

## CUBE ORGANIZED MATERIALIZED VIEWS, DO THEY DELIVER?

*Peter Scott, Rittman Mead Consulting*

### OVERVIEW

The recent 11g release of the Oracle relational database included many new or enhanced features that may benefit people creating Business Intelligence or Data Warehouse solutions. This paper will concentrate on one such feature - Cube Organized Materialized Views. This new feature exposes Oracle On-Line Analytical Processing (OLAP) cubes as relational materialized views, giving the possibility of SQL access to data stored in an OLAP cube. In the same way as relational materialized views, cube organized materialized views can support query rewrite; for example, aggregation queries written against the fact and dimension relational tables upon which the OLAP cube is built can be transparently rewritten to access the cube. This is a particularly valuable technique when using non-Oracle OLAP aware query tools. In addition to looking at query rewrite, this paper will also consider the use of materialized view refresh as a method to rebuild OLAP cubes. Materialized views exposing OLAP dimensions will also be discussed, as these are relevant to the methods of refreshing cube organized materialized views. Finally, alternative methods to access aggregated data and the relative merits of these approaches will be discussed.

### BACKGROUND

Before talking about the Oracle 11g functionality, it is useful to revisit the underpinning OLAP and Materialized View technologies.

### ORACLE OLAP

Oracle OLAP has its origins in an acquired technology originally marketed by Oracle as Oracle Express. This used a standalone OLAP server to store an OLAP (or ROLAP) cube. Oracle also sold two applications that used Oracle Express as the OLAP server - Oracle Financial Analyzer (OFA) and Oracle Sales Analyzer (OSA). Otherwise, few query tools (especially third-party ones) existed. Oracle 9i saw the initial integration of the Express engine into the relational database to allow a single point of management for both the cube structure and the relational database; various changes and enhancements were made to the Express DML at this time. Oracle introduced a release of Discoverer that was able to query Oracle OLAP and a Java BI Bean for developers, but the third-party query tool market remained small. Further integration in Oracle 10g gave us, amongst other things, *OLAP Views* - OLAP structures were exposed as database views which for the first time allowed non-Oracle OLAP aware query tools to access data from an OLAP cube. However the queries to access specific cube data required joins between multiple OLAP views and could look quite complex.

### MATERIALIZED VIEWS

The term 'materialized view' was first used in Oracle 8i; in Oracle 7 the predecessor technology was called 'snapshots'. Although the snapshot word may linger in some of the data dictionary tables, materialized views are more sophisticated than the database snapshots they replaced. There are two main uses for materialized views - data replication and data aggregation. It is the second use that is more relevant to this paper. A materialized view is basically a stored query definition where the query results are *materialized* or stored in a physical database table (this changes somewhat in Oracle 11g). The query definition becomes part of the metadata stored in the data dictionary and can be used to refresh the contents of the materialized view. This query definition, in association with other data dictionary data (notably constraints and database dimension objects), facilitates query rewrite; that is, where an SQL query is transparently rewritten by the query engine to access a 'better fit' materialized view than the original query table. Through the various releases of Oracle from 8i, the sophistication of the rewrite engine has greatly improved, as has the number of circumstances where you can use 'fast' materialized view refresh (an incremental update based on logged changes) to update the materialized view content. Oracle 10g also introduced rewrite equivalence, where the DBA can associate a specific query to another. This will be discussed later in this paper. From a BI perspective, query rewrite is a very significant feature in allowing many summary tables to be mapped to a reporting tool as a *single* base fact table.

### SOFTWARE

All of the Oracle software used in the preparation of this paper were released versions obtained from the Oracle Technology Network website (<http://www.oracle.com/technology/index.html>)

Initial investigations were carried out on 64-bit LINUX Oracle 11.1.0.6.0. Later work was carried out on 32-bit LINUX Oracle 11.1.0.6.0 with the p6459753 server and p6368282 Analytic Workspace Manager (AWM) client patches installed; these patches change the OLAP version to 11.1.0.6.0A and resolve many OLAP bugs found in the initial release of the 11g database and AWM client. Most investigation was carried out against a modified SH sample schema, but some tests also used the GLOBAL schema. The GLOBAL schema however is very small and densely populated so is not suited for many tests.

