PharmaSUG2011 – Paper CC25

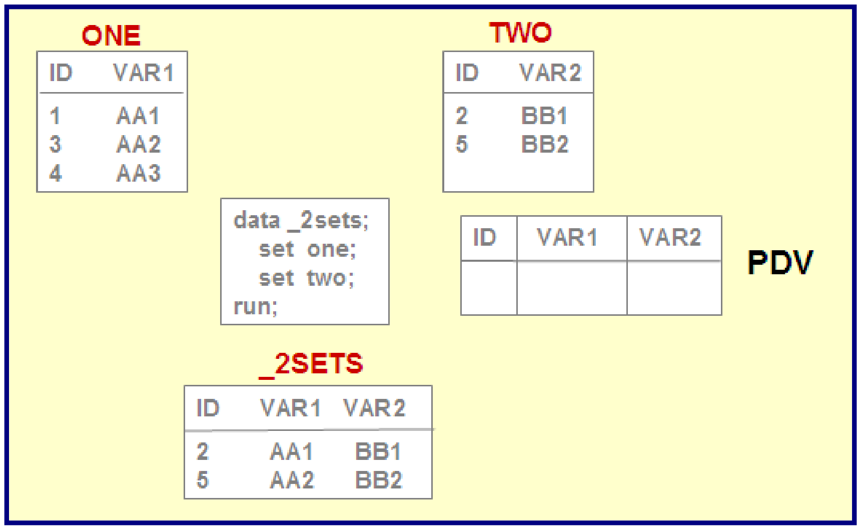
Using Two SET Statements in One DATA Step

Ben Cochran, The Bedford Group, Raleigh, NC

ABSTRACT

It has been said that the DATA step in Base SAS® Software is the most powerful data manipulator in the business. The power and flexibility of the DATA step can be enhanced by using two set statements instead of one MERGE, or UPDATE statement. This Coder's Corner presentation shows a number of examples of when and why you may want to consider doing this. Let us begin.

Example 1: Illustrate how two SET statements work in one DATA step by using a simple example. Each SET statement executes once for each iteration of the DATA step. Every time each SET statements executes, values are read into the Program Data Vector (PDV). But, the resulting dataset (_2SETS) only has two observations. Why? What stops the execution of a DATA step? The answer is 'when a reads operation fails'. In this case, when the second SET statement hits the end-of-file (eof) marker, the DATA step stops immediately. This happens in the third iteration of the DATA step. But the third 'loop' of the DATA step does not finish. The RUN statement at the end of the step is not reached a third time, so there are only two observations written to the 'output' dataset.



ONE

ID	VAR1
1	AA1
3	AA2
4	AA3

TWO

ID	VAR2
2	BB1
5	BB2

```
data _2sets;
  set one;
  set two;
run;
```

ID	VAR1	VAR2

PDV

_2SETS

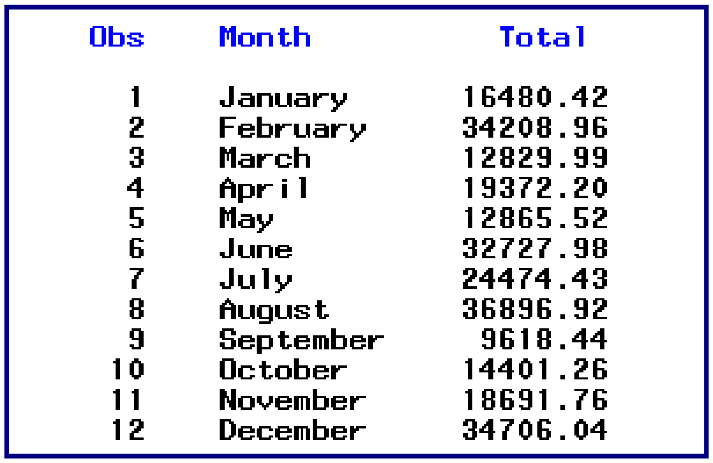
ID	VAR1	VAR2
2	AA1	BB1
5	AA2	BB2

Fig. 1. First DATA Step example.

