SESUG Paper 98-2017

Advanced Programming Concepts: History of the List Processing and Cardinality Ratio Memes
Ronald J. Fehd, Stakana Analytics

Abstract Description: A statement in a natural language contains three parts: subject, verb and object. A statement in a computer language contains only two parts: verb and an object; the subject, or actor, is the computer’s operating system. The predecessor of the meme list processing is the computer language LISP, in which every statement is a function call and the object is a list. The list processing paradigm of programming contains these steps: 1. identify an object, an item; 2. write a function, process, or procedure for an item; 3. prepare a list of items; 4. use a loop on the list, to process each item.

This paper reviews the author’s development of the concept of list processing and its implementation in SAS® software.

Purpose: The purpose of this exposition is to highlight the author’s papers published on these topics and to provide a critique of earlier ideas. This is accomplished by a review of the development of processes for calculating the cardinality ratios cardinality types of the variables in a data set.

Audience: programmers
Keywords: arrays of macro variables; cardinality ratio; compiled or executed contents, extended; database vocabulary; data structure and algorithm; dosUBL function; list processing; macro arrays; scl functions: open, close, attrN(nobs, nvars), fetchobs, getVarC, getVarn, varname, varType,

style: This paper uses font changes to identify SAS words in text, documentation words in italics and notable key words in SMALL CAPS; e.g.: the libname statement allocates a libref; CARDINALITY TYPES are in FEW, MANY and UNIQUE.

quote: If you think of standardization as the best that you know today, but which is to be improved tomorrow — you get somewhere. — Henry Ford

In this paper

<table>
<thead>
<tr>
<th>Introduction</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition of a list</td>
<td>2</td>
</tr>
<tr>
<td>Learning a computer language</td>
<td>4</td>
</tr>
<tr>
<td>Arrays of macro variables</td>
<td>4</td>
</tr>
<tr>
<td>Database vocabulary</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The three basic lists</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable names</td>
<td>7</td>
</tr>
<tr>
<td>Data set names: memnames</td>
<td>9</td>
</tr>
<tr>
<td>Values</td>
<td>10</td>
</tr>
</tbody>
</table>
Loops: processes for lists

Specific Routines .................................................. 12
Call execute an %include ........................................ 14
Demo macro function which reads a data set .................. 15
Call-macro .............................................................. 16
Call-text ................................................................. 17

A function to calculate cardinality ratios and types 18

N steps ................................................................. 19
Two steps with freq, nlevels and scl functions ............... 20
One step with scl functions and dosubl ....................... 23
Cardinality types of all memnames in a libref ............... 25

Summary 26

References 27

Introduction

Overview

This is the overview, which consists of a list of topics in this section.

- definition of a list
- learning a computer language
- arrays of macro variables
- database vocabulary

Definition of a list

Overview

This section contains these topics.

- list processing definitions
- contents listing
- contents program
- contents output data set
List processing definitions

- **list**: contains items
- **item**: can be an attribute, or a list
- **attribute**: description, information, number, or string
- **CARDINALITY**: of a list is the number of items in the list, same as dimension of an array, number of observations in data set
- **SAS list**: is a ***CONTROL DATA SET***, where each row contains list of values for a process, and the variable names are parameter names of that process; it is like a database ***DIMENSION TABLE***

**Notes:**
- compare to **proc format options** `cntlin`, `cntlout` (control-in and -out) and data structure provided by **proc contents**

contents listing

```sas
proc contents data = sashelp.class;

The CONTENTS Procedure

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>Member Type</th>
<th>Observations</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>SASHELP.CLASS</td>
<td>DATA</td>
<td>19</td>
<td>5</td>
</tr>
</tbody>
</table>

Alphabetic List of Variables and Attributes

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Age</td>
<td>Num</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>Height</td>
<td>Num</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>Name</td>
<td>Char</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Sex</td>
<td>Char</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Weight</td>
<td>Num</td>
<td>8</td>
</tr>
</tbody>
</table>
```

**Notes:**
- **ATTRIBUTES** of a data set include: `libref` (sashelp) is a reference to a folder; data set name (class) is the name of a file in the folder; data set label (Student Data) provides a description of the data set; dimensions of the data matrix are height (Observations 19) and width (Variables 5); list of the attributes of variables in the data set includes the **ROW IDENTIFIERS**, variable number (labeled #) and name and type in (Char,Num).

contents program

```sas
*name: make-list-names-contents-demo.sas;
%let data = sashelp.class;
PROC contents data = &data noprint
   out = list_names
      (keep = libname memname varnum nobs
       name type length format label);
run; * updates sysnobs and syslast;
PROC sql; describe table &syslast;
quit;
proc print data = &syslast;
run;
```

contents output data set

```sas
create table WORK.LIST_NAMES
   LIBNAME char(8) label='Library Name',
   MEMMNAME char(32) label='Library Member Name',
   NAME char(32) label='Variable Name',
   TYPE num label='Variable Type',
   LENGTH num label='Variable Length',
   VARNUM num label='Variable Number',
   LABEL char(256) label='Variable Label',
   FORMAT char(32) label='Variable Format',
   NOBS num label='Observations in Data Set'
```

**Notes:**
- type is numeric which differs from listing of type in (1=Num,2=Char)
<table>
<thead>
<tr>
<th>Obs</th>
<th>LIBNAME</th>
<th>MEMNAME</th>
<th>NAME</th>
<th>TYPE</th>
<th>LENGTH</th>
<th>VARNUM</th>
<th>LABEL</th>
<th>FORMAT</th>
<th>NOBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SASHELP</td>
<td>CLASS</td>
<td>Age</td>
<td>1</td>
<td>8</td>
<td>3</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SASHELP</td>
<td>CLASS</td>
<td>Height</td>
<td>1</td>
<td>8</td>
<td>4</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SASHELP</td>
<td>CLASS</td>
<td>Name</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SASHELP</td>
<td>CLASS</td>
<td>Sex</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>SASHELP</td>
<td>CLASS</td>
<td>Weight</td>
<td>1</td>
<td>8</td>
<td>5</td>
<td>19</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes: Type in in (1=Num,2=Char), see standardization on page 7

Learning a computer language

- Variables: local or global, type (boolean, char, num)
- Conditions: if then... else add code, or branch
- Loops: conditional exit, while, leave, until enumerate (start, stop, step)
- Functions: macro, method, process, procedure
- Setup for startup: configuration, autoexec, libraries
- Compile or execute? data structure or algorithm?

Notes: Q: What is the difference between process and procedure?

