MWSUG 2017 - Paper BB114

Tackling Unique Problems Using TWO SET Statements in ONE DATA Step

Ben Cochran, The Bedford Group, Raleigh, NC

ABSTRACT

This paper illustrates solving many problems by creatively using TWO SET statments in ONE DATA step. Calculating percentages, Conditional Merging, Conditionally using Indexes, Table Lookups and Look ahead operations are investigated in this paper.

INTRODUCTION

This is not the same as placing TWO datasets on ONE SET statement, this paper is all about using TWO SET statements with ONE dataset each in ONE DATA step. This paper shows several examples of many data manipulation tasks that can be done with two set statements in one DATA step. First, let's look at a simple example of a DATA step with TWO SET statements.

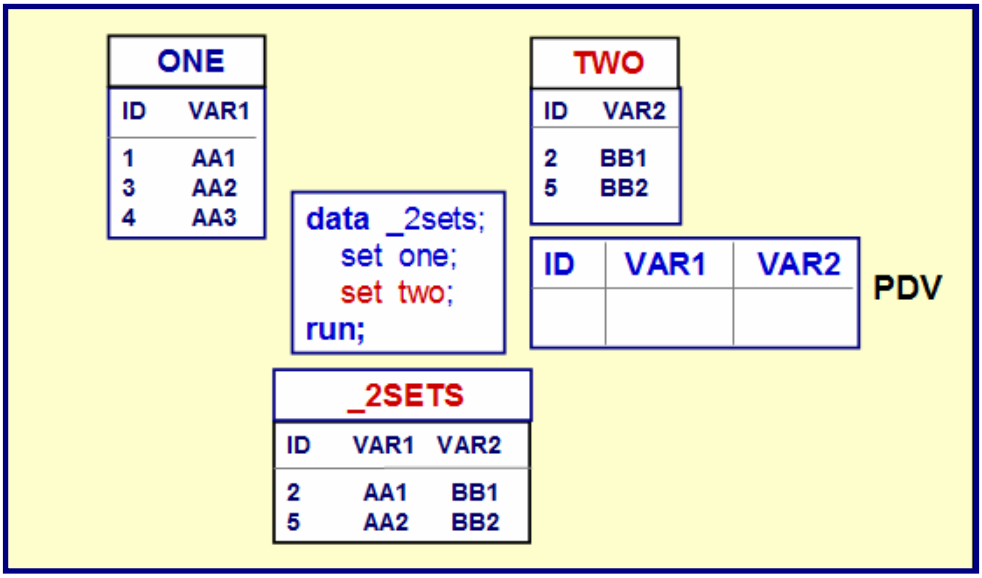

Figure 1. Introductory Example

As in all DATA steps, by default the statements are executed sequentially from top to bottom. In each execution of the DATA step, the first SET statement reads an observation from the ONE dataset and writes it's values into the program data vector (PDV). Next, the second SET statement executes and reads an observation from the TWO dataset and likewise writes it's values into the PDV. The results are displayed in the output dataset _2SETS. One question that arises is why are there only 2 observations in the output dataset when clearly there are three observations to read from the ONE dataset. What stops the execution of this DATA step? The textbook answer is when a read operation fails. Because this DATA step has two SET statements, it has two read operations per execution of the DATA step. A DATA step stops when ANY read operation fails. So, in the third execution of this DATA step, the first SET statement successfully reads the third observation from the ONE dataset, but when the second SET statement tries to read data, it detects the end-of-file marker (a read operation that fails) and stops immediately.

So, this is one of the primary concerns when using more that one SET statement in a DATA step, preventing a SET statement from prematurely ending the DATA step. Be very aware of this as we go through the following examples.

EXAMPLE 1: CREATING PERCENTAGES

This example uses the following data which contains monthly sales figures.

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. Monthly Sales Data

The Vice President of Sales wants to know what percentage each month's sales represents of the yearly total. The next task is to write a DATA step to read the **Monthly Sales** data set and generate a monthly sales report that includes the percentage that each months sale represents of the yearly total. And, this will be done in ONE DATA step.

```
data percent ( drop = yr_total ) ;
   if _n_ = 1 then do until( e = 1 );
      set m_report( keep = Total ) end = e ;
      Yr_Total + Total;
   end;
   set m_report;
   Percent = Total / Yr_Total;
run;
```

Figure 3. DATA step.

In calculating monthly percentages, the first task is to calculate the YEARLY TOTAL. The next task is to divide each month's sales by the yearly total. In this DATA step, the first SET statement (green) is used to calculate the yearly total. Notice the syntax. The first SET statement is used inside a DO LOOP. This DO LOOP is only executed during the first iteration (execution) of the DATA step. Notice the IF condition: If _N_ = 1, that will happen only one time, in the first iteration of the DATA step. Look at the DO UNTIL condition: e=1. When does E equal 1? Notice that the special variable E is created with the END= option on the first SET statement. E is a variable who's value will become 1 when the first SET statement reads the last observation. This logic prevents the first SET statement from going past the last observation to read, and thereby prevents it from detecting the end-of-file marker. The work of the first SET statement is done only in the first iteration of the DATA step. Next, the program's logic causes it to fall out of the loop.

