

Paper 431-2012

10,000 Leagues of Data... Divide and Conquer OLAP Cubes: Best Practices for High Volumes of Data

Stephen Overton, Zencos Consulting, Cary, NC, USA

ABSTRACT

High volumes of data can be summarized most efficiently using OLAP technology. As volumes of data approach the millions and even billions of rows of data, loading an OLAP cube can become extremely challenging due to system constraints. This paper presents a technique that uses SAS® macro programming to divide and load data incrementally into an OLAP cube. Using this technique ensures the cube will build successfully but also provides users access to the cube while it is being loaded. This paper also highlights OLAP programming techniques that allow you to streamline the cube building process and minimize the impact of errors.

CONCEPT

The amount of data that organizations must be able to analyze in an ever growing competitive market continues to grow larger and larger. This change in volume creates constraints for loading data into an OLAP cube as the size and structure of source data grows. Build times run longer as the data volume grows. As more dimensions and measures are defined in the OLAP cube, the build time grows even longer because more data has to be analyzed to build the NWAY aggregation. This is because the NWAY aggregation is the summarization of all measures by the grouping of each level defined in the OLAP cube. One way to circumvent this lengthy build process with high volumes of data is to avoid building the NWAY aggregation and define a relational OLAP cube, also known as ROLAP technology. When large volumes of data are queried, the end user's wait time increases as the data is summarized at run time.

This paper presents a technique that loads large volumes of data into an OLAP cube by dividing the source data into segments and incrementally loading the OLAP cube in place. This technique uses the default NWAY aggregation which avoids the shortcomings of ROLAP. The reader of this paper should understand OLAP technology and how to build an OLAP cube using SAS programming. The conceptual process is shown in Figure 1.

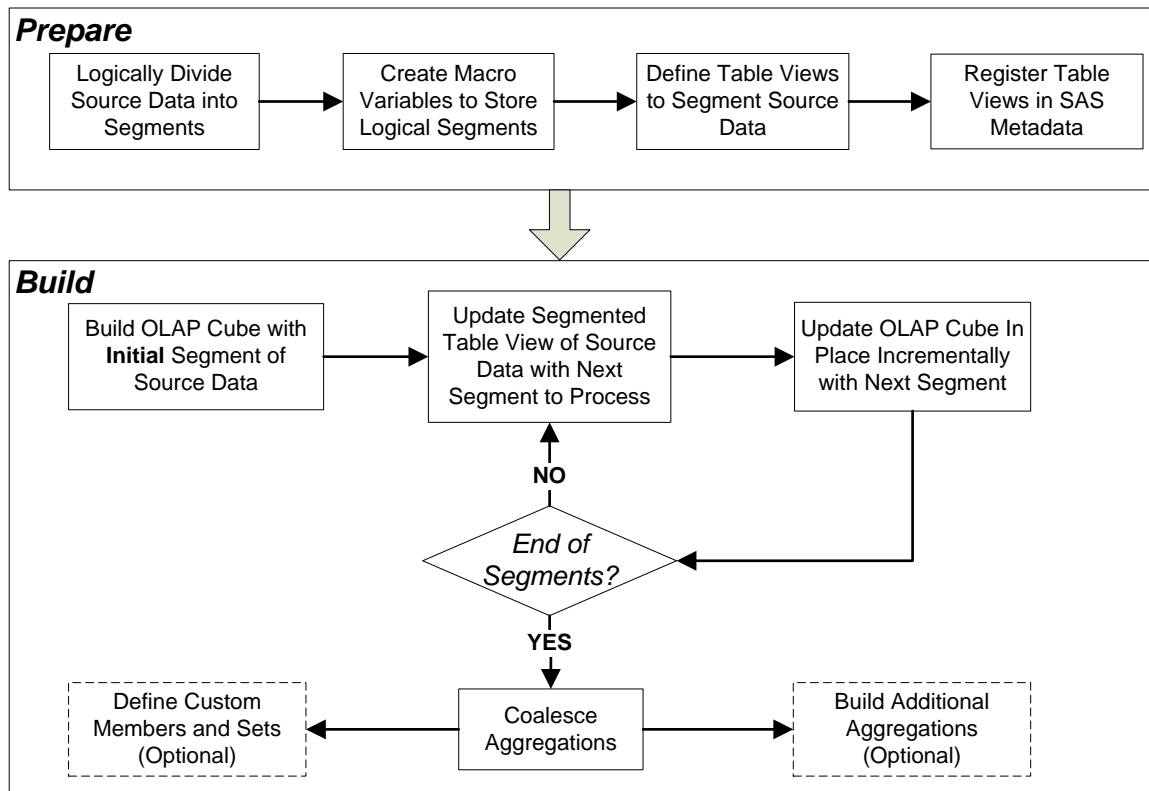


Figure 1. Conceptual Process to Divide and Conquer OLAP Cubes

10,000 Leagues of Data... Divide and Conquer OLAP Cubes: Best Practices for High Volumes of Data, continued

