Paper 3128-2019

DOSUBL Function + SQL View + Hash Object ≅ FedSQL + PROC DS2 Hash Package

Yutaka Morioka, Jun Hasegawa, EPS Corporation, Japan

ABSTRACT

The DS2 procedure brought the object-oriented concept to the world of SAS® programming and enabled a high level of data handling that the DATA step could not reach for years. One detailed but noteworthy feature is that the hash package, a predefined package in the DS2 procedure, enables us to assign a FedSQL query to the argument tag of the dataset method. By specifying a FedSQL query for the argument, you can directly store the result of the query into the hash object without creating an unnecessary subset data set beforehand. Despite this and other powerful advantages in data handling, the DATA step still prevails among SAS users, and in most cases, it makes us feel content to continue using the DATA step. Given the situation, this paper introduces an alternative way to achieve what the DS2 procedure enables us to, by the traditional DATA step combining a hash object, the SQL procedure, and the DOSUBL function, which enables the immediate execution of SAS code.

INTRODUCTION

The hash package of the DS2 procedure has enabled us to program in an advanced and simple manner when dealing with data handling. But given the situation of the DATA step dominancy, it seems demanded to work on reproducing the feature of the DS2 procedure by the traditional DATA step. This article hence gives a comparative analysis of a several programming approaches and tries to seek DATA step techniques which replicate or might exceed the DS2 procedure, while explaining how usable the hash package is for data manipulation.

TEST DATA SET AND GOAL DATA SET

The CLASS data set of the SASHELP library, which is presented as Figure 1, will be used for a main example. This paper introduces various programming approaches trying to create a data set as presented in Figure 2, the data set in which a variable for the average value of age by sex is added to the original CLASS data set.

To begin with, the CLASS data set needs to be copied to the WORK library and renamed to be SASHELP_CLASS for the convenience of explanation.

Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84
John	M	12	59	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90
Louise	F	12	56.3	77
Mary	F	15	66.5	112
Philip	M	16	72	150
Robert	M	12	64.8	128
Ronald	M	15	67	133
Thomas	M	11	57.5	85
William	M	15	66.5	112

Figure 1. Test Data Set

Name	Sex	Age	Height	Weight	age_mean
Alfred	M	14	69	112.5	13.4
Alice	F	13	56.5	84	13.222222222
Barbara	F	13	65.3	98	13.222222222
Carol	F	14	62.8	102.5	13.222222222
Henry	M	14	63.5	102.5	13.4
James	M	12	57.3	83	13.4
Jane	F	12	59.8	84.5	13.222222222
Janet	F	15	62.5	112.5	13.222222222
Jeffrey	M	13	62.5	84	13.4
John	M	12	59	99.5	13.4
Joyce	F	11	51.3	50.5	13.222222222
Judy	F	14	64.3	90	13.222222222
Louise	F	12	56.3	77	13.222222222
Mary	F	15	66.5	112	13.222222222
Philip	M	16	72	150	13.4
Robert	M	12	64.8	128	13.4
Ronald	M	15	67	133	13.4
Thomas	M	11	57.5	85	13.4
William	M	15	66.5	112	13.4

Figure 2. Goal Data Set

SOLUTION 1: CREATE A SUMMARY DATA SET FIRST AND RETRIEVE THE SUMMARY NEXT

The simplest and presumably the most common approach to achieve the goal would be to run a summary calculating procedure as a first step, and to store the summary into a hash object setting sex as a key and retrieve the average from the object through the FIND method in the next DATA step.

Of course, instead of using a hash object, the MERGE statement can be used to join the data sets. However, the MERGE statement requires the data sets to be sorted in advance and hence additional steps only for sorting become necessary. In my opinion, increasing the number of steps for the solo purpose of merging data sets will make the program just

redundant. Therefore, this paper does not use the MERGE statement and a hash object will be utilized for joining data sets.

The following source code best describes this approach:

```
/*summary*/
proc summary data = sashelp_class nway;
  class sex;
  var age;
  output out = sex_summary(drop = _TYPE_ _FREQ_) mean=age_mean;
run;

/*find method*/
data output1;
  set sashelp_class;
  if 0 then set sex_summary;
  if _N_=1 then do;
    declare hash h1(dataset:"sex_summary");
    h1.definekey("sex");
    h1.definedata("age_mean");
    h1.definedone();
  end;
  if h1.find() ne 0 then call missing(age_mean);
run;
```

[sex_summary]

Sex	age_mean
F	13.222222222
M	13.4

This approach seems quite reasonable, but one pragmatic problem is inherent in this program, which is you have to prepare two separate sections of code, one for calculating a summary and the other for joining data sets. This separation is a potential risk of making the code less readable. Although this example is still easy to read since it creates only one subset summary data set for the convenience of explanation, it is certain that more subset summary data sets are created, less readable the program becomes. If you try to join several summary data sets in the last DATA step, then you need to create all the summary data sets beforehand and hence the visual distance becomes much longer between the code which creates the subset data sets and the DATA step which retrieves variables from the subset data sets at the end.

