Paper HOW-006

How To Use Proc SQL select into for List Processing

Ronald J. Fehd, Centers for Disease Control and Prevention, Atlanta, GA, USA

ABSTRACT

Description : The SAS® macro language is simple, yet powerful. List Processing with Proc SQL is also simple, yet powerful. This Hands On Workshop paper provides programmers with knowledge to use the Proc SQL select into clause with the various SQL dictionaries to replace macro arrays and %do loops.

Purpose : list processing, review sql dictionaries, writing text with sql

Audience : intermediate to advanced users, and macro programmers.

Keywords : dynamic programming, list processing, macro, SQL

Information : writing alphabetical list of macro variables, processing items in list

Contents

Introduction

Overview

This paper reviews the theory of programming, how to process every item in a list using the simplicity of Proc SQL select into list processing with SQL's dictionary tables. It provides basic answers to these questions:

1. How do I process every column in a dataset?
2. How do I process every file in a folder?
3. How do I process every member in a libref?

   or
4. How do I process every *item* in a *list*?

Prerequisites

Students are expected to have the following minimum background:

programming : three to seven years

data step : data structure allocation
with attribute or length statements

macro language : allocate macro variables
write macros with one or more steps

procedures : Contents, Print

Topics

This paper cover the following topics:

- programming theory: vocabulary
- Proc SQL syntax
- list processing (dynamic programming) with dictionaries:
  - columns
  - dictionaries, v9
  - filenames, not an sql dictionary, read with scl functions
  - macros
  - options, v9: group
  - tables
- v9 concatenation functions: catt, catx

Basics and Concept Review

Overview

This section provides background information.

Programming Theory and Vocabulary

We communicate in a natural language English (or Chinese, Dutch, French, or German) about the artificial language SAS. These computer science terms and concepts are used throughout this paper.

Terms Used Here		
program	data structure	(also: metadata)
	algorithm	
data structure	attributes	declarative, information (compile-time) statements name : variable or column type : character or numeric length: in bytes character: 1–32,767 numeric : 1– 8 format label see Online Help: Index: declarative DATA step statements

	organization	executable: action or control statements
		array: has numbered elements
		`do I=1 to dim(array-name);`
		`put array-name(I);`
		list : *has unnumbered items*
		`do over array-name;`
		`put array-name;`
		drop, keep
		retain
		where

algorithm	input:	libref, data, variable
	process	perform actions
	output:	libref, data, to log or list

SQL syntax

There are five SQL statements:

1. proc
2. create
3. describe
4. select
5. quit

ProcSQL-0-syntax.sas

```
PROC SQL;
PROC SQL noprint;
         create   table table-name as
                  query-expression
                   <order by order-by-item
                    <, ...   order-by-item>>;
         describe table table-name <, ... table-name>;
         select   <distinct> object-item
                  <function>(object-item)
                  <, ...     object-item>
         into    :macro-variable <separated by ' '>
                 :macro-variable-A, :macro-variable-B
                 :macro-variable1 - :macro-variable&SysMaxLong.
         from     Libref.Data
         where    ColumnChar  eq 'value'
           and    ColumnChar2 eq 'value2'
            or    ColumnNum   eq <num-value>
         group by group-by-item
                   <, ... group-by-item>>
         having   sql-expression
         order by order-by-item
                   <, ... order-by-item>>;
         quit;
```

The keyword `select` has one required clause, `from`, and five optional clauses: `into`, `where`, `group by`, `having`, and `order by`, which may be viewed conceptually in this hierarchy:

```
select
       into
       from
                where
       group by
                having
       order by
```

SQL Basic Processes

Overview

Proc SQL can be used to do each of the basic processes:

- list data structure
- list data
- list only subset
- create data
- make unique list

Listing Data Structure

Proc SQL produces a data structure listing in the log like Proc Contents does in the listing. Instead of the `data =` option, SQL has the `describe table` statement.

ProcSQL-1-describe-table-libref-data.sas

```
PROC Contents data =     SAShelp.Class;

PROC SQL; describe table SAShelp.Class;
          quit;
```

Listing Data

Proc SQL works like Proc Print. In the Print method the object is referred to with the `data =` option. In SQL the object reference is the `from` clause. Star — asterisk (*) — means all variables.

ProcSQL-1-select-star.sas

```
PROC Print data = SAShelp.Class;
           var    _all_;

PROC SQL;  select *           /* _all_ */
           from   SAShelp.Class;
           quit;
```

Listing Subsets

The `where` statement is available in all procedures. This shows it as a data set option. The SQL `select ... from` statement has a `where` clause.

ProcSQL-1-select-subset.sas

```
PROC Print data  =  SAShelp.Class
          (where = (      Sex eq 'F'
                    and   Age ge 14));
           var      Name  Age;

PROC SQL;  select   Name, Age
           from     SAShelp.Class
           where          Sex eq 'F'
                    and   Age ge 14;
           quit;
```

Creating Data

A common task is to copy a permanent data set from a permanent storage library to the work library. The SQL statement `create table` provides a similar data manipulation environment.

ProcSQL-1-create-table-libref-data.sas

```
DATA    Work.Class;
set  SAShelp.Class;

PROC SQL; create table     Work.Class as
                 select *
                 from   SAShelp.Class;
          quit;
```

Making Unique List

The SQL `select` statement has a `distinct` function, which can be used to collapse many instances of variable values into a unique list.

ProcSQL-1-unique.sas

```
PROC Sort data  = SAShelp.Class
         (keep  = Age)
                  nodupkey
          out   = Work.UniqueAge;
          by      Age;

PROC SQL; create  table UniqueAge  as
          select  distinct    Age
          from    SAShelp.Class
          quit;
```

Summary of Basic SQL

There are several differences between the syntax of proc SQL and other procedures.

The two most important to note are that column (variable) names are separated by commas, and dictionary tables' values are upper case.

select : use comma as delimiter between column names

wrong : `select Column1  Column2  Column3`

right : `select Column1, Column2, Column3`

where : values in dictionary tables are upper case

wrong : `where Libname eq 'SAShelp'`

right : `where Libname eq 'SASHELP'`

References

These papers provide an introduction to SQL: Hermansen [26, sugi22.035] Ronk [34, sugi29.268] Wells [39, sugi26.105] Winn, Jr. [43, sugi22.067]

Concatenation Functions

The v9 concatenation functions are used here to replace the double bang (!!), also known as exclamation point, or double vertical bar (||).

Program : This program compares the concatenation operator and cat* functions.

demo-cat-functions.sas

```
DATA Work.Cat_functions;
attrib Verb   length = $16
       Text   length = $32
       Prefix length = $ 8
       Infix  length = $ 8
       Suffix length = $ 8
       One    length =   4
       Two    length =   4
       Three2 length =   8;
Prefix = ’prefix- ’;
Infix  = ’ *infix* ’;
Suffix = ’ -suffix’;
One    = 1;
Two    = 2;
Three2 = 3.2;
Verb = ’concat’;
Text = Prefix !! Infix !! Suffix ;
output;
Verb = ’concat trim’;
Text = trim(Prefix) !! trim(Infix) !! trim(Suffix) ;
output;
Verb = ’concat left trim’;
Text = trim(left(Prefix)) !! trim(left(Infix))
    !! trim(left(Suffix)) ;
output;
Verb = ’cat’;
Text = cat(Prefix,Infix,Suffix) ;
output;
Verb = ’catt’;
Text = catt(Prefix,Infix,Suffix) ;
output;
Verb = ’cats’;
Text = cats(Prefix,Infix,Suffix) ;
output;
Verb = ’catx’;
Text = catx(’+’,Prefix,Infix,Suffix) ;
output;
Verb = ’catx numbers’;
Text = catx(’+’,One,Two,Three2) ;
output;
stop;
Proc Print;
```

Listing : Notice the presence of double spaces in obs one and four.

demo-cat-functions.lst

```
Obs    Verb                Text

 1     concat              prefix-  *infix* -suffix
 2     concat trim         prefix- *infix* -suffix
 3     concat left trim    prefix-*infix*-suffix
 4     cat                 prefix-  *infix* -suffix
 5     catt                prefix- *infix* -suffix
 6     cats                prefix-*infix*-suffix
 7     catx                prefix-+*infix*+-suffix
 8     catx numbers        1+2+3.2
```

Writing Constant Text

Overview

SQL provides the ability to select text as well as columns in its queries. In this section we examine first a simple example of selecting text and then see how to place both values from columns and surrounding text which comprise complete SAS statements into a macro variable, which is used to submit the text for processing.

