ORACLE
DATABASE

An Oracle White Paper
June 2013

On-line Analytic Processing with Oracle Database 12c

ORACLE

Introduction

The OLAP Option to Oracle Database 12c is a full featured on-line analytical processing server embedded within the Oracle Enterprise Edition Database. The OLAP Option is used in the following roles:

- A summary management solution to SQL-based business intelligence applications. In this role, the Oracle cube is used to manage summary data and is accessed transparently by SQL-based business intelligence applications as a cube-organized materialized view using the query rewrite feature of the Database.

- A calculation engine that provides SQL-based business intelligence applications with rich analytic content. The Oracle cube may be enhanced with additive and non-additive aggregations, calculated measures, statistical forecasts, allocations and dimension calculation models. Applications query the cube using SQL.

- A full-featured multidimensional server, servicing dimensionally oriented business intelligence applications. Applications query all data, including calculations, using a dimensional query language such as the Oracle OLAP API or MDX.

As an embedded solution, the OLAP Option runs within the Oracle Database instance. There are no separate servers, users or database files to manage. It inherits the security and high availability features that make Oracle an enterprise-ready database. OLAP runs on Oracle Exadata Database Machine, gaining significant performance advantage with Smart Flash Cache. With a SQL interface to OLAP cubes, it allows any application that can query a star schema to easily query OLAP cubes and benefit from improved query performance and analytic content.

One Cube – Three Use Cases

The architecture of the Oracle Database allows a single OLAP cube to play three different roles simultaneously:

- A summary management solution, transparent to your applications using the query rewrite feature of the Oracle Database.
- A supplier of rich analytic content to SQL-based business intelligence applications.
- A full featured multidimensional OLAP server.

Oracle OLAP cubes, when used as *cube organized materialized views*, allow SQL-based applications to automatically access summary data managed within the cube using query rewrite capabilities of the Oracle Database. With this usage of the cube, Oracle aggregates and manages summary data within the cube and the application continues to query the detail relational tables using SQL. When a query requires summary data, Oracle automatically rewrites the query to the cube. The SQL-based application is completely unaware of the existence of the cube, yet it benefits from the performance of the cube.

SQL-based business intelligence applications can query the cube directly using relational views of the Oracle cube, gaining both performance improvements and access to analytic content of the cube. The relational views of the cube, dimensions and hierarchies present data as a relational star schema. Unlike the typical star schema, cube views provide detail and aggregate level data as well as interesting measures that are calculated dynamically within the cube. In this usage, the SQL-based application queries relational views of the cube but is typically unaware of the cube itself.

The cube can also by queried by dimensionally aware applications using MDX[1]. This style of application provides the end user with a query experience that reveals the dimensional query and calculation model. It might provide users with an experience that includes drill and pivot, a dimensionally aware query builder and the ability to define custom measures (facts). This application is designed for the OLAP Option and is fully aware of cubes.

One cube can service each of these styles of query simultaneously, thereby leveraging the investment in the cube across many applications and many users for maximum benefit while providing the entire organization with a single version of the truth.

Oracle OLAP and Exadata Database Machine

The OLAP Option is the perfect analytical upgrade for the Exadata Database Machine. The full capabilities of the OLAP Option are available on the Exadata Database Machine. The OLAP Option benefits significantly from the parallel processing capabilities, large amounts of memory and Smart Flash Cache that is available on the Exadata Database Machine, making Exadata the preferred platform for Oracle cubes.

Exadata Benefits for Cube Processing

Oracle cubes may be partitioned and cube partitions may be processed in parallel, one partition for each database process. (Partitions and parallel processes are automatically managed by the Database.) Ideally, the optimal number of parallel processes is equal the lesser of the number of cube partitions or

[1] MDX query of cube requires third party MDX provider.

available processors so that either all partitions are processed in parallel or all processors are being used concurrently.

In practice, the limiting factor on the degree of parallelism for cube processing is I/O bandwidth and the amount of available memory. Not enough memory and low I/O bandwidth causes I/O wait, rendering high degrees of parallelism less effective or even counterproductive. The Exadata Database Machine solves this problem. Large amounts of memory often allow the database to cache whole partitions in memory, completely eliminating I/O wait. When cube partitions do not fully fit into memory, high bandwidth disks minimize I/O wait as compared to less capable systems.

When combined with the appropriate cube partitioning strategy, it is common to observe high degrees of parallel processing with little to no I/O wait on the Exadata Database Machine.

Smart Flash Cache and Cube Queries

Queries to OLAP cubes can generate a high number of small, random disk reads as many lower level cells are accessed in order to calculate higher level aggregate cells. This is quite different from what is seen with tables in data warehouse environments where the tendency is towards a relatively few, but large, disk reads (as will occur during a table scan).

Smart Flash Cache, which is ideal for high-volume random I/O, can significantly improve the (already fast) query performance of OLAP cubes, particularly for larger cubes that might require accessing large number of lower level cells to compute higher level aggregate cells.

Benchmark testing on a large cube shows that the median query performance was improved by 2.6 times when the cube as set to FLASH_CACHE = KEEP as compared to FLASH_CHACHE = DEFAULT. More importantly, Smart Flash Cache had the greatest effect on the most difficult and longest running queries resulting in very few long running queries and even more consistent query performance. Flash cache improved both the average query performance and reduced the longest running queries by 16 times.

Benchmark Test – OLAP on Exadata

In a benchmark test designed to measure the build and query performance of an Oracle cube, the database achieved median query performance of .72 seconds per query on a cube with over 1 billion source rows on a Exadata Database Machine X3-2 'half rack'. There were 60 concurrent users with no waits between queries, so this simulated a much larger user community where users would be pausing to view data before continuing on to a different query.

The cube was four dimensional, with the largest dimension being customers at 2.2 million detail members. The cube contained 5 years of day level data and was partition at the month level, resulting in 60 partitions. Using sixty parallel processes, the full cube build was completed in 42 minutes.

SQL queries to the cube were designed to emulate the style of query seen with business intelligence tools such as Oracle Business Intelligence Enterprise Edition. Queries included time series calculations such as sales year over year, sales change year over year and sales percent change year over year. A series of queries started at a high aggregate level and then drilled randomly on a member of any

dimension of the report. As a result, the query pattern was completely unpredictable in respect to which members, time periods and levels of summarization were being accessed.

Blended Relational-Dimensional Model

The OLAP Option, as an embedded feature of the Oracle Database, allows applications to blend elements of the relational and dimensional model within a single application.

The calculation model of the cube is dimensional, which allows developers and database administrators to embed dimensional and hierarchical calculations into the Database. These are easy to define and efficiently executed at runtime.

Because Oracle offers a full-featured SQL interface to the cube, cubes can simply be thought of as relational objects that happen to offer improved performance and advanced of analytic content. Dimensions can be thought of as relational objects that happen to include columns with information useful for creating hierarchical queries.

Using SQL, an application is free to blend relational and dimensional concepts. An application is also free to join relational tables with cubes and dimensions. For example, an application might query the cube in a dimensional context with drill, pivot, hierarchical filters and calculations, perform attribute based reporting with SQL aggregation functions and drill to detail in a relational table. This is a version common implementation when using Oracle Business Intelligence Enterprise Edition with the Oracle cube.

Benefits of Cubes

Oracle OLAP cubes are data types within the Oracle Database that can enhance your business intelligence applications with important benefits, including:

- Improved query performance.
- Fast, incremental update.
- Rich analytic content.
- Metadata that describes the logical business model and the relational representations of the cube.

Any application, whether SQL or MDX based, has the potential to be improved by taking advantage or one or more of these features.

