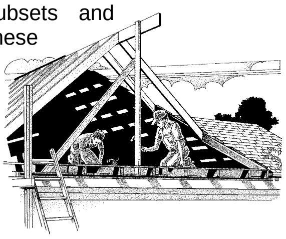
Paper 110

Constructing a Data Warehouse

Curtis A. Smith, *Defense Contract Audit Agency, La Mirada, Ca.*

ABSTRACT

Lately, we have heard and read much about data warehousing. While many papers discuss the concepts and reasons for data warehousing - here the author will describe methods to build a data warehouse. A data warehouse must be well organized, easy to understand, easy to use, and easy to reach. It will likely draw data from many sources, perhaps for multiple company segments, for multiple time periods, and varying degrees of detail. Should the developer create multiple SAS System® data libraries, or keep all the SAS data sets in a common library? Should the developer use a naming convention for SAS data libraries and data sets? What about documenting the data warehouse? Are indexes a good thing? Can indexes be overdone? Subsets and summarized SAS data sets? All these questions and more will be answered in the pages to follow as the author describes learns how to construct a data warehouse.



INTRODUCTION

When we set out to construct our data warehouse, we have already decided that it is a good thing to do and it has its merits. We already have an idea what we are going to put in it, that is, what it will house. However, we may not yet know what it will look like, where we will erect it, or what we will use as building materials. Now is the time to become a data warehouse architect and make plans to build our data warehouse. Let us start with a model of our enterprise.

OUR MODEL

Our goal is to build a data warehouse that serves data for pre-made (automated) user applications and provides data for users to create ad hoc queries, analyses, and reports. Our site has three company segments, several years of historical data, and four types of data: labor, overhead, material transactions, and work-in-process. We get the labor, overhead, and work-in-process data each month in flat files on tape stored under IBM® MVS®. And boy, are they big files. We get the material data on-line from an IMS® database. We need the labor and overhead data detailed each month, but need the work-in-process data only as a current year-to-date total. To meet our needs, the material transactions must always be current. Stored in flat files on MVS, we also have several tables containing descriptions for a variety of data entries, such as account titles. Our target user platform is a Pentium PC connected to the company network via TCP/IP, which connects us to the IBM mainframe plus other systems. Finally, we will run the PC under MS-Windows, connected via SAS/CONNECT® software.

DATA WAREHOUSE COMPOSITION

As SAS users, our data warehouse has two components: permanent SAS® data libraries and SAS files. SAS data libraries are collections of SAS files. To be most useful, SAS data libraries will contain homogeneous SAS files. That is, the files in the SAS data libraries will bear some relationship to each other. Or, in other words, each file has something in common. For example, we might have SAS data libraries set up for files from each different year of which we are interested. Here, the commonality is the year.

SAS files come in several varieties. Most of the SAS files in my SAS data libraries are SAS data sets and SAS data views. Usually, we create SAS data sets by importing data from other (external) sources. We might create them from a flat file, or a database, or from other sources. SAS data sets can be stored permanently, or just for the current session. On the other hand, SAS data views are structures for SAS data sets, but do not hold any observations. They are used to read data from sources external to SAS and create a permanent or temporary SAS data set.

Permanent SAS data sets provide the benefit of having the data we need already in the SAS System format. However, they cost storage space. SAS data views provide the same functionality as SAS data sets, but because they do not hold any observations, do not cost us storage space. When we use SAS data views, however, SAS must create SAS data sets from the source data each time we reference the SAS data view. If we need to access the same source data more than once, using SAS data views causes us to reread the same source data, wasting processing. Be careful - reading raw data into SAS can be very time consuming. Therefore, we will want to consider carefully when to use SAS data sets and SAS data views.

