Building Your Own SAS® Data Warehouse And Developing A Tool Set for Managing it

Di Meng & Charlie Bastnagel
Cigna Healthcare, Chattanooga TN

ABSTRACT

Although the SAS Data Engine is not natively a Relational Database, there are times when it is the best choice for a Data Warehouse Implimentation. If your Data Warehouse requires flexibility and the ability to change its shape and dimension automatically or your data is de-normalized, then the SAS platform can provide you with the tools to build a Data Warehouse that changes as your data changes.

SAS Data Warehouse Administrators (abbreviated as DWA) can develop their own Data Warehouse Administration Tools that fit their exact business situation and by making those tools table driven, your Data Warehouse can grow and change in directions limited only by foresight and imagination.

This paper will discuss the development of a Data Warehouse Administrator Toolset and focuses on the use of Client PC platforms like Windows and OS2 with the Unix platform playing the key Server role.

Some of the techniques demonstrated are:

- Creating your own driver for reading external data sources like Redbrick and Oracle when SQL Passthrough is too slow for your situation.
- Developing Interfaces that help the DW Admin manage his/her work.
- Using the Web to provide Information About the SAS DW to the User Community .
- Creating your own Performance Monitors that allow you to best use your Server resources.

INTRODUCTION

While there are many DW Administration Tools available in the commercial arena, our experience is that most of these tools force companies to change their business processes to fit their DW. If you have spent effort on crafting your business processes this reverse engineering may not be acceptible, and developing your own DW Administration Toolset with SAS Software may be the answer for you.

Remember that your Data Warehouse is designed to help you better understand your business and to make decisions about what direction it is going. In other words, the Data Warehouse is there to serve you and not vice-versa. In our experience, most people seem to be living in Vice-Versa.

What are some of the requirements in building and maintaining a SAS Data Warehouse Environment?:

- **Your data does not fit a relational model.**
- **A flexible and expandable Server platform that is not overwhelmed with hardware problems.**
- **A driver or other methodology for transferring the data from its original source into your SAS DW.**
- **A Data Warehouse Administration Tool to manage the Data Warehouse environment.**
- **Software tools that allow your users the ability to actually provide answers to your business problems.**

OUR ENVIRONMENT
How we fulfilled our DW requirements

A relational model?

Though the data in our SAS Data Warehouse still has relational dimensions, the limits of a RBDMS did not provide us with the flexibility we needed. The relational dimensions in our data were not the road block to completing our more difficult projects.

In our company, each SAS department has their own projects and has needed the flexibilty and freedom to run the data through often massive data transformations in order to produce a 3rd or 4th generation versions of the data related to their projects. Because our SAS Data Warehouse and the SAS user platform are on the same Unix Server we can provide the data and the platform for successfully completing complicated analytical projects without drowning our network in massive data transfers like the ODBC tools do.

The input source to our DW is the Redbrick Data Warehouse which is a RBDMS. Since the SAS Community was often fed the projects the ODBC Compliant tools could not complete, our SAS Data Warehouse was designed as a 2nd Generation Warehouse that allowed us to focus on the data needs for the more complicated projects and using Middleware Methodologies add important new information to the base data.

Our Server Platform

The SAS data warehouse resides on an IBM RS6000 running AIX 4.2. This Server has 600 gig of drive space and 4 Gig of RAM. Not only do the SAS users use this as their primary server platform but it also feeds our SAS Powered Web Applications and allows ODBC tools access to one of our 5 SAS Share Servers that run on the Server at all times.

In order to reduce data duplication and to feed our Web Applications we run an FTP Server on this machine. In addtion to this we are currently setting up an HTTP Proxy on this Server so that we can run our Web Applications on it. Why? Since our Web Applications are SAS table driven we have found that this will provide us with the best possible speed and put the least amount of stress on our network.

The SAS Data Warehouse is maintained in a client-server environment. The warehouse administration tool is running on the client (OS/2, MS Windows) side. We employ SAS version 6.12 on all platforms. We allow SAS Connect access as well as a 3rd tier batch software called SASBATCH which allows our users to send their programs in batch mode from their PC or Unix clients with a point and click interface.

The Redbrick Data Warehouse that feeds our Warehouse is running on the same type of IBM RS6000 as ours and their linked together with a FDDI ring. Redbrick allows access to the end-users through ODBC and Sybase Open Client. RISQL is its native SQL language and is the language we use to pull data from Redbrick..

