

OPTIMIZING BI QUERIES

OLAP Option Compressed Composites

Overview

Anyone who has ever worked with Oracle's OLAP Option knows just how significantly it can improve query performance for data warehouses and relational OLAP environments, regardless of what tools or applications you are running against Oracle. But OLAP is more than just an option that you switch on – it is a powerful and flexible solution that can be tuned and optimized for your particular application. One such capability is leveraging compressed composites inside OLAP to dramatically improve query performance while reducing data volumes.

If you have worked with hierarchies, you know well that there are many summary nodes that are repeats. This happens anywhere a parent has only one child – the parent and the child values will be the same. A “skip-level” hierarchy has very noticeable redundancy, but even value-based hierarchies will have some.

To address this situation, the Oracle OLAP Option in the Enterprise Edition of the database offers “compressed composites”. This method will NOT store the redundant cells, will NOT create a space for them in its composite structure, and will NOT build an index entry to map them. Thus, all these structures are smaller and perform better.

Areas that will benefit from the use of compressed composites:

- BI and Data Warehouse systems looking to reduce query response time.
- Systems support staff looking to eliminate the effort of managing Materialized Views for summary level data.
- Legacy Express applications (OFA/OSA/OEO) that have been migrated into the OLAP Option.

We will look at the performance difference by comparing query run times of data arrays with standard composites and data arrays with compressed composites.

The structures described below were built using Escendo Architect, a workbench for developing, deploying, and managing Oracle OLAP environments. Architect allows complete access to all of the underlying capabilities of Oracle OLAP including its analytic functions, the OLAP DML, and capabilities such as Compressed Composites. To learn more about Escendo Architect and the rest of the Escendo Analytics family of OLAP-optimized BI solutions, visit www.Escendo.com.

Data Model

Relational View

The star schema has 5 dimension tables:

Dimension	Rows	Hierarchy	Description
-----	-----	-----	-----
CUST	54,217	Yes	Customers
KIT	251	Yes	Product with components/accessories as put in a box
SMNTH	50	Yes	Shipping month
CHNL	6	Yes	Distribution channel
REGCTR	5	Yes	Regional processing center

The fact table has 18 attribute columns in addition to the 5-column primary key. These attributes track volume, direct cost, allocated expenses, revenue, discounts, etc.

OLAP View

```
DEFINE array VARIABLE data_type <METRIC composite <CUST KIT SMNTH CHNL REGCTR>>
```

The measure has 6 dimensions, with the last 5 being in a composite; these map to the relational dimension tables. The sixth dimension, METRIC, contains 18 members corresponding to the attribute columns of the fact table.

Data Volume

The unsummarized fact table has 1,917,582 rows. Fully summarized, the row count expands to 44,279,098. These two numbers correspond to the tuple count of an OLAP standard composite for leaf values only and for a fully calculated state.