Therefore, it can be argued that this approach is reasonable but includes a risk of decreasing readability, so this paper continues seeking another programming solution.

SOLUTION 2: STORE SUMS AND COUNTS INTO HASH OBJECTS AND CALCULATE THE AVERAGE IN THE DO WHILE LOOP

One solution to avoid the separation of code can be the so-called DOW-loop, which assigns the END option of the SET statement to the condition of the DO WHILE loop. This article skips a detailed explanation of the DOW-loop logic and its implementation in order not to get sidetracked. Please refer to previous SAS® conference proceedings if interested since there have been a number of papers regarding the DOW-loop technique. The source code below illustrates the second approach, implementing the DOW-loop logic to avoid the separation of code:

```
data output2;
if _N_=1 then do;
```

```
  declare hash h1(suminc:'total');
    h1.definekey('sex');
    h1.definedata('sex', 'total');
    h1.definedone();
  declare hash h2(suminc:'count');
    h2.definekey('sex');
    h2.definedata('sex', 'count');
    h2.definedone();
end;

/*--Read all the observations of the original data set, and store the sums
  and counts into hash objects.--*/
do while(^fl);
 set sashelp_class end=fl;
  total=age;
  count=1;
  h1.ref();
  h1.sum(sum:total);
  h1.replace();
  h2.ref();
  h2.sum(sum:count);
  h2.replace();
end;

/*--Read all the observations again, and retrieve the sums and counts by
FIND method, and calculate the average.--*/
do while(^fl2);
  total=0;
  set sashelp_class end=fl2;
   h1.find();
   h2.find();
  age_mean=total/count;
  output;
end;
stop;
run;
```

sex	total
F	119
M	134

sex	count
F	9
M	10

In the preceding sample source code, the SUMINC option is specified for the DECLARE statement. This SUMINC is an internal accumulator for each key of a hash object and its internal value accumulates when the key is looked up by the FIND, CHECK or REF method. The accumulator is retrieved and output to the data set by the SUM method. In this example, the number of subjects and the total sum of age for each sex are stored in the first DO WHILE loop. The average age by sex is calculated in the next DO WHILE loop.

The advantage of this programming approach is that one DATA step can accommodate the entire process and you can expand the logic simply by adding hash objects in the DATA step.

However, needless to say, this approach is too complicated to understand at a glance. Hence this cannot be a desired solution.

SOLUTION 3: DS2 PROCEDURE (HASH PACKAGE)

Before reaching a deadlock of this challenges, let us take a look at how it would be implemented if resorting to the hash package of the DS2 procedure:

```
proc ds2;
```

```
data output3(overwrite=yes);
declare package hash h1();
dcl double age_mean;

method init();
  h1.dataset('{select sex,mean(age) as age_mean
            from sashelp_class {options locktable=share}
            group by sex }');
  h1.keys([sex]);
  h1.data([age_mean]);
  h1.definedone();
end;

method run();
  set sashelp_class(locktable=share);
  h1.find();
 end;

enddata;
run;
quit;
```

By using the hash package of the DS2 procedure, you can create and retrieve subset data sets in one procedure in a concise and viewer-friendly way.

You can notice that the programming flow is similar between the DS2 hash package and a hash object in the traditional DATA step in a sense that the both start with initializing a hash object and retrieve the data from the object by the FIND method. On the other hand, when you create multiple subset data sets with the DS2 procedure, all the subset creating processes will be contained only in the INIT method and which makes the code with the DS2 procedure more readable.

When implementing the DS2 procedure, you need to be careful not to forget declaring new variables on the top. In addition, the LOCKTABLE option needs to be set SHARE to specify the access method to the data set when the same data set is refereed multiple times in the procedure.

SOLUTION 4: DOSUBL FUNCTION + SQL VIEW + HASH OBJECT

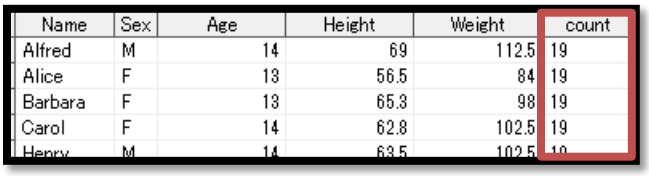
It goes without saying that the DS2 procedure allows advanced data manipulation and it exceeds the traditional DATA step in many ways. Yet it does not prevail at this time and most programs are written using the DATA step. Furthermore, it seems difficult to replace the entire traditional code with the DS2 procedure for it contains a quite number of new features.