## WHAT ARE CUBE ORGANIZED MATERIALIZED VIEWS?

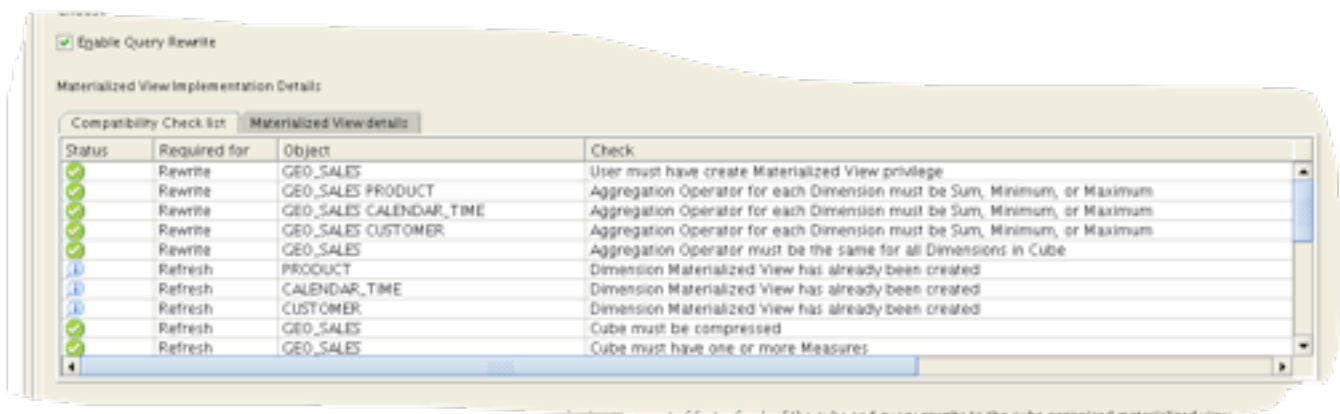
Put simply, a cube organized materialized view is an OLAP cube with an associated materialized view definition in the data dictionary. Unlike other materialized views, the storage container for the materialized view is the Oracle OLAP cube and not a simple table. The materialized view created over the cube can be used:

- directly in SQL (as in `select ... from CB$my_cube_name ...`)
- to refresh the contents of the materialized view using the `DBMS_MVIEW` family of PL/SQL packages
- in certain circumstances with query rewrite

Although an interesting usage, direct SQL query access to cube was already obtainable in Oracle 10g using OLAP views. The most significant features from a Data Warehouse/BI perspective are the use of a cube build method that is familiar to many DBAs - materialized view refresh; and the use of query rewrite to enable third-party non-OLAP aware query tools to access data stored within the OLAP cube by the use of simple aggregation queries on the base fact and dimension tables.

## DOCUMENTED LIMITATIONS

A number of limitations of cube organized materialized views are cited in the Oracle 11g OLAP User Guide and the patch and product install documentation. Some of these limitations might be removed in subsequent releases of Oracle OLAP as the technology is further developed. These limitations either place restrictions on the structure of the OLAP cube being used, or identify conditions that need to be met at permit query rewrite. Helpfully, the AWM client provides visual feedback on *some* of these conditions on the cube materialized view tab.



*AWM Cub Materialized view tab query rewrite conditions*

The following limitations exist on the creation of cube organized materialized views:

- The cube must be compressed
- The cube must have one or more measures
- The cube must have one or more dimensions
- Each dimension must have one or more hierarchies
- Each hierarchy must have one or more levels
- Only 'normal' hierarchies are supported - cannot use ragged or skip-level hierarchies
- The cube must be fully mapped, that is all the measures and dimension members are mapped to relational columns
- You cannot include calculated measures that are calculated by OLAP scripting in the cube build process

Additionally, for query rewrite:

- Only SUM, MINIMUM or MAXIMUM can be used to aggregate the dimensions
- The same aggregation operator must be used on all dimensions of the cube
- Supporting *relational* constraints and dimension objects are required for query rewrite to work. They are not, however, required to create the cube organized materialized view
- There is a 64 column limit for the materialized view GROUP BY

Some refresh mechanisms are not yet supported, these include log based refresh of an *aggregated* source fact table and partition aware fast refresh.

## **CREATING CUBE ORGANIZED MATERIALIZED VIEWS**

### **RELATIONAL STRUCTURES**

Before you can define an OLAP cube you need to have in place the underlying fact and dimensional (reference data) tables. The fact table should consist of the measures and dimensional keys, the dimensional data should either be fully denormalized (star schema) or in a parent-child snowflake form; it is probably easier to work with a star schema approach. All of the relational data must exist in database tables or materialized views, it not possible to create a materialized view based on a database view.

Database constraints and database dimension objects will be needed for query rewrite but these can be created from the script generated by the Relational Schema Advisor in AWM.