Query Scenarios

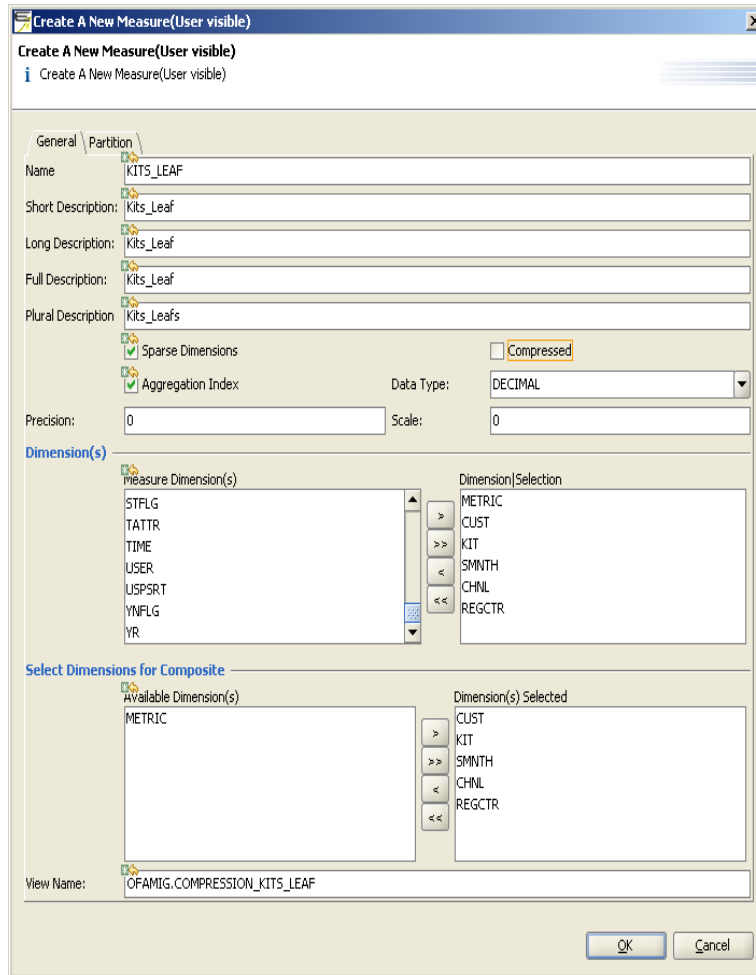
Explore the query response time in the OLAP option of:

1. On-the-fly aggregation using a standard composite with only leaf values loaded.
2. On-the-fly aggregation using a compressed composite with only leaf values loaded.
3. On-the-fly aggregation using a compressed composite with some summary values precomputed.

Query times with fully loaded values will be tested as a benchmark.

Create the Test Measures

Regular Composite



The screen shot above shows the specifications of creating a new 6-dimensional measure with the last 5 dimensions in the composite. This will instantiate the following OLAP objects:

User Visible Measure (Materialized View)

The user visible object will return all data cells as fully resolved, that is, as if all values are stored in the database. This is done on-the-fly; it is similar to a Materialized View, except that you don't actually precompute and store summary values.

```
describe kits_leaf
DEFINE KITS_LEAF FORMULA DECIMAL <METRIC CUST KIT SMNTH CHNL REGCTR>
EQ AGGREGATE(this_aw!KITS_LEAF_STORED using this_aw!KITS_LEAF_AGGMAP)
```

Data Array (Fact Table)

Some data is of course stored. Here we will store only the lowest level of detail, no summary data at all.

```
describe kits_leaf_stored
  DEFINE KITS_LEAF_STORED VARIABLE DECIMAL <METRIC KITS_LEAF_COMPOSITE
    <CUST KIT SMNTH CHNL REGCTR>>
```

Composite (Primary Key / Index)

A composite tracks all combinations of its dimensions that are actually used; it is maintained automatically, so it is never stale or out of date.

