Paper 209-2023

List Processing using SQL Select Into to Replace Call Symputx Creating Indexed Arrays of Macro Variables

Ronald J. Fehd, senior maverick, theoretical programmer,
Fragile-Free Software Institute

Abstract

Description : SAS® software provides the ability to allocate a macro variable in a data step with its call symputx routine. This routine can be used to make a sequentially-numbered series of macro variables — mvar1–mvarN — which is refered to as an indexed array of macro variables.

purpose : The purpose of this paper is to examine the algorithm of macro variable array usage and provide sql consolidations of the various tasks.

audience : programmers of all levels

keywords : call symputx, catt, &&mvar&i, %do loop, syslast, sysnobs, subroutine

In this paper:

List of programs:

Introduction

The list processing algorithm has these steps:

- can you use a by statement? No
- identify two or more similar processes
- identify parameters and write subroutine to produce specified output
- make list of parameter values, an associative array
- write output subroutine calls:
  ```
  %subroutine(value=row1,col1) ...
  %subroutine(value=rowM,colN)
  ```

In SAS software, the last step has been divided into these tasks:

- use `call symputx` to create an indexed array of macro variables: `var1, ... varN`
- write a macro with a `%do` loop to fetch each value and call the macro output subroutine

For each different list, the number and names of the variables change, which necessitates writing a custom macro definition with a `%do` loop which can be used only with that list.

This paper shows that sql can produce the text of the macro output subroutine calls eliminating the use of both `call symputx` to create the indexed array of macro variables and a macro definition with a simple `%do` loop. This is done by using sql dictionary.columns to write a list of variables and values; this list is then used in the second list which are the macro calls to the output subroutine.

The key concepts are: what is constant text, and what is variable text.

You'll need this `autoexec.sas` for your programs to call macro definitions in separate files.

```
*name: autoexec.sas;
filename project '.';
options mautosource sasautos = (project sasautos);
```

The output subroutines

overview

Writing the output subroutines defines the variables needed in the list.

The output subroutines provided here are very simple; they require the basic set of variables need to fetch information from a dataset:

libname, memname and variable name.

Note use of SAS software procedure variable names and their reference to sql column names: `data = &libname..&memname ... var = &name`

This section has these topics.

- print_subset_char
- print_subset_freq

Program 1 is a basic reporting subroutine: read data, pick only a subset.

Program 1 print_subset_char

```
%macro print_subset_char
      (data  = &libname..&memname /*global*/
      ,var   = &name              /*global*/
      ,value = );
proc print data = &data (where = (&var eq "&value")) ;
run;
%mend print_subset_char;
```

notes: **&var eq "&value"** this output subroutine is fragile, it won't work with a numeric variable; it's left for the reader to write `print_subset_num.sas`.

As soon as we get program 1 working our customer asks for more information. Program 2 displays the associative array: key and two labels, provided by the frequency procedure: `value`, `count` and `percent`.

Program 2 print_subset_freq

```
%macro print_subset_freq
      (data    = &libname..&memname /*global*/
      ,var     = &name              /*global*/
      ,value   = /*key of associative array */
      ,count   = /*label.1                  */
      ,percent = /*label.2                  */
      );
proc print data  =  &data
             (where = (&var eq "&value")) ;
```

```
title1 "&data (where = (&var eq  &value ))";
title2 "count= &count, percent= &percent";
run;
%mend print_subset_freq;
```

notes: **&var eq "&value"** this output subroutine is also fragile:
it won't work with a numeric variable;

Make list of values from procedures: sort, freq

overview

In this section we write two list-creating subroutines which produce examples of an *associative array*. The sort procedure produces an output dataset with only one variable, no labels. The frequency procedure produces an output dataset, — the *associative array* — with `value` and two labels: `count` and `percent`.
This section has these topics.