In this section:

Selecting Text

The select select statement accepts strings as one of its arguments; each string can be either single- or double-quoted, which allows the use of macro variables.

ProcSQL-2-select-constant-text.sas

```
%Let Rows = rows;

PROC SQL; select MemName, ’has’, Nobs, "&Rows."
          from   Dictionary.Tables
          where  LibName eq "SASHELP"
            and  MemName eq "CLASS"
            and  MemType eq "DATA";
          quit;
```

Note: the length of column MemName is 32, which accounts for the wide space between the words `Class` and `has`.

ProcSQL-2-select-constant-text.sas

```
                                          Number of
                                           Physical
Member Name                            Observations
---------------------------------------------------------
CLASS                             has            19  rows
```

Creating a Macro Variable

This example shows how to concatenate concatenate text and variable value, using the catt (concatenation, trimmed) function and put that text into a macro variable.

For each row the statement is:

`Proc Contents data = SAShelp.`*`MemName`*`;`

ProcSQL-2-select-text-into-List.sas

```
%Let List = *missing-;* initialize for no rows selected;

PROC SQL; select catt(’Proc Contents data = SAShelp.’
                     ,MemName
                     ,’;run;’
                     )
          into  :List   separated by ’ ’
          from   Dictionary.Tables
          where  LibName eq   "SASHELP"
            and  MemName like "V%"
            and  MemType eq   "VIEW";
          quit;
&List.;* execute statements in mvar List;

```

Notes:

into : The `into` clause creates the macro variable named after the colon. Notice the the last argument of the catt function is a semicolon and that the delimiter of `separated by` is space. This could also be: `catt(...,Memname) ...separated by ';'`.

like : The `like` operator chooses only names beginning with 'V'; these tables correspond to sql dictionaries.

quit : The statements in the macro variable are procedure statements, therefore they must be executed *after* the `quit` statement.

List Processing Statements

These statements are in the macro variable List.

ProcSQL-2-select-text-into-List.lst, snip 1

```
Proc Contents data = SAShelp.VALLOPT;
Proc Contents data = SAShelp.VCATALG;
...
Proc Contents data = SAShelp.VVIEW;
```

Program Output

Output from statements in the macro variable List.

ProcSQL-2-select-text-into-List.lst, snip 2

```
The CONTENTS Procedure

Data Set Name      SASHELP.VALLOPT        Observations        .
Member Type        VIEW                   Variables           6
Engine             SQLVIEW                Indexes             0
```

References

SAShelp views associated with SQL dictionaries are reviewed by DiIorio and Abolafia [15, sugi29.237].

Compare with program `ProcSQL-D-Dictionaries-list-describe-table` below, which lists SQL dictionaries.

List Processing Summary

There are several steps in writing statements to a macro variable and executing them:

1. input: data structure
    (a) identify the input table
    (b) examine its data structure
2. process: expected text
    (a) concatenation of text and values into macro variable
        i. clarify the text surrounding the variable value(s):
           prefix, infix(es), suffix
        ii. identify subset, if any; are values in ALL CAPS?
        iii. volume of text in macro calls: can information be gotten by the subroutine macro rather than passed as a parameter? this caveat is especially important when passing long character variables such as data set or variable labels.
           see: ¶ problems below
    (b) remember closure or delimiter:
        clause(s): comma, space, or other
        statement(s): semicolon (`;`)
        step boundary: `run;`
    (c) execute the statements: SQL: *before* `quit;`
        SAS: procedures, macros: *after* `quit;`
3. output: consider ODS

In the next sections we use this list processing check list to examine several of the more commonly used dictionaries.

Commentary

The first examples in the next section — dictionaries, macros and options — illustrate writing constant text. In the second section we look at how to write macro calls to generate more complicated amounts of text when reading columns (variables), filenames, and tables.

Dictionary Dictionaries

Overview

SQL provides a meta-dictionary, Dictionary.Dictionaries, which contains a description of all other dictionaries.

Can we use this meta-dictionary to write statements which describe all the other dictionaries?

Problem Statement

The task is to write these statements:

`describe table Dictionary.`*`MemName`*`;`

Dictionary.Dictionaries is unique on MemName and Name (Column Name).

We need to make a data set (list) unique on MemName.

Review Data Structure

The `describe table` statement lists the data structure.
Note: V9 only.

ProcSQL-D-Dictionaries-describe-table.sas

```
PROC SQL; describe table Dictionary.Dictionaries;
          quit;
```

The data structure is written to the Log:

ProcSQL-D-Dictionaries-describe-table.log

```
create table DICTIONARY.DICTIONARIES
  (
   memname char(32) label='Member Name',
   memlabel char(256) label='Dataset Label',
   name char(32) label='Column Name',
   type char(4) label='Column Type',
   length num label='Column Length',
```

Program

This program makes a data set with the names of all the SQL dictionaries using `select distinct` function, writes `describe table` statements for each, then executes those statements. The log contains the data structure of each SQL dictionary.

ProcSQL-D-Dictionaries-list-describe-table.sas

```
*note: need v9 for D.Dictionaries;
Proc SQL; create table  List as
                 select distinct MemName
                 from   Dictionary.Dictionaries;
          select catt('describe table Dictionary.'
                     , MemName
                     ,';'
                     )
          into   :List   separated by ' '
          from    List ;
                 &List.;
          quit;
run;
```

The statements in the macro variable are SQL statements, therefore they must be executed *before* the `quit` statement.

List Processing Statements

These are the statements in the macro variable List:

ProcSQL-D-Dictionaries-describe-table.lst

```
describe table Dictionary.CATALOGS;
describe table Dictionary.CHECK_CONSTRAINTS;
...
describe table Dictionary.VIEWS;
```

Program Output

SQL table descriptions are written to the log.

ProcSQL-D-Dictionaries-describe-table.log

```
NOTE: SQL table DICTIONARY.CATALOGS was created like:
...
NOTE: SQL table DICTIONARY.CHECK_CONSTRAINTS was created like:
...
NOTE: SQL table DICTIONARY.VIEWS was created like:
```

Commentary

Compare with program `ProcSQL2-select-text-into-List` above, which lists SAShelp views.

This program illustrates a two-statement solution to this problem. See program `ProcSQL-D-Options-list-groups` below for a single-statement solution.

Dictionary Macros

Overview

SQL maintains a list of all macro variables.

One problem with global macro variables is checking their values.

The statement `%Put _user_` writes an unsorted list of macro variable names and values to the log.