Since the second SET statement has not yet executed, it is still pointing to the first observation. (Each SET statement has its own set of pointers).

This DATA step will execute twelve times. In each 'loop', the monthly percentage is calculated. We the second SET statement detects the end-of-file marker, the DATA step stops immediately.

A PROC PRINT step is executed next, and the output is shown on the next page.

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 4. The output.

EXAMPLE 2: CONDITIONALLY CHOOSING AN INDEX

This example not only shows conditionally choosing an index, but is also a table lookup example.

The **P_CLAIMS** dataset does not have a NAME field. The **PROV** dataset has NAME and is used to supply this value in a Lookup operation.

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 5. The CLAIMS and PROVIDER datasets

The **WORK.PROV** has **3 indexes:** on ID_1, on ID_2, and one on Both (BOTH). Note: If an index is based on a single variable, the index name is the same as the variable name. If an index is based on more than one variable, the name is provided the programmer. Here, an index called BOTH is created based on two variables: ID_1 and ID_2.

Here are the requirements for index usage:

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

So, the task becomes, how do you write a DATA step program that **'conditionally'** uses different variables to match on to do the table lookup? The solution involves using a KEY = option on the second SET statement. The KEY= option forces SAS® use an index.

```
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 ne '.' and id_1 = '.' then set prov key=id_2;
    _error_ = 0;
    output;
    name=' ';
 run;
```

Figure 6. DATA step using a KEY= option.

Notice the:

1. **KEY =** option on each SET statement that reads the PROV (PROVIDER) dataset.
2. **_ERROR_** statement setting the value to 0.
3. The **NAME = ' '** statement at the end of the DATA step.

Notice that the execution of the SET statements that read the PROV dataset are controlled with IF / THEN logic. Each of these SET statements use the KEY= option, but they are referencing a different index. So, consequently we are using conditional logic to specify which index will be used.

Without resetting the _ERROR_ variable (_ERROR_ =0 ;), the contents of the PDV are dumped to the SASLog when there is not a match found in the PROV dataset; (because _ERROR_ = 1). This is NOT an error, but the LOG is 'cleaner' when we 'reset' the _ERROR_ variable to 0.

Without resetting the value of NAME to ' ' then Don would appear on the last two observations.

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 7. The CLAIMS_2 Dataset.

The SAS Log shows what would happen if _ERROR_ = 0 is not used. Notice the dumping of the PDV.

Log - (Untitled)

```
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. The SAS Log.

EXAMPLE 3: LOOK AHEAD

A certain rental agency keeps their **housing** data in a SAS data set shown below. (Each row represents a transaction per housing unit). This dataset is used to generate a series of vacancy reports. The dataset is sorted by Unit_No then DATE. Only the first nine observations are shown here.

Obs	Unit_No	Status	Date	Comment
1	101	V	02JAN2009	Resident moved out
2	101	R	04JAN2009	Plumbing
3	101	P	08JAN2009	Painting
4	101	O	22JAN2009	New Resident moved in
5	101	V	30JAN2009	Resident moved out
6	101	O	11FEB2009	New Resident moved in
7	201	V	01JAN2009	Resident moved out
8	201	R	03JAN2009	Minor Repairs done
9	201	O	07JAN2009	New Resident moved in

Figure 9. Input dataset

They need to convert this data into a report shown below. The report needs to show one row per unit per vacancy. Specifically, the agency wants to know how long was the unit vacant. The agency cant make money on empty units.

Obs	Unit_No	Vacancy_Start	Vacancy_End	Days_Vacant
1	101	02JAN2009	22JAN2009	20
2	101	30JAN2009	11FEB2009	12
3	201	01JAN2009	07JAN2009	6

Figure 10. Resulting dataset

One of the keys to doing this task is to examine the values of **STATUS. V** means vacated, **R** means repairs were done, **P** means painting was done, and **O** means the unit has become occupied again.

Since the goal is to calculate how long each unit has been empty, the approach taken here will be as follows:

1. Read each row until the status is V (for vacant).
2. Capture the date - (when the unit becomes empty).
3. Continue to read rows for that unit until status becomes O (for occupied).
4. Capture the date - (when the unit becomes occupied).
5. Calculate the number of days between the Vacant Date and the Occupied Date.
6. Do this for each unit.

One tool that is used in this example that will make the job a lot easier is the **POINT=** option on the SET statement.

By default, the SET statement reads observations from a data set in sequential order. The **POINT=** option on the **SET** statement allows a user to access any observation from a data set and in any order. This is commonly known as **direct access** read mode. The typical syntax of the point= option is:

SET *SAS dataset* **POINT =** *variable*. . . **;**

The *variable* **:**

- is named by the user**,**
- is a **numeric** variable **,**
- contains the number of the observation to read**.**

Now that we know about the point= option, let's write the DATA step to accomplish our task.