Query Performance

There are two significant and related challenges to achieving excellent query performance in a business intelligence application. First, almost every query made by the user of a business intelligence application requires summary data. Second, users tend to want to explore data on their own by defining their own queries and reports. This exploration results in unpredictable, or ad-hoc, query patterns.

Ad-Hoc vs. Predictable Query Patterns

When query patterns are predictable, it is relatively easy to optimize query performance of an application by pre-calculating data that satisfies particular queries. If, for example, a BI dashboard contains twenty different queries it might be reasonable to create summary tables or table-based materialized views to support each of those queries and achieve excellent query performance.

As query patterns become less predictable, pre-materialization for specific queries becomes impractical. Consider a data model that has five dimensions (perhaps time, customer, product, distribution channel, and supplier), each of which have six levels of summarization (for example, in the time dimension, day, week, month, quarter, half year and year). In this example, there are 15,624 possible unique level combinations representing summary level data that users might query[2].

DBAs and BI application administrators often try to solve this problem by constraining the end user community to predefined queries that can be tuned by pre-materializing a certain amount of summary data using summary tables or materialized views.

Constraining the users to predefined reports or queries reduces the scope of the problem and allows the DBA to create specific summary tables or materialized views that satisfy those queries with good query performance. This solution, however, also reduces the user's ability to ask their own questions and reduces the effectiveness and value of the business intelligence application. Or, if users are allowed to explore data using ad-hoc queries, they might be exposed to query performance problems.

Instead, the DBA might choose to create a collection of summary tables or materialized views that serve as a general-purpose summary management solution. This will typically be a collection of tables or materialized views with data at certain level combinations. The hope is that queries will either select directly from those level combinations or that a summary table or materialized view is close enough to the query to provide good performance with runtime summarization. The effectiveness of this solution depends on how many summary tables are created and the query patterns. This solution often provides inconsistent performance. Queries that select directly from or are very close to summary tables will perform well. Queries that are not well serviced by summary tables might perform poorly.

Providing excellent query performance throughout the entire data model can be extremely difficult because it would likely require very large numbers of summary tables or materialized views. Again, consider the example with 15,624 possible level combinations. Creating even five percent of the possible level combination (which would result in hundreds of summary tables or materialized views) would likely be only a partial solution for query performance and would create significant overhead in the management of the database.

[2] Five dimensions with six levels each, less one base level or $(5^6 - 1)$

Optimized For Summary Data

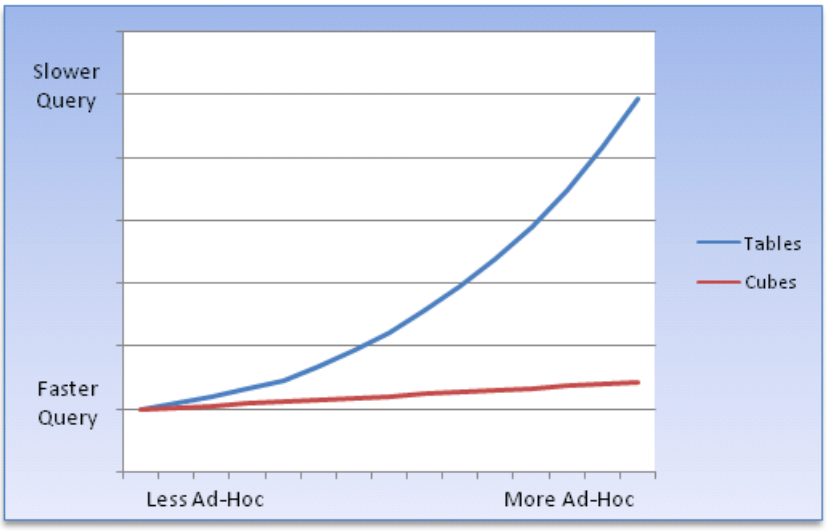
Oracle cubes are designed to handle just this situation, a business model that might be queried by end users with unpredictable query patterns that exposes the entire business model to query at any given moment.

A single OLAP cube represents all levels of summarization within a single object in the Oracle Database. Cubes are highly optimized for hierarchical aggregations and the management of summary data. Furthermore, OLAP cubes are highly optimized for the random cell (or row, from a SQL point of view) access that is typical of ad-hoc query patterns.

- The OLAP option applies patented technologies for the aggregation and storage of summary level data. These methods are highly optimized for the very sparse data sets that are common to business intelligence applications. These technologies include compressed cubes and cost-based aggregation, both of which are unique to cubes.
- References to the cells of a fact (measure) use offset addressing and specialized indexing methods in which dimensions act as indices to the data. Array-based storage eliminates the redundant storage of keys.
- Measure data is pre-joined to dimensions, allowing cubes to be extremely efficient when applying filters to the data, when resolving outer joins (as are required for many time series calculations) and when data from more than one cube is required by the query.
- The cube organized materialized view represents all possible summaries in a single object. This allows the SQL optimizer to quickly choose the object that is the best candidate for query write.

SQL-based business intelligence applications can take advantage of the cube's query performance by either substituting cube-based materialized views for the current summary management strategy or by querying OLAP cube views directly.

The following chart can be used to help understand query performance of summary tables or table-organized materialized views as compared to cubes. In general, both data types (tables and cubes) are appropriate for applications that feature predefined reports or relatively little ad-hoc query. As the queries become more ad-hoc, the cube offers improved query performance.



Query performance advantages of cube and cube organized materialized increase as query patterns become more ad-hoc

Fast Incremental Update

Query performance is only part of the performance equation – any summary management solution should be easy to manage and offer performance advantages for periodic updates of the data set.

Oracle OLAP cubes are highly optimized for fast, incremental update. Because all summaries are managed in a single object, cubes are more easily managed as compared to a large number of summary tables or materialized views.

- Cubes are aggregated and stored using patented algorithms that take advantage of sparsity characteristics typical to hierarchical aggregations.
- Aggregation within the cube is almost always incremental. The cube recognizes changes to base level data and only re-aggregates summary level data when base level data has changed.
- The cube is a single object within the database. Detail tables that are the source to the cube are only scanned once; all aggregations are derived from that single scan of the source table.
- The cube can be updated using the Oracle materialized view refresh system. The cube can process changes to source tables from materialized view log tables, eliminating the need to scan the source tables. Inserts, updates and deletes are incrementally processed from the materialized view log tables.

It should be understood that cubes typically calculate most summary level data at runtime in response to queries; rarely are cubes fully materialized by pre-calculating all summary data. In most cases, only a relatively small amount of data is pre-aggregated during the update of the cube.

The cube has some distinct advantages when it comes to runtime aggregation of data. Two of the most important advantages are a fine-grained approach to pre-aggregating summary data and the pre-joining of dimensions to cubes.

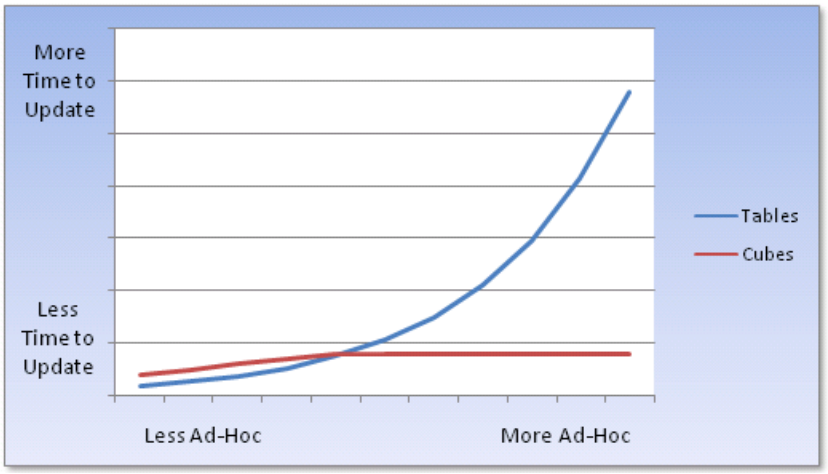
Cost based aggregation utilizes a very fine-grained approach to pre-aggregation of data. Rather than pre-aggregating data using a coarse strategy such as choosing level combinations, cost based aggregation determines which cells (or rows, in SQL terms) are expensive to calculate dynamically and stores only those cells. This determination is made, in part, by examining the data and understanding how many child cells are needed to produce the aggregate for a parent cell. The cost threshold can be influenced by the DBA. The cost based aggregation strategy results in a very 'fine mesh' of pre-aggregated cells and more consistent performance throughout the entire cube.

