PharmaSUG 2018 - Paper BB-10

Exploring techniques of using multiple SET statements in a DATA step

Yizhong (John) Huang, Celgene Corporation

ABSTRACT

SET statement is one of the most used statements in SAS programs. Yet, some powerful features of SET statement have been widely ignored. Usually, PROC SQL or MERGE statement in a DATA step will be used to handle “One-to-One” or “Many-to-One” merge. When there is “Many to Many” merge, PROC SQL procedure seems to be the only choice for SAS programmers. This is NOT true! Using multiple SET statements in a DATA step, this paper will explore implementations of "Many-to-Many" merge compared to PROC SQL joins, as well as other techniques that give you more flexibility and make your programs more efficient.

INTRODUCTION

This paper will explore how to correctly utilize multiple SET statements in DATA step and to compare “Many to Many” joins between DATA step and PROC SQL. It will also present some examples of complex tasks using multiple SET statements techniques.

1. SET STATEMENT SYNTAX

By “SAS 9.4 DATA Step Statements: Reference”, SET statement reads an observation from one or more SAS data sets. Its syntax:

SET <SAS-data-set(s)> <(data-set-options(s))>> <options>;

There are three options that will be used in the following multiple SET statements discussions and examples.

1.1 END=variable

Creates and names a temporary variable that contains an end-of-file indicator. The variable, which is initialized to zero, is set to 1 when SET reads the last observation of the last data set listed. This variable is not added to any new data set.

1.2 NOBS=variable

Creates and names a temporary variable whose value is usually the total number of observations in the input data set or data sets.

1.3 POINT=variable

Specifies a temporary variable whose numeric value determines which observation is read. POINT= causes the SET statement to use random (direct) access to read a SAS data set.

2. HOW DO MULTIPLE SET STATEMENTS WORK IN DATA STEP?

We know that, one SET statement in DATA step will read one observation at a time from one or more data set(s). When there are multiple SET statements in a DATA step, how will those SET statements work? When will the DATA step stop? Here is a simple example to give you some ideas.

Codes:

```
data res;
   step=_n_;
   set dsn1 end=eof1;
   put step= ID1= Name= eof1=;
   set dsn2 end=eof2;
   put step= ID2= Age= eof2=;
run;
```

data set DSN1

ID1	NAME
11	Joe
12	Sarah
13	Ben

data set DSN2

ID2	AGE
11	25
13	36

data set res

STEP	ID1	NAME	ID2	AGE
1	11	Joe	11	25
2	12	Sarah	13	36

Outputs of PUT statements in logs

```
step=1 ID1=11 Name=Joe eof1=0
step=1 ID2=11 Age=25 eof2=0
step=2 ID1=12 Name=Sarah eof1=0
step=2 ID2=13 Age=36 eof2=1
step=3 ID1=13 Name=Ben eof1=1
```

Last observation of data set dsn2

Example 1 – Simple DATA step with multiple SET statements

In the Example 1, there are two SET statements. The first SET statement reads in data set dsn1 which has 3 observations. The second SET statement reads in data set dsn2 which has 2 observations. There is one PUT statements right after each SET statement to print out the read data.

In the logs, outputs of PUT statements shows that, after reading in two observations from DSN1 and DSN2, DSN2 reaches the last record and end-of-file indicator (variable eof2) is set as 1. DATA step continues next iteration, which reads the third observation of DSN1, then executes the statement of SET DSN2. With DSN2's end-of-file indicator as 1, DATA step stops without executing DSN2's PUT statement and output of the third observation of DSN1. So, the output data set RES has only two observations.

The output data set RES does like merging two data sets, dsn1 and dsn2, together. One observation from DSN1 (variables ID1 and Name) and one from DSN2 (variables ID2 and Age) are combined and output to data set RES, which is like one-to-one match by data sets' row ID.

Such approach in the Example 1 is NOT practically useful: 1) It requires that observations in data sets are perfectly matched by row IDs; 2) extra observations are fine to be truncated.

So far, I didn't get any chance to apply simple multiple SET statements in DATA steps to real data yet.

3. CARTESIAN PRODUCT BY MULTIPLE SET STATEMENTS

From previous discussions, multiple SET statements do merge observations from multiple data sets. If, in each iteration of DATA step, we can make one SET statement read one observation a time, other SET statement(s) read all observations in a loop, and output combined observations, then we will get Cartesian Product results.