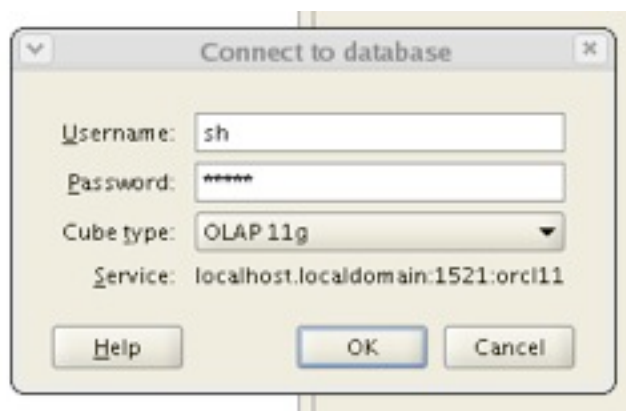
### **DESIGN NOTES**

An Oracle document (Instructions for Creating an Analytic Workspace From the Sales History Schema) suggests that for performance reasons, separate OLAP cubes should be created for each of the two time dimensions (fiscal or calendar) rather than a single dimension with two hierarchies. This is especially sound advice if the cubes to be constructed are partitioned on say, fiscal quarter as the calendar dimension data will only be in the *cap* partition. However, if you create two OLAP time dimensions, you should still create a single relational dimension, as it is not possible to refer to the same relational table column (in this case *time\_id*) in more than one relational dimension.

The SH schema was modified to include additional columns required in an OLAP time dimension and a materialized view constructed to pre-join countries (geography) with customers so that a simple star schema model could be used.

### **OLAP STRUCTURES**

The easiest way to create cube organized materialized views is through the AWM client application. When connecting to the database you need to select the type of OLAP cube you will be developing. To use these new features you should select OLAP 11g, then navigate through the the schema tree and create a new analytic workspace, remembering to specify an appropriately sized tablespace.

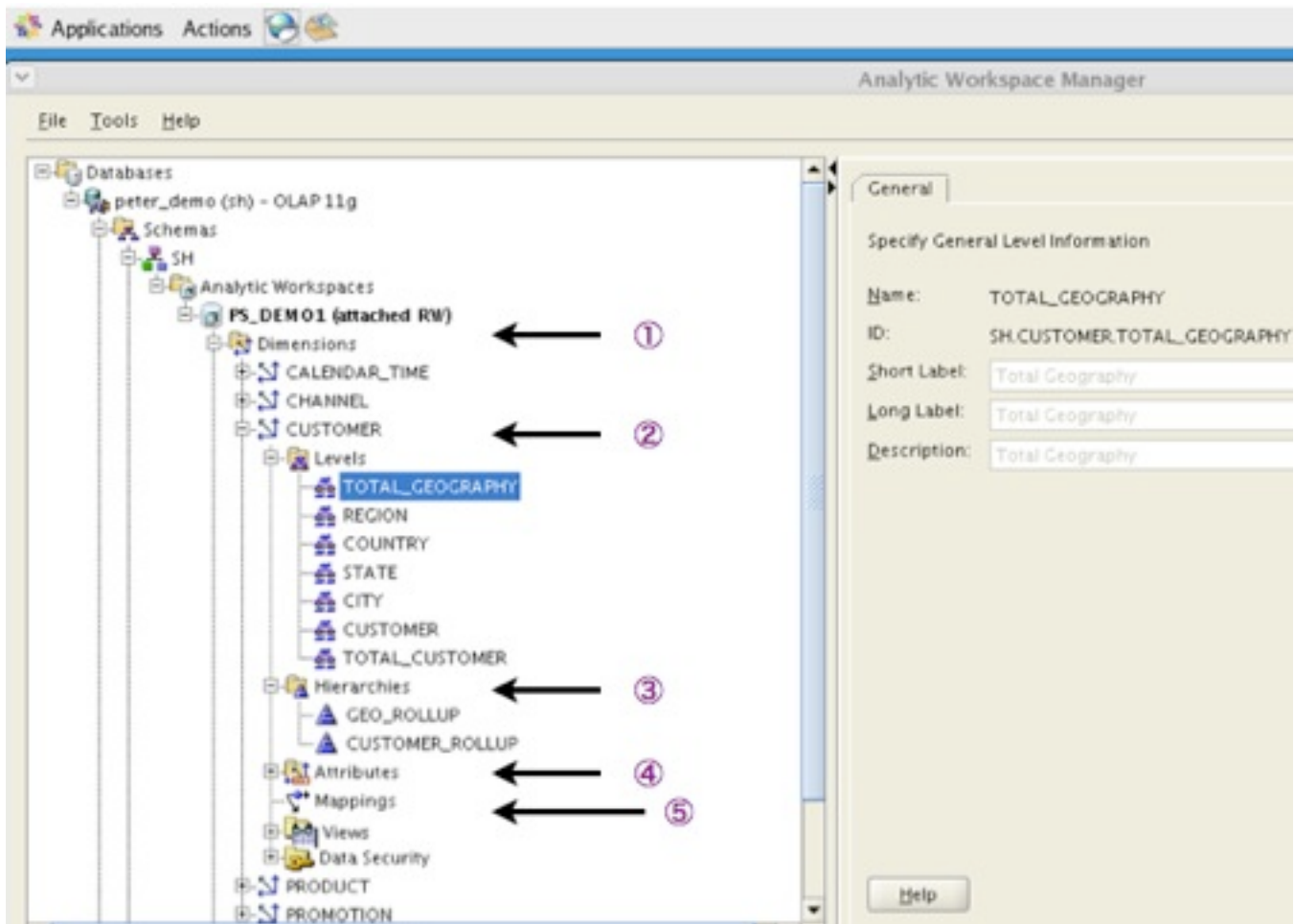


*Cube type selector on AWM log-on screen*

### CREATING THE DIMENSIONS

After creating the empty workspace the dimensions are created in the normal way. For each dimension:

- create the empty dimension
- define the levels
- define the hierarchies for the levels
- define any attributes
- map the dimension levels and attributes to the relational table(s)



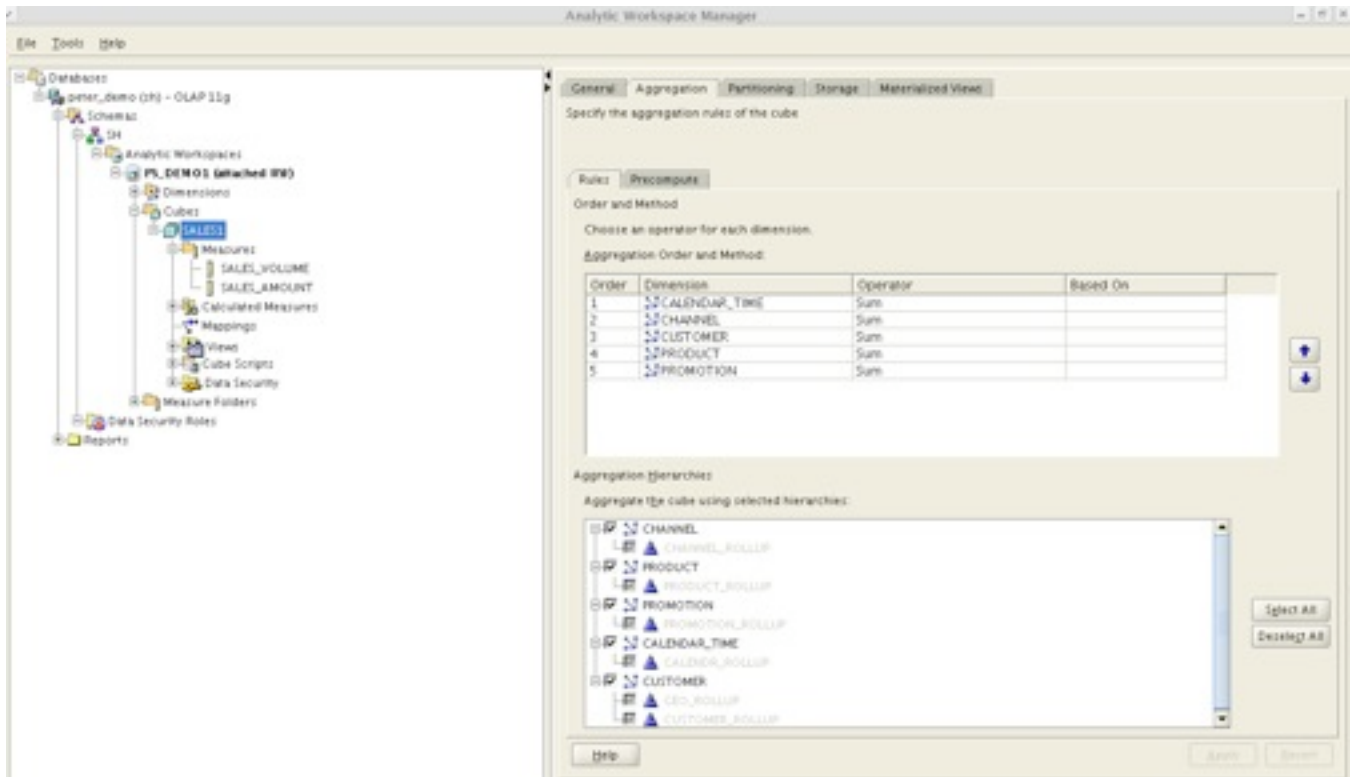
#### Creating OLAP dimensions in AWM

After the levels, hierarchies and mappings are defined, the status indicators on the dimension object's materialized view tab will indicate that it is possible to create a dimension materialized view. Click the check box and click on the APPLY button - the dimensional materialized view is created. This process does not affect the now automatic creation of OLAP views over the dimension and hierarchies.

When all of the required dimensions have been defined, the cube can be created.

