

Using Oracle Business Intelligence Enterprise Edition with the OLAP Option to Oracle Database 11g

*An Oracle White Paper
July 2008*

NOTE:

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Introduction	5
Implementing OBI EE on Cube-Organized Materialized Views	6
OBI EE Implementation Tasks.....	7
Designing the Cube-Organized Materialized View for OBI EE.....	7
Preparing for Query Rewrite.....	8
Understanding if Queries Rewrite to the Cube-Materialized Cube.....	8
Implementing OBI EE on OLAP CUBE VIEWS	9
About OLAP Cube Views.....	10
What's Different about Configuring OBI EE to Query Cube Views?	11
Sample Implementation	13
Physical Layer	13
Business Model Layer	14
Presentation Layer	29
Conclusion.....	31

Using Oracle OBI EE with the Oracle OLAP Option

INTRODUCTION

The Oracle OLAP Option is designed to enhance SQL-based business intelligence applications with improved query performance and enhanced analytic content. The OLAP Option cube is designed to efficiently manage summary data and to provide fast and consistent query performance anywhere within the data model. This makes the OLAP Option an excellent choice for improving performance of queries requiring summary data and to support ad-hoc query tools.

OLAP Option cubes can also be easily enhanced with rich analytic content. For example, it is easy to add calculations such as time series, indexes, market shares, rankings and much more to the cube. To the query application, calculations such as these are simply additional fact columns. Applications do not need to understand how the calculations are defined in order to query them.

Oracle Business Intelligence Enterprise Edition (OBI EE) is a product suite based on the OBI EE Server. The OBI EE Server can map a logical business model to many different physical data sources and present the logical model for query to variety of client applications including Interactive Dashboards, Answers and Oracle Business Intelligence Plug-in for Microsoft Office.

Oracle Business Intelligence Enterprise Edition is most commonly used with the Oracle Database using SQL as the query language. Although the OLAP cube is a multidimensional data type, it is represented in the Oracle database as a collection of relational views and is easily queried by SQL. This creates the perfect situation – the OLAP cube can be used to improve the query performance and analytic content of OBI EE applications.

The following report illustrates how the OLAP Option can be used to enhance an OBI EE application.

Calendar Year	Department	Sales Rank Within Parent	Sales	Sales YTD	Sales YTD Prior Year	Sales YTD % Change Year Ago
CY2005	Computers	1	88,709,396	88,709,396	79,158,141	12%
	Portable Music and Video	2	16,064,530	16,064,530	14,801,493	9%
	Cameras and Camcorders	3	8,865,197	8,865,197	7,308,918	21%
CY2006	Computers	1	101,669,050	101,669,050	88,709,396	15%
	Portable Music and Video	2	17,328,207	17,328,207	16,064,530	8%
	Cameras and Camcorders	3	10,657,143	10,657,143	8,865,197	20%
CY2007	Computers	1	115,957,793	115,957,793	101,669,050	14%
	Portable Music and Video	2	19,101,368	19,101,368	17,328,207	10%
	Cameras and Camcorders	3	12,387,923	12,387,923	10,657,143	16%

Sample OBI EE report enhanced with measure columns that are selected from the OLAP cube.

First, note that the report contains summary data (calendar year and department level data) and allows the user to explore by drilling down on either the time or product dimensions. The cube will provide excellent query performance for this query and any drill down queries. The user is free to explore and analyze any region of the data model with the expectation that query performance will be uniformly good.

Next note that the report contains interesting columns such as Sales Rank Within Parent, Sales YTD, Sales YTD Prior Year and Sales YTD % Change Year Ago. Each of these measures are defined in the cube and revealed to OBI EE as a column in the fact view. Because the calculations are defined in the cube they work anywhere within the data model.

There are two approaches that may be taken for using OBI EE and the OLAP cube together. If the sole objective is to improve the query performance of OBI EE, the summary data in the cube can be accessed transparently using automatic query rewrite to a *cube-organized materialized view*. In this case, summary data is managed in the OLAP cube, OBI EE queries the base relational tables and the Oracle Database automatically rewrites queries that require summary data into the cube. The entire process is transparent to OBI EE.

If the objective is to both improve query performance and enhance the analytic content of an OBI EE application, OBI EE can query the cube directly. In this case, summary data and calculations are managed in the cube and OBI EE queries relational views of the cube. Since the cube is represented as relational views and is queried using SQL, this is easy for OBI EE to do.

It is worthwhile to note that one cube can be used as a cube-organized materialized view and can be queried directly with SQL. This allows a single cube to serve different purposes depending on the requirements of the application.

The remainder of this paper describes how to implement OBI EE on both cube-organized materialized views and on relational views of the cube.

IMPLEMENTING OBI EE ON CUBE-ORGANIZED MATERIALIZED VIEWS

The magic of the cube-organized materialized view is that it is transparent to OBI EE. OBI EE continues to query relational tables and the Oracle Database automatically rewrites OBI EE queries requesting summary data from the relational tables to the cube.

Cube-organized materialized views, new to Oracle 11g, enhance the materialized view system (which were first introduced in Oracle 8i) with better query performance and fast incremental update. Summary level data is managed in a cube. A materialized view provides the means for query rewrite and materialized

view based refresh of the cube. Cube-organized materialized views can co-exist with table-based materialized views.

If you are familiar with materialized views, think of a cube-organized materialized view as a materialized view on pre-built cube (analogous to materialized view on pre-built table). The materialized view is a metadata-only object; data is not replicated into the cube-organized materialized view.

Data can be loaded into the cube at the same levels as the source tables or it may be loaded into the cube purely as a summary over the tables. Therefore, the cube does not need to replicate the data of the source tables. For example, the tables that OBI EE queries might contain data at the day level and the cube might begin at the month level. Queries that are at the month level and higher will be rewritten to the cube.

OBI EE Implementation Tasks

In this implementation there is literally nothing extra or different to be done in OBI EE. The physical layer of the OBI EE repository remains mapped to relational tables and the Oracle Database automatically rewrites queries to the cube as appropriate. OBI EE applications just run faster. It's that simple.

Designing the Cube-Organized Materialized View for OBI EE

The Oracle Database supports *general* query write to the OLAP Cube. This means that the Database understands the content of the cube and can rewrite different styles of GROUP BY into the cube, including the style used by OBI EE. As such, there are no unique considerations for designing cubes that will be used as a cube-organized materialized view with OBI EE.

It is important that the design of the cube be consistent with the OBI physical and business models. General guidelines follow:

- Cubes are based on a dimensional model and are suited to providing summary management services to relational schema that are also dimensional. Therefore, OBI EE applications that are based on star and snowflake schema are good candidates for benefiting from cube-organized materialized views.
- An OLAP dimension is derived from each dimension table of the star or groups of dimension tables in a snowflake, just as OBI EE dimensions are in the Business Layer.
- The hierarchies in the OLAP dimension should be consistent with hierarchies in OBI EE.
- Each OLAP cube selects data from a single fact table (along with the dimension tables associated with that fact table).
- Aggregation functions of measures in the cube must be the same as aggregation functions used in the OBI EE repository.

A cube is an object in the Oracle database that stores data in a multidimensional format for fast query and incremental update. It is highly optimized for hierarchical data and aggregation. Generally speaking, a cube can be thought of as a fact table with multidimensional data storage.

What is most important is to make sure that dimension model of the cube (dimensions, hierarchies, levels and cubes) is designed to complement the star or snowflake schema that OBI EE is mapped to. Otherwise the process of designing the cube for OBI EE and query rewrite is the same as for any other application.

