The Missing Semicolon™

Volume 11, Number 1 Spring 2009

Standardize Your Data Preparation in SAS: Use SQL!

SQL is a fantastic option to use in preparing data for reporting and analysis. Capable of so much more than simple data extraction, SAS programs can use PROC SQL for data transformations, manipulations, summarizations and calculations. Reducing use of the SAS DATA step and other PROC steps and increasing use of PROC SQL can make new programs more maintainable by I.T. staff, your co- workers, and even you.

Oftentimes, after a SAS process has proved its value in a company, it will be handed over to I.T. to support should something go wrong, especially for regularly scheduled production processes. Because some I.T. departments do not have staff to support SAS, they may have more informal SAS "ETL" (extract, transform, load) processes moved into other specialized ETL tools like Informatica or DataStage (it should be noted that SAS has excellent tools for formal ETL, like SAS Data Integration Studio, though not all sites have them licensed). Having the reporting and analytic steps separated from the SAS ETL steps facilitates this process. Regardless of whether or not your company subscribes to this model, many benefits can still be had by using SQL for the data preparation. We will see that SQL is a very flexible, robust language that can be used elegantly within SAS.

Separating Data Preparation From Analysis and Reporting

Many of us need to bring data together, clean it, and create new variables before we can create our reports and/or analyze our data. In other words, we need to prepare our data for analysis. Formally, this process is known as ETL. The vision at many organizations is to have IT handle any ETL processes that are needed and create a data mart for the business end-users.

Most of us still need to at least occasionally do our own data preparation. As analysts, when we are asked to create a new report, create a new model, or run a statistical procedure, we may approach the situation in a couple of different ways:

- Trial and Error: We sketch out a report layout, and attempt to create it using our procedure of choice from our main data source. If we realize we need to bring in an additional data source, we will add a step to create a new SAS data set and merge it with the current data source. This step, along with any cleaning or reworking is all in one program.
- Formal Approach: Before even entering SAS, a specification is put together to show what the data sources are, what transformations need to be done, what the structure of the SAS dataset(s) is, and a draft of the reports and/or analyzes that need to be done.

Well Segregated Programs Make for Quicker Changes

No matter what your personal style is, we strongly suggest that either at the beginning or at the end of your process, you create separate programs for your data preparation steps so they are not intermingling with necessary reports and analyses. Having more intuitive program names and separation of tasks will make it easier to know where to make changes if needed.

A good approach is to first create a data preparation stream that brings together the data and creates or updates permanent datasets needed for analytics and reporting.

Continued on page 3

Understanding the SAS® DATA Step and the Program Data Vector

SAS Global Forum 2009 Paper

Written and Presented by Steven First

www.sys-seminar.com/presentations.php

Systems Seminar Consultants, Inc.

2997 Yarmouth Greenway Drive
Madison, WI 53711
www.sys-seminar.com
train@sys-seminar.com
1-800-997-7081

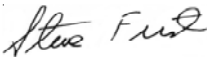
In This Issue

Letter From the President

Recent events in our office brought up some interesting newsletter topics. First was the writing of my paper for the SAS Global Forum in March. I was invited to present a paper on the DATA step and the Program Data Vector. I have been long time DATA step programmer and it was a fun topic to write about. The second was a series of large SAS projects that we were involved in with interfaces with relational databases and a few conversions to SQL based ETL tools. Our SQL experts tout the great attributes of SQL as a language both within and outside of SAS. We as a company, and I personally, have grown to appreciate not only the DATA step and its unique features, but also SQL's simplicity, power, and standardized features across the industry.

Luckily for us we don't have to choose one or the other. From within SAS we can use SQL and take advantage of its great design, simplicity, and power when we can, and use DATA steps when files are non-standard or if some unique feature required by the application. With all that being said, this issue will include Part One of an article on the DATA step's design and internals, along with and article on standardizing data preparation using SQL. We hope that just as we see applications and languages evolving, that you will too, and we can all write better systems in the future.

Please look for more on this topic in upcoming newsletters and live webinars.



Report from SAS Global Forum

SAS Global Forum met at the new Gaylord National Harbor Resort just outside of Washington D.C. As always, it was great to see new SAS offerings and talk with developers. The overall message was that in tough economic times, SAS and predictive analytics is needed more than ever. The keynote speaker, Miami Herald columnist, Dave Barry gave a hilarious take on everything from dogs obsessed with screen doors left by hurricanes to exploding whales in Oregon. As always, we enjoyed meeting with fellow SAS users and customers most of all.

