Paper 118

Data Warehousing on a Shoestring

Rick Nicola, SPS Software Services Inc., Canton, OH

Abstract: Perhaps the largest stumbling block in developing a data warehouse using SAS (or any other) products is cost justification. In many cases, the difficulty in gaining acceptance is not because of manpower expense, but in obtaining a long-term commitment to absorb the cost of additional software.

In many companies, "basic" SAS products are available to mainframe users. This seemingly "free" software offers an opportunity to create a workable data warehouse as a "proof of concept" without substantial software investment. In some cases it is possible, given some fairly low level SAS tools, to create a very usable, functional data warehouse. Note that a data warehouse created with these limited tools and platforms will have limited, specific functionality. However, the existence and success of this can be very helpful in cost justifying future projects and additional software.

This paper is a discussion of some of the techniques used to create a mainframe based data warehouse using a minimal SAS software configuration. This system contains many of the classical characteristics of a data warehouse and the flexibility to be extended to include other capabilities as supporting software products become available.

The Project: This project is based on work done for an Ohio based insurance company. Our consulting group, SPS Software Services Inc., worked with company analysts to devise a system that would meet the information needs of the "Special Lines" area of the business. Prior to this activity, the insurance company had created a DB2 based "Data Warehouse" for the "automobile" portion of the business. Reporting against this DB2 repository was accomplished with SPF screens, SAS and GQL. Since auto represents such a large part of their business, their reporting needs were more complex and their investment justification was much larger. Because the Special Lines area represented a much smaller revenue base, such a development investment was not justifiable. Fortunately, because of the size of this business area, there are fewer analytic requirements and it was possible to exploit the data with simple, straightforward, SAS tools and techniques. The Data Warehouse was built and is used under MVS, with some data exploitation elements that were developed on the PC and ported to MVS.

The Problem: Marketing analysts from the Special Lines department had a need to examine a small number of key business measures across a very large number of specific class or categorization variables. This need was being partially, but poorly met by some very long, expensive mainframe SAS batch jobs that ran directly against operational data. In addition to being difficult and expensive, there were also questions about the validity of the results and the handling of exceptions and data aberrations. It was agreed that these problems hampered the business analysts' ability to study data and properly adjust insurance rating factors.

The Request: A marketing analyst from Special Lines recognized the need for a major change in the way they reported on their business measures. The analyst also recognized that their area of the business could not cost justify investment in a DB2 based Data Warehouse similar to that used for the Auto end of the business.

Fortunately, the clarity of the Special Lines pricing model did not call for the complexity of such a Data Warehouse. Based on the analyst's input, the Special Lines business unit called on MIS for assistance in creating a "stats" system to meet their specific need. Their request was for a consolidated, clean vertical file, with an ad-hoc, economical reporting system creating reports from this file.

The Solution: The processes in building a data warehouse could be presented as follows:

1. Identifying detailed data requirements and source operational data.
2. Determining the structure of detail data in the warehouse (includes reviewing report layouts and mock-ups.)
3. Consolidating and cleansing data.
4. Building base metadata.
5. Data transformation, structuring, building star schema.
6. Reporting.
7. Summarizing data.
8. Creating "decision support" or data access front end.

In a perfect world, the project would have proceeded approximately in the steps shown above, with a detailed analysis of specific data requirements, identification of "class" variables, collection of meta-data like information and a myriad of other tasks addressing various data warehousing issues. The reality is that the Special Lines business unit had a very precise idea of what they needed from both a data and a reporting standpoint. The building of the system would focus around creating a "clean," vertical file that contains key business measures for all possible class variables. The file would be the ultimate in simplicity with one record for each insurance coverage (e.g. collision) within each policy. Each of these records contains all other pricing variables and each of the key business measures (e.g. earned premium.) All data preparation, post processing, and reporting would focus on feeding to or feeding from this file.

In summary, while the eight steps above were essentially followed, the Special Lines business unit dictated the first three because these steps were driven by overwhelming business needs. And perhaps that is the way things should be.