As part of the array based implementation of the cube, dimension and cube are pre-joined. As a result, join processing is completely eliminated from the process of producing aggregate data within the cube.

Because somewhat more work is done up front in the process of maintaining cubes – for example compiling dimensions, enforcing referential integrity and automatically indexing data – it can be expected that the cube will have a longer minimum data preparation time as compared with simply inserting data into a relational table. Once indexing tables and the creation of summary tables or materialized views is taken in account, cubes begin to have an advantage of shorter overall data preparation times.

Whether the cube or summary tables are available for query more quickly depends on the query load and required query performance. Once again, consider the difference between predictable and ad-hoc query patterns. If the query patterns are very predictable, only a few summary tables or materialized views might be required. Building those tables might be faster than building the cube.

As the query patterns become more unpredictable, the cube has the advantage because it tends to become fully optimized for query relatively quickly due to the highly efficient aggregation algorithms (again, compressed cubes and cost based aggregation). Pre-aggregating a relatively small amount of data using cost based aggregation tends to optimize the cube for query quickly. In the case of summary tables or materialized views, more and more tables need to be created to improve query performance as the query load becomes more ad-hoc. This relationship is shown in the following graph.



Cubes are quickly optimized for ad-hoc query environments

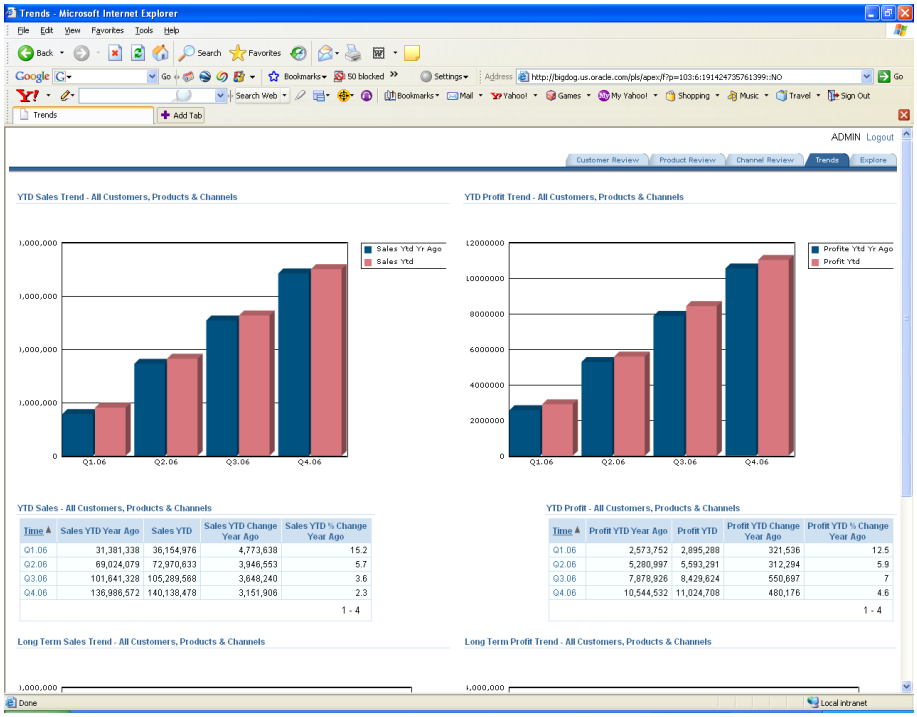
Given that the cube is well suited to servicing both predictable and unpredictable query loads, organizations should consider the cube and cube organized materialized views as the summarization strategy for any dimensional data set.

Enhancing the Analytic Content of BI Applications

The performance characteristics of the cube alone should be enough to cause organizations to consider adding cubes to their data warehouse or data mart. Performance is, however, only one of the benefits of the cube. Another benefit is the ability to dramatically improve the analytic content of business intelligence applications.

A wide variety of analytic calculations can be defined within the cube. These include hierarchical aggregation, calculated measures, allocations, statistical forecasts and member calculation models.

The cube is automatically represented in the database as a star schema. There are relational views for each of the cube's dimensions and a single view to represent the measures of the cube. Analytic calculations of the cube are simply exposed as relational columns in the cube view. For example, there may be a column for sales revenue, sales year to date, inventory balance and inventory cost. Although the column may return a complex calculation (e.g. sales year to date) or require an advanced aggregation method (e.g. inventory balance) – the query tool does not need to understand the calculation rule. The tool issues simple SQL and the cube performs the calculation automatically.



An application built using Oracle Application Express, which has no specific knowledge of business intelligence or cubes, is an example of an application that can query analytic content such as year to date calculations managed in the cube.

Leveraging the Dimension Model for Defining Calculations

The cube uses the dimensional model to simplify the definition of calculations. For example, measure calculations can be defined using syntax that is aware of hierarchical relationships such as parentage, children, ancestors and descendants. Although the syntax used to define measure calculations is dimensionally aware, OLAP functions are based on SQL grammar and will be familiar to developers and DBAs who are experienced with SQL analytic functions such as LAG and RANK.

One significant benefit of calculation functions that are hierarchically aware is that a single expression works everywhere within the cube. With traditional SQL-based applications the expression may change based on what columns are being queried.

For example, consider the LAG function. If a product dimension has the levels Total, Manufacturer, Brand, Item and SKU and an application needs to rank products within parents, it would need to define one expression for each level. For example:

```
-- Rank brands with Total
rank() over(PARTITION BY p.TOTAL ORDER BY f.sales DESC NULLS LAST) RANK
-- Rank items within Brand
rank() over(PARTITION BY p.BRAND ORDER BY f.sales DESC NULLS LAST) RANK
-- Rank SKUs within Items
rank() over(PARTITION BY p.ITEM ORDER BY f.sales DESC NULLS LAST) RANK
```

While some of the more sophisticated query applications can solve this problem by defining hierarchies and calculations in the middle tier, many applications cannot. In these cases, the end user must define the expression for each level of summarization. This can be a significant burden, particularly in applications that need to drill up or down within the data.

With the cube, a single expression can be used to define hierarchical calculations that work everywhere within the model. The following rank expression will rank products within their parents at any level in the hierarchy and can replace the three different SQL rank functions previously discussed.

```
-- Rank within parent
RANK() OVER HIERARCHY (PRODUCT.PRIMARY ORDER BY UNITS_CUBE.SALES DESC NULLS
LAST WITHIN PARENT)
```

Note that the OLAP expression is similar to the SQL expression, but a hierarchy and the PARENT keyword replace the query partition clause. Because the OLAP expression describes a relative relationship rather than a hard coded column, it works at any level of summarization. This same pattern can be seen in another example, this one based on a period-to-date time series calculation:

```
-- SQL expression
SUM(f.sales)
  OVER(PARTITION BY t.calendar_year
  ORDER BY t.end_date
  RANGE BETWEEN unbounded preceding AND current row)
```

```
-- OLAP expression
SUM(units_cube.sales)
  OVER HIERARCHY (time.calendar
  BETWEEN unbounded preceding AND current member
  WITHIN ancestor at level time.calendar.calendar_year)
```

Like the previous example, OVER HIERARCHY replaces OVER and elements of the hierarchy replace column names.