**Process**: is a set of numbered steps which are always performed; testing is simple because output is consistent

**Procedure**: is a set of choices which may add extra statements — e.g., options — or branch:
- if type eq char then ... else if type eq num then ...;
- testing is complex due to multiple paths through the code

Citations: learning a computer language Statz [49], Heilmann and W3C [45]
- Conditions: using logical operators and, or, not [42]
- Sysfunc and ifc: [28] writing testing-aware programs: [22]
- Loops [23]
- Macro design [36]
- Setup for startup: autoexec companion, [37], and sysparm companion, [40]

Arrays of macro variables

Overview
- Data array and loops
- Warning when only single type
- Subroutine: make list values sort
- Make macro array: data, symputx
- Log: array of global macro variables
- Make macro array: sql
- Demo macro %do loop
- Programming issues
- Citations
data array and loops

Here is a generic method to write each row of a data set to the log.

```
*name: demo-data-array.sas;
%let lib_data = sashelp.class;
%let lib_data = sashelp.class(keep=age height weight);*type=n;
DATA _null_;  
   if 0 then set &lib_data; *read data structure;  
array _c(*) $32 _character_; *fragile;  
array _n(*) _numeric_ ; *fragile;  
do _obs = 1 to _n_obs; *loop, enumerated;  
set &lib_data nobs = _n_obs  
point = _obs;  
do _i = 1 to dim(_c); putlog _c(_i)= @;  
end;  
do _i = 1 to dim(_n); putlog _n(_i)= @;  
end;  
p;*CR/LF: newline;  
end;  
stop;  
run;
```

Notes: The array statement is paired with the dim function. The dimension of an array is the CARDINALITY of the array. Compare to programs cx-include page 14 and demo-macro-function page 15.

warning when only single type

The above program gives a warning when all variables are a single type.

```
2 %let lib_data = sashelp.class(keep=age height weight);*type=n;  
3 DATA _null_;  
4 if 0 then set &lib_data;  
5 array _c(*) $32 _character_; *fragile;  
WARNING: Defining an array with zero elements.  
```

Notes: See how to eliminate this warning in program cx-include on page 14.

subroutine: make list values sort

This program is used as a subroutine in the next two examples.

```
*name: make-list-values-sort.sas;
PROC sort data = &lib_data (keep = &var) nodupkey  
   out = list_values;  
by &var;  
run;  
%put echo &=sysnobs;
```