One of the lessons to be learned here is what happens when a SET statement, any SET statement, reaches the end-of-file marker. In this example, the first SET statement reads an observation into the Program Data Vector (PDV) from the ONE data set. Next, the second SET statement reads an observation into the PDV. The implied OUTPUT statement executes at the end of the DATA step writing the contents of the PDV to the _2SETS data set as the new observation. This continues for two full executions of the DATA step. But, in the third iteration of the DATA step, the first SET statement reads the third observation from the ONE data set, the second SET statement detects the end-of-file (eof) marker and the DATA step stops immediately. The implied OUTPUT statement does NOT execute a third time.

Once we have learned the lesson about a failed read operation stopping the DATA step, we can do things to prevent a SET statement detecting the eof marker.

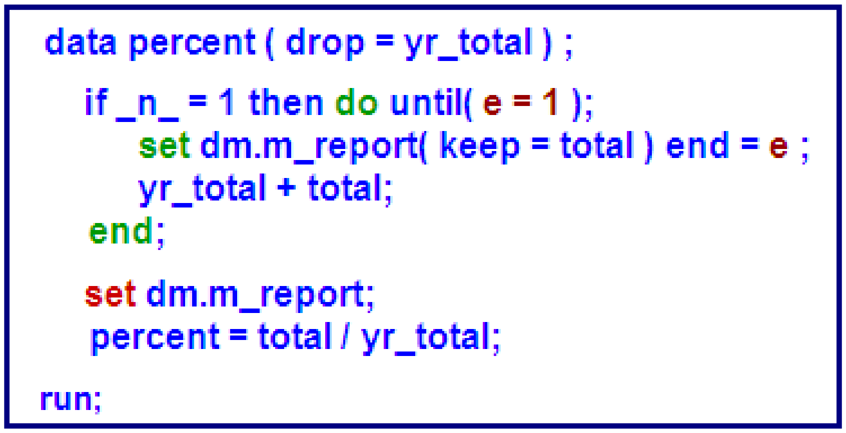
Example 2. Next, a DATA step is used to read the twelve observations from a SAS dataset named M_REPORT (each observation represents a month) and calculate the percentage each row represents of the entire dataset (a year). Before the DATA step is written, it might be a good idea to review some basic math. Look at the dataset below and think about how percentages are calculated.



```
    Obs     Month            Total

      1     January       16480.42
      2     February      34208.96
      3     March         12829.99
      4     April         19372.20
      5     May           12865.52
      6     June          32727.98
      7     July          24474.43
      8     August        36896.92
      9     September      9618.44
     10     October       14401.26
     11     November      18691.76
     12     December      34706.04
```

Figure 2. The M_REPORT dataset.

First, the total for the year is calculated. Then, each monthly value is read and is divided by the total. This can be done with two SET statements in one DATA step as shown in Figure 3 below.



```
data percent ( drop = yr_total ) ;

   if _n_ = 1 then do until( e = 1 );
      set dm.m_report( keep = total ) end = e ;
      yr_total + total;
   end;

   set dm.m_report;
   percent = total / yr_total;

run;
```

Figure 3. DATA step to calculate percentages.

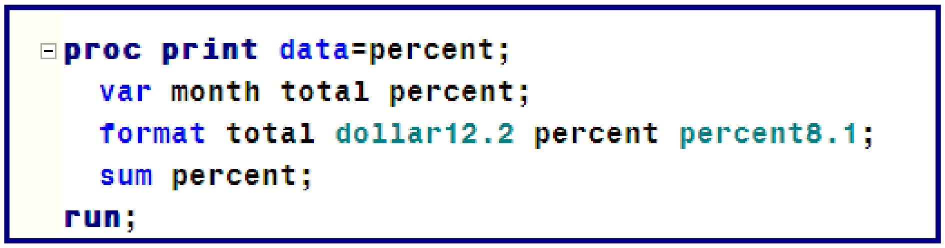
Notice the two SET statements in the above DATA step. The first one (green) is used to generate the total for the year. The second one (red) is used to read each monthly value in order to calculate the percentage that each month (observation) represents of the year.

How many times do we need to calculate the annual total? (Once). When do we want to calculate the annual total? (At the beginning of the DATA step). When you want to do something once at the beginning of the DATA step, use the conditional logic shown above. The special variable _N_ counts the number of times the DATA step begins execution., so the above IF statement will allow us to do something only during the first execution of the DATA step.

