Paper 278-28

# Using a HOLAP Solution to Analyze Large Volumes of Data via the Web

Chris Peterson, Valero Energy Corporation, San Antonio, TX
Stuart B. Levine, SAS Institute, Rockville, MD

## ABSTRACT

This paper focuses on the challenges met by implementing a HOLAP solution on large amounts of data on an AIX platform using SAS MDDB Report Viewer. The base tables have approximately 310 million records. The OLAP solution contains approximately 200 GB of data. Query response time is approximately 20-40 seconds per average query.

## INTRODUCTION

Last year, Valero Energy Corporation was looking to improve both the build and query performance of their sales velocity reporting system. This system allows users the option to review sales information by levels as granular as UPC and Store for as long as 3 years back from the current date. Valero turned to The SAS System to provide an application that gave users the query performance they were looking for, as well as the ability to generate the data each week in an efficient manner. This paper will describe the process that was followed to provide that system as well as discussing the issues involved in maintaining the system for the past 18 months.

## THE BUSINESS NEED

One of Valero's business lines is a chain of over 1700 convenience stores throughout the Midwest and Southwest US. The sales and marketing associates want information on how items in their respective product categories or geographic areas are selling so business decisions can be made concerning these items. The requirement was to have a rolling 3 years of weekly data available to the system with data down to the detail level of item numbers (UPC) and store.

## WHY RE-DEVELOP?

Valero had 3 main issues with their previous reporting system:

- Time to generate data each week;
- Time to install/set up a new user;
- Time to retrieve data

These issues and how SAS addressed them follows:

### TIME TO GENERATE DATA:

Valero's data warehouse is updated weekly. The primary table in the warehouse contains a record for each individual item scanned in a store, representing hundreds of millions of rows in an Oracle database. Extracting the data for the most recently completed fiscal week, along with creating the summarized data for reporting, was a process that required 3 ½ - 4 hours. Multiply that by the need to extract and summarize data for the previous 4 weeks due to the possibility of adjustments, and the weekly ETL process became a 17-20 hour process. As the business grew, this would become a larger problem, as there would be more data each week as time went on.

Using the power and capabilities of SAS and Oracle, the process was rewritten to extract the data and create the summarized data stores in only 15-20 minutes. This was accomplished by using a data-driven approach and the SAS macro language to determine the correct dates for the extraction then execute a stored Oracle procedure from SAS to perform the extraction:

```
proc sql;
   connect to oracle(user=logonid
                password=passwrd
                path='server.database');
   execute (execute
         sas_weekly_extract_sp(&one_date))
         by oracle;
   disconnect from oracle;
quit;
```

This query is executed twice, once for the current week's data and a second time to extract the same week's data for the previous year, as part of the reporting system includes comparing information from one year to the next. Once these two queries are finished and the results are combined, a SAS MDDB is created for reporting.

### TIME TO INSTALL/SET UP A NEW USER:

The old system was a fat-client application. To set up a new user of the system required approximately ½ day to install the software, make connections to the data, and perform testing to make sure the system was working properly.

The SAS solution was much easier to install for a new user, requiring only a browser, a network connection, and a URL. The reporting mechanism was the SAS/IntrNet Multidimensional Report Viewer (MRV), the web-based OLAP viewer included as part of the SAS/IntrNet software.

### TIME TO RETRIEVE DATA:

Valero's previous reporting application was a client/server system; the data resided on a UNIX server and the viewer was on the Windows platform. This, combined with the software and the architecture of the data, resulted in queries taking from 1 to 2 minutes at the higher levels, as much as 5 minutes or more when drilling to the lower levels of the hierarchies.

By structuring the SAS MDDBs with subtables to support the users' requests and optimizing the metadata to target only the data sources needed to satisfy the request, query time was reduced to less than 20 seconds for most queries and 1-2 minutes when data was accessed at the lowest levels.

