Paper T09

How To Use Proc SQL select into for List Processing

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

ABSTRACT

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.
Expected audience is intermediate to advanced users, and macro programmers.

Keywords: dynamic programming, list processing, macro, SQL.

INTRODUCTION

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*?

In this paper I review the theory of programming, how to process every item in a list using the utter simplicity of Proc SQL select into :list processing with SQL's dictionary tables.

PREREQUISITES

Students are expected to have the following minimum background:

- programming experience: 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

- 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

Contents

PROGRAMMING THEORY

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

program	data structure algorithm	(also: metadata)
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: process output:	libref, data, variable perform actions libref, data, to log or list

SQL SYNTAX

There are five SQL statements:

1. proc
2. create, closure (;): line 6
3. describe
4. select, closure (;): line 22
5. quit

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

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

ProcSQL-0-syntax.sas

```
1  PROC SQL;
2  PROC SQL noprint;
3          create   table table-name as
4                   query-expression
5                    <order by order-by-item
6                     <, ...   order-by-item>>;
7          describe table table-name <, ... table-name>;
8          select   <distinct> object-item
9                   <function>(object-item)
10                  <, ...     object-item>
11         into    :macro-variable <separated by ' '>
12                 :macro-variable-A, :macro-variable-B
13                 :macro-variable1 - :macro-variable9999
14         from     Libref.Data
15         where    ColumnChar  eq 'value'
16                  ColumnNum   eq  <num-value>
17           and    ColumnChar2 eq  'value2'
18         group by group-by-item
19                   <, ... group-by-item>>
20         having   sql-expression
21         order by order-by-item
22                   <, ... order-by-item>>;
23       ; quit;
```

SQL BASIC PROCESSES

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

1. list data structure
2. list data
3. list only subset
4. create data
5. unique

LISTING DATA STRUCTURE

Proc SQL works like Proc Contents. Instead of the `data =` option, SQL has the `describe table` statement.

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

```
1  PROC Contents data =     SAShelp.Class;
2
3  PROC SQL; describe table SAShelp.Class;
4            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

```
1  PROC Print data = SAShelp.Class;
2            var    _all_;
3
4  PROC SQL;  select *           /* _all_ */
5             from   SAShelp.Class;
6             quit;
```

LISTING SUBSETS

The `where` statement is available in all procedures. I illustrate it here as a data step option. The SQL `select ... from` statement has a `where` clause.

ProcSQL-1-select-subset.sas

```
1  PROC Print data  =  SAShelp.Class
2            (where = (      Sex eq 'F'
3                      and   Age ge 14));
4             var      Name  Age;
5
6  PROC SQL;  select   Name, Age
7             from     SAShelp.Class
8             where           Sex eq 'F'
9                      and   Age ge 14;
10            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

```
1  DATA    Work.Class;
2  set  SAShelp.Class;
3
4  PROC SQL; create table     Work.Class as
5                  select *
6                  from   SAShelp.Class;
7            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

```
1  PROC Sort  data  = SAShelp.Class nodupkey
2             out   =       UniqueAge
3            (keep  =             Age);
4             by                  Age ;
5
6  PROC SQL;  create  table UniqueAge  as
7             select  distinct    Age
8             from    SAShelp.Class
9             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: an introduction to SQL: Hermansen [21, sugi22.035] Ronk [27, sugi29.268] Wells [31, sugi26.105] Winn, Jr. [34, sugi22.067]

LIST PROCESSING: WRITING CONSTANT TEXT

The 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-select-constant-text.sas

```
1  %Let Rows = rows;
2  PROC SQL; select MemName, 'has', Nobs, "&Rows."
3            from   Dictionary.Tables
4            where  LibName eq "SASHELP"
5              and  MemName eq "CLASS"
6              and  MemType eq "DATA";
7            quit;
8  run;
```

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

ProcSQL-select-constant-text.lst

```
6                                              Number of
7                                               Physical
8  Member Name                                Observations
9  ------------------------------------------------------------
10 CLASS                                 has            17  rows
```

The above example selects four objects. Now let us concatenate text and variable value, using double bang (!!, two exclamation points) as the join operator and put that text into a macro variable.
Note: line 9; The `like` operator chooses only names beginning with 'V'.
Note: line 12; the statements in the macro variable are procedure statements, therefore they must be executed *after* the `quit;` statement in line 11.