When using cube-organized materialized views you may choose to include all dimensions of the relational fact table in the cube or only a subset of dimensions. When all dimensions of the fact table are included in the cube, any summary level query written against the fact table can be rewritten into the cube. Depending on query patterns, including all dimensions of the fact table might, however, cause the cube to be larger than necessary. In some cases, it might make sense to create one or more cubes that each contains only a subset of the dimensions of a fact table.

Consider the example where a fact table is dimensioned by product, channel, geography, age, income and gender. A single cube could be designed with all dimensions. This might make sense if all dimensions are commonly used in a single query. If query patterns tend to be clustered in a subset of dimensions – for example time, product and geography or time, product, age, income and gender – it might be more efficient to create two smaller cubes such as a geography cube and a demographics cube since these might be quicker to update.

Preparing for Query Rewrite

Just like any other implementation of materialized views, there are a few things that must be done to prepare the relational schema for query rewrite. Either before or after the cube and cube-organized materialized view is created, be sure the relational schema includes:

- Primary key constraints on each dimension table and the fact table.
- NOT NULL constraints on columns in the dimension table that represent levels.
- Foreign key constraints on the fact table joining to the dimension tables.
- SQL dimension objects with hierarchies, levels and attributes that match the tables and the cube dimensions.

The Relational Schema Advisor, which is accessible from Analytic Workspace Manager's Cube/Materialized View tab, can evaluate the schema and provide a SQL script with recommendations for constraints and SQL dimension objects. It will also make recommendations for materialized view logs that are required for fast refresh of materialized views.

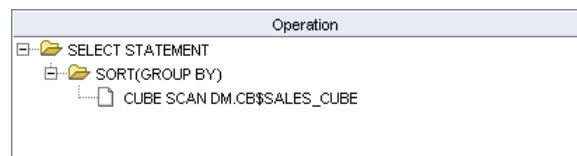
Understanding if Queries Rewrite to the Cube-Materialized Cube

A very high percentage of OBI EE queries are rewritten to cube-organized materialized views when the design of the cube aligns correctly to the base schema and the OBI EE model. This can be observed by capturing OBI EE queries in the query log and examining the SQL execution plan for the query.

The following is a typical query issued by OBI EE against the base tables of a star schema.

```
SELECT t19965.calendar_year_id AS c1,
       t19932.all_products_id AS c2,
       t19880.all_regions_id AS c3,
       SUM(t19957.sales) AS c4,
       t19965.calendar_year_end_date AS c5
FROM times t19965,
     products t19932,
     customers t19880,
     sales_fact t19957
WHERE(t19880.customer_id = t19957.customer_id
      AND t19932.item_id = t19957.item_id
      AND t19957.day_id = t19965.day_id)
GROUP BY t19880.all_regions_id,
         t19932.all_products_id,
         t19965.calendar_year_id,
         t19965.calendar_year_end_date
ORDER BY c5, c2, c3;
```

The SQL execution plan shows that this query is automatically rewritten to the cube-organized materialized. Cube-organized materialized views are easily recognized by the CB\$ prefix in the MV name.



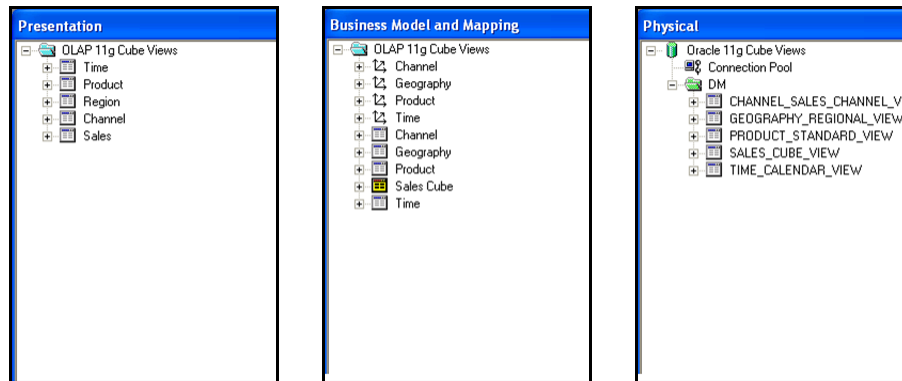
In this example, the entire query (dimensions and fact data) is resolved entirely in the cube. It is not uncommon for the query to require data from both the cube and in tables, in which case joins will be seen in the SQL execution plan.

IMPLEMENTING OBI EE ON OLAP CUBE VIEWS

Many interesting calculations can be defined in the cube and be accessed by SQL-based BI application. For example, it is very easy to embed time series, rankings, indices and market shares in the cube and query these measures as columns of a fact view over the cube. As a result, the cube can greatly enhance the analytic content of an OBI EE application. To allow OBI EE to access analytic content of a cube it must be configured to select data directly from the relational views of the OLAP cube.

Since OLAP cubes are presented to SQL based applications as dimension and fact views the process of implementing OBI EE on a cube is very similar to implementing OBI EE on a star schema. Like implementing OBI EE on tables, the three basic steps for implementing OBI EE on cube views are:

1. Creating the Physical Model by importing the views into the physical layer of the OBI EE repository.
2. Creating the Business Model by defining the structure of the model and the relationships between tables.
3. Creating the Presentation Layer by choosing objects from of the Business Model that should be made available to end users.



OBI EE Presentation, Business Model and Physical Layers mapped to OLAP cube views

Because OLAP cube views have some special rows and columns, there are a few differences in the OBI EE implementation. The remainder of this section describes these differences and walks through an implementation.

About OLAP Cube Views

Before getting into the specifics of implementing OBI EE on an OLAP cube it is useful to understand a few details about how data in the cube is represented in relational views.

OLAP cube views are relational views that select data from OLAP cubes and dimensions. Together, cube and dimension views form a star schema. Cube views are different than most relational tables or views in a star schema in two ways:

- First, dimension and cube views contain rows for all members of the dimension. That means that they contain rows for all levels of summarization, not just the lowest level values. In the case of the cube view, both detail and summary level data is presented as rows in the view.
- Second, cube views often contain columns with interesting non-additive facts. For example, it is likely that facts such as rankings, market shares and percent change period over period will be in the cube view.

Examine the following rows and columns from a cube view. Note that there are rows in the TIME column for both quarter and month level data and there are also summary level rows for the GEOGRAPY and CHANNEL columns. Also note that there are columns for facts such as prior period, period year, percent change

prior year, year-to-date and shares. Each of these columns represents a calculation that is defined and resolved dynamically in the cube.

TIME	GEOGRAPHY	CHANNEL	SALES	prior period	prior year	% chg...	ytd	share...
Feb-2007	North America	Internet	1335881	1744888	1143895	16.78	3080769	68
Feb-2007	Oceania	Internet	6799	4181	5841	16.41	10980	83
Feb-2007	Europe	Internet	739024	1053768	717856	2.95	1792792	66
Feb-2007	Asia	Internet	2388610	3063319	2070649	15.36	5451929	67
Feb-2007	Africa	Internet	205997	273322	201775	2.09	479320	65
Feb-2007	South America	Internet	541878	730242	472371	14.71	1272120	65
Q1-CY2007	Asia	Internet	8557143	7933142	7459199	14.72	8557143	68
Q1-CY2007	Africa	Internet	775200	726521	716793	8.15	775200	65
Q1-CY2007	Europe	Internet	2918844	2664679	2513493	16.13	2918844	67
Q1-CY2007	South America	Internet	2096128	2034985	1810034	15.81	2096128	67
Q1-CY2007	Oceania	Internet	14937	13230	16127	-7.38	14937	67

A sampling of rows and columns from an OLAP cube view. Note summary data and calculated measure columns.