Other considerations: In creating the data warehouse, a very direct, very flexible approach was called for. While the base file was simple, there were other circumstances that added a layer of complexity. For example, though Special Lines is the general category of this insurance offering, there are really distinct product offerings for each of five "special" vehicle types.

The five products all share the same key business measures and some class variables. But each product introduces many class variables that are distinct to that offering. In building the system, it was important to create a structure that took advantage of the similarities, but was flexible enough to accommodate the differences. This basic philosophy was to drive the design of the data warehouse.

The Process: As was stated earlier, building the Data Warehouse did indeed follow those distinct eight distinct steps.

1. Identifying detailed data and source operational data:

The task of identifying the detailed data requirements fell to the Special Lines business unit. Being familiar with their own reporting needs and prior “interesting” class variables, the business analysts were able to hone in on their specific requirements. The difficult task of finding the source of these variables from numerous operational data files fell to the analysts in MIS.

2. Determine structure of detail data:

As was mentioned previously, determining the structure of detail data in the warehouse was dictated and simplified by the business unit’s need. Driven by a clear idea of the business measures and class variables required, the simple vertical structure of the file was determined. Each of the five products would contain upwards of 40 class variables (such as policy) and five business measures (such as earned premium.) For example, fields could include

Policy number:	**1111111**
Coverage:	**Collision**
Driver age:	**22**
Earned premium	**12**
Other business measure	**0**
etc.	

3. Data consolidation and cleansing:

This step also included some initial data transformation. This exercise was, without question, the most difficult and time consuming of the entire process. The effort was led by a senior MIS systems analysts who was familiar with potential operational data feeds from existing systems. With the help of additional contract manpower she put together a series of data collection programs that pulled, cleaned, and consolidated data from these multiple feeds. These processes were written in COBOL and also performed the task of de-normalizing or structuring the data in a wholly rectangular format.

4. Build base metadata:

Creating metadata for this system was again driven by very specific needs. In lieu of a more complex method of storing information about the data, a simple table was built to house descriptions of both class and business measure variables. The table included the variable name, the type of variable it is, a default output format, and a default SAS label or header. This table was stored simply as a member of an MVS Partitioned Data Set (PDS). For example a particular age group variable would be identified in this table as:

Variable name:	**AGEGRP**
Default format:	**$6.**
Type of variable:	**CLASS**
Default label:	**Rate Age Group**

This variable level definition was available for both defined and derived variables. In order to feed this information to SAS, a simple routine was written to generate FORMAT and LABEL statements using the appropriate values from this meta-data table. These statements are included “on the fly” within each of the warehousing processes.

5. Structuring, building star schema:

The building of a “star schema” or a categorization model for class variables was done together with the Special Lines business analyst. Most of the class variables needed refinement or categorization into groups. For example, one of the class variables is “driver age.” We wish to categorize a customer’s age (such as 22) into a driver age category. (driver's age 20 through 23.)

In this system two things can be done. A new, derived variable can be created (AGEGRP) and / or the format defining the mapping can be assigned to Driver Age. Sometimes both steps are taken. A new variable might be created (let us call it AGEGRPV) so that summarization may be done only on the age group, not on a specific age. Furthermore, we may also retain the simple variable, Driver Age, and assign the AGEGRP format to it for future reporting.

This transformation is essentially a star schema table or mapping. One might configure these descriptions as simple tables, or manage them with an FSP front end. In this case, however, the Special Lines business analyst had considerable SAS savvy, and preferred to have the mappings stored as SAS formats in an MVS PDS.
For example:

```
VALUE  AGEGRP
        0            =  ‘0’
    0<-21            =  '14-19'
    20-23            =  '20-23'
    24-34            =  '24-34'
    35-HIGH          =  'OTHER'  ;;
```

Just as the format and label statements were built dynamically, so are the appropriate PROC FORMAT builds. This is done by pulling in all of the members in the format PDS and again including the code “on the fly” to each warehousing process.