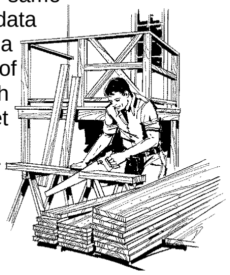
SETTING UP A SAS DATA LIBRARY

When we are ready to build our SAS data libraries, we must consider how we will organize our SAS files into our libraries, where we will store our libraries, the devices on which to store them, precisely how to structure them, and what to name them. If we do not construct our SAS data libraries well, our data warehouse may be user unfriendly and inefficient.

ORGANIZATION

A data warehouse should be logically arranged. It will not be very useful if our developers and users cannot find the SAS files they want. When designing our data warehouse, we want to think about what the developers and users need. That is, what arrangement of SAS data sets and SAS data views will best serve our developers when building applications and what arrangement will best serve our users when running queries, analyses, and reports. What is best for our developer might not be the best for our user.

Let us get back to our model. We have four types of data, for three segments, covering multiple years. Because we rarely need data for more than one segment in the same application, we will create a separate SAS data library for each segment. Because we usually need data for only one year for the same application, we will create a separate SAS data library for each year. This will give us a separate library for each combination of company segment and year. Within each SAS data library, we will keep a SAS data set for each of the four types of data. We will also have a single SAS data library for our tables.



LOCATION

Now that we have decided how to organize our SAS data libraries, we must select our construction site(s). If we are operating in one environment, for example MVS, then we do not have any choice where we place our library. On the other hand, in a Windows-based client-server environment, we can choose our library locations based upon several factors, like cost, accessibility, I/O speed, and user friendliness. We might have our entire library in the same place, or we might distribute our libraries across multiple platforms.

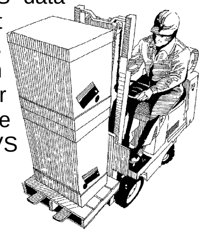
In our model we will be using a Windows-based PC with SAS/CONNECT to the mainframe. All four of our data types come from the IBM mainframe - three are flat files under MVS. Therefore, in our model, we choose to create our SAS data libraries under

MVS, from where we can also access our IMS database. Our tables are small and relate to all of our SAS files. So, we opt to build them in a SAS data library on the PC.

Storage Devices

Having decided what platforms to use, we must decide what storage devices to use. Each platform offers many devices from which to choose. Typically, a disk device will provide the best performance and utility. However, sometimes disk devices will not have the capacity we will require, or may be too costly. In these instances we must use something else. Tape storage, for example, is an excellent alternative for mass storage. The SAS System can create a SAS data library on tape almost as well as on disk. We cannot do some things when using tape libraries that we can do with disk libraries because tape drives are sequential devices. (Other papers address the tradeoffs of tape devices - we will not discuss them here.)

The physical structure of the SAS data library depends on the platform and device. This could have a bearing on our selection of a platform and device. For example, on the PC and under VMS a SAS data library is a subdirectory. While under MVS it is a physical file (but not a partitioned data set). SAS data libraries that are subdirectories have an advantage over SAS data libraries that are physical files: we can use host utilities to maintain SAS files within the libraries. For our model, we will use disk devices on both the PC and on the mainframe under MVS. Our SAS data libraries on the PC will be subdirectories and our SAS data libraries on MVS will be physical files.



File Allocation

When creating a SAS data library under MVS we must set several DD parameters, such as the SPACE, data control block (DCB), and disposition (DISP). SAS data libraries on disk may be created from within SAS or externally. Those created on tape under MVS must be created externally. Many host allocation parameters are available as options under SAS. While many SAS books, including the SAS Companion for the MVS Environment, deal with these parameters quite well, I do have a couple favorite tips for allocating new and existing SAS data libraries.

- Under MVS, catalog SAS data libraries. Assigning SAS data libraries will be greatly simplified if you only need to refer to the SAS data libraries with the data set name (DSN) and maybe the disposition parameter (DISP). In fact, if you want to allocate SAS data libraries from within SAS (using a LIBNAME statement), you must catalog your SAS data libraries.