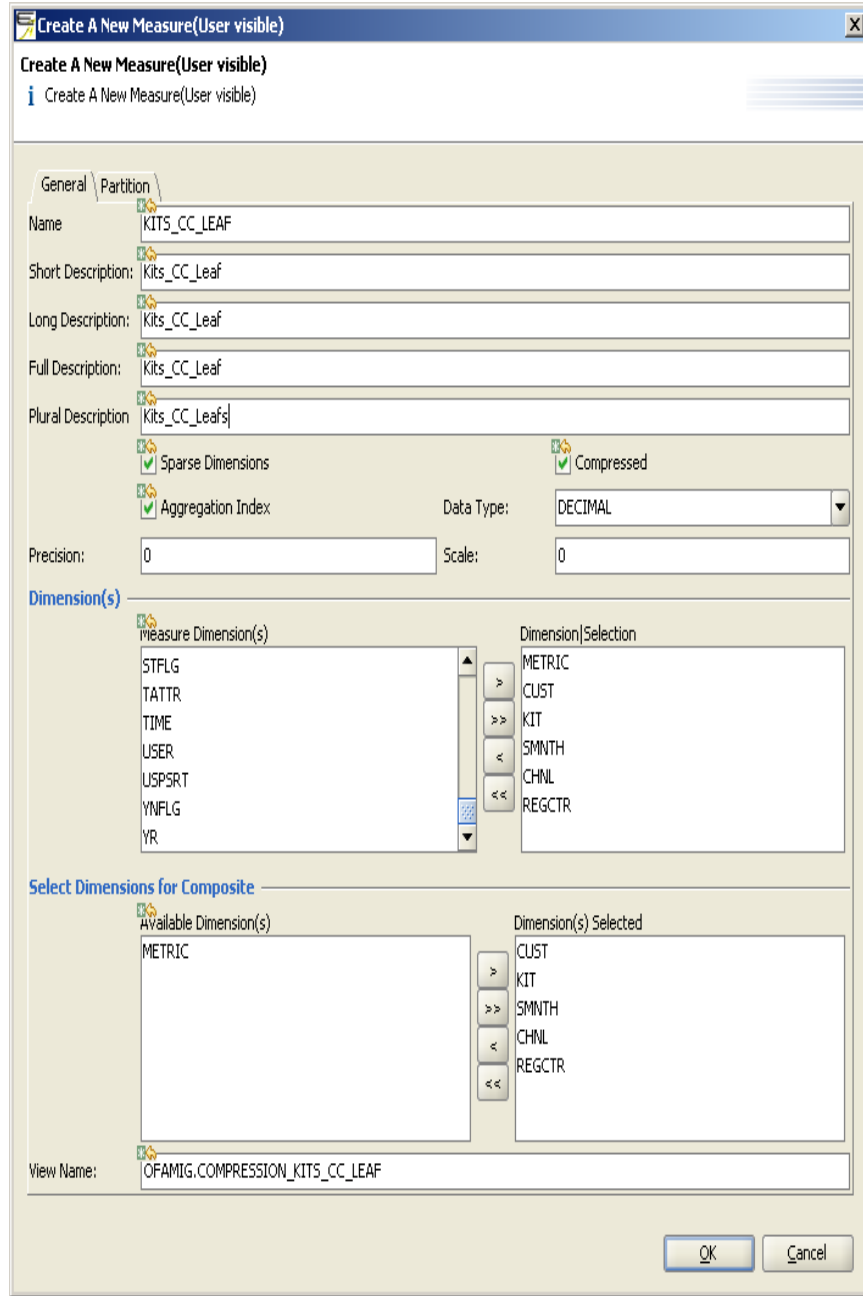
```
describe kits_leaf_composite
  DEFINE KITS_LEAF_COMPOSITE COMPOSITE <CUST KIT SMNTH CHNL REGCTR>
```

Aggregation Map (Materialized View part II)

This specifies the on-the-fly details. Here we are using the dimension hierarchies and are doing a simple arithmetic sum, and are not precomputing any values, i.e., all summarized data is calculated on-the-fly.

```
describe kits_leaf_aggmap
  DEFINE KITS_LEAF_AGGMAP AGGMAP
  AGGMAP
  RELATION this_aw!CUST_parentrel operator SUM precompute (NA)
  RELATION this_aw!KIT_parentrel operator SUM precompute (NA)
  RELATION this_aw!SMNTH_parentrel operator SUM precompute (NA)
  RELATION this_aw!CHNL_parentrel operator SUM precompute (NA)
  RELATION this_aw!REGCTR_parentrel operator SUM precompute (NA)
  END
```

Compressed Composite



The screen shot above shows the specifications of creating a new 6-dimensional measure with the last 5 dimensions in a compressed composite. This will instantiate the same type of OLAP objects as above with the exception of the definition of the compressed composite:

```
describe kits_cc_leaf_composite
    DEFINE KITS_CC_LEAF_COMPOSITE COMPOSITE <CUST KIT SMNTH CHNL REGCTR>
    COMPRESSED
```

Fully Resolved

Two more measures were created. KITS_ALL was cloned from KITS_LEAF, including the leaf-level data. Then it was aggregated using the aggmap, but this time the calculated data was saved. KITS_CC_ALL was similarly cloned and fully resolved from KITS_CC_LEAF, then stored with the calculated data.