With “POINT=” option, a SET statement will use random (direct) access to read a SAS data set. One benefit of random access is that it would not flag the end-of-file indicator, even a SET statement does read the last observation of the input data set or data sets. This is very important, because the DATA step won't stop when executes this SET statement again. Also, SET statement's option 'NOBS=' will give you the total number of observations, which will make random access easier. With knowing the total number of observations, we can use “POINT=” option to read data set from first observation to last observation one by one in a loop.

Here is the example provided by SAS Institute to generate every combination of observations between data sets (with renaming data set names in the sample codes):

```
data every_combination;
  set DSN1;
  do i=1 to n;
    set DSN2 point=i nobs=n;
    output;
  end;
run;
```

Example 2 - Generate every combination of observations between data sets

In the Example 2, for each iteration of DATA step, the first set statement, “set DSN1;”, will read one observation from data set DSN1 into the PDV. A DO loop (do i=1 to n;) contains the second SET statement, “set DSN2 point=i nobs=n;”, and an OUTPUT statement. With the temporary variable of “point=” option serving as DO loop's index variable (variable i) and the temporary variable of “nobs=” option serving as end value (variable n) of the DO loop, the DO loop will read in observations of data set DSN2 one by one combining with the observation read from data set DSN1, and output to data set every_combination.

Below is the result of data set every_combination:

ID1	Name	ID2	Age
11	Joe	11	25
11	Joe	13	36
12	Sarah	11	25
12	Sarah	13	36
13	Ben	11	25
13	Ben	13	36

With Cartesian Product results, we can easily generate many-to-many merge results.

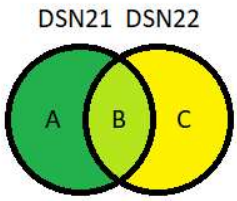
4. IMPLEMENTATIONS OF SQL'S JOINS BY MULTIPLE SET STATEMENTS IN A DATA STEP

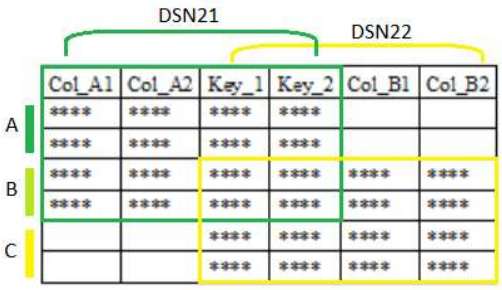
In SQL world, joins are done by Cartesian Product. Just like the results of Example 2, the output data set will contains all combinations of every observation in first data set and every observation in second data set. In another way, if first data set has n observations and second data set has m observations, the results of Cartesian Product will have n*m observations.

In SAS DATA step, data merge/join are usually done by MERGE-BY statement. If BY variables of data sets are One-to-One or One-to-Many match, MERGE-BY in DATA step will generate similar results as Cartesian Product. But MERGE-BY statement in DATA step doesn't utilize Cartesian Product approach. If two data sets are many-to-many match, for example, MERGE-BY subject IDs, and for one subject, there are two observations in both data sets, with Cartesian Product, we will get 4 observations (2*2=4) for that subject; with MERGE-BY, we will get only 2 observations for that subject, and SAS system will give a note, “MERGE statement has more than one data set with repeats of BY values.”. Such results usually are not what we want. So, when we have Many-to-Many joins, we usually turn to PROC SQL which will lose benefits of powerful DATA step.

Below, we will demonstrate that, with multiple SET statements, we can use DATA step to merge multiple data sets and generate same results as PROC SQL even with Many-to-Many join.

There are two types of Joins in SQL, Inner Join and Outer Join. Inner Join means returning only results from both data sets meeting the selection criteria. Outer Join can be Left Join, Right Join, and Full Join. Depending on different types of Outer Joins, results will be from one data sets (master data set) combining all or part of another data set. See the illustration below.





Two data sets, DSN21 and DSN22, join by certain criteria (in the illustration, Key_1 and Key_2).

Inner Join: Part B, which combines both observations from DSN21 and DSN22.

Outer Left Join or Left Join: Part A from DSN21 + Part B from both DSN21 and DSN22; DSN21 is the master data set.

Outer Right Join or Right Join: Part C from DSN22 + Part B from both DSN21 and DSN22; DSN22 is the master data set.

Outer Full Join or Full Join: Part A from DSN21 + Part C from DSN22 + Part B from both DSN21 and DSN22.

To demonstrate the implementations by multiple SET statements in DATA step, we use the following two simple data sets. Both data sets have duplicated ID values (ID=1001).