### CREATING THE CUBE

From the cube level of the workspace navigation tree, right mouse click to access the create cube worksheet. This worksheet now has five main tabs on it but only the first tab (General) can be accessed until a cube name and the required dimensions are selected. The next tab (Aggregation) has two sub-tabs; *rules* to define the required hierarchies and aggregation rules, and *precompute* where the aggregation method is selected. Only the aggregation rules sub-tab needs to be completed. The other tabs cannot be used until the fact measures and the table mappings have been defined.



*AWM Cube Aggregation rules sub-tab*

### **PARTITIONING TAB**

This tab allows a partitioning scheme to be chosen for the cube. Alternatively, an advisor is available that will make a partitioning suggestion based on whether a time dimension or statistics analysis based scheme is required. For the SH schema, the advisor recommends to partition at the calendar quarter level. Accepting the advisor's advice triggers a prompt to rebuild the cube based on the new metadata. This should be deferred until after the storage and materialized view definitions are completed. If you decide to partition the cube manually, this can only be done once at cube creation time; whereas it is possible to rerun the partitioning advisor.

Partitioning the cube can give significant advantages in query performance and cube maintenance.

### **STORAGE TAB**

Strangely, the storage tab is not described in the 11g OLAP user guide. This tab gives the option to choose to create a compressed cube and to arrange dimension order for optimal storage. However, to create a cube organized materialized view it is mandatory to use cube compression. Again there is an advisor available to suggest the best approach. Note, for the GLOBAL schema it recommends that the cube is not compressed. As always, there is the option to decline the advice given.

### **MATERIALIZED VIEWS TAB**

Three main options are set from this tab:

- Whether to create a materialized view
- How the cube is refreshed
- Whether to permit query rewrite

Checking the create materialized view box allows access to the refresh options and rewrite checkbox.

Like conventional, relational, materialized views there are several options for the method of refresh and the event that triggers the refresh. There is one new refresh method, FAST\_SOLVE that is specific for cube organized materialized views. This method incrementally refreshes the cube from the source data rather than using materialized view logs over the source tables. FAST\_SOLVE can handle more complex cubes than the dimensions and simple aggregates than that can be resolved in a fast refresh. Probably the most usual form of refresh is the FORCE refresh this will initially try for a FAST refresh, and if this is

not possible, a FAST\_SOLVE refresh. Options to trigger refresh include on commit, on demand and at fixed intervals; with ON DEMAND probably being the most useful in a production OLAP system; on commit runs the risk of the cube organized materialized view refreshing at inappropriate times and interval based refresh regimes may be inflexible to operating requirements.

Selecting the query rewrite option adds additional items to the list of prerequisites for cube organized materialized view creation. It is not possible to accept the creation of the materialized view until the prerequisites are satisfied, that is are marked with either a green or blue blob. Pressing accept creates the cube organized materialized view but does not force the population of the underlying cube.

A relational schema advisor (for which the *advisor* database privilege is required) will generate a script to create the necessary constraints and relational dimensions to maximize the potential of query rewrite - the generated script should be considered carefully before implementation especially if more than one OLAP dimension shares a database column. It should also be noted that constraint DDL is generated in alphabetic table order and some foreign keys may be generated before their parent primary keys, which raises an error in Oracle 11g.

#### *AGGREGATION - PRECOMPUTE SUB-TAB*

This tab permits the level of pre-computation of aggregates to be selected. By default, AWM specifies a 20% aggregation of the cube, or in the case of partitioned cubes 20% for the base partitions and 0% for the cap partition - the partition that contains all data that does not fit in any other partition. The 20% value means around 1/5 of the cube is pre-populated with data and the remaining data is fetched from the relational source at query time. These percentage numbers should be considered as relative degrees of pre-aggregation; 20% is more aggregated than 5%, but neither number corresponds exactly to a fraction of the data set stored in the cube. In general terms the higher the percentage of pre-aggregation, the faster the response to queries but at the expense of cube build time and disk real-estate.

Specify the aggregation rules of the cube

Rules Precompute

Choose an aggregation method:

Cost-based aggregation (recommended for compressed cubes)

Top Partition:

Bottom Partition:

Level-based aggregation (required for uncompressed cubes)

Choose the levels of the cube to be aggregated and stored.

Dimension:

Dimension	Levels
CHANNEL	<input type="checkbox"/> CHANNEL
PRODUCT	<input type="checkbox"/> CHANNEL_CLASS
PROMOTION	<input type="checkbox"/> TOTAL_CHANNEL
CALENDAR_TIME	
CUSTOMER	

Select All  
Deselect All

*AWM Cube Aggregation Precompute sub-tab*

## **REFRESHING THE CUBE ORGANIZED MATERIALIZED VIEW**

As in Oracle OLAP 10g, cubes can be refreshed from the AWM. However for production purposes, cube refresh should be scheduled or at least scripted. SQL commands to refresh materialized views can be generated from one of the refresh wizard pages in AWM. Fundamentally, the refresh method boils down to a choice between the DMBS\_CUBE.BUILD or DMBS\_MVIEW.REFRESH families of database packages.