This sampling of rows from a cube view illustrates two concepts which influence how to properly query the cube views with SQL:

- Since the cube calculates summary data, sometimes with aggregation methods not available in SQL, in most cases OBI EE should query summary data directly from the cube rather than using GROUP BY in SQL.
- The cube will often contain measures that should never be aggregated in SQL because doing so will return the wrong results. For example, it would rarely make sense to aggregate measures that are expressed in percentages or rankings on SQL above the cube.
- There are some columns in the cube view that users should not aggregate in SQL.

With the typical star schema, OBI EE must be configured so that it generates SQL that aggregates data from detail tables. Measure calculations might also be defined in the OBI EE server. This is unnecessary, and might even be undesirable, when querying the cube. In short, OBI EE must be configured in a way that allows the cube to do all the hard work.

What's Different about Configuring OBI EE to Query Cube Views?

To allow OBI EE to generate SQL that probably queries relational views of the OLAP cube, OBI EE should be configured:

- Using multiple Logical Table Source objects, each representing a specific level of summarization.
- With a security filter that forces joins between dimension views and the cube view in the case where a dimension is not represented in a query.

While not really different from any other implementation, it is useful to point out that it is very likely that cube views will contain measures such as rankings and percentages that should generally not be aggregated in SQL. The cube will calculate these measures at summary levels and aggregation should be turned off in OBI EE.

To OBI EE, the selection a column from a dimension table or view is an indication of what level of summarization is required. For example, if a user selects the DEPARTMENT column from a product dimension table OBI EE understands this as being a request for data summarized to the DEPARTMENT level. When querying a table, OBI will select as SUM(SALES) ... GROUP BY DEPARTMENT.

Because the cube view already includes rows for summaries, OBI EE needs to use a different method to indicate that it needs data at a particular level of summarization. Instead of using GROUP BY, queries need to filter on the LEVEL_NAME columns in the dimension views. For example, if department level data is needed the query should include a filter such as WHERE PRODUCT_VIEW.LEVEL_NAME = 'DEPARTMENT' as shown in the following query.

```
SELECT DISTINCT t269.calendar_year_long_descr AS c1,
               t200.department_long_descript AS c2,
               t230.sales_rank_by_prod_pr AS c3,
               t230.sales AS c4,
               t230.sales_ytd AS c5,
               t230.sales_ytd_pr_year AS c6,
               t230.sales_ytd_pr_yr_pct_c AS c7,
               t269.end_date AS c8
FROM channel_sales_channel_view t156,
     geography_regional_view t174,
     product_standard_view t200,
     time_calendar_view t269,
     sales_cube_view t230
WHERE (t156.dim_key = t230.channel
      AND t174.dim_key = t230.geography
      AND t200.dim_key = t230.product
      AND t230.TIME = t269.dim_key
      AND t156.level_name = 'ALL_CHANNELS'
      AND t174.level_name = 'ALL_REGIONS'
      AND t200.level_name = 'DEPARTMENT'
      AND t269.level_name = 'CALENDAR_YEAR')
ORDER BY c8, c2, c3, c4, c5, c6, c7;
```

In this example SELECT statement, note the following:

- There are no aggregation functions on the measure columns in the select list. The cube aggregates data, so OBI EE does not need to.

- There is no GROUP BY clause because there are no aggregation functions.
- There are filters on LEVEL_NAME columns of the dimension views. This filter effectively replaces the group by clause by indicating the level of summarization for data required from the cube.
- Although only the time and product dimensions included in this query, all dimensions are represented in the select statement. Data for the customer and channel dimensions is returned for the top, or total, of each dimension.

The techniques described in the sample implementation will cause OBI EE to generate this style of query automatically.

Sample Implementation

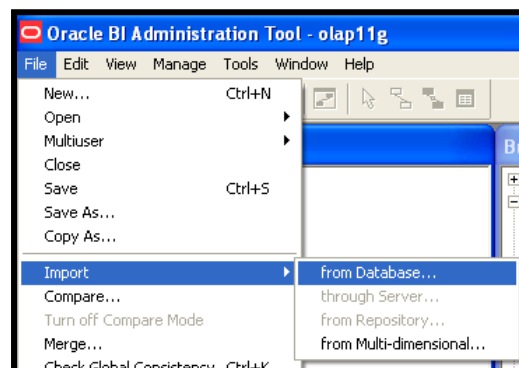
The remainder of this section will walk through a sample implementation of OBI EE on cube views. All major tasks will be presented, but special attention will be given to those tasks that are specific to the cube view implementation.

An outline of implementation tasks follows:

1. Create new physical layer by importing cube and hierarchy views from the Oracle database.
2. Create a new Business Model and Mapping layer by creating Logical Source tables, OBI EE Dimension objects and joins between hierarchy and cube views.
3. Create new Presentation layer.
4. Define Security filter to force filters on level_name columns for the “dropped dimension” case.

Physical Layer

The only task that needs to be completed in the Physical Layer is importing cube and dimension or hierarchy views. Because relational views of cube and hierarchies are used in this implementation, use the File > Import > from Database command (rather than File > Import > from Multidimensional command).

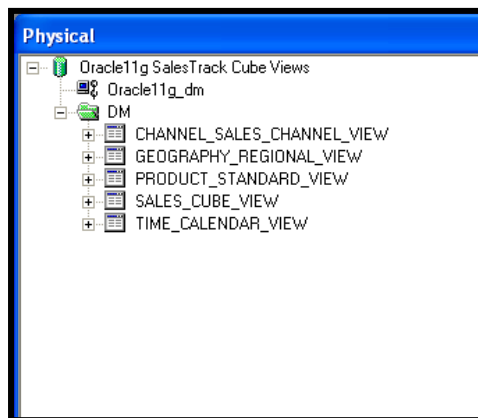


Use the Import > from Database command when importing cube and hierarchy views.

The Database provides two styles of views for OLAP Option dimensions – dimension views and hierarchy views. Dimension views contain rows for all dimension members. If there are multiple hierarchies in the dimension, the dimension view will contain members from all hierarchies. Hierarchy views contain rows for only those dimension members that belong to a specific hierarchy.

Either style view can be used with OBI EE. This paper uses a hierarchy view as the example. The techniques are the same if dimension views are used instead. Depending on your data, there might be subtle differences in the results of queries depending on which style view is used. Experiment and choose the style view that is best for your application.

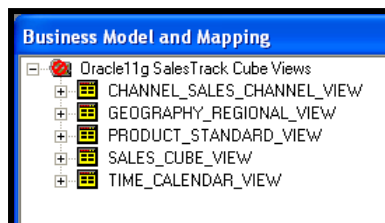
Import the cube view and one dimension or hierarchy view per dimension. Nothing else needs to be done in the Physical Layer (for example, primary and foreign keys do not need to be defined in the Physical Layer).



Physical Layer with cube and hierarchy views.

Business Model Layer

Begin the process of working in the Business Model layer by defining a new business model bringing the cube and dimension or hierarchy views into it.