make macro array: data, symputx

This is a classic program which uses the row number, _n_, as a natural key.

```
*name: demo-make-macro-array-data.sas;  
%let lib_data = sashelp.class;  
%let var = sex;  
%include 'make-list-values-sort.sas';  
DATA _null_;  
set &syslast end = endofile nobs = dimension;  
call symputx(catt('value',_n_),&var);  
if endofile then call symputx("dim_values",dimension);  
run;  
%put _global_;
log: array of global macro variables

This log shows the macro variables of the parameters lib_data and var, and the sequentially-numbered macro variables with prefix value and the CARDINALITY of the macro array dim_values.

GLOBAL LIB_DATA  sashelp.class
GLOBAL VAR sex
GLOBAL VALUE1 F
GLOBAL VALUE2 M
GLOBAL DIM_VALUES 2

make macro array: sql

Arrays of macro variable can also be produced with the sql procedure.

*name: demo-make-macro-array-sql.sas;
%let lib_data = sashelp.class;
%let var = sex;
%include 'make-list-values-sort.sas';
PROC sql noprint;
   select &var into :value1 -
       from &syslast;
   quit;
%let dim_values = &sqlobs;
%put _global_
run;

Notes: the high value may be also coded into :value1 - :value&sysmaxlong

demo macro %do loop

Arrays of macro variables are referenced in a macro %do loop using the double ampersand syntax.

*name: demo-echo-macro-array.sas;
%include 'demo-make-macro-array-data.sas';
%include 'demo-make-macro-array-sql.sas';

%macro echo_items();
   %do i = 1 %to &dim_values;
      %put value&i: &&value&i;
   %end;
%mend echo_items;
%echo_items()

programming issues

The name of the macro array is hard-coded and is separate from the macro %do loop.
There is no function for the dimension, it must be fetched.
Macro variables are created in the global symbol table.
Macro with %do loop reads the global symbol table.
Code inside loop cannot be tested as a unit.
All these issues are addressed in the section on loops, page 12.

citations:  
- macro array using call symput used in [12] and [13]  
- macro array using proc sql select into [16]  
- macro array as 1. function which returns dimension, and 2. procedure so that calling macro can declare macro variables local [1]  
- macro do_loop [7]
### Database vocabulary

- **database contains columns**  
  - primary key: row identifier, UNIQUE NATURAL KEY has values 1–nobs  
  - foreign keys: categories, FEW, link to primary key of dimension table  
  - composite key: set of foreign keys  
  - fact: continuous, MANY, measurement or quantity; summable  
  - text: information about primary key  
  - boolean: event success  
  - date: and/or time: event granularity for start/end of periodic snapshot  

- **database contains tables**  
  - fact: is a record of events ex: transaction history, inventory observation  
  - dimension: is also known as a lookup table contains primary key, text  
  - snapshots: are called reports  
  - periodic: e.g. monthly (grain) summary  
  - accumulating: tasks of project, budget record with monthly sums  

- **citations:** database vocabulary, [32] and Kimball and Ross [46]

### The three basic lists

#### Lists

In programming we use the lists of variables (names), data sets (mem-names), and values. These lists are produced with these processes.

- **names**: contents, data with scl functions, sql dictionary.columns  
- **memnames**: contents, sql dictionary.tables  
- **values**: frequency, sort, sql, summary  

- **citations:**  
  - Other commonly-used lists are available from sql dictionaries; these include user- and system-defined global macro variables, `catrefs`, `filerefs`, `librefs`, options, and running text in footnotes and titles.  
  - How to use sql select into for list processing [29] code: [4]  
  - SAS software maintains other functions in catalogs; these include user-defined formats and macro definitions. SASautos companion, [18] lists macro catalogs, code: [3]

### Variable names

#### Overview

Procedures contents and sql produce a list of variable names; scl functions can be used in a data step. Each method differs in the values of type.

- contents (1,2) scl (C,N) sql (char,num)

The programs shown below standardize type into `(c,n)`.

This section contains these topics.

- list-names, data structure  
- list-names, printed  
- list names, contents  
- list names, scl  
- list names, sql
create table work.list_names
libname char(8),
memname char(32),
varnum num,
name char(32),
type char(1),
length num,
format char(40),
label char(40)

libname memname varnum name type length format label
sashelp class 1 Name c 8
sashelp class 2 Sex c 1
sashelp class 3 Age n 8
sashelp class 4 Height n 8
sashelp class 5 Weight n 8

*name: make-list-names-contents-x.sas;
%put echo &libname..&memname;
PROC contents data = &libname..&memname noprint
  out = list_names
  (keep = libname memname varnum name nobs
   type length format label );
run;
DATA &syslast (label = "&memname");
  if 0 then do; * arrange data structure ;
    set &syslast(keep = libname memname nobs varnum name);
    attrib type length = $1;
    set &syslast(keep = length format);
    set &syslast(keep = label);
    end;
  retain _max_length_char 0;
  drop _:; * _temp vars;
  do until(endofile);
    set &syslast (rename = (type = _type_n))
      end = endofile;
    if _type_n = 1 then type = 'n';
    else do;
      type = 'c';
      _max_length_char = max(_max_length_char,length);
    end;
  output;
end;
call symputx('_max_length_char',_max_length_char);
run;
%put echo &=_max_length_char;
PROC sort data = &syslast;	*add attribute: sorted-by;
  by name;
run;
%put trace make-list-names-contents-x ending;

*name: make-list-names-scl-x.sas;
%put echo &libname..&memname;
DATA list_names(label = "&memname");
  attrib libname length = $ 8
    memname length = $32
    n_obs length = 8
    varnum length = 8
    name length = $32
    type length = $ 1
    length length = 8
    format length = $40
    label length = $40;
  drop _::; ** temp vars;
  retain libname "&libname" memname "&memname"
    _max_length_char n_obs 0;
  _dsid = open ("&libname..&memname");

8
n_vars = attrn(_dsid,'nvar');
n_obs = attrn(_dsid,'nobs');
do varnum = 1 to _n_vars;
   name = varname (_dsid,varnum) ;
   type = lowcase(vartype (_dsid,varnum));
   length = varlength(_dsid,varnum) ;
   format = varfmt (_dsid,varnum) ;
   label = varlabel (_dsid,varnum);  
   output;
   if type eq 'c' then
      _max_length_char = max(_max_length_char,length);
   end;
   _rc = close(_dsid);
call symputx('_max_length_char',_max_length_char);
run;

list-names, sql

*name: make-list-names-sql-x.sas  note: does not have nobs;
%put echo &libname..&memname;
PROC sql noprint;
   create table list_names (label = "&memname") as
   select libname, memname, varnum, name, type length=1,
          length, format, label
   from dictionary.columns %*list-names;
   where libname eq "%upcase(&libname)
            and memname eq "%upcase(&memname)
            and memtype eq 'DATA';
   select max(length) into :_max_length_char trimmed
   from &syslast where type eq 'c';
   quit;
%put echo &_max_length_char;
%put trace make-list-names-sql-x ending;

citations:  Code for the above programs and others is available on 2.

Data set names: memnames

Overview  This list is called memnames because that is the variable name in the output data sets of both the contents and sql procedures.

- list-memnames, data structure
- list-memnames, printed
- list-memnames, contents
- list-memnames, sql
- programming issues

The programs shown below add memnum with values (1 : n_obs), which is the natural key described on page 7.

list-memnames, data structure

create table WORK.LIST_MEMNAMES
   libname char(8) label='Library Name',
   memnum num,  label='Member Number',
   memname char(32) label='Member Name',
   nobs num label='Number of Physical Observations',
   nvars num label='Number of Variables',
   memlabel char(256) label='Data Set Label'
list-memnames, printed

<table>
<thead>
<tr>
<th>libname</th>
<th>memnum</th>
<th>memname</th>
<th>nobs</th>
<th>nvars</th>
<th>memlabel</th>
</tr>
</thead>
<tbody>
<tr>
<td>SASHELP 1</td>
<td>AADCMP</td>
<td>1644</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SASHELP 2</td>
<td>AARFM</td>
<td>61</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SASHELP 3</td>
<td>ADSMSG</td>
<td>426</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SASHELP 4</td>
<td>AFMSG</td>
<td>1090</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SASHELP 5</td>
<td>AIR</td>
<td>144</td>
<td>2</td>
<td></td>
<td>airline data (monthly: JAN49-DEC60)</td>
</tr>
</tbody>
</table>

...

list-memnames, contents

*name: make-list-memnames-contents.sas;
%put echo &=libname;
PROC contents data = &libname.._all_ noprint
   out = list_memnames
       (keep = libname memname nobs nvars memlabel);
run;
%put echo &=sysnobs;
DATA &syslast (label = "&libname");
   attrib libname length=$ 8 label='library name'
   memnum length= 8 label='member number'
   memname length=$ 32 label='member name'
   nobs length= 8 label='number of observations'
   nvars length= 8 label='number of variables'
   memlabel length=$256 label='member label';
   retain libname "&libname" memnum nvars 0;
set &syslast;
   nvars +1;
by memname;
if last.memname then do;
   memnum +1;
   output;
   nvars = 0;
end;
run;
%put trace make-list-memnames-contents ending;

list-memnames, sql

*name: make-list-memnames-sql.sas;
PROC sql noprint;
   create table list_memnames(label = "&libname") as
      select libname,
         monotonic() as memnum label='member number',
         memname, nobs, nvar as nvars, memlabel
      from dictionary.tables
      where libname eq "%upcase(&libname)"
      and memtype eq 'DATA';
quit;
%put trace make-list-memnames-sql ending;

programming issues

The sql dictionaries of all librefs are refreshed before each procedure call; the contents procedure reads only the libref specified and is therefore faster.
This information is also available in a set of sashelp views; these views read from sql dictionaries and are deprecated by the author for that reason.

Values Overview

This section shows the macro proc-freq and the data structure. The purpose of this macro is produce a frequency of each variable, standardize the data structure and stack the outputs into one data set.

This is the list of topics in this section.

- macro proc-freq
- list-values, data structure
- list-values, printed
This macro is called by two programs, demo-call-macro-call-execute on page 12 and demo-call-macro-sql-basic on page 13. Compare to macro proc-freq-dosub on page 23.