Put-User.log

```
1          %Let A = 1;
2          %Let b = 22;
3          %Let z = end of list;
4          %Put _user_;
GLOBAL Z end of list
GLOBAL A 1
GLOBAL B 22
```

Problem Statement

The task is to write a `%put` statement for each macro variable. Hard-coded this would be:

```
%put a: &a.;
%put b: &b.;
%put z: &z.;
```

Review Data Structure

These statements list the data structure of Dictionary.Macros:

ProcSQL-D-Macros-describe-table.sas

```
PROC SQL; describe table Dictionary.Macros;
          select   distinct Scope
          from     Dictionary.Macros;
          quit;
run;
```

Here is the data structure:

ProcSQL-D-Macros-describe-table.log

```
create table DICTIONARY.MACROS
  (
   scope char(32) label='Macro Scope',
   name char(32) label='Macro Variable Name',
   offset num label='Offset into Macro Variable',
   value char(200) label='Macro Variable Value'
```

The values of scope are:

ProcSQL-D-Macros-describe-table.lst

```
Macro Scope
-----------
AUTOMATIC
GLOBAL
```

Program

This program writes the `%put` statements into the macro variable named List and executes them.

ProcSQL-D-Macros-select-into-list-ordered.sas

```
%Let Z = the last one;
%Let A = 1st item;
%Let M = middle;
Proc SQL; select   catx(’ ’,’%Put’,Name
                           ,’:’   ,Value
                           ,’;’
                       )
          into    :List   separated by ’ ’
          from     Dictionary.Macros
          where    Scope eq ’GLOBAL’
            and    not(Name like ’SQL%’)
          order by Name;
          quit;
&List.;
```

Notes:

quit : The statements in the macro variable are SAS statements, therefore they must be executed *after* the `quit` statement.

Exclusion : The clause `not(Name like ’SQL%’)` excludes the macro variables created by the proc sql statement: SqlObs, SqLoops, SqlXobs, and SqlRc.

List Processing Statements

The statements in macro variable List:

ProcSQL-D-Macros-select-into-list-ordered.lst

```
%Put A : 1st item ;
%Put M : middle ;
%Put Z : the last one ;
```

Program Output

The statements written to the log are a sorted list of global macro variables.

ProcSQL-D-Macros-select-into-list-ordered.log

```
14         &List.;
A : 1st item
M : middle
Z : the last one
```

Dictionary Options

Overview

Let’s take a look at Dictionary.Options. These next programs show how to find out:

- definitions and values of each option
- what options are in each group

Problem Statement

The task is to write these statements:

`Proc Options define value option = OptName;`

Review Data Structure

What is the data structure?

ProcSQL-D-Options-describe-table.log

```
create table DICTIONARY.OPTIONS
  (
   optname char(32) label=’Option Name’,
   opttype char(8) label=’Option type’,
   setting char(1024) label=’Option Setting’,
   optdesc char(160) label=’Option Description’,
   level char(8) label=’Option Location’,
   group char(32) label=’Option Group’
```

Note: group is available in v9.

Program

This program writes text for each option.

ProcSQL-D-Options-define-value.sas

```
PROC SQL; select catt(’Proc Options define value option = ’
                     ,OptName
                     ,’;’
                     )
          into  :List   separated by ’ ’
          from   Dictionary.Options;
          quit;
&List.;
run;
```

List Processing Statements

These statements are in macro variable List:

ProcSQL-D-Options-define-value.lst

```
Proc Options define value option =APPLETLOC;
Proc Options define value option =ARMAGENT;
...
Proc Options define value option =XCMD;
```

Program Output

Here is the list of values and definitions for the first option written to the log.

ProcSQL-D-Options-define-value.log

```
Option Value Information For SAS Option APPLETLOC
    Option Value: C:\Program Files\SAS\Shared Files\applets\9.1
    Option Scope: SAS Session
    How option value set:  Config File(s)
Option Definition Information for SAS Option APPLETLOC
    Group= ENVFILES
    Group Description: SAS library and file location information
    Description: Location of Java applets
    Type: The option value is of type CHARACTER
```

Options in Each Group

In program `ProcSQL-D-Dictionaries-list-describe-table` above, using the distinct function, we created a table of the unique dictionaries before we could write the text of the `describe table` statements. We can reduce two statements to one by creating a named column with the distinct values and then refer to the new column in our text concatenation. This trick requires that we make two macro variables, Item and List, for each of the columns: Group as Item, and text.

Using Calculated Columns

This example shows that the variable value can be used more than once, and there is no limit to how much text can be concatenated, either before or after use of the variable value.

ProcSQL-D-Options-list-distinct-groups.sas

```
Proc SQL; select distinct Group as Item,
                 catx(’ ’,’%Put Group: ’
                         , calculated Item
                         ,’;Proc Options group =’
                         , calculated Item
                         ,’;run;’
                     )
          into   :Item, :List  separated by ’ ’
          from   Dictionary.Options;
          quit;
&List.;
```

The `calculated` keyword indicates that the column Item has been created — `distinct Group as Item` — and is not in the the table being read: `from Dictionary.Options`.

List Processing statements

Note that the column Item retains the label of Group.

ProcSQL-D-Options-list-distinct-groups.lst

```
Option Group
----------------------------------------------------------
COMMUNICATIONS   %Put Group: COMMUNICATIONS ;
                 Proc Options group = COMMUNICATIONS ;run;
DATAQUALITY      %Put Group: DATAQUALITY ;
                 Proc Options group = DATAQUALITY ;run;
```

The `run` statement ensures that the the `%Put Group:` statement is written before each group.

Program Output

This is the first group written to the log.

ProcSQL-D-Options-list-groups.log

```
12         &List.;
Group: COMMUNICATIONS
    SAS (r) Proprietary Software Release 9.1  TS1M3

 NOAUTOSIGNON   SAS/CONNECT remote submit will not
                automatically attempt to SIGNON
 COMAMID=TCP    Specifies the communication access method
                to be used for SAS distributed products
```

Writing Macro Calls

Overview

Macros provide the ability to generate more than a single statement and also conditional processing of parameter values.

In the following sections we examine program templates for processing the three most commonly used lists:

- columns (variables)
- tables (data set names)
- files

Several macro templates are provided for your later usage that show minimal processing of the parameters passed.

In this section:

Macros

Macro ProcDsn

This macro is an example program that processes a data set in a library.

ProcDsn.sas

```
 /*    name: ProcDsn.sas
description: process a data set
purpose    : called by SQL list processing
usage:
%ProcDsn(Data=SAShelp.PrdSal2,MemType=data);
******/
%Macro ProcDsn(Data    = &Libname..&MemName. /* global */
              ,MemType = data          /* catalog view */
              ,Testing = 0);

*upcase for logical expression evaluation;
%let MemType= %upcase(&MemType.);

%let Testing = %eval(&Testing
          or   %sysfunc(getoption(mprint)) eq MPRINT);

%if &Testing %then %put _local_;

%if       &MemType. eq CATALOG %then %do;
      Proc Catalog catalog = &Data.;
                   contents;
                   quit;
      %end;
%else %if &MemType. eq DATA %then %do;
      Proc SQL; describe table &Data.;
                quit;
      %end;
%else %if &MemType. eq VIEW %then %do;
      Proc Contents data = &Data.;
      run;
      %end;
%else %Put &Data. type unknown: &type.;

title3 "&Data. type: &MemType.";
%Mend;
```

Macro ProcFile

This macro is an example program that processes a file in a folder.

ProcFile.sas

```
 /*    name: ProcFile.sas
description: process a file in a folder
purpose    : called by SQL list processing
******/
%Macro ProcFile(Directory = &Folder. /*global*/
               ,Filename  = );
%if %index(&Filename.,.) %then
      %put note: filename.ext:<&FileName.>;
%else %put note: other<&FileName.>;
run; %mend;

```

Macro ProcVar

This macro is an example program that processes character and numeric variables with different procedures.