ProcSQL-select-text-into-List.sas

```
1  %Let List = *missing-;%* initialize for no rows selected;
2  *PROC SQL  noprint;
3  PROC SQL; select 'Proc Contents data = SAShelp.'
4                 !! trim(MemName)
5                 !! ';'
6            into  :List   separated by ' '
7            from   Dictionary.Tables
8            where  LibName eq   "SASHELP"
9              and  MemName like "V%"
10             and  MemType eq   "VIEW";
11           quit;
12 &List.;%* execute statements in mvar List;
13 run;
```

Statements in the macro variable List.

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

```
8  Proc Contents data = SAShelp.VALLOPT;
9  Proc Contents data = SAShelp.VCATALG;
```

Dilorio and Abolafia [14, sugi29.237] discuss the SAShelp views associated with SQL dictionaries.

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

Output from statements in the macro variable List.

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

```
43 The CONTENTS Procedure
44
45 Data Set Name        SASHELP.VALLOPT          Observations          .
```

LIST PROCESSING SUMMARY

The 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:
   (a) concatenation of text and values into macro variable
      i. identify subset, if any; note: values in ALL CAPS?
      ii. clarify the text surrounding the variable value(s): prefix, infix(es), suffix
   (b) remember closure or delimiter:
   clause(s): comma, space, 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 I use this list processing check list to examine several of the more commonly used dictionaries.

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

DICTIONARY DICTIONARIES

Now let's look at everything you ever wanted to know about all those SQL dictionaries.

Statements to list the data structure:
Note: V9 only.

ProcSQL-describe-table-D-Dictionaries.sas

```
1  PROC SQL; describe table Dictionary.Dictionaries;
2            quit;
3  run;
```

Log with the data structure:

ProcSQL-describe-table-D-Dictionaries.log

```
26 create table DICTIONARY.DICTIONARIES
27   (
28    memname char(32) label='Member Name',
29    memlabel char(256) label='Dataset Label',
30    name char(32) label='Column Name',
31    type char(4) label='Column Type',
```

Dictionary.Dictionaries (new in V9) is unique on MemName + Name (Column Name). We need a data set (list) unique on MemName in order to write these statements:

ProcSQL-list-describe-table-dictionaries.lst

```
8  describe table Dictionary.CATALOGS;
9  describe table Dictionary.CHECK_CONSTRAINTS;
10 describe table Dictionary.COLUMNS;
```

This program reads the SQL table Dictionary.Dictionaries, makes a data set with the names of all the SQL dictionaries, writes `describe table` statements for each, then executes those statements on line 10. The log contains the data structure of each SQL dictionary.

ProcSQL-list-describe-table-dictionaries.sas

```
1  %*note: need v9 for D.Dictionaries;
2  Proc SQL; create table  List as
3                  select distinct MemName
4                  from   Dictionary.Dictionaries;
5            select 'describe table Dictionary.'
6                !! trim(MemName)
7                !! ';'
8            into   :List separated by ' '
9            from    List ;
10                  &List.;
11           quit;
12 run;
```

Note line 10: the statements in the macro variable are SQL describe statements, therefore they must be executed *before* the `quit;` statement.
Compare with program `select-text-into-List` above, which lists SAShelp views.
This program illustrates a two-statement solution to this problem. See program `ProcSQL-list-groups` below for a single-statement solution.

DICTIONARY MACROS

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

Put-User.log

```
22 1          %Let A = 1;
23 2          %Let b = 22;
24 3          %Let z = end of list;
25 4          %Put _user_;
26 GLOBAL Z end of list
27 GLOBAL A 1
28 GLOBAL B 22
```

Here is the data structure of Dictionary.Macros:

ProcSQL-describe-table-D-Macros.log

```
26 create table DICTIONARY.MACROS
27   (
28    scope char(32) label='Macro Scope',
29    name char(32) label='Macro Variable Name',
30    offset num label='Offset into Macro Variable',
31    value char(200) label='Macro Variable Value'
```

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.;
```

This program writes the put statements into the macro variable named list and executes them, line 13.

ProcSQL-order-D-Macros-log.sas

