Comparing Materialized Views and Analytic Workspaces in Oracle Database 11g

An Oracle White Paper
March 2008

Comparing Materialized Views and Analytic Workspaces in Oracle Database 11g

Materialized Views and Analytic Workspaces – Contrasting Approaches in Oracle Database 11g

INTRODUCTION

Businesses today require information in a timely fashion. It is no longer acceptable to produce a report once a day, now some people need it hourly or even every few minutes. To be responsive to the business need for analysing an ever growing set of business data, requires performing any analysis in a timely manner. This is an increasing challenge when the analysis is over vast quantities of data and can involve time consuming operations such as joins and aggregations and performing other analytical functions and comparisons. Being able to pre-compute these operations can deliver fast query response times and quicker analysis.

This white paper discusses the two approaches available in Oracle Database 11g for pre-computing these query operations and improving query response times: Materialized Views with Query Rewrite and cubes in Analytic Workspaces.

FAST REPORTING – COMPARING THE APPROACHES

Today, terabyte databases are not uncommon. When none of the techniques described in this white paper are used, queries will access this base data directly which can result in a considerable delay before the results are returned to the user. Therefore, to be able to improve their performance, parts of these query operations must be performed in advance. This can be done using materialized views that contain pre-computed results, and query rewrite that transparently rewrites the SQL query to access the materialized views, or it can be performed using Analytic Workspaces and cubes. Although in Oracle Database 11g, analytic workspaces can now also take advantage of materialized views and query rewrite.

Firstly we need to get a better appreciation of these two features within the database. Throughout the white paper, it is assumed that the reader is familiar with data warehousing fundamentals, such as fact tables and dimensions and hierarchies, and how they are used and implemented by the Oracle Database.

Overview of Materialized Views

Materialized Views and Query Rewrite are parts of a feature known as Summary Management that has been included in the database since Oracle 8i. Materialized Views pre-compute and store the results of a database query that can optionally involve a join, an aggregation or both. Like an index, a materialized view requires space in addition to that required for the tables. In terms of the benefit they bring to our problem of query performance, materialized views have already performed the time consuming joins and aggregation prior to them being utilised to answer the user's query.

The real power behind materialized views and query rewrite is that their use is transparent to the user. In the same way that a user doesn't have to know about the indexes on a table to use them, then likewise, a user doesn't have to know about the presence, structure and content of the materialized view. Query Rewrite enables this transparent use of materialized views and is a query optimisation mechanism whereby the original query SQL, which is written against the base tables, is automatically rewritten by the optimiser to access the appropriate materialized views.

Overview of Analytic Workspaces

An alternative approach is to use Oracle OLAP, which has been an option for Oracle Database Enterprise Edition since Oracle 9i. Oracle OLAP allows you to store your data in a special format inside the database using cubes and dimensions that enables fast analysis and query reporting.

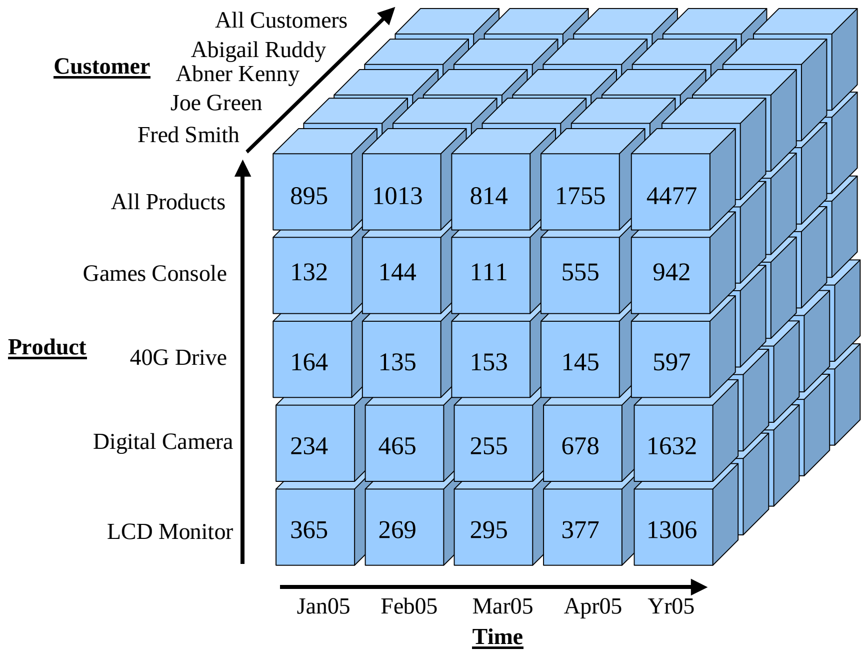
Traditionally, queries against a data warehouse were created to answer questions like *'How much profit did each regional division make?'* but organizations were quickly realizing that the wealth of data in their warehouse enabled much more interesting and sophisticated reports and analysis to be performed. Now a typical query might be *'For calendar quarters this year, show the percentage change in sales for Electronic products sold to customers who live in the USA compared to the same quarters last year'*.

This type of query is often called a *multi-dimensional* query and business analysts create these queries on-line and 'slice and dice' and refine their view of the data to uncover and highlight underlying trends and hence *online analytical processing*, or OLAP, was born. The multi-dimensional query expresses a business question in terms of the multiple dimensions that describe the data. In our example above, the data is Sales data and the dimensions are Time, Products and Customers (in this case, the geographical location where the customers live in the USA). This type of query can be difficult to express relationally using SQL, but would be simple to describe multi-dimensionally using the Oracle OLAP option.

Analytical processing still has to store the data it accesses in a specialized format within the Oracle database and this is an integral part of the Oracle OLAP implementation: data is composed of cubes containing the measures (the data) and whose 'edges' are the dimensions that define their levels and hierarchies. The diagram below shows one cube that is defined by three dimensions, Time, Product and Customer, and the cells within the cube are the data values, (which are also known as *measures*). In Figure 1, the dimensions have been simplified and only depict the lowest levels, for example the months on the Time dimension, but dimensions also incorporate hierarchies that define the grouping of the data. For example, that Months roll up to Quarters that roll up to Years.

Within Oracle Database 11g, the OLAP Option provides specialized storage via the analytic workspace and processing for multidimensional data, using the Multidimensional Calculation Engine.

Figure 1 An Analytical Workspace Cube



The *Analytic Workspace* (AW) is used to store the multidimensional data types, e.g. the dimensions, measures and cubes. An Oracle database schema can contain one or more analytic workspaces in addition to owning the normal relational objects such as the tables, indexes and materialized views.

The *Multi-dimensional Calculation Engine* provides the calculation functionality that enables the user to create sophisticated analytical queries that execute efficiently. For example, queries that can show trends in the data by comparing results to previous time periods or to other groupings of the data such as product categories or geographic regions. The engine executes the analytical queries but also enables forecast and model trends to be constructed, and to run other "what if" types of examinations that are also commonly performed analytical operations.

In addition, there is the SQL interface that enables tools to use regular SQL to query the analytic workspace. The SQL is transformed by the database to operate against the analytic workspace objects and the results returned as rows and columns by the SQL interface back to the SQL query.