## THE DEVELOPMENT PROCESS

### DEFINING THE WORK

The first step in the development process was not coding but talking and, even more importantly, listening. It had already been determined that a single MDDB could not be built because of the cardinality and inter-relationships of three of the variables involved. There were approximately 1700 store numbers, over 30,000 UPC numbers, and 156 dates (each Wednesday, the end of the reporting week, for 3 years). Conversations were held with the administrator for the previous system, the users, and a DBA. Based on those, along with the density/sparsity of the data, it was determined that if the NWAY table for an MDDB could be constructed, it would contain approximately 450 million cells. Taking the length of these variables into account, this would have required between ¼ and 1 full *terabyte* of memory!

It was therefore obvious that the data had to be built in pieces, but the question remained: How to break it up? Again, after talking to various people within the organization about how the data is updated in the warehouse, how it is summarized, and how it is accessed, the first logical option was to create separate MDDBs for each week and link them through the SAS HOLAP functionality. The data warehouse is updated weekly, users look at the information by week, and they choose a range of weeks to look at, typically the most recent 6-12 weeks.

Tests were performed first to determine if a single week's cube could be built. Once that was successful, the next step was to build a few more weeks of cubes, begin defining the HOLAP Data Group, and benchmark the data access performance. Then, as that test was successful and the performance satisfactory, the subsequent steps were to keep increasing the amount of data being defined to the HOLAP Data Group and being accessed by the viewer. Ultimately, once 39 weeks of data had been built and successfully accessed through the viewer, everyone involved agreed that this was the correct solution.

Once it was agreed that this was a viable solution, now it to be made production-ready. This meant creating an ETL process that could be executed each week to generate the latest MDDBs in a scheduled production environment. A few things were known about the data and the process that had to be incorporated:

- Because adjustments could be made to the warehouse data as far as 4 weeks prior to the current week, data had to be extracted and rebuilt for a five-week period.
- There could be changes as frequently as weekly to the geographic and product organizations that had to be reflected in the data.
- There were dimension tables in the Oracle database that contained the organizational information and that were updated each week as part of the overall warehouse ETL process.
- The data extracted from Oracle into SAS only contained codes for variables such as Region, Product Category, etc., meaning these codes had to be formatted to display meaningful information to the users.

## EXTRACTING THE DATA

The ETL program for this system was written to include steps that read the dimension tables for the geographic and product data into SAS data sets, renaming and creating variables to allow these data sets to be used as the input for generating formats:

```
proc format library=formats
            cntlin=work.compfmt;
run;
```

These formats were then applied to the appropriate variables when the data was extracted from the Oracle warehouse.

Once all of the formats and other needed parameter files were created, the data had to be aged. For reasons described later in this paper, there is a separate base table for each of the 156 MDDBs in this system. They are numbered 1-156, rather then date-stamping them. The AGE statement in the DATASETS procedure offered the simplest method to rename all of the data sets:

```
proc datasets lib=retaildm;
   age velocity_base_table_1
       velocity_base_table_2
       velocity_base_table_3
       velocity_base_table_4…;
run;
```

The next step was to extract the data from Oracle. This was performed by executing the SQL procedure shown earlier in this paper within a %DO loop set to run 5 times, once for the most recent week and then for the previous 4 weeks to capture any adjustments made to the data.

The SQL queries and a couple of other necessary steps resulted in new base tables for the 5 latest MDDBs. However, more than just 5 MDDBs had to be generated. As mentioned earlier, there are separate base tables for each of the 156 cubes in this system. This is because Valero needs to be able to compare 'apples to apples'; that is, if a store is in a certain geographic area, they need to look at that store as if it had been in that area for all of the past 3 years, even though it may have just been reorganized into that area very recently. That way, sales can properly be compared over time for all levels of the geographic hierarchy. So, as the data in an MDDB cannot be modified, they have to be rebuilt every week. To make this process easier, each week's data is stored separately, but without the formats applied to the codes representing the store and UPC numbers that define the higher levels of those hierarchies. The formats generated each week from the dimension tables are applied to these data sets in views, then the views are the sources for the MDDB procedure calls:

```
data work.velocity_base_table /
        view=work.velocity_base_table;
   set sasdl.velocity_base_table_&i;
   format comp s_comp.;
run;

proc mddb data=work.velocity_base_table
          out=retaildm.velocity_mddb_&I
          label="Velocity Item Cube &I";
   *(more code);
run;
```