LOGICALLY DIVIDE SOURCE DATA INTO SEGMENTS

To start the process, source data needs to be logically divided into segments prior to any OLAP processing. This can vary for any source of data and should be carefully planned. The volume of data and build time for each segment should determine how logical segments are defined. Source data is not physically divided in this step.

This paper uses sample candy sales data from the SAS® Enterprise Guide sample library to demonstrate the process. The code used to build this sample data as well as demonstrate the segmented OLAP cube build process is referenced at the bottom of this paper. The following PROC SQL step demonstrates how the year of the sale is used as a segment.

```
proc sql;
  /* Years are used as segments to load cube. Get from fact table. */
  create table all_segments as
    select distinct year(date) as year from SGF2012.FACT_CANDY_SALES;
  /* select the max year to load initially */
  create table initial_segment as
    select * from all_segments where year =
      (select max(year) from all_segments) order by year;
  create table remaining_segments as
    select * from all_segments where year ^=
      (select max(year) from all_segments) order by year desc;
quit;
```

In this sample, the initial segment loaded in the OLAP cube is the latest year of source data. The remaining segments are sorted descending order. For OLAP cubes with lengthy build times, it may be beneficial to start loading the OLAP cube with the latest data first, then work backwards to make more recent data available faster.

CREATE MACRO VARIABLES TO STORE LOGICAL SEGMENTS

Segments are loaded into macro variables which will be used later in the build process.

```
proc sql;
  /* define initial segment */
  select year into :initial_segment from initial_segment;
  /* insert segments into macro list to iterate through */
  select year into :segment1 - :segment99999 from remaining_segments;
  /* get the number of records for a counter for loop */
  select count(year) into :segment_count from remaining_segments;
quit;
```

DEFINE TABLE VIEWS TO DIVIDE SOURCE DATA

Table views are used to physically divide the source data into segments. Views are easy to create and update but will need to be registered in SAS metadata. One key advantage of using a table view is that data is not physically replicated and the table view can be easily changed to point to a new data segment.

Two table views are defined in the SAS code below. The SGF2012.FACT_CANDY_SALES_INITIAL_VIEW table view will not change throughout the entire process and will only be used in the initial build of the OLAP cube. The SGF2012.FACT_CANDY_SALES_SEGMENT_VIEW table view will be updated to filter for each segment of data as the OLAP cube is built. The segmented table view is initialized using the first of the remaining segments to load.

```
proc sql;
  create view SGF2012.FACT_CANDY_SALES_INITIAL_VIEW as
    select * from SGF2012.FACT_CANDY_SALES
    where year(date) = &initial_segment;
  create view SGF2012.FACT_CANDY_SALES_SEGMENT_VIEW as
    select * from SGF2012.FACT_CANDY_SALES
    where year(date) = &segment1;
quit;
```

Important Tip: Index the column used to segment the source data for better performance. In the example used in this paper, this would be the "DATE" column from the SGF2012.FACT_CANDY_SALES table.

REGISTER TABLE VIEWS IN SAS METADATA

The following code registers the two table views in the SAS metadata. This would be the same as updating or

10,000 Leagues of Data... Divide and Conquer OLAP Cubes: Best Practices for High Volumes of Data, continued

registering tables through SAS® Management Console.

```
libname _temp meta rename="Foundation" library="SGF2012" metaout=data;
proc metalib;
  omr (library="SGF2012"
      metarepository="Foundation");
  select ("FACT_CANDY_SALES_INITIAL_VIEW");
  update_rule=(delete);
  report;
run;
proc metalib;
  omr (library="SGF2012"
      metarepository="Foundation");
  select ("FACT_CANDY_SALES_SEGMENT_VIEW");
  update_rule=(delete);
  report;
run;
libname _temp clear;
```

BUILD OLAP CUBE WITH INITIAL SEGMENT OF SOURCE DATA

Once the source data is segmented, the OLAP cube build process can begin. Not only is the source data segmented, the OLAP cube code is also divided to build the cube incrementally, build aggregations, and define any additional members or sets if they exist – all in one SAS program. Separating the OLAP code into separate steps can help mitigate future problems. For example, if the syntax for an aggregation step is wrong, the OLAP cube will still build if the aggregation step is separated into its own PROC OLAP step.