Similar functions are available for common calculations such as shares, prior/future periods, period to date, period from date, moving aggregates and cumulative aggregates. With each type of calculations, variations are available for calculations relative to parents, ancestors at particular levels and within levels.

Time series calculations are further simplified because the cube automatically executes a partitioned outer join to densify the data. This ensures that the calculation always returns the correct results for prior and future period calculations, even when the value of the prior or future period is null. Because dimensions and cubes are automatically joined, this operation is very efficient in the cube.

OLAP functions can be combined or nested to design new functions. Also, most SQL single row functions (e.g., DECODE and ROUND functions) can be used within OLAP calculations that are embedded in the cube. As a result, very powerful user defined calculations can be defined within the cube.

As mentioned earlier, all measure calculations reveal themselves in the SQL cube view as a column. Applications simply select from the column in the SQL cube view to access the results of a calculation.

Summary of Calculations Available in the Cube

The different types of calculations that are available in the cube can be described as fitting into one of the following categories: hierarchal aggregations, measure calculations, allocations, statistical forecasts and model. Each of these calculations are described in the following sections.

Hierarchical Aggregations

Hierarchical aggregations are calculations that aggregate data from lower level members in a hierarchy to higher-level members. The OLAP Option supports a wide variety of aggregation methods including sum, hierarchical weighted averages and scaled sums. Aggregation methods can vary by dimension. For example, the aggregation of a Headcount measure might be a sum over an Organization dimension and an average from days to months, quarters and years of a Time dimension.

Calculated Measures

A calculated measure is a measure that exists in the cube as a formula that is calculated dynamically during query. Examples include time series calculations, market shares and indices, variance and rankings. The cube can process measure calculations very efficiently even when they require inter-row calculations, outer joins and joins between different cubes.

Calculated measures are added to the SQL cube view as additional fact columns.

Allocations

Allocations distribute data from high-level members within a hierarchy to lower level members. For example, a corporate budgeting system might use the allocation system to distribute budgets for the next fiscal year from divisions to product groups, and finally to individual products.

The OLAP Option supports a wide variety of allocation methods including copy methods (hierarchical copy, minimum, maximum, first, last), even distribution methods (even, hierarchical even) and proportional (including weighted distributions).

The results of an allocation are stored in the cube and are queried as a fact column in a SQL cube view.

Statistical Forecasting

OLAP Option provides a complete statistical forecasting system. Forecasting methods include non-linear regression, single exponential smoothing, double exponential smoothing and Holt/Winters. The forecasting system also supports features such as seasonality of data.

The results of a forecast are stored in the cube and are queried as a fact column in a SQL cube view.

Member Calculation Models

OLAP models calculate the value of individual dimension members, each according to their own unique calculation rules. The OLAP Option automatically orders the calculations and supports simultaneous equations. Models are often used in financial applications, particularly for defining calculations within non-hierarchical dimensions such as a chart of accounts.

BI Meta Data in the Oracle Data Dictionary

The cube represents several import assets of an enterprise business intelligence platform: data, the logical business model, calculation rules and a physical representation of the business model. Each of these elements is described in the Oracle data dictionary. Business intelligence applications can query the data dictionary to discover any aspect of the cube and automatically populate their own metadata repositories with the information needed to query SQL cube views.

Generally speaking, OLAP metadata can be grouped into one of two categories:

- Information about the cube's structure, data and how it is calculated.
- Information about how the cube is represented for query using SQL cube views.

The data dictionary views that describe the cubes structure allow applications to find cubes dimensions, hierarchies, levels, attributes and measures. In the case of cubes and measures, the dictionary views describe how the cube is calculated and the calculation expression of a measure. Applications can use this information to understand the logical model of the cube.

The data dictionary views that describe how the cube is exposed to SQL describe the cube, dimension and hierarchy views with information such as view names, column names and their roles. Applications can use this information to map their applications to the SQL cube views.

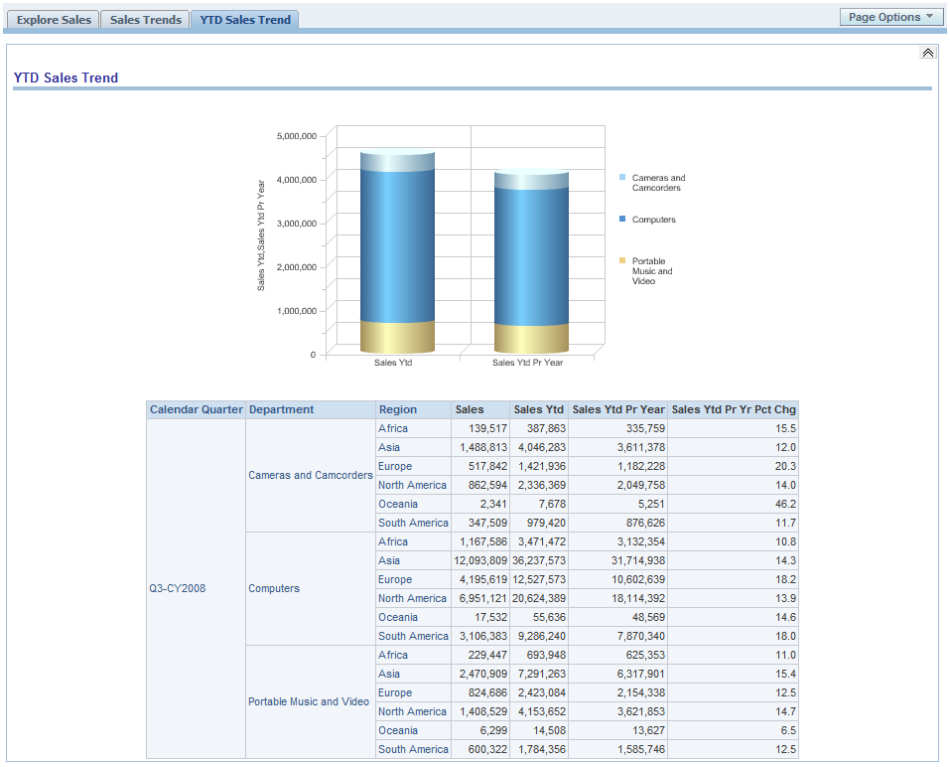
OLAP data dictionary views are easily found by looking for the prefix XXX_CUBE (for example USER_CUBE, ALL_CUBE and DBA_CUBE).

Using Oracle Business Intelligence with Cubes

Oracle Business Intelligence Enterprise Edition is a full-featured business intelligence reporting system featuring dashboards and alerts that can access a variety of data sources. As a SQL-based system, Oracle Business Intelligence can use Oracle cubes as summary management solution (transparently, using cube-organized materialized views) or as an analytically rich data source (querying cube views directly).

When using cube-organized materialized views, Oracle Business Intelligence simply queries relational tables and the database automatically rewrite queries to the cube. There are no changes to the Oracle Business Intelligence repository.

To support direct query of the cube, thus allowing Oracle Business Intelligence to access the calculations within the cube, the Oracle OLAP Option provides a feature that automatically creates an Oracle Business Intelligence repository that represents one or more cubes. The SQL issued by Oracle Business Intelligence based on the generated repository is perfectly tuned for the cube, ensuring optimal performance.