ProcVar.sas

```
 /*    name: ProcVar.sas
description: process a variable
purpose    : called by SQL list processing
usage:
%ProcVar(Data=SAShelp.Class,Name=Sex,Type=char)
NOTE: SQL dictionary.tables.type in (char,num)
      compare Contents.type in (1==num,2==char)
******/
%Macro ProcVar(Data    = &Libname..&MemName. /* global */
              ,Name    = Sex
              ,Type    = char
              ,Testing = 0);

*lowcase for logical expression evaluation;
%let Name = %lowcase(&Name.);

%Let Testing = %eval(&Testing
           or  %sysfunc(getoption(mprint)) eq MPRINT);

%If &Testing. %then %put _local_;

%if       &Type. eq char %then %do;
      Proc Freq       data     = &Data.;
                      tables   &Name.;
      %end;
%else %if &Type. eq num  %then %do;
      Proc Summary    data = &Data. print;
     *Proc Univariate data = &Data.;
                      var    &Name.;
      %end;

title2 "&Data..&Name. type: &Type.";
run;%Mend;
```

Dictionary Columns

Overview

Dictionary Columns contains the attributes of each variable in a data set, including name, type, length, label and format.

Problem Statement

The task is to process every variable in a data set, or: every column in a table.

Review Data Structure

ProcSQL-D-Columns-describe-table.sas

```
PROC SQL; describe table Dictionary.Columns;
          quit;
```

ProcSQL-D-Columns-describe-table.log

```
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’,
```

Program

This program processes every variable in a data set, using the macro ProcVar.

ProcSQL-D-Columns-select-into-list.sas

```
options mprint ; *testing: view macro statements;
options source2; *testing: echo include to log;

%Let SQLprint =   print;*testing;
*Let SQLprint = noprint;

%Let LibName  = SAShelp;

%Let MemName  = Class;
*Let MemName  = PrdSal2;
*Let MemName  = PrdSal3;
*Let MemName  = PrdSale;

%Let MemType  = data;
*Let MemType  = view;

%Include ’ProcVar.sas’;

PROC SQL &SQLprint.;
         select catt(’%ProcVar(name=’ , Name
                    ,’,type=’         , Type
                    ,’)’
                    )
         into  :List   separated by ’ ’
         from   Dictionary.Columns
         where  LibName eq "%upcase(&LibName.)"
           and  MemName eq "%upcase(&MemName.)"
           and  MemType eq "%upcase(&MemType.)";
         quit;
&List.; *execute;
```

Notes:

Testing : Place an asterisk (*) before the options statements to disable either or both of the macro option mprint or include option source2.

Change the `%Let` verb to `*Let` to disable SQL print of query results.

Parameters : The program has three required parameters:

LibName : The primary parameter is the libref LibName.

MemName : The secondary parameter is the table name, MemName, in the libref.

MemType : A tertiary parameter, provided for clarity, is the member type, MemType, in (data, view).

Macro : The `%Include` statement brings the macro definition into the program.

List Processing Statements

These are the statements in the macro variable List. Spaces have been added for clarity.

ProcSQL-D-Columns-select-into-list.lst

```
%ProcVar(name=Name  ,type  =char)
%ProcVar(name=Sex   ,type  =char)
%ProcVar(name=Age   ,type  =num )
%ProcVar(name=Height,type  =num )
%ProcVar(name=Weight,type  =num )
```

Program Output

The macro ProcVar generates steps.

ProcSQL-D-Columns-select-into-list.lst

```
58         &List.; *execute;
MPRINT(PROCVAR):   Proc Freq data = SAShelp.Class;
MPRINT(PROCVAR):   tables Name;
...
MPRINT(PROCVAR):   Proc Freq data = SAShelp.Class;
MPRINT(PROCVAR):   tables Sex;
...
MPRINT(PROCVAR):   Proc Summary data = SAShelp.Class print;
MPRINT(PROCVAR):   var Age;
```

Problems

There are limits associated with this method.

catt : The catt function returns a value whose maximum length is 200 in proc sql.

RnD-catt-function-warning.log

```
WARNING: In a call to the CATT function, the buffer allocated
         for the result was not long enough to contain the
         concatenation of all the arguments. The correct
         result would contain 200+ characters, but the actual
         result may either be truncated to 200 character(s) or
         be completely blank, depending on the calling
         environment.
```

macro variable : The maximum length of a macro variable is $2^{16} - 2$ = 65534.

These statements show the error.

RnD-macro-variable-length-error.log

```
2          %Let MaxLen = %eval(2**16);
3          %Put MaxLen = &MaxLen.;
MaxLen = 65536
4          %Let List = %sysfunc(repeat(*,&MaxLen.-2));
ERROR: The text expression length (65535)
       exceeds maximum length (65534). The text expression
       has been truncated to 65534 characters.
```

Workarounds

These are some workarounds for consideration:

- replace catt function with concatenation operator: double bang or exclamation points: (!!) or double vertical bars (||)
- reduce macro name to one character
- replace named parameters with positional parameters
- eliminate macro parameters when information can be discovered by the subroutine macro; e.g.: variable name is a foreign key whose associated information — type, etc. — can be gotten by subroutine macro ProcVar.

Here is an example program showing the workarounds.

```
%let LibName  = SAShelp;
%let MemName  = Class;
*previous: Macro ProcVar(data=,name=,...);
%Macro V(Name      /* positional parameter */
                   /* global macro vars: */
        ,Data    = &LibName..&Memname.
        ,Testing = 0);
*todo: get information: type, format, etc.;
...
PROC SQL &SQLprint.;
         select ’%V(’ !! trim(Name) !! ’)’
...
```

Dictionary Tables

Overview

This program illustrates the concept of having one program define the parameters of another.

The task is to process every data set in a library. What are the columns whose values are to be passed to the processing macro?

Review Data Structure

This program lists the data structure of Dictionary Tables;

ProcSQL-D-Tables-describe-table.sas

```
PROC SQL; describe table Dictionary.Tables
        ; quit;
run;
```

ProcSQL-D-Tables-describe-table.log

```
create table DICTIONARY.TABLES
  (
   libname char(8) label=’Library Name’,
   memname char(32) label=’Member Name’,
   memtype char(8) label=’Member Type’,
   nobs num label=’Number of Physical Observations’,
   ...
   nvar num label=’Number of Variables’,
```

Program

This program processes every data set in a library, using the macro ProcDsn.

ProcSQL-D-Tables-select-into-list.sas

```
options mprint;*testing: view macro statements;
options source2;*testing: echo include to log;

%Let SQLprint = noprint;*production;
%Let SQLprint =   print;*testing;

%Let LibName = SAShelp;%*testing;

*    Libname   MyLib ’<folder-name>’;
*Let LibName = MyLib; *production;

%Let MemType = ’data’; *NOTE: MUST BE QUOTED;
*Let MemType = ’catalog’ ’data’ ’view’;

%Include ’ProcDsn.sas’;

%Let List = *empty: no rows selected;

PROC SQL &SQLprint.;
         select catt(’%ProcDsn(data=’,MemName
                    ,’,type =’       ,MemType
                    ,’,nobs =’       ,Nobs
                    ,’,nvar =’       ,Nvar
                    ,’)’
                    )
         into  :List       separated by ’ ’
         from   Dictionary.Tables
         where  LibName eq "%upcase(&LibName.)"
           and  MemType in (%upcase(&MemType.));
         quit;
```

```
&List.;*execute macro calls;
%symdel List;
```

Testing : Place an asterisk (*) before the options statements to disable either or both of the macro option mprint or include option source2.

Change the `%Let` verb to `*Let` to disable SQL print of query results.

Parameters : These parameters are required.

LibName : The primary parameter is the libref, LibName.