Systems Seminar Consultants, Inc.

Copyright © 2009 Systems Seminar Consultants, Inc. Madison, WI. All rights reserved. Printed in USA. The Missing Semicolon is a trademark of Systems Seminar Consultants, Inc. SAS, SAS/IntrNet, and SAS/ACCESS are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.

SAS Puzzler

Exercise your SAS brain! From each group of words below, find the one SAS term. Answers are on page 8.

Underlay	Lineinput
Truncover	Kaput
Passover	Stayput
Layover	Symput
Nway	Merge
Noway	Combine
Mway	Unite
Pway	Fuse
Coalesce	PROC CORR
Confluent	PROC CONCEAL
Complot	PROC CORRELATE
Cohere	PROC CORRESPOND
Inane	Noodge
Inboard	Noobs
Infile	Nopar
Incept	Norad
Nocol	Desk
Noturn	Armoire
Nolens	Table
Nomad	Chair

New Webinars!

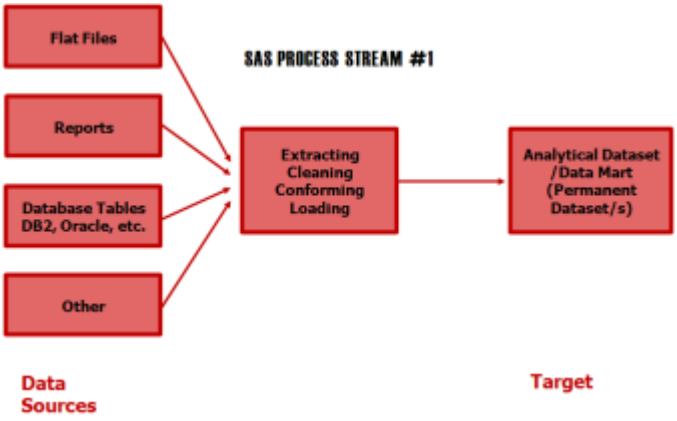
Recorded and Live Webinars Available

SQL Topics
Dimensional Data Modeling
SAS Tips, Tricks, and Techniques

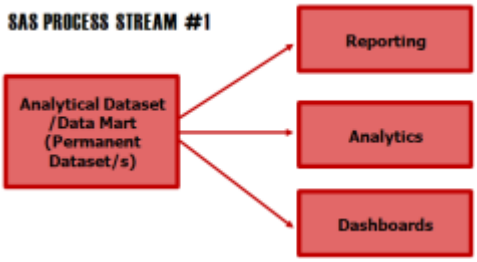
Visit www.sys-seminar.com/webinars.php

Standardize Your Data Preparation

CONTINUED FROM PAGE 1



Second, move the programs that analyze and report on the data to separate processes. This will make your programs easier to maintain.



What can SQL do, and why is it often underused in SAS?

When considering data preparation in SAS, many programmers often use SQL exclusively for pulling data from relational databases, and then use other SAS procedures or the DATA step to finish the ETL process. However, SQL has the ability to join, concatenate, filter, summarize, sort and transform data all in one step.

So why do many of us think about PROC SQL only for data extraction? PROC SQL was introduced in version 6 of SAS. Around this time, the PROC SQL pass-through facility became the preferred way to access relational databases. The code for accessing a relational database like Oracle, DB2, Sybase or SQL server involved a subquery written in the native SQL language and executed on the host database from an outer query executed in SAS. In the example below we are connecting to Oracle, joining two Oracle tables in order to create a SAS dataset called claimData.

```
proc sql;
   connect to oracle ( user=sscdata
                       orapw=sscpw
                         path==ssctrain=);
   create table work.claimData as
   select patientID, patname,
          serviceDate, billAmt
   from connection to oracle
   (select m.patientID,     m.patName,
              c.serviceDate, c.billAmt
     from   members m left
            claims  c
        on m.patientID=c.patientID);
%put &sysdbrc &sysdbmsg;
   disconnect from oracle;
quit;
```

For many SAS users, this would be the end of their use of SQL. Subsequent DATA and PROC steps would be used for additional data preparation, including transformations, merges, summarizations, and sorts. The pass-through syntax is not especially intuitive; so many users ignore SQL's abilities, instead opting for the more familiar PROCs and DATA steps for their data processing. However, the majority of tasks that can be done in the DATA step, PROC SORT, and PROC SUMMARY can also be done in a single PROC SQL step. Let's take a brief look at the main clauses that make up SQL.