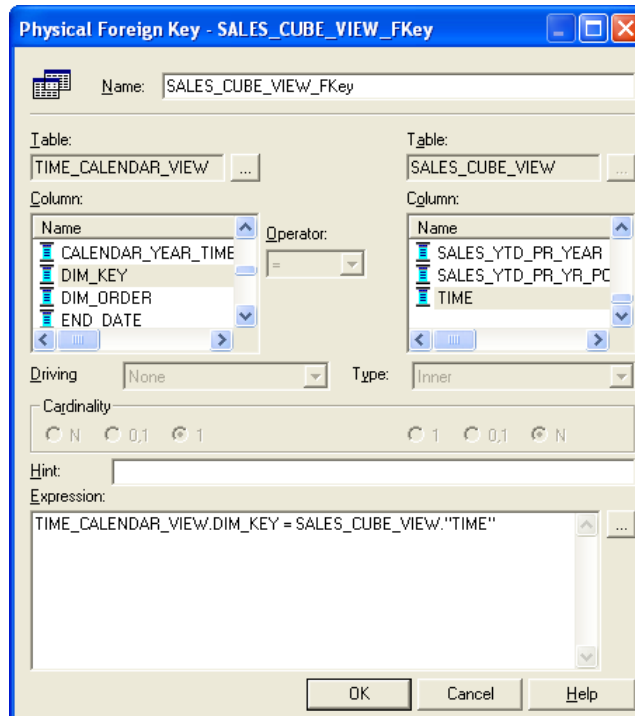


Oracle11g SalesTrack Cube Views business model with cube and hierarchy views.

Create Foreign Keys

Next, go to the Physical Diagram and create foreign keys on the cube view. For each dimension create one foreign key that joins the DIM_KEY column of the

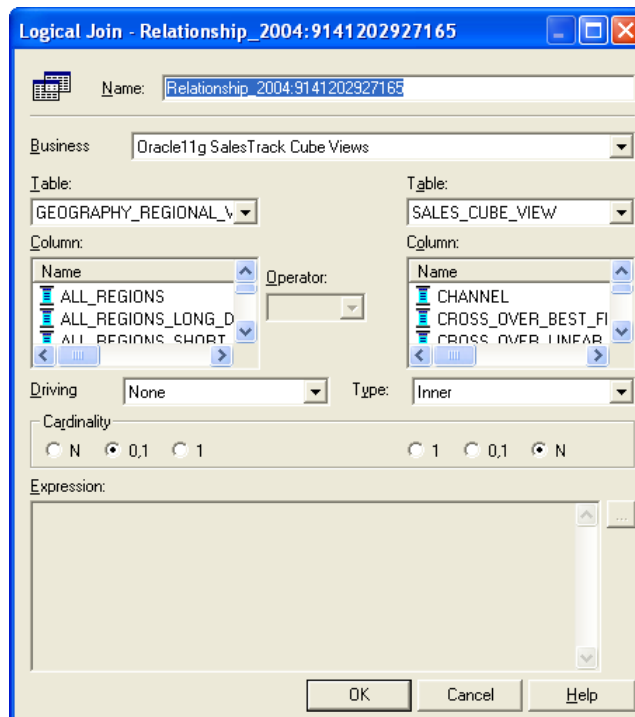
dimension or hierarchy view to the column representing that dimension in the cube view.



Creating a foreign key for the Time dimension.

Create Logical Joins

After creating foreign keys for each dimension, enter the Business Model diagram and create logical joins between each dimension or hierarchy view and the cube view as shown in the following example.



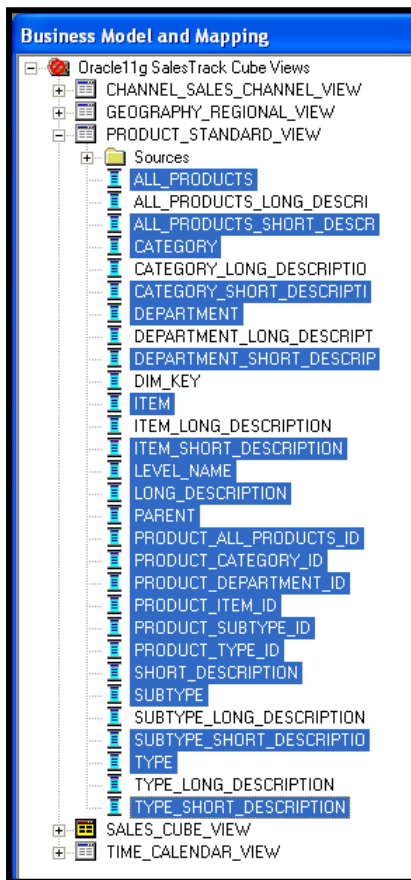
Creating logical join between Geography hierarchy view and the cube view.

Remove Unneeded Columns from the Logical Table

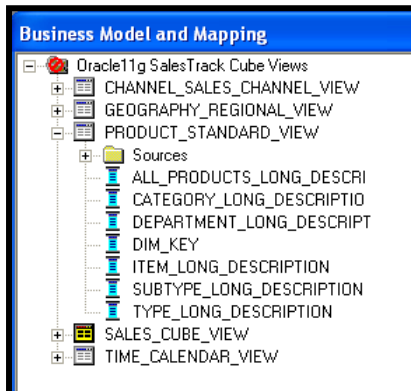
Not all columns in the dimension or hierarchy view are needed by OBI EE. In most cases, the following columns should be retained and others can be removed from the Logical Table objects:

- The *level_LONG_DESCRIPTION* columns are used to view and filter dimension members. (There will be one of these columns for each level in the dimension or hierarchy view when the *Create level attribute column in views* option is selected for the attribute in Analytic Workspace Manager. This is the default for long description attributes.)
- The DIM_KEY column.
- Columns representing attributes (e.g., color, size, income range, etc.).
- The END_DATE column in the time dimension view.

Other columns can be removed from the Logical Tables.



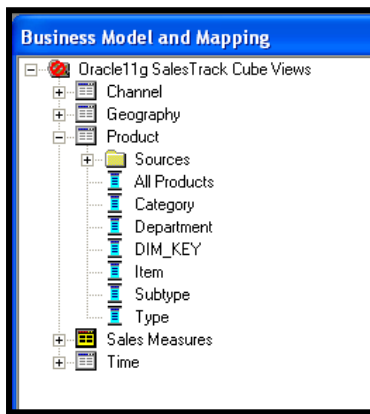
These selected columns are not needed in OBI EE business model and can be removed from the Logical Table.



Columns that have been retained for use in the OBI EE business model.

Clean Up Names of Logical Tables and Columns

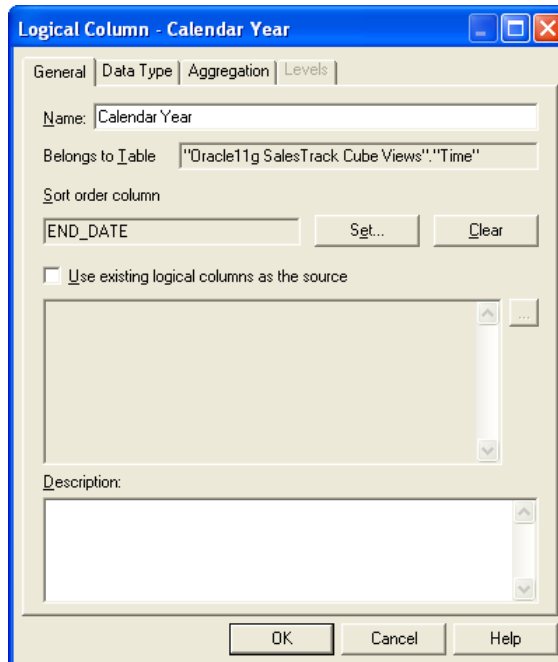
Now is a good time to clean up the names of Logical Tables and Columns. Edit them to provide the names that will be seen by end users.