MemType : Libraries contain catalogs, data sets, and views. Remember to provide the value in single or double quotes.

List Processing Statements

As in the processing of Dictionary.Columns program above, the statements written include most of the list attributes to be passed to the macro.

ProcSQL-D-Tables-select-into-list.lst

```
%ProcDsn(data=ADOMSG,type =DATA,nobs =  458,nvar =6)
%ProcDsn(data=ADSMSG,type =DATA,nobs =  426,nvar =6)
%ProcDsn(data=AFMSG ,type =DATA,nobs = 1088,nvar =6)
...
%ProcDsn(data=ZTC   ,type =DATA,nobs =18161,nvar =6)
```

Program Output

ProcSQL-D-Tables-select-into-list.lst

```
65         &List.;*execute macro calls;
SAShelp.ADOMSG nobs: 458 nvar: 6
MPRINT(PROCDSN):   Proc SQL;
MPRINT(PROCDSN):   describe table SAShelp.ADOMSG;
NOTE: SQL table SASHELP.ADOMSG was created like:
```

Compare with program ProcSQL-D-Columns-select-into-list, above.

Filenames

Overview

Processing a list of filenames is different from the previous examples as there is no SQL dictionary of folders and filenames. Reading the list of files using SQL requires some interesting tricks. This example is a polished version of a program written by Hamilton [25, sugi31.046]

Program

This program processes every file in a folder, using the macro ProcFile. It uses the scl functions: filename, dopen, and dread.

ProcSQL-filenames-select-into-list.sas

```
 /*       Name: ProcSQL-select-into-list-from-filenames.sas

Requirements  : description  : Process Each File in Folder
                purpose      : list processing: read file list
                                                process each file

-------------------------------------------------------------------
Contexts      : program group: list processing
                program  type: routine
                SAS      type: program with macro
                uses routines: in-program macro %ProcFile

-------------------------------------------------------------------
Specifications: input  : folder: directory-specification
                process: open folder
                         read filenames
                         SQL  writes macro calls of ProcFile
                         execute list: ProcFile of each filename
                output : user-defined in macro ProcFile

-------------------------------------------------------------------
Usage Examples:
%Let Folder   = c:\;

Information   : concept author: Jack Hamilton,
                DIGITS and DATES The SQL Procedure Goes "Loopy"
                  Proceedings of the 31st Annual SAS(R) Users
                  Group International Conference, 2006.
                http://www2.sas.com/proceedings/sugi31/046-31.pdf
                polishing: Ronald J. Fehd
                           for PNWSUG-2006 Hands On Workshop
NOTE: The execution of this query involves performing one
      or more Cartesian product joins that can not be optimized.
*** .................................... */

%Let Folder   = c:\;
*Let Folder   = .; *here;
*Let Folder   = %sysget(sasroot); *directory-spec;
*Let Folder   = %sysget(sasroot)\core\sasexe; *922 files;

options mprint ;*testing: echo macro    statements;
options source2;*testing: echo included statements;

%Let SQLprint = noprint; *production;
%Let SQLprint =   print; *testing;

%Include ’ProcFile.sas’;

Data Work.Digits;
length Digit 4;
do Digit = 0 to 9;
   output;
   end;
stop;
run;

* see: from (&&E&Evalue.);
%Let E1 = select ones.digit  as FileNmbr from Digits
```

```
                             as ones;
%Let E2 = select ones.digit  +  10 * tens.digit
                             as FileNmbr
                 from Digits as ones, digits as tens;
%Let E3 = select ones.digit  +  10 * tens.digit
                             +  100 * hundreds.digit
                             as FileNmbr
                 from Digits as ones, digits as tens,
                                      digits as hundreds;
%Let E4 = select ones.digit  +  10   *      tens.digit
                             +  100  *  hundreds.digit
                             +  1000 * thousands.digit
                             as FileNmbr
                 from Digits as ones, digits as      tens,
                                      digits as  hundreds,
                                      digits as thousands;
Proc SQL &SQLprint.;
         select filename(’DirSpec’, "&Folder."),
                   dopen(’DirSpec’) as Dir_id,
                    dnum(calculated    Dir_id)
         into  :FileName_Rc,  :Dir_id,  :NmbrFiles
         from   Digits   where Digit = 0;
         %Let NmbrFiles = &NmbrFiles.; *trim;
         %Let Evalue    = %length(&NmbrFiles.);
         select catt(’%ProcFile(filename=’
                    ,dread(&Dir_id., FileNmbr)
                    ,’)’
                    )
         into  :List   separated by ’ ’
         from  (&&E&Evalue.)
         where  FileNmbr between 1 and &NmbrFiles;
         quit;
%Let FileName_Rc = dclose(&Dir_id.);
&List.;
%symdel E1 E2 E3 E4 Evalue
        Folder List SQLprint
        FileName_Rc Dir_id NmbrFiles;
```

Testing : Place an asterisk (*) before the options statements to disable either or both of the macro option mprint or include option source2.

Change the `%Let` verb to `*Let` to disable SQL print of query results.

Parameters :

Folder : The primary parameter is the folder name.

Macro : The `%Include` statement brings the macro definition into the program.

Data Set : The Digits data set is used to make a larger data set, FileNmbrs, containing the file-numbers used by the dread function.

Macro Variables E* : Macro variables E1–E4 are necessary for processing lists of 9, 99, 999 or 9999 files.

Select : Two select statments are required: the first to assign a fileref, open the folder and read the number of files present; the second statement writes the macro calls.

Using Macro %Do Loops

Overview

This section provides two methods which replace the `select ...into :List` clause which is fragile.

These methods are:

- delimited list: one macro variable contains the list of values and each value is delimited by a special character
- macro array: a sequentially-numbered set of macro variables

These methods are better than the set of workarounds shown above.

Note: Programs shown here are calling routines for macro ProcDsn. The zip file provided contains similar programs for macro ProcVar:

- ProcVars
- ProcVars-caller
- ProcVars-Test
- ProcVarsDL
- ProcVarsDL-Test

List processing techniques shown here are from Fehd and Carpenter [22, sgf2007.113].

In this section:

Using Items in Delimited List

Overview

This method stores all values in one macro variable with each item separated by a delimiter; the items are retrieved for processing using the scan function.

demo-delimited-list.log

```
1          %Let List = ItemA * ItemB * itemC;
2          %Let Item1 = %scan(&List.,1,*);
3          %Put Item1: &Item1.;
Item1: ItemA
4          %Let Item3 = %scan(&List.,3,*);
5          %Put Item3: &Item3.;
Item3: itemC
```

Program

This program processes every data set in a libref, using the macro ProcDsn and the method of one macro variable containing a delimited list of all values.

ProcDsnsDL.sas

```
 /*    name: ProcDsnsDL.sas
description: process all data sets in libref
             using named macro
purpose    : list processing
note       : uses delimited list of values
usage:
%ProcDsns(libref=SAShelp,macroname=ProcDsn);
******/
%Macro ProcDsnsDL(LibName    = SAShelp
               ,MemType   = data /* or view */
               ,MacroName = put ProcDsn
               ,SQLprint  = noprint
               ,Testing   = 0);

%Let Testing = %eval(&Testing
           or  %sysfunc(getoption(mprint)) eq MPRINT);

%if &Testing. %then %let SQLprint = print;

%local Dlm I;
%Let Dlm = *;

PROC SQL &SQLprint.;
         select MemName, MemType
         into  :List_Name separated by "&Dlm."
             , :List_Type separated by "&Dlm."
         from   Dictionary.Tables
         where  LibName eq "%upcase(&LibName.)"
           and  MemType eq "%upcase(&MemType.)";
         quit;

%if &Testing %then %put _local_;

%do I = 1 %to &SqlObs.;
    %Let Item_Name = %scan(&List_Name,&I.,&Dlm.);
    %Let Item_Type = %scan(&List_Type,&I.,&Dlm.);
    %if &Testing %then %do;
        %put note2:Item_Name&I: &Item_Name.;
        %put note2:Item_Type&I: &Item_Type.;
        %end;
    %&MacroName.(libname=&Item_Name.,memtype=&Item_Type.);
    %end;
run;
%Mend;
```

Program: Testing

This program provides a unit test of the routine ProcDsnsDL.