In wall-clock timing, a complete materialized view refresh is marginally faster, within experimental error, than a cube build, but the materialized view refresh does not automatically refresh the dimension materialized views and they should be refreshed before the cube. This requirement to refresh dimensions first, means that a REFRESH\_ALL option is not appropriate. One advantage of using materialized view refresh is that the same calls can be used to refresh conventional (relational) materialized views. These calls are probably already familiar to a DBA looking after a data warehouse.

Refresh scripts that execute OLAP DML to create measures are not compatible with cube organized materialized views, because the measures so created cannot be expressed in relational terms. As stated previously, the time required to rebuild a cube organized materialized view depends on the degree of pre-computation of aggregates requested. Another factor that influences rebuild time is the amount of parallelism that can be achieved within the system; this is particularly the case with partitioned cubes.

## **QUERY REWRITE**

For query rewrite to occur, the query optimizer must believe that the results stored in the materialized view can be trusted. This is achieved through a combination of constraints (not null, foreign key, primary key) and relational dimension objects. In addition the optimizer must be informed that it is allowed to trust the content of the materialized view. This is achieved by setting QUERY\_REWRITE\_INTEGRITY to either STALE\_TOLERATED or TRUSTED.

## **METHODOLOGY**

The majority of the query tests conducted for this paper were against a simple two measure cube built from the SALES table of the SH schema. As installed (the SH schema is part of the Oracle sample set) the SALES table is time partitioned (at the quarter level) and bitmap indexed on the dimension keys. It however is very small in comparison with a real-world data warehouse. The SALES table only has five dimensional keys, which again is somewhat small. The cube organized materialized view built over the SALES table was partitioned on calendar quarter and dimensioned by time, product, customer, channel and promotion. A simple test query was constructed to aggregate sales data by the dimensions and for either a fixed year or month. By adjusting the query\_rewrite\_integrity setting query, rewrite could be disabled or re-enabled at will, an example query is given below:

### *TEST QUERY 1*

```
SELECT country_name,
       SUM(quantity_sold),
       channel_class, PROD_CATEGORY
FROM sales s,
       channels ch,
       sh_customer cu,
       products p,
       times t
WHERE s.cust_id = cu.cust_id
     AND s.channel_id = ch.channel_id
     AND s.time_id = t.time_id
     and s.prod_id = p.prod_id
     AND t.calendar_year = 2000
GROUP BY country_name,
         channel_class, PROD_CATEGORY;
```

When executed against the relational schema and with rewrite disable, the query executed in around two seconds and returned 188 rows. The query plan (from auto trace) is given below.

Elapsed: 00:00:02.01

Execution Plan

-----  
 Plan hash value: 1396963405  
 -----

Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		11	1122		1761 (3)	00:00:22		
1	HASH GROUP BY		11	1122		1761 (3)	00:00:22		
* 2	HASH JOIN		229K	22M		1748 (3)	00:00:21		
3	TABLE ACCESS FULL	PRODUCTS	72	1512		3 (0)	00:00:01		
* 4	HASH JOIN		229K	17M		1743 (2)	00:00:21		
5	TABLE ACCESS FULL	CHANNELS	5	55		3 (0)	00:00:01		
* 6	HASH JOIN		229K	15M	2968K	1738 (2)	00:00:21		
7	MAT_VIEW ACCESS FULL	SH_CUSTOMER	64571	2207K		557 (1)	00:00:07		
* 8	HASH JOIN		229K	7855K		522 (5)	00:00:07		
9	PART JOIN FILTER CREATE	:BF0000	365	4380		18 (0)	00:00:01		
* 10	TABLE ACCESS FULL	TIMES	365	4380		18 (0)	00:00:01		
11	PARTITION RANGE JOIN-FILTER		918K	20M		497 (4)	00:00:06	:BF0000	:BF0000
12	TABLE ACCESS FULL	SALES	918K	20M		497 (4)	00:00:06	:BF0000	:BF0000

-----  
 Predicate Information (identified by operation id):  
 -----

```

2 - access("S"."PROD_ID"="P"."PROD_ID")
4 - access("S"."CHANNEL_ID"="CH"."CHANNEL_ID")
6 - access("S"."CUST_ID"="CU"."CUST_ID")
8 - access("S"."TIME_ID"="T"."TIME_ID")
10 - filter("T"."CALENDAR_YEAR"=2000)

```

Note

-----  
 - dynamic sampling used for this statement

Statistics

```

-----
33 recursive calls
0 db block gets
3119 consistent gets
1941 physical reads
0 redo size
8446 bytes sent via SQL*Net to client
552 bytes received via SQL*Net from client
14 SQL*Net roundtrips to/from client
6 sorts (memory)
0 sorts (disk)
188 rows processed

```

A typical plan for a query against the OLAP cube is given below. Note the execution time (for a precompute level of 20%) is around 9 seconds, nearly 5 time longer than a straight query against the relational table.