SELECT – The SELECT clause is used to list all the columns and transformations that will be a part of the new dataset or query. New columns can be either moved directly from the original source, or created with transformation logic. CASE-WHEN logic (the equivalent of IF-THEN logic) is used to conditionally set values for a new column. All SAS functions (with a few exceptions) are available for use within SAS SQL (all database host functions are available within the pass-through facility query). Example:

```
SELECT custid, case when orderAmt < 100
                    then 'LOW'
                     when orderAmt < 250
                     then 'AVERAGE'
                    else 'LARGE'
               end as transactionSize
```

This example groups each order into one of three categories: LOW, AVERAGE or LARGE. The results of this query result will be two columns: custID and transactionSize.

FROM – Source tables are listed on the FROM clause. If information is needed from more than one table, various types of joins can be employed including FULL, INNER, LEFT and RIGHT.

Continued on next page

The SAS SQL Procedure Class

- Combine the functionality of the DATA and PROC Steps into a single procedure.
- Use PROC SQL to retrieve, update, and report data.
- Learn SQL syntax and use it to access information from existing SAS data sets.

Public, Live Web, or Private Class

To register or find out details,

Call 1-800-997-7081 or

Visit www.sys-seminar.com/sas_training.php

Additionally, various methods of concatenation are available for stacking tables together. Example:

```
FROM  datamart.customers  c  inner  join
      datamart.orders     o  on  c.custid=o.custid
```

This query retrieves only rows with custIDs that are on both the customer and the order tables.

WHERE – WHERE clauses can be used to subset the rows in our source data. As with many of the other clauses, subqueries can be used on the FROM clause to compare individual rows of the source data with individual rows or summarized information from another query. Example:

```
WHERE  state='NY'  and
         custid in (select custid
                    from datamart.orders
                   group by custid
                    having sum(orderAmt) > 5000)
```

This query pulls all customers from NY who have total order amounts greater than 5,000.

GROUP BY – The GROUP BY clause is used to summarize rows by the columns listed. In SQL, the final query result or dataset can contain the same variables as listed on the GROUP BY statement, along with any summary statistics or the data can be both summarized simply to select the detail rows whose summarized value meets a certain criteria (see the HAVING clause). Example of simple summary:

```
SELECT  custid,  SUM(orderAmt)  as  TotalOrderAmt,
                 MEAN(orderAmt) as AvgOrderAmt
FROM  datamart.orders
GROUP  BY  custid
```

This query returns the total and average sale amount by customer.

HAVING – The HAVING clause allows us to subset on summarized information. This is different from the WHERE clause where selection is done on individual rows. Example:

```
SELECT  custid,  SUM(orderAmt)  as  TotalOrderAmt,
                 MEAN(orderAmt) as AvgOrderAmt
FROM  datamart.orders
GROUP  BY  custid
HAVING  MEAN(orderAmt)  >  100
```

This query returns only customers whose average order amount is greater than $100. If we would have applied this filter on the WHERE clause (WHERE orderAmt > 100), the individual orders would have resulted in an understated totalOrderAmt and an overstated AvgOrderAmt.

ORDER BY – The ORDER BY clause allows us to sequence our query results in ascending or descending order. Columns used for sorting do not necessarily need to be on the query results. Example:

```
ORDER  BY  orderAmt  desc
```

The desc option allows for sorting in descending order. Ascending is the default order for columns listed on the ORDER BY statement.

These clauses can be used together to perform an impressive amount of data processing and manipulation all in one logical step. The SQL language is even more extensive than the clauses listed above, but these are the clauses we use most frequently in SAS.

Using SQL for Data Preparation

SAS is a very powerful, flexible, and extensive language. Give 10 programmers the same task, and you are likely to get 10 very different solutions. It is often stated that there is no 'right' way to program or no correct style. However, programmers that inherit legacy systems often spend a lot of time trying to understand the style and syntax of the original author. We see the same issues when using code developed by our co-workers, or even when looking back on our own code from prior projects. Countless hours can be spent reviewing existing code, when debugging or making changes. The problem is exaggerated with all of the unique programming styles that exist.

What if we were able to standardize the way programmers code, so that everyone would not have such drastically different styles? By encouraging or mandating the use of SAS PROC SQL for data preparation whenever possible, programs would look a lot less diverse, and would be easier to maintain and debug.

Many programs can be shortened significantly through the use of SQL. Often many regular PROC and DATA steps can be combined into one powerful SQL query. Though this is not always the case, programs often have shorter run times from the removal of superfluous steps.