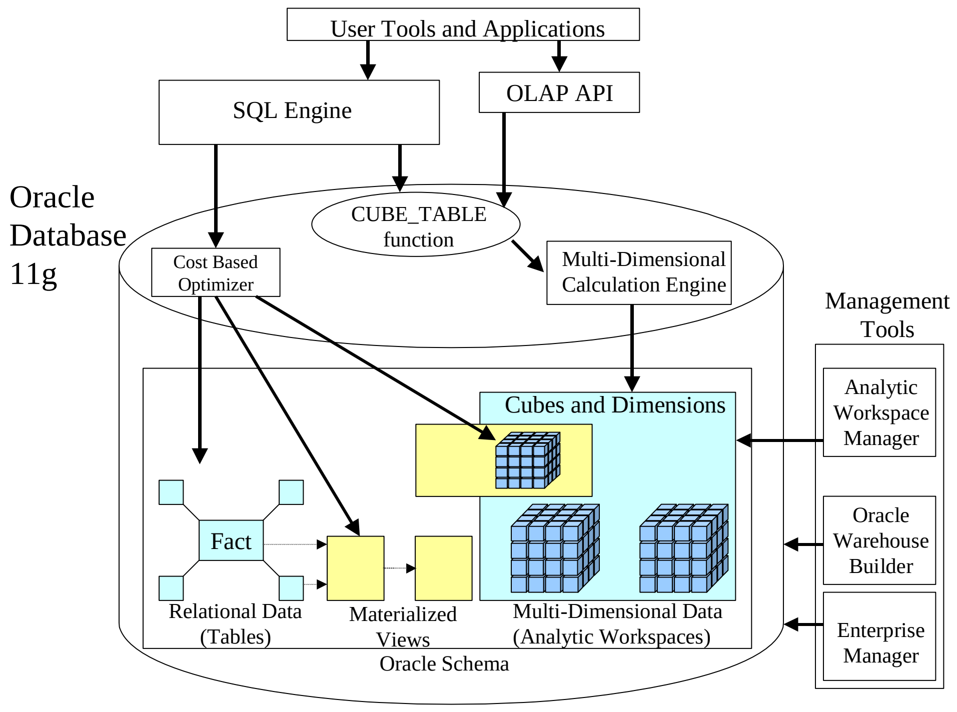
Finally, there is the OLAP API. This is a programming interface that enables tools and applications to access the analytic workspace and calculation engine directly.

Oracle products, such as Oracle Business Intelligence Discoverer 10g and the Excel Spreadsheet Add-In, use the OLAP API to access analytic workspaces.

Working Together in the Oracle Database

Figure 2 illustrates how analytic workspaces sit alongside materialized views and the relational data in Oracle Database 11g and the different tools used for their administration. Materialized views are used for pre-computed results for both relational and multi-dimensional data: the cost based optimizer can now rewrite SQL queries to the analytic workspace and materialized views can be defined over the cubes.

Figure 2 Relational and Multidimensional Data in Oracle Database 11g



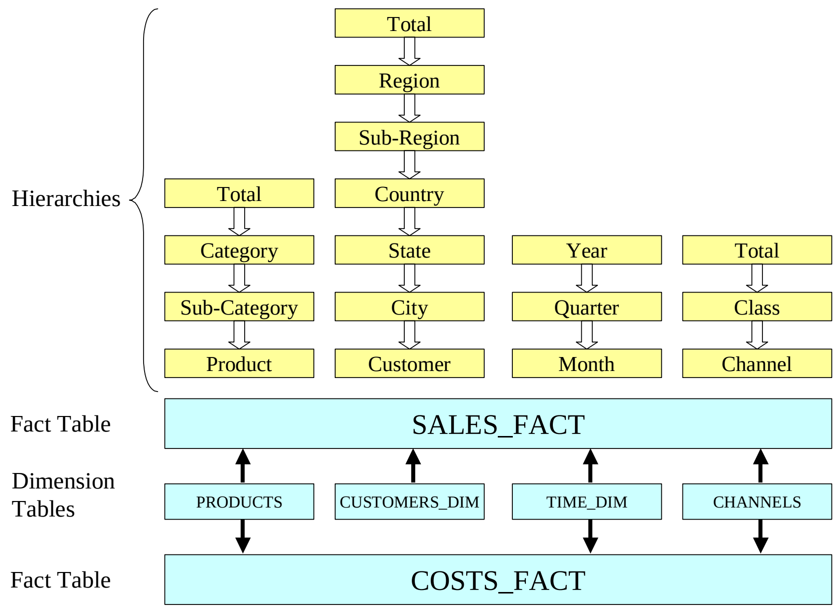
EXPLANATION OF DATA USED FOR THIS WHITE PAPER

We will begin this comparison by first explaining the data being used and then stepping through the methodology used to create the materialized views and analytic workspace.

The data has been derived from the standard Sales History (SH) example schema provided on the Companion CD. The SALES data is used for the fact table however it has been simplified to aggregate out and remove the Promotions dimension and also to aggregate the fact data up the Time dimension from the daily level to the monthly level, i.e. the month is now the bottom of the Time dimension. Product COST data is also used for another fact table and has been similarly modified to remove the Promotions dimension and have the month as the lowest level of the Time dimension. Therefore, our dimensions are Product, Channel, Customer and Time (Month).

Figure 3 shows the resulting fact and dimension tables with the hierarchies and their levels in each dimension.

Figure 3 The Query Data, Tables, Dimensions and Hierarchies



Methodology for Designing and Building the Materialized Views

There are two methods that can be used to define our materialized views; create them manually or generate them from advice given by the SQL Access Advisor.

Manually Creating the Materialized View

Step 1 Understand the Queries

The starting point for creating materialized views is to understand the types of joins and aggregations that are used in the users queries which can then be analyzed to identify the set of materialized views necessary to support them. The goal for the database designer is to create as few materialized views as possible that support the widest range of queries.

Step 2 Create the Materialized View

Let's take the example of a query about customer spend by geographic area and month. This query will benefit from a materialized view that pre-computes the join between the SALES_FACT fact table and the TIME_DIM and

CUSTOMERS_DIM dimension tables and performs the aggregation functions. The SQL for creating the materialized view is shown below:

```
CREATE MATERIALIZED VIEW state_sales_mv
BUILD IMMEDIATE REFRESH FAST ENABLE QUERY REWRITE AS
SELECT c.cust_state_province, c.country_iso_code, m.month_desc,
    min(amount_sold) min_amount, max(amount_sold)   max_amount,
    avg(amount_sold) avg_amount, count(amount_sold) cnt_amount,
    count(*) count_all
FROM  sales_fact s, time_dim m, customers_dim c
WHERE s.month_id = m.month_id AND s.cust_id = c.cust_id
GROUP BY c.cust_state_province,c.country_iso_code, m.month_desc
```

For large materialized views, a partitioning clause can be added to the statement above, which brings the same benefits and advantages to materialized views as partitioning brings to large tables, namely, improvements to scalability, maintenance and query performance. In addition, using database partitioning also introduces better refresh possibilities using parallel DML.