- Under MVS, optimize the block size (BLKSIZE). The block size is set when you create a physical file. This relates to the number of observations transferred together as a group. Each block is separated by a marker, known as the interblock gap, that occupies space but does not hold any data. The smaller the block size, the more interblock gaps that will exist in the same amount of physical space. Therefore, smaller block sizes result in more interblock gaps in your SAS data libraries, and thus, more unusable space. Also, larger block sizes transfer more observations as a group and can increase I/O throughput (decreasing EXCPs). Larger SAS data sets within your SAS data libraries work better with a large block size, like 27648. (SAS data libraries on tape have their own peculiar block size. Refer to the SAS Companion for MVS.) You can read a thorough and technical discussion of block sizes in Michael A. Raithel's book "Tuning SAS Applications in the MVS Environment", available through the SAS Institute.

- Allocate and keep enough disk space. When allocating space for your SAS data libraries, you may not know how big and how many SAS data sets you will eventually place in the SAS data libraries. It is quite possible that at some point in time you will fill your SAS data libraries. Be careful when using the MVS or SAS release (RLSE) option in the MVS or SAS SPACE parameter. Using this option will cause the operating system to release all unused allocated space in your SAS data libraries when the SAS job completes. The next time you try adding to your SAS data libraries, you might not find any available space.

- Share your SAS data libraries. When allocating an existing SAS data library, use the MVS or SAS DISP parameter of share (SHR) when you plan to only read data from your SAS data libraries. This will allow others to access to the SAS data library at the same time.

Naming

How we name our SAS data libraries is very important. Cryptic names are of no use to the user, although they may be fine for the developer. Users will not find cryptic names easy to use or remember. Therefore, SAS data libraries destined for users should be named with something that is readable and relates to the content. Standardized names are important for macro variable purposes. (Standardized applications will likely rely heavily on macro variable references.) Let us go back to our model. We have three company segments. They are called segments 'AB', 'CD', and 'EF'. We currently have data for years 1995, 1996, 1997, and 1998. Our SAS data libraries will be on disk, but we anticipate adding other SAS data libraries later for other types of data that will be too big for disk. So, we might name our SAS data libraries something like:

hilevel.sas.disk.%%####

where *hilevel* is our high level qualifier; '%%' is the two character code for the segment; and '####' is the four digit numeral for the year. (Don't create a year 2000 problem by using only two digits for the year. And remember, the Y2K problem won't be an issue until the year 2048.) Or, for example:

hilevel.sas.disk.ab1996

This naming convention allows us to keep adding libraries for more years and more company segments without the possibility of duplicating the name of any file. Also, if we need SAS data libraries on tape for other data, we can change the *disk* to *tape* and have just as many SAS data libraries on tape, each named uniquely from their disk counterparts. Macro referencing becomes simple, because we can create a macro variable for the device, company segment, and year. For example:

```
%let device=disk;
%let segment=ab;
%let year=1996;
libname mylib "hilevel.sas.&device..&segment.&year.";
```

When developing a naming convention, I try to envision future expansion. Also, I always keep the variable part of the names to the same length. For example, I would have all company segment names the same length and all the year references in the same length and format.

Setting up SAS Data Sets And Views

Now that we have SAS data libraries constructed to hold our files, it is time to build SAS files. When we build SAS data sets for end users, it is very important that the files are user friendly. All the new GUI interfaces are designed to put the data into the hands of users who cannot readily write program code. This is a hint that many of our target users are not programmers, system designers, or system analysts. So, we need to design files for the users in a way that will reduce the chance of misuse and misinterpretation.