ProcDsnsDL−Test.sas

```
*name: ProcDsnsDL−Test;
options mprint ; *testing: view macro statements;
*either of:
*autoexec:;
*filename Project ’.’;
*options sasautos = (Project SASautos);
*
*or;
%Include ’ProcDsnsDL.sas’ ’ProcDsn.sas’;
*unit test;
*default = SAShelp;
%ProcDsns();

%ProcDsnsDL(Libname=SAShelp,MacroName = ProcDsn);

*Libname Library ’<directory-specification>’;
*ProcDsnsDL(Libname=Library);
```

Using Arrays of Macro Variables

Overview

This method stores each value in a macro variable; the items are retrieved for processing using an index variable.

demo-macro-array.log

```
1          %Macro DemoMacroArray();
2          %let Item1 = valueA;
3          %let Item2 = value.B;
4          %let Item3 = value C;
5          %let Dim   = 3;
6          %do I = 1 %to &Dim.;
7              %Put Item&I.: &&Item&I.;
8              %end;
9          %Mend;
10         %DemoMacroArray
Item1: valueA5
Item2: value.B
Item3: value C

```

Program

This program processes every data set in a libref, using the macro ProcDsn and the macro array method of allocating a series of sequentially-numbered macro variables for each value.

ProcDsns.sas

```
 /*    name: ProcDsns.sas
description: process all data sets in libref
             using named macro
purpose    : list processing
usage:
%ProcDsns(libref=SAShelp,macroname=ProcDsn);
******/
%Macro ProcDsns(LibName   = SAShelp
               ,MemType   = data /* or view */
               ,MacroName = put ProcDsn
               ,SQLprint  = noprint
               ,Testing   = 0);
%local I;
%let   I = 0;

%Let Testing = %eval(&Testing
           or  %sysfunc(getoption(mprint)) eq MPRINT);

%if &Testing. %then %let SQLprint = print;

PROC SQL &SQLprint.;
         select MemName,    MemType
         into  :MemName1 - :MemName&SysMaxLong.
             , :MemType1 - :MemType&SysMaxLong.
         from   Dictionary.Tables
         where  LibName eq "%upcase(&LibName.)"
           and  MemType eq "%upcase(&MemType.)";
         quit;

%if &Testing %then %put _local_;

%do I = 1 %to &SqlObs.;
    %&MacroName.(data=&Libname..&&MemName&I.,memtype=&&MemType&I.);
    %end;
run;
%Mend;
```

Notes on program ProcDsns:

Testing : Fehd [19, nesug07.cc12] shows the discipline of using options to turn on self-reporting by macro routines.

SysMaxLong : This macro variable contains the largest integer available to the operating system and is used to ensure the upper bound of the macro array is not truncated.

Usage here replaces any smaller number — `:MemName9999` — which may result in allocating less than the number of macro variables of the rows in the table; i.e.: no macro variable `MemName10000` is created.

Double dots : The prefix of a macro variable reference is an ampersand (&); the suffix of a reference is a dot (.). The dot suffix is necessary for a two-level data set name: Work.Data:

```
&Libname..&MemName.
```

Item reference : A reference to an item in the macro array contains the array name and the index. The array name must be preceeded by two ampersands in order to resolve correctly:

```
&&MemName.&I. :: &MemName1
```

Program: Module

This program is a module and provides a template for use of the macro routine ProcDsns.

ProcDsns-caller.sas

```
*name: ProcDsns-caller;
options mprint ; *testing: view macro statements;
*either of:
*autoexec:;
*filename Project ’.’;
*options sasautos = (Project SASautos);
*
*or;
%Include ’ProcDsns.sas’ ’ProcDsn.sas’;
%ProcDsns(Libname=SAShelp,MacroName = ProcDsn);

*Libname Library ’<directory-specification>’;
*ProcDsns(Libname=Library);
```

Conclusion

Any data set is a candidate for use by list processing. To produce dynamic programs follow these simple steps:

- identify the data set (table)
- examine its data structure
- identify the variables (columns) that contain parameter values
- develop a program with example code
- use proc sql to write that code as text or macro call, substituting variable names for values
- sit back and watch the log zoom by

Summary : be aware of limitations:

- maximum length of 200 for cat functions
- maximum length of one macro variable: $2^{16} - 2$ = 65534

Recommendation : Use select into macro arrays, they are the easiest to debug, test and understand.

Acknowledgements

Ian Whitlock and Sig Hermansen have piqued my interest in SQL over the years with their contributions to SAS-L. Toby Dunn provided commentary on numerous examples. Jack Hamilton provided the file-handling program.

Suggested Readings

bookshelf : Carpenter [6, saspress.59224], Celko [9, Celko.2000], Celko [10, Celko.2005] Schreier [35, saspress.60500]

basics : Dickstein and Pass [13, sugi29.269], Hu [27, sugi29.042], Lund [31, sugi30.257], Ronk [34, sugi29.268], Wells [39, sugi26.105], Winn, Jr. [43, sugi22.067]

intermediate : and advanced concepts: Barber [4, sugi22.198], Hamilton [25, sugi31.046] Hermansen [26, sugi22.035] Lafler [28, sugi28.019], Loren and Nelson [30, sugi23.031], Winn, Jr. [44, sugi23.035]

list processing : Abolafia [1, sugi30.031], Andrews [3, sugi31.039], Beakley and McCoy [5, sugi29.078], ch. 9, dynamic programming, Carpenter [6, saspress.59224], Carpenter [7, sugi30.028], Fehd [20, sgf2007.028], Fehd and Carpenter [22, sgf2007.113], Fehd [21, sgf2008.003], Pollack [33, sugi30.057], Varney [38, sugi31.045]

macros and SQL : Adams [2, sugi28.087], Chiu and Heaton [11, sugi28.097], Delaney and Carpenter [12, sugi29.128], DiIorio and Abolafia [15, sugi29.237], DiIorio [14, sugi30.268], Droogendyk and Fecht [16, sugi31.251], Fehd [18, sugi29.070], First and Ronk [24, sugi31.107], Stroupe [36, sugi28.056], Sun and Wong [37, sugi30.040], Whitlock [41, sugi29.244], Whitlock [42, sugi30.252]

Bibliography

[1] Jeff Abolafia. What would I do without proc SQL and the macro language? In *Proceedings of the 30th SAS User Group International Conference*, 2005. URL `http://www2.sas.com/proceedings/sugi30/031-30.pdf`. Coders' Corner, 5 pp.; creating and using arrays of macro variables, creating partial sas statements using catx function.

[2] John H. Adams. The power of recursive sas macros — how can a simple macro do so much? In *Proceedings of the 28th SAS User Group International Conference*, 2003. URL `http://www2.sas.com/proceedings/sugi28/087-28.pdf`. Coders' Corner, 5 pp.; reading directories and subdirectories using recursive macro calls.

[3] Rick Andrews. Sas macro dynamics — from simple basics to powerful invocations. In *Proceedings of the 31st SAS User Group International Conference*, 2006. URL `http://www2.sas.com/proceedings/sugi28/087-28.pdf`. Coders' Corner, 6 pp.; creating macro variables, comparison of SQL and call symput, making and using lists of values in macro variables, writing to file then including, using call execute for list processing, creating and using array of macro variables, bibliography.