Data set DSN21

ID	ARM
1001	TRT1
1001	GRP1
1002	TRT2
1003	TRT3
1003	GRP1

Data set DSN22

ID	EVENT
1001	Headache
1001	Nausea
1002	Headache
1002	Rash
1004	Fever

4.1 IMPLEMENTING INNER JOIN

Data Step with Multiple SET statements

Codes:

```
DATA inner1(drop=_ID);
   SET DSN21;
   DO i=1 TO n;
      SET DSN22(rename=(ID=_ID))
               point=i nobs=n;
      if ID=_ID then output;
   END;
RUN;
```

selection criteria

PROC SQL

Codes:

```
PROC SQL;
   CREATE TABLE inner2 AS
   SELECT a.ID, a.ARM, B.EVENT
   FROM DSN21 a inner join DSN22 b
      ON (a.ID=b.ID)
   ;
QUIT;
```

Output data set inner1

ID	ARM	EVENT
1001	TRT1	Headache
1001	TRT1	Nausea
1001	GRP1	Headache
1001	GRP1	Nausea
1002	TRT2	Headache
1002	TRT2	Rash

Output data set inner2

ID	ARM	EVENT
1001	TRT1	Headache
1001	GRP1	Headache
1001	TRT1	Nausea
1001	GRP1	Nausea
1002	TRT2	Headache
1002	TRT2	Rash

Example 3 – Inner Join

In the Example 3, the codes of DATA step for Inner Joint are very similar to the codes of generating every combination of observations (Example 2). One IF-THEN statement is added to output observations when two IDs are matched (DSN21.ID=DSN22.ID). In the output data sets (inner1 and inner2), only ID values in 1001 and 1002 are in final data sets, because these are common key values in both data sets, DSN21 and DSN22.

In the second SET statement of DSN2, variable DSN2.ID should be renamed, so that its values won't overwrite values of DSN1.ID.

Using PROC SQL, INNER JOIN is defined in FROM clause between two data sets, and ON expression to define matching criteria.

Compared the output data sets of PROC SQL and DATA step with multiple SET statements, they are almost identical, except sorting order.

4.2 IMPLEMENTING LEFT JOIN

Multiple SET statements
Codes:

```
DATA left1(drop=_ID hasDSN22);
   SET DSN21;
   DO i=1 TO n;
      SET DSN22(rename=(ID=_ID))
               point=i nobs=n;
      if ID=_ID then do;
         hasDSN22=1;
         output;
      end;
   END;
   if hasDSN22 ne 1 then do;
      EVENT="";
      output;
   end;
RUN;
```

Selection Criteria

set variable EVENT as blank/null

PROC SQL
Codes:

```
PROC SQL;
   CREATE TABLE left2 AS
   SELECT a.ID, a.ARM, B.EVENT
   FROM DSN21 a left join DSN22 b
      on (a.ID=b.ID)
   ;
QUIT;
```

Output data set left1

ID	ARM	EVENT
1001	TRT1	Headache
1001	TRT1	Nausea
1001	GRP1	Headache
1001	GRP1	Nausea
1002	TRT2	Headache
1002	TRT2	Rash
1003	TRT3	
1003	GRP1	

Output data set left2

ID	ARM	EVENT
1001	TRT1	Nausea
1001	GRP1	Nausea
1001	TRT1	Headache
1001	GRP1	Headache
1002	TRT2	Headache
1002	TRT2	Rash
1003	GRP1	
1003	TRT3	

Example 4 – Left Join

Compared to Inner Join, Left Join outcomes should contain the observations in data set DSN21 but not in data set DSN22 by selection criteria, besides the results of inner join, because data set DSN21 is the master data set in the left join.

In the Example 4, to implement Left Join (compared to Inner Join), the codes above add a flag variable hasDSN22, which has initial value null. If there is a match of ID between DSN21 and DSN22, the flag will be set to 1, otherwise, it will remain as null after the DO loop. So, if after the DO loop, variable hasDSN22 has value 1, we know the ID of DSN21 has found match ID in DSN22, and matched data had been output to data set left1. Otherwise, variable hasDSN22 will remain null, the codes will set variables from data set DSN22 as null and output mismatched DSN21 data.

Same as in the Example 3 (inner join example), variable DSN2.ID should be renamed, so that its values won't overwrite values of DSN1.ID.