## MULTIDIMENSIONAL DATA AND METADATA

One last consideration was how long it would take to build all 156 MDDBs every week. Even though each one only requires about 4 minutes to build, over 150 of them would still take over 10 hours, and the production window wasn't that long. So, the last step was to take advantage of the multiple processors on the server. The UNIX machine had 4 CPUs, therefore the code was written to build the MDDBs in 4 separate processes. The code above was not executed as it appears here; rather, it was included in PUT statements within a DO loop and output to .sas programs. SAS commands were then generated and executed within other loops, sending each of the 4 individual programs to different processors, allowing 4 MDDBs to be built at a time. By having separate data sets for each week's base tables, there was no contention reading the data. And, with 4 separate jobs creating 39 MDDBs each, the cube-building part of the ETL process completes in less than 3 hours.

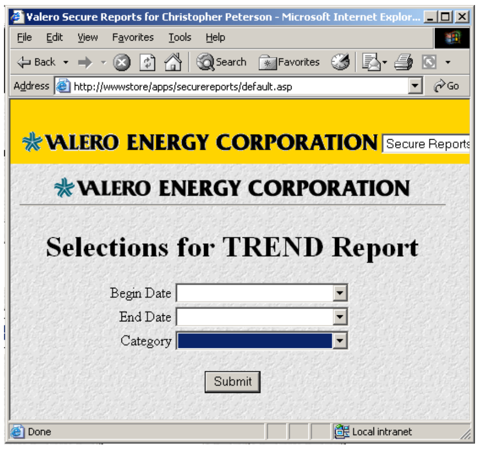
The last step of the process takes care of updating the metadata for the HOLAP Data Group. The initial release of the application did not include this step, but it was found that by adding some information to the metadata, the access times were improved. However, this optimization would require updating the metadata each week as new data was added to the system.

Even though each MDDB contains data for only 1 week, the application requires that the user choose a range of weeks to look at, and this range is generally only 6-12 weeks, the HOLAP Data Provider still was trying to extract data from each of the 156 cubes. By adding information to the HOLAP metadata about which week's data was in which MDDB, the data provider now only reads the cubes that contain data for the weeks that are requested. However, this information changes every week. For example, at the time of this writing, MDDB_1 contains data for week ending November 13 (Valero's week ends on Wednesday night). By the end of the following week, MDDB_1 will contain data for the week ending November 20. If the metadata is not updated with that new information and a user asks for data for November 13, the metadata will attempt to read from MDDB_1, and the data is really in MDDB_2, and nothing will be retrieved.

Using SAS Component Language (SCL), a program was written that opens the Repository classes, reads a macro variable containing the first week-ending date for the system (from 3 years ago), and rewrites the subsetting information in the metadata to match the actual date values in the MDDBs. This SCL entry is called at the end of the ETL program in a DM statement.

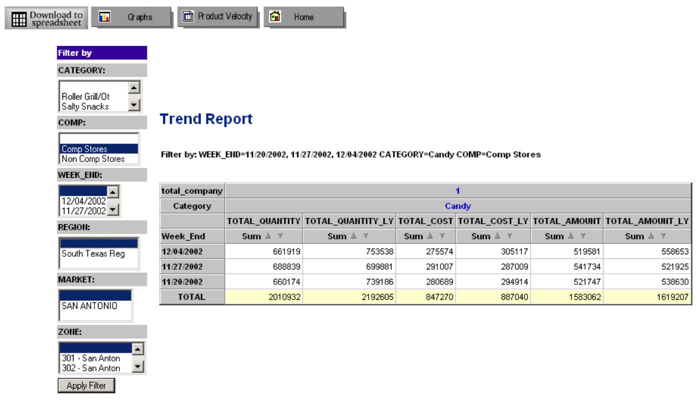
## REPORTING

Three different reports were initially envisioned to utilize the data. The MDDB Report Viewer (MRV) was chosen to be the reporting tool for 2 of the reports. A customized front end was necessary to focus user choices. Below is an example of one of those front ends.