Step 3 Define Dimensions

Some queries can benefit from the definition of SQL dimensions because they will allow more types of query rewrite to occur. SQL Dimensions are Oracle objects that define the hierarchical parent/child relationships and are highly recommended because they provide additional information to query rewrite to enable better rewrite decisions to be made

The CREATE DIMENSION statement for our TIME dimension is shown below and illustrates the use of a single hierarchy (CALENDAR) that has three levels and how data value attributes are associated with each of these levels.

```
CREATE DIMENSION SH.MONTH_DIM
LEVEL MONTH   IS (MONTH_STAR.MONTH_ID )
LEVEL QUARTER IS (MONTH_STAR.QUARTER_ID)
LEVEL YEAR    IS (MONTH_STAR.YEAR_ID)
HIERARCHY CALENDAR (MONTH CHILD OF QUARTER CHILD OF YEAR)
ATTRIBUTE MONTH   DETERMINES (MONTH_STAR.MONTH_DESC,
       MONTH_STAR.MONTH_ENDDATE, MONTH_STAR.MONTH_TIMESPAN)
ATTRIBUTE QUARTER DETERMINES (MONTH_STAR.QUARTER_DESC,
       MONTH_STAR.QUARTER_ENDDATE, MONTH_STAR.QUARTER_TIMESPAN)
ATTRIBUTE YEAR    DETERMINES (MONTH_STAR.YEAR_DESC,
       MONTH_STAR.YEAR_ENDDATE, MONTH_STAR.YEAR_TIMESPAN)
```

Step 4 Build Indexes on the Materialized View

Just as indexes on tables can benefit queries against those tables, so can indexes on materialized views be used to improve the performance of queries that are re-written to utilise the materialized view.

The same rules for choosing indexes on tables should be followed when deploying indexes against materialized views:

- Index columns in the materialized view which are the primary key columns of the base tables (because the materialized view can be joined to the base table for certain types of query rewrite), and,
- Index columns which are used in the WHERE clauses of user queries.

Remember that the indexes will be maintained as part of the refresh of the materialized view so it is a good idea to constrain the total number of indexes deployed so that the refresh time is not excessive.

Step 5 Populate the Materialized View

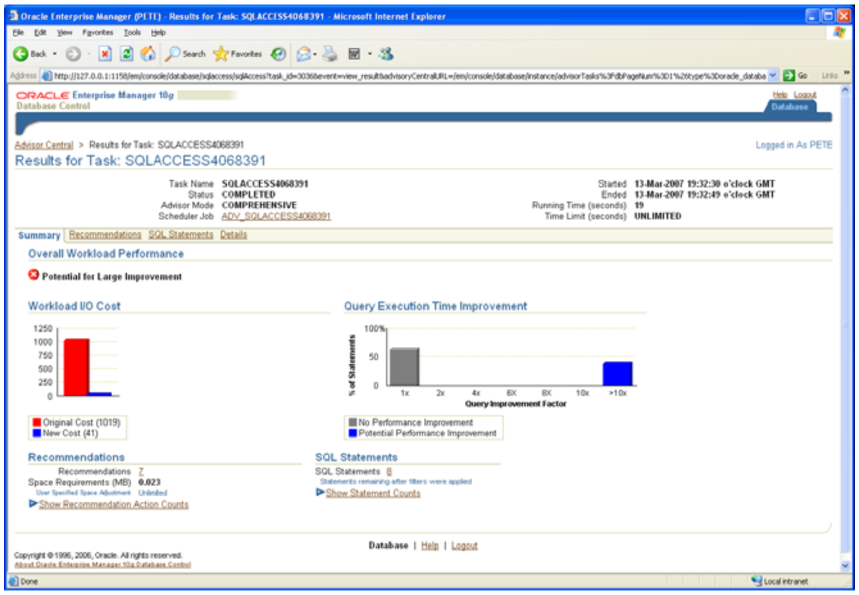
Materialized views provide mechanisms and control over when and how they are populated and refreshed. In our STATE_SALES_MV created in step 2, we have specified the BUILD IMMEDIATE clause which means that the materialized view is populated when the create statement is issued. Alternatively by using BUILD DEFERRED, populating the materialized view can be delayed until an appropriate point during the next warehouse refresh operation. Optimizer statistics will need to be gathered on the materialized view.

Generating Recommendations from the SQL Access Advisor

On a large and complex database, manually determining the optimal set of materialized views and their indexes that support the users queries can be a time consuming task. The SQL Access Advisor, which is part of the Tuning Pack, and has been available since Oracle Database 10g,is available to make this task considerably easier and it is an invaluable tool for this purpose. The SQL Access Advisor can be found in Advisor Central in Oracle Enterprise Manager or can be invoked from the command line using SQL*Plus by calling one of the procedures in the DBMS_ADVISOR package. Using as input a workload of SQL statements, the advisor takes you step-by-step through the process to recommend the materialized views, their indexes and the materialized view logs and how to implement them. The results of this process are provided as a set of recommendations, which can be implemented either by the SQL Access Advisor or manually.

Figure 4 shows the Summary screen resulting from running the SQL Access Advisor with a user workload containing our eight scenario queries. We can see that the advisor has made some recommendations that it predicts can result in significant performance improvements. From here it is possible to navigate to other screens to examine and modify the generated scripts, for example to change the materialized view names and tablespaces, and then a task can be scheduled for their deployment in the warehouse database.

Figure 4 The SQL Access Advisor Summary Screen


Oracle Enterprise Manager (PETE) - Results for Task: SQLACCESS4068391 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://127.0.0.1:1158/em/console/database/sqlaccess/sqlAccess?task_id=3036&event=view_result&advisoryCentralURL=/em/console/database/instance/advisorTasks%3FdbPageNum%3D1%26type%3Doracle_databa
Go Links
ORACLE Enterprise Manager 10g
Database Control
Help Logout
Database

Advisor Central > Results for Task: SQLACCESS4068391

Logged in As PETE

Results for Task: SQLACCESS4068391

Task Name	SQLACCESS4068391	Started	13-Mar-2007 19:32:30 o'clock GMT
Status	COMPLETED	Ended	13-Mar-2007 19:32:49 o'clock GMT
Advisor Mode	COMPREHENSIVE	Running Time (seconds)	19
Scheduler Job	ADV_SQLACCESS4068391	Time Limit (seconds)	UNLIMITED

Summary
Recommendations
SQL Statements
Details

Overall Workload Performance

Potential for Large Improvement

🟥 Original Cost (1019)
🟦 New Cost (41)

⬛ No Performance Improvement
🟦 Potential Performance Improvement

Recommendations

Recommendations	7
Space Requirements (MB)	0.023
User Specified Space Adjustment	Unlimited

Show Recommendation Action Counts

SQL Statements

SQL Statements 8

Statements remaining after filters were applied

Show Statement Counts

Copyright © 1996, 2006, Oracle. All rights reserved.