Calendar Quarter	Department	Region	Sales	Sales Ytd	Sales Ytd Pr Year	Sales Ytd Pr Yr Pct Chg
Q3-CY2008	Cameras and Camcorders	Africa	139,517	387,863	335,759	15.5
		Asia	1,488,813	4,046,283	3,611,378	12.0
		Europe	517,842	1,421,936	1,182,228	20.3
		North America	862,594	2,336,369	2,049,758	14.0
		Oceania	2,341	7,678	5,251	46.2
		South America	347,509	979,420	876,626	11.7
	Computers	Africa	1,167,586	3,471,472	3,132,354	10.8
		Asia	12,093,809	36,237,573	31,714,938	14.3
		Europe	4,195,619	12,527,573	10,602,639	18.2
		North America	6,951,121	20,624,389	18,114,392	13.9
		Oceania	17,532	55,636	48,569	14.6
		South America	3,106,383	9,286,240	7,870,340	18.0
	Portable Music and Video	Africa	229,447	693,948	625,353	11.0
		Asia	2,470,909	7,291,263	6,317,901	15.4
		Europe	824,686	2,423,084	2,154,338	12.5
		North America	1,408,529	4,153,652	3,621,853	14.7
		Oceania	6,299	14,508	13,627	6.5
		South America	600,322	1,784,356	1,585,746	12.5

Year-to-Date dashboard page in Oracle Business Intelligence Enterprise Edition. All data is queried from an Oracle cube using SQL.

Using Microsoft Excel with Cubes

Using Microsoft Excel pivot tables, users can query Oracle cubes in a dimensional context. Excel queries Oracle cubes directly using an MDX provider, allowing users to access any data in the cube – detail and summary data, and calculations – interactively. Users can define their own queries, drill within hierarchies from summary to detail levels and change the orientation of a report by pivoting dimensions.

Excel's formatting and graphing features can be used with Oracle cube data. Cube data can be referenced by other cells in the worksheet or other worksheets. Excel pivot tables and charts can be embedded into other MS office applications such as PowerPoint and Word, allowing those applications to display data live from the Oracle OLAP cube.

	Values				
Row Labels	Product Alert	Sales	Sales Ytd	Sales Ytd Pr Year	Sales Ytd Pr Yr Pct Chg
Q4-CY2008		**40,469,228.71**	**147,447,084.93**	**129,654,400.02**	**13.72**
Cameras and Camcorders		**3,575,268.30**	**12,387,923.49**	**10,657,143.31**	**16.24**
Computers		**31,628,255.80**	**115,957,793.25**	**101,669,049.64**	**14.05**
All Computer Furniture		17,817.00	57,008.60	41,414.80	37.65
Computer Printers and Supplies		5,282,123.93	18,903,506.59	17,175,902.29	10.06
PDAs		24,419.98	77,051.16	62,305.38	23.67
Total Personal Computers		25,908,098.49	95,382,154.70	83,091,648.37	14.79
Computer Books		100,254.46	373,905.10	339,296.54	10.20
Computer Displays and Projectors		515,707.05	1,891,349.57	1,650,894.03	14.57
Computer Storage		4,045,677.10	14,747,040.44	13,294,687.72	10.92
Display Cards and Accessories		30,416.63	118,225.43	102,469.16	15.38
Docking Stations, Stands and Protection		94,732.89	349,020.17	316,981.16	10.11
Input Devices		887,916.10	3,252,596.23	2,946,081.45	10.40
Memory		52,642.80	197,066.40	206,151.80	-4.41
Misc. Computer Accessories		30,782.78	113,371.37	102,972.13	10.10
Misc PC Accessories		8,004.29	31,056.67	28,885.89	7.52
Networking		2,246,554.60	8,135,975.31	7,359,682.55	10.55
PC Sound		7,496,957.59	27,176,114.38	24,742,478.27	9.84
Personal Computers		4,544,029.20	17,363,296.43	12,614,328.10	37.65
Powerbook Cases and Backbacks		69,889.98	243,167.41	222,094.67	9.49
Scanners		111,483.91	438,696.02	399,174.22	9.90
Software		2,100,897.62	7,838,228.26	6,805,913.44	15.17
UPS and Power Protection		3,510,210.37	12,908,944.85	11,802,504.45	9.37
Video		61,941.12	204,100.66	157,052.79	29.96
Total Server Computers		395,796.40	1,538,072.20	1,297,778.80	18.52
Portable Music and Video		**5,265,704.61**	**19,101,368.19**	**17,328,207.07**	**10.23**

Querying an Oracle cube using Microsoft Excel pivot tables.

Querying the Cube Using SQL

SQL is the most commonly used query language within business intelligence and query and reporting applications. This is not surprising given that the vast majority of data is managed in relational databases. The OLAP Option embraces SQL-based applications and allows them to query the cube using standard SQL. The application can access any data in the cube – detailed, summary, stored and calculated – with simple SQL and without any understanding of how data within the cube is calculated.

An application typically falls into one of two categories:

- Applications that seek only a performance improvement from the cube. These applications can use cube-organized materialized views to access summary data in the cube.
- Applications that seek the performance and analytic content of the cube. These applications can query cube views to gain performance and access any content within the cube.

Cube-Organized Materialized Views and Query Rewrite

Cube-organized materialized views allow any application to transparently access summary data managed by the cube for an immediate query performance improvement. Cube-organized materialized views, introduced in Oracle Database 12c, play the same role as table-based materialized views. That is, a summary management solution that is transparent to the querying application. Like table-based materialized views, the application queries the detail tables and the database automatically rewrites the query to access summary data in the materialized view.

In the case of cube-organized materialized views, the data is managed in the cube rather than a table. As such, cube-organized materialized views automatically inherit the query and refresh performance benefits of the cube, including:

- Compressed cube aggregation technology.
- Cost-based aggregation.
- Management of all possible summaries in a single database object.
- Array based storage and fast cell (row) access.
- Fast, incremental refresh and aggregation.

A cube-organized materialized view is similar to a materialized view on pre-built table in that it is a metadata-only object. The data is managed and stored in the cube; the materialized view contains only the metadata that is needed by the database for query write and refresh of the cube via the materialized view refresh system. Data is not replicated from the cube to the cube-organized materialized view.

There is a single materialized view that represents all summary data in the cube. This single cube-based object, rather than the tens, hundreds or even thousands of table based materialized views that might be used as a summary management solution, improves the efficiency of the query rewrite feature by dramatically reducing the number of materialized views that the optimizer needs to consider before completing the execution of the query (in addition to the other advantages of the cube).

Because the cube-organized materialized view represents all possible summaries, the rate at which the database rewrites queries to the materialized view is typically very high. Testing with leading business intelligence applications has shown this to be the case.

Cube-organized materialized views can be recognized by their CB$ name prefix. For example, the materialized view representing a cube named UNITS_CUBE will be named CB$UNITS_CUBE. The DBA adds the cube-organized materialized view to the cube using Analytic Workspace Manager (the OLAP administration tool) or the OLAP API. The database then automatically maintains the definition of the materialized view.

Cube organized materialized views are completely transparent to the querying application and the processes of managing and refreshing the source tables. The application continues to query relational tables. The process of managing the tables is unchanged by the existence of the cube.

Query Example

The following example illustrates how cube-organized materialized views allow applications to transparently access summary data in a cube.

In this example, assume that there are five tables: TIME_DIM, PRODUCT_DIM, CUSTOMER_DIM, CHANNEL_DIM and UNITS_FACT. The _DIM tables are dimension tables. The UNITS_FACT table is a fact table. Together, this collection of tables comprises a star schema.

A cube has been created that mirrors the dimensions, hierarchies and fact data of these tables and the DBA has turned on cube-organized materialized views. The cube-organized materialized view is named CB$UNITS_CUBE.