The Business Model after object names has been edited for viewing by end users.

Choose the Sort Order Column for the Time Dimension

The END_DATE column can be used as the sort order column to present members of a time dimension in chronological order. Repeat this for each logical column.

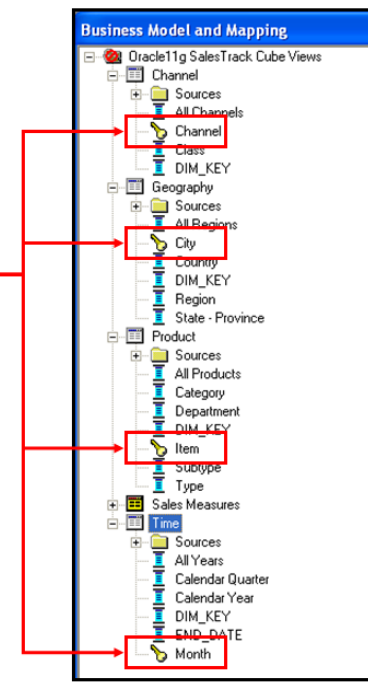


The END_DATE column is used for sorting members of a time dimension.

Create Logical Keys for Each Dimension

For each dimension, define a key using the column representing the lowest level of the dimension.

⇒ For each Logical Table, create a Key using the lowest level Logical column.



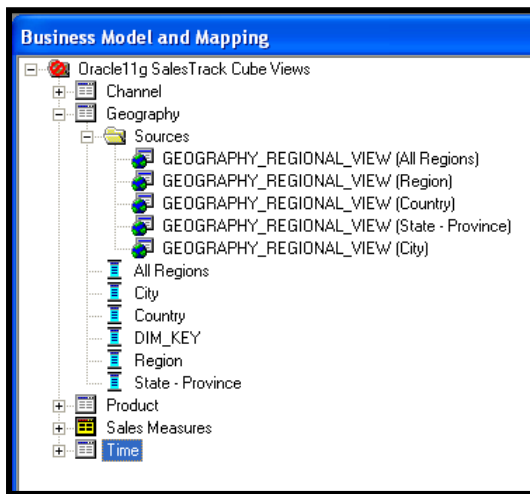
Keys defined for each logical table representing a dimension or hierarchy.

Create Logical Table Source for Each Level of a Dimension

Because the dimension, hierarchy and cube views contain rows for all levels of summarization, OBI EE needs add filters to queries so that only data needed by selected columns is returned. This is done by creating on logical table source for each dimension and adding a where clause filter to each.

Begin this process by creating one Logical Table Source for each level in the dimension or hierarchy view. The name of the Logical Table Source is not important to OBI EE, but naming the Logical Table Source after the level helps to document its use.

The Logical Table Source objects should be in order of highest to lowest level of summarization as shown in the following example.

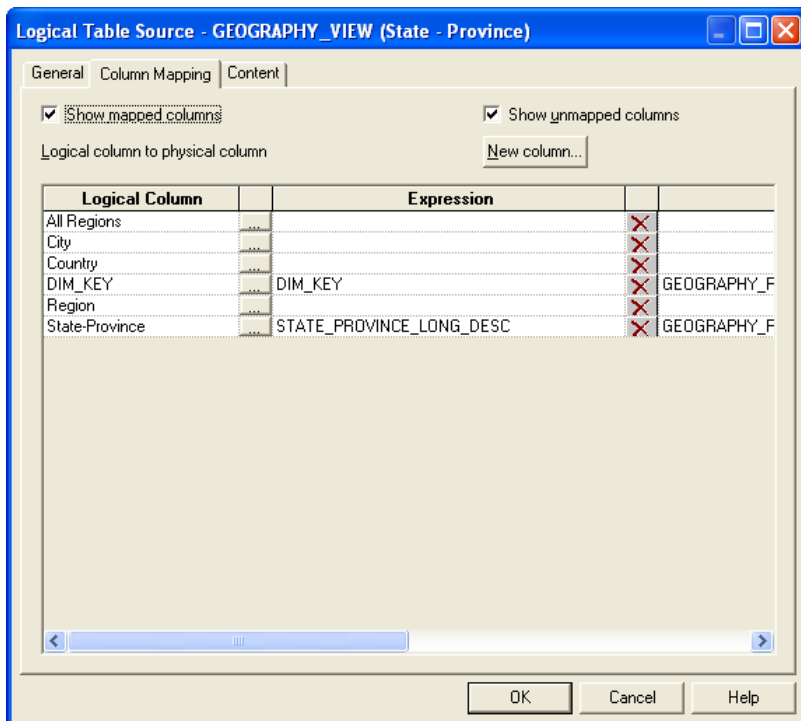


One Logical Table Source has been created for each level of the Geography logical table. Note the logical table sources are sort from highest to lowest level of summarization.

Edit Logical Table Source Properties

When logical table sources are created, they will contain expressions for each logical column. In the case of cube and dimension views, some expressions need to be removed and a where clause filter needs to be added. For each logical table source, do the following:

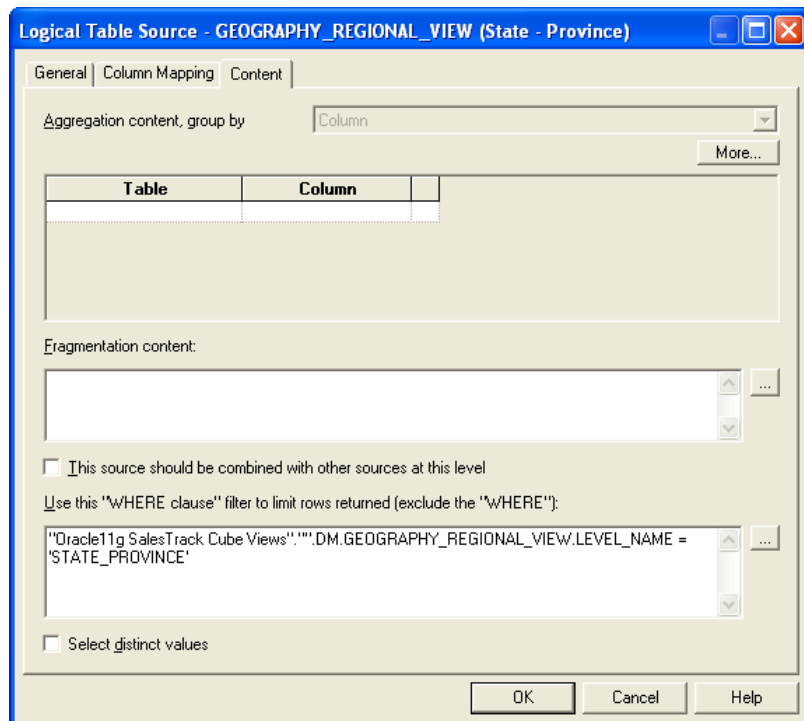
- Remove expressions for each logical column that do not belong to the level of the logical table source.
- Add a where clause filter that uses the LEVEL_NAME column to filter the dimension to members of the level of the logical table source.
- Keep the expression of the DIM_KEY logical column.



This logical table source represents the State-Province level. Keep expressions for the DIM_KEY column and the State-Province logical columns and remove expressions for columns representing other levels.

If the logical table contains logical columns that represent attributes, keep expressions for the columns when the attribute applies to the level of the logical table. For example, if there is a State-Province Population column used as an attribute, the expression should be kept for the State-Province logical table source. Expressions for the attribute columns should be removed for logical table sources representing other levels.