Notice the technique used to prevent the SET statement from detecting the eof marker too soon. The END= option creates a variable (in this case E) whose value is 1 when the SET statement has read the LAST observation. This variable (E) is created and given an initial value (0) at compile time. At execution, when the SET statement reads the last observation, E is given a value of 1. This logic is given to the DO UNTIL loop. We are wanting the DO LOOP to execute UNTIL E=1. Since the first SET statement is inside this loop, this loop (including the SET statement executes as long as there are observations to read. Since the condition of a DO UNTIL loop is evaluated at the BOTTOM of the loop, this loop will not execute more times than there are observations to read. So, in other words, this SET statement will never detect the eof marker.

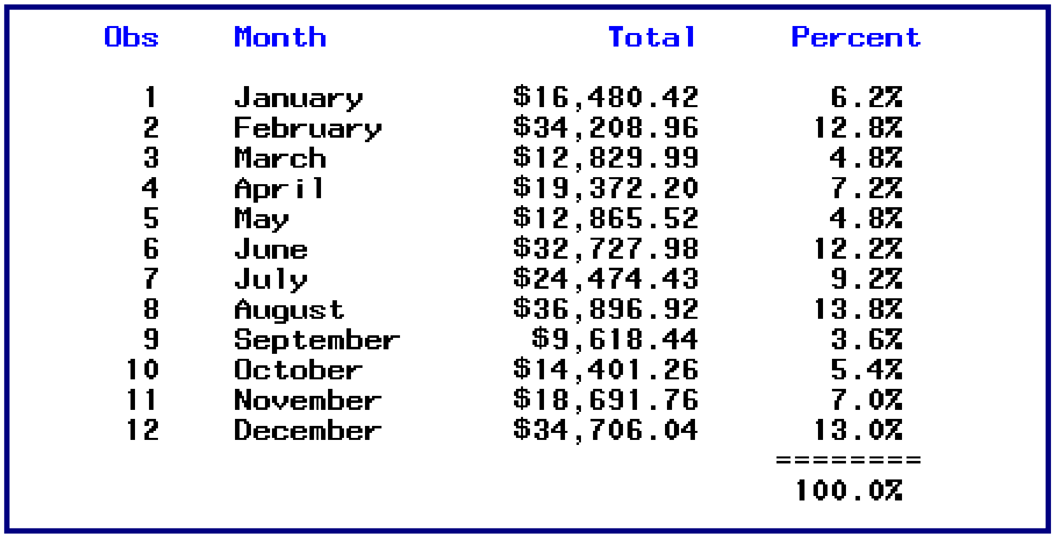
So, the DO LOOP is used to create YR_TOTAL which holds the total amount of TOTAL for all 12 observations. And again, this is only done once (_N_ = 1) at the beginning of the DATA step.

When the DO UNTIL loop ends, and while _N_ = 1, the second SET statement executes reading the first observation from the M_Report dataset. It is important to know that each SET statement has its own independent 'pointer'. So, while the first SET statement's pointer has read all the observations from the dataset, the second SET statement has not as yet executed, so its pointer is still pointing to observation 1. Now, in each iteration of the DATA step, the second SET statement will read an observation and divide the value of TOTAL by YR_TOTAL to get the value for PERCENT. The Proc PRINT output is shown below.



```
proc print data=percent;
   var month total percent;
   format total dollar12.2 percent percent8.1;
   sum percent;
run;
```