The OLAP cube is deleted if it already exists.

```
proc olap
  CUBE          = "/Shared Data/SGF2012/Data/Candy Sales"
  DELETE;

  METASVR
  HOST          = "&SYSHOSTNAME"
  PORT         = 8561
  OLAP_SCHEMA  = "SASApp - OLAP Schema";

run;
```

The OLAP cube build process is started with only the initial view of source data. This is shown in Figure 2.

```
proc olap
  CUBE          = "/Shared Data/SGF2012/Data/Candy Sales"
  PATH         = 'C:\SAS Projects\SGF2012\data'
  DESCRIPTION  = 'Candy sales demo cube for SGF2012'
  FACT         = SGF2012.FACT_CANDY_SALES_INITIAL_VIEW;

  METASVR
  HOST          = "&SYSHOSTNAME"
  PORT         = 8561
  OLAP_SCHEMA  = "SASApp - OLAP Schema";

  Dimensions
  Measures

run;
```

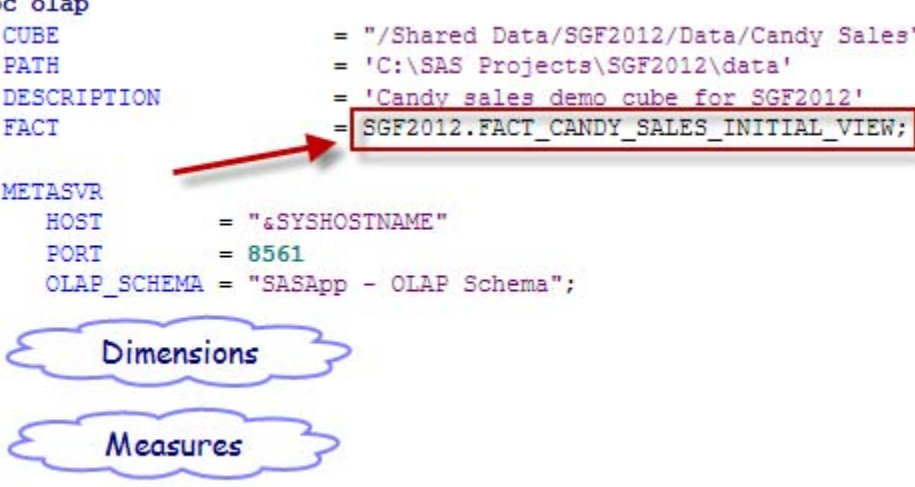


Figure 2. Initial Build of OLAP Cube using Initial Segment of Source Data

10,000 Leagues of Data... Divide and Conquer OLAP Cubes: Best Practices for High Volumes of Data, continued

LOAD REMAINING SEGMENTS WITH MACRO LOOP TO CONQUER THE OLAP CUBE

The following macro is used to iterate through the remaining segments of source data and incrementally load the OLAP cube. The two major pieces will be explained in the next two sections.

```

/** Macro to iterate through remaining segments and load cube */
%macro LoadSegmentsIntoCube;
/* incrementally load cube by updating a view in each step then incrementally adding
data to the cube */
%do x=1 %to &segment_count;
/* update view */
%put ***** UPDATE View to Incrementally load cube for segment = &&segment&x *****;
proc sql;
create view SGF2012.FACT_CANDY_SALES_SEGMENT_VIEW as
select * from SGF2012.FACT_CANDY_SALES
where year(date) = &&segment&x
;
quit;

/* Incrementally update cube */
%put ***** LOAD cube for snapshot = &&segment&x *****;
proc olap
CUBE = "&cube"
DATA = SGF2012.FACT_CANDY_SALES_SEGMENT_VIEW
ADD_DATA
UPDATE_INPLACE
IGNORE_MISSING_DIMKEYS = VERBOSE;

METASVR
HOST = "&SYSHOSTNAME"
PORT = 8561
OLAP_SCHEMA = "SASApp - OLAP Schema";

/** IMPORTANT: Update this list if new dimensions are added */
DIMENSION Products UPDATE_DIMENSION = MEMBERS;
DIMENSION Time UPDATE_DIMENSION = MEMBERS;
DIMENSION Customer UPDATE_DIMENSION = MEMBERS;
DIMENSION Region UPDATE_DIMENSION = MEMBERS;
run;

%put ***** FINISH loading for snapshot = &&segment&x *****;
%end;
%mend LoadSegmentsIntoCube;

/** Execute macro */
%LoadSegmentsIntoCube;