As mentioned earlier, many business areas own analytics while data preparation is either developed or passed on to the I.T. department. Having the business areas start the development process in SQL will often make the potential transition to an I.T. department much easier.

SAS SQL is an often underutilized procedure with the ability to make your programs more efficient and standardized. Please watch for our upcoming webinars where we will walk through some data preparation steps using SQL recommended rules and examples.

1 Hour SAS SQL Webinars

Standardize Your Data Preparation in SAS: Use SQL!
May 7, 1-2 PM Central
Complimentary (Registration Limited)

Creating Buckets with SQL
May 26, 1-2 PM Central, $50/person

Visit www.sys-seminar.com/webinars.php

Understanding the SAS® DATA Step

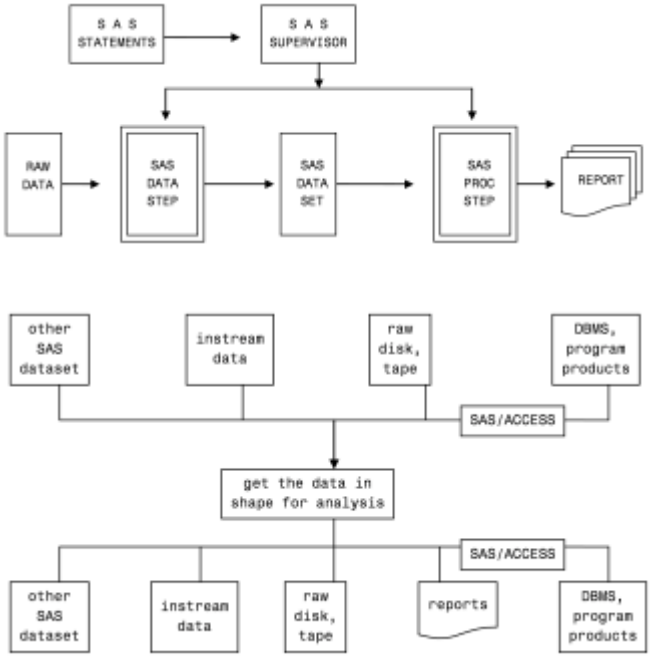
The concepts in SAS's overall design scheme include "self defining files", a system of default assumptions, procedures for commonly used routines, and a data handling step.

The DATA step provides a programming language that allows programs to read and write almost any type of data value, convert and calculate new data, control looping, and much, much more.

This article will discuss the default assumptions. An upcoming article will examine various methods of overriding these defaults.

Purpose of the DATA Step

Oftentimes, when we utilize the SAS system, we first run a DATA step to get our data in shape for analysis and then use the data in a PROC step to provide the analysis. The DATA step gives us the power to read and write virtually any kind of file and do calculations and computations on a single row of data.



Default Assumptions

As SAS DATA step programmers, our job is not only to write logical and efficient programs, but to understand the SAS system defaults and know how to work with them (and override them when necessary - future article). Assumptions are made in all aspects of DATA step processing – in both the compilation phase as well as in the execution phase.

These defaults include:

- Handling the compile and execution naming and storage details for the programmer
- A dataset descriptor that makes SAS datasets "self defining"
- Generating data set names if omitted
- Reading the most recently created dataset if the dataset is not specified

Did You Know We Do Small Projects?

Need something done quickly?

We can help!

- Save Your Staff Time
- Quick Turn Around
- Learn Best Practices

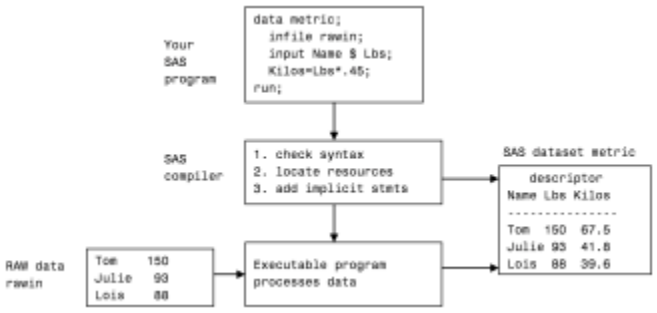
See examples of our projects:
www.sys-seminar.com/project_resume.php

Call 1-800-997-7081 to discuss details!