Next, add a where clause filter that uses the LEVEL_NAME column and filters to the level of the logical table source. Note that the level name is the name of the level object in the OLAP dimension.

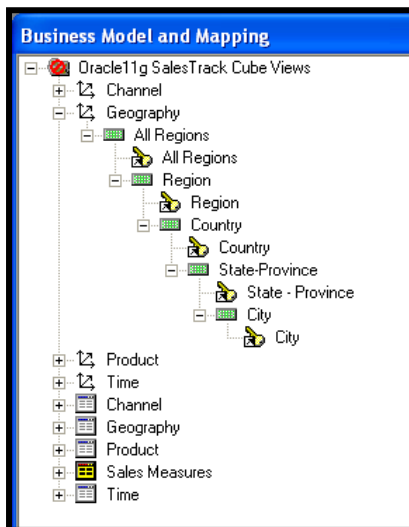


The where clause filter of the logical table source representing the State-Province level.

Create OBI EE Dimension Objects

The next step is to create OBI EE dimension objects. There are no cube-specific parts to this task. For each dimension:

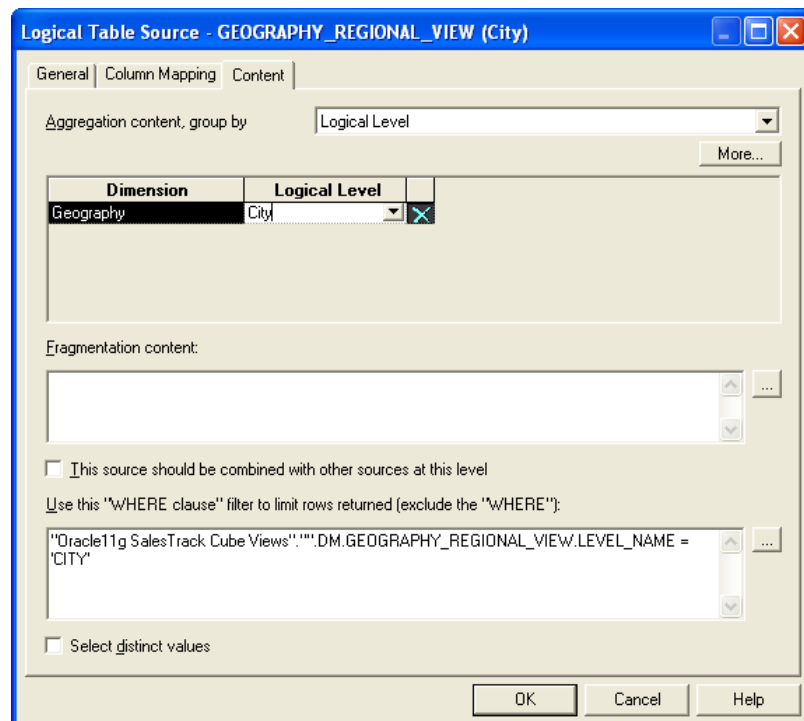
5. Create an OBI EE dimension object.
6. Add levels to the OBI EE dimensions.
7. Add logical columns to levels of the dimension objects.
8. Create keys for each level of the dimension object.



A business model with dimensions.

Set Logical Level of Logical Table Sources

After creating OBI EE dimensions, set the logical level of each logical table source as shown below. The logical level should be consistent with the level in the where clause filter.



Logical table source with logical level and where clause filter.

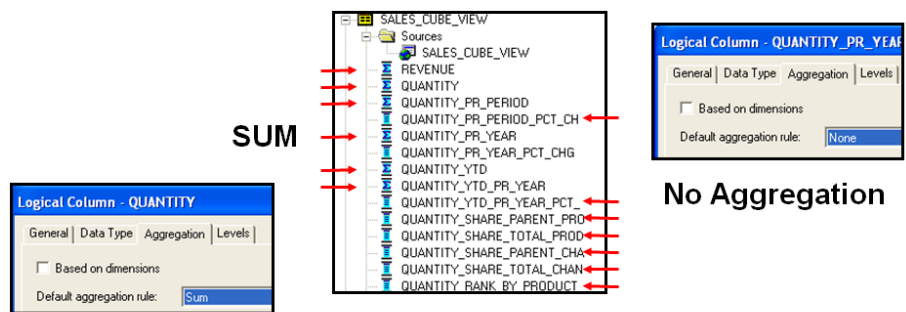
Set Aggregation Methods for Fact Columns

OBI EE typically includes an aggregation function and GROUP BY clause in each query of a fact table. With the cube, the aggregation function is not needed when querying columns representing levels because the cube provides the aggregated data. If there is an aggregation function in a query that selects only level columns and facts it will have no effect because all rows returned by the cube are be unique.

When querying columns representing attributes an aggregation function is required. For example if income range and education range are attributes in the OLAP dimension and are queried using SQL, a GROUP BY on income range and education range will be required because this data is not aggregated in the OLAP cube.

Aggregating some measures in SQL yields useful results. For example, a SUM(sales) group by income range and education range makes sense. Aggregating other types of measures in SQL might not yield useful results. For example SUM(rank) ... GROUP BY income range and education range is probably not very meaningful. In general, this will be true of measures such as rankings and those that are expressed as percentages (e.g., percent changes in years year-to-date).

If the measure can be aggregated in SQL above the cube, choose the appropriate aggregation function. In most cases, the SQL aggregation function should be consistent with the aggregation function in the cube. If aggregation in SQL will not yield meaningful results, consider setting the aggregation rule in OBI EE to 'none'.



Setting aggregation rules for measures.

Create Security Filter for the "Dropped Dimension" Case

In OBI EE it is quite easy to create a query that includes columns that represent only a subset the dimensions of a cube. For example, if the cube is dimensioned by time, product, geography and channel a user can easily create a report that includes only product by time as shown in the following example.

Calendar Year	Department	Sales
CY2002	CAMERAS_AND_CAMCORDERS	4,848,695
	COMPUTERS	62,913,851
	PORTABLE_MUSIC_AND_VIDEO	12,291,733
CY2003	CAMERAS_AND_CAMCORDERS	5,820,123
	COMPUTERS	69,704,478
	PORTABLE_MUSIC_AND_VIDEO	13,247,392
CY2004	CAMERAS_AND_CAMCORDERS	7,308,918
	COMPUTERS	79,158,141
	PORTABLE_MUSIC_AND_VIDEO	14,801,493
CY2005	CAMERAS_AND_CAMCORDERS	8,865,197
	COMPUTERS	88,709,396
	PORTABLE_MUSIC_AND_VIDEO	16,064,530
CY2006	CAMERAS_AND_CAMCORDERS	10,657,143
	COMPUTERS	101,669,050
	PORTABLE_MUSIC_AND_VIDEO	17,328,207
CY2007	CAMERAS_AND_CAMCORDERS	12,387,923
	COMPUTERS	115,957,793
	PORTABLE_MUSIC_AND_VIDEO	19,101,368

OBI EE Report of Sales by Year and Department

When querying [tables](#), the query that OBI EE generates for this report follows.

```

SELECT t19965.calendar_year_id AS c1,
       t19932.department_id AS c2,
       SUM(t19957.sales) AS c3,
       t19965.calendar_year_end_date AS c4
FROM times t19965,
     products t19932,
     sales_fact t19957
WHERE(t19932.item_id = t19957.item_id
      AND t19957.day_id = t19965.day_id)
GROUP BY t19932.department_id,
         t19965.calendar_year_id,
         t19965.calendar_year_end_date
ORDER BY c4, c2;

```

Since the SALES_FACT table contains only detail level data, this query can aggregate all values of geography and channel to arrive at the correct result. If this query was issued against an OLAP cube view it would not return the expected results because the cube contains both detail and aggregate level data. This query would re-aggregate data for geography and channel that has already been aggregated in the cube.

