

ADVANCED OLAP – MAKING THE HARD STUFF EASY

Chris Claterbos, Vlamis Software Solutions, Inc.

claterbos@vlamis.com

INTRODUCTION

As Business Intelligence systems gets more integrated in today's businesses many businesses find that they need more complex questions answered. The simple questions such as sales by customer within a region over time are now easy to answer. But now companies want to ask questions about how many unique purchasers do they have on their webstore each month. Did a promotion bring in any new purchasers and did they buy anything after the promotion. While many of today's BI systems can allow for such questions but getting the answers in seconds rather than minutes or hours is a big problem. Throwing more hardware and cache memory is not always the best option.

Using Oracle OLAP to solve some of the advanced analytic problems is easy and straightforward if you know how. This presentation, using examples and actual cases, will show how to solve many of these problems. Three specific problems will be discussed and using a "Cook Book" approach the solution will be presented. These problems include: Unique Counts, Complex Time Series and Measure Hierarchies. Using the solutions presented BI analysts and developers will be able to go back to their companies and apply them to their complex analytic questions easily and improve their BI Solutions with less effort and improved performance.

ORACLE OLAP – WHAT IS IT?

Businesses need to analyze their businesses in ways that decision makers at all levels can quickly respond to changes in the business climate. While a standard transactional query might ask, "How many bolts were sold last month?" An analytical query might ask, "How do sales in the Midwest for the last 3 months compare with the forecast? Now how does that compare to a year ago?"

Analytical queries require an online analytical processing (OLAP) solution. The Oracle provides comprehensive support for OLAP:

- The Oracle relational database management system (RDBMS) remains the most efficient and secure way to store your data. By developing a data warehouse, you can provide data in a form suitable for business analysis.
- The OLAP Option to the Oracle 11g database is full featured multidimensional on-line analytical processing server fully embedded with the Oracle Database Enterprise Edition.
- Integration with the RDBMS core allows for SQL based access to the data.
- There are drivers and plug-ins that allow Excel to report the Oracle OLAP data with easy point and click simplicity.
- Oracle Business Intelligence 11g has built in access to Oracle OLAP data that makes building the metadata simple and allows the data to be reported with highly accelerated speeds.

OLAP OPTION

The Oracle OLAP Option provides the query performance and calculation capability of a multidimensional database. Unlike other marriages of OLAP and RDBMS technology, Oracle 11g OLAP Services is not a thinly disguised multidimensional database using bridges to move data from the relational data store to a multidimensional data store. Instead, it is truly an OLAP enabled relational database.

The OLAP Option can be used to improve query performance, to add rich analytic content to business intelligence applications and to more efficiently maintain data sets that are used by business intelligence applications. The option's query performance optimizations – with most queries satisfied within a few seconds or less – enables business users to engage in ad-hoc exploration and analysis of data. The application developer is able to embed rich analytic content such as time series calculations, shares, indices, rankings and non-additive aggregation methods within the Oracle Database and make them available to virtually any SQL-based business intelligence application.

The Fast, incremental updates of data sets allow organizations to update data sets more often and more efficiently, providing business users with access to the most current data in the shortest amount of time possible.

Finally, the OLAP Option can be used as an alternative to table-based materialized views as a summary management solution, providing the benefits of improved query performance and fast, incremental update.