- sort, out = sex
- frequency: value, count, percent

sort, out = sex

Program 3 shows the sort procedure, with the option `nokupkey`; this option removes duplicates to provide only the unique values of the variable; compare to sql: `select distinct name`.

Program 3 make-list-values-sort.sas

```
proc sort data = &libname..&memname   (keep = &name)
                  nodupkey
          out  =  list_values_sort;
          by     &name;
run;
```

notes: **keep = &name** this list-creating subroutine produces a list of values with the same name as the variable requested; but this limits the output subroutines that can be called to exactly those with the same name as the variable requested
For reuse, we need to standardize the variable name.

frequency: value, count, percent

Program 4 solves the problem of the variable name being static:
by renaming the input parameter: `(rename = (&name eq &value))`
This `rename` makes this list-creating subroutine robust: no matter which dataset or variable is read, the output dataset has the same names which makes writing a reusable output subroutine easier, as shown above in program 2, `print_subset_freq.sas` on pg. 3.

Program 4 make-list-values-freq.sas

```
proc freq data   = &libname..&memname;
          format   &name;  * remove format;
          tables   &name   / list missing noprint
```

```
            out = list_values_freq
          (rename = (&name = value));
run;
```

notes: **format &name** if the variable has a format associated with it, then this statements removes that association in the output dataset

Loops with call symputx and %do

overview

This section shows the ordinary usage of `call symputx` plus a macro definition with a `%do` loop to call the output subroutine.
This section has these topics.

- one variable: sashelp.class.sex
- three variables from freq: value, count percent

one variable: sashelp.class.sex

Program 5 can process exactly and only the output from program 3, i.e. the input dataset must have a variable named `sex`.
Note that the output macro variable assignment statements from `call symputx` are:

```
%let sex1 = F;
%let sex2 = M;
```

For the algorithm change, we see: *constant text = variable text*
In the sql solution, these macro variable assignment statements are reduced to clauses:

```
(sex = F)
(sex = M)
```

which are the arguments to the output subroutine call.

Program 5 loop_sex.sas

```
%macro loop_sex(subroutine=put echo);
%put info: n-items: &=sysnobs &=syslast;
data _null_;  *let name? =   value;
set  &syslast;
call symputx(catt('sex',_n_),sex);
run;
%do i = 1 %to &sysnobs;
    %&subroutine(value=&&sex&i);
    %end;
%mend loop_sex;
```

notes: **%put info: n-items** this loop routine reads the dataset named in `&syslast`, which was created by program 3; the upper-bound of the `%do i =` loop is the number of observations, which is available in the system macro variable: `&sysnobs`;

Program 6 is the test program for the above program.

Program 6 loop_sex-demo.sas

```
%let libname = sashelp;
%let memname = class  ;
%let    name = sex    ;
%include 'make-list-values-sort.sas';
%loop_sex();%*testing: review calls;
%loop_sex(subroutine=print_subset_char);
```

log

```
81    %loop_sex();%*testing: review calls;
info: n-items: sysnobs=2 syslast=WORK.list_values_sort
echo(value=F)
echo(value=M)

82    %loop_sex(subroutine=print_subset_char);
print_subset_char(value=F)
NOTE: There were 9 observations read
      from the data set SASHELP.CLASS WHERE sex='F';

print_subset_char(value=M)
NOTE: There were 10 observations read
      from the data set SASHELP.CLASS WHERE sex='M';
```

three variables from freq: value, count percent

Program 7 reads the associative array created by program 4, `make-list-values-freq.sas`, on page 4.

Program 7 loop_value_freq.sas

```
%macro loop_value_freq(subroutine=put echo);
%put info: n-items: &=sysnobs &=syslast;
data _null_;  *let name?   =     value;
set  &syslast;
call symputx(catt('value'  ,_n_),value  );
call symputx(catt('count'  ,_n_),count  );
call symputx(catt('percent',_n_),percent);
run;
%do i = 1 %to &sysnobs;
    %&subroutine(value  =&&value&i
                ,count  =&&count&i
                ,percent=&&percent&i);
    %end;
%mend loop_value_freq;
```

Program 8 is the test program for the above.