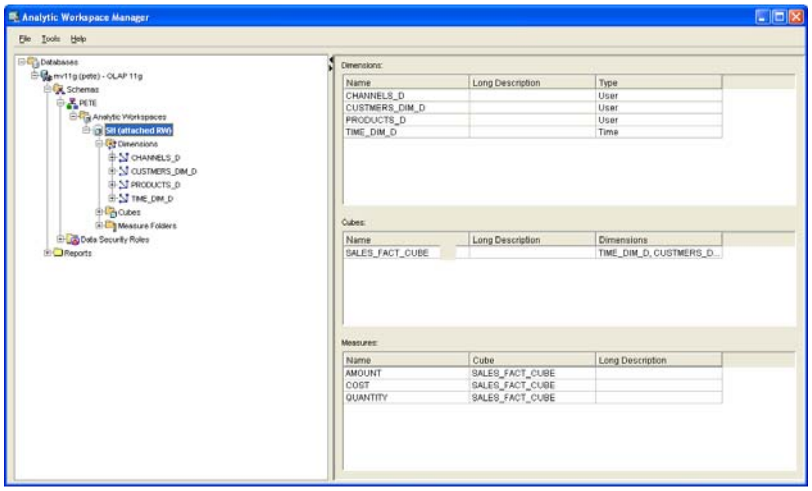
About Oracle Enterprise Manager 10g Database Control

Methodology for Defining the Analytic Workspace

The process for building the Analytic Workspace is quite different to the approach we have just seen for materialized views. There is no equivalent of the SQL Access Advisor to recommend the cubes required, however, there is the Analytic Workspace Manager (AWM) which is the principal GUI based administrative tool for building and managing analytic workspaces. AWM has been enhanced for Oracle Database 11g and is provided on the Client CD.

AWM, as illustrated in Figure 5, considerably simplifies the process of creating the Analytic Workspace and enables anyone from a DBA to a business analyst to design the OLAP data model with the dimensions and hierarchies, map their data sources to the dimensions and cubes and then populate them.

Figure 5: The Analytic Workspace Manager



```
Databases
  mv11g (pete) - OLAP 11g
    Schemas
      PETE
        Analytic Workspaces
          SH (attached RW)
            Dimensions
              CHANNELS_D
              CUSTMERS_DIM_D
              PRODUCTS_D
              TIME_DIM_D
            Cubes
            Measure Folders
      Data Security Roles
    Reports
```

Dimensions:

Name	Long Description	Type
CHANNELS_D		User
CUSTMERS_DIM_D		User
PRODUCTS_D		User
TIME_DIM_D		Time

Cubes:

Name	Long Description	Dimensions
SALES_FACT_CUBE		TIME_DIM_D, CUSTMERS_D...

Measures:

Name	Cube	Long Description
AMOUNT	SALES_FACT_CUBE	
COST	SALES_FACT_CUBE	
QUANTITY	SALES_FACT_CUBE	

Step 1 Analysis to Determine the Analytic Workspace Content

Before the analytic workspace can be defined, as per the approach for creating a materialized view, some analysis work must first be performed to identify what information it must contain. With respect to Oracle Database 11g, consideration should also be given to whether the new MV OLAP feature will be used. This will be discussed in a later section on accessing the analytic workspace using SQL.

We begin by looking at the type of queries our business users will be running and our data and determine which dimensions we need; which will be Time, Customers, Products and Channels and the physical tables where that data resides. Then we need to determine what cubes we require and the actual data we need to hold in the cubes. In this example, this is amount sold and quantity sold, which are physically stored in the SALES_FACT table, and the product unit cost data from the COSTS_FACT table.

Typically, only a single analytic workspace (AW) is needed to contain the required objects to answer the users business queries. For example, our one will comprise of four dimensions and two cubes and the table below lists what must be created and the data source for each object.

Target AW Object	AW Object Type	Relational Source Table
PRODUCT	DIMENSION	PRODUCTS
CHANNEL	DIMENSION	CHANNELS
CUSTOMER	DIMENSION	CUSTOMERS_DIM
TIME	DIMENSION	TIME_DIM
SALES	CUBE	SALES_FACT
COSTS	CUBE	COSTS_FACT

Step 2 Creating the Analytic Workspace

The first step is to create the analytic workspace in which the dimensions and cube will reside. Creating the analytic workspace using AWM simply requires naming the analytic workspace and identifying the tablespace where it will be located.

Step 3 Creating the Dimensions

An analytic workspace actually comprises a number of different types of objects, the principal ones we will be concentrating on are:

- Dimensions including their hierarchies, levels and attributes,
- Cubes

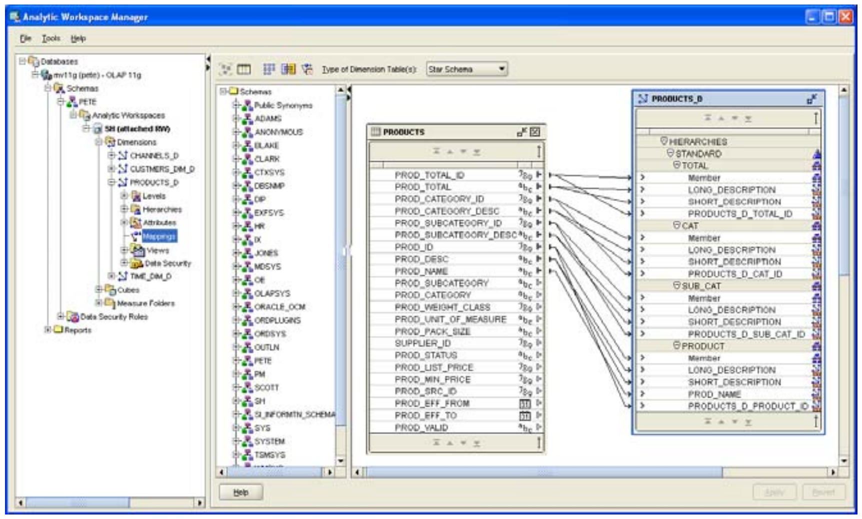
Dimensions are used to define the multi-dimensional data in a cube and must be created first in order to use them to create the cube. Dimensions in an analytic workspace can be either level based, the same as the relational dimension object, or can be value based (which is also known as a parent-child hierarchy).

The information required to create a dimension within an analytic workspace is very similar to that for the SQL dimension created earlier. A dimension is created by first creating the level information and then the hierarchies that group the levels in the correct order top to bottom. Finally, any additional attributes for the levels are created and assigned to the correct levels.

Step 4 Populating the Dimensions

In order to load the data into the dimension, mappings must be created to associate the columns in the tables holding the dimension source data and the attributes in the dimensions. Using AWM this is a simple task of dragging lines between the source table column and the target dimension attribute as shown in Figure 6 for the Product dimension. Here we can see columns in the PRODUCT table, such as PROD_DESC, linked to their attributes in the Product analytic workspace dimension, for example, the long and short descriptions at the lowest Product level of the hierarchy.

Figure 6: Creating Mappings in AWM



The last step in creating the dimension is to populate it by executing the mappings and is performed by selecting the *Maintain* option from the right click menu for the dimension. This executes a process that transfers the data from the source object into the dimension object within the analytic workspace and implicitly validates the dimensional data. Compare this to SQL Dimensions which are metadata about the structure of a dimension held in the relational tables: when new data is loaded into the tables used for the dimensions, it is not automatically validated to ensure that is conforms to the dimension structure. Consequently, SQL dimensions need the additional step of validating their structure by calling the VALIDATE_DIMENSION procedure in the DBMS_DIMENSION package.

Step 5 Creating the Cubes

Once the Customer, Channel and Month dimensions have also been created, the cubes may be created and populated with equal ease by right clicking on the Cube sub-tree in the navigation pane and selecting the "Create" option. The dimensions that define the shape of the cube are selected. In our example, the four dimensions Time, Products, Customers and Channel define the SALES cube and the dimensions Time, Product and Channel are used for the COSTS cube.