The query was run against the _STORED variable for these two measures. There is no need for on-the-fly aggregation when all values have been already computed.

Running the Query

The following OLAP query was used:

```
show tod
limit CUST    to 'TOT.CUST'
limit KIT     to all
limit SMNTH   to 'Y2009'
limit CHNL    to 'TOT.CHNL'
limit REGCTR  to 'TOT.REGCTR'
limit METRIC  to first 3
rpr down KIT w 20 < measure_name > e.g., < kits_leaf >
show tod
```

Measure	KITS_LEAF	KITS_CC_LEAF	KITS_ALL	KITS_CC_ALL
Run Time (seconds)	52	31	0	0
Measure Cells	34,516,476	34,516,476	797,023,764	134,327,772
Measure Size (MB)	272	272	6,290	1,060
Composite Tuples	1,917,582	1,917,582	44,279,098	7,461,845
Composite Size (MB)	19	19	867	73

Results

Comparing KITS_LEAF and KITS_CC_LEAF shows that by simply defining a composite as compressed you can get better response time. Comparing KITS_ALL and KITS_CC_ALL shows the physical side of the compressed composites' feature of not storing repeated summary values. The stored array, along with the composite, is much smaller. Note however, that even the larger stored array gives excellent response time – returning with the same timestamp second as the request timestamp.

Partial Aggregation

Although simply aggregating everything using a compressed composite may be a good solution for some reporting, there may be other applications where this is not viable. For example, budget submission or modeling with shop-floor or RSS feeds entails data changing in real or near-real time. Summary results need to be calculated on-the-fly on the slice of data under consideration. Running and saving a full aggregation may have to wait until off-peak time.

There are several approaches for partially aggregating results. If any of the hierarchies are level based, one

method is to aggregate alternating levels. Another method is to simply aggregate along one of the dimensions and then remove that from the aggmap.

Looking at the dimensions for this example, it seems obvious that the CUST dimension is the largest by far and may be suitable for precomputing. An evaluation shows that customers roll up into marketing groups, then into the total. Since there are several hundred marketing groups (for 50,000+ customers) this means that precomputed results along this dimension may have a great impact. For example, if the numbers change for one CUST, then only the summary for its marketing group parent needs to be calculated – no other marketing groups have changed. Not only are thousands of CUSTs not touched, they are not touched across other dimensions, e.g., TIME, CHNL and REGCTR are not recalculated for these other CUSTs. The aggregation now uses precomputed, stored summary values as a starting point for all the other marketing groups rather than leaf level data.

To accomplish this, create a new aggmap to precompute totals along the CUST dimension, run the aggregation, store the data, and remove the CUST line from the original aggmap. This separates the process into a partial aggregation at data load time and the regular on-the-fly run time aggregation.

Create a partial aggregation aggmap

```
describe kits_cc_load_agg
  DEFINE KITS_CC_LOAD_AGG AGGMAP
  AGGMAP
  RELATION this_aw!CUST_parentrel
  RELATION this_aw!KIT_parentrel operator SUM precompute (NA)
  RELATION this_aw!SMNTH_parentrel operator SUM precompute (NA)
  RELATION this_aw!CHNL_parentrel operator SUM precompute (NA)
  RELATION this_aw!REGCTR_parentrel operator SUM precompute (NA)
  END
```

Run and Save the data

```
aggregate KITS_CC_LEAF_STORED using KITS_CC_LOAD_AGG
update
commit
```

Modify the original aggmap

Remove the reference to the CUST dimension in the aggmap called by the Measure definition

```
describe kits_leaf_aggmap
  DEFINE KITS_CC_LEAF_AGGMAP AGGMAP
  AGGMAP
  RELATION this_aw!KIT_parentrel operator SUM precompute (NA)
  RELATION this_aw!SMNTH_parentrel operator SUM precompute (NA)
  RELATION this_aw!CHNL_parentrel operator SUM precompute (NA)
  RELATION this_aw!REGCTR_parentrel operator SUM precompute (NA)
  END
```

New Results:

Measure	KITS_LEAF	KITS_CC_LEAF	KITS_ALL	KITS_CC_ALL	KITS_CC_LEAF (Partial Agg)
Run Time (seconds)	52	31	0	0	1
Measure Cells	34,516,476	34,516,476	797,023,764	134,327,772	37,537,002
Measure Size (MB)	272	272	6,290	1,060	296
Composite Tuples	1,917,582	1,917,582	44,279,098	7,461,845	2,084,471
Composite Size (MB)	19	19	867	73	21

This is more like it! This time was verified by by stopping and restarting the Oracle database to clear any caching, and running the query as the very first command.

Using OLAP Data for Applications

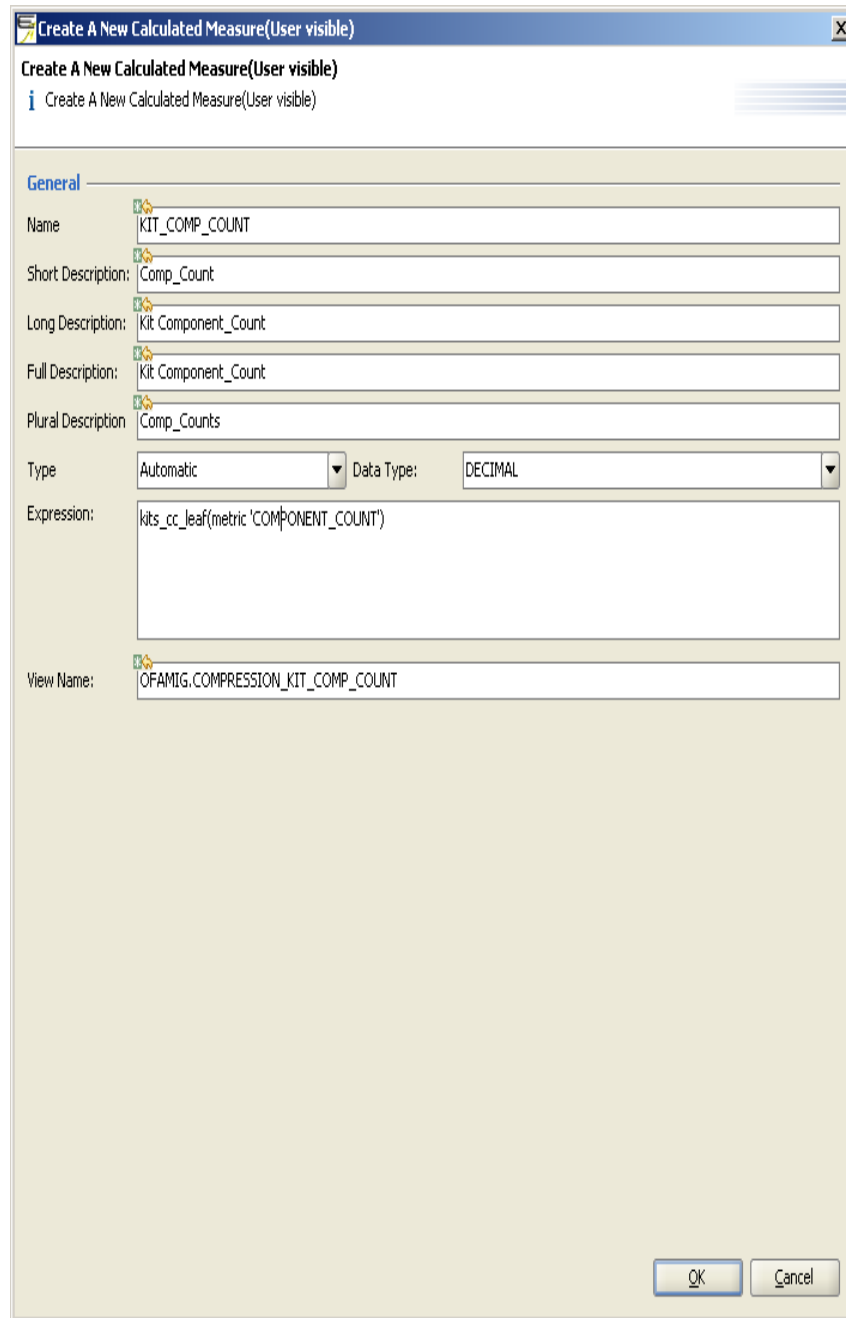
Now that we can quickly calculate results, we can make this available to all applications, tools, users, etc who can access Oracle. We do this via the built-in SQL function, `olap_table`.