PROC SQL codes are almost same as INNER JOIN, except replacing INNER JOIN with LEFT JOIN.

Compared the output data sets of PROC SQL and multiple-SET DATA step, they are almost identical, except sorting order.

4.3 IMPLEMENTING RIGHT JOIN

In SQL, Right Join and Left Join have same concept, only difference is that master table is on the right side or left side of two tables. When implementing Right Join by DATA step, the codes are similar to Left Join, except that we need to switch two data set names (switching the master data set).

4.4 IMPLEMENTING FULL JOIN

In SQL, compared to Left/Right Join, Full Join has two master tables/data sets, which means mismatched data from both data sets will be included in output data set. There are a couple approaches to implement Full Join by DATA steps.

To demonstrate implementation of full join by DATA steps, we use most straight forward two-step approaches:

Step 1 generates one master data set which contains one data set and unique key values in another data set. So, the master data set will have all key values from both data sets and data from one data set.

Step 2 generates full join data set by left join the master data set with another data set.

Below, like previous examples, are codes and output data sets.

Multiple SET statements
Codes:

```
** full0: DSN21 + DSN21's ID **;
proc sort data=DSN22(keep=ID)
   out=dsn22id nodupkey;
   by ID;
run;
proc sort data=DSN21;
   by ID;
run;
data full0;
   merge dsn21 dsn22id;
   by ID;
run;
** full1: left join full0 and dsn22 **;
data full1(drop=_ID hasDSN22);
   set full0;
   do i=1 to n;
      set DSN22(rename=(ID=_ID))
               point=i nobs=n;
      if ID=_ID then do;
         hasDSN22=1;
         output;
      end;
   end;
   if hasDSN22 ne 1 then do;
      event='';
      output;
   end;
run;
```

PROC SQL
Codes:

```
proc sql;
   create table full2 as
   select
     coalesce(a.id,b.id) as id,
     a.arm, b.event
   from dsn21 a full join dsn22 b
      on (a.id=b.id)
   ;
quit;
```

Output data set full1

ID	ARM	EVENT
1001	TRT1	Headache
1001	TRT1	Nausea
1001	GRP1	Headache
1001	GRP1	Nausea
1002	TRT2	Headache
1002	TRT2	Rash
1003	TRT3	
1003	GRP1	
1004		Fever

Output data set full2

ID	ARM	EVENT
1001	TRT1	Nausea
1001	TRT1	Headache
1001	GRP1	Nausea
1001	GRP1	Headache
1002	TRT2	Headache
1002	TRT2	Rash
1003	GRP1	
1003	TRT3	
1004		Fever

Example 5 – Full Join

Compared to previous examples, PROC SQL codes for Full Join are almost same as codes of Inner Join or Left Join, while the codes by DATA steps are more complicated. This is because SQL language has wrapped complicated implementations behind, which may cause less flexibility. By DATA step, we need to design and implement Full Join algorithm step by step.

In the Example 5, the codes follows above two-step algorithm:

1) To generate master data set full0: merging data set DSN21 (presorted) with data set dsn22id (containing data set DSN22's unique ID values) by ID; So, data set full0 will have all DSN21 data and extra unique ID values from dsn22id if any;

2) To generate data set full1 by left join the master data set full0 and data set DSN22. Because master data set full0 has all DSN21 data and all DSN22's ID values, output data set full1will keep all DSN21 data and add all DSN22 data by full0.ID.

Again, compared the output data sets of PROC SQL and multiple-SET DATA step, they are almost identical, except sorting order.

5 OTHER APPLICATIONS

5.1 CALCULATING PERCENTAGES

data set pop

SUBJECT	TRT01AN
pt1	1
pt2	3
pt3	2
pt4	3
pt5	1
pt6	2
pt7	1
pt8	
pt9	2
pt10	3
pt11	1

data set aefrq

AETERM	TRT01AN	COUNT
AE1	1	2
AE1	2	1
AE2	1	3
AE3	1	1
AE3	2	3
AE3	3	2
AE4	2	1
AE4	3	1

```
data res(drop=id);
   array Ns {3}  _temporary_ (0 0 0);
   if _N_=1 then do i=1 to nobs;
      set pop(keep=TRT01AN) point=i nobs=nobs;
      if TRT01AN ne . then
         Ns(TRT01AN)=Ns(TRT01AN)+1;
   end;
   set AEFRQ;
   N=Ns(trt01an);
   pchg=count/n*100;
   format pchg 5.1;
run;
```

output data set res

TRT01AN	AETERM	COUNT	N	PCHG
1	AE1	2	4	50
2	AE1	1	3	33.3
1	AE2	3	4	75
1	AE3	1	4	25
2	AE3	3	3	100
3	AE3	2	3	66.7
2	AE4	1	3	33.3
3	AE4	1	3	33.3

Example 6 – Calculating percentages

In the Example 6, we have a common case in safety analysis: After using PROC FREQ to get AE counts by treatment arms, we need to calculate the percentages.