- Processing all the rows and columns in a file
- Automatically opening and closing of files
- Automatically controlling data initialization, DATA step looping, data set output, and end of file checking
- Automatically defining storage areas for each variable referenced without the need to predefine them
- Assigning a default length of 8 is assumed for all variables
- Assigning a variable type of numeric if not specified
- LIST input assumes that data values would be separated by blanks rather than specifying exact columns (Ex. INPUT Name Years;)
- Using SUBSETTING IF statements to continue processing if a condition is true, else delete the observation (Ex. IF state='WI';)
- Using an IF statement with no comparison operator to assume checking for 1 (true) (Ex. If EOF …)
- Abbreviated sum statements. (Ex. Salestot+sales)

Compiling a DATA Step

The DATA step compiler examines SAS statements for syntax and data structures and then generates an executable program. Distinct from other languages, the SAS compiler checks for the existence of resources and also makes assumptions that it "inserts" into the source code.



Continued on next page

Data Structures

To "get the data in shape", SAS needs data structures to hold the data as it is processed.

Raw File Buffers

If the DATA step is going to read "raw" or non-SAS data, a memory buffer is needed to temporarily hold at least one input record at a time. There are also times when multiple lines of input can be held in buffers. This allows the program to logically read later rows before earlier ones. If the DATA step is writing to a raw file, then similar buffers are needed for each output file. It should be noted that a buffer contains the complete input and output record, regardless of whether the INPUT statement reads all of the columns. When reading SAS datasets and RDMS (which usually appear as SAS datasets), there is no need for raw buffers as the files are already in "shape".

Logical Program Data Vector (PDV)

The DATA step requires a second memory area for:

- Inputting and formatting (informatting) variables
- Modifying existing values
- Computing new variables
- System indicators and flags

This second memory area is called the Logical Program Data Vector (PDV). This is similar to COBOL's Working Storage Area. All variables referenced in the DATA step will be automatically defined in the PDV by the compiler. The characteristics or attributes of the variables are based on the compiler's first reference to that variable. That is, if the statement - `agemo=age/12;` - is the first time the compiler sees the variable AGEMO in the DATA step, AGEMO will be defined using the same characteristics/attributes as AGE, which in this case is numeric.

When the compiler processes the DATA step, it needs to define a slot for each variable referenced in the program. These PDV slots will be defined in the order referenced in the program, and each variable has the following attributes:

- Relative variable number
- Position in the dataset
- Name
- Data type
- Length in bytes
- Informat
- Format
- Variable label
- Flags to indicate dropping and retaining of variables

RETAINed variables, which are DATA step variables that are not automatically initialized on each DATA step pass, are stored separately from the non-retained variables. Even though the variables are not stored contiguously, we can logically consider them as contiguous.

Data Types and Conversion

Although the originating data sources could have different data types, there are only two types of PDV variables: numeric and character. In the PDV, every character value is stored as a native EBCDIC or ASCII value with a length between 1 and at least 32767. Every numeric variable is stored as double precision floating point values with a length between 3 and 8 bytes. Storing only two data types greatly simplifies SAS datasets. The complication of converting different data types (packed, binary, etc.) is handled by the INPUT and PUT statement along with appropriate INFORMATS and FORMATS. The choice of floating point for numbers with a length of 8 allows for storage of very large numbers (though floating point does have minor mathematical issues of its own).

Need A Media Spend Datamart?

Organize your Paid Search, Display, and other Marketing Spend Information

Integrate it with Sales Data for Sales Funnel and R.O.I. Reporting.

Call 1-800-997-7081 to discuss details!

Pseudo Variables

There are several special variables that the compiler creates in the PDV that are not written to the output file, hence the name 'pseudo'. One such variable, _N_, contains the number of times the DATA step has looped. Another, _ERROR_, is set to 0 if there were no input errors, otherwise 1. Several others can be requested by the programmer to indicate when the end of the file is reached (end= dataset option), beginning and ending of by groups (BY statement), access to system control blocks, etc. Many of these variables are switches with values of 0 and 1, others contain longer character values. These pseudo variables can be referenced by the DATA step, but are dropped and do not appear on our output dataset.

Output Datasets

The final structures needed by the DATA step when raw files are being created, are output buffers. These buffers will receive the results of FILE and PUT statements. These structures act conversely to the INFILE and INPUT statements. This time the values are

Continued on next page

being written to a raw/non-SAS file. Though DATA steps usually build SAS datasets, raw files can be extremely useful for passing data to other programs. This makes the DATA step a very versatile tool.