At this point when the cube is being created, there are a number of important options that may be specified which define the cubes storage structure and that can considerably improve the time to build the cube. The most important of these are:

- *Dimension Order and Sparsity.* Specify the dimension order to improve build and aggregation performance by listing the dense dimensions first. Dense dimensions are those that have fact data values recorded against a high proportion of their dimension values.
- *Compression.* Select the compression option if the cube data is very sparse, i.e. the ratio of real data values to 'null' data values is extremely small. In most cases, this option is turned on.
- *Aggregation.* Specify the aggregation method and operators to use. In Oracle Database 11g there are now two methods: the new Cost Based Aggregation method and the previous level based method.
- *Partitioning*: Specify how the cube is to be partitioned in order to improve load performance.

New in Oracle Database 11g is the Cost Based Aggregation method where the OLAP Engine determines the most expensive cells to aggregate and stores the aggregated value for those cells. A value of 0 to 100 can be set – the higher the value then the larger the data set that will be pre-computed and stored. A good rule of thumb is to use a value of 35. The level based aggregation method can still be used where the levels are specified in the hierarchies at which the aggregations are to be physically stored. This provides more control over the levels that will be pre-computed and good strategy is to specify the aggregates to be calculated at alternate hierarchy levels.

Partitioning cubes should not be confused with the Database Partitioning option: for cubes it defines how the cube is stored as separate components within the analytic workspace. To partition your cube, choose a level within one hierarchy of one dimension: a different partition is then used for each data value at that level. For example, our SALES cube uses a time dimension that has month as its lowest level and so specifying the cube partition strategy at either the Quarter or the Year level would be a good strategy. Partitioning cubes benefits the cube load and aggregation performance. It can also improve query performance because a query may be isolated to a single partition.

Step 6 Creating the Measures and the Cube Mappings

Now we have defined the dimensions for our cube and its storage structure, we must specify the data it is to contain (the *measures*) and the mappings to the source data.

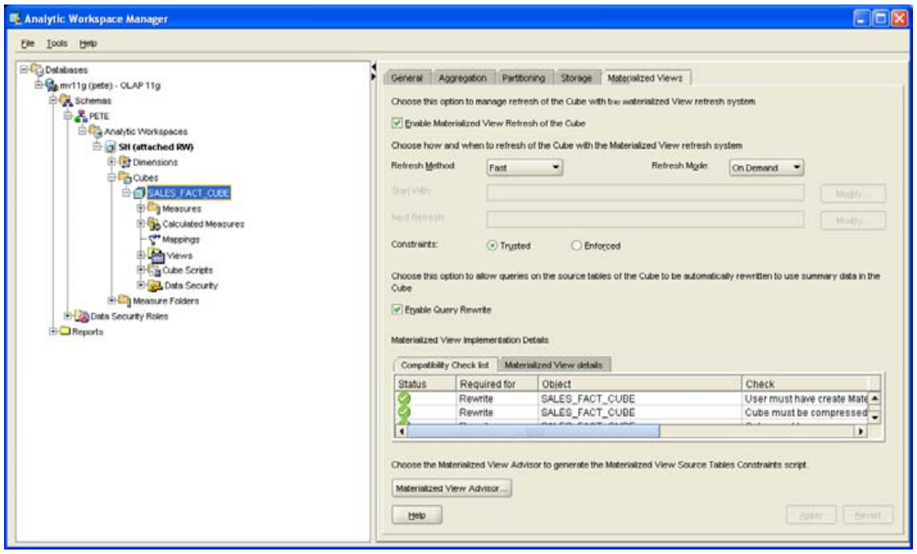
When we define the measures for a cube, we define how the fact data is held in each cell of the cube. The cube mappings are defined just as we did for dimensions by dragging lines between the source relational table and the cube definition.

Step 7 Creating the Cube Materialized Views in Oracle Database 11g

Oracle Database 11g introduces the capability to create cube materialized views over the analytic workspace cubes and dimensions and enable query rewrite to transparently rewrite to them for user queries that are written in SQL against the base source tables.

The new cube materialized views are controlled via AWM using a new Materialized View tab for the cube as shown in Figure 8.

Figure 8: The Materialized View Tab in AWM 11g



Choose this option to manage refresh of the Cube with the materialized View refresh system

[✓] Enable Materialized View Refresh of the Cube

Choose how and when to refresh of the Cube with the Materialized View refresh system

Refresh Method: Fast   Refresh Mode: On Demand

Constraints: [✓ Trusted] Enforced

Choose this option to allow queries on the source tables of the Cube to be automatically rewritten to use summary data in the Cube

[✓] Enable Query Rewrite

Materialized View Implementation Details

Status	Required for	Object	Check
✓	Rewrite	SALES_FACT_CUBE	User must have create Mate
✓	Rewrite	SALES_FACT_CUBE	Cube must be compressed

Choose the Materialized View Advisor to generate the Materialized View Source Tables Constraints script.

By selecting the "*Enable Materialized View Refresh of the Cube*" check box, the new cube materialized views are created for the cube and its dimensions when the cube is maintained. The cube materialized view naming convention uses the cube name with a "CB$" prefix. The new cube materialized views are also able to benefit from the refresh capabilities for loading data which is also controlled from this tab and is explored in more detail in a later section. Don't forget to select the option "*Enable Query Rewrite*" otherwise the cube materialized views will not be made available for query rewrite to use.

Step 8 Populating the Cube

The final step in defining the analytic workspace on Oracle Database 11g is to maintain and populate the cube, after which it is available for querying. During this stage, data is loaded from the cube data source, for example the relational table SALES_FACT in our case for the SALES cube, and joined to its dimension values and stored in the cube. It is also at this point that the aggregations that were specified for the cube are performed and stored.

The cube organized materialized views are also created for our cubes and dimensions and are refreshed.

Once the data has been loaded then the analytic workspace is ready to be queried.

Step 9 Refreshing The Cube Materialized Views

If we had maintained the cube without checking the tick boxes on the Materialized View tab, then our new cube materialized views would be built but marked as *unusable*. In this case, before they can be used we need to manually ensure that their data content is up to date and for them to be marked as *fresh*. To do this we call the REFRESH procedure in the DBMS_MVIEW package from SQL*Plus as shown below to perform a complete refresh on the cube materialized view for the SALES_FACT_CUBE cube.

```
EXEC DBMS_MVIEW.REFRESH(list=>’CB$SALES_FACT_CUBE’,method=’C’);
```

Alternatively, we can also use the BUILD procedure in the new DBMS_CUBE package, which will automatically refresh any stale or unusable materialized views over analytic workspace dimensions prior to refreshing the cube materialize view.

```
EXEC DBMS_CUBE.BUILD(’SALES_FACT_CUBE’);
```

The final step before the cube materialized views are usable by query rewrite is to collect statistics on them using the new DBMS_AW_STATS package as follows:

```
EXEC DBMS_AW_STATS.ANALYZE(‘SALES_FACT_CUBE’);
```

TOOLS FOR QUERYING

Now we have seen how to build the materialized views and the analytic workspace and its cube, we will look at how they are queried.

A significant advantage is provided in Oracle Database 11g with the ability for materialized views and query rewrite to be used to extract data from the analytical workspace because the user and the reporting tools do not need to even be aware of the existence of the analytic workspace or of its structure, its cubes or dimensions in order to take advantage of all of its capabilities. The tools construct SQL against the relational base tables and query rewrite will simply translate it to use the materialized views wherever possible. The table below summarizes this advantage where query rewrite is used to deliver analytic workspace performance and functionality to tools using regular SQL against the relational base tables.