Elapsed: 00:00:08.57

Execution Plan

-----  
 Plan hash value: 4075507672  
 -----

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		11	1188	332 (40)	00:00:04
1	HASH GROUP BY		11	1188	332 (40)	00:00:04
* 2	HASH JOIN		1543K	158M	237 (15)	00:00:03
3	VIEW		18	540	4 (25)	00:00:01
4	HASH UNIQUE		18	378	4 (25)	00:00:01
5	TABLE ACCESS FULL	PRODUCTS	72	1512	3 (0)	00:00:01
* 6	HASH JOIN		428K	31M	222 (11)	00:00:03
7	VIEW		5	105	4 (25)	00:00:01
8	HASH UNIQUE		5	55	4 (25)	00:00:01
9	TABLE ACCESS FULL	CHANNELS	5	55	3 (0)	00:00:01
* 10	HASH JOIN		257K	13M	215 (9)	00:00:03
11	VIEW		4	52	19 (6)	00:00:01
12	HASH UNIQUE		4	32	19 (6)	00:00:01
* 13	TABLE ACCESS FULL	TIMES	365	2920	18 (0)	00:00:01
* 14	CUBE SCAN	CB\$SALES1	321K	13M	193 (8)	00:00:03

-----  
 Predicate Information (identified by operation id):  
 -----

```

2 - access("from$_subquery$_013"."PROD_CATEGORY_ID"=SYS_OP_ATG(VALUE(KOKBF
$),11,12,2))

```

```

6 - access("from$_subquery$_016"."CHANNEL_CLASS_ID"=SYS_OP_ATG(VALUE(KOKBF
    $),6,7,2))
10 - access("from$_subquery$_010"."CALENDAR_YEAR_ID"=SYS_OP_ATG(VALUE(KOKBF
    $),22,23,2))
13 - filter("CALENDAR_YEAR"=2000)
14 - filter(SYS_OP_ATG(VALUE(KOKBF$),45,46,2)=81641727)

```

#### Statistics

```

-----
34 recursive calls
1600 db block gets
329401 consistent gets
5396 physical reads
0 redo size
8065 bytes sent via SQL*Net to client
552 bytes received via SQL*Net from client
14 SQL*Net roundtrips to/from client
9 sorts (memory)
0 sorts (disk)
188 rows processed

```

As can be seen from the query plan, rewrite against the cube occurred but access to some of the dimension tables was still required to resolve the query. In effect, we are returning more data from the cube than is needed for our query and then using relational filters to get the required data. This dividing of a larger query into rewritable query blocks and then combining the results of the blocks is typical of the way the query rewrite mechanism works.

The key fact is that we used very simple SQL to access data in the OLAP cube without needing to know the name of the cube being used. It is unsurprising that the relational query was faster, given the simplicity of the query involved, the fact that the sales table was partitioned on quarter and the fact that the cube was only partially pre-aggregated so that a significant amount of data still had to be retrieved from the relational database.

## QUERY PERFORMANCE

However, simple queries such as the one given above are atypical of an ad-hoc query load generated by a user query tool. Many users need to access data at a much finer level of detail across just one or two dimensions and over a shorter range of dates. Perhaps a better example query is the one given below:

### TEST QUERY 2

```

SELECT country_name,
       SUM(quantity_sold),
       channel_class, PROD_CATEGORY
FROM sales s,
     channels ch,
     sh_customer cu,
     products p,
     times t
WHERE s.cust_id = cu.cust_id
     AND s.channel_id = ch.channel_id
     AND s.time_id = t.time_id
     and s.prod_id = p.prod_id
     AND t.calendar_quarter_desc = '2000-01'
     and COUNTRY_REGION = 'Europe'
     and prod_category <> 'Photo'
GROUP BY country_name,
         channel_class, PROD_CATEGORY
order by 1,3;

```

For this test the query was executed (twice) against a series of cubes built at differing levels of pre-aggregation. In addition, the time taken to completely refresh the cube (on the same single processor virtual machine) and the space taken by the resulting cube were recorded. Each test only considered the performance of cube organized materialized views; relational access against the base table was not investigated.

<i>% PRECOMPUTE</i>	<i>BUILD TIME (SECONDS)</i>	<i>QUERY TIME 1 (SECONDS)</i>	<i>QUERY TIME 2 (SECONDS)</i>
0	250	600	-
5	462	46	5.6
20	600	14	2
30	640	4	0.5
35	700	2	0.2

*Table - effect of pre-computation of aggregates*

As expected the time to maintain the cube increased with the degree of pre-computation. The query times for a 30% pre-computed cube were not much different to those for a relational query. For 35% pre-computation, the cube performance bettered that of a relational query. Little increase in cube size was noted for increasing pre-aggregation; this is to be expected as the the test data cube only utilized two measures.