[4] Brenda M. Barber. Overcoming kainophobia: Replacing complex merges with proc sql. In *Proceedings of the 22nd SAS User Group International Conference*, 1997. URL `http://www2.sas.com/proceedings/sugi22/POSTERS/PAPER198.PDF`. Poster, 6 pp.; merging using SQL joins.

[5] Steven Beakley and Suzanne McCoy. Dynamic sas programming techniques, or how not to create job security. In *Proceedings of the 29th SAS User Group International Conference*, 2004. URL `http://www2.sas.com/proceedings/sugi29/078-29.pdf`. Coders' Corner, 10 pp.; using SQL to make lists of values, example programs.

[6] Arthur L. Carpenter. *Carpenter's Complete Guide to the SAS Macro Language, Second Edition*. Cary, NC: SAS Institute Inc., 2004. URL `http://www.sas.com/apps/pubscat/bookdetails.jsp?pc=59224`. 13 chap., 475 pp., appendices: 5, glossary: 3 pp., bibliography: 19 pp., index: 13 pp.

[7] Arthur L. Carpenter. Storing and using a list of values in a macro variable. In *Proceedings of the 30th SAS User Group International Conference*, 2005. URL `http://www2.sas.com/proceedings/sugi30/028-30.pdf`. Coders' Corner, 6 pp.; making macro arrays using symputx or sql, iterating macro arrays, using scl functions, bibliography.

[8] Arthur L. Carpenter. The path, the whole path, and nothing but the path, so help me windows. In *Proceedings of the SAS Global Forum*, 2008. URL `http://www2.sas.com/proceedings/forum2008/023-2008.pdf`. Applications Development, 8 pp.; pathname function, dictionary.libnames, how to get a UNC.

[9] Joe Celko. *Joe Celko's SQL for Smarties: Advanced SQL Programming, Second Edition*. Morgan-Kaufmann, San Francisco, CA, USA, 2000. URL `http://www.celko.com`.

[10] Joe Celko. *Joe Celko's SQL Programming Style*. Elsevier, San Francisco, CA, USA, 2005. URL `http://www.elsevier.com`. 10 chap., 216 pp., bibliography: 3 pp., index: 16 pp.

[11] Grace Chiu and Edward Heaton. An efficient approach to combine SAS data sets with voluminous variables that need name and other changes. In *Proceedings of the 28th SAS User Group International Conference*, 2002. URL `http://www2.sas.com/proceedings/sugi28/097-28.pdf`. Coders' Corner, 5 pp.; renaming variables, proc contents and proc sql.

[12] Kevin P. Delaney and Arthur L. Carpenter. SAS macro: Symbols of frustration? %Let us help! a guide to debugging macros. In *Proceedings of the 29th SAS User Group International Conference*, 2002. URL `http://www2.sas.com/proceedings/sugi29/128-29.pdf`. Hands-on Workshops, 20

pp.; using options to aid debugging, scan and qscan functions, problems when using single or double quotes, SQL selecting values into macro variable.

[13] Craig Dickstein and Ray Pass. Data step vs. proc sql: What's a neophyte to do? In *Proceedings of the 26th SAS User Group International Conference*, 2001. URL `http://www2.sas.com/proceedings/sugi29/269-29.pdf`. Beginning Tutorials, 9 pp.; comparison of merge and joins, using proc SQL for recoding, SQL functions.

[14] Frank DiIorio. Rules for tools — the SAS utility primer. In *Proceedings of the 30th SAS User Group International Conference*, 2005. URL `http://www2.sas.com/proceedings/sugi30/268-30.pdf`. Tutorials, 19 pp.; reusing macros and programs, sql dictionary tables.

[15] Frank DiIorio and Jeff Abolafia. Dictionary tables and views: Essential tools for serious applications. In *Proceedings of the 29th SAS User Group International Conference*, 2002. URL `http://www2.sas.com/proceedings/sugi29/237-29.pdf`. Tutorials, 19 pp.; comparison of data structure of dictionary tables and sashelp views, review of differences between v6, v8, and v9, 12 examples.

[16] Harry Droogendyk and Marje Fecht. Demystifying the SAS macro facility — by example. In *Proceedings of the 31st SAS User Group International Conference*, 2006. URL `http://www2.sas.com/proceedings/sugi31/251-31.pdf`. tutorials, 15 pp.; list processing, macro functions: symdel, symexist, syslput, sysrput; proc sql, scl functions: open, attrn, close.

[17] Peter Eberhardt and Ilene Brill. How do i look it up if I cannot spell it: An introduction to SAS(R) dictionary tables. In *Proceedings of the SAS Global Forum*, 2007. URL `http://www2.sas.com/proceedings/forum2007/235-2007.pdf`. Tutorials, 14 pp.; reviews many sql dictionaries, example list processing programs, bibliography.

[18] Ronald Fehd. Array: Construction and usage of arrays of macro variables. In *Proceedings of the 29th SAS User Group International Conference*, 2002. URL `http://www2.sas.com/proceedings/sugi29/070-29.pdf`. Coders' Corner, 6 pp.; using proc sql, bibliography.

[19] Ronald J. Fehd. Writing testing-aware programs that self-report when testing options are true. In *Proceedings of the NorthEast SAS User Group Annual Conference*, 2007. URL `http://www.nesug.org/proceedings/nesug07/cc/cc12.pdf`. Coders' Corner, 20 pp.; use of options echoauto, mprint, source2, and verbose during unit and integration testing, bibliography.

[20] Ronald J. Fehd. Journeymen's tools: Data review macro FreqAll – using Proc SQL list processing with Dictionary.Columns to eliminate macro do loops. In *Proceedings of the SAS Global Forum*, 2007. URL `http://www2.sas.com/proceedings/forum2007/028-2007.pdf`. Coder's Corner, 10 pp.; attributes, dictionary.columns, metadata, proc append, proc freq, proc sql, program header; bibliography.

[21] Ronald J. Fehd. SmryEachVar: a data-review routine for all data sets in a libref. In *Proceedings of the SAS Global Forum*, 2008. URL `http://www2.sas.com/proceedings/forum2008/003-2008.pdf`. Applications Development, 24 pp.; successor to FreqAll, list processing using parameterized includes, utilities which write: 1. attribute statements 2. format statements, used to provide missing variable attributes, bibliography.

[22] Ronald J. Fehd and Art Carpenter. List processing basics: Creating and using lists of macro variables. In *Proceedings of the SAS Global Forum*, 2007. URL `http://www2.sas.com/proceedings/forum2007/113-2007.pdf`. Hands On Workshop, 20 pp.; comparison of methods: making and iterating macro arrays, scanning macro variable, writing calls to macro variable, write to file then include, call execute; 11 examples, bibliography.

[23] Ying Feng. The SQL procedure: When and how to use it. In *Proceedings of the 31st SAS User Group International Conference*, 2006. URL `http://www2.sas.com/proceedings/sugi31/044-31.pdf`. Coders' Corner, 7 pp.; using SQL to make and iterate a macro array, bibliography.

[24] Steven First and Katie Ronk. Intermediate and advanced SAS macros. In *Proceedings of the 31st SAS User Group International Conference*, 2006. URL `http://www2.sas.com/proceedings/sugi31/107-31.pdf`. Hands-on Workshops, 16 pp.; call routines: execute, symdel, symput, symputn, symputx; functions: resolve, symexist, symget, symgetn, symglobal, symlocal, sysexec, sysget; macro arrays, proc sql, sas/connect: syslput, sysrput.