Our user community consists of 100 SAS Analysts and 75 Web Users who only use SAS via our Web Applications. These 175 users are spread out across the country in various regional offices.

BUSINESS RULES AND DESIGN REQUIREMENTS

What were the driving factors behind the design of our warehouse? One important feature is that most of our SAS users were also Redbrick users and as such we wanted the SAS DW to mimic the Redbrick DW as much as possible.

In Redbrick There are 3 Databases.

- CLAIM - Tables related to Healthcare Claims.
- MEMBER - Tables realated to Healthcare Members.
- PROVIDER - Tables related to Healthcare Providers.

In SAS we accomplished the same structure using libnames. So for example the Arkansas Claim Tables were stored in a library named AR_CLMDW, Kentucky's Provider Tables were stored in KY_PRVDW etc....

What were some of the requirements for our SAS DW?

Speedy Performance:

Our table sizes range from 20 megabytes to 6 Gigabytes and that is SAS compressed (actual size is 45% greater). We have 30 hours in which to pull 100 gigabytes (600 tables) from Redbrick and then load, verify, and test the resulting SAS Tables.

What makes this tough is that we cannot pull our data until the Redbrick DW is refreshed. And while Redbrick has the entire weekend to be built, we must build our SAS DW starting Monday morning at around 5 am and then we activate the new SAS tables Monday night during off hours.

Because of the way Healthcare data is structured we must always completely re-build our warehouse. Therefore speed is of the utmost importance as one might imagine.

Flexibility

As in any business that is growing and changing, our warehouse goes through constant changes. Fields and Tables are constantly being added and changed. This management of change becomes in fact the primary challenge of the DWA.

Thus, we needed a Data Warehouse Tool that would make these constant changes manageable. Having to go in and edit SAS programs for each of the 600 tables would not suffice. A table driven strategy was required to allow the flexibility we needed.

As mentioned above we needed a table driven strategy to tie our SAS Data Warehouse to the Redbrick Data Warehouse so that when a change occurred in Redbrick it would automatically affect a change in the SAS Data Warehouse. Also, some of our Data Sources came from outside of Redbrick and therefore we needed a DW Tool that could accomodate various types of Data Sources.

More importantly not all of these Data Sources had SAS Access engines so this meant we needed the ability to create our own Data Source Adapters to read in the data. In other words, what do you do when there is no SQL Passthrough to the data file you are wanting to load into your warehouse?

Data Quality Control

One of the major benefits of data warehousing is that data from multiple and disparate operational systems can be brought into a data warehouse and cleaned up.

We promised our SAS users a clean, user-friendly, SAS-formatted, uniform means of viewing data. This required us to develop a methodology to validate data during the update processes based on business requirements and certain acceptible variations.

It seems that the bane of many RBDMS's is that they often get corrupted indices. In order to make sure that the tables in our DW made sense, we not only had to verify that we balanced to the penny with our data input source, Redbrick, but also that if we compared our SAS DW to the previous version of itself, that the variation in record counts and money made sense in our business terms.

Resource Limits

Here we are talking about the space requirement for database operation, data transportation, data processing, and temporary mirror sites used to perform data validation.

We are the owner of the SAS UNIX box, but we are just users on the Redbrick System and we like other users had limits we had to work within. There is only so much memory and only so much spill space and only so many processes could be active at one time.

In order to meet our need for speed, finding better and faster ways that helped us get around the resource limits was a primary concern.

METHODOLOGY

How did we meet the design requirements in building our SAS DW?

Data Extraction and System Adapters

To bring our SAS Data Warehouse to life, the first thing that we were faced with was how to get the Redbrick Tables and other Data Sources into the SAS system.

Since Redbrick is much like SYBASE and ORACLE, we experimented with SQL Passthru facility provided by the SAS/ACCESS product. Since, in this way, we were sharing the REDBRICK server resources with all the other users most of whom were using MS Access, the processes spent a great deal of time waiting on I/O. We were not able to meet our speed requirements using the SQL Passthru method. Because at the same time we were starting to build our SAS DW, Redbrick was also being hit by other groups as well.

What could we do? We discovered that Redbrick had an SQL language called RISQL that could be sent to it using plain text file scripts. This meant we could run our queries in the background on the Redbrick Server without maintaining an active network connection as all the ODBC and Sybase Open-Client Tools forced us to do.

