

Virtual Accessing of a SAS® Data Set Using OPEN, FETCH, and CLOSE Functions with %SYSFUNC and %DO Loops

Amarnath Vijayarangan, Emmes Services Pvt Ltd, India

ABSTRACT

One of the truths about SAS® is that there are, at a minimum three approaches to achieve any intended task and each approaches has its own pros and cons. Identifying and using of efficient SAS programming techniques are recommended and efficient programming becomes mandatory when the larger data sets are accessed. This paper describes the efficiency of virtual access and various situations to virtual access of the data sets using OPEN, FETCH and CLOSE functions with %SYSFUNC and %DO LOOPS.

INTRODUCTION

There are several ways to get the data set attributes like number of observations, variables, type of variables, and so on. One of the efficient approaches is using the OPEN function with %SYSFUNC. In addition to attribute access, the function FETCH with OPEN gives the ability to access the values of the variables from a SAS data set for conditional and iterative executions with %DO LOOPS. Though, the data sets used in this paper are small in size but reader should virtually assume that the data sets are large enough (Terabyte). This paper discusses the various situations of virtual access of a SAS data set as follows:

Situation 1: It is often required to split the master data set into several pieces depending upon a certain criterion for the parallel run, as it is required that data sets be independent.

Situation 2: For most of the statistical analysis, particularly for correlation and regression, a widely and frequently used SAS procedure requires dynamic variable list to be passed. Creating single macro variable list might run into an issue with the macro variable length on the larger transactional data sets. It is recommended that you prepare the list of the variables as a data set and access them using the proposed approach in many places instead of creating several macro variables.

Situation 3: For the array process and for many reports it is required to keep the relevant variables together instead of maintaining the original data set order.

SITUATION 1.1

BREAKING MASTER DATA SET INTO PIECES BASED ON ONLY ONE CRITERION

Let us use the SASHELP.SNACKS data set and perform an analysis for each product. Usually the list of products to be analyzed is given by the client. In this case, let us first identify the list of products.

```
*** Define library;
libname snacks ".";

*** Identify the list of products;
proc sort data=SASHELP.SNACKS (keep=product) out=product nodupkey;
  by product;
run;

*** Id creation for each product;
data product;
  set product;
  id=_n_;
run;
```

For the parallel process, break the SASHELP.SNACKS data set using %datasplit macro to create permanent data set for each product.

```
%macro datasplit;
```

```
data
```

```
  %let dsid=%sysfunc(open(product));
```

```
  %do %while(not %sysfunc(fetch(&dsid)));
```

```
    snacks.pid%sysfunc(getvarn(&dsid, %sysfunc(varnum(&dsid, id))))
```

```
  %end;
```

```
  %let dsid=%sysfunc(close(&dsid));
```

```
  set SASHELP.SNACKS;
```

```
  %let dsid=%sysfunc(open(product));
```

```
  %let i=0;
```

```
  %do %while(not %sysfunc(fetch(&dsid)));
```

```
    %let i = %eval(&i + 1);
```

```
    %if &i=1 %then %let elsest=;
```

```
    %else %let elsest=else;
```

```
&elsest if product = "%sysfunc(getvarc(&dsid, %sysfunc(varnum(&dsid,
product))))" then output snacks.pid%sysfunc(getvarn(&dsid,
```

```
%sysfunc(varnum(&dsid, id)));
```

```
  %end;
```

```
  %let dsid=%sysfunc(close(&dsid));
```

```
run;
```

```
%mend;
```

```
%datasplit
```

- ✓ Read in product data set
- ✓ Fetch observation
- ✓ Read in id variable

- ✓ Read in product data set
- ✓ Fetch observation
- ✓ Read product & id variables