Program 8 loop_value_freq-demo

```
%let libname = sashelp;
%let memname = class;
%let    name = sex;
%include 'make-list-values-freq.sas';
%loop_value_freq;%*testing: review calls;
%loop_value_freq(subroutine=print_subset_freq);
```

log

```
12  %loop_value_freq;%*testing: review calls;
info: n-items: sysnobs=2 syslast=work.list_values_freq
echo(value = F, count =  9, percent = 47.3)
echo(value = M, count = 10, percent = 52.6)
```

SQL, research on catt and dictionary.columns

overview

In this section we look at the hard-coded sql solutions with a view to designing a list that contains the arguments to the `catt` function: `'name' eq value`.

This section has these topics.

- RnD.0: syntax of catt
- RnD.1: hard-coded solutions
- RnD.2: making list for catt function

RnD.0: syntax of catt

The `catt` function name is *concatenate all strings, after trimming*. The strings can be text, either single or double quoted, or variable names which return values from the row being processed; all strings have trailing spaces removed and are separated by commas.

RnD.1: hard-coded solutions

Program 9 is the first example sql solution; note the *name eq value* pattern.

Program 9 sql-call-sex.sas

```
proc sql; select  catt('%put echo(sex=',sex,')')
            into :list    separated by ';'
          from   &syslast;
          quit;
&list;
```

---

log

```
echo sex=F
echo sex=M
```

Program 10 is the second example sql solution; here there are three variables, delimited by commas.

Program 10 sql-call-freq.sas

```
proc sql; select catt(
          '%put echo(value=',value  ,','
                   ,'count=',count  ,','
                  ,'percent=',percent,')' )
            into :list  separated by ';'
          from   &syslast;
          quit;
&list;
```

---

log

```
echo(value = F, count =  9, percent = 47.4)
echo(value = M, count = 10, percent = 52.6)
```

RnD.2: making list for catt function

Program 11 shows the data structure of sql `dictionary.columns`.

Program 11 sql-d-columns-describe.sas

```
proc sql; describe table dictionary.columns;quit;

NOTE: SQL table DICTIONARY.columns was created like:
      CREATE TABLE DICTIONARY.columns
     (libname char(8)   label='Library Name',
      memname char(32)  label='Member Name',
      memtype char(8)   label='Member Type',
      name    char(32)  label='Column Name',
      type    char(4)   label='Column Type',
      length  num       label='Column Length',
      varnum  num       label='Column Number in Table',
      label   char(256) label='Column Label',
      format  char(49)  label='Column Format',
```

Program 12 illustrates how sql `dictionary.columns` can be used to create the *name = value* clause for the `catt` function.

Program 12 sql-d-columns-demo.sas

```
%let libname = sashelp;
%let memname = class;
%let    name = sex;
%include 'make-list-values-freq.sas';
%let _libname = %scan(&syslast,1,.);
%let _memname = %scan(&syslast,2,.);
proc sql; select  name
            into :list1 separated by ' '
            from  dictionary.columns
           where  libname eq "%upcase(&_libname)"
             and  memname eq "%upcase(&_memname)";
           %put echo &=list1;
```

log

```
echo list1=value count percent
```

select.2

sql can use the catt function to fetch a variable name, add constant text, =, and repeat the variable name

```
          select  catt(name,'=',name)
            into :list2 separated by ','
            from  dictionary.columns
           where  libname eq "%upcase(&_libname)"
             and  memname eq "%upcase(&_memname)";
```

```
%put echo &=list2;
```

log

```
echo list2=value=value count=count percent=percent
```

select.3

this example shows the build of a list of macro variable assignment clauses of the form: `name = value`, separated by commas, which will be used as the argument of the `catt` function in the next statement.

```
select catt('"',name,'=",',name)
into  :list3 separated by ",',',"
  from  dictionary.columns
 where  libname eq "%upcase(&_libname)"
   and  memname eq "%upcase(&_memname)";
 %put echo &=list3;
```

log

```
echo list3="value=",value,',',"count=",count,',',"percent=",percent
```

SQL, call macro from any dataset

overview

In this section we replace the `call symputx` assignments with a read of the variable names from sql `dictionary.columns`; this list of variable names becomes the argument to the `catt` function that writes the output subroutine calls.