[25] Jack Hamilton. Digits and dates: The SQL procedure goes loopy. In *Proceedings of the 31st SAS User Group International Conference*, 2006. URL `http://www2.sas.com/proceedings/sugi26/061-26.pdf`. Coders' Corner, 11 pp.; simulating looping thru dates, using scl functions in sql, using date functions in sql.

[26] Sigurd Hermansen. Ten good reasons to learn SAS software's SQL procedure. In *Proceedings of the 22nd SAS User Group International Conference*, 1997. URL `http://www2.sas.com/proceedings/sugi26/061-26.pdf`. Advanced Tutorials, 5 pp.; comparison of SQL to data step.

[27] Weiming Hu. Top ten reasons to use proc sql. In *Proceedings of the 29th SAS User Group International Conference*, 2004. URL `http://www2.sas.com/proceedings/sugi29/042-29.pdf`. Coders' Corner, 6 pp.; joining, selecting values into list, text wrapping, summary functions, matching, inserting, using coalesce function, summarizing, fuzzy merges, examples, bibliography.

[28] Kirk Paul Lafler. Undocumented and hard-to-find SQL features. In *Proceedings of the 28th SAS User Group International Conference*, 2003. URL `http://www2.sas.com/proceedings/sugi28/019-28.pdf`. Advanced Tutorials, 6 pp.; case logic, coalesce function, SQL statement options _method and _tree, SQL dictionary tables, automatic macro variables, performance issues, examples.

[29] Stuart Long and Ed Heaton. Using the SAS(R) data step and proc SQL to create macro arrays. In *Proceedings of the SAS Global Forum*, 2008. URL `http://www2.sas.com/proceedings/forum2008/105-2008.pdf`. Aplications Development, 11 pp.; comparison of macro arrays and delimited list, symputx.

[30] Judy Loren and Gregory S. Barnes Nelson. Sql step by step: An advanced tutorial for business users. In *Proceedings of the 23nd SAS User Group International Conference*, 1998. URL `http://www2.sas.com/proceedings/sugi23/Advtutor/p31.pdf`. Advanced Tutorial, 10 pp.; examples, bibliography.

[31] Pete Lund. An introduction to SQL in SAS. In *Proceedings of the 30th SAS User Group International Conference*, 2005. URL `http://www2.sas.com/proceedings/sugi30/257-30.pdf`. Tutorials, 22 pp.; basic syntax, where clause operators: null, missing, between, contains, like, sounds like; aggregating functions.

[32] William C. Murphy. Changing data set variables into macro variables. In *Proceedings of the SAS Global Forum*, 2007. URL `http://www2.sas.com/proceedings/forum2007/050-2007.pdf`. Coders Corner, 4 pp.; code reuse, macro arrays and do loops, call set.

[33] Stuart Pollack. Techniques for effectively selecting groups of variables. In *Proceedings of the 30th SAS User Group International Conference*, 2005. URL `http://www2.sas.com/proceedings/sugi30/057-30.pdf`. Coders' Corner, 4 pp.;.

[34] Katie Minten Ronk. Introduction to PROC SQL. In *Proceedings of the 29th SAS User Group International Conference*, 2004. URL `http://www2.sas.com/proceedings/sugi29/268-29.pdf`. Tutorials, 13 pp.; examples.

[35] Howard Schreier. *PROC SQL by Example: Using SQL within SAS*. Cary, NC: SAS Institute Inc., 2008. URL `http://www.sas.com/apps/pubscat/bookdetails.jsp?pc=60500`. 14 chap., 275 pp.

[36] Jane Stroupe. Nine steps to get started using SAS macros. In *Proceedings of the 28th SAS User Group International Conference*, 2003. URL `http://www2.sas.com/proceedings/sugi28/`

`056-28.pdf`. Beginning Tutorials, 5 pp.; creating macro variables using proc sql, developing macros from programs.

[37] Helen Sun and Cindy Wong. A macro for importing multiple excel worksheets into SAS data sets. In *Proceedings of the 30th SAS User Group International Conference*, 2005. URL `http://www2.sas.com/proceedings/sugi30/040-30.pdf`. Coders' Corner, 9 pp.; application: import .xls, using scan function in macro do loops, bibliography.

[38] Brian Varney. Using metadata and project data for data-driven programming. In *Proceedings of the 31st SAS User Group International Conference*, 2006. URL `http://www2.sas.com/proceedings/sugi31/045-31.pdf`. Coders' Corner, 10 pp.; assertions, call execute, call symput, macro arrays, scl functions, SQL dictionary tables, stored processes, subsetting, write to file and %include.

[39] Clayton Wells. Tips for using proc SQL to extract information from medical claims. In *Proceedings of the 26th SAS User Group International Conference*, 2001. URL `http://www2.sas.com/proceedings/sugi26/p105-26.pdf`. Coder's Corner, 4 pp.; comparison of proc transpose to sql, practical usage, SQL functions.

[40] Ian Whitlock. Proc sql: Is it a required tool for good sas programming? In *Proceedings of the 22nd SAS User Group International Conference*, 1997. URL `http://www2.sas.com/proceedings/sugi26/p060-26.pdf`. Beginning Tutorials, 6 pp.; using SQL to create macro arrays, selecting distinct text into macro variable.

[41] Ian Whitlock. A second look at SAS macro design issues. In *Proceedings of the 29th SAS User Group International Conference*, 2004. URL `http://www2.sas.com/proceedings/sugi26/244-29.pdf`. Tutorials, 18 pp.; examples, list processing, macro arrays, macro parameter naming conventions, sashelp views, SQL dictionary tables.

[42] Ian Whitlock. Macro bugs: How to create, avoid, and destroy them. In *Proceedings of the 30th SAS User Group International Conference*, 2005. URL `http://www2.sas.com/proceedings/sugi30/252-30.pdf`. Tutorials, 20 pp.; call execute, call symput, debugging, design, how macro facility works, options used when testing: mfile mlogic mprint, readability, single and double quoting, scope of macro variables, testing, understanding difference between macro and sas bugs writing messages to log with %put.

[43] Thomas J. Winn, Jr. Introduction to using proc sql. In *Proceedings of the 22nd SAS User Group International Conference*, 1997. URL `http://www2.sas.com/proceedings/sugi26/p105-26.pdf`. Beginning Tutorial, 7 pp.; create tables and views, examples, joins, operators.

[44] Thomas J. Winn, Jr. Intermediate proc sql. In *Proceedings of the 23nd SAS User Group International Conference*, 1998. URL `http://www2.sas.com/proceedings/sugi23/Advtutor/p35.pdf`. Advanced Tutorial, 7 pp.; examples, summary functions, views.

Get Programs

To get the code examples in this paper search http://www.sascommunity.org for the HOW SQL for List Processing zip.

Author

Author: Ronald J. Fehd mailto:RJF2@cdc.gov
Centers for Disease Control
4770 Buford Hwy NE
Atlanta GA 30341-3724

	about the author:	
education:	B.S. Computer Science, U/Hawaii,	1986
	SAS User Group conference attendee since	1989
	SAS-L reader	since 1994
experience:	programmer: 20+ years	
	data manager at CDC, using SAS: 17+ years	
	author: 30+ SUG papers	
SAS-L:	author: 5,000+ messages to SAS-L since	1997
	Most Valuable SAS-L contributor:	2001, 2003

Document Production: This paper was typeset in LaTeX. For further information about using LaTeX to write your SUG paper, consult the SAS-L archives:

```
http://www.listserv.uga.edu/cgi-bin/wa?S1=sas-l
Search for                    :
The subject is or contains: LaTeX
The author's address      : RJF2
Since                         : 01 June 2003
```

Index