Of course, in both Oracle Database 10g and 11g it is possible to define relational views over the cubes and use slightly differently structured SQL queries to access these views and hence the AW. This was a manual process via AWM for Oracle Database 10g but is fully automated in Oracle Database 11g. We will look at both methods of using SQL later in the whitepaper.

If our data warehouse only uses the analytic workspace and not relational tables for reporting, then using the correct tool to access the analytic workspace is paramount for getting the best value from your data and from your investment. Oracle provides a number of powerful graphical tools and interfaces specifically to enable queries against the analytic workspace to be easily constructed and executed:

REFRESHING THE DATA

Another important consideration when trying to decide upon whether to use materialized views or analytic workspaces is to review the mechanisms for maintaining these objects to ensure that they always contain the latest information. In this section we consider the mechanisms that are available for their refresh, and the speed at which the refresh is performed.

Refreshing Materialized Views

One of the advantages of using materialized views is that there are a number of refresh mechanisms available within the Oracle Database for refreshing them as the data changes in the base tables:

- Complete by re-executing the defining query
- Fast by applying incremental changes to the data
- Partition Change Tracking (PCT) refresh

When a complete refresh is performed the materialized view is fully rebuilt by re-executing its defining query: depending upon the size of the materialized view this could potentially be a costly operation. By tracking the changes to the base data using a materialized view log, a fast refresh is able to apply only these changes to the materialized view. Alternatively, a fast refresh can be performed by transparently detecting when changes to the data in partitions of the base tables occurs and then only the contents of those partitions need be re-computed for refreshing the materialized view. Similarly, during direct path load operations, such as SQL*Loader direct path loads, the database automatically tracks the new data that is loaded at the block level. These two techniques do not require a materialized view log for fast refreshes. It should be noted, however, that not all materialized views are fast refreshable and this can be identified by using the packaged procedure DBMS_MVIEW.EXPLAIN_MVIEW.

Materialized views are refreshed either on demand or on-commit. An on-demand, fast refresh is illustrated below using the DBMS_MVIEW package.

```
DBMS_MVIEW.REFRESH( list => ’state_sales_mv,
channels_count_mv’, method => ’F’);
```

Refreshing Analytic Workspace Cubes

With the introduction of cube based materialized views in Oracle Database 11g, this means that the refresh mechanisms that materialized views have enjoyed, are now available to Analytic Workspaces. The most significant of these are the abilities to use materialized view logs and partition change tracking to perform a fast refreshes with considerable improvement in performance. A new refresh mechanism, only available for cube materialized views, is introduced in Oracle Database 11g called FAST_SOLVE. This method incrementally re-aggregates the cube by detecting the data changes without using materialized view logs. Previously in Oracle Database 10g, the ability to fast refresh just the changed data was not available and identifying the changed data in the source table had to be built into the mappings. For example, by using a view based on the source table as the data source rather than the table itself the view definition can used a WHERE clause to identify new or updated rows.

Using a partitioned cube will significantly improve the data load and refresh times compared to an unpartitioned cube because it enables the database to automatically use parallel load processes.

OTHER ISSUES TO CONSIDER

With the convergence in functionality between materialized views and analytic workspaces for supporting query performance, what other factors should be considered to help you determine which is the correct approach for your organization? The table below illustrates some important considerations that may affect your decision.

Issue	**Materialized Views**	**Analytic Workspaces**
Requires Enterprise Edition	Yes	Yes
Requires OLAP Option	N/A	Yes
Report Generated when Data is missing from AW or MV	Yes	No (10g) Yes (11g)
Supporting Join Performance	Yes	Yes
Aggregate Rollup	Yes	Yes
Data Completeness and Sparsity	No	Yes
Adhoc Analytic Functions	No	Yes
Wizards and Tools for Recommendations	Yes	No (10g)

		Yes (11g)
Fast Data Refresh	Yes	No (10g) Yes (11g)
Support for different hierarchy types	Yes (partially)	Yes
Uses SQL	Yes	Yes
Forecast Models	No	Yes
Query Overheads	None	Yes (10g) Improved (11g)

Attribute Presence For Querying

What happens if our query refers to an attribute that is not present in the materialized view or analytic workspace? The relational query returns results because a query rewrite mechanism known as *join back* can automatically join the existing materialized view back to the base table to access the missing data attribute. Even if no viable materialized view were available then the optimizer allows the query to return results by operating against the base tables where all the attributes and fact data are present.

In Oracle Database 11g, the use of query rewrite to access the analytic workspace means that the queries can now also take advantage of the join back rewrite mechanism so that even if attributes are missing from the analytic workspace, they can still be derived from the base tables. Previously in Oracle Database 10g, query rewrite was not available to access the analytic workspace and if the attribute had not been loaded into the analytic workspace dimension then it was not even possible to construct the query and thus no results could be returned. To resolve this, the dimension would have to have been rebuilt to add the missing attribute information.

Data Completeness and Sparsity

In relational tables, records are only present for data that exists, whereas in analytic workspace cubes, an empty value is used where no data exists. The cube stores the real data values and whereas the empty values are easily addressable and queryable, they are not actually stored in the cube. For example, you can easily refer to last month's data value in a query even if that month has no data, but the empty value is not stored. This highlights the very important feature of analytic workspace cubes in that they operate as if they are fully populated with both actual data and zero data for the full combination of their dimensions' values. This can have a very important benefit because it makes defining calculations easier as it can be assumed that all data points are present in the cube. For example, the following formula can be defined even if there is no actual value, or physical storage used, for Tents in Feb2002:

```
nvl(sales('Feb2002','Tents'),0)
             - nvl(sales('Jan2002', 'Tents'), 0)
```

Analytic workspaces have significant improvements to data loading, aggregation performance and query performance resulting from efficient handling of the sparsity due to industry leading compression technology in Oracle Database 11g. If your reporting requirements need to process null value rows, serious consideration should be given to using analytic workspaces because they can handle this information by default rather than having to write a more complex SQL statement to generate the same result.

Supporting Join Performance

By pre-computing the join results in advance, materialized views are excellent at improving the performance of relational joins between any relational tables in the database. In contrast, a single analytic workspace cube only pre-computes and stores the join between that cubes measure data and its dimensions.

However, even though analytic workspaces do not pre-join cubes together in advance, due to their internal storage structure, they are very efficient at performing this type of join operation on the fly at query time. In addition, the ability to define this join is significantly easier and less complex in the analytic workspace than when defining it relationally. We look at this operation in the analytic workspace in more detail in the section Writing SQL Queries Against The Analytic Workspace Cube.

Aggregate Rollup

Maintaining aggregates, such as the sum and average, requires time and resources and if possible it is advantageous to minimize the number that must be maintained. If aggregates are available at a monthly level then it is useful to be able to use these and roll them up at query time to the quarterly or yearly level rather than explicitly maintaining the quarterly or yearly totals themselves at load time.

Aggregate rollup in both analytic workspaces and as used by query rewrite against materialized views, are equally powerful and efficient in using the metadata in their dimension definitions to perform this rollup operation. Both approaches can utilize pre-computed lower level aggregates to answer user queries requiring aggregates at a higher level in the dimension hierarchy.