To resolve this issue every dimension should be to be represented in the SQL query and the dropped dimension should be filtered to a single member at the top of the hierarchy. Any aggregation that SQL might do in the query is harmless because it will not reaggregate data that has been aggregated in the cube. An OBI EE security filter that includes joins between every dimension view and the cube view can be

used to cause OBI EE to include every dimension in the query, even if it is not included in the OBI EE report. To create this security filter, do the following:

9. Create a new User Group.
10. Add a Security Filter to that User Group.
11. Assign users who will query the cube to the User group.

The security filter should be applied to the cube view in the business model layer. The expression of the filter should take the form:

```
dimension logical table.DIM_KEY = cube view logical table.dimension  
AND dimension logical table.DIM_KEY = cube view logical table.dimension  
AND dimension logical table.DIM_KEY = cube view logical table.dimension
```

For example:

```
"OLAP 11g SalesTrack Cube Views"."Time"."DIM_KEY" =  
    "OLAP 11g SalesTrack"."SALES_CUBE_VIEW"."TIME"  
AND "OLAP 11g SalesTrack"."Product"."DIM_KEY" =  
    "OLAP 11g SalesTrack"."SALES_CUBE_VIEW"."PRODUCT"  
AND "OLAP 11g SalesTrack"."Customers"."DIM_KEY" =  
    "OLAP 11g SalesTrack"."SALES_CUBE_VIEW"."CUSTOMER"  
AND "OLAP 11g SalesTrack"."Channel"."DIM_KEY" =  
    "OLAP 11g SalesTrack"."SALES_CUBE_VIEW"."CHANNEL"
```

With this security filter, OBI EE will automatically:

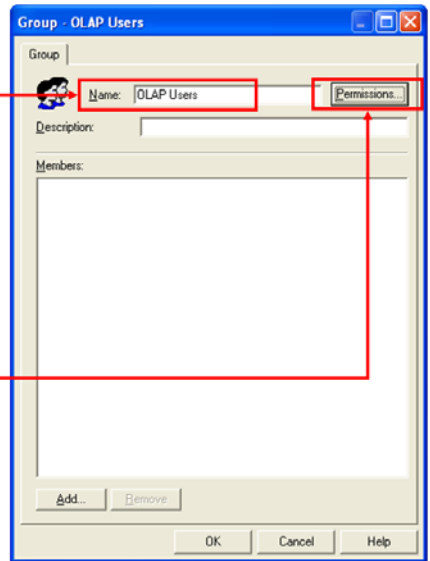
- Include every dimension or hierarchy view in the query.
- All a level filter for any dropped dimensions. OBI EE will filter to the highest level of the dropped dimensions.
- Join the dimension or hierarchy and cube views.

As a best practice, include a level in every OLAP hierarchy that contains a single member representing the aggregation of the entire dimension. For example, All Years or Total Product. This will ensure that all aggregation takes place in the OLAP cube and allows the use of non-additive aggregation methods and custom measures (e.g., ranks and percentages) within OBI EE queries.

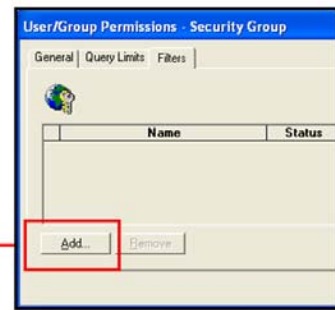
The following illustrations walk through the process of creating a security filter for OLAP cube views.

1 ⇐ Choose the Choose the Action/New/User Group command to access the Group dialog and name a new user group.

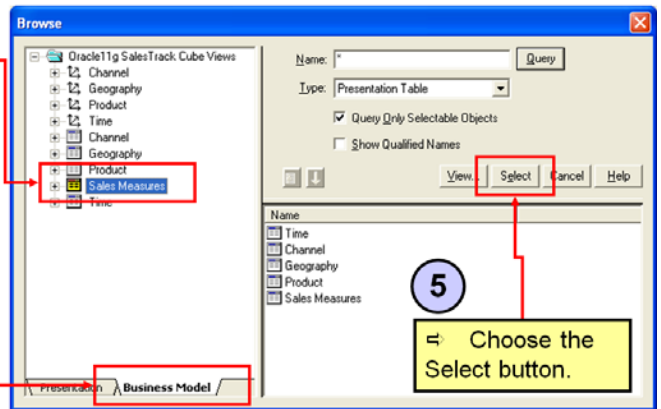
2 ⇐ Choose the Permissions button to begin defining a new security filter.



3 ⇐ Choose the Filters tab and the Add button.



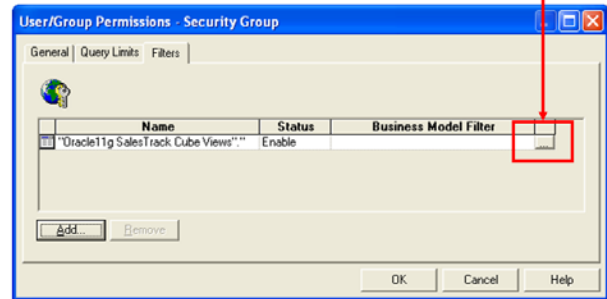
4 ⇐ Choose the Business Model tab and the cube view.



5 ⇐ Choose the Select button.

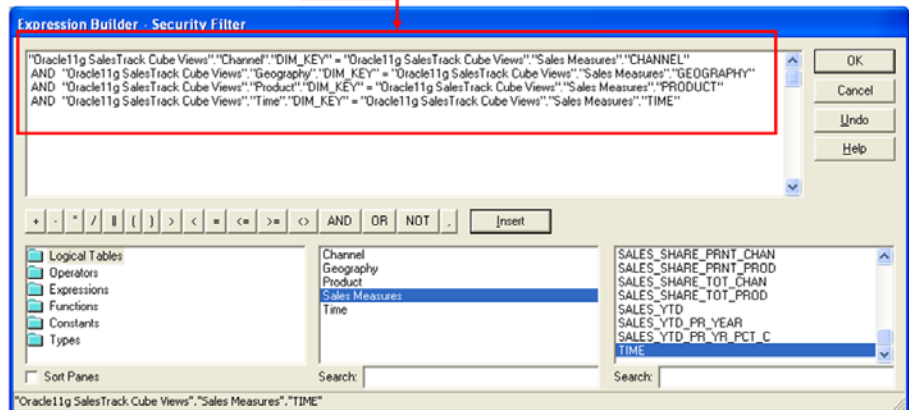
6

⇐ Choose to edit the Business Model Filter.



7

⇐ Define a filter by joining the DIM_KEY of each hierarchy view to the corresponding key of the cube view.



```
"OLAP 11g SalesTrack Cube Views"."Time"."DIM_KEY" =
  "OLAP 11g SalesTrack"."SALES_CUBE_VIEW"."TIME"
AND "OLAP 11g SalesTrack"."Product"."DIM_KEY" =
  "OLAP 11g SalesTrack"."SALES_CUBE_VIEW"."PRODUCT"
AND "OLAP 11g SalesTrack"."Customers"."DIM_KEY" =
  "OLAP 11g SalesTrack"."SALES_CUBE_VIEW"."CUSTOMER"
AND "OLAP 11g SalesTrack"."Channel"."DIM_KEY" =
  "OLAP 11g SalesTrack"."SALES_CUBE_VIEW"."CHANNEL"
```

To test the security filter, run a report that contains only a subset of the dimensions of the cube and view the SQL of that report. For example, the following report includes sales by year and drops the product, geography and channel dimensions.