Usually we would use 1) one PROC FREQ to get Ns of treatment arms, 2) two PROC SORT procedures to sort data sets before merging them, 3) one DATA step to merge AE count and N value data sets and calculate the percentages.

By using multiple SET statements, we eliminate the part of one PROC FREQ and two PROC SORT, and use only one DATA step without sorting and MERGE-BY statement to get same results. The codes above loop data set POP to populate treatment arms' N numbers in an array Ns. Because of “IF _N_ = 1 THEN”, populating N values will only execute once at the beginning of the DATA step. When executing “set AEFRQ;”, N of each treatment arms has been populated in the array Ns, and percentages could be calculated by retrieving treatment arm's N from array Ns.

5.2 FINDING AE'S TREATMENT CYCLE NUMBERS

data set ae

SUBJID	AETERM	AESTDT
1	AE10	01JUN2017
1	AE11	22SEP2017
2	AE20	02APR2017
2	AE21	30MAY2017

data set cycle

SUBJID	CYCLEN	CYCSTDT	CYCENDT
1	1	05JUN2017	02JUL2017
1	2	03JUL2017	30JUL2017
1	3	31JUL2017	27AUG2017
2	1	15MAR2017	16APR2017
2	2	17APR2017	15MAY2017

```
data ae;
   set ae;
   do i=1 to n;
      set cycle(keep=subject cycnum cycstdt cycendt
          rename=(subject=cycpt)) point=i nobs=n;
      if subject=cycpt
         and cycstdt <= aestdt <= cycendt then
         aecyc=cycnum;
   end;
   keep subject aeterm aestdt aecyc;
run;
```

output data set ae

SUBJECT	AETERM	AESTDT	AECYC
1	AE10	01JUN2017	
1	AE11	22JUL2017	2
2	AE20	02APR2017	1
2	AE21	30MAY2017	

Example 7 – Deriving AE's Treatment Cycle Numbers

Oftentimes, we have subjects' AE data and treatment cycles' start/end dates. We need to derive AE cycles for by-cycle analysis. This is another common situation that is like many-to-many match which MERGE-BY in DATA step cannot handle. Sticking to DATA steps, I saw different approaches to solve similar issues. Some programmers populated all calendar dates within subject' treatment cycles, then merged with subject's AEs by date. Some programmers transposed subject's treatment cycle data, merged with AE data, and found out AE cycle numbers. Those solutions usually require extra procedures and couple DATA steps.

With Multiple SET statements in one DATA step, it is much simple. Like the codes in the Example 7, in each iteration of data set AE, the DO loop will iterate data set CYCLE and populate variable AECYC (AE Cycle Number) if subject's AE start date falls in one subject's treatment cycle by cycle start date and end date. It is very straight forward to populate AE's cycle numbers, without PROC TRANSPOSE, PROC SORT, and DATA MERGE.

CONCLUSION

This paper had discussed how multiple SET statements work in DATA step, compared implementation codes of inner and outer joins between PROC SQL and DATA step with multiple SET statements, provided some examples that would make your codes simpler and more flexible.

As we discussed above, with multiple SET statements, one DATA step can do many-to-many merge and achieve desired results which traditionally need multiple PROCs and DATA steps. Such simplicity usually will bring you more flexibility and efficiency.

REFERENCES

Sample 24652: Generate every combination of observations between data sets, Cary, NC: SAS Institute Inc

Using Two SET Statements in One DATA Step, Ben Cochran, The Bedford Group, Raleigh, NC, PharmaSUG2011 – Paper CC25

Multiple Set Statements in a Data Step: A Powerful Technique for Combining and Aggregating Complex Data, Renu Gehring, SAS Instructor in Ace-Cube, LLP, Health Care Analyst in CareOregon, Inc.

Multiple SET Statements in a DATA Step - Why and how to use them, Thomas Wollseifen, PHUSE

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Yizhong (John) Huang

Celgene Corporation

(973)960-0806

jhuang98@gmail.com