The application is unaware of the cube or the materialized view. It simply queries the tables, as it would have before the cube was created, with a query such as:

```
SELECT t.calendar_year_id time,
  p.class_id product,
  c.region_id region,
  SUM(f.sales) sales
FROM time_dim t,
  product_dim p,
  customer_dim c,
  units_fact f
WHERE t.month_id = f.month_id
 AND p.item_id = f.item_id
 AND c.ship_to_id = f.ship_to_id
GROUP BY t.calendar_year_id,
  p.class_id,
  c.region_id;
```

The Oracle optimizer recognizes that the cube contains the summary data requested in this query and rewrites the query to the cube-organized materialized view. The resulting SQL execution plan follows.

```
OPERATION
------------------------------
SELECT STATEMENT
 HASH
  CUBE SCAN CB$UNITS_CUBE
```

The CUBE SCAN operation indicates that data is being returned by the cube row source. CB$UNITS_CUBE is the cube-organized materialized view.

Managing Stale Data

Like table based materialized views, the database understands when changes have been made to source tables and that the cube is no longer a current representation of the tables. When a cube is a current

representation of the source data it is said to be *fresh*. When it is not, the materialized view is said to be *stale*. The materialized view system allows the database administrator to allow or prevent rewrite to stale materialized views. In this regard, table and cube-organized materialized views are identical.

Cube Views

Oracle OLAP cube views provide organizations with the ability to both improve the performance and analytic content of SQL-based business intelligence applications. OLAP cube views are relational views of OLAP cubes, dimensions and hierarchies that reveal the full content of cubes and dimensions.

Overview

Cube views allow applications to query stored facts and analytic content such as calculated measures that are embedded in the cube. Dimension and hierarchy views allow applications to query for dimension members and attributes of the members. In an OLAP dimension, attributes can include hierarchical attributes such as levels, parentage, ancestors and descendants that can be used to add elements of the dimensional query model to SQL-based applications.

Each cube, dimension and hierarchy within a dimension is always represented to SQL-based application as a relational view. There are three types of views:

- Cube views.
- Dimension views.
- Hierarchy views.

These views are automatically created and managed by the database. Application developers and DBAs are welcome to create additional views from the system maintained cube views.

The collection of views that represent the cube, dimensions and hierarchies are structured in the form of a star schema.

Cube Views

Cube views represent the fact data of the cube. The cube view includes columns for each key (representing each dimension) and each fact. An innovative feature of the cube view is that it represents all data in the cube: stored measures, calculated measures, detail data and summary level data. This means that the application does not need to understand how a fact is aggregated or calculated in order to query it with SQL. This allows the cube view to serve as both a summary management solution and as a calculation rich data source.

A typical cube view might include the following columns:

```
Name                           Type
------------------------------ -------------
TIME                           VARCHAR2(100)
PRODUCT                        VARCHAR2(100)
CUSTOMER                       VARCHAR2(100)
```

```
CHANNEL                        VARCHAR2(100)
SALES                          NUMBER
UNITS                          NUMBER
COST                           NUMBER
PROFIT                         NUMBER
SALES_YR_AGO                   NUMBER
SALES_DIFF_YR_AGO              NUMBER
SALES_PCTDIFF_YR_AGO           NUMBER
SALES_YTD                      NUMBER
SALES_YTD_YR_AGO               NUMBER
SALES_DIFF_YTD_YR_AGO          NUMBER
```

Note – The roles and descriptions of cube view columns are available in the xxx_CUBE_VIEW_COLUMNS data dictionary view.

Dimension Views

Dimension and hierarchy views both represent data in dimensions. Each style view presents the dimensions somewhat differently and an application may use either, or both, style views depending on the needs to the application.

Dimension views represent all dimension members – detail and summary, from all hierarchies – in a single view. The rows include both detail and summary members. There is a single key column. Additional columns contain attributes of the dimension member in the key column.

A typical dimension view might have the following columns:

```
Name                           Type
------------------------------ -------------
DIM_KEY                        VARCHAR2(100)
LEVEL_NAME                     VARCHAR2(30)
LONG_DESCRIPTION               VARCHAR2(100)
SHORT_DESCRIPTION              VARCHAR2(100)
CUSTOMER_TOTAL_ID              VARCHAR2(100)
CUSTOMER_REGION_ID             VARCHAR2(100)
CUSTOMER_WAREHOUSE_ID          VARCHAR2(100)
CUSTOMER_SHIP_TO_ID            VARCHAR2(100)
CUSTOMER_MARKET_SEGMENT_ID     VARCHAR2(100)
CUSTOMER_ACCOUNT_ID            VARCHAR2(100)
```

The DIM_KEY column is the primary key of the view and contains dimension members (again, both detail and summary level members).

The LEVEL_NAME column contains the level of the member. In this example, SHIP_TO, WAREHOUSE, REGION, ACCOUNT, MARKET_SEGMENT and TOTAL levels). The

LEVEL_NAME column allows applications to easily indicate what level of summarization the query requires.

The SHORT_DESCRIPTION and LONG_DESCRIPTION columns contain descriptive names of the dimension members.

The _ID columns contain the summary level members of the DIM_KEY column. These are useful for finding the ancestor or descendants of a dimension member.

Note – The roles and descriptions of dimension view columns are available in the xxx_DIMENSION_VIEW_COLUMNS data dictionary view.

Hierarchy Views

There is one hierarchy view for each hierarchy of a dimension. Hierarchy views are the same as dimension views, with three exceptions:

- Hierarchy views only contain rows for members that belong to that hierarchy.
- Hierarchy views include a PARENT column that returns the parent of the dimension member (in the DIM_KEY column).
- Hierarchy views include two columns representing each level, one for the original key and another for the surrogate key that might have been generated in the dimension as part of the process of loading data from the dimension table to the OLAP dimension.[3]

The hierarchy views are particularly useful for queries that drill up or down within a hierarchy and that join the cube to source tables.

A typical hierarchy view might contain the following columns:

```
Name                           Type
------------------------------ -------------
DIM_KEY                        VARCHAR2(100)
LEVEL_NAME                     VARCHAR2(30)
LONG_DESCRIPTION               VARCHAR2(100)
SHORT_DESCRIPTION              VARCHAR2(100)
PARENT                         VARCHAR2(100)
TOTAL                          VARCHAR2(100)
CUSTOMER_TOTAL_ID              VARCHAR2(100)
REGION                         VARCHAR2(100)
CUSTOMER_REGION_ID             VARCHAR2(100)
```

[3] Surrogate keys can be generated in the OLAP dimension for cases where dimension members are not unique across levels in the source tables.

```
WAREHOUSE                      VARCHAR2(100)
CUSTOMER_WAREHOUSE_ID          VARCHAR2(100)
SHIP_TO                        VARCHAR2(100)
CUSTOMER_SHIP_TO_ID            VARCHAR2(100)
```

Note – The roles and descriptions of dimension view columns are available in the xxx_HIERARCHY_VIEW_COLUMNS data dictionary view.