Thus, this article continues to seek an alternative way to realize the powerful feature of the DS2 procedure by the traditional DATA step.

DOSUBL FUNCTION

Prior to the introduction of the next programming approach, one has to understand the nature of the DOSUBL function. The DOSUBL function can submit SAS code to run on the side while the DATA step is still running. The function is sometimes mistakenly understood to have the same feature with the CALL EXECUTE, but it is definitely different between the two. The CALL EXECUTE submits the parameter SAS code after the DATA step finished. In contrast, the DOSUBL function can submit SAS code while the DATA step is still running. Please execute the following programs to tell the difference. The programs try to add a

variable of the total number of the observations to the original SASHELP_CLASS data set as presented in Figure 3:



Name	Sex	Age	Height	Weight	count
Alfred	M	14	69	112.5	19
Alice	F	13	56.5	84	19
Barbara	F	13	65.3	98	19
Carol	F	14	62.8	102.5	19
Henry	M	14	63.5	102.5	19

Figure 3. Test Data to Compare the CALL EXECUTE and the DOSUBL Function

✖

```
data sashelp_class1;
  set sashelp_class;
  if _N_=1 then do;
      call execute("proc sql;select count(*) into:obs1 trimmed
                    from sashelp_class;
                    quit;");
    end;
    count=symget("obs1");
run;
```

○

```
data sashelp_class2;
  set sashelp_class;
  if _N_=1 then do;
      rc=dosubl("proc sql;select count(*) into:obs2 trimmed
                 from sashelp_class;
                 quit;");
    end;
    count=symget("obs2");
    drop rc;
run;
```

The example with the CALL EXECUTE cannot run successfully because it fails at the SYMGET function. As mentioned above, the CALL EXECUTE submits its parameter SAS code after the DATA step ended and thus the macro variable "obs1" cannot be resolved when the SYMGET function executed. On the other hand, the DOSUBL function enables the immediate execution of SAS code. Since the macro variable "obs2" is prepared before the SYMGET function, the second DATA step can finish successfully.

It is speculated that the DATA step mainstream processing stays on hold while the DOSUBL function is being executed. Figure 4 presents my understanding of the DOSUBL function, in which the DATA step mainstream is depicted as the thick straight arrow heading to the bottom.

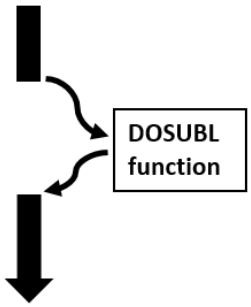


Figure 4. Visual Image of the DOSUBL Function Executed in a DATA Step

DOSUBL FUNCTION + SQL VIEW + HASH OBJECT

Now this paper can finally address the solution for this challenge, which is a combination of the DOSUBL function, the SQL procedure and a hash object.

One day an idea was come up with almost like an intuitive inspiration, and I implemented it in SAS code. The idea was simple. There would be a chance for SAS code to run successfully as long as a data set specified for the DATASET option of the DECLARE statement is created before the execution of the DECLARE statement even if the data set was not prepared until the preceding DATA step. The following code illustrates the idea. It creates a SQL view through the DOSUBL function and stores the view into a hash object. It is no problem to create a data set by the SQL procedure instead of creating a view:

```
data output4;
  set sashelp_class;
  if _N_ = 1 then do;

    rc=dosubl("proc sql noprint;
               create view v1 as
               select sex ,mean(age) as age_mean
               from sashelp_class
               group by sex;
               quit;");
    drop rc;

    declare hash h1(dataset:"v1");
    h1.definekey("sex");
    h1.definedata("age_mean");
    h1.definedone();

   end;

   if h1.find() ne 0 then call missing(age_mean);
  run;
```

Sex	age_mean
F	13.222222222
M	13.4

V1

Surprisingly the program works without an error. Since the view to be loaded in the hash object is prepared before the DECLARE statement, this program works just fine. However, this approach takes longer time for processing and requires memory capacity. Yet as you already noticing, the program achieves accommodating processes of creating and retrieving subset data sets in one DATA step, and it is concise and readable for programmers can easily understand the role of each step.