Adhoc Analytic Functions

The multi dimensional calculation engine in Oracle OLAP that operates on the cube data provides a very rich set of analytical functionality. In addition, because of the specialized storage structures of the cubes, adding a new derived data value, known as a *calculated measure*, does not require the cube to be rebuilt. To achieve the same performance relationally, it might be necessary to create a new materialized view which would take additional time to build thus delaying the time to generate the report and increasing the time required to maintain the materialized views. Contrast that with the analytic workspace, because the analytic function is already present there is practically no reduction in performance compared to accessing the cube base data values themselves.

Wizards and Tools for Recommendations

We saw in the introductory section that both materialized views and analytic workspaces require a fair amount of analysis work and setup before they can be used. Therefore any tools that are available to ease this process, especially for new users to the functionality, are always welcome. Materialized views are well supported with the SQL Access Advisor, as illustrated in Figure 4, which accepts a set of SQL statements, recommends the materialized views and provides a script to implement the recommendation. In addition, the following PL/SQL procedures are provided to help troubleshoot any problems with your materialized views and query rewrite:

- EXPLAIN_MVIEW procedure in the DBMS_MVIEW package to report on the capabilities of a materialized view such as whether or not fast refresh is possible and what types of query rewrite are supported.
- TUNE_MVIEW procedure in the DBMS_ADVISOR package to help optimize the materialized view to make it fast refreshable if possible and maximize the possibility for query rewrite to occur
- EXPLAIN_REWRITE procedure in the DBMS_MVIEW package which reports why query rewrite did, or didn't, use a materialized view.

Analytic workspaces do not have an advisor to recommend the correct dimensions and cubes required to support the user queries. However, there are a number of advisors available to assist the development process when constructing the analytic workspace to ensure that the cubes are correctly built.

Sparsity Advisor

The Sparsity Advisor in AWM 10.2.0.3 will analyze the dimensions for the cube to determine which are sparse and which are dense to be able to recommend the order to be used in the cube construction, compression and whether or not cube partitioning is required. Correctly specifying this order can have a significant impact on the time that it takes to load and query the cube.

Advisors in AWM 11g

The Sparisty Advisor has been replaced with three new in Oracle Database 11g:

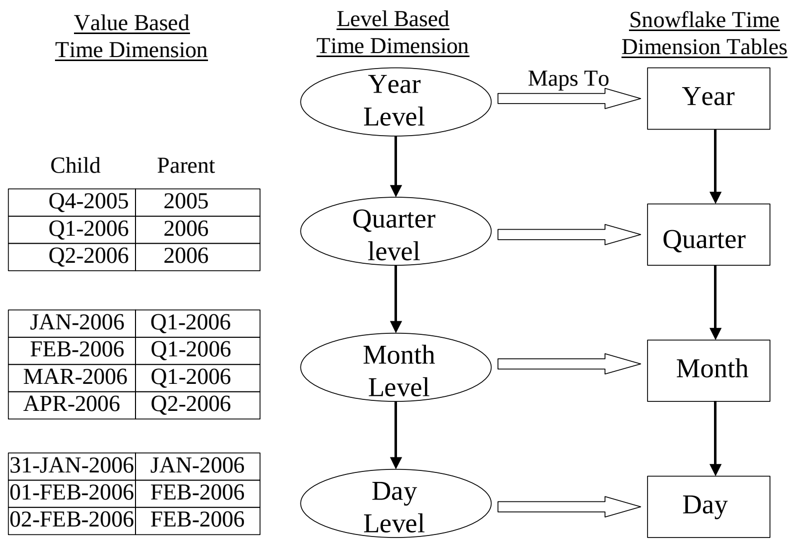
- The *Cube Partitioning Advisor* makes recommendations for partitioning the cube based on the partitioning of its source table and the sparsity of the data in its dimensions.

- The *Cube Storage Advisor* analyzes an existing cube and its dimensions and makes recommendations about how the cube could be more optimally rebuilt. e.g. the Advisor will examine which dimensions in the cube are dense or sparse in order to determine their correct storage order in the cube and whether or not compression should be used.
- The *Relational Schema Advisor* generates a SQL script for creating all of the necessary database objects and constraints required for query rewrite to operate against cube materialized views.

Enhanced Reporting Via Dimensions

Supporting different types of hierarchies in the dimensional data, and allowing more flexibility in how hierarchies are defined, can enable a wider variety of business scenarios to be modeled. The SQL Dimension, which is used by materialized views, only supports level based type dimensions, whereas analytic workspaces can use both value and level based, as illustrated in Figure 8.

Figure 8 Dimension and Hierarchy Types



Within the analytic workspace dimensions, *ragged* hierarchies can also be implemented, which enables a different number of levels to be used when different paths in the hierarchy are traversed from the root level to the bottom of the hierarchy. Use of ragged hierarchies enables more flexibility in the data modeling and can increase the number of business scenarios that the dimension can model.

Forecast Models

Frequently, analyzing the historical data is required so that businesses can predict future trends. This is an area that analytic workspaces do very well because it is not easily accomplished relationally.

In Oracle Database 10g, forecasting models are implemented using *Calculation Plans* defined using a wizard. The wizard takes the user step-by-step through the process of defining the forecasting model to be used to and how the model operates for each dimension of the cube. A number of different models are provided, or, an automatic method can be selected where the historical data is analyzed for you to determine the best fit model to use. Once the calculation plan has been defined it must be executed to generate the forecast data. Defining the forecast functionality will be available in a future version of AWM 11g but can still be easily performed in the current version by using the analytic workspace programming language OLAP DML.

In comparison, materialized views pre-compute joins and aggregations using actual historical data present in the tables and do not have the capability to forecast trends. Relationally this can be done using the SQL MODEL Clause, which was introduced in Oracle Database 10g, however there are no wizards available to define and manage the forecasts in the same way that AWM does for analytic workspaces.

Query Overheads

The overhead for attaching the analytic workspace has been significantly reduced and is negligible in Oracle Database 11g, where as in Oracle Database 10g, when the first relational query against an analytic workspace in a database session is executed, an overhead of approximately 20 seconds is incurred as the analytic workspace is attached to the session for the first time. There are also overheads as part of the first aggregation operations as the analytic workspace caches the aggregation results. Subsequent queries against the analytic workspace will not suffer these overheads.

There is no equivalent overhead when performing a relational query against tables and using query rewrite to use materialized views.

RELATIONAL ACCESS TO THE ANALYTIC WORKSPACE

Accessing the analytic workspace and cubes from SQL in Oracle Database 11g can be performed in two ways:

- by using an interface known as CUBE_TABLE
- via query rewrite.

Using CUBE_TABLE for SQL Access to the Analytic Workspace

Using SQL to access the cubes and dimensions is a significant feature of Oracle OLAP because it enables reporting tools that only generate SQL to utilize all of the powerful features of the analytic workspace. In Oracle Database 11g this is achieved by the use of the CUBE_TABLE function that extracts multidimensional data from a cube in an analytic workspace and presents it to the relational SQL engine in the form of a two dimensional table i.e. as a set of rows and columns. It provides a mapping between the cube in the analytic workspace and the rows and columns that the SQL sees.