Query Examples

The following query examples continue on the previous example where there are four dimensions (Time, Product, Customer and Channel) and a cube (Units Cube). The Time dimension has two hierarchies, Calendar and Fiscal. The Customer dimension has two hierarchies, Shipments and Segment. Both the Product and Channel dimensions have a single hierarchy, both named Primary. This model will result in eleven views:

- TIME_VIEW
- TIME_CALENDAR_VIEW
- TIME_FISCAL_VIEW
- PRODUCT_VIEW
- PRODUCT_PRIMARY_VIEW
- CUSTOMER_VIEW
- CUSTOMER_SHIPMENTS_VIEW
- CUSTOMER_SEGMENTS_VIEW
- CHANNEL_VIEW
- CHANNEL_PRIMARY_VIEW
- UNITS_CUBE_VIEW

The following query is the equivalent to the example used in the materialized view example:

```
SELECT t.time_calendar_quarter_id time,
  p.product_class_id product,
  cu.customer_region_id customer,
  f.sales sales
FROM time_view t,
  product_view p,
  customer_view cu,
  channel_view ch,
  units_cube_view f
WHERE t.dim_key = f.time
```

```
AND p.dim_key = f.product
AND cu.dim_key = f.customer
AND ch.dim_key = f.channel
AND t.level_name = 'CALENDAR_QUARTER'
AND p.level_name = 'CLASS'
AND cu.level_name = 'REGION'
AND ch.level_name = 'TOTAL';
```

This query follows the style of a star query, but differs from the materialized view example in the following ways:

- It does not require an aggregation operator in the fact column SALES.
- It does not require a GROUP BY clause because an aggregation operator is not used.
- It includes filters on the LEVEL_NAME columns.
- It includes a join between the CHANNEL_VIEW view and the UNITS_CUBE_VIEW view.

Each of these changes is made possible by the innovative cube view that includes both detail and summary level data. Because the view includes summary level data, the application can query it directly. This allows for the elimination of both the aggregation operator and the GROUP BY. The level filters are used to indicate desired level of summarization.[4] The join between CHANNEL_VIEW and the UNITS_FACT_VIEW is used so that the query utilizes the TOTAL value of the channel dimension, which is the equivalent of aggregating across all detail rows of channel.

The fundamental principle illustrated by this example is that the query can utilize the aggregations that are provided by the cube. There is no need to re-aggregate the data within the query. Complex aggregations and other calculations simply made available by the cube, allowing very simple SQL to be used to query the cube. This simple SQL can significantly improve developer productivity

The value of presenting both detail and summary level data in the cube view is seen when querying certain types of data that is calculated within the cube. The most common cases are when the cube:

- Calculates summary data using an aggregation operator unique to the cube. For example, a hierarchical weighted average.
- Aggregation operators differ across dimensions. For example, the Ending Balance can be calculated using the LAST aggregation operator for the Time dimension and SUM for other dimensions such as Account.

[4] Note that the inclusion of SUM and GROUP BY would not change the results of this query because it already returns summary data. Each row is unique and the GROUP BY has no practical effect in this example.

- Calculates certain types measures that cannot be aggregated. For example, ranking and percent change measures cannot be aggregated from one level to another. They must be calculated at the desired level after data are aggregated.

What is common to these types of calculations is that they cannot be aggregated using SUM (or another aggregation function) and GROUP BY. They can be easily defined and executed within the cube, but they should not be aggregated in SQL.

Consider the following example that selects sales, sales year to date and sales percent change year to date from the year ago:

```
SELECT t.time_calendar_quarter_id time,
  f.sales sales,
  f.sales_ytd sales_ytd,
  ROUND(f.sales_pctdiff_ytd_yr_ago,1) pct_chg_ytd
FROM time_view t,
  product_view p,
  customer_view cu,
  channel_view ch,
  units_cube_view f
WHERE t.dim_key = f.time
 AND p.dim_key = f.product
 AND cu.dim_key = f.customer
 AND ch.dim_key = f.channel
 AND t.level_name = 'CALENDAR_QUARTER'
 AND p.level_name = 'TOTAL'
 AND cu.level_name = 'TOTAL'
 AND ch.level_name = 'TOTAL'
 and t.time_calendar_year_id in ('CY2005','CY2006')
 ORDER BY time;

TIME      SALES         SALES_YTD      PCT_CHG_YTD
--------- ------------- -------------- -----------
CY2005.Q1    31381338.07   31381338.07        -4.8
CY2005.Q2    37642741.22   69024079.29         0.4
CY2005.Q3    32617248.57  101641327.86        -0.6
CY2005.Q4    35345244.10  136986571.96        -5.1
CY2006.Q1    36154818.61   36154818.61        15.2
CY2006.Q2    36815657.09   72970475.70         5.7
CY2006.Q3    32318934.94  105289410.64         3.6
CY2006.Q4    34848910.74  140138321.38         2.3
```

In this example, PCT_CHG_YTD for Quarter level cannot be calculated by aggregating the PCT_CHG_YTD from Month level data. This measure needs to be calculated after data are

aggregated to Quarter. A single function in the cube to calculate PCT_CHG_YTD can replace very complex SQL (including a partitioned outer join) and the application can simply select the measure at the desired level of summarization.

As final query examples, consider a drill down and drill up (collapse) style queries. Given that the parent-child relationship data is revealed in the hierarchy views, drill down and drills up queries are both easy and efficient.

The following query illustrates drilling down to the quarters within a year. Note that in this example the query returns measure data for both the year that was drilled on and children of the year. This is very easy when querying a cube view.

```
SELECT t.dim_key time,
  f.sales sales,
  f.sales_ytd sales_ytd,
  ROUND(f.sales_pctdiff_ytd_yr_ago,   1) pct_chg_ytd
FROM time_calendar_view t,
  product_view p,
  customer_view cu,
  channel_view ch,
  units_cube_view f
WHERE t.dim_key = f.time
 AND p.dim_key = f.product
 AND cu.dim_key = f.customer
 AND ch.dim_key = f.channel
 AND p.level_name = 'TOTAL'
 AND cu.level_name = 'TOTAL'
 AND ch.level_name = 'TOTAL'
 AND(t.dim_key = 'CY2006' OR t.parent = 'CY2006')
ORDER BY time;

TIME        SALES          SALES_YTD     PCT_CHG_YTD
----------- -------------- ------------- -----------
CY2006      140138478.38   140138478.38          2.3
CY2006.Q1    36154975.61    36154975.61         15.2
CY2006.Q2    36815657.09    72970632.70          5.7
CY2006.Q3    32318934.94   105289567.64          3.6
CY2006.Q4    34848910.74   140138478.38          2.3
```

This query simply selects the members that have a particular parent. Since the hierarchy view has members for all levels as rows, this query works for children of any member. (Note that a level filter is not required in this query.)

A similar query can be used to find the parent of a member.

```
SELECT t.dim_key time,
  f.sales sales,
  f.sales_ytd sales_ytd,
  ROUND(f.sales_pctdiff_ytd_yr_ago,1) pct_chg_ytd
FROM time_calendar_view t,
  product_view p,
  customer_view cu,
  channel_view ch,
  units_cube_view f
WHERE t.dim_key = f.time
 AND p.dim_key = f.product
 AND cu.dim_key = f.customer
 AND ch.dim_key = f.channel
 AND p.level_name = 'TOTAL'
 AND cu.level_name = 'TOTAL'
 AND ch.level_name = 'TOTAL'
 AND t.dim_key = (SELECT DISTINCT parent
    FROM time_calendar_view
    WHERE dim_key ='CY2006.Q2')
ORDER BY time;
```

Through these examples, it can be seen how SQL-based applications can use columns with hierarchical attributes to easily query cube views in the style of an interactive OLAP application.

Designing and Managing Cubes

At this point, it should be clear that the cube provides significant performance and analytic content benefits to SQL-based and dimensional business intelligence applications. Although cubes might at first seem exotic to someone who is accustomed to relational data types and data warehousing, they are not difficult to create and maintain.

There are two phases to the creation of a cube-based solution. First, there is the process of designing the cube. The design process will be driven by reporting and analysis requirements and is usually implemented using the OLAP administration tool, Analytic Workspace Manager. Second, there is the periodic refresh of the cube with new data. This can be driven from Analytic Workspace Manager and the materialized view refresh system.