This was not the end however, for there were several hurdles that we had to leap over:

- We would have to have RISQL queries for each table we wanted to pull.

- Since RISQL only generated text files we would have to transport the data from their UNIX to our SAS UNIX server on the fly. **Because the Redbrick Server had no extra spill space for us, this forced us to devise a very speedy solution that reduced our query time for a table that took 25 hours using SQL Passthru to under an hour using our new method.**

In short what we did was to use a FILENAME PIPE statement to run the Redbrick RISQL query from within a SAS Data Step. What this meant is that as Redbrick returned the results of the query, it would be written to the SAS Dataset immediately with no ancillary files being created on the way.

Here is an example block of code for piping the data from one server to another:

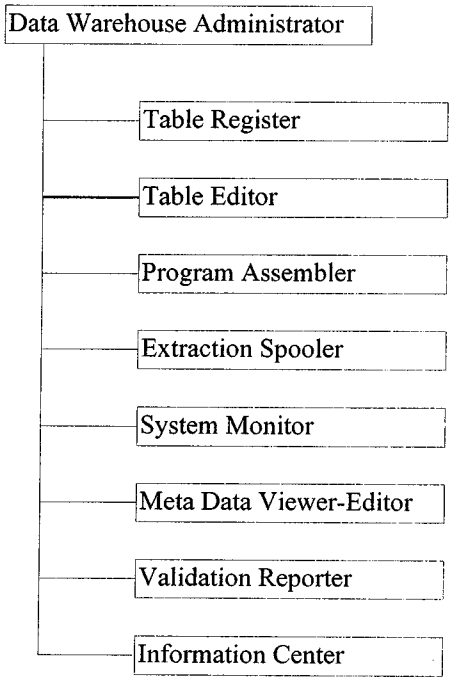
```
filename CLAIM PIPE
  "cat claim.sql  |
   remsh  ourserver risqlrpt -q -d oh_clmdw "
   lrecl=10000;

data oh_clmdw.claim;
   infile CLAIM;
   input statements;
run;
```

The Data Warehouse Administrator Interface

- The DWA interface was to serve two main functions: Table Register and Query Assembler. Through the interface the DWA would register new tables and SQL Adapter we developed would then write the appropriate SAS and RISQL Code required to make the SAS code above possible.

- The SQL Adapter is an SCL entry that updates the meta database, takes the requests from the DWA Assembler to generate the RISQL query, and generates a SAS program to load the the data.

In simplest terms our SQL queries are stored in meta-tables that drive the entire Data Warehousing Tool. Some of these tables are: DW_TABLE, DW_COLS and DW_LOAD and they hold the information about data warehouse tables, columns of tables and table updates.



As shown in the figure below the DWA Tool has 8 components.

They are functioning together to initiate, modify and maintain our SAS data warehouse. The interface itself is a one AF Frame application with each component showing up as a Page build on the fly using what we call flying objects. . Because we implement the application under multiple O/Ss and screen resolutions, static frames make testing and modification become a nightmare, especially when you try to add or change any widget. We decided to employ one blank frame as our canvas and to create all the widgets on the fly, and then organize them into individual pages to function like a frame.

How to create a FLYING WIDGET was discussed in detail in Charlie Bastnagel's paper presented in SUGI22.

- First component of the DWA Tool is Table Register, which allows the administrator to initiate a table name, assign the table to a warehouse where it belongs to and give other information for system control and data validation. It will bring the administrator to a text editor where he/she can enter a RISQL query. All the information includes source fields names, assigned SAS fields names, where clauses and so on will be sent to the system adapter to update the meta database.
- The Table Editor modifies an existing table instead of a new one, but it works in the same way as Register to update the system information.
- The Program Assembler works in the opposite way to the Register. It retrieves the information from the meta database and produces not only a modified RISQL query but also a corresponding SAS program that can take the RISQL output and convert it to SAS data table with correct informats and standardized formats.
- The Extraction Spooler takes extraction parameters set by the administrator, spool the jobs for multiple multiple tables and send each job as a batch process.
- At a given time, the System Monitor can query a data source system and the SAS server for accessibility, resources and other information. It also can compare the update date of SAS data table with the one from data source to confirm the necessity of an extraction.
- The Meta Data Viewer-Editor is straight forward. The Validation Reporter and Information Center will be discussed in detail with other topics.