```sas
%macro proc_freq(name,type);
  %put echo &libname..&memname..&name &=type;
  %let type = %lowcase(%substr(&type,1,1));
  PROC freq data = &libname..&memname; *fragile: global;
    tables &name / noprint
       out = frequency
         (rename = (&name = valu_&type));
  run;
  DATA &syslast;
    attrib memname length = $32
      name length = $32
      valu_c length = $32 %*fragile;
    /** valu_c length = $&_max_length_char /**robust*/
      valu_n length = 8
      count length = 8
      percent length = 8
      level length = 8;
    retain memname "&memname" name "&name"
      valu_c '.' valu_n . ;
  set &syslast;
    level = _n_; *natural key;
  run;
  PROC append data = &syslast
    base = list_values;
  run;
  %put trace &sysmacroname(&=name) ending;
%mend proc_freq;
```

This macro is adapted from [20].

**list-values, data structure**

Create table WORK.LIST_VALUES

<table>
<thead>
<tr>
<th>libname char(8),</th>
<th>memname char(32),</th>
<th>varnum num,</th>
</tr>
</thead>
<tbody>
<tr>
<td>name char(32),</td>
<td>valu_c char(??),</td>
<td>valu_n num,</td>
</tr>
<tr>
<td>valu_n char(??),</td>
<td>valun num,</td>
<td>count num,</td>
</tr>
<tr>
<td>percent num,</td>
<td>leven num</td>
<td></td>
</tr>
</tbody>
</table>

**list-values, printed**

This report shows cardinality of each variable on its last line <<<<<.

<table>
<thead>
<tr>
<th>Obs</th>
<th>memname</th>
<th>name</th>
<th>valu_c</th>
<th>valu_n</th>
<th>count</th>
<th>percent</th>
<th>level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>class</td>
<td>Age</td>
<td>11.0</td>
<td>2</td>
<td>10.5263</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>class</td>
<td>Age</td>
<td>12.0</td>
<td>5</td>
<td>26.3158</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>class</td>
<td>Age</td>
<td>15.0</td>
<td>4</td>
<td>21.0526</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>class</td>
<td>Age</td>
<td>16.0</td>
<td>1</td>
<td>5.2632</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>class</td>
<td>Height</td>
<td>51.3</td>
<td>1</td>
<td>5.2632</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>class</td>
<td>Height</td>
<td>56.3</td>
<td>1</td>
<td>5.2632</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>[skipped]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>class</td>
<td>Height</td>
<td>69.0</td>
<td>1</td>
<td>5.2632</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>class</td>
<td>Height</td>
<td>72.0</td>
<td>1</td>
<td>5.2632</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>[skipped]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>class</td>
<td>Name</td>
<td>Alfred</td>
<td>1</td>
<td>5.2632</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>class</td>
<td>Name</td>
<td>Alice</td>
<td>1</td>
<td>5.2632</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>[skipped]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>class</td>
<td>Name</td>
<td>Thomas</td>
<td>1</td>
<td>5.2632</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>class</td>
<td>Name</td>
<td>William</td>
<td>1</td>
<td>5.2632</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>class</td>
<td>Sex</td>
<td>F</td>
<td>9</td>
<td>47.5884</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>class</td>
<td>Sex</td>
<td>M</td>
<td>10</td>
<td>52.6316</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>[skipped]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>class</td>
<td>Weight</td>
<td>50.5</td>
<td>1</td>
<td>5.2632</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>[skipped]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>class</td>
<td>Weight</td>
<td>150.0</td>
<td>1</td>
<td>5.2632</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

**programming issues**

The length of variable valu_c is marked as fragile; note that the programs that produce the data set list-names shown above on 8-9 also provide a macro variable max-length-char.
Loops: processes for lists

Overview

This section contains the specific routines that are the predecessors to the three general-purpose, reusable routines.

- specific solutions
  - type macro calls, how to automate this?
  - call execute macro
  - sql select into
- general solutions, reusable
  - cx-include: call execute an %include
  - call-macro: call a macro
  - call-text: create and expand references to macro variable in text

Notes: Both call-macro and call-text are macro functions; they do not return SAS statements, only macro language text.

type the macro calls

In the beginning, cut and paste the list of variable names from the contents output and type in the constant text of the macro calls.

*name: demo-typed-list-values.sas;
/** use list from proc contents printed output;
Alphabetic List of Variables and Attributes
# Variable Type Len
3 Age Num 8
4 Height Num 8
1 Name Char 8
2 Sex Char 1
5 Weight Num 8
/********************/
%let libname = sashelp;
%let memname = class;

%proc_freq(Age ,Num )
%proc_freq(Height,Num )
%proc_freq(Name ,Char)
%proc_freq(Sex ,Char)
%proc_freq(Weight,Num )
PROC print;
run;

Specific Routines

call execute macro calls

The call execute routine has been around since SAS.v6. This program shows how to call a macro with two parameters.

*name: demo-call-macro-call-exec.sas;
%let libname = sashelp;
%let memname = class;
%include 'make-list-names-contents-x.sas';
DATA _null_;
do until(endofile);
  set list_names  end = endofile;
  *call exec of a macro is fragile without nrstr;
call execute(catt('%proc_freq(name=',name,',type=',type,')'));
end;
stop;
run;
This solution may fail, not because of the code in this program, but when the macro definition contains what I call complexity: it contains any of these statements: %do, %if, call symput, or symputx.

The error is difficult to diagnose because the macro will pass a unit test and this program will work with simple macros but not with complex ones.

An explanation of the timing issues is in Fehd and Carpenter [44].

The solution with %nrstr is shown in program cx-include on page 14.

`sql select into

The sql dictionary.columns eliminates the separate step of making the list of variable names.

*name: demo-call-macro-sql-basic.sas;
%let libname = sashelp;
%let memname = class;
PROC sql noprint;
  select catt('%proc_freq(',%*fragile length=200;
    ,name,',',type,')')
  into :list %*fragile length=65534;
  separated by ',' %*list-names;
  from dictionary.columns %*list-names;
  where libname eq '%upcase(&libname)'
    and memname eq '%upcase(&memname)'
    and memtype eq 'DATA';
quilt;
&list
%symdel list;
run;
PROC print;
run;

Notes: Ah, yes, this is my most spectacular fail! I published this in the fall of 2007 [21], and one week later got an e-mail from a user complaining that it broke on a data set with three thousand (3,000) variables!

It may fail for either of two reasons.

- cat function: the default length of text returned by any of the cat functions is $200; the short length results in truncation of the macro call, which has no closing parenthesis. This can be fixed by adding a length clause after the function call: catt(...) length = $nnnn.

- macro variable: the maximum length of text in a macro variable is $(2^{16}) - 2 = 65,534; a possible solution is to reduce the name of the macro definition to a single character: %macro z(...) otherwise, this error cannot be fixed; consider one of the robust solutions, shown next.

citations: A tutorial on sql is available in [29].
Call execute an %include

Overview

This is the main algorithm of CallXproc.sas from the SmryEachVar suite of sgf.2008 [24]. This program was published later as CallXinc.sas [27]. Compare to the demonstration program demo-data-array.sas on page 5.

Note that the length of the variable _stmt is marked as fragile. The value is max(for i=1 to n-vars (length(%let var(i)=value(i));),length(%include &cx_inclu))

where var(i) is length(name(i)) and value(i) is length(i).

Citations:

Robust code for cx-include is available as callxinc on [26] which has a data step to calculate the maximum length for _stmt.

cx-include

*name: cx-include.sas;
%put echo &cx_data &=cx_include;
DATA _null_;
  if 0 then set &cx_data;
  attrib _stmt length = $128 %*fragile;
  _name length = $ 32
  _i length = 4;
  array _var_c(*) _character_; *robust: _name;
  array _var_n(*) _numeric_; *robust: _i;
  do until(endofile);
    set &cx_data end = endofile;
    do _i = 1 to dim(_var_c) -2; *note: dim()-2;
      _name = vname(_var_c(_i));
      _stmt = catx(' ', '%let', _name, '=' , _var_c(_i));
      link cx_stmtnt;
    end;
    do _i = 1 to dim(_var_n) -1; *note: dim()-1;
      _name = vname(_var_n(_i));
      _stmt = catx(' ', '%let', _name, '=' , _var_n(_i));
      link cx_stmtnt;
    end;
  _stmt = "%include &cx_include"
  link cx_stmtnt;
  end;
  stop;
  cx_stmtnt: call execute(cats('%nrstr(', _stmt , ');'));
  return;
run;

Notes:

This program hides the complexity of the call execute(...) in the data step subroutine labeled cx_stmtnt which is called with the link statement.

Example usage of this program is shown in make-list-cr-types-n-steps on page 19 and make-list-cr-types-1-step-libname on page 25.
Demo macro function which reads a data set

Overview
This demonstration macro shows the common code in macros call_macro and call_text. Compare to programs demo-data-array.sas on page 5 and cx-include on page 14. This program uses scl functions like make-list-names-scl-x on page 8.

Notes: This function calls either of the functions getvarc or getvarn with the reference getvar&type. This trick eliminates the conditional statements %if &type eq c %then getvarc(); %else getvarn().

demo macro function

%MACRO demo_macro_function(data = sashelp.class);
  %local dsid n_obs n_vars obs rc name type value;
  %let dsid = %sysfunc(open (&data ));
  %let n_obs = %sysfunc(attrn(&dsid,nobs ));
  %let n_vars = %sysfunc(attrn(&dsid,nvars));
  %do obs = 1 %to &n_obs;
    %let rc = %sysfunc(fetchobs(&dsid,&obs));
    %do varnum = 1 %to &n_vars;
      %let name = %sysfunc(varname(&dsid,&varnum));
      %let type = %sysfunc(vartype(&dsid,&varnum));
      %let value = %sysfunc(getvar&type(&dsid,&varnum));
      %put echo &name = &value;
    %end;
  %end;
  %let rc = %sysfunc(close(&dsid));
%mend demo_macro_function;

log

echo Name = Alfred
echo Sex = M
echo Age = 14
echo Height = 69
echo Weight = 112.5

echo Name = Alice
echo Sex = F
echo Age = 13
echo Height = 56.5
echo Weight = 84

15
The purpose of macro `call_macro` is to assemble a macro call of the form:
\%macro-name(var(i)=value(column(i),row(j)),...)

The %do loops for obs and varnum and the fetch of name and type are exactly the same as shown above in demo-macro-function.

---

**call-macro**

\%MACRO call_macro
  (data = sashelp.class /* required */
   .macro_name = put); /* default for testing */
  %let dsid = %sysfunc(open (&data));
  %let n_obs = %sysfunc(attrn(&dsid,nobs));
  %let n_vars = %sysfunc(attrn(&dsid,nvars));
  %do obs = 1 %to &n_obs;
  %let list_parameters =;
  %let rc = %sysfunc(fetchobs(&dsid,&obs));
  %do varnum = 1 %to &n_vars;
    %let name = %sysfunc(varname(&dsid,&varnum));
    %let type = %sysfunc(vartype(&dsid,&varnum));
    %* add name=value to list;
    %let list_parameters=&list_parameters&name=%left(%sysfunc(getvar&type(&dsid,&varnum)));
    %* add comma to end of parameter list;
    %if &varnum lt &n_vars %then
      %let list_parameters = &list_parameters,;
    %end;
  %&macro_name(&list_parameters); %*semicolon not required!;
%end;
%let rc = %sysfunc(close(&dsid));
%mend call_macro;

---

citations: Macro `callmacr` is described in [34], code is available on [5].
The purpose of macro `call_text` is to assemble macro assignment statements and then expand `&text` which contains references to `&var1--&varn`. 
\%let var1=value1;...;\%let varn=valuen;\%unquote(&text)