Figure 4. Proc PRINT step



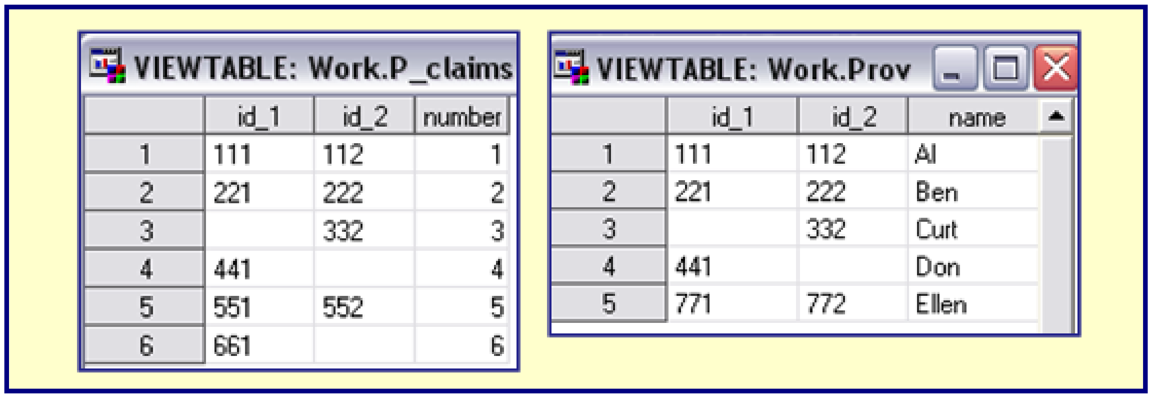
```
     Obs    Month                Total     Percent

       1    January         $16,480.42       6.2%
       2    February        $34,208.96      12.8%
       3    March           $12,829.99       4.8%
       4    April           $19,372.20       7.2%
       5    May             $12,865.52       4.8%
       6    June            $32,727.98      12.2%
       7    July            $24,474.43       9.2%
       8    August          $36,896.92      13.8%
       9    September        $9,618.44       3.6%
      10    October         $14,401.26       5.4%
      11    November        $18,691.76       7.0%
      12    December        $34,706.04      13.0%
                                          ========
                                            100.0%
```

Figure 5. Proc PRINT output.

Example 3. The next example shows how to write a DATA set that will conditionally use an index to perform a table lookup operation. The **PROV** dataset is used to provide the value for NAME. But, depending what is found in **P_CLAIMS** dataset, 1 of 3 indexes will be used. The requirements for the lookup operation are**:**

1. If both **ID_1** and **ID_2** are NON-Missing in **P_CLAIMS,** then use both for matching values with the **PROV** dataset.
2. If **ID_1** is Missing in **P_CLAIMS,** then use **ID_2** only for matching values in the **PROV** dataset.
3. If **ID_2** is Missing in **P_CLAIMS,** then use **ID_1** only for matching values in the **PROV** dataset.

NOTE: **WORK.PROV** has 3 indexes: on ID_1, on ID_2, and one on Both (BOTH).


VIEWTABLE: Work.P_claims

	id_1	id_2	number
1	111	112	1
2	221	222	2
3		332	3
4	441		4
5	551	552	5
6	661		6

VIEWTABLE: Work.Prov

	id_1	id_2	name
1	111	112	Al
2	221	222	Ben
3		332	Curt
4	441		Don
5	771	772	Ellen

Figure 6. Example 3 datasets.

The task is to write a DATA step program that 'conditionally' uses different variables to 'match on' to do the table lookup.

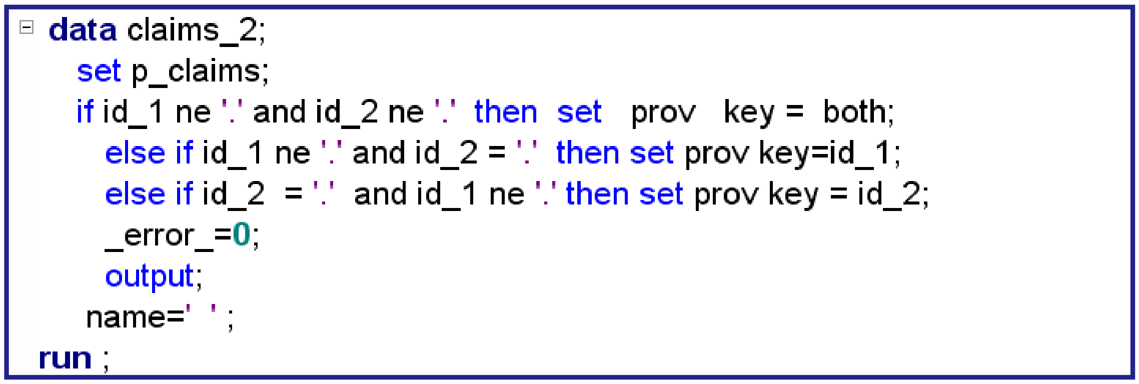

```
data claims_2;
   set p_claims;
   if id_1 ne '.' and id_2 ne '.'  then  set   prov   key =  both;
      else if id_1 ne '.' and id_2 = '.'  then set prov key=id_1;
      else if id_2  = '.'  and id_1 ne '.' then set prov key = id_2;
      _error_=0;
      output;
    name='  ' ;
run ;
```

Figure 7. DATA Step.

