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 statement 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.

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.
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.

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

Figure 4. Proc PRINT step

<table>
<thead>
<tr>
<th>Obs</th>
<th>Month</th>
<th>Total</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>January</td>
<td>$16,480.42</td>
<td>6.2%</td>
</tr>
<tr>
<td>2</td>
<td>February</td>
<td>$34,208.96</td>
<td>12.8%</td>
</tr>
<tr>
<td>3</td>
<td>March</td>
<td>$12,829.99</td>
<td>4.8%</td>
</tr>
<tr>
<td>4</td>
<td>April</td>
<td>$19,372.20</td>
<td>7.2%</td>
</tr>
<tr>
<td>5</td>
<td>May</td>
<td>$12,865.52</td>
<td>4.8%</td>
</tr>
<tr>
<td>6</td>
<td>June</td>
<td>$32,727.98</td>
<td>12.2%</td>
</tr>
<tr>
<td>7</td>
<td>July</td>
<td>$24,474.43</td>
<td>9.2%</td>
</tr>
<tr>
<td>8</td>
<td>August</td>
<td>$36,896.92</td>
<td>13.8%</td>
</tr>
<tr>
<td>9</td>
<td>September</td>
<td>$9,618.44</td>
<td>3.6%</td>
</tr>
<tr>
<td>10</td>
<td>October</td>
<td>$14,401.26</td>
<td>5.4%</td>
</tr>
<tr>
<td>11</td>
<td>November</td>
<td>$18,691.76</td>
<td>7.0%</td>
</tr>
<tr>
<td>12</td>
<td>December</td>
<td>$34,706.04</td>
<td>13.0%</td>
</tr>
</tbody>
</table>

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

```sas
% 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;
```

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;
```

Figure 8. SAS Log

The SAS Log 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](image)

<table>
<thead>
<tr>
<th>id_1</th>
<th>id_2</th>
<th>number</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>111</td>
<td>112</td>
<td>Al</td>
</tr>
<tr>
<td>2</td>
<td>221</td>
<td>222</td>
<td>Ben</td>
</tr>
<tr>
<td>3</td>
<td>332</td>
<td></td>
<td>Curt</td>
</tr>
<tr>
<td>4</td>
<td>441</td>
<td>4</td>
<td>Don</td>
</tr>
<tr>
<td>5</td>
<td>551</td>
<td>552</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>661</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

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® is a registered trademark of SAS Institute, Inc., Cary, NC.