Valero Secure Reports for Christopher Peterson - Microsoft Internet Explor...
File Edit View Favorites Tools Help
Back Search Favorites
Address http://wwwstore/apps/securereports/default.asp Go
VALERO ENERGY CORPORATION
Secure Reports
VALERO ENERGY CORPORATION
Selections for TREND Report
Begin Date
End Date
Category
Submit
Done
Local intranet

All of the parameters on the screen above are required. The Begin Date and End Date values are passed first to the metadata to select which weeks of data to access, identifying which of the 156 MDDBs to extract data from. For example, suppose a user chooses a Begin Date of 10/2/2002 and an End Date of 11/13/02. Not only will he see data for just that 7-week period, but only those 7 MDDBs will be accessed by the system due to the HOLAP metadata. Without having the date information in the metadata, the system would attempt to extract data from all 156 MDDBs and only succeed when reading those 7 cubes, but spend the time trying to extract data from the other 149 files.

Once through this initial screen the MRV is displayed with the appropriate options.

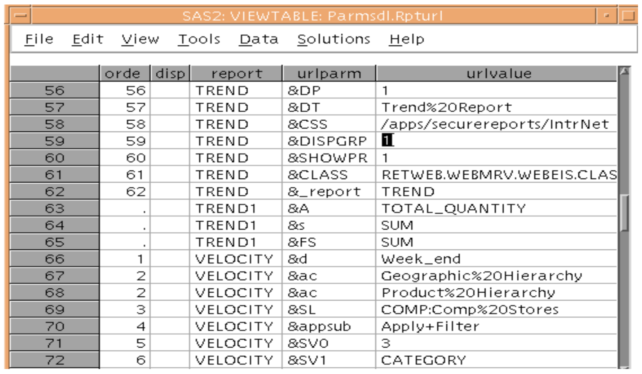


Trend Report

Filter by: WEEK_END=11/20/2002, 11/27/2002, 12/04/2002 CATEGORY=Candy COMP=Comp Stores

| total_company | 1 | | | | | |
|---|---|---|---|---|---|---|
| Category | Candy | | | | | |
| | TOTAL_QUANTITY | TOTAL_QUANTITY_LY | TOTAL_COST | TOTAL_COST_LY | TOTAL_AMOUNT | TOTAL_AMOUNT_LY |
| Week_End | Sum | Sum | Sum | Sum | Sum | Sum |
| 12/04/2002 | 661919 | 753538 | 275574 | 305117 | 519581 | 558653 |
| 11/27/2002 | 688839 | 699881 | 291007 | 287009 | 541734 | 521925 |
| 11/20/2002 | 660174 | 739186 | 280689 | 294914 | 521747 | 538630 |
| TOTAL | 2010932 | 2192605 | 847270 | 887040 | 1583062 | 1619207 |

The screen above is the customized MRV front end. The customizations are driven by observations read from SAS data sets by SCL code that is part of an override of the SASHELP.WEBEIS.WEBEIS.CLASS. The code snippet below is the SCL that reads a parameter data set, also pictured below, that contains the name-value pairs that become part of the dynamically-constructed URL for the report. Information in the data set include the dimensions to appear as rows and columns on the table, which variables should be displayed as filter variables, and which analysis variables and statistics should be on the report. This information is listed in the data set by report, so each of the reports in the system can have different settings. This allows for a new “Front End” to be added with no coding at all.