ONE APPLICATION EXAMPLE OF SOLUTION 4

By applying the DOSUBL function approach, theoretically, you can create subset data sets as many as possible in one DATA step. It is not recommended due to the memory capacity and readability, but the following DATA step runs successfully. The program sets a flag to a person whose BMI is the lowest in the data set:

```
data output5;
  set sashelp_class;
  if _N_ = 1 then do;

  rc=dosubl("data temp1;
              set sashelp_class;
              bmi=weight/height**2*703;
             run;
             proc sort data=temp1;
               by bmi;
             run;
             data temp2;
               set temp1;
               by bmi;
               if _N_=1;
             run;");
drop rc;

declare hash h1(dataset:"temp2");
  h1.definekey("name");
  h1.definedone();
end;

if h1.check() eq 0 then Min_BMI_FL="Y";
run;
```

ANOTHER APPLICATION EXAMPLE OF SOLUTION 4

Another advantage of creating subsets by the DOSUBL function is that it enables to generate and execute SAS code dynamically using values defined in a DATA step. Using a DO loop, the following program firstly generates SQL queries to calculate summary statistics of age for a respective group of people whose first letter of their name is "A" or "J" and outputs the summary values to a data set as presented in Figure 5:

```
data output6;
length Age 8.;
do fchar="A","J";
 code=cats("proc summary data=sashelp_class;
                   where first(name)='",fchar,"';
                   var age; output out=out;
                  run;");
 rc=dosubl(code);

 declare hash h1(dataset:"out");
  h1.definekey("_STAT_");
  h1.definedata("Age");
  h1.definedone();

 _STAT_="MEAN";
  if h1.find() ne 0 then call missing(age);
```

```
    output;

  end;
  drop rc;
  run;
```

Age	fchar	code	_STAT_
13.5	A	proc summary data=sashelp.class;where first(name)='A';var age; output out=out;run;	MEAN
12.714285714	J	proc summary data=sashelp.class;where first(name)='J';var age; output out=out;run;	MEAN

Figure 5. Dynamic Generation and Execution of SAS Code by the DOSUBL Function

CONCLUSION

The DS2 procedure has provided a number of powerful and flexible features for data set manipulation. Yet I could not give up on implementing the powerful features by using the traditional DATA step. The programming approach suggested in this article is the fruit of my curiosity and challenges. As a result, by combining the DOSUBL function, the SQL procedure and a hash object, this paper succeeded in accommodating processes of creating and referring subset data sets in one DATA step in a readable way.

In addition, this article was able to introduce an interesting finding that SAS code could work successfully as long as the subset data set was prepared before the execution of the DECLARE statement of a hash object.

Furthermore, even compared with the DS2 procedure, some advantages of using a combination of the DATA step and the DOSUBL function were found. One is that you can create subset data sets not only through SQL queries, and another is that the function allows dynamic generation and execution of SAS code using values defined in a DATA step. These advantages may suggest that the combination approach will not end up just imitating the DS2 procedure and that there is still room for researching the DATA step.

Not just being satisfied with these findings, I will continue my journey to pursue a better use of the DATA step for fun and pragmatic benefit.

REFERENCES

Loren, Judy. 2008. "How Do I Love Hash Tables? Let Me Count The Ways!" *Proceedings of the SAS Global Forum 2008 Conference.* Available at https://support.sas.com/resources/papers/proceedings/pdfs/sgf2008/029-2008.pdf

Eberhardt, Peter and Yao, Xue. 2015. "DS2 with Both Hands on the Wheel." *Proceedings of the SAS Global Forum 2015 Conference.* Available at https://support.sas.com/resources/papers/proceedings15/2523-2015.pdf

Dorfman, Paul and Vyverman, Koen. 2009. "The DOW-Loop Unrolled." *Proceedings of the SAS Global Forum 2009 Conference.* Available at http://support.sas.com/resources/papers/proceedings09/038-2009.pdf

Langston, Rick. 2013. "Submitting SAS® Code On The Side." *Proceedings of the SAS Global Forum 2013 Conference.* Available at https://support.sas.com/resources/papers/proceedings13/032-2013.pdf

ACKNOWLEDGMENTS

Special thanks to Shinya Sakai, Chie Matsuyama and Takeshi Shinohara of EPS Corporation for their tremendous support to realize this foreign publication, and also express my deepest gratitude to my colleagues in the Osaka headquarter for always providing thought-provoking opinions and their unlimited support.

CONTACT INFORMATION

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

Yutaka Morioka
EPS Corporation
E-mail1: morioka038@eps.co.jp
E-mail2: sasyupi@gmail.com
blog: http://sas-tumesas.blogspot.com/
twitter: https://twitter.com/sasyupi

Jun Hasegawa
EPS Corporation
E-mail: hasegawa322@eps.co.jp