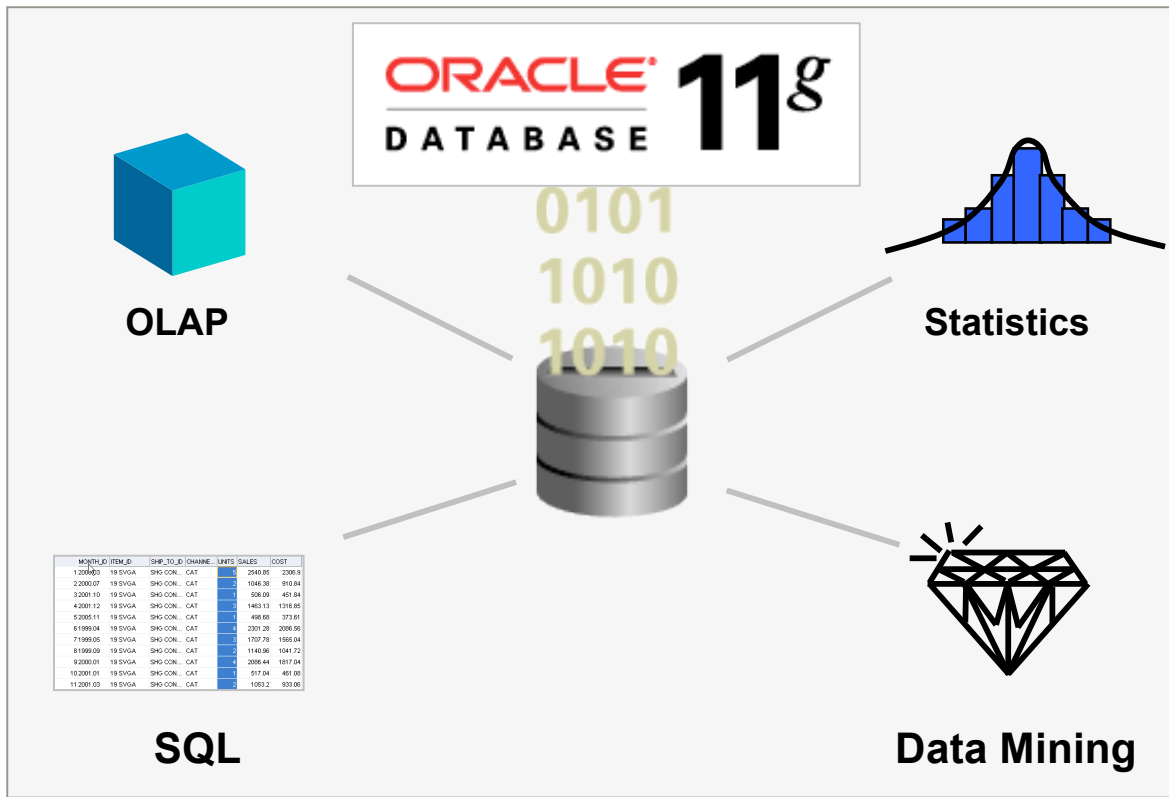


Figure 1. Oracle 11g Strategy for DW

SPEEDING UP THE COMPLICATED!

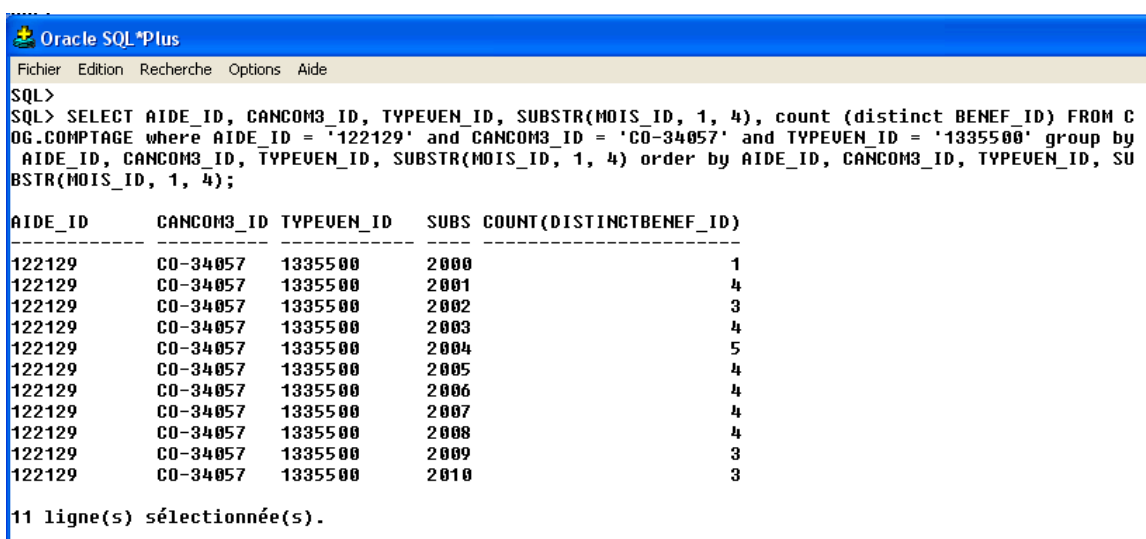
Many implementations today want to be able to do complex analytics. It is not unusual to want to determine how many unique users shop at a webstore each month or quarter or year. Or to report year to date sales compared with prior year and month last year. And then report the percent change. We also have situations where measures have a hierarchical relationship with rollups and special calculations. Things like this present a challenge to a pure relational data warehouse or reporting system. While the calculations may actually be able to be performed but can they be performed and presented to a user in real-time in a few seconds? Usually the answer is NO! If you had deep pockets you could throw money and hardware at the problem and get it done. But it is this type of thing that OLAP databases are designed to solve. Oracle OLAP is no exception. Let's talk about solving some of these problems.