The usual approach of storing value of the product list into a macro variable may run into a macro variable length issue on a larger data set. Alternatively the product variable list can be split and create several macro variables or for each product one macro variable can also be created. In all these cases, the above approach reduces the code length, resolves macro variable length issue and avoids several macro variables creation. The display 1 depicts actual code created from macro %datasplit:

```
1186
1187 %datasplit
MPRINT(DATASPLIT):  data snacks.pid1 snacks.pid2 snacks.pid3 snacks.pid4 snacks.pid5 snacks.pid6
snacks.pid7 snacks.pid8 snacks.pid9 snacks.pid10 snacks.pid11 snacks.pid12 snacks.pid13
snacks.pid14 snacks.pid15 snacks.pid16 snacks.pid17 snacks.pid18 snacks.pid19 snacks.pid20
snacks.pid21 snacks.pid22 snacks.pid23 snacks.pid24 snacks.pid25 snacks.pid26 snacks.pid27
snacks.pid28 snacks.pid29 snacks.pid30 snacks.pid31 snacks.pid32 snacks.pid33 snacks.pid34
snacks.pid35 ;
MPRINT(DATASPLIT):  set sashelp.snacks;
MPRINT(DATASPLIT):  if product = "Baked potato chips" " then output
snacks.pid1;
MPRINT(DATASPLIT):  else if product = "Barbeque pork rinds" " then output
snacks.pid2;
MPRINT(DATASPLIT):  else if product = "Barbeque potato chips" " then output
snacks.pid3;
MPRINT(DATASPLIT):  else if product = "Bread sticks" " then output
snacks.pid4;
```

Display 1. Partial Log Report

SITUATION 1.2

Breaking master data set into pieces based on more than one criterion: Let us modify the above code to break the SASHELP.SNACKS data set by product and region assuming region variable is present in SASHELP.SNACKS data set. The only change required in the above %datasplit is highlighted in bold letters as follows:

```
&elsest if product= "%sysfunc(getvarc(&dsid, %sysfunc(varnum(&dsid,
product))))" and region="%sysfunc(getvarc(&dsid, %sysfunc(varnum(&dsid,
region))))" then output snacks.pid%sysfunc(getvarn(&dsid,
%sysfunc(varnum(&dsid, id)));
```

Of course, the product id creation should be done for each product and region level. This approach can be extended for various conditions and this approach does not require creation of additional macro variables.

SITUATION 2

Let us use SASHELP.SNACKS data set to create a dynamic variable list to perform a regression analysis. For each product, rests of the product are treated as competitor product. A model is built for the product id 1 by keeping rest of the product's Price and Advertised variable as independent variables. The below data set MDSN is used for the modeling which is converted to Date level and kept only fewer variables:

	Date of sale	price7	price8	adv7	adv8	Qty7	Qty8
1	01JAN2002	1.49	2.99	0	0	2	0
2	02JAN2002	1.49	2.99	0	0	12	0
3	03JAN2002	1.49	2.99	0	0	2	0
4	04JAN2002	1.49	2.99	0	0	2	0
5	05JAN2002	1.49	2.99	0	0	6	0
6	06JAN2002	1.49	2.99	0	0	2	0
7	07JAN2002	1.49	2.99	0	0	1	0
8	08JAN2002	1.49	2.99	0	0	0	0
9	09JAN2002	1.49	2.99	0	0	3	0
10	10JAN2002	1.49	2.99	0	0	2	0
11	11JAN2002	1.49	2.99	0	0	3	0

Display 2. Transposed Format of SASHELP.SNACKS Data Set

SITUATION 2.1

It would be nice to keep the relevant variables together. The below macro can be used to order variables as needed.

```
%macro fetchobs1(dsn=);
  %let dsid=%sysfunc(open(&dsn));
  %do %while(not %sysfunc(fetch(&dsid)));
    price%sysfunc(getvarn(&dsid, %sysfunc(varnum(&dsid, pid))))
    adv%sysfunc(getvarn(&dsid, %sysfunc(varnum(&dsid, pid))))
    qty%sysfunc(getvarn(&dsid, %sysfunc(varnum(&dsid, pid))))
  %end;
  %let dsid=%sysfunc(close(&dsid));
%mend fetchobs1;
data mdsn;
  length date %fetchobs1(dsn=ProductList) 8;
  set mdsn;
run;
```

	Date of sale	price1	adv1	qty1	price2	adv2	qty2	price3	adv3	qty3
1	01JAN2002	1.99	0	0	1.49	0	3	1.49	0	6
2	02JAN2002	1.99	0	0	1.49	0	11	1.49	0	12
3	03JAN2002	1.99	0	0	1.49	0	1	1.49	0	0
4	04JAN2002	1.99	0	0	1.49	0	1	1.49	0	6
5	05JAN2002	1.99	0	0	1.49	0	13	1.49	0	12
6	06JAN2002	1.99	0	0	1.49	0	1	1.49	0	0
7	07JAN2002	1.99	0	0	1.49	0	5	1.49	0	8