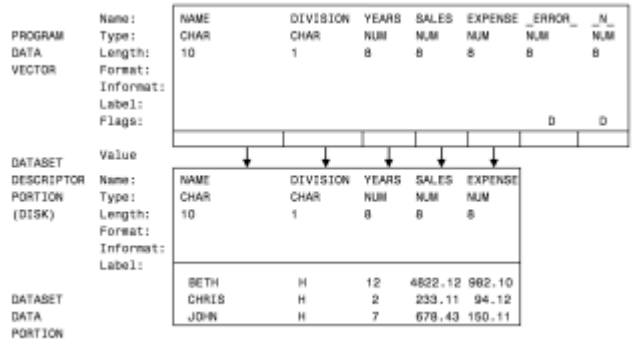
Traditionally DATA steps produce SAS data sets. This is a simple operation for a programmer because SAS automatically builds the structures needed and outputs the observation at DATA step return. All variables, except dropped and pseudo variables, will be included in the output data set. The attributes of the PDV are written to the SAS dataset descriptor which is stored with the SAS file along with the variable values. This descriptor gives subsequent steps all the information they need in order to process the dataset. This allows the programmer to concentrate on results rather than file structure - layouts, data types, etc.
You can use PROC CONTENTS or the SAS dictionary tables (Ex. SASHELP.VCOLUMN) to view the descriptor information, which can serve as dataset documentation.

A Typical SAS Job

Read a raw file and create a SAS data file.

```
                    12345678901234567890123456789012345
input buffer        BETH       H  12  4822.12   982.10
                    CHRIS      H   2   233.11    94.12
                    JOHN       H   7   678.43   150.11
```

```
data softsale;
  infile rawin;
  input name $110 division $12 years 1516
        sales 1925 expense 2734;
run;
```



Questions About the DATA Step

If a traditional non-SAS programmer were to look at the DATA step, some questions that might be asked are:

- Where is the file open and close?
- Where do we write out records?
- How does the looping occur?
- When does the program stop?

Assumptions Made In The DATA Step

To accommodate most programs with the least amount of work, the DATA step makes use of the SAS dataset descriptor and the many assumptions noted previously in this article. The SAS compiler makes the following assumption and inserts code to do the following:

- A DATA step will be entered at the top, and statements will be executed sequentially.
- At the start of each iteration through the DATA step, a check is made to see if any data was read in the previous iteration. If not, the DATA step is stopped.
- If the second iteration detects that no data was read (i.e. an empty dataset/file), the DATA step is stopped with a looping message.
- All values from non-SAS files are cleared before executing any statements.
- If any reading statement would read a record after end of file, the step stops.
- If the program reaches the last statement in the step, or if a RETURN statement is executed, the current contents of the PDV (all columns for each row) is output to the SAS data set being built.
- A branch is executed to go to the top and enter the DATA step for another pass.

In the design of the DATA step:

- All files are automatically opened at the beginning and closed at the end of the program step.
- Input files are read starting with the first record and continuing until the end of the file.
- To eliminate errors and programmer work, all values from a previous record are cleared before processing a new record.
- All records will be included on the resulting output file.
- All variables referenced will be included on the output file.
- SAS Data file definitions are passed automatically from one step to another.
- Programs will not continue to loop if no data is read in a previous pass.

Another way of thinking about it would be as if the compiler inserted the following bold italicized code into our program.

```
DATA softsale;
if there was no input last time thru loop
then stop
initialize PDV
INFILE rawin;
   if at End Of File then stop
  INPUT Name       $1-10
        Division   $12
        Years       15-16
        Sales       19-25
        Expense     28-34
        State      $36-37;
   output to the output SAS Dataset/s

   go back to the top of the DATA step

RUN;
```

Not only do these inserted statements save us work, but they make it virtually impossible for the DATA step to go into an infinite-loop.

Hopefully, this article will clarified the default assumptions in the DATA step. Please watch for the upcoming article on overriding these defaults.

Practical SAS® Applications: Purchase Price Discounts

Shoppers love discounts, and retailers know it. Percent off sales with displays of discounted merchandise are especially common. During many of these large storewide sales, retailers also may also use coupons sent ahead of time to their best customers. These coupons offer an additional percentage discount from the total purchase price. Some lucky customers can leverage both discounts for a cumulative "double discount."

Why are percent off sales and promotional coupons so popular? Customers love the feeling of getting a special deal when they use a promotional coupon. Retailers benefit from percent off coupons by not lowering consumers' price expectations. In other words, if the original price of an item was $45 and the consumer has a 20% off coupon, the final price is $36. Consumers are less likely to remember they paid $36 the next time they are shopping for the item if the discount was obtained with a coupon than if the item had been hard marked.