```
%MACRO call_text
   (data = sashelp.class
    ,text = \%nrstr(\%put echo: &=name sex=&sex;
    ,global = 0); /* make mvars global? */
\%let _global = \%eval(not(0 eq &global));
\%let _text = &text;
\%let _dsid = \%sysfunc(open (&data ));
\%let _n_obs = \%sysfunc(attrn(&_dsid,nobs ));
\%let _n_vars = \%sysfunc(attrn(&_dsid,nvars));
\%do _obs = 1 %to &_n_obs;
\%let _rc = \%sysfunc(fetchobs (&_dsid,&_obs ));
\%do _varnum = 1 %to &_n_vars;
\%let _name = \%sysfunc(varname (&_dsid,&_varnum));
\%let _type = \%sysfunc(vartype (&_dsid,&_varnum));
\%if &_global %then %global &_name;
\%else %local &_name;
\%let &name= %sysfunc(getvar&_type(&_dsid,&_varnum));
\%*** note functions: getvarc and getvarn;
\%end;
\%unquote(&_text)
\%end;
\%let _rc = \%sysfunc(close(&_dsid));
\%mend call_text;
```

### programming issues

Macros `call-macro` and `call-text` fail when text variables such as labels contain special characters of the macro language, ampersand (`&`), percent (`%`), or function delimiter, comma (`,`), or statement delimiter, semicolon (`;`), or unmatched (single or double) quotes or parentheses.

The solution is to drop those variables, `data=x(drop=label)`.

### citations:

Macro `calltext` is described in [41], code is available on [6].

Code for macros `call-macro` and `call-text` contains tests for the existence of the data set; tests for existence of various SAS objects is shown in [43] code: [9].

Macro `dateloop` is a function which returns a series of dates and surrounding text; it is described in [31] code: [8]
A function to calculate cardinality ratios and types

Overview

The cardinality of a set is the number of elements in the set. The cardinality of a data set is the number of rows in the data set. The cardinality of a variable is the number of distinct values within the variable; in SAS this is called *n-levels* and is available from the frequency procedure. The CARDINALITY RATIO (CR) of a variable is the cardinality divided by the data set nobs. The CARDINALITY TYPE (cr-type) is determined by comparing the CR with the mean of all the CR in the data set.

This section contains these programs.

- N steps
- Two steps with freq.nlevels and scl functions
- One step with scl and dosubl
- Cardinality ratio types of all memnames in a libref

This overview contains these topics.

- cr-types, data structure
- cr-types, printed
- algorithm

