data into an Essbase cube, we will need to create two cubes (or “databases”) to this data of differing granularity:

1. The first database will extract Product information from the Product Category level, and time information from the Month level, and will include both the Quantity and Quota measures, aggregating them both up the two dimensions’ hierarchies, and
2. A second database, that will take Customer, Product and Time information from the lowest level of detail, together with just the Quantity measure, and aggregate just this single measure up all of the dimension hierarchies.

Then, when we come to map these two databases into the logical layer of our Oracle BI Server
metadata, we will configure the model such that it uses the first database when a user requests both measures to be aggregated across just the Product and Time dimensions at the appropriate level or higher; otherwise, if a user requests just the Quantity measure to be aggregated across any of its dimensions, the other database data source will be used. If none of these scenarios apply, the original relational data source will be used and aggregated using SQL functions.

To start aggregating your Oracle BI Enterprise Edition data using Essbase, we start the Essbase Studio server application and connect to Essbase Studio console. Next, we create a new data source and select Oracle Business Intelligence as the data source type. Then, we enter the connection details to the Oracle BI Server and select the presentation model that we wish to extract data from, as shown in the screenshot below.

![Connection Wizard](image)

We can now use Essbase Studio to create the required joins between the tables exposed through Oracle Business Intelligence. Note that our Oracle Business Intelligence presentation model will need to expose the columns that act as keys for each of the tables, so that we can recreate the joins. Note also that Oracle Business Intelligence does not expose the dimensions that we may have created in the Business Model and Mapping layer, and we will therefore need to recreate these later on in the process.

The screenshot below shows the Minischema diagram for our OBIEE+ data source, displayed in Essbase Studio.
Now that we have the basic relational model of our Oracle Business Intelligence data within the Essbase Studio repository, we can use this information to start creating the dimensions and hierarchies that our two Essbase databases will use. To start this process, we create Product and Time hierarchies that start from the second-from-bottom level of each dimension, which will be used subsequently to analyze both the Quantity and Quota measures. To accompany these two hierarchies, we create a measure hierarchy that includes both of these measures. Example of these hierarchies are shown in the screenshot below:
We then use the Essbase Studio interface to create and then deploy the two Essbase databases, which we then preview in Essbase Administration Services Console to confirm that the numbers they contain match those displayed in Oracle BI Answers. Once the numbers confirm, we then import the new Essbase databases into our Oracle Business Intelligence metadata physical layer, ready for mapping into our logical model.
Using the mapping features of the Business Model and Mapping layer, we then map the keys from the Essbase physical database model to the corresponding columns in the logical model, starting with the dimension keys and then mapping the measures themselves. Once we have mapped all of the dimension keys and measures, we then edit the new logical table sources created by the mappings and select at what level these new sources apply, in the same way as the earlier examples, as shown in the screenshot below:
This last step is the key to adding Essbase data at the summary level in our logical model, and when users come to create a new query that requests aggregated data for these measures, analyzed by just the product and time dimensions, an MDX query will be sent to the Essbase server, rather than an SQL query to the database, using the same federated query approach that our earlier Essbase-to-relational drill example used.

Logical Request (before navigation):

```sql
-- Logical Request (before navigation):

RqList
  Product.Category as c1 GB,
  Quantity:[DAggr(Items.Quantity by [ Product.Category] )] as c2 GB
OrderBy: c1 asc
```

Sending query to database named w2k3vm (id: <<30675>>):

```sql
-- Sending query to database named w2k3vm (id: <<30675>>):

With
  set [ProductHier2] as '[ProductHier].Generations(2).members'
select
  { [MeasureHier].[Quantity] } on columns,
  NON EMPTY {{[ProductHier2]}} properties ANCESTOR_NAMES, GEN_NUMBER on rows
from [CusApp.CubeFull]
```

However, if the user then includes the customer dimension in the request, the request will **CAST as NULL** the quantity measure as it cannot be sourced from either the Essbase cube we just created, or indeed the original data source that provided the measure. Indeed, as the Essbase cube we just created does not include the *Customer* dimension, even if just this measure was included in the report and the customer dimension was added, the data would be sourced and then aggregated from the original relational source.