SUBJECT ORIENTED

We probably want our files to be subject oriented. Users of our data warehouse will likely need reports, analyses, and queries related to a subject - like a labor hour analysis or an overhead trend. Therefore, separating our data into subject groups usually makes good sense. For example, we may want to have a labor file, a material file, an overhead file and a work-in-process file. Though we have four sources for these four types of data, we could combine them into a single file. However, that might not be very friendly, because our users usually need these types of data separately. On the other hand, we might decide to break the source files up into various subjects. It may be that our users of labor data typically need only indirect or direct labor at a time. So, we may choose to separate our labor into direct and indirect labor files. We might also decide to create a separate labor file for overtime labor. We might split our material data into subcontract data and parts data. A survey of our users and their needs should reveal to us the types of data they will need.

LEVEL OF DETAIL

The level of detail (or granularity) we retain in our files is also important. We will want to give our users as much detail as they need, but we do not want to waste space with records they do not need. The number of observations in our files depends on the level of detail. The number of observations affects performance and storage costs. But the level of detail could also have user friendliness issues. Data could exist at such a level of detail that understanding what it means could be difficult. What if, for example, we have transaction detail down to peculiar line items on invoices. Our users may never need such detail, but they might unknowingly exclude some detail from their queries, thus understating the true balances. If we need all the detail - great, but if we do not, we might be safer eliminating from our data warehouse the detail our users do not need.

Use the SUMMARY procedure to summarize a SAS data set into a new SAS data set of lesser detail. You can use PROC SUMMARY without first sorting your detail file if the detail file is indexed on the variables you want to use as your summary key. If your detail file is not sorted or indexed, you can still summarize it using the CLASS statement instead of the BY statement. But be careful. The CLASS statement will drop observations if the value of a summary variable is 'missing' unless precautions are taken. To prevent the PROC SUMMARY from dropping these observations, use the MISSING option.

Also, the CLASS statement creates multiple levels of detail (level of interaction among CLASS variables), one level for each combination of summary variables. Within the summarized SAS data set there will be a new variable named *_type_*. This variable holds a value indicating the level of detail. Creating summarized data sets this way can be dangerous for your users as they can easily double, triple, quadruple, etc, count the values in the data set unless they know how to select on the *_type_* variable. The solution to creating the multiple levels of detail is to add the NWAY option to the PROC SUMMARY statement, or use the BY statement rather than the CLASS statement. But when you use the BY statement, the detail file must be sorted or indexed. Following is an example using the CLASS statement.

```
proc summary data=detail missing nway;
  class account dept;
  var amount;
  output out=lessdtl sum=amount;
run;
```

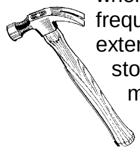
TIME PERIODS

The time periods we keep in our data is another important aspect of our SAS data sets. In our model, we get our labor, overhead, and work-in-process data each month. We could choose to fill our data warehouse with twelve monthly files for each of our file types. However, this would probably not be very user friendly. A better approach might be to design the files as year-to-date files with the twelve monthly source files appended together. As developers, we could design a program that we would run each month to append the current month data into to our existing file containing the prior months. If the users do not need monthly details, just the sum of the months, then we could eliminate some by summarizing the file.

TYPE OF FILES

When creating files for our data warehouse we can choose between SAS data sets and SAS data views. The difference being that SAS data sets actually hold data in the file, while SAS data views do not. SAS data sets have the advantage of reducing processing time when used, as the data has already been converted to SAS and is waiting for the user. Reading external data into SAS is a slow process compared to reading data from an existing SAS data set. SAS data views, not holding any data, have the advantage of not requiring much storage space.

When we have static source data, such as our labor data, and we plan to use the data frequently, SAS data sets make good sense. Here, we read the external data into SAS once and use it often. Reading the same external data into SAS data sets more than once does not make good processing sense, which is what happens when using SAS data views for external data that users access frequently. On the other hand, making a SAS data set from external data that will be infrequently used may not make good storage sense. Our material data is an example of where we might want to use a SAS data view. Here, our material data changes hourly, and our users need the data current at a point in time. Setting up our material data as a SAS data view will provide the user with the most current version of the data whenever they call for it.