Important note: When a user is logged into an OBI EE web client such as Answers with the Administrator role, security filters are not applied, so queries selecting from cubes can be very slow and results will likely be incorrect.

Year	Sales YTD	Sales YTD % Chg Pr Year
CY2002	80,054,280	
CY2003	88,771,993	11
CY2004	101,268,551	14
CY2005	113,639,123	12
CY2006	129,654,400	14
CY2007	147,447,085	14

Sales Year-to-Date and Percent Change Year-to-Date by Year

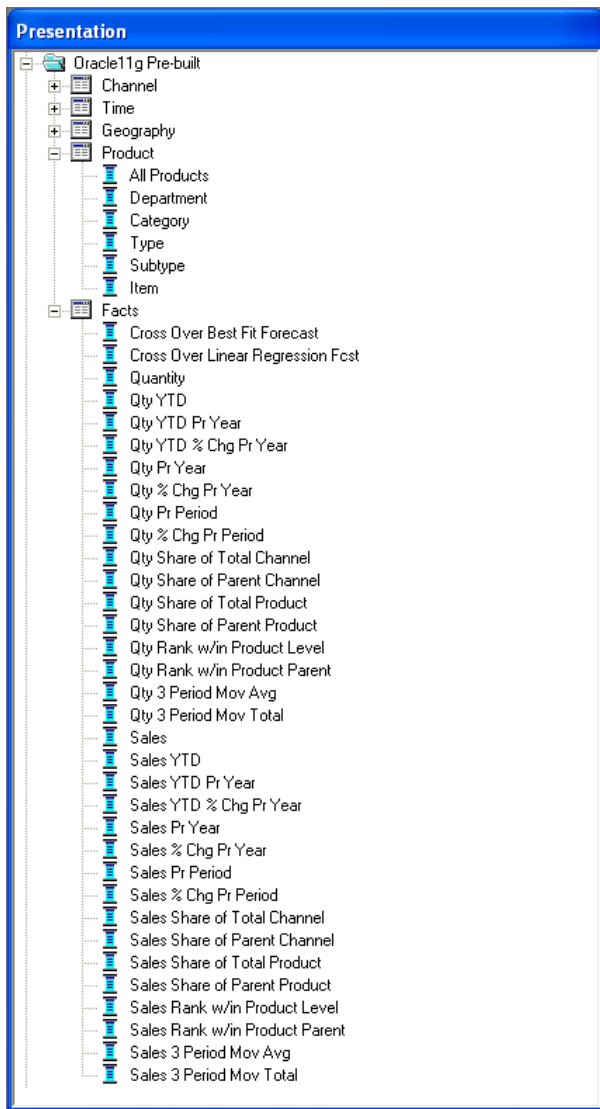
The SQL issued by OBI EE for this report includes level filters and joins for all dimensions, including the 'dropped' dimensions (shown in bold text).

```
SELECT DISTINCT t19262.calendar_year_long_descr AS c1,
               t19225.sales_ytd AS c2,
               t19225.sales_ytd_pr_yr_pct_c AS c3,
               t19262.end_date AS c4
FROM channel_sales_channel_view t19151,
     product_standard_view t19195,
     geography_regional_view t19169,
     time_calendar_view t19262,
     sales_cube_view t19225
WHERE(t19151.dim_key = t19225.channel
      AND t19169.dim_key = t19225.geography
      AND t19195.dim_key = t19225.product
      AND t19151.level_name = 'ALL_CHANNELS'
      AND t19169.level_name = 'ALL_REGIONS'
      AND t19195.level_name = 'ALL_PRODUCTS'
      AND t19225.TIME = t19262.dim_key
      AND t19262.level_name = 'CALENDAR_YEAR')
ORDER BY c4,c2,c3;
```

Presentation Layer

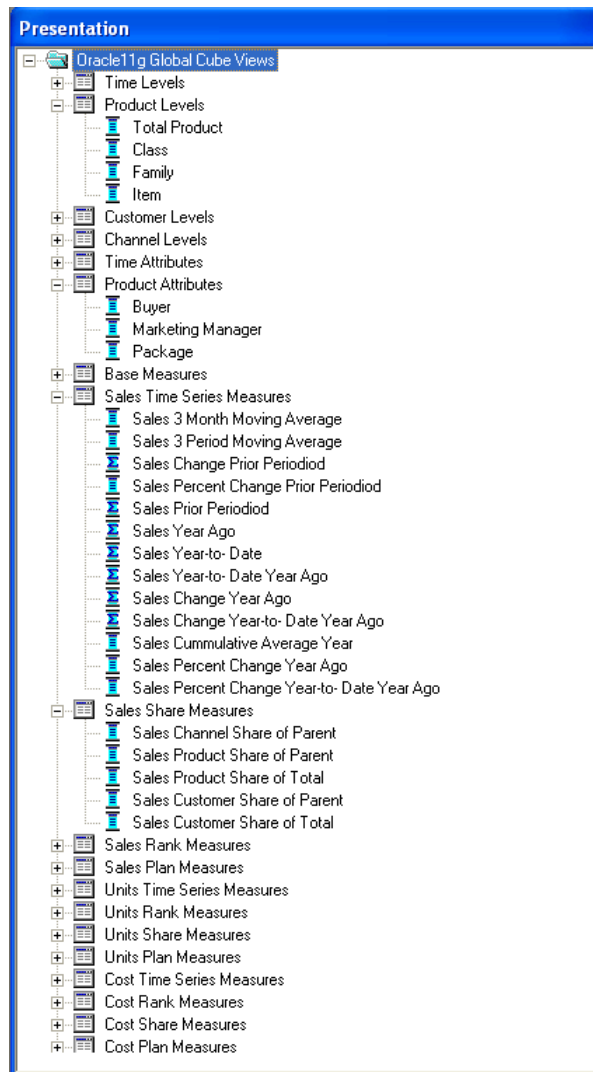
After the business model layer is complete, create the presentation layer. There are no tasks that are specific to the OLAP cube in the presentation layer. What might be different about creating a presentation layer that queries an OLAP cube is the likelihood that there will be many, many interesting measures and that some measures might not be appropriate for reporting by attribute columns.

If the business model is relatively simple, a simple presentation layer might work well as shown in the following example.



Presentation layer with a simple organization.

If the model is more complex, perhaps with many attribute columns and many calculated measures it might be useful organize columns into different presentation tables as shown in the next example.



Presentation layer that organizes columns by level, attribute and measure type.

CONCLUSION

Oracle cubes in Oracle Database 11g provide the means to enhance the query performance and analytic content of Oracle Business Intelligence Enterprise Edition. As cube-organized materialized views, cubes can be introduced into the OBI EE environment transparently. Summary data is managed in the cube and is accessed transparently via the automatic query rewrite feature of the Database. End users experience improved query performance. The database administrator benefits from fast, incremental updates and the requirement that they only need to manage a single cube instead of 10's, 100's or 1,000's of table based materialized views.

When OBI EE queries cube views directly, it can access all of the advanced analytic content that is available in the cube. End users benefit from excellent query performance and an enriched reporting and analysis experience.



The OLAP Option to Oracle Database 11g

July 2008

Author: William Endress

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

Copyright © 2007, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.