Many retailers use double discounts to further entice consumers to make a purchase. For instance, a display of items may be marked at 50%, and an additional 15% off can be had by using a coupon. To many individuals, this feels like a larger discount than it really is. Many assume that these two discounts will sum to 65%, when in reality the cumulative discount of 50% and an additional 15% comes to 57.5%.

The final purchase price, much less the true discount percentage, can be cumbersome to figure out. However, both shoppers and retailers need to know what the real cumulative discount will be. SAS can be used to create a dataset containing the individual discount amounts along with the cumulative discount; a report can also be created to easily find the cumulative discount amount from two separate offers. This final table can be used as an aid for retailers making promotion decisions, or it can be joined with product data to calculate the final price on each individual item. Perhaps more importantly to shoppers, it can be used to figure out what the true discount is when going on a shopping spree.

Let's take a look at how this dataset could be created. What is the formula to create the cumulative discount?

```
TotalDiscount=1-((1-discount1)*(1-discount2));
```

A new variable called totalDiscount is created to represent the cumulative discount. This code assumes two other variables already exist in the dataset: discount1 and discount2. What should discount1 and discount2 look like? Usually discounts are in increments of 5%, so all five number increments between 5% and 95% would be appropriate to start. All combinations of these numbers should be used to start with.

Many options are available to create this initial dataset. One easy way, is to use DO loops. Here is the first loop:

```
do discount1=.05 to .95 by .05;
   (SAS code inserted later)
end;
```

Within the first DO loop, the second loop is nested to ensure all combinations of the data are processed. We can also insert our totalDiscount calculation and an OUTPUT statement which will write out an observation to the new SAS dataset called DISCOUNTS.

Continued on next page

Puzzler Answers

```
Truncover
Symput
Nway
Merge
Coalesce
PROC CORR
Infile
Noobs
Nocol
Table
```

SAS has hundreds if not thousands of PROCs, functions, and other options. If you are unfamiliar with any of these terms, check them out on SAS Help. You may find them useful in your programs!

run;

Confusing? Time Intensive? Expensive?
Hard to Change? Error Prone?

Do these words describe your SAS processes?

Our experts can help with an assessment:

- Drastically improve current processes
- Custom training to improve future development

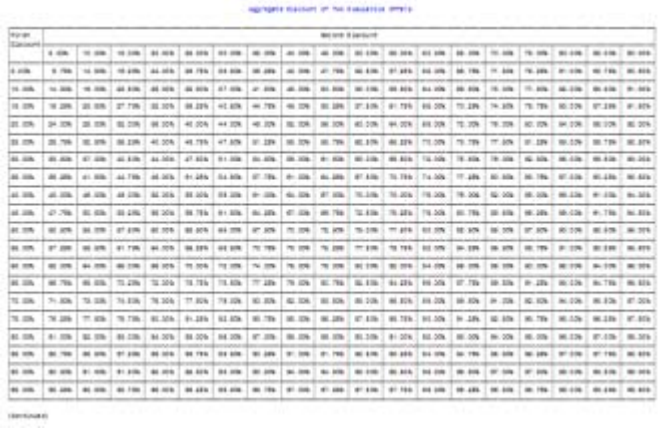
Call for information: 1-800-997-7081 or email consult@sys-seminar.com

```
data discounts;
discount1=.05 to .95 by .05;
   do discount2=.05 to .95 by .05;
         TotalDiscount=1-((1-discount1)*(1-
discount2));
     output;
   end;
end;
run;
```

This dataset of 361 observations will have all combinations of discount1 and discount2. This data is difficult to look through, but PROC TABULATE can create a report with easier to view results. It should be noted that by default, TABULATE produces sums of analysis variables (those listed on the VAR statement). Since we only have one observation for each combination of CLASS variables, summing the variable totalDiscount will not matter.

```
title 'Aggregate Discount of Two Cumulative
Offers';
proc tabulate;
  class discount1 discount2;
  var totaldiscount;
  format discount1 discount2 percent8.2;
   tables discount1=' ',discount2='Second
Discount'*totaldiscount=''*f=percent8.2
          / row=float
             box='First Discount'
            rts=10;
  keylabel sum=' ';
run;
```

This table is painfully small and hard to read. ODS can be a good option to move this report into Excel to utilize the printing and report formatting options.