```
1  %Let Z = the last one;
2  %Let A = 1st item;
3  %Let M = middle;
4  Proc SQL; select   '%Put ' !! trim(Name)
5                !!   ': '    !! trim(Value)
6                !!   ';'
7            into    :List   separated by ' '
8            from     Dictionary.Macros
9            where    Scope eq 'GLOBAL'
10             and    not(Name like 'SQL%')
11           order by Name;
12           quit;
13 &List.;
14 run;
```

Note: line 13; the statements in the macro variable are SAS statements, therefore they must be executed *after* the `quit;` statement.

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

ProcSQL-order-D-Macros-log.log

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

The statements in macro variable List:

ProcSQL-order-D-Macros-log.lst

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

DICTIONARY OPTIONS

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

What is the data structure?

ProcSQL-describe-table-D-Options.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.

Use list processing technique to write text:

ProcSQL-list-Options-define-value.sas

```
options   linesize = max;
PROC SQL; select 'Proc Options define value option = '
               !! trim(OptName)
               !! ';'
          into  :List   separated by ' '
          from   Dictionary.Options;
          quit;
&List.;
run;
```

Here's the list of values and definitions of all options. The log is over 4600 lines long.

ProcSQL-list-Options-define-value.lst snip 1

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

ProcSQL-list-Options-define-value.lst snip 2

```
Proc Options define value option = MEMBLKSZ;
Proc Options define value option = MEMCACHE;
```

ProcSQL-list-Options-define-value.log

```
Option Value Information For SAS Option APPLETLOC
    Option Value: C:\Program Files\SAS Institute\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
          Maximum Number of Characters: 256
```

In program `list-describe-table-dictionaries` 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.

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.
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`.

ProcSQL-list-groups.sas

```
%Let List = *no rows selected;
Proc SQL; select  distinct Group as Item,
                 '%Put Group: '
              !!  calculated Item
              !! ';Proc Options group = '
              !!  calculated Item
              !! ';run;'
          into   :Item, :List separated by ' '
          from    Dictionary.Options;
          quit;
&List.;
run;
```

Note that the column Item retains the label of Group.

ProcSQL-list-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.

ProcSQL-list-groups.log

```
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
```

LIST PROCESSING: WRITING MACRO CALLS

In the following secions I provide program templates for processing the three most commonly used lists: columns (variables), tables (data set names), and files.

DICTIONARY.COLUMNS

The task is to process every column in a data set.

ProcSQL-describe-table-D-Columns.log snip 1

```
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',
```

ProcSQL-describe-table-D-Columns.log snip 2

```
   label char(256) label='Column Label',
   format char(49) label='Column Format',
```

The primary parameter is the libref, line 1; see references in lines 22 and 27.
The secondary parameter is the table name in the libref, lines 3–6, referred to in lines 23 and 28.

A tertiary parameter, provided for clarity, is the member type, in (data, view), lines 8–9, see line 24.
Do you want to review the statements generated? or is this program running in production? Choose to enable either of lines 11 or 12.

The macro `ProcVar` is not yet defined at this time; this does not matter as we are writing the statements but have not yet executed them, line 44.

Both global macro variables `LibName` and `MemName` are indirectly referenced via macro parameters `libref` and `data` as their defaults.

This is a simple processing example.
Your own code goes here.

Last, but not least, execute.

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

```
1  %Let LibName  = SAShelp;
2
3  %Let MemName  = Class;
4  *Let MemName  = PrdSal2;
5  *Let MemName  = PrdSal3;
6  *Let MemName  = PrdSale;
7
8  %Let MemType  = data;
9  *Let MemType  = view;
10
11 %Let SQLprint =   print;%*testing;
12 *Let SQLprint = noprint;
13
14 PROC SQL &SQLprint.;
15           select '%ProcVar(name=' !! trim(Name)
16               !! ',type  ='       !! trim(Type)
17               !! ',length='       !!  put(Length,5.)
18               !! ',format='       !! trim(Format)
19               !! ',label ='       !! trim(Label)    !! ')'
20           into  :List   separated by ' '
21           from   Dictionary.Columns
22           where  LibName eq "%upcase(&LibName.)"
23             and  MemName eq "%upcase(&MemName.)"
24             and  MemType eq "%upcase(&MemType.)";
25           quit;
26
27 %Macro ProcVar(Libref = &LibName. /*global*/
28               ,Data   = &MemName. /*global*/
29               ,Name   =
30               ,Type   =
31               ,Length =
32               ,Format =
33               ,Label  = );
34 %If       &Type. eq char %then                    %do;
35       Proc Freq       data   = &Libref..&Data.;
36                       tables   &Name.;          %end;
37 %Else %if &Type. eq num  %then                    %do;
38       Proc Univariate data =   &Libref..&Data.;
39                       var      &Name.;          %end;
40 title2 "&Libref..&Data..&Name. type: &Type. "
41        "format: &Format. label: &Label.";
42 run;%Mend;
43
44 &List.;%*execute;
```

DICTIONARY.TABLES

In this example I show a production example of two programs where the first program containing the processing macro ProcDsn, contains a call (using the %include statement) of the second list processing program. This 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?

ProcSQL-describe-table-D-Tables.log snip 1