To provide aggregation for the *Quantity* measure across the *Customer* dimension as well as the other two dimensions, another Essbase cube will need to be created in Essbase Studio and this will also need to be mapped into the Oracle Business Intelligence logical model. This new cube will take just the *Quantity* measure, but will dimension it by all three dimensions, as shown in the screenshot below:
When generated, we then map this additional database into the OBIEE+ business model, creating additional logical table sources for each logical table that uses measures from this new Essbase datasource.
Then, when a request asks for the quantity measure aggregated over the customer dimension as well as the product and time dimensions, the Oracle BI Server will switch to this alternative Essbase cube to provide the aggregated data.

-------------------
Logical Request (before navigation):

RqList
  Product.Category as c1 GB,
  Times.Year as c2 GB,
  Quantity:[DAggr(Items.Quantity by [Customer.State, Product.Category, Times.Year])] as c3 GB,
  Customer.State as c4 GB
OrderBy: c1 asc, c2 asc, c4 asc

+++Administrator:850000:85000a:---2009/03/05 16:30:05

-------------------
Sending query to database named w2k3vm (id: <<36589>>):

With
  set [CustHier3] as '[CustHier].Generations(3).members'
  set [ProductHier2] as '[ProductHier].Generations(2).members'
  set [TimeHier2] as '[TimeHier].Generations(2).members'
select
  { [QuantHier].[Quantity] }
on columns,
NON EMPTY {crossjoin ({[CustHier3]},crossjoin
({[ProductHier2]},{[TimeHier2]}))} properties ANCESTOR_NAMES, GEN_NUMBER on rows
In this way, therefore, either a single Essbase cube or indeed a collection of Essbase cubes can be used to provide aggregation for an OBIEE+ business model, even if the model contains measures that are of differing granularity and from different data sources. The Oracle BI Server takes care of the mappings between the various data sources, picking the data source that provides results to users in the most efficient way.

**Conclusions**

Oracle Essbase can significantly enhance the functionality of Oracle BI Enterprise Edition Plus, by making available data in multi-dimensional sources and extending the calculation capabilities of the Oracle BI Server. The BI Server can report on Essbase data in isolation within its own business model, or it can combine it with data from relational sources into a hybrid OLAP/relational federated business model. Typical use cases for this type of integration include providing drill-down for Essbase data through a relational transactional database, and creating single business models that combine measures from both OLAP servers and from a relational data warehouse.

The Oracle BI Server connects to Essbase through the Essbase Client API and provides native access to Essbase MDX functions, either through function shipping or through the `EVALUATE` or `EVALUATE_AGGR` functions. Customers taking advantage of this functionality can extend the calculation capabilities of the Oracle BI Server to include native time-series functionality, forecasting, allocations and cross-dimension calculations and comparisons.

Oracle Essbase can also provide another means of aggregating data currently mapped in to the Oracle BI Server semantic model, with the new 11g release of the Essbase Studio utility having the ability to create an Essbase cube directly from the Oracle BI Server semantic model. As such, Oracle Essbase can be either a provider of data for Oracle BI Enterprise Edition Plus or indeed a consumer, with its data being capable of participating in federated queries in the same was as regular relational data.

**About the Authors**

Mark Rittman is Technical Director for Rittman Mead Consulting, a specialist Oracle partner with offices in the UK and USA. Mark is an Oracle ACE Director and a regular writer for Oracle Magazine, Oracle Technology Network and other publications. Mark also co-chairs the ODTUG BI&DW SIG, is the Editor of Oracle Scene, the UK Oracle User Group publication and can be contacted at mark.rittman@rittmanmead.com

Venkatakrishnan Janakiraman is a BI Architect working for Oracle Corporation out of Bangalore, India. Venkat has over six years experience of Oracle BI implementations and training and maintains a blog at [http://oraclebizint.wordpress.com](http://oraclebizint.wordpress.com).