UNIQUE COUNTS

Performing unique counts over dimensional data can be slow and complicated.

"I want to see unique registrations for my web store by day, month, year across my storefronts and departments."

This is not simple additive measure. In, fact this is what is called a non-additive measure so tools that add these counts together using the sum command will present the wrong results. You must perform a count(distinct) starting at the bottom and computing the value for every combination of dimensions and levels.



```

Oracle SQL*Plus
Fichier Edition Recherche Options Aide
SQL>
SQL> SELECT AIDE_ID, CANCOM3_ID, TYPEVEN_ID, SUBSTR(MOIS_ID, 1, 4), count (distinct BENE_ID) FROM C
OG.COMPTAGE where AIDE_ID = '122129' and CANCOM3_ID = 'C0-34057' and TYPEVEN_ID = '1335500' group by
AIDE_ID, CANCOM3_ID, TYPEVEN_ID, SUBSTR(MOIS_ID, 1, 4) order by AIDE_ID, CANCOM3_ID, TYPEVEN_ID, SU
BSTR(MOIS_ID, 1, 4);

AIDE_ID      CANCOM3_ID  TYPEVEN_ID  SUBS  COUNT(DISTINCTBENE_ID)
-----
122129      C0-34057    1335500     2000           1
122129      C0-34057    1335500     2001           4
122129      C0-34057    1335500     2002           3
122129      C0-34057    1335500     2003           4
122129      C0-34057    1335500     2004           5
122129      C0-34057    1335500     2005           4
122129      C0-34057    1335500     2006           4
122129      C0-34057    1335500     2007           4
122129      C0-34057    1335500     2008           4
122129      C0-34057    1335500     2009           3
122129      C0-34057    1335500     2010           3

11 ligne(s) sélectionnée(s).

```

Figure 2. Count Distinct using SQL

Figure 2 shows an example of what the select statement would look like to present data for more than one level. This can take tens of seconds or minutes to return results depending on the size of the fact table.

Using Oracle OLAP to speed this up seems like the best thing to do. Unfortunately, Oracle OLAP does not currently have a count(distinct) function. But there is a way to construct the cubes so that the counts are done using aggregations. If you think about it what we are doing is performing a count(distinct) of a certain dimension across the other dimensions. But we are really not reporting at the individual members of the dimension we are counting. What this means we are taking a Max of the Sum of the members of the counted by dimension for each of the other dimensions. So this now turns into a simple aggregation definition for Oracle OLAP.