This section has these topics.

- sql-call-macro.sas
  - documentation
  - provide defaults
  - loop.1
  - loop.2
  - clean up
- sql-call-macro-demo.sas
- sql-call-macro-demo2.sas

Program 13 is the sql solution that replaces `call symputx` and `%do` loop in a macro definition. This sql loop routine reads the list of variable names of the list-of-values dataset and creates the list of macro variable assignment statements, used in the call to the output subroutine. This list, name_eq_value, is used in the reading of the list-of-values dataset to create the calls of the output subroutine.

Program 13 sql-call-macro.sas

documentation

```
 /* name: sql-call-macro.sas
description: read data set, assemble _subroutine calls
             converts values of all variables
                   to parameters of macro call
    purpose: list processing loop routine
usage: NOTE _UNDERSCORE_ is prefix!
%let _libname    = sashelp;
%let _memname    = class;
%let _subroutine = put echo;
%include 'sql-call-macro.sas';
output: _name_eq_value  : "  name  ="   name;
                            separated by comma
        _call_subroutine: calls of output _subroutine:
        %<subroutine>(<name1>=value(row1,col1), ...
                      <nameN>=value(row1,colN)  )
        ...
        %<subroutine>(<name1>=value(rowX,col1), ...
                      <nameN>=value(rowX,colN)  )
```

provide defaults

```
%sysfunc(ifc(not %symexist(_libname)
         and not %symexist(_memname)
            ,%nrstr(%put note deconstructing &=syslast;
             %let _libname = %scan(&syslast,1,.);
             %let _memname = %scan(&syslast,2,.); )  ,))
%sysfunc(ifc(not %symexist(_subroutine)
            ,%nrstr(%let _subroutine = put echo:; )  ,))
%put trace sql-call-macro beginning%str(;
) &=_libname &=_memname &=_subroutine;
```

loop.1

```
proc sql; select catt('"',name,'=",',name)
          into  :_name_eq_value separated by ",',',"
          from   dictionary.columns
          where  libname eq "%upcase(&_libname)"
            and  memname eq "%upcase(&_memname)"
            and  memtype eq 'DATA';
          %put   echo &=_name_eq_value;
```

loop.2

```
          select catt('%',"&_subroutine("
                          , &_name_eq_value,')' )
          into  :_call_subroutines separated by ';'
          from  &_libname..&_memname;
          quit;
```

! → clean up

```
&_call_subroutines;

%symdel _libname       _memname   _subroutine
        _name_eq_value _call_subroutines;
%put trace sql-call-macro ending;
```

Program 14 is the demo program for the above.

Program 14 sql-call-macro-demo.sas

```
%let libname = sashelp;
%let memname = class;
%let    name = sex;

%include 'make-list-values-sort.sas';
%include 'sql-call-macro.sas';
%let _subroutine = print_subset_char;
%include 'sql-call-macro.sas';

%include 'make-list-values-freq.sas';
%include 'sql-call-macro.sas';
%let _subroutine = print_subset_freq;
%include 'sql-call-macro.sas';
```

log

```
echo:(Sex=F)
echo:(Sex=M)

echo:(value=F,COUNT= 9,PERCENT=47.4)
echo:(value=M,COUNT=10,PERCENT=52.6)
```

Program 15 shows how to use program 13 with any data set.