6. Reporting:

As was mentioned earlier, the reports coming from the system were to be very simple: Summarize five key business measures for any number of class variables that are in the file. For sample purposes, we use a subset of the five business measures including **written premium** and **earned premium**. In such a case, a columnar report may contain:

Age group
Marital Status
Rate Revision Date
Accident rating
Written Premium
Earned Premium

This simple report was made using a PROC REPORT based macro. As time went on, it became more sophisticated in its ability to use BY (top of page) variables, as well as simple class variables, introduce new defined variables at report time, and several other features.

```
S P E C I A L   L I N E S   M A R K E T I N G
LINE BY AGEGRP RATEDMS    CRITERIA:  ALL DATA

RATED       RATE                        INCRD
AGE   RATE  REV   WRITTEN    EARNED      LOSS
GROUP MS    DATE  PREMIUM   PREMIUM    AMOUNT
_________________________ _________ _________
14-19 MARR JAN94   99,999    99,999    99,999
           JAN95   99,999    99,999    99,999
_____ ____        _______ _________ _________
14-19 MARR         99,999    99,999    99,999

_____ ____        _______ _________ _________
14-19 TOTAL        99,999    99,999    99,999

20-23 MARR JAN94   99,999    99,999    99,999
           JAN95   99,999    99,999    99,999
_____ ____        _______ _________ _________
20-23 MARR         99,999    99,999    99,999

_____ ____        _______ _________ _________
20-23 TOTAL        99,999    99,999    99,999

24-34 MARR JAN94   99,999    99,999    99,999
_____ ____       ________ _________ _________
24-34 MARR         99,999    99,999    99,999

_____ ____       ________ _________ _________
24-34 TOTAL        99,999    99,999    99,999

===== ====      ========= ========= =========
GRAND TOTAL        99,999    99,999    99,999
```

The use of the reporting element also emphasized the need for some type of summarization. For example, it seemed absurd to process 6 million records from tape at an incredible elapsed time and machine cost, to create a one-page report. This led to the creation of summary files.

7. Data summarization:

The time and expense in reporting across large amounts of data for very simple reports forced the creation of summary files. Since our detailed file is rectangular, we could “roll up” on any class variables. This rolling up was based on a simple, mathematical principle: When dealing with simple sums of numbers, a summary file with *y* class variables can be further summarized to a subset of those y variables.

Consider a summary file containing:

Class Variables	Business Measures
Marital Status	**Earned premium**
Sex	**Written premium**
Age group	**etc.**

We can create distinct reports by marital status, by sex, by age group, by marital status and sex, by marital status and age group, and any other combination of those variables. Note that these roll-ups are simple, PROC SUMMARY nway summarizations. These are not MDDB data structures, though the concepts are similar.

Given these facts, we must ask **what summary files, containing which class variables, should we create?** These variable groupings were identified by discussing what "class" variables needed to be analyzed together and what reports would be interesting on a regular basis. Ultimately, each of the product areas ended up with five or six summary files with as many as fifteen class variables.

Interestingly enough, this summary process was not that different from the reporting. Thus, we were able perform this summarization with some variations on the PROC REPORT reporting macro. All summarization is done at the time that the detailed data file is built (monthly) directly from the original rectangular file. Processing is performed at night when run costs are the lowest.

8. Creating front end.

Creating a "decision support" or data access front end was also desirable. As the system stood at this point, the business analyst could submit a fairly simple macro invocation to create his desired report. But this could be tedious and prone to error. In addition, one would expect that the analyst would have a whole series of pre-written report invocations that he would store in his own personal TSO PDS. Furthermore, the proper use of summary files would demand constant awareness of the structure of these files to optimize the creation of the reports. This led to the realization that a "front end" would be required to build optimized reports, and store them for future usage.