```
27 create table DICTIONARY.TABLES
28   (
29    libname char(8) label='Library Name',
30    memname char(32) label='Member Name',
31    memtype char(8) label='Member Type',
32    dbms_memtype char(32) label='DBMS Member Type',
```

ProcSQL-describe-table-D-Tables.log snip 2

```
37    nobs num label='Number of Physical Observations',
38    obslen num label='Observation Length',
39    nvar num label='Number of Variables',
```

The primary parameter is the libref, line 1.

Do you want to subset the processing by choosing the member type before writing the statement? Enable either line 6 or 7.

Do you want to review the statements generated? or is this program running in production? Choose to enable any or all of the testing statements in lines 11–13.

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

This is a simple processing example. Your own code goes here.

Call the list processing program that will read all tables in the library defined here and execute the macro ProcDsn defined in this program. Note that you could have other programs defining the macro ProcDsn and relying on the subroutine `select-from-D-Tables`.

Compare with program `select-from-D-Columns` above.

ProcDsn.sas

```
1  %Let LibName = SAShelp;%*testing;
2
3  %*Libname       MyLib '???';
4  %*Let LibName = MyLib;%*production;
5
6  %Let Op_MemType = eq 'data';
7  %*Let Op_MemType = in('catalog' 'data' 'view');
8
9  %Let SQLprint = noprint;%*production;
10
11 %Let SQLprint =   print;%*testing;
12 %*options source2;%*testing: echo include?;
13 %*Put Op_MemType<%upcase(&Op_MemType.)>;%*testing;
14
15 %Macro ProcDsn(Libref = &LibName. /*global*/
16               ,Data   =
17               ,Type   =
18               ,Label  =
19               ,Nobs   =
20               ,Nvar   =
21               ,TitleN = 2);
22
23 %If       &Type. eq CATALOG %then %do;
24       %Put &Libref..&Data. type: &Type.;
25       Proc Catalog catalog = &Libref..&Data.;
26                    contents;
27                    quit;
28       %end;
29 %Else %if &Type. eq DATA %then %do;
30       %Put &Libref..&Data. nobs: &Nobs. nvar: &Nvar.;
31       Proc SQL; describe table &Libref..&Data.;
32       %end;
33 %Else %if &Type. eq VIEW %then %do;
34       Proc Contents data = &Libref..&Data.;
35                     title2&TitleN.
36                     "&Libref..&Data. type: &Type. "
37                     "nobs: &Nobs. nvar: &Nvar.";
38       %end;
39 %Else %Put &Libref..&Data. type unknown: &type.;
40 run; %Mend;
41
42 %Include 'ProcSQL-select-into-list-from-D-Tables.sas';
```

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

```
1  %*note: parameters assigned by calling program;
2
3  %*Let LibName    = SAShelp;
4  %*Let Op_MemType = eq 'data';
5  %*Let SQLprint   = print;%*testing;
6  %*Inc 'MacroProcessDsn'; %*testing;
7
8  %Let List = *empty: no rows selected;
9
10 PROC SQL &SQLprint.;
11          select '%ProcDsn(data='  !! trim(MemName)
12              !! ',type ='         !! trim(MemType)
13              !! ',label='         !! trim(MemLabel)
14              !! ',nobs =' !! compress(put(Nobs,32.))
15              !! ',nvar =' !! compress(put(Nvar,32.))
16              !! ')'
17          into  :List       separated by ' '
18          from   Dictionary.Tables
19          where  LibName eq "%upcase(&LibName.)"
20            and  MemType     %upcase(&Op_MemType.);
21          quit;
22 &List.;%*execute macro calls;
23 %symdel Libname List Op_MemType SQLprint;
```

FILENAMES

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. In this example, I have polished a program written by Hamilton [20, sugi31.046]

The primary parameter is the folder name.

ProcSQL-select-into-list-from-filenames.sas snip 1