DATA QUALITY CONTROL PROCESS

- We developed UNIX scripts and use them with SAS &syserr macro variable to monitor extract process and

capture any possible alert and error step by step and to categorize the messages into different groups.

- We have automated our E-Mail process. After we capture the messages, if any of them jeopardize the process or cause a serious alert, a E-Mail will be sent out to the administrator.
- We also developed data validation modules written in SCL and MACROS to verify the success of data extraction and transportation. We compare important information such as records count and major numeric field summarys between the data source and the result SAS table, and we also compare current extract with previous extract to check variation allowance based on the business definition. (See example program 2).
- All the information we collect from the validation process will be updated to the meta data table DW_LOAD. The Validation Reporter will be able to check all the necessary information and the logs and reports generated in the SAS session.

Here is an example of our Validation Macro:

```
A block of code the validation module to check duplicates.
%MACRO CHKSQL(plan_dw =,
                        dw_table=,
                        clm_sys =,
                        database=);
************************************
*Open control table DW_TABLE to get   *
 *information and assign macro variables. *
************************************;
 data _null_;
  set metabase.dw_table;
   more statements;
 run;
****************************************
*If there is a primary key then check duplicates *
*by this key and  output the dupe records        *
****************************************
%if &dupe_by ^= %then
    %do;
         proc datasets lib=&extr_lib memtype=data;
          modify &qryname;
          index create &idx_vars;
         quit;
;

         data shrvalid.&dupeqry;
          set &&extr_lib..&qryname;
          by &idx_by;
          if not first.&first_by then output;
         run;
         proc sql noprint;
          select
              case
               when nobs=. then 0
               else nobs
              end into :dupe_cnt
          from dictionary.tables
          where libname = "SHRVALID" and
                memname = "%upcase(&dupeqry)";
         quit;
    %end;
%else
    %do;
        %let dupe_cnt=0;
    %end;
*********************************************
*Collect the necessary information to send out e-mail   *
*if there is any  ERROR or WARNNING signal          *
*********************************************;
%let err_file = &saswork/err_rept.txt;
filename err_rept "&err_file";

%if &dupe_flg ^= 0 %then
      %do;
           proc printto print=err_rept;
           run;
           data _null_;
            file print notitle;
            put "ERROR:" //
                 more message;
                ;
           run;
      %end;
%if &dupe_flg ^= 0 or &sumv_flg ^= 0 %then
      %do;
           %let subject = Extract Validation for &plan_id
                          %trim(&tbl_desc);
          %sysexec mailx -s "&subject" -c "&users"
                                mengd@hlthsrc.com < &err_file;
      %end;
 %MEND;
```

COMMUNICATION

Since our users are spread out over the country, our means of communication is complicated. In order to establish a simple, easy and efficient communication channel and to better serve our end users, we brought two more components into the environment.

On the administrator's end, we added a series of tables with summary information about the DW Tables.

On the user's end, we developed a WEB application implemented on our company's Intranet. The WEB application is powered by SAS. The WEB Application provides run time information which includes a introduction on the data warehouse to new users, the schedule and actual dates of DW update, current table layouts and field attributes, on-line validation reports, etc.. An on-line support page let users send questions and feedback, report problems and submit special requests.

CONCLUSION

This paper discusses the reasons when developing a SAS Data Warehouse is in your companies best interest. It demonstrates that one is not forced to accept products off the shelf in order to manage a Data Warehouse. Using

SAS Software you can build your own SAS Data Warehouse and an effective set of tools with which to manage it.

REFERENCES

Welbrock, Peter R. (1996) "Structuring Your Data Warehousing Project: Moving From the Concept to the Reality", SUGI22.

Bastnagel, Charlie and Gates, Kevin (1996) "Using OOP to Enhance the SAS® System: An Adhoc Reporting System Using SAS Software, 3D Graphics and Web Interfaces", SUGI22.

ACKNOWLEDGEMENTS

We would like to acknowledge Mark Dalesandro for creating SCL methods to make flying widgets.

For Further Information Contact:

Di Meng & Charlie Bastnagel
CIGNA
2 Fountain Square 5-N
Chattanooga, TN 37402

mengd@hlthsrc.com; bastnach@hlthsrc.com

APPENDIX

SAS Data Warehouse Operational Structure