CUBE_TABLE is parameterized by the name of an analytic workspace dimension or cube. For a dimension, CUBE_TABLE returns the dimension data as a table of records for all of the dimension's members with columns for each of the hierarchy levels. The dimension parameter can also name a specific hierarchy that results in additional columns being available for parental data for levels in the hierarchy to provide additional flexibility when querying and navigating the hierarchy. The hierarchy phrase also limits the dimension rows to those that are in the hierarchy. For example, for the Fiscal Time hierarchy you won't see Calendar quarters in the view if the Fiscal hierarchy is specified. However if you don't specify the hierarchy, then you get all dimension members across all hierarchies. AWM automatically creates normal views using the CUBE_TABLE function for each analytic workspace dimension and cube and these cubes can be queried by SQL in order to access the analytic workspace objects. There are two types of view automatically created:

1. Dimension Views
    - The general dimension view is named using the dimension name and the suffix '_VIEW'.
    - A dimension hierarchy view for each hierarchy in the dimension and has the suffix '<hierarchy name>_VIEW'. This view contains additional parent child columns to assist navigation up and down the hierarchy.
2. Cube Views, which contain the measures and the foreign keys for the cubes dimensions for joining to the dimension views and are named using the cube name and the suffix '_VIEW'

In Oracle Database 11g, a CUBE_TABLE based normal view is created automatically by AWM for each cube and each dimension. They are queried by joining them together very much as if they are a relational star schema, i.e. as you would query a fact table and its dimension tables. This means normal query SQL, can be used to access these views and the analytic workspace. The example query below uses these views to access the analytic workspace to answer the query:

```
SELECT t.long_description time, f.amount amount
FROM time_dim_view t,  products_dim_view p,  customers_dim_view cu,
     channels_dim_view ch, sales_fact_cube_view f
WHERE t.long_description IN ('2005', '2006')
  AND t.level_name  = 'YEAR'  AND p.level_name  = 'TOTAL'
  AND cu.level_name = 'TOTAL' AND ch.level_name = 'TOTAL'
  AND t.dim_key = f.time_dim         AND p.dim_key = f.products_dim
  AND cu.dim_key = f.customers_dim_d AND ch.dim_key = f.channels_dim;
```

When writing SQL against the CUBE_TABLE views, it is important to include WHERE clauses to fully qualify the data required for every dimension in order to restrict the slices of the cube that is being queried and therefore the volume of data that is accessed and returned back to the SQL statement. Because the cube operates as if it is fully populated i.e. with all of the null data values due to its sparsity this could result in a very large volume of data values being returned from the cube via the cube view unless restricted. A benefit is that the cube contains *all* levels of aggregation and it is very easy to write queries that return data from different levels of aggregation. Also note that no GROUP BY clause is needed and no aggregation operators (e.g. SUM) are required. This is because the cube presents the results as already aggregated and therefore:

1. You don't want to aggregate again in the SQL
2. You want to leverage the business rules in the aggregations as implemented within the analytic workspace, for example, account balance or headcount aggregate differently over time than they do over product
3. The calculated measures already work correctly in the cube, for example, Year To Date (YTD) and YTP Prior Year, and don't require any further computation.

Query Rewrite To Cube Organized Materialized Views

Querying either the relational tables or the analytic workspace, via the CUBE_TABLE views, requires our report developers and power users to decide which type of data they need to access. So wouldn't it be easier if they could always query just one type of data and let the database decide how it wants to process the query: relationally or multi-dimensionally?

This is exactly what we are now able to do in Oracle Database 11g with the ability for query rewrite to use the new cube organized materialized views: we use SQL to query the relational base tables and the optimizer transparently translates the SQL to access either the table materialized views or the cube materialized views (and hence the analytic workspace cubes and dimensions) depending upon which provides the better performance. This allows all of the benefits of the analytic workspace to be easily available to any product using regular SQL.

Analytic workspaces are also very good at joining cubes together. A good multi-dimensional design will lead to data of different dimensionality being in different cubes. When it is necessary to join two cubes together and make the resulting cube available to SQL then it is best to perform the operation in the analytic workspace rather than to use SQL to perform the join. We can do this join very easily in the analytic workspace using AWM to define a calculated measure.

For example, consider joining the 3-dimensional COST data with dimensions (TIME, PRODUCT, CHANNEL) to the 4-dimensional SALES data (TIME, PRODUCT, CHANNEL, CUSTOMERS). To define the join relationally between SALES_FACT and COSTS_FACT relational tables requires at least joining the three common dimension columns and performing an aggregation, whilst taking care not to "double count" because of the missing Customer dimension. To write the SQL for this join requires detailed knowledge of the structure of the fact and dimension tables and their primary keys.

In contrast in the analytic workspace, the join can be implemented using AWM by the simple step of defining a new calculated measure in the SALES cube that references the required measure in the COSTS cube. The new calculated measure column in the SALES cube is then exposed to SQL via the CUBE_TABLE interface that we have already discussed. No WHERE clauses to perform the join need to be written in our SQL query because we are only accessing the SALES cube which contains the calculated measure as a reference to the measure in the COSTS cube. Even though COSTS is a 3-dimensional cube and SALES is a 4-dimensional cube, the analytic workspace can intelligently and automatically account for the missing customer dimension to perform this join operation on cubes of different dimensionality. Because this is a calculated measure, no other build time is required other than defining the calculated measure itself: AWM automatically rebuilds its views to include the new column.

CONCLUSION

Either approach is a "best of breed" in its own right or both can be used in a hybrid solution to provide the best of both worlds.

Database size is rapidly increasing and more areas of the business are demanding the value of being able to quickly analyze the data, which is resulting in a heavier demand upon the database. Therefore, providing query results immediately is an essential business requirement.

Both approaches, using materialized views and analytic workspaces, provide powerful features to achieve this goal and with the new cube materialized views and their query rewrite available in Oracle Database 11g, it is becoming ever easier to use all of these powerful features.

Summary Management enables materialized views to be easily deployed and maintained, they are quick to load and refresh and because of Query Rewrite their use can be completely transparent to the end user. However, a complex reporting environment can lead to a large number of materialized views being required which can result in more administration being necessary. To offset this, one of the advantages of using materialized views is the availability of advisors to help build and refine them, which can be invaluable if expertise is limited within your organization. Materialized views also benefit from the availability of fast refresh mechanisms.

There is a significant difference between Analytic Workspaces in Oracle Database 10g and 11g. Previously, relational access was limited and was more difficult to implement but now in Oracle Database 11g, relational access is via automatically created database views and materialized views. Analytic Workspaces can now use the fast refresh capability of materialized views and query rewrite and provide fast access to sophisticated analytical capabilities. They can also easily be accessed from relational SQL to make their fast analytical capabilities available to a wider range of tools and users.

This white paper has compared the two approaches and we can see that prior to Oracle Database 11g, materialized views were considerably easier to manage and advisors were available to recommend the best set of materialized views to implement.

With the introduction of Oracle Database 11g, analytic workspaces now offer many of the manageability features of materialized views, although they still lack the sophisticated advisors available for Summary Management.

In conclusion, both approaches offer powerful features for improving your database query performance and either approach can suit your reporting requirements, but one may be preferable depending on the type of analysis to perform.

Comparing Materialized Views and Analytic Workspaces in Oracle Database 11g
March 2008
Authors: Pete Smith and Dr. Lilian Hobbs

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