```

UPDATE SEGMENTED TABLE VIEW OF SOURCE DATA

The macro loops through the remaining segments defined above in macro variables segment1-segmentX, X representing the number of remaining segments defined above. This is shown in Figure 3.

```

%do x=1 %to &segment_count;
/* update view */
%put ***** UPDATE View to Incrementally load cube for segment = &&segment&x *****;
proc sql;
create view SGF2012.FACT_CANDY_SALES_SEGMENT_VIEW as
select * from SGF2012.FACT_CANDY_SALES
where year(date) = &&segment&x
;
quit;

```

Figure 3. Update Segmented Table View of Source Data

10,000 Leagues of Data... Divide and Conquer OLAP Cubes: Best Practices for High Volumes of Data, continued

UPDATE CUBE INCREMENTALLY WITH SEGMENTED TABLE VIEW OF SOURCE DATA

After the segmented table view is updated with the next segment of source data to load, the OLAP cube is incrementally updated. The dimension parameters listed in the incremental OLAP update process need to be updated whenever structural changes are made to the OLAP cube. If the initial OLAP cube build is edited with additional dimensions, this incremental step may fail if it is not updated. Documentation on incrementally updating an OLAP cube can be found online at SAS Support which is provided below in the References section.

COALESCE NWAY AGGREGATIONS

The final step is to coalesce any additional aggregation racks, which are produced by the incremental update process. Incrementally updating an OLAP cube produces incremental aggregation data which is written to a new SPDE table, also known as a rack. Each time a cube is incrementally updated, a new rack is produced. OLAP query response can slow because additional racks are accessed. More information on coalescing OLAP aggregations can be found in the References section at the end of this paper. Coalescing an OLAP cube is done using the following code.

```
proc olap
  CUBE           = "/Shared Data/SGF2012/Data/Candy Sales"
  COALESCE_AGGREGATIONS;

  METASVR
    HOST         = "&SYSHOSTNAME"
    PORT         = 8561
    OLAP_SCHEMA = "SASApp - OLAP Schema";
run;
```

BUILD ADDITIONAL AGGREGATIONS FOR PERFORMANCE (OPTIONAL)

Additional aggregations should be defined after the entire OLAP cube is built. If a small number of aggregations are needed for performance during the incremental load process, they can be defined after the initial load but will generate additional aggregation racks which will need to be coalesced as described above.

DEFINE CUSTOM MEMBERS AND SETS (OPTIONAL)

If any custom members or sets need to be defined, they should be done after the entire OLAP cube is completely finished loading. This will prevent MDX errors due to a lack of data defined in the OLAP cube. For example, if certain members of a dimension are not available until certain segments of the cube are loaded, defining the custom members or sets will fail because respective members have not been discovered yet.

NEXT STEP, UPDATING THE OLAP CUBE MOVING FORWARD

After all historical source data has been summarized in an OLAP cube during the initial build, the next logical step would be loading incremental data as it grows over time. This can be accomplished using a separate process which runs at the same update frequency of the source data to load changes. A similar approach can be used to segment new data using a table view which selects the latest segment inserted by the prior ETL process.

CONCLUSION

OLAP cubes are very effective for high performance reporting and analytics but can be a burden to build if the source data has an enormous number of records. As data volumes grow, the OLAP cube will spend more time building the NWAY aggregation and defining dimensions which can impact user accessibility. The more dimensions an OLAP cube has, the longer the NWAY takes to build due to more crossings of data. In some cases, just building the physical NWAY aggregation can fail because the data storage runs out of temporary space. The technique presented in this paper avoids physical system constraints by dividing source data into smaller segments which are easily summarized in the OLAP cube.

REFERENCES

- SAS code which builds sample data and demonstrates the OLAP build process described in this paper is available at <http://www.stephenoverton.net/SASCode/SGF2012/>
- SAS Support. "Coalescing Cube Aggregations." March 18, 2012. Available at <http://support.sas.com/documentation/cdl/en/olapug/59574/HTML/default/viewer.htm#a003241634.htm>.

10,000 Leagues of Data... Divide and Conquer OLAP Cubes: Best Practices for High Volumes of Data, continued

- SAS Support. "Updating a Cube In-Place." March 18, 2012. Available at <http://support.sas.com/documentation/cdl/en/olapug/59574/HTML/default/viewer.htm#a003241629.htm>.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Stephen Overton

Zencos Consulting (<http://www.zencos.com>)

Work Email: soverton@zencos.com

Personal Email: stephen.overton@gmail.com

Personal Website: <http://www.stephenoverton.net>

LinkedIn: <http://www.linkedin.com/in/overton>

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.