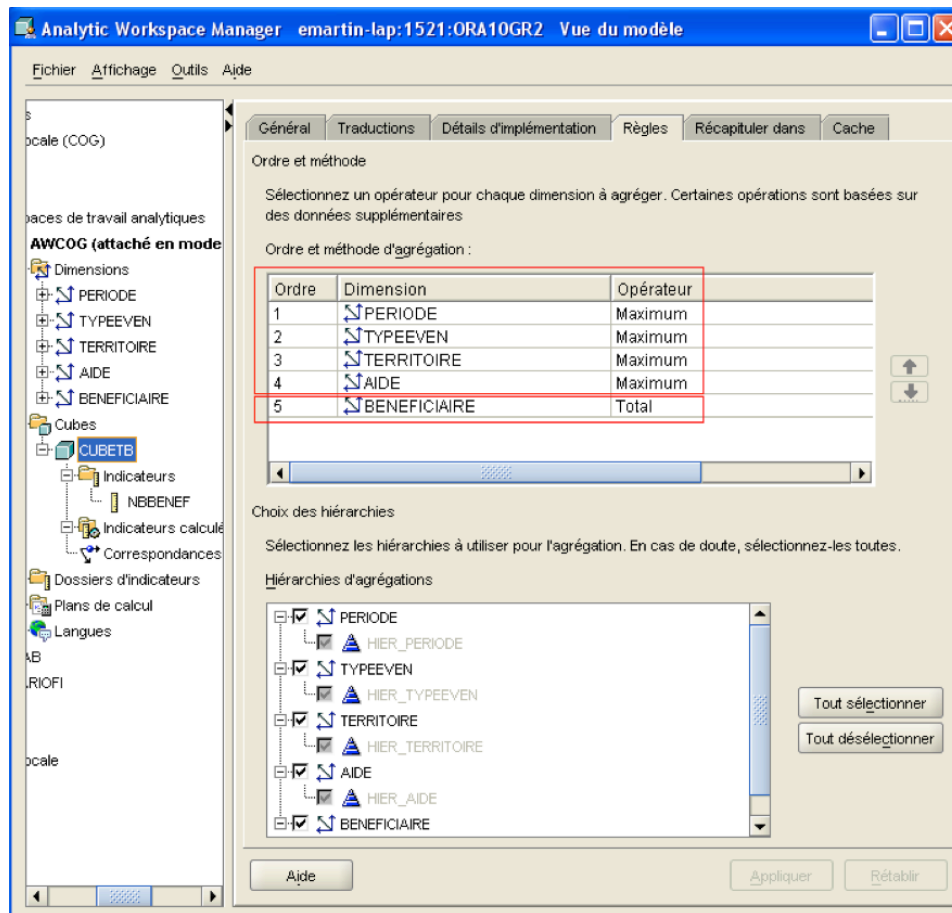


Figure 3. Count Distict Cube in Oracle OLAP

What we see in Figure 3 is how we setup the aggregation of the cube to perform the calculations for the unique count, in this case by beneficiary. When the cube is loaded it will be aggregated and then it can be reported. Any reports against the cube will set the Beneficiary to ALL and then the unique counts will be accurately represented for all the other dimensions at any level. As you can see in Figure 4, the counts are not simple sums. The query performance should be excellent. Our experience shows that for even large datasets (millions of counted members) is only slightly slower than measures that are simply summed.

A	B
Hébergement foyer logement HPH	
CASTELNAU LE LEZ	
Renouvellement Demande HE PH	
Total BENEf	
	NBBENEf
2001_01	1
2001_02	1
2001_03	1
2001_04	1
2001_05	2
2001_06	3
2001_07	3
2001_08	3
2001_09	3
2001_10	4
2001_11	4
2001_12	4
+ 2001	4

Figure 4. Unique Count results

TIME SERIES ANALYSIS

Most reports show data by time in some way or another. Most meaningful analysis of data reporting over time eventually looks at historical data and tries to compare today with yesterday. Asking questions like “show me month to date sales compared to last year at the same time” are not uncommon. Doing this is a simple select statement is complicated and can perform poorly. Adding in multiple levels of time like year, quarter and month and then asking for percent change only makes the SQL even more complicated and even more slow. As you can see in Figure 6, the Select statement is not pretty! While some of the new SQL functions that can do lead and lag and even window can help make things run faster it would mean writing custom code to implement these queries. Most reporting tools do not use those advanced functions. And performance is still slow.

```
WITH sales_dense AS
(SELECT [breakout columns]
sales,
SUM(sales) over(PARTITION BY [breakout columns]
ORDER BY [time column] ASC range BETWEEN unbounded
preceding AND CURRENT ROW) AS sales_ytd
FROM
(SELECT [breakout columns]
a.sales
FROM
(SELECT [breakout columns]
SUM(f.sales) sales
FROM [table list]
WHERE [star join and other filters]
GROUP BY [breakout columns]
a PARTITION BY(breakout columns)
RIGHT OUTER JOIN
(-- need list of all time periods
SELECT DISTINCT [time columns]
FROM time_dim
b ON([join on relevant time level]))
) ...
Continued...
```