Program 15 sql-call-macro-demo2.sas

```
%let _libname = sashelp;
%let _memname = class;
*let _subroutine = my_output_subroutine;
%include 'sql-call-macro.sas';
```

log

```
echo:(Name=Alfred ,Sex=M,Age=14,Height=69  ,Weight=112.5)
echo:(Name=Alice  ,Sex=F,Age=13,Height=56.5,Weight= 84  )
...
echo:(Name=Thomas ,Sex=M,Age=11,Height=57.5,Weight= 85  )
echo:(Name=William,Sex=M,Age=15,Height=66.5,Weight=112  )
```

Suggested reading

learning sql :	Hermansen, “Ten Good Reasons To Learn SAS(R) Software’s SQL Procedure”
	Fehd, “How To Use proc SQL select into for List Processing”
	Whitlock, “PROC SQL — Is it a Required Tool for Good SAS Programming?”
	Fehd, “Using SQL Dictionaries to Research the Global Symbol Table”
list processing :	Clay, “Five Easy To Use Macros”
	Crawford, “List Processing — Make Light Work of List Processing in SAS(R)”
	Fehd and Carpenter, “List Processing Basics: Creating and Using Lists of Macro Variables”
	Fehd, “List Processing Routine CallXinc: Calling Parameterized Include Programs Using a Data Set as List of Parameters”
	Fehd, “List Processing Macro Call-Macro”
	Fehd, “List Processing Macro Call-Text”
	Fehd, “Introduction To SCL Functions For Macro Programmers”
	Whitlock, “Names, Names, Names — Make Me a List”
macro arrays, horizontal :	Crawford, “List Processing — Make Light Work of List Processing in SAS(R)”
macro arrays, sql :	Fehd, “How To Use proc SQL select into for List Processing”
	Fehd, “Array: construction and usage of arrays of macro variables”
macro arrays, symputx :	Fehd, “%ARRAY: construction and usage of arrays of macro variables”
	Whitlock, “CALL EXECUTE versus CALL SYMPUT”

Conclusion

The SAS software solution to list processing has two steps:

1. creating an indexed array of macro variables:
   `%let mvar = value1;`
2. then accessing each value in a macro definition `%do` loop:
   `mvar = &&value&i`

The sql solution does the same tasks:

1. creating a list of macro variable assignment clauses:
   `mvar = value1, ...`
2. then writing this clause into a macro call
   `%my_subroutine(mvar1=value1,...)`

Author information

Ronald J. Fehd — Ron.Fehd.macro.maven at gmail dot com

LinkedIn — https://www.linkedin.com/in/ronald-fehd-5125991/

affiliation — senior maverick, theoretical programmer, Fragile-Free Software Institute

also known as — `macro maven` on SAS-L

References

Clay, Ted (Oct. 2006). “Five Easy To Use Macros”. In: *Pacific NorthWest SAS Users Group Annual Conference Proceedings*. 32 pp.; five macros: dir, makefmt, transpo, array and do_over. URL: https://www.lexjansen.com/pnwsug/2006/PN22TedClayFiveMacros.pdf.

Crawford, Peter (Mar. 2006). “List Processing — Make Light Work of List Processing in SAS(R)”. In: *SAS Users Group International Annual Conference Proceedings*. Applications Development, 6 pp.; macro mLoopsX, uses horizontal list. URL: https://support.sas.com/resources/papers/proceedings/proceedings/sugi31/012-31.pdf.

Fehd, Ronald J. (1997). “%ARRAY: construction and usage of arrays of macro variables”. In: *SAS Users Group International Annual Conference Proceedings*. 4 pp.; uses call symput. URL: http://www2.sas.com/proceedings/sugi22/CODERS/PAPER80.PDF.

— (2004). “Array: construction and usage of arrays of macro variables”. In: *SouthEast SAS Users Group Conference Proceedings*. 4 pp.; uses sql select into. URL: http://analytics.ncsu.edu/sesug/2004/SY03-Fehd.pdf.