NAMING

How we name our SAS data sets is just as important as how we name our SAS data libraries. Naming conventions are a good thing. Using names that make sense to the user is also a good thing. Remember all those cryptic file names we find in mainframe systems? Forget them! Our users will never remember such cryptic names and will never find the files they need with such cryptic names. Simple, boring, everyday names like 'labor' and 'material' will go a lot farther. But what if we have more than one file for a type of data? Or more than one time period? While we are stuck working with eight characters in our file names, we can do marvelous things with those eight characters, if we are clever. We can use combinations of abbreviations. For example, '*mtd*', '*ytd*', '*itd*', for month-to-date, year-to-date, and inception-to-date; '*sum*' and '*dtl*', for summary and detail; and '*lbr*', '*mtl*', '*ovh*', '*wip*' for labor, material, overhead, and work-in-process. We might combine them into '*lbr_ytd*', '*mtl_dtl*', and '*ovh_mtd*'.

In our model, we have separated our company segments and years into different libraries. This allows us to name the same type of data for each company segment and year with the same name. That is, we can call all of our year-to-date labor files 'lbr_ytd', while maintaining distinct files. To illustrate, consider the following example where we concatenate and subset two years labor year-to-date files for the same company segment:

```
libname ab1996 'hilevel.sas.disk.ab1996';
libname ab1997 'hilevel.sas.disk.ab1997';
data work.subset;
   set ab1996.lbr_ytd ab1997.lbr_ytd;
   where account='1234';
run;
```

DATA WAREHOUSE EFFICIENCIES

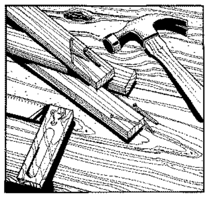
Optimizing your data warehouse, both your SAS data libraries and SAS data sets, can have a tremendous impact on performance and cost. Let us look at a few of my favorite tips.

SAS DATA LIBRARIES BLOCK SIZE

As discussed earlier, the block size of our SAS data library physical file can greatly affect performance. The block size has two important aspects for our performance. Larger block sizes should transfer more data, faster, to SAS. Larger block sizes within the same physical space will result in fewer interblock gaps, which will save storage space. When I first read of this, I did not really expect much. Willing to try any thing to reduce the size of my allocated space to my SAS data libraries, I created new physical files with larger block sizes and copied my existing SAS data sets into the new SAS data libraries. I was extremely surprised when I noticed the amount of allocated space dropped dramatically.

ELIMINATE UNNEEDED OBSERVATIONS AND VARIABLES

Unneeded observations and variables cost both storage space and processing time. While we do not want to eliminate something the users will need, we also do not want to keep something they will not use. Sometimes, after our data warehouse has been in use for a while, we can reevaluate how our users are using the data warehouse and determine that we can eliminate some observations and variables from our files. Remember, unneeded observations can cost processing time as do unneeded variables. SAS must keep track of the variables, too. What is more, when SAS data sets are stored on tape, the operating system must read the tape through the end of the file. Because data is stored in a long sequence, observation by observation, variable by variable, bit by bit, the more variables and observations we can eliminate, the better.



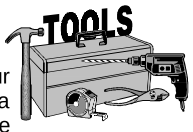
SUMMARIZED DATA VS. DETAIL

Detailed data offers the user more visibility, and thus, more information. Summarized data is smaller, and thus, improves performance and reduces storage costs. It is usually a good idea to keep just the detail that will satisfy most, if not all, needs. Often, that level will be much more than many users will need. Frequently, we find that maintaining both detailed and summary files is a great idea. When users do not need the detail, they can use the summary files, saving processing time. This can be very important when operating in a client-server environment and using the client's processor. Sometimes, we have a level of detail rarely needed. Retaining that level would generally waste much space and processing time for most uses. A solution is to keep a SAS data set with a level of detail that serves most needs and keep another SAS data view to access the source data at that rarely needed level of detail. We might then have three levels of detail, a SAS data view for the greatest amount of detail, a SAS data set containing typically needed level of detail, and one or more SAS data sets containing summarized data.