Figure 5. Times Series Query

Again this is where OLAP shines. In this case Oracle knows that the time dimension is time based, it has additional information like number of days in the period and the end-dates of each level. From there it makes leading and lagging based upon time very simple. It is very simple to setup calculated measure in OLAP to perform all these times-series calculations

and percent change ratios. In fact, the OLAP administration tool used to create and manage the cubes has a wizard that will build all these measures for you. The end result is the ability to write simple SQL to return the complex results. Figure 7 illustrates how simple a query now becomes. And performance is just as fast as reporting the base measures.

```
SELECT [breakout columns],
       sales,
       sales_prior_year
       sales_ytd,
       sales_ytd_prior_year
FROM sales_cube_view
WHERE [star join]
```

Figure 6. SQL Query of OLAP Cube for Time Series

MEASURE HIERARCHIES

There are occasions where simple measures will not satisfy the requirements. Gross Margin is not just the simple Sales minus Cost. It is in fact based upon a hierarchy of measures as illustrated in Figure 7.

	2008	2009	2010	Grand Total
Measures	336,227	782,819	783,045	1,902,092
Gross Margin	336,227	782,819	783,045	1,902,092
Net Revenue	21,579,338	21,184,025	21,638,729	64,402,092
Gross Revenue	23,500,000	23,000,000	23,500,000	70,000,000
Discount Amount	1,920,662	1,815,975	1,861,271	5,597,908
Units	2,082,264	1,793,724	1,781,233	5,657,221
Net Costs	21,243,111	20,401,206	20,855,684	62,500,000
Fixed Costs	9,033,594	8,581,538	8,884,869	26,500,000
Variable Costs	12,209,517	11,819,668	11,970,816	36,000,000

Figure 6. Measure Hierarchy

While Oracle OLAP measures are not hierarchical it is not hard to model this and present the data as a hierarchy. The solution to this is to create a dimension called measures and build the dimension from a table of view that establishes the parent child relationship of the hierarchy. Once you have this built, shown in figure 7, you simply create a virtual cube that adds this measure dimension to the data cube and using calculated measures assign data measures to the measure dimension members.

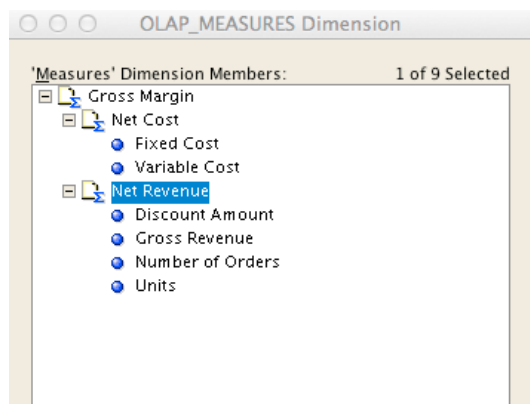


Figure 7. Measure Dimension

Figure 8, shows how one of these calculated measures would look. And the end results, shown in Figure 9 demonstrates that this is simple to do and performance is extremely fast.