Simply adding ODS options around the report will move the data to an HTML report which can easily be opened in Excel with the addition of the '.xls' extension.

```
ods html file='c:\temp\cumulativeDiscount3.xls'
style=minimal;
title '<td colspan=20><center><b>Aggregate Discount
of Two Cumulative Offers</b></center></td>';

proc tabulate;
  class discount1 discount2;
  var totaldiscount;
  format discount1 discount2 percent8.2;
  tables discount1=' ',discount2='Second
Discount'*totaldiscount=''*f=percent8.2
          / row=float
             box='First Discount'
            rts=10;
  keylabel sum=' ';
run;
ods html close;
```

HTML tags can be used in the TITLE so less reformatting needs to be done in Excel.

Partial Output:

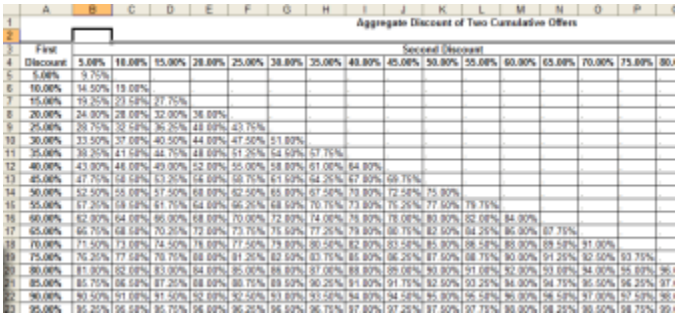
	A	B	C	D	E	F	G
1	Aggregate Discount of Two Cumulative Offers						
2							
3	First	Second Discount					
4	Discount	5.00%	10.00%	15.00%	20.00%	25.00%	30.00%
5	5.00%	9.75%	14.50%	19.25%	24.00%	28.75%	33.50%
6	10.00%	14.50%	19.00%	23.50%	28.00%	32.50%	37.00%
7	15.00%	19.25%	23.50%	27.75%	32.00%	36.25%	40.50%
8	20.00%	24.00%	28.00%	32.00%	36.00%	40.00%	44.00%
9	25.00%	28.75%	32.50%	36.25%	40.00%	43.75%	47.50%
10	30.00%	33.50%	37.00%	40.50%	44.00%	47.50%	51.00%
11	35.00%	38.25%	41.50%	44.75%	48.00%	51.25%	54.50%

After examining the output, it becomes apparent that the order of the discounts does not matter- the cumulative discount will be the same regardless. Because this report is so unwieldy and half the values appear to be redundant, a new approach is in order: the duplicate values can be removed from the dataset and the report can be recreated.

Moving back up to the DATA step, the OUTPUT statement can be tweaked to limit the observations being written out to the DISCOUNT dataset. Removing values of discount1 that are larger than discount2 would take care of the redundancy issue. The change below creates a dataset of 190 observations instead of 391:

```
if discount1 => discount2 then output;
```

By sending this updated dataset through the report, the data that was removed becomes obvious:



Simple SAS DATA steps and quick SAS reports like TABULATE can often be used to provide easy answers to perplexing questions. While this report could have been created in Excel, the SAS dataset might be useful to use in conjunction with pricing data. Regardless, tackling small projects like these in SAS are always helpful in keeping those programming skills sharp!

Webinars

Learn from the Comfort of Your Office!

Over 26 Years of SAS Training Experience

Complimentary Follow Up Help Desk

Our Trainers Are Also Expert Consultants

1-3 Day Live Web Classes

Advanced SAS	May 4-6
Exploiting SAS® ODS	May 7-8
SAS® Macros	May 18-19
Advanced SAS® Macros	May 20
SAS® SQL Procedure	May 21
SAS® Efficiencies	May 22
SAS® Report Writing	June 8-9
PROC Report	June 10
Tips, Tricks, & SAS® Techniques	June 11-12
Introduction to SAS®	June 15-17
SAS® Enterprise Guide	June 18-19

1 Hour Webinars

Standardize Your Data Preparation in SAS: Use SQL!
May 7, 1-2 PM Central
Complimentary (Registration Limited)

Creating Buckets with SQL
May 26, 1-2 PM Central, $50/person

Tips, Tricks, & Techniques from the Experts
Free Recorded Presentation
For more Tips, attend our full 2 day class.

Advantages of Dimensional Data Modeling
Free Recorded Presentation

For questions or registration, contact
Our Training Coordinator at 1-800-997-7081, ext. 306
or visit www.sys-seminar.com

TECHNICAL CREDIT

Steve First
President

Katie Ronk
Director of Operations

Dan Fischer
Consultant/Trainer

Rosalind Gusinow
Senior Consultant/Trainer

Jennifer First
Operations Manager