Since we want to make the OLAP array look like a table, we will create measures corresponding to the columns, add these into a Cube, and create a SQL view on the Cube.

Column Measures

Since the "METRIC" dimension corresponds to the relational fact table columns, we will create a measure for each dimension member. Each measure will represent a slice just for one member, so it will appear as a single column of data. Note however, we are not restricted to this:

- Any result along any dimension (or combination of dimensions) can be presented as a view column.
- These "new" columns can be calculations such as variances between different arrays, such as budget to actual.
- View columns can be the result of running models, functions (over 250 built-in, such as moving average), or custom programs/functions (the OLAP DML is a complete procedural 4GL).
- The base array does not even have to be in the view. The columns can all be calculations on the base array, or the base array plus other arrays.



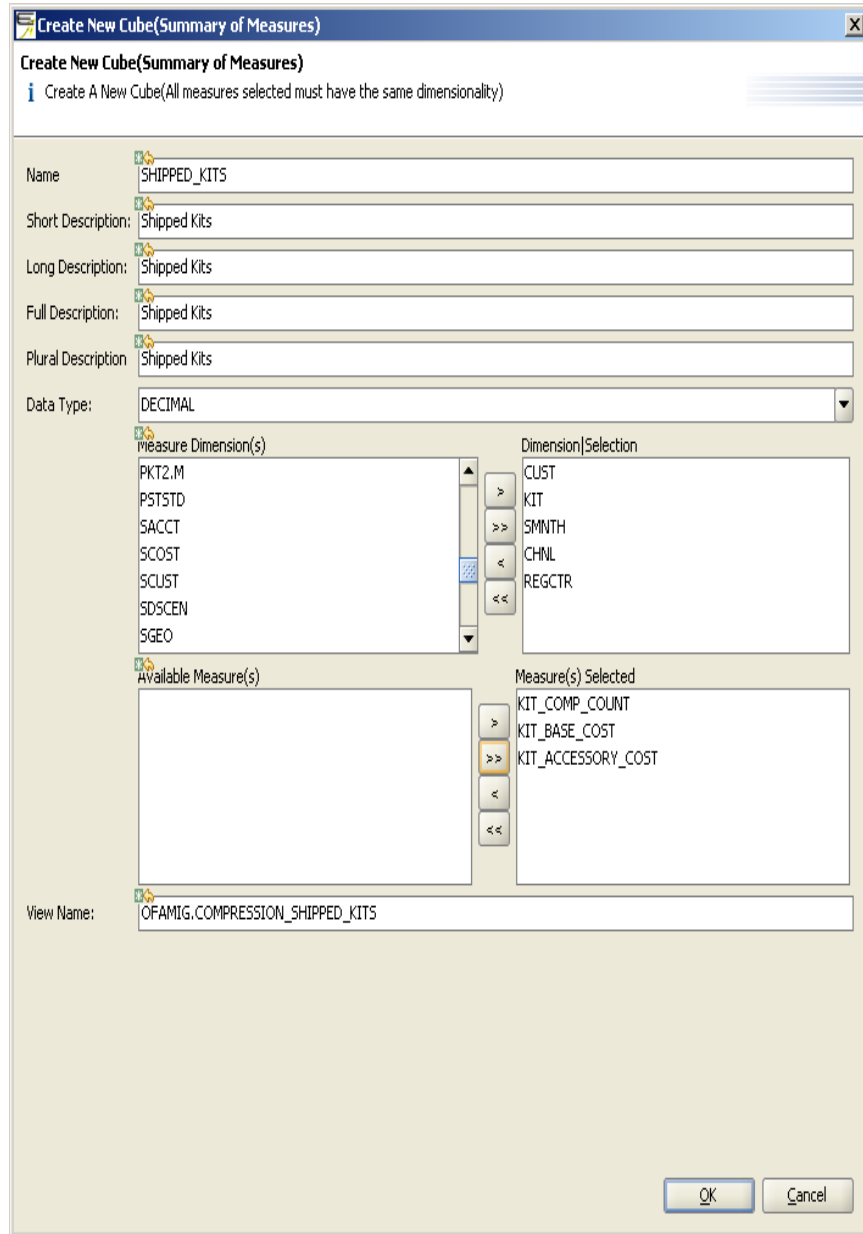
This creates a formula that will contain the slice of data for one column, in this case "COMPONENT_COUNT" which is the same as one of the columns in the relational fact table. The formula is 5 dimensional; since only one value of the METRIC dimension is needed, that dimension is not in the definition. The selection of "Automatic" as the Type means OLAP will figure out the dimensionality of the definition for you.

```
describe kit_comp_count
    DEFINE KIT_COMP_COUNT FORMULA DECIMAL <CUST KIT SMNTH CHNL REGCTR>
    EQ kits_cc_leaf(metric 'COMPONENT_COUNT')
```