```sql
create table WORK.CR_TYPES(label='class nobs=19 nvars=5')
    memname char(32),
    varnum num,
    cr_type char(10),
    card_ratio num format=BESTD7.5,
    n_levels num format=BEST8. label='Number of Levels',
    name char(32) label='Table Variable',
    type char(1),
    length num,
    format char(49),
    label char(256)
```

```plaintext
cardinality types, printed

<table>
<thead>
<tr>
<th>memname</th>
<th>varnum</th>
<th>cr_type</th>
<th>card_ratio</th>
<th>n_levels</th>
<th>name</th>
<th>type</th>
<th>length</th>
<th>format</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>1</td>
<td>unique</td>
<td>1</td>
<td>19</td>
<td>Name</td>
<td>c</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>2</td>
<td>few</td>
<td>0.10526</td>
<td>2</td>
<td>Sex</td>
<td>c</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>3</td>
<td>few</td>
<td>0.31579</td>
<td>6</td>
<td>Age</td>
<td>n</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>4</td>
<td>many</td>
<td>0.89474</td>
<td>17</td>
<td>Height</td>
<td>n</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>5</td>
<td>many</td>
<td>0.78947</td>
<td>15</td>
<td>Weight</td>
<td>n</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

algorithm

1. fetch n-obs: nobs(data set) or from list-memnames
2. for each variable, fetch n-levels
3. cardinality-ratio = n-levels / n-obs
4. calculate mean of cardinality ratios
5. cardinality-ratio-type = compare cardinality ratio to mean

citations: The first paper to identify cardinality ratio was [25]; other papers include [32] and [33]. Macro `cr-calc`, shown in [35], is the basis for the n-steps programs shown next. This paper also includes programs for the mode and summary procedures.
Overview

Implementation of this algorithm requires $7 + n\text{-vars}$ steps. This is the list of topics in this section.

- make-list-cr-types-n-steps
- proc-freq-out-name
- make-list-card-ratios-from-work-name
- proc-smry-cr
- make-list-types-from-cr

make-list-cr-types-n-steps

This program may also use the macro calltext for the loop.

```sas
*name: make-list-cr-types-n-steps.sas;
%let libname = sashelp;
%let memname = class;
%include 'make-list-names-contents-x.sas'; *steps=2;
/**** loop: freq of each variable */
%let cx_data = &syslast;
%let cx_include = 'proc-freq-out-name.sas'; *steps=nvars;
%include 'cx-include.sas'; *steps=1;
/*************************************************************/
%include 'make-list-card-ratios-from-work-name.sas'; *steps=1;
%include 'proc-smry-cr.sas'; *steps=1;
%include 'make-list-types-from-cr.sas'; *steps=1;
PROC print; *steps=1;
  title3 "Cardinality Ratios and Types: &libname..&memname";
run;
```

proc-freq-out-name.sas

```sas
*name: proc-freq-out-name.sas;
PROC freq data = &libname..&memname;
tables &name / noprint out = freq_&name; * fragile: length=$32;
run;
```

Compare to programs proc-freq page 11 and proc-freq-dosub page 23

make-list-card-ratios-from-work-name.sas

```sas
*name: make-list-card-ratios-from-work-name.sas;
PROC contents data = work._all_ noprint out = list_n_levels (keep = memname name nobs label where =( memname like 'FREQ_%' and label eq ' ')
rename =(nobs = n_levels));
run;
PROC sort data = &syslast;
  by name;
DATA list_card_ratios;
do until(endofline);
  merge list_names &syslast (keep = name n_levels) end = endofile;
    by name;
    card_ratio = n_levels / nobs;
    output;
end;
stop;
run;
```
**proc-smry-cr.sas**

```sas
*name: proc-smry-cr.sas;
PROC summary data = list_card_ratios;
   output out = _mean_cr (drop = _freq_ _type_)
      mean(card_ratio)= _mean_cr;
run;
```

**make-list-types-from-cr.sas**

```sas
*name: make-list-types-from-cr.sas;
DATA list_cr_types;
drop _:;
set _mean_cr;
putlog 'echo' _mean_cr=;

do until(endofile);
   set list_card_ratios end = endofile;
   select;
      when(n_levels eq 1) cr_type = 'n-levels=1';
      when(card_ratio eq 1) cr_type = '.unique';
      when(card_ratio gt _mean_cr) cr_type = 'many';
      otherwise cr_type = 'few';
      end;
   output;
   end;
stop;
run;
```

**programming issues**

This algorithm contains many steps which are interdependent; this makes unit testing a sequential process.

**Two steps with freq.nlevels and scl functions**

**Overview**

This section contains these topics.

- fetch n-obs, n-vars
- fetch n-levels
- describe output data structure
- calculate cardinality ratios and mean
- assign cr-type

**fetch n-obs, n-vars**

```sas
%let _dsid = %sysfunc(open (&libname..&memname,i));
%let _n_obs = %sysfunc(attrn(&_dsid,nobs));
%let _rc = %sysfunc(close(&_dsid));
```

**fetch n-levels**

```sas
ODS select none; * noprint;
PROC freq data = &libname..&memname nlevels;
   odss output
      nlevels= list_names
         (keep = tablevar nlevels)
   rename=(tablevar= name nlevels= n_levels);
run;
ODS select all;
```
describe data
structure

DATA &syslast;
  attrib memname length = $32
  varnum length = 8
  cr_type length = $ %length(n-levels=1)
  card_ratio length = 8 %*range=(0:1];
  format = bestd7.5
  n_levels length = 8
  name length = $32
  type length = $ 1
  length length = 8
  format length = $49
  label length =$256;

array _cr(&_n_vars);

** loop: for each row, calculate cardinality ratio;
do _i = 1 to dim(_cr);
  set &syslast (keep = n_levels) point = _i;
  _cr(_i) = n_levels / &_n_obs; %*global mvar;
end;

***** calculate mean for select card_ratio to cr_type;
  _mean_cr = mean(of _cr(*));

fetch variable
information

_dsid = open("&libname..&memname");
do _i = 1 to dim(_cr);
  set &syslast point = _i;
  ....;
  card_ratio = _cr(_i);

Notes: The assumption is data and freq.n-levels are in varnum order.

assign cr-type

select;
  when(n_levels eq 1 ) cr_type = 'n-levels=1';
  when(card_ratio eq 1 ) cr_type = '.unique';
  when(card_ratio gt _mean_cr) cr_type = 'many';
  otherwise cr_type = 'few';
end;
output;

citations: There are a number of functions for arrays, this paper [38] shows call sortn.