```
data vacant_time(drop= Status Comment Date);
  set houses;
  if status='V' then do;
     grabit=_n_;
     Vacancy_Start = date;
     do until(status='O');
        grabit + 1;
        set houses(keep=date status
            rename=(date=Vacancy_End)) point=grabit;
     end;
  end;
    Days_Vacant = Vacancy_End - Vacancy_Start;
    if Days_Vacant ne . then output;
 run;
```

Figure 11.

The first SET statement reads a row at a time, starting with row one. The value of STATUS is examined. If it is a **'V',** then **GRABIT** is set to this row number *. Then one is added to the row number, and the second SET Statement reads the next row.... and continues reading a new row until a value of 'O' is found for STATUS. The POINT= option allows the second SET statement to read a specific row number.

GRAB_IT is set to the value of **_N_** (an automatically created variable) which counts the number of times the DATA step begins execution. In this example, for each execution of the DATA step, an observation is read from the **houses** dataset.

The first DO group is entered if the value of STATUS=V. Once that happens, the DO Until Loop executes until a row is read where the value of STATUS = 'O'. Once that happens, the program logic 'falls' out of the LOOP and the number of days vacant is calculated. To see the report, PROC PRINT is used.

```
proc print data=vacant_time;
     format Vacancy:  date9.;
run;
```

Obs	Unit_No	Vacancy_Start	Vacancy_End	Days_Vacant
1	101	02JAN2009	22JAN2009	20
2	101	30JAN2009	11FEB2009	12
3	201	01JAN2009	07JAN2009	6

Figure 12. The Report.

What if a unit does not get a new tenant? The number of vacant days needs to be calculated from the time the unit becomes vacant until the day the program is run.

EXAMPLE 4: LOOK AHEAD - PART II

Lets take a quick look at the data used to create the report for this example.

Obs	Unit_No	Status	Date	Comment
1	101	V	02JAN2009	Resident moved out
2	101	R	04JAN2009	Plumbing
3	101	P	08JAN2009	Painting
4	101	O	22JAN2009	New Resident moved in
5	101	V	30JAN2009	Resident moved out
6	201	V	01JAN2009	Resident moved out
7	201	R	03JAN2009	Minor Repairs done
8	201	O	07JAN2009	New Resident moved in

Figure 13. The Dataset.

In this scenario (dataset is HOUSES2) , UNIT_NO 101 becomes vacant (for the second time) on 30Jan2009, and has been vacant ever since. Management needs to know how long the unit has been vacant. The calculation for this is the number of days between 30Jan2009 and the day the program is run **(April 6, 2009).**

The previous DATA step will be modified to accomplish this task.

```
proc sort data=houses2;
   by unit_no date;
run;

data vacant_time2(drop= Status Comment Date);
   set houses2;
   by unit_no;
   if status='V' then do;
      grabit=_n_;
      Vacancy_Start = date;
      if last.unit_no ne 1 then do;
         do until(status='O' or last.unit_no=1);
            grabit + 1;
            set houses2(keep=date status unit_no
                rename=(date=Vacancy_End)) point=grabit;
         end;
      end;
      if last.unit_no=1 then Vacancy_End=today( );
   end;
   Days_Vacant = Vacancy_End - Vacancy_Start;
   if Days_Vacant ne . then output;
run;
```

Figure 14. The DATA Step for Example 4.

In this scenario, we need to know when we have read the last observation for each unit. So, the first thing we need to do is to **sort** the dataset by **UNIT_NO.** The DATA step uses a **BY** statement (by unit_no;). This creates 2 variables for the DATA step to use (first.unit_no and last.unit_no). **Last.unit_no** has a value of **1** when the last row for each unit_no is read**.**

The modified logic is seen in the inner most DO UNTIL where the condition is: STATUS='O' or LAST.UNIT_NO = 1. This tells the loop to execute until the status is occupied, or the last row for that unit has been read (the assumption being that it has not been occupied again since all the rows for that unit have been read and the unit was not occupied again).

The PROC PRINT step generates the final report.

```
proc print data=vacant_time2;
     format Vacancy:  date9.;
run;
```

Obs	Unit_No	Vacancy_Start	Vacancy_End	Days_Vacant
1	101	02JAN2009	22JAN2009	20
2	101	30JAN2009	06APR2009	66
3	201	01JAN2009	07JAN2009	6

Figure 15. The Final Report

CONCLUSION

Some people say that the SAS DATA step is the best data manipulator in the business. There are dozens and dozens of options and structures (DO LOOPS, etc) that give the DATA step tremendous power and flexibility. Some of this flexibility and power is shown in this paper. I hope that the user will gain a deeper appreciation of some of this power and flexibility by using TWO SET statements in ONE DATA step.

ACKNOWLEDGMENTS

I would like to thank one of my clients from several yeas ago for posing some of the challenges discussed in this paper. Their challenge has been the source of inspiration for this presentation. And, as always, I want to thank the kind and patient folks in the Technical Support division at SAS. Without their help, this paper would not exist.

CONTACT INFORMATION

If you have any questions or comments, the author can be reached at:

Ben Cochran
The Bedford Group
3224 Bedford Avenue
Raleigh, NC 27607
Work Phone: 919.741.0370
Email: bencochran@nc.rr.com