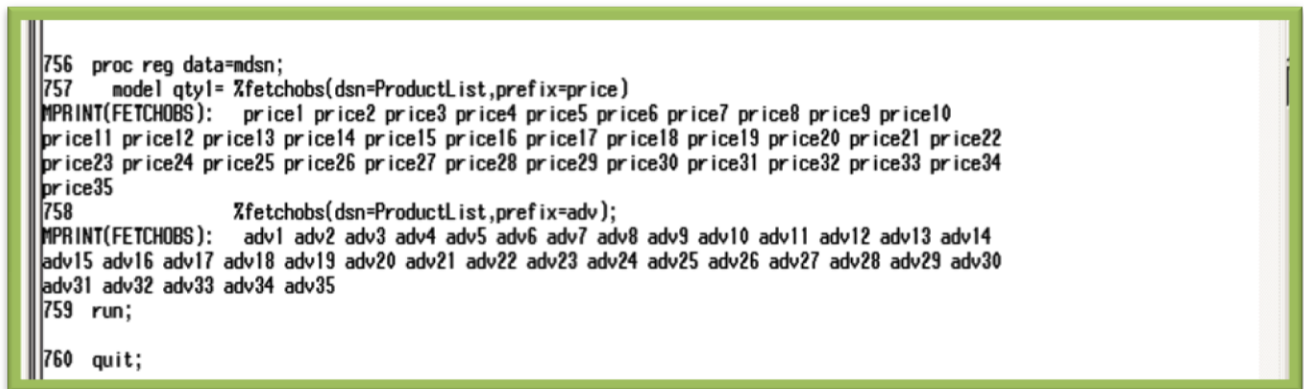
Display 3. Transposed Format of SASHELP.SNACKS Data Set with Variable Order

SITUATION 2.2

The macro fetchobs2 creates the dynamic list of price, adv and Qty variable list that can be used as independent variables list. While performing the regression analysis, there are various steps involved like variation check, data availability on each variable and correlation analysis. In all these places the same macro can be used to form a variable list.

```
%macro fetchobs2(dsn=);
  %let dsid=%sysfunc(open(&dsn));
  %do %while(not %sysfunc(fetch(&dsid)));
    %sysfunc(getvarn(&dsid, %sysfunc(varnum(&dsid, pid))))
  %end;
  %let dsid=%sysfunc(close(&dsid));
%mend fetchobs2;

proc reg data=mdsn;
  model qty1= %fetchobs2(dsn=ProductList,prefix=price)
             %fetchobs2(dsn=ProductList,prefix=adv);
run;
quit;
```



```
756 proc reg data=mdsn;
757   model qty1= %fetchobs(dsn=ProductList,prefix=price)
MPRINT(FETCHOBS):  price1 price2 price3 price4 price5 price6 price7 price8 price9 price10
price11 price12 price13 price14 price15 price16 price17 price18 price19 price20 price21 price22
price23 price24 price25 price26 price27 price28 price29 price30 price31 price32 price33 price34
price35
758           %fetchobs(dsn=ProductList,prefix=adv);
MPRINT(FETCHOBS):  adv1 adv2 adv3 adv4 adv5 adv6 adv7 adv8 adv9 adv10 adv11 adv12 adv13 adv14
adv15 adv16 adv17 adv18 adv19 adv20 adv21 adv22 adv23 adv24 adv25 adv26 adv27 adv28 adv29 adv30
adv31 adv32 adv33 adv34 adv35
759 run;
760 quit;
```

Display 4. Partial Log Report

To drop the insignificant variables, it is enough to delete the unwanted PIDs from the DSN data set instead of updating the macro variable list as it is done in the usual approach. The similar approach can be used in any other statistical analysis to form a dynamic analysis variable list.

SITUATION 3.1

ARRAY PROCESS: It is required to use the relevant and right variables for the array process when any formulas involved in creating the new variables. With the above example, let us create the dollar sales for each product as a new variable in MDSN.