This is repeated for the remaining columns we want. For this demo, we added two more, not all 18.

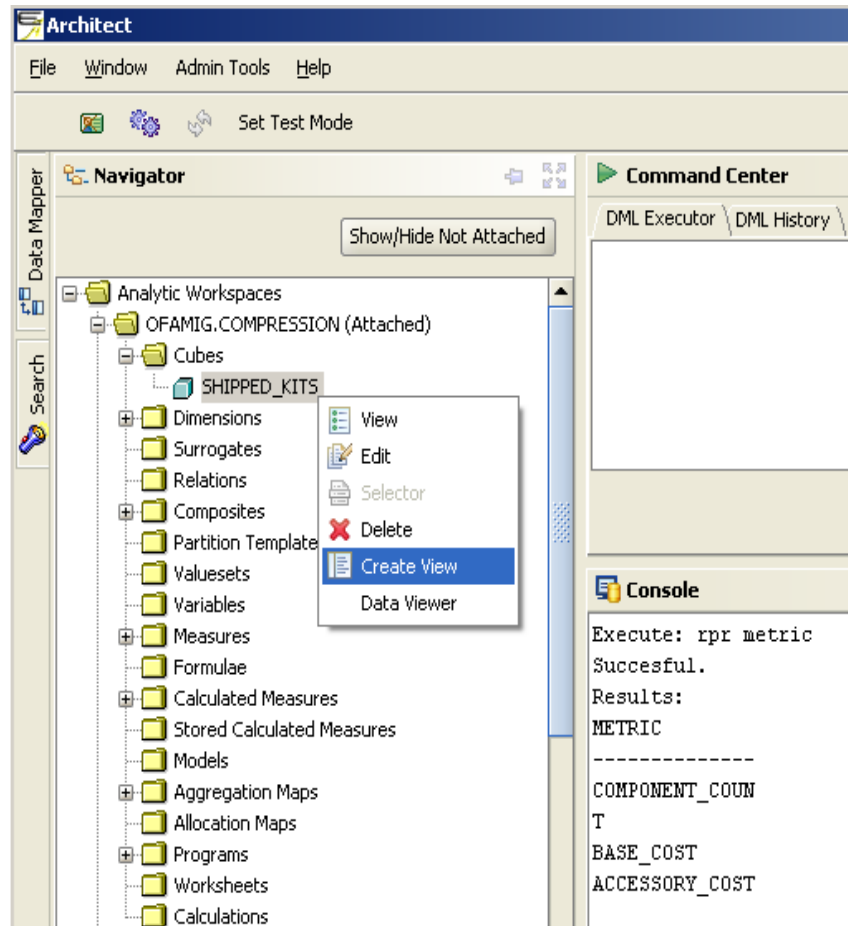
Cube

The next step is to create a cube to hold these measures.