Analytic Workspace Manager

Analytic Workspace Manager is designed to allow the Oracle application developer, database administrator or tech savvy line of business user create and manage a cube. Using Analytic Workspace Manager, the user works interactively and directly with the dimensional model and cube. This is

typically the case during prototyping and application development. The design focus of Analytic Workspace Manager is ease of use, modeling, and the definition of aggregations and other calculations.

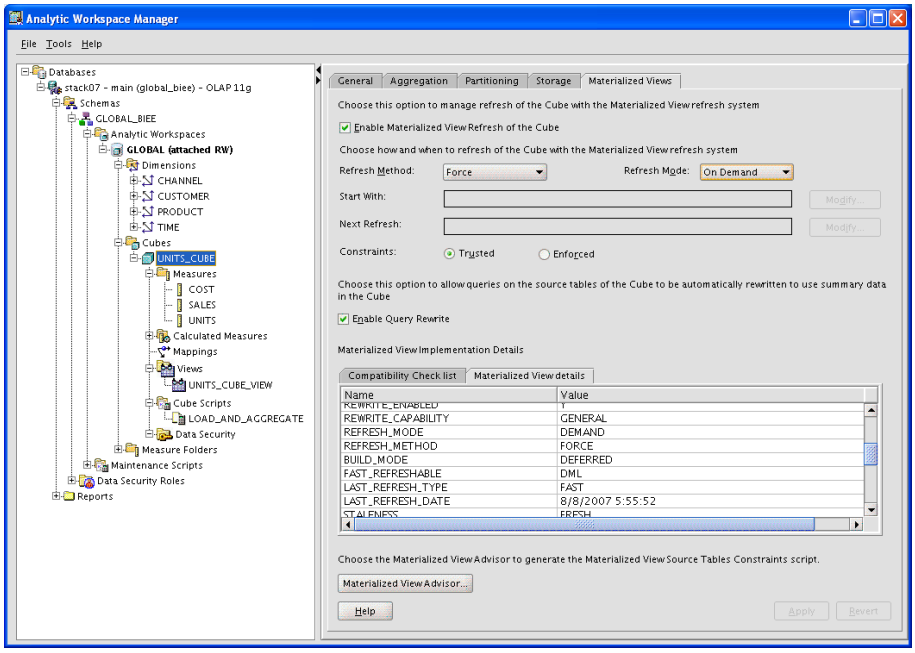
Analytic Workspace Manager allows the user to:

- Define the logical dimensional model (dimensions, hierarchies and cubes).
- Map the model to relational data sources (star, snowflake and other style schema).
- Define aggregations and measure calculations.
- Perform periodic maintenance such as data loading and re-aggregation.
- Support a collaborative administrative process by allowing models, or parts of models, to be shared with other users or applications via templates.

The typical process of creating a cube is as follows:

- Based on the reporting requirements of the user community, design the logical dimensional model (dimensions, hierarchies, cubes, base measures). The model can be updated at a later time as needed.
- Where appropriate, map the logical model to sources in Oracle Database. These sources might be tables, views, flat files as external tables or other relational objects.
- Build the analytic workspace for the first time. This involves loading data and, if desired, aggregating some data as a performance optimization.
- Define new measure calculations as needed.
- Review the design in the context of the BI tool or application that will be used to query the data.
- Revise as needed.

Analytic Workspace Manager supports this process from beginning to end in a single, dimensionally aware design environment.



Enabling the cube-organized materialized view for a cube using Analytic Workspace Manager.

Materialized View Refresh of the Cube

Once the cube is designed, the primary ongoing cube management task is the periodic update of data. This task can be accomplished using Analytic Workspace Manager or the materialized view refresh system. It is likely that organizations will choose to use the materialized view refresh system in production environments because it is easily scripted and improves performance of incremental refreshes.

The materialized view refresh system allows DBAs to refresh a cube-organized materialized view in the same way as a table based materialized view, by calling the DBMS_MVIEW.REFRESH program. DBMS_MVIEW.REFRESH takes the name of a materialized view as the primary argument. The database administrator does not need to know that the materialized view is cube-organized or even know that a cube exists – they only know the object as a materialized view. This makes cubes completely transparent from the perspective of day-to-day management.

A FAST refresh reads materialized view logs on the source tables for inserts, updates and deletes. These changes are applied incrementally to the cube. The cube is then incrementally pre-aggregated, with only the ancestors of new or changed members being recalculated.

A COMPLETE refresh will truncate fact data in the cube, load all data from the source table and aggregate the cube[5]. This behavior is just like a table based materialized view.

A FORCE refresh, like with table based materialized views, attempts to do a FAST refresh if possible and performs a COMPLETE refresh if a FAST refresh is not possible. If the Database cannot do a FAST refresh it will perform a complete load of data from the source table into the cube. The cube, however, can still be incrementally pre-aggregated after a COMPLETE refresh from the source table

Other Benefits of an Embedded OLAP Solution

At this point, it should be apparent that one of the main benefits of including cubes in the Oracle database is the ability for one cube to play three different rules: a summary management solution to SQL-based applications, a content rich data source to SQL-based applications and a multidimensional server to dimensionally oriented applications. These multiple uses, plus business intelligence metadata in the data dictionary, allow organizations to leverage the investment in the cube across any number of applications.

The discussion about materialized view refresh of the cube reveals the benefits of OLAP integrated into Oracle Database from the perspective of the database administrator. Materialized view refresh of the cube is designed to both make the cube more efficient to update (with automatic incremental loads from materialized view logs) and transparent to manage.

Transparency of the cube in the database is a common theme. This can be seen in SQL query and materialized view refresh. It can also been seen in many other aspects of database administration. Important highlights include:

- OLAP is embedded within the Oracle instance. There is no separate software to install or separate instance to manage. There is no requirement for addition server computers.
- OLAP cubes are stored in Oracle data files. Cubes are backed up and restored along with all other data in the database. No separate processes or special procedures are required.
- OLAP users are simply database users. There are no separate users or entities to create and manage.
- Oracle cubes are secured using standard Oracle database features. SQL GRANT and REVOKE is used to grant access privileges to OLAP cube, dimensions, cube views, dimension views and hierarchy views. Virtual Private Database can be used to control access to cube, dimension and hierarchy views. Definitions of fine-grained cube security are managed within the Oracle Extensible Data Security framework.

[5] Cubes are typically partially pre-aggregated as a performance optimization. In most cases, only a small amount of data is pre-aggregated and stored in the cube.

- OLAP is fully compatible with Real Applications Clusters and Grid Computing – enhancing both reliability and scalability.

In short, OLAP cubes are simply another data type in the database and are managed as such. A cube is a sophisticated data type that manages vast amounts of summary data and performs sophisticated calculations – and the design process must take this into account. However, the database administrator that has no specific knowledge of the cube or OLAP easily accomplishes day-to-day management of the cube.

Conclusion

OLAP cubes in Oracle Database 12c provide organizations with the means to improve the business intelligence applications they already use with improved query performance and analytic content. Using cube-organized materialized views, applications transparently access summary data managed within the cube for an immediate improvement in query performance. SQL applications can also be enhanced with the rich analytic content that is embedded within the cube. Simple SQL access to the analytic content of the cube greatly improves developer productivity and simplifies application development.

The OLAP Option is embedded within Oracle Database 12c that your organization already uses. As an embedded solution, OLAP cubes benefit from the architecture and features that make Oracle the leading enterprise relational database. Oracle cubes are highly secure, scalable and manageable. Organizations leverage the database, server computers and skills they already for an accessible and cost effective OLAP solution.

On-line Analytic Processing with Oracle Database 12c
January 2013
Author: William Endress
Contributing Authors:

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

Oracle is committed to developing practices and products that help protect the environment

Hardware and Software, Engineered to Work Together