SAS2: VIEWTABLE: Parmsdl.Rpturl
File Edit View Tools Data Solutions Help

| | orde | disp | report | urlparm | urlvalue |
|---|---|---|---|---|---|
| 56 | 56 | | TREND | &DP | 1 |
| 57 | 57 | | TREND | &DT | Trend%20Report |
| 58 | 58 | | TREND | &CSS | /apps/securereports/IntrNet |
| 59 | 59 | | TREND | &DISPGRP | 1 |
| 60 | 60 | | TREND | &SHOWPR | 1 |
| 61 | 61 | | TREND | &CLASS | RETWEB.WEBMRV.WEBEIS.CLAS |
| 62 | 62 | | TREND | &_report | TREND |
| 63 | . | | TREND1 | &A | TOTAL_QUANTITY |
| 64 | . | | TREND1 | &s | SUM |
| 65 | . | | TREND1 | &FS | SUM |
| 66 | 1 | | VELOCITY | &d | Week_end |
| 67 | 2 | | VELOCITY | &ac | Geographic%20Hierarchy |
| 68 | 2 | | VELOCITY | &ac | Product%20Hierarchy |
| 69 | 3 | | VELOCITY | &SL | COMP:Comp%20Stores |
| 70 | 4 | | VELOCITY | &appsub | Apply+Filter |
| 71 | 5 | | VELOCITY | &SV0 | 3 |
| 72 | 6 | | VELOCITY | &SV1 | CATEGORY |

```
BUILD_URL_STRING:
/********************************************/
/* This sections builds that URL for the
selected report.                           */
/********************************************/

category = symget('category');

begdate = input(symget('begdate'),8.);
enddate = input(symget('enddate'),8.);

url_string=trim(left(url_string))||'_program=
SASHELP.WEBEIS.SHOWRPT.SCL&_service='!!symget
('_SERVICE');

url_string=trim(left(url_string))||'&_debug='
||trim(left(_debug));

/********************************************/
/* Since the user only selected a beginning
and an end date we have to read the base
calendar to add all the dates for each week
between the range selected.                */
/********************************************/

dsid=open('parmsdl.base_cal(where=('!!begdate
!!' le week_end le '!!enddate!!'))','I');

do while (fetch(dsid_Data) ^= -1);
     weekend_dt=put(getvarn(dsid,varnum(dsid,'
     WEEK_END')), mmddyy10.);

     url_string=trim(left(url_string))||
     '&SL=WEEK_END%3A'||urlencode(weekend_dt);
end;

rc = close(dsid_data);

link BUILD_LAYOUT_SELECTION;

/********************************************/
/* This opens and reads the default URL named
values pairs that have been defined        */
/********************************************/

dsid=open('parmsdl.rpturl(where=(upcase(repor
t)='!!upcase(quote(_report))!!'and display ne
```

```
   "N"))','I');

   do while (fetch(dsid) ^= -1);
        url_string=trim(left(url_string))||
        getvarc(dsid,varnum(dsid,'URLPARM'))!!'='
        !!urlencode(getvarc(dsid,varnum(dsid,'URL
        VALUE')));
   end;

   rc = close(dsid);

   return;


   BUILD_LAYOUT_SELECTION:
   /*******************************************/
   /* This opens and reads the default name
   value pairs for the selected report and gets
   the corresponding macro variables and sets
   them for the URL to generate the report based
   on the user selections from the layout
   page.\*/
   /*******************************************/

   if length(category) ge 1 then
        url_string=trim(left(url_string))||
        '&SL=CATEGORY%3A'||
        urlencode(trim(left(category)));

   return;
```

The third report utilizes a similar front end, but does not utilize the MRV. It pulls data from the Base Tables and uses PROC PRINT and PROC GCHART to produce the results. Over time these Base Tables have become the source for several SAS/INTRNET applications.

## CONCLUSION

The first, and maybe the most important, conclusion is that no application development effort will be successful without the first step, requirements gathering. Talking to representatives of all areas of the organization and gathering requirements for the system and information about the data and the hardware are key activities to the success of any project.

Also, designing the system properly is vital, as well as knowing how the software works. Large data such as Valero's can pose a challenge, but by designing the database and the programs to take advantage of the software and the environment, it can be dealt with. Both of these steps are critical and should be completed before the coding starts. Without them, what do you code?

## ACKNOWLEDGMENTS

Most of the code for the front-end customizations was written by members from Qualex Consulting. Without their contributions, this application would not have been successfully implemented.

## CONTACT INFORMATION

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

Chris Peterson
Valero Energy Corporation
6000 North Loop 1604 West
San Antonio TX, 78249-1112
Work Phone: 210-592-4787
Fax: 210-592-3452
Email: chris.peterson@valero.com

Stuart B. Levine
SAS Institute Inc.
111 Rockville Pike, Suite 1100
Rockville, MD 20850
Work Phone: 301-838-7030, ext. 3363
Fax: 301-838-7049
Email: Stuart.Levine@sas.com
Web: www.sas.com