## **ALTERNATIVE APPROACHES TO MATERIALIZED CUBE VIEWS**

### **RELATIONAL TECHNIQUES**

Traditionally, reporting systems have relied on pre-built relational summary tables to aggregate selected dimensional level combinations. These summary tables were often chosen to meet the requirements of key business queries. However, a few well chosen summary tables could satisfy a large proportion of a BI system's general workload. One particular problem associated with the use of summary tables was that they would need to be mapped into some form of 'aggregate aware' query tool, so that the most appropriate table was chosen to resolve an ad-hoc query; for many query tools this was not a trivial task. The use of Materialized Views and their query rewrite capabilities was a significant step forward in the ease of use of a summary table approach. Far fewer summaries needed to be mapped into the query tool (perhaps just one) and the query optimizer in the database selected the best fit summary.

In Oracle 10g an extension to the GROUP BY syntax allowed the construction of the relational equivalent of a cube and if used to construct a materialized view would allow query rewrite. In fact the query definition of an OLAP cube organized materialized view is very similar to the one required to build a relational materialized view using the GROUP BY ROLLUP or GROUP BY GROUPING SET extensions. The principal difference with the use of relational summaries and OLAP summaries is that the relational summaries are fully computed (or built.) This means they are potentially larger and probably take a significant time to refresh. On the plus side they should be very fast to return results, especially if appropriately partitioned and indexed.

### **OLAP CUBE ACCESS VIA SQL**

Unlike the MDX data cube format used by some other vendors, there are few products with native support for querying Oracle OLAP cubes. However, SQL access to OLAP cube and dimension data through relational views was introduced in Oracle 10g. This enabled specific queries against OLAP data to be constructed; and if registered through the rewrite equivalence functionality of Oracle 10g could be accessible to simple SQL against a 'base' table. However rewrite equivalence only registers specific query pairs as equivalent and does not allow for a more generalized rewrite to other parts of the OLAP cube.

## **CONCLUSIONS**

Cube Organized Materialized Views do both of the things claimed of them: they allow the cube content to be refreshed through calls to DBMS\_MVIEW.REFRESH (and are able to utilize various FAST refresh options), and they permit query rewrite to occur. They do not however allow access to some of the more sophisticated Oracle OLAP functionality such as time series aggregation; this is not surprising as there is a need to be able to present the query that underpins the view definition as 'straight SQL'.

Materialized view refreshing of cubes is straight forward and uses an already familiar syntax, but it needs to be remembered that the cube and each dimension are stored as separate materialized views and that the dimensions must be refreshed prior to the cube.

At present, the query optimizer over-favors the use of cube organized materialized views, which can lead to degraded performance of some queries where SQL against the base table is perhaps a better choice. This is the first release of a new technology and Oracle often improves optimizations in subsequent releases.

The true utility of the cube organized materialized view is that it allows a single database object to be created as an alternative to multiple summary tables. By choosing the degree of pre-aggregation and the cube partitioning schemes, it will be possible to build a cube that performs well yet balances the time to build; tests have shown that a pre-compute level of 35% gives good query performance with acceptable maintenance performance. Of course, for a real-world data set it would be necessary to benchmark before deploying to production.

### **SPEAKER BIOGRAPHY**

Peter Scott is a principal consultant at Rittman Mead Consulting, a leading BI and DW consultancy based in the UK. Peter has worked with Oracle databases since Oracle 7.3 and has specialized in the design and management of terabyte scale data warehouses for many years. He has particular interests in Data Warehouse performance optimization and data modeling. Peter contributes to the wider Oracle community through his Blog writing and through speaking at user group meetings.

### **BIBLIOGRAPHY**

The following Oracle resources were consulted in the preparation of this paper:

- Oracle Database, Data Warehousing Guide 11g Release 1 (11.1): Publication B28313-02
- Oracle OLAP User's Guide 11g Release 1 (11.1): Publication B28124-02
- Instructions for Creating an Analytic Workspace From the Sales History Schema: Oracle on-line document [http://www.oracle.com/technology/products/bi/olap/doc\\_sample\\_schemas/shschema.html](http://www.oracle.com/technology/products/bi/olap/doc_sample_schemas/shschema.html)
- Oracle White paper: Comparing Materialized Views and Analytic Workspaces in Oracle Database 11g (March 2008) [http://www.oracle.com/technology/products/bi/db/11g/pdf/comparision\\_aw\\_mv\\_11g\\_twp.pdf](http://www.oracle.com/technology/products/bi/db/11g/pdf/comparision_aw_mv_11g_twp.pdf)