Formula: Dollar sales= Price * QtySold

```
data mdsn;
  set mdsn;
  array price(*) %fetchobs2(dsn=ProductList,prefix=price);
  array qty(*) %fetchobs2(dsn=ProductList,prefix=qty);
  array dsale(*) %fetchobs2(dsn=ProductList,prefix=dsale);
  do i=1 to dim(qty);
    if price(i) and qty(i) then dsale(i)=price(i)*qty(i);
  end;
  drop i;
run;
```

This approach ensures that for each product the corresponding price and sale quantities are used for the calculation and avoids the several macro variable creations.

```

1006 data mdsn;
1007   set mdsn;
1008   array price(*) %fetchobs2(dsn=ProductList,prefix=price);
MPRINT(FETCHOBS2):  price1 price2 price3 price4 price5 price6 price7 price8 price9 price10
price11 price12 price13 price14 price15 price16 price17 price18 price19 price20 price21 price22
price23 price24 price25 price26 price27 price28 price29 price30 price31 price32 price33 price34
price35
1009   array qty(*) %fetchobs2(dsn=ProductList,prefix=qty);
MPRINT(FETCHOBS2):  qty1 qty2 qty3 qty4 qty5 qty6 qty7 qty8 qty9 qty10 qty11 qty12 qty13 qty14
qty15 qty16 qty17 qty18 qty19 qty20 qty21 qty22 qty23 qty24 qty25 qty26 qty27 qty28 qty29 qty30
qty31 qty32 qty33 qty34 qty35
1010   array dsale(*) %fetchobs2(dsn=ProductList,prefix=dsale);
MPRINT(FETCHOBS2):  dsale1 dsale2 dsale3 dsale4 dsale5 dsale6 dsale7 dsale8 dsale9 dsale10
dsale11 dsale12 dsale13 dsale14 dsale15 dsale16 dsale17 dsale18 dsale19 dsale20 dsale21 dsale22
dsale23 dsale24 dsale25 dsale26 dsale27 dsale28 dsale29 dsale30 dsale31 dsale32 dsale33 dsale34
dsale35
1011   do i=1 to dim(qty);
1012     if price(i) and qty(i) then dsale(i)=price(i)*qty(i);
1013   end;
1014 run;

```

Display 5. Partial Log Report

SITUATION 3.2:

Let us perform correlation analysis among the price variables.

```

*** Perform correlation analysis;
proc corr data=mdsn out=Corr(where=(_TYPE_='CORR'));
  var price;;
run;

*** Extract product ID variable;
data corr1;
  set corr;
  id=input(compress(_name_, 'kd'), best.);
  drop _;;
run;

*** Fetch the Product name;
data corr2;
  merge corr1 product;
  by id;
run;

*** Report;
%macro corr_rpt;
ods tagsets.excelxp file="CorrelationMatrix.xls" style=statistical;
proc print data = corr2 label noobs ;
  label %let dsid=%sysfunc(open(product));
  %do %while(not %sysfunc(fetch(&dsid)));
    price%sysfunc(getvarn(&dsid, %sysfunc(varnum(&dsid, id)))=
    %nrbquote(%sysfunc(getvarc(&dsid, %sysfunc(varnum(&dsid, product)))))
  %end;;
  %let dsid=%sysfunc(close(&dsid));;
  var product price;;
run ;
ods tagsets.excelxp close;
%mend corr_rpt;

%corr_rpt;

```

- ✓ Read in product data set
- ✓ Prefix each id with price
- ✓ Fetch the product name

Product name	Baked potato chips	Barbeque pork rinds	Barbeque potato chips	Bread sticks
Baked potato chips	1	-0.14938	-0.11403	0.16121
Barbeque pork rinds	-0.14938	1	0.29435	-0.01687
Barbeque potato chips	-0.11403	0.29435	1	-0.18389
Bread sticks	0.16121	-0.01687	-0.18389	1

Table 1. Correlation Matrix with Variable labels

It is important to report the descriptions about the variable instead of variable names to be more informative for the review. Depending on the type of variables, this can be extended to any number of variables with little tweaking in the above macro.

CONCLUSION

While processing data sets especially larger data sets, speed and time are of essence. Parallel processing, dynamic variable creation, variable list creation and ordering the variables using a reference key data set help to reduce the execution time, amount of coding and also avoid the possible errors that can creep in while handling the larger data sets with numerous variables. Various scenarios were described in which the combination of OPEN, FETCH, and CLOSE Functions with %SYSFUNC and %DO Loops becomes handy and more efficient.

ACKNOWLEDGMENTS

The author would like to thank Ravi Kumar for his suggestions while reviewing this paper.

CONTACT INFORMATION

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

Amarnath Vijayarangan
Emmes Services Pvt Ltd, Bangalore, India
avijayarangan@emmes.com
amarnath7@gmail.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.