Again in a perfect SAS world, we would have written a cute SAS/AF® FRAMES app using SAS/CONNECT® to the mainframe to extract and report on the data. The reality was that SAS/CONNECT® was not available at the time, nor was SAS/AF on any platform. The approach taken was that a SAS/AF application was created on a PC owned by out consulting company, SPS Software Services Inc. As a SAS Quality Partner, we were licensed for the product. (Since that time the insurance company has also become licensed for SAS/AF on the PC.) Some sample data (detailed and summary) were dropped down to the PC and stored in SAS files, as they were on the mainframe. An AF system was then developed and ported to the mainframe. This front end allows the user to create custom reports. These reports are:

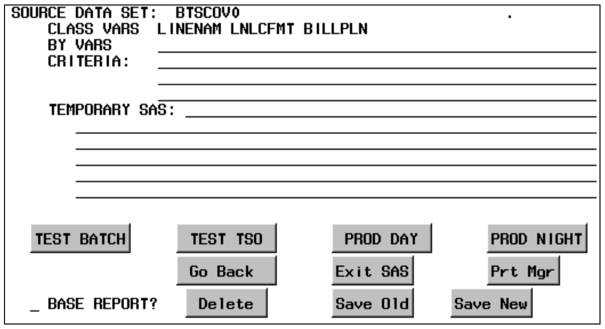
1. Data driven and built dynamically by pointing at class variables in the summary and detailed files.

2. Stored for future usage.

3. Optionally tagged for regular submission during regular monthly procession.

4. Include a test facility that allows the analyst to validate the report by testing it interactively or in a batch run.

A sample **build screen** is as follows:

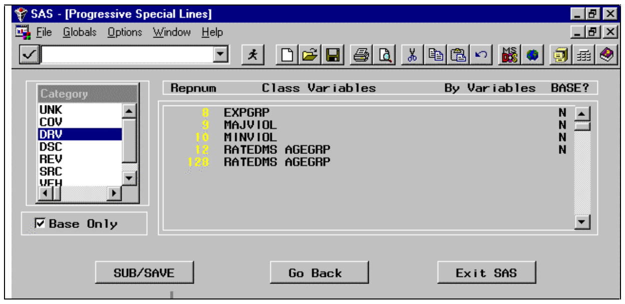


```
SOURCE DATA SET:  BTSCOV0
    CLASS VARS  LINENAM LNLCFMT BILLPLN
    BY VARS
    CRITERIA:

    TEMPORARY SAS:

_ BASE REPORT?
```

A **selection / save screen** for existing reports follows:



SAS - [Progressive Special Lines]

```
Category     Repnum   Class Variables      By Variables   BASE?
UNK               8   EXPGRP                              N
COV               9   MAJVIOL                             N
DRV              10   MINVIOL                             N
DSC              12   RATEDMS AGEGRP                      N
REV             128   RATEDMS AGEGRP
SRC
VEH

[✓] Base Only
```

The interactivity of the system allows the user to select class variables based on their existence in the summary or detailed file. This interactivity coupled with a testing capability cuts down enormously on incorrect report submissions.

Summary and futures:

Quite basically, we have determined that we can create a Data Warehouse with minimal software investment. But this warehouse is limited with moderate capabilities. Given the "win" created by the initial effort, we can identify extended functionality given by other products. Specifically:

1. Extending to OLAP and other facilities requires investment in more software such as **SAS/EIS**® and **SAS/MDDB**®.

2. Application "frontending" is limited on the mainframe. Cross system communications provided by **SAS/CONNECT**® is called for to have access to data on the mainframe while providing an object oriented front end on the PC.

3. While limited warehouse and metadata tracking can be done "by hand", the low cost approach can prove expensive in the long run. Tracking mechanisms and development standards forced by the **SAS/Data Warehouse Administrator**® may help in initial development costs, as well as long term maintenance and management costs.

Numerous other opportunities exist to expand and add functionality to the data warehouse. Obtaining the products necessary can begin with a foundation of base level software and some ingenuity.

For additional information contact:

Rick Nicola
SPS Software Services Inc.
314 21'st Street NW
Canton, OH 44709
Ph: 330-454-4241   Fax: 330-454-4468
e-mail: spssoft@compuserve.com