Previous papers on cardinality ratio include [32, 33], [35]
make-list-cr-types-2-steps

*name: make-list-cr-types-2-steps.sas;
*called by: make-list-cr-types-2-steps-memname.sas
ODS select none; * noprint;
PROC freq data = &libname..&memname nlevels;
  ods output
    nlevels= list_names
      (keep = tablevar nlevels
        rename=(tablevar= name nlevels= n_levels));
run;
ODS select all;
%let _dsid = %sysfunc(open (&libname..&memname,i));
%let _n_obs = %sysfunc(attrn(&_dsid,nobs));
%let _n_vars = %sysfunc(attrn(&_dsid,nvar));
%let _rc = %sysfunc(close(&_dsid));
DATA cr_types(label = "&memname nobs=&_n_obs nvars=&_n_vars");
  attrib memname length = $32
    varnum length = 8
    cr_type length = $ %length(n-levels=1)
    card_ratio length = 8 %*range=(0:1];
  format = bestd7.5
    n_levels length = 8
    name length = $32
    type length = $ 1
    length length = 8
    format length = $49
    label length =$256;
  array _cr(&_n_vars);
  drop _:; * _temp vars;
  retain memname "&memname";
  ** loop: for each row, calculate cardinality ratio;
  do _i = 1 to dim(_cr);
    set &syslast (keep = n_levels) point = _i;
    _cr(_i) = n_levels / &_n_obs; *global macro variable;
  end;
  ***** calculate mean for select card_ratio to cr_type;
  _mean_cr = mean(of _cr(*));
  _dsid = open("&libname..&memname");
  do _i = 1 to dim(_cr);
    set &syslast point = _i;
    varnum = varnum (_dsid,name );
    type = lowcase(vartype (_dsid,varnum));
    length = varlength(_dsid,varnum);
    format = varfmt (_dsid,varnum);
    label = varlabel (_dsid,varnum);
    card_ratio = _cr(_i);
    select;
      when(n_levels eq 1 ) cr_type = 'n-levels=1';
      when(card_ratio eq 1 ) cr_type = '.unique';
      when(card_ratio gt _mean_cr) cr_type = 'many';
      otherwise cr_type = 'few';
    end;
    output;
  end;
  _rc = close(_dsid);
stop;
run;

Notes: The freq.nlevels data set is output in varnum order so _i is varnum.
One step with scl functions and dosubl

Overview
This is the list of topics in this section.
- make list cr types for memname
- proc freq dosub
- programming issues
- make list cr types 1 step
- calculate card-ratios and mean
- assign cr-type

make list cardinality
types for memname

*name: make-list-cr-types-l-step-memname.sas;
* see also make-list-cr-types-l-step-libname.sas;
%let libname = sashelp;
%let memname = class;
%let memnum = 0;
%let out_lib = library;
%let out_lib = work;

%!include 'make-list-cr-types-l-step.sas';

PROC print data = &syslast noobs;
title3 "Cardinality Ratios, Types of &libname..&memname";
run;

proc freq dosub
%macro proc_freq_dosub
(data = &libname..&memname /*global*/
 ,name =
 ,type =
 ,varnum =
 ,memnum = 0
 ,out_lib = work );
%put echo &sysmacroname &=name &=type &=varnum &=memnum;
PROC freq data = &data;
tables &name / noprint
 out = &out_lib..freq_&memnum._&varnum
 (label = "&name"
 rename = (&name = valu_&type));
run;
%put echo &sysmacroname &=sysnobs;
%global _n_levels;
*robust: avoid divide by zero in calling program;
*** sysnobs returns -1 for data-set w/nobs=0;
%if &sysnobs gt 0 %then
 %let _n_levels = &sysnobs;
%else %let _n_levels = 1;
%end proc_freq_dosub;

programming issues
In the first edition of this macro I assumed that the global system macro variable sysnobs would be updated. As you can see in the log the statement
%put echo &sysmacroname &sysnobs;
%global _n_levels;
*robust: avoid divide by zero in calling program;
*** sysnobs returns -1 for data-set w/nobs=0;
%if &sysnobs gt 0 %then
 %let _n_levels = &sysnobs;
%else %let _n_levels = 1;
%end proc_freq_dosub;
Thus the solution shown here: allocate a global macro variable n-levels and conditionally set its value to sysnobs.

citations: Function dosubl is described in Langston [47].
* name : make-list-cr-types-1-step.sas;
* called by: make-list-cr-types-1-step-memname.sas;
* called by: make-list-cr-types-1-step-libname.sas;
%put echo &libname..&memname &=memnum &=out_lib;
%let _dsid = %sysfunc(open (&libname..&memname));
%let _n_obs = %sysfunc(attrn(&_dsid,nobs));
%let _n_vars = %sysfunc(attrn(&_dsid,nvar));
%let _rc = %sysfunc(close(&_dsid));
DATA &out_lib..list_cr_types_&memnum
  (label = "memname=&memname,obs=&_n_obs,vars=&_n_vars");
attrib memname length = $32
memnum length = 8 label = 'mem num'
varnum length = 8 label = 'var num'
cr_type length = $ %length(n-levels=1)
  label = 'card. ratio type'
card_ratio length = 8 label = 'card. ratio'
  format = bestd7.5
n_levels length = 8 label = 'n-levels'
name length = $32
type length = $ 1
length length = 8
format length = $49
label length = $256;
array _cr (&_n_vars) ;
array _name (&_n_vars) $32;
array _nlvl (&_n_vars) ;
array _type (&_n_vars) $ 1;
drop _;:
retain memname "&memname" memnum &memnum;
_dsid = open("&libname..&memname");
do varnum = 1 to dim(_name);
  _name (varnum)= varname(_dsid,varnum) ;
  _type(varnum)= lowcase(vartype(_dsid,varnum));
  _rc = dosubl(catt('%proc_freq_dosub(name=','_name(varnum)
    ,type=','_type(varnum)
    ,varnum=' ,varnum
    ,memnum=&memnum,out_lib=&out_lib)"));
  _nlvl(varnum) = input(symget('_n_levels'),32.);
  if _nlvl(varnum) and &_n_obs then
    _cr(varnum) = _nlvl(varnum)/&_n_obs;
  else _cr(varnum) = 1/$_sysmaxlong;*small and non-zero;
end;
***** calculate mean for select card_ratio to cr_type;
_mean_cr = mean(of _cr(*));
do varnum = 1 to dim(_name);
  name = _name ( varnum);
  card_ratio = _cr ( varnum);
  n_levels = _nlvl ( varnum);
  type = _type ( varnum);
  length = varlength(_dsid,varnum);
  format = varfmt (_dsid,varnum);
  label = varlabel (_dsid,varnum);
select;
  when(n_levels eq 1 ) cr_type = 'n-levels=1';
  when(card_ratio eq 1 ) cr_type = '.unique';
  when(card_ratio gt _mean_cr) cr_type = 'many';
  otherwise cr_type = 'few';
end;
output;
end;
_rc = close(_dsid);
run;
Cardinality types of all memnames in a libref