SQL View

Create the view by right clicking on the cube name in the navigator.



The view that is created can be used like any other MV selection query, and can be joined to existing or new views using standard SQL where clauses, etc. The only difference is that there is never a need to actually build an MV since the run time is so fast over the entire scope of the view.

The SQL view is shown on the next page.

```
create or replace view OFAMIG.COMPRESSION_SHIPPED_KITS_v as
select *
from table(OLAP_TABLE('OFAMIG.COMPRESSION duration session',
'',
'aw attach EAARCHITECT.ea_code ro onattach onattach',
'&(amp;OFAMIG.COMPRESSION!EARCH.LIMIT_MAP_CATALOG( OFAMIG.COMPRESSION!
EARCH.all_limit_maps 'SHIPPED_KITS_CUBE_LIMIT_MAP' ) )')
))
MODEL
DIMENSION BY(
  C_CUST,
  C_KIT,
  C_SMNTH,
  C_CHNL,
  C_REGCTR
)
MEASURES(
  M_KIT_COMP_COUNT,
  M_KIT_BASE_COST,
  M_KIT_ACCESSORY_COST
)
RULES UPDATE SEQUENTIAL ORDER();
```

APEX Example

The generated view can be copied and pasted into APEX and adjusted for local combo boxes.

```
select
  C_CUST,
  C_KIT,
  C_SMNTH,
  C_CHNL,
  C_REGCTR,
  M_KIT_COMP_COUNT,
  trunc(M_KIT_BASE_COST, 2),
  trunc(M_KIT_ACCESSORY_COST, 2)
from OFAMIG.COMPRESSION_SHIPPED_KITS_v
where C_CUST = nvl(:P1_CUST, 'TOT.CUST')
-- and C_KIT = nvl(:P1_KIT, 'TOT.KITS')
  and C_SMNTH = nvl(:P1_SMNTH, 'Y2009')
  and C_CHNL = nvl(:P1_CHNL, 'TOT.CHNL')
  and C_REGCTR = nvl(:P1_REGCTR, 'TOT.REGCTR')
```

SQL Developer Example

The generated view can be popped into SQL Developer and tested using where clauses to limit the scope of the return. Don't leave out the where clauses – a select (*) will return millions of rows. This view can obviously be joined with other views for Discoverer reporting or any facility or product that can use SQL views.

```
select
  C_CUST,
  C_KIT,
  C_SMNTH,
  C_CHNL,
  C_REGCTR,
  M_KIT_COMP_COUNT,
  M_KIT_BASE_COST,
  M_KIT_ACCESSORY_COST
from OFAMIG.COMPRESSION_SHIPPED_KITS_v
where C_CUST = 'TOT.CUST'
      -- and C_KIT = 'TOT.KITS'
      and C_SMNTH = 'Y2009'
      and C_CHNL = 'TOT.CHNL'
      and C_REGCTR = 'TOT.REGCTR' ;
```

Environment

Server – 2 hyper-threaded (i.e., before dual core) xeon processors @ 3GHz, 8 GB ECC ram, SATA disc drives;
Centos 5.4 Linux

Oracle – 11gR2 Linux 64 bit

Client – Windows XP SP3 desktop

Notes

1. The OLAP object definitions and commands shown are what is actually fed to the data dictionary. They can be entered via an application, or a tool that emulates a command line, such as AWM, OX or OLAP Worksheet, or even SQL or SQL PLUS using EXEC DBMS_AW.EXECUTE(). And of course, Escendo Architect.
2. There are parameters for AGGMAP beyond what is shown above. Refer to the Oracle documentation to see further functionality.
3. The original application was an OFA-Web implementation converted to OLAP. It actually used MCALC on one of the dimensions to keep the composite to 22 million tuples. For legacy Express applications, keeping MCALC is definitely a viable option, especially since it may be embedded in many programs and formulas. All that is needed is to modify the MCALC call for meta-data name changes.