LOOK-UP TABLES

SAS data sets containing descriptive or coded information and keyed to variables in other files can be called *tables*. Within the tables we would typically store a unique observation for each key variable. Tables provide us the means to store repetitive information outside our main SAS data sets. This reduces storage space. For example, if our SAS data set contains a variable for our four digit account and there are 100 different values for the account and the SAS data set contains 100,000 observations, we will have many observations with the same account number. This is okay, because that is the way the data is recorded. But, if we also had a variable in the SAS data set for the 30 character account description, we would have 30 extra characters per each of 100,000 records, much of which would be repeated information. In contrast, if we maintained a table with the 100 account numbers and their associated description, our table would be small and our SAS data set would be greatly reduced in size. If our source data contains data that we have placed into a table, we can choose to drop that data when we create our SAS data sets from our source data.

We can create SAS data sets as tables for a variety of codes and descriptions and store them in a common SAS data library. Because our users will likely use the tables with SAS files from all of our SAS data libraries, we will want to place our table library on the same platform as our other SAS data libraries, or on another platform accessible to our client-server session. We will probably want to name our SAS data library's physical file name or subdirectory name with something that will suggest that the library contains our tables. So, we might use the word 'table' as a subdirectory name or part of a physical file name.



Here is a tip if you are working in a client-server environment and are deciding which platforms to store you data libraries for your data and your tables. Typically, your SAS data set tables will be sorted or indexed so they are ready to use. Not all operating systems sort with the same rules. For example, ASCII orders numbers greater than characters, while EBCDIC orders numbers less than characters. If we are using SAS/CONNECT between a Windows client and an MVS server and we are using the PC CPU to process and our tables library is on the server, the tables data sets will be sorted or indexed using the EBCDIC order. But the SAS data set we wish to merge with the table might have just been sorted by the PC CPU, which would be done with the ASCII order. In our SAS log we would find an error message that our table was not in the proper sort order.

COMPRESSING SAS DATA SETS

As SAS data sets get bigger, storage space becomes an issue. To reduce storage requirements, and thus costs, SAS has an option to compress SAS data set observations by reducing redundancy. To compress SAS data sets, you set the SAS options, COMPRESS=YES, when you create the SAS data set. You use the option with the OPTIONS statement or as a data set option. For example:

```
options compress=yes;
```

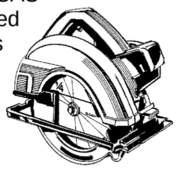
Compressed SAS data set observations require less I/O to process. Also, unlike uncompressed SAS data sets, space occupied by deleted observations in a compressed SAS data set can be reused. Specifying the option REUSE=YES when the compressed SAS data set is create will allow deleted observation space to be reused. For example:

```
options compress=yes  reuse=yes;
```

Compressing SAS data sets does have a down side: writing and reading compressed observations requires additional CPU time.

INDEXES

To avoid having to sort SAS data sets before using a BY statement, you can create one or more indexes on your SAS data sets. Indexing does not rearrange the observations in the SAS data set, it creates pointers used to locate the indexed observations. Indexing is a good user friendliness technique, as your users will not have to worry about sorting before doing tasks, such as merging SAS data sets. However, using indexes rather for tasks such as PROC SUMMARY may actually be slower than when using sorting SAS data sets.



Also, WHERE selections process much more quickly when done on indexed variables. However, indexing your SAS data sets has a tradeoff. The index is a separate file in your SAS data library. The more indexes you created and more the complicated they are, the bigger the index file. Having a couple indexes and perhaps, compound indexes, can greatly increase the size of your SAS data libraries. I find indexes to be extremely useful if I have one or more variables that I BY process often.

