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.

```sas
*** 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)));
    %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

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:

```
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))));
```

Display 1. Partial Log Report
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:

![Display 2. Transposed Format of SASHELP.SNACKS Data Set](image)

**SITUATION 2.1**

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

```sas
%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;
```

![Display 3. Transposed Format of SASHELP.SNACKS Data Set with Variable Order](image)
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.

```sas
%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;
```

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

```sas
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.

Display 5. Partial Log Report

**SITUATION 3.2:**

Let us perform correlation analysis among the price variables.

```sas
*** 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;
```

✓ Read in product data set
✓ Prefix each id with price
✓ Fetch the product name
Virtual Accessing of a SAS® Data Set Using OPEN, FETCH, and CLOSE Functions with %SYSFUNC and %DO Loops

<table>
<thead>
<tr>
<th>Product name</th>
<th>Baked potato chips</th>
<th>Barbeque pork rinds</th>
<th>Barbeque potato chips</th>
<th>Bread sticks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baked potato chips</td>
<td>1</td>
<td>-0.14938</td>
<td>-0.11403</td>
<td>0.16121</td>
</tr>
<tr>
<td>Barbeque pork rinds</td>
<td>-0.14938</td>
<td>1</td>
<td>0.29435</td>
<td>-0.01687</td>
</tr>
<tr>
<td>Barbeque potato chips</td>
<td>-0.11403</td>
<td>0.29435</td>
<td>1</td>
<td>-0.18389</td>
</tr>
<tr>
<td>Bread sticks</td>
<td>0.16121</td>
<td>-0.01687</td>
<td>-0.18389</td>
<td>1</td>
</tr>
</tbody>
</table>

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.