— (2009). “List Processing Routine CallXinc: Calling Parameterized Include Programs Using a Data Set as List of Parameters”. In: *Western Users of SAS Software Annual Conference Proceedings*. Applications Development, 20 pp.; call execute, data review, data structure, dynamic programming, list processing, parameterized includes, examples. URL: http://www.lexjansen.com/wuss/2009/app/APP-Fehd2.pdf.

— (2010). “How To Use proc SQL select into for List Processing”. In: *SouthEast SAS Users Group Conference Proceedings*. Hands On Workshop, 40 pp.; writing constant text, and macro calls, using macro %do loops; URL: http://analytics.ncsu.edu/sesug/2010/HOW06.Fehd.pdf.

— (Oct. 2014). “List Processing Macro Call-Macro”. In: *MidWest SAS Users Group Annual Conference Proceedings*. Coders Corner, 19 pp.; using %sysfunc with SCL functions to read a list, a control data set, and for each observation, call a macro with variable names and values as named parameters. URL: http://www.mwsug.org/proceedings/2014/BB/MWSUG-2014-BB04.pdf.

— (2016). “List Processing Macro Call-Text”. In: *MidWest SAS Users Group Annual Conference Proceedings*. Tools of Trade, 10 pp.; using %sysfunc with SCL functions to read a list, a control data set, and for each observation, return %unquoted text. URL: http://www.mwsug.org/proceedings/2016/TT/MWSUG-2016-TT06.pdf.

— (Sept. 2022). “Introduction To SCL Functions For Macro Programmers”. In: *Western Users of SAS Software Annual Conference Proceedings*. 20 pp.; URL: https://www.lexjansen.com/wuss/2022/WUSS-2022-Paper-97.pdf.

— (May 2023). “Using SQL Dictionaries to Research the Global Symbol Table”. In: *Pharmaceutical SAS Users Group Conference Proceedings*. 15 pp.; sql, dictionary.dictionaries, dictionary.catalogs, dictionary.columns, dictionary.tables, dictionary.options; catalog and print procedures; finding name collisions in catalogs of formats and macro definitions. URL: https://www.pharmasug.org/proceedings/2023/AP/PharmaSUG-2023-AP-015.pdf.

Fehd, Ronald J. and Art Carpenter (2009). “List Processing Basics: Creating and Using Lists of Macro Variables”. In: *SouthEast SAS Users Group Conference Proceedings*. 8.of.9. URL: http://analytics.ncsu.edu/sesug/2009/HOW008.Fehd.Carpenter.pdf.

Hermansen, Sigurd W. (Mar. 1997). “Ten Good Reasons To Learn SAS(R) Software’s SQL Procedure”. In: *SAS Users Group International Annual Conference Proceedings*. Advanced Tutorials, 5 pp.; URL: https://support.sas.com/resources/papers/proceedings/proceedings/sugi22/ADVTUTOR/PAPER35.PDF.

Whitlock, Ian (1994). “CALL EXECUTE versus CALL SYMPUT”. In: *NorthEast SAS Users Group Conference Proceedings*. URL: http://www.lexjansen.com/nesug/nesug94/NESUG94043.pdf.

— (2001). “PROC SQL — Is it a Required Tool for Good SAS Programming?” In: *SouthEast SAS Users Group Conference Proceedings*. 6 pp., compared to procedures print, sort, summary; sql concepts: cartesian product, fuzzy matching, summarization, macro lists. URL: http://analytics.ncsu.edu/sesug/2001/P-829.pdf.

— (2008). “Names, Names, Names — Make Me a List”. In: *SouthEast SAS Users Group Conference Proceedings*. 11 pp., macro functions %str, %qscan, %superq, %sysfunc, %unquote; nine example macros. URL: http://analytics.ncsu.edu/sesug/2008/SBC-128.pdf.