USER FRIENDLY DATA WAREHOUSE

Our data warehouse will be much more successful if it is user friendly. When designing the SAS files in our data warehouse, as much as possible, we want the SAS files to be ready to use the way the users need. We do not want, for example, our users repeatedly appending twelve monthly files before generating a report or constantly sorting a file before merging it with a table. And there is the tip off. If users have to do the same things to the files often, then we probably did not design right. Here are some of my favorite ideas to increase the user friendliness of the data warehouse.

INTUITIVE NAMES

When I originally created my data warehouse, I did not use intuitive names for my SAS data sets, although I did use intuitive names for my SAS data libraries. I assumed that my users would get used to the programmer's cryptic file names. Much later, I discovered that even my most sophisticated users were not using the summary files when they could - they did not know what was in the SAS data libraries other than the detail files. So, for years they always used the detail, frequently wasting CPU time and waiting longer for their results. So, I rebuilt my data warehouse using intuitive names for the SAS data libraries and SAS data sets.

ELIMINATE PITFALLS

In days of old, programmers created big COBOL systems that read data from files no one else, other than a computer wiz, was ever expected to access. These files were constructed with some strange stuff in them, rendering them not very user friendly. If we create our data warehouse by reading all records and all fields, as is, from the source files, we may create pitfalls and traps that our users will fall into. Let us review a few examples.

Many files have multiple record types, each having a separate record layout. Others may have many time periods or transitions of the same data. That is, there might be detailed records and summary records in the same file, identified by some code field. If our SAS data sets have multiple record types, time periods, or transitions, our users will likely include in their queries all records, without discrimination. What if we have a source file with detail records and hash totals and we create our SAS data sets containing both. When the user generates a report from all records, the report totals are double what they should be. In cases like this, we need to eliminate the multiple record types, or the multiple time periods, or the multiple transitions that the user will fall upon. If they are truly needed, then we must find a way to deal with them. Perhaps creating separate SAS data set for the different record types, time periods, or transitions.

Of course, we should always validate the data we place in the data warehouse before letting our users access it. Be sure all the data is there that should be and that totals and subtotals match known amounts. You do not want to assume that all is well, just because the SAS System read all the records from the source data. The source data could be somehow erroneous or corrupted, for example.

Sometimes we need to normalize our data. Suppose our source data has a field for a home department and another for a work department. To save space, the system generating the source data leaves the work department blank when the value is the same as the home department. All the old programs using the data know during report generation to fill the work department with the home department value when the work department is blank. If we read this data into our data warehouse, as is, our users will use the data as is - no matter how often we tell them how the work department is stored. You can bet they will produce a report with work department 'blank'. So, we need to fill those blanks. We need to correct the variables and observations in the data warehouse so the data is ready to use. And that makes sense, anyway. Correcting the data once ourselves when placing it into the data warehouse is more efficient than to expect users to correct the data each time they use it.

DOCUMENT YOUR DATA WAREHOUSE

Documenting your SAS data libraries, SAS data sets, and SAS data views is so very important. Without some type of documentation, users will not know which SAS data libraries are which, or what is in them; nor will they know one SAS data set from another. SAS makes documentation very simple. You can use the data set LABEL option to attach a descriptive label to the SAS data sets. The variable LABEL statement can be used to attach a descriptive label to each variable. Use them both on all SAS data sets and SAS data views in the data warehouse and on each variable within each SAS data set and SAS data view. Do not be fooled into thinking that some variable names and SAS data sets names are self-explanatory. Surely, everyone will know what is in the variable named 'amount', right? Well, maybe it represents a year-to-date amount, or a summarized amount, or an overtime amount. You can place more information in the label than you can in the name. Use the label.