```
36 %Let Folder   = c:\;
37 *Let Folder   = .;%*here;
38 *Let Folder   = %sysget(sasroot);%*directory-spec;
39 *Let Folder   = %sysget(sasroot)\core\sasexe;%*922 files;
40
41 %Let SQLprint = noprint;%*production;
42 %Let SQLprint =   print;%*testing;
43
44 Data Digits; length Digit 4; do Digit = 0 to 9; output;
45                                  end;  stop; run;
46
47 %* see: from (&&E&Evalue.);
48 %Let E1 = select ones.digit  as FileNmbr from digits
49                              as ones;
50 %Let E2 = select ones.digit  +  10 * tens.digit
51                              as FileNmbr
52                  from digits as ones, digits as tens;
```

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

Omitted lines 53–64 contain allocations of macro variables E3 and E4 which are necessary for processing lists of 999 and 9999 files.

Two select statments are required: the first to assign a fileref, open the folder and read the number of files present.

ProcSQL-select-into-list-from-filenames.sas snip 2

```
65 Proc SQL &SQLprint.;
66          select filename('DirSpec', "&Folder."),
67                    dopen('DirSpec') as Dir_id,
68                     dnum(calculated    Dir_id)
69          into  :FileName_Rc, :Dir_id, :NmbrFiles
70          from   Digits   where Digit = 0;
71                 %Let NmbrFiles = &NmbrFiles.;%*trim;
72                 %Let Evalue    = %length(&NmbrFiles.);
73          select '%ProcFile(filename='
74              !! dread(&Dir_id., FileNmbr) !! ')'
75          into  :List   separated by ' '
76          from  (&&E&Evalue.)
77          where  FileNmbr between 1 and &NmbrFiles;
78          quit; %Let FileName_Rc = dclose(&Dir_id.);
79
80 %Macro ProcFile( directory = &Folder. /*global*/
81                , filename  = );
82 %if %index(&Filename.,.) %then
83       %put filename.ext:<&FileName.>;
84 %else %put other<&FileName.>;
85 run; %mend;
86
87 &List.;
88 %symdel E1 E2 E3 E4 Evalue
89         Folder List SQLprint
90         FileName_Rc Dir_id NmbrFiles;
```

The second statement writes the macro calls.

Line 76 generates the list of file numbers.

Your filename-processing statements go here.

After executing the macro calls then do housecleaning: symbol-delete all the macro variables used.

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

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.

SUGGESTED READINGS

bookshelf Carpenter [6, saspress.59224], Celko [8, Celko.2000], and Celko [9, Celko.2005]

basics Dickstein and Pass [12, sugi29.269], Hu [22, sugi29.042], Lund [25, sugi30.257], Ronk [27, sugi29.268], Wells [31, sugi26.105], Winn, Jr. [34, sugi22.067]

intermediate and advanced concepts: Barber [4, sugi22.198], Hamilton [20, sugi31.046] Hermansen [21, sugi22.035] Lafler [23, sugi28.019], Loren and Nelson [24, sugi23.031], Winn, Jr. [35, 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 [17, sgf2007.028], Fehd and Carpenter [18, sgf2007.113], Pollack [26, sugi30.057], Varney [30, sugi31.045],

macros and SQL Adams [2, sugi28.087], Chiu and Heaton [10, sugi28.097], Delaney and Carpenter [11, sugi29.128], DiIorio and Abolafia [14, sugi29.237], DiIorio [13, sugi30.268], Droogendyk and Fecht [15, sugi31.251], Fehd [16, sugi29.070], First and Ronk [19, sugi31.107], Stroupe [28, sugi28.056], Sun and Wong [29, sugi30.040], Whitlock [32, sugi29.244], Whitlock [33, 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://support.sas.com/publishing/bbu/companion_site/59224.html`. 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] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.;.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

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

Author: Ronald 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
	SUGI attendee	since 1989
	SAS-L reader	since 1994
experience:	programmer: 20+ years	
	data manager at CDC, using SAS: 18+ years	
	author: 12+ SUG papers	
SAS-L:	author: 4,000+ 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
```