Overview

Notice that program make-list-cr-types-1-step-memname on page 23 has a parameter memnum.
Routine make-list-cr-types-1-step on page 24 creates a report with a unique name make-list-cr-types-* and passes this on to macro proc-freq-dosub on page 23 which creates a unique name for each output data set freq-(memnum)-(varnum) from the frequency procedure.

This section has these topics.

- make list cr types for libname
- list of data sets

make list cr types for libname

* name:  make-list-cr-types-1-step-memname.sas;
* see also make-list-cr-types-1-step-memname.sas;
%let libname = sashelp; *17K files! time= 5+ hours!;
%let libname = library;
*let libname = my_lib;
%let out_lib = library;
%let out_lib = work;

%include 'make-list-memnames-contents.sas';
%let cx_data = &syslast(keep = nobs memname memnum
where =(nobs));
%let cx_include = 'make-list-cr-types-1-step.sas'/source2;
%include 'cx-include.sas'/nosource2;

PROC sql; select memname, nobs, nvar, memlabel length=32
from dictionary.tables
where libname eq "%upcase(&out_lib)"
and memtype eq 'DATA';
quit;
runk;

citations:  Programs in this paper are available on [10].

list of data sets

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Number of</th>
<th>Number of</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Physical</td>
<td>Vars</td>
</tr>
<tr>
<td>FREQ_1_1</td>
<td>9</td>
<td>3 locale</td>
</tr>
<tr>
<td>FREQ_1_2</td>
<td>184</td>
<td>3 key</td>
</tr>
<tr>
<td>FREQ_1_3</td>
<td>1</td>
<td>3 lineno</td>
</tr>
<tr>
<td>FREQ_1_4</td>
<td>1345</td>
<td>3 text</td>
</tr>
<tr>
<td>FREQ_2_1</td>
<td>1</td>
<td>3 locale</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIST_CR_TYPES_1</td>
<td>4</td>
<td>11 memname=AACOMP, obs=1544, vars=4</td>
</tr>
<tr>
<td>LIST_CR_TYPES_2</td>
<td>4</td>
<td>11 memname=AARFM, obs=61, vars=4</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Summary

Conclusion

A SAS data set is an example of a list, the attribute items of which — \( n \)-obs and \( n \)-vars — contain information that can be used to process each variable and provide additional attributes of that variable — \( n \)-levels — that can be compared not only to the other variables but also to the data set attribute, \( n \)-obs. The new attributes of cardinality ratio and cardinality type explain the three major categories of variables: row-identifier: unique, classification: few, and analysis: many.

This information can then be used to coordinate other programming tasks.

future work

This paper describes a set of tools which can be used to analyze a data set or all data sets in a libref. Here are my plans for future work.

- make list of user-defined formats and convert data set of frequency of variable to frequency of variable with format, \([15]\)
- for card-type equal many and type equal numeric,
  - find macro freqall in \([20]\) and modify it into macro freq_hilo; produce a report of all the high and low values of each numeric
  - write a macro smry_from_freq where the summary procedure reads the frequency output data set with weight count;
- for few: calculate max-length-char and write a report to stack (set) or append into one data set all the categorical variables
- \( all(few) \times each(many) \): make a list of the cr-type eq ‘few’ variables and create a program to do a cross-tabulation and summary of each cr-type eq ‘many’ variables, where type eq numeric per algorithm of Raithel \([48]\)
- exclude cr-type of UNIQUE and N-LEVELS=1 from calculation of \( \text{mean}(cr) \)

The SAS community wikipedia has pages for the programs shown here. See also the category macros by Ron Fehd.

- macro arrays \([11]\) 1998, v1, call symputx
- \([17]\) 2004, v2, sql select into \([11]\) 2007, v3, a function and procedure
- making lists contains make-list-memnames, make-list-names and ml-name-x:
  - make-list-names-cardinality-ratio-2-steps, page \([22]\)
- loops \(cx\)-include, formerly callxinc 2008 and cxinclude 2012
  - call-macro
  - call-text
- cardinality ratio database vocabulary summarize each var
- program design \([19]\), \([22]\), \([39]\) and \([30]\)

program downloads

Acknowledgements

Quentin McMullen and Rick Langston reviewed the next-to-last draft and provided helpful commentary. Art Carpenter shared the idea of a list as a control data set.

Author Information

Ronald J. Fehd

Ron.Fehd.macro.maven@gmail.com

http://www.sascommunity.org/wiki/Ronald_J._Fehd

www.linkedin.com/Ronald.Fehd

Stakana Analytics, Senior Maverick

also known as macro maven on SAS-L, Theoretical Programmer

sco.wiki

LinkedIn

affiliation

About the author:
References


Ronald J. Fehd. Writing macro do loops with dates from then to when. In *MidWest SAS Users Group Annual Conference Proceedings*, 2013. URL http://analytics.ncsu.edu/seesug/2013/CC-03.pdf 20 pp.; topics: dates are integers, formats and functions to convert date references to integers, calculations, bibliography.


Ronald J. Fehd. A sysparm companion, passing values to a program from the command line. In *MidWest SAS Users Group Annual Conference Proceedings*, 2016. URL http://www.mwsug.org/proceedings/2016/TT/MWSUG-2016-TT04.pdf. Tools of Trade, 8 pp.; shows use of sysparm as macro variable and option which can be assigned value on command line in batch programs; program parse-sysparm parses a list of comma-separated values (csv) of form var1=value1,var2=value2,...,varN=valueN into macro variables.

Ronald J. Fehd. List processing macro call-text. In *MidWest SAS Users Group Annual Conference Proceedings*, 2016. Tools of Trade, 10 pp.; using %sysfunc with SCL functions to read a list, a control data set, and for each observation, return %unquoted text.