You can use the CONTENTS procedure to generate a detailed report of your SAS data sets, SAS data views, and their SAS data libraries. Use the DETAILS option to get the most information. The _ALL_ parameter allows you to generate a report of the entire SAS data library. For example,

```
proc contents data=mylib._all_ details;
```

Using the data set label and the variable label will greatly increase the usefulness of your CONTENTS report. In addition to listing your SAS data sets and their variables, PROC CONTENTS will list the number of variables and the number of observations in each SAS data set, it will identify any sort sequences, it will identify whether or not the SAS data sets are compressed, it will list any indexes, and it will list a wealth of host information about the physical file. Once you generate a PROC CONTENTS of your data warehouse, you can print the report and provide it to your users. You can then supplement that with anything else your users and developers need to know about the data warehouse.

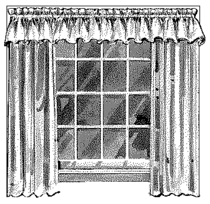
The SAS System has a new product, called SAS/Warehouse Administrator®. Based upon what I have heard, it appears to be a great tool for better documenting and maintaining your data warehouse.

ELIMINATE TABLE LOOK UPS

While tables are a great way to reduce storage space, having data related to our SAS data sets in separate files is not user friendly. If we add the related data to our files and have it ready for our users and developers within our main SAS data sets, our data warehouse will be much more friendly. As easy as the SAS System is, merging data files and tables is not so easy for many of our users. So weigh the tradeoffs of user friendliness vs. storage space.

LAST MINUTE TIPS

A problem I have had to deal with often is running out of disk space in my SAS data libraries. This can happen on the PC, under MVS, or anywhere. Some situations are more easy to deal with than others. If you run out of space on the PC, you can move or delete other files from the disk - the problem is solved. Under MVS it is not so easy. You will run out of space in your SAS data library when you fill the space you allocated for your SAS data library's physical file. Once that happens, you will likely have to allocate a greater amount of space to a new SAS data library. Then, using the DATASETS procedure or the COPY procedure, move or copy the SAS data sets to the new SAS data library. When you create new SAS data libraries under MVS, try to anticipate how much space you will need and allocate enough space for the library.



Under MVS, be careful not to release the space you allocate for your SAS data libraries. This will happen if you use the RLSE option when you create the SAS data library, or if you use a host utility to free unused space. If your site uses the SMS disk manager, you may need to take precautions to prevent SMS from releasing unused space in your SAS data libraries. SMS will, on a regular basis, release unused disk space within physical files. SMS has an option, management class (MGMTCLAS), that can be set to 'MCNORL#S' to tell SMS not to release the unused space. The SAS System supports this option on the LIBNAME statement when creating new SAS data libraries. For example:

```
Libname mylib 'hilevel.sas.disk.ab1997'
   disp=(new,catlg,delete) mgmtclas='mcnorl#s';
```

CONCLUSION

Useful data warehouses do not just happen. Careful and thoughtful planning will result in the blueprints for a good, user friendly SAS data warehouse. If your data warehouse proves to be less than user friendly, do as I did - spend the time to go back to the drafting board and redesign it. The time spent designing or redesigning will be quickly recovered by users who have a good data warehouse.

REFERENCES

PROC DATASETS:
SAS Procedures Guide, Version 6, Third Edition, Chapter 17

PROC SUMMARY:
SAS Procedures Guide, Version 6, Third Edition, Chapters 21 and 36

PROC CONTENTS:
SAS Procedures Guide, Version 6, Third Edition, Chapter 12

SAS MACRO Variables:
SAS Guide to Macro Processing, Version 6, Second Edition, Chapter 2
SAS Language Reference, Version 6, First Edition, Chapter 20

BLKSIZE
SAS Companion for the MVS Environment, First Edition, Chapter 17
Tuning SAS Applications in the MVS Environment, Michael A. Raithel, Chapter 4

LIBNAME Statement:
SAS Language Reference, Version 6, First Edition, Chapter 9

DATA Step:
SAS Language Reference, Version 6, First Edition, Chapter 2

INDEXES:
SAS Language Reference, Version 6, First Edition,Chapter 6

Questions: e-mail the author at casmith@mindspring.com