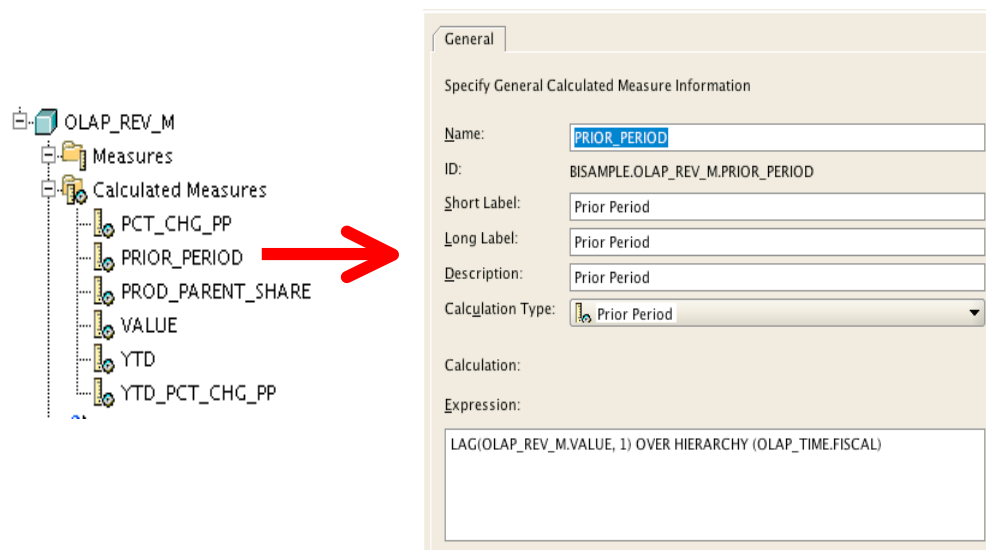


Figure 8. Measure Calculation

	Total Products			
		+ BizTech	+ FunPod	+ HomeView
Gross Margin	1,902,092	-1,878,401	1,864,090	1,916,402
Net Cost	62,500,000	25,130,497	18,938,006	18,431,498
Fixed Cost	26,500,000	10,652,422	8,025,442	7,822,136
Variable Cost	36,000,000	14,478,075	10,912,564	10,609,362
Net Revenue	64,402,092	23,252,096	20,802,096	20,347,900
Discount Amount	5,597,908	2,247,904	1,697,904	1,652,100
Gross Revenue	70,000,000	25,500,000	22,500,000	22,000,000
Number of Orders	71,000	28,474	21,587	20,939
Units	5,657,221	2,223,811	1,695,983	1,737,427

Figure 9. Measures with Hierarchy from Oracle OLAP.

CONCLUSION

As you can see none of this is terribly hard to do, but I hope you can see that it is much easier to do that doing it in SQL and having to maintain it. Furthermore, performance will be much improved and scales very well.

Figure 2. Traditional MV Architecture

Figure 3. Cube Organized Materialize Views

SQL ACCESS

In addition to the query rewrite features, it is now easier than ever before to access the OLAP data with SQL. In previous version of Oracle OLAP it has always been possible to create SQL Views to access the OLAP data. But crafting the views and making them perform well has required a DBA with special knowledge and skill. With 11g this is a thing of the past. When a dimension or cube is created the views necessary to access the data is automatically created. These views are immediately available as a standard view in the schema that owns the OLAP data. Using these views anyone with SQL knowledge can report any data they want. Since the data is already summarized at all levels of aggregation query performance is significantly faster.

With standard SQL access to the data it is now possible to use any SQL base reporting or query tool to access the analytic data.

CONCLUSION

There are many advantages to embedded an OLAP server within the Oracle Database:

- It runs within the same Oracle instance; there is no separate instance to install or manage. There is no separate server computer. It allows your organization to leverage the servers, Oracle DBAs and developers it already has.
- OLAP cubes are stored in Oracle Data Files, just like any other data type of the database. Use the same backup and restore procedures that you already use.
- OLAP data is safe and secure in the Oracle Database. OLAP data is secured with Oracle object and data security features, just like other data in the Database.
- The OLAP Option is fully compatible with scalability and high availability features such as Real Application Clusters and Grid Computing.

- OLAP cubes and dimensions are easily queried with SQL, allowing to you extend the investment in the business tools and applications you already have. This allows the development teams to leverage their valuable skills and provide more comprehensive mission critical solutions to meet the needs of all the users.

The result is quite simple – better business decisions due to more informed decision-makers.