Notice the:

1. **KEY =** option on each SET statement. This option names the Index to be used in the lookup operation. Based on values of ID_1 and ID_2, any one of three indexes could be used.
2. **_ERROR_** statement. Without resetting the _ERROR_ variable to 0, the contents of the PDV are dumped to the Log when there is not a match found in the PROV dataset, (_ERROR_=1).

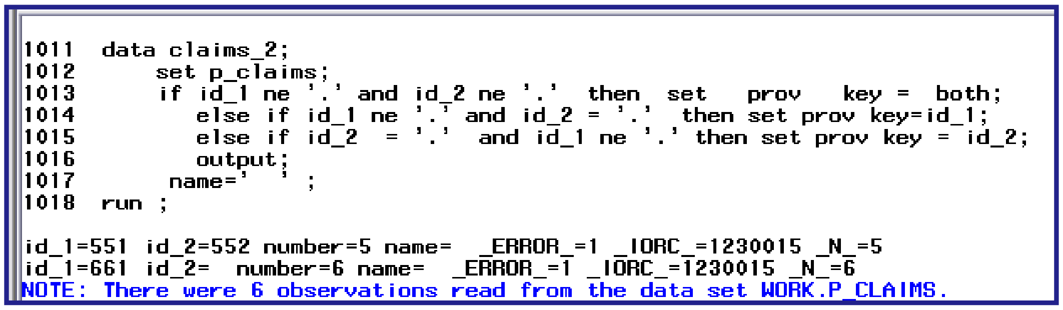
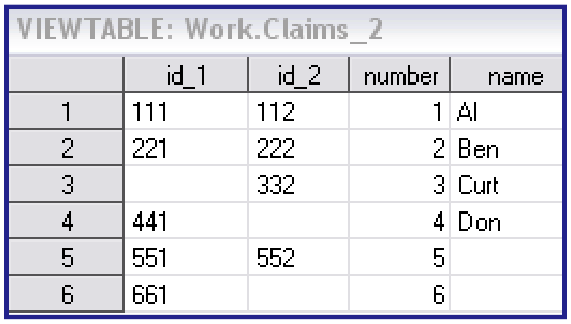

```
1011   data claims_2;
1012       set p_claims;
1013       if  id_1 ne '.' and id_2 ne '.'  then  set   prov   key =  both;
1014          else if id_1 ne '.' and id_2 = '.'  then set prov key=id_1;
1015          else if id_2  =  '.'  and id_1 ne '.' then set prov key = id_2;
1016          output;
1017        name='   ' ;
1018   run ;

id_1=551 id_2=552 number=5 name=  _ERROR_=1 _IORC_=1230015 _N_=5
id_1=661 id_2=  number=6 name=  _ERROR_=1 _IORC_=1230015 _N_=6
NOTE: There were 6 observations read from the data set WORK.P_CLAIMS.
```

Figure 8. SAS Log

The SASLog above shows what would happen if _ERROR_ was not set to 0. We would get a dump of the Program Data Vector. Notice the values of _ERROR_ in the log. THIS IS NOT AN ERROR_ but it is cleaner if we suppress this additional log output.

3. The NAME = ' ' statement at the end of the DATA step. Without this, observations 5 and 6 below would have Don as the value for name. In other words, in an operation like this, NAME is automatically retained. This is not what we want, so we are setting NAME to ' '.



VIEWTABLE: Work.Claims_2

	id_1	id_2	number	name
1	111	112	1	Al
2	221	222	2	Ben
3		332	3	Curt
4	441		4	Don
5	551	552	5	
6	661		6	

Figure 9. The Final dataset

CONCLUSION

By using two SET statements in one DATA step, much flexibility is added to the already powerful DATA step. Oftentimes, SAS programs can be made simpler by this method. In the second example, the same results could be obtained by using a Proc MEANS and then a DATA step. But, with two SET statements in the DATA step, the task can be accomplished in one step. There is an old SAS adage that states if you can write a SAS program in a few steps, it is more efficient than if you accomplish the same results using many steps.

The author can be reached at:

Ben Cochran
The Bedford Group
(919) 741-0370
bedfordgroup@nc.rr.com

sas® | SAS Alliance *Affiliate Member*