

A Cross-reference for SAS Data Libraries

John R. Gerlach, Maxim Group, Plymouth Meeting, PA
Cindy Garra, IMS HEALTH; Plymouth Meeting, PA

Abstract

SAS® data libraries often resemble a relational model when the several data sets contain common variables, or in the vernacular of relational theory, common attributes exist across two or more tables. In order to develop an understanding of a SAS data library, it is useful to recognize these common variables, especially those that define observations, in an attempt to perceive a relational model. Using Dictionary tables in SAS, it's possible to produce a cross-reference of all the variables and data sets (i.e., attributes and tables) that exist in a data library. The cross-reference offers immediate insight about the data library, its organization and relationships. This paper discusses Dictionary tables and explains a macro that produces a cross-reference report on any SAS data library.

Introduction

When doing data analysis that requires the use of many SAS data sets, even creating more data sets in the process, the analyst / programmer must have and maintain an intimate understanding of the data library. That is, it's imperative to know how the data sets are related to each other, as well as to recognize that which uniquely identifies an observation in a given data set. How else can you perform merges, concatenations, or SQL joins, efficiently and correctly?

The CONTENTS procedure offers the easiest method for acquiring an understanding of a SAS data library. Consider the simple example below.

```
proc contents data=saslib._all_;  
  title 'Contents of a SAS Data Library';  
run;
```

The automatic variable `_ALL_` allows the procedure to produce a report on the whole data library, listing the contents of every data set in the library. However, the procedure does not show any relationships between the several tables, that is, how variables function as primary and foreign keys across tables.

Given a SAS data library, especially an inherited one with which you are unfamiliar, it would be nice to have a report that lists all the variables in the data library cross-referenced with the names of the data sets. And, the report would indicate the number of observations in each data set. Even better, the report would indicate the data type, N for numeric and C for character type data, or

a dash when the variable does not exist in the respective data set. To illustrate this report, assume a cross-reference involving three SAS data sets, as follows:

Cross-reference of Data Library (STUDY01)			
RX Attribute	COHORT	Tables	DX
	(865)	(2,149)	(5,387)
DIAG	-	C	-
DXDATE	-	N	-
GENDER	C	-	-
NDC	-	-	N
PATID	C	N	C
RXDATE	-	C	N
USC	-	-	N

Notice that the report displays the data sets (tables) in alphabetical order and indicates the number of observations in each data set, as well as listing the variables (attributes) and their existence in each data set, denoted by the data type of the variable. Obviously, the dash denotes that the variable does not exist in the respective data set. Notice, in the example above, the report reveals that the data type for PATID is not consistent across data sets, which poses a possible problem when processing the data.

In developing a relational understanding of a SAS data library, it's important to realize the primary key for each data set. In the example, the variable PATID uniquely identifies each observation in the COHORT data set; whereas, the variables PATID and DXDATE uniquely identifies observations in the DX table; and, the variables PATID and RXDATE uniquely identifies observations in the RX table.

It makes sense that the tables DX and RX would contain more observations. Also, assuming that every patient has at least one diagnosis and one prescription, a projection of either table should identify every patient in the COHORT table, exactly. In relational theory, this property is known as referential integrity. That is, in our case, there cannot be a patient diagnosis or prescription without that patient existing in the COHORT table.

Clearly, the cross-reference report offers a concise understanding of a SAS Data Library, its size and organization. Of course, however, the code that generates the report offers challenging lessons in SAS.

Dictionary Tables

Dictionary tables are special read-only SQL object files that contain useful information about SAS data libraries, data sets, catalogs, and other pertinent details about a SAS session. In lieu of using the SQL procedure and Dictionary tables, there are SAS views stored in the SASHELP library that access the same information. The CONTENTS procedure affords a way to list these views, as follows:

```
proc contents data=sashelp._all_
    memtype=view nods;
run;
```

In order to facilitate our understanding of these sixteen SQL views, they are listed by purpose, that is, the kind of information available. Indeed, some of the views produce similar, albeit more specific, information.

SAS Data Sets:

**VTABLE, VSTABLE, VSTABVW, VCOLUMN,
VSVIEW, VVIEW, VMEMBER**

SAS Catalogs:

VCATALG, VSCATLG

Options, Titles:

VOPTION, VTITLE

Macros:

VMACRO

Data Sources:

VEXTFL, VINDEX, VSACCES, VSLIB

Each of the views listed above represents a stored SQL query that utilizes Dictionary tables. The actual query can be attained by using the DESCRIBE statement in the SQL procedure, as illustrated in the following example:

```
proc sql;
    describe view sashelp.vstable;
quit;
```

The DESCRIBE SQL command prints to the SAS log the contents of the stored query (the VSTABLE view) showing the actual SQL query that uses the MEMBERS Dictionary table, as shown below.

```
proc sql;
    select libname, memname
    from dictionary.members
    where memtype='DATA'
    order by libname asc, memname asc;
quit;
```

Using either the VSTABLE view or its respective query with Dictionary tables provides the same results, a listing of all active data sets in a SAS session. However,

Dictionary tables are confined to the SQL procedure; whereas the respective views can be used by other procedures or, even, in a Data Step.

The Process

The %xref macro accomplishes the task of producing a cross-reference report, which is shown in its entirety at the end of this paper. The %xref macro contains rather sophisticated techniques; hence, in order to explain this SAS solution, each step inside the macro will be discussed.

First, we need to know how many data sets are in the SAS data library. Using the TABLES Dictionary table, the following SQL procedure creates a macro variable &nt that denotes the number of tables in the data library. Then, the %let statement left-justifies the contents of the macro variable so that it resolves correctly when used in the next query. Even though the %left macro function is superfluous, it is self-documenting.

```
proc sql noprint;
    select count(memname) into :nt
    from dictionary.tables
    where libname eq "%upcase(&lib.)"
    and memtype eq "DATA";
quit;
```

```
%let nt = %left(&nt.);
```

Next, a similar SQL step creates a macro variable in order to identify each data set in the data library. The &nt macro variable created in the prior step completes the syntax needed for the INTO clause that creates the &tbl&i macro variables.

```
proc sql noprint;
    select memname into :tbl1 - :tbl&nt.
    from dictionary.tables
    where libname eq "%upcase(&lib.)"
    and memtype eq "DATA";
```

A subsequent SELECT statement creates macro variables that denote the number of observations in the respective data sets. In other words, the macro variable &nobs1 represents the number of observations in the data set &tbl1, respectively. Again, the macro variable &nt completes the syntax of the INTO clause, thereby creating the needed macro variables.

```
select put(nobs,comma10.)
into :nobs1 - :nobs&nt.
from dictionary.tables
where libname eq "%upcase(&lib.)"
and memtype eq "DATA";
quit;
```

The following portion of the %xref macro is a bit abstract. Ultimately, it creates a data set for each data set in the data library storing the variable identifiers and, especially, the data type of each variable. How? Well, the %do loop generates a query for the *i*th table by using the COLUMNS dictionary table in order to obtain the meta-data needed for each data set. By employing the SAS option MPRINT, prior to invoking the macro, you can

view the queries generated, which includes the resolved macro variables (i.e., `&&tbl&i` and `&&nobs&i`) that produce the label portion of the query.

Notice that the macro variable `&&tbl&i` stipulates several items:

- the data set being processed with respect to the Dictionary table,
- the variable containing the data type of itself
- the variable name stored in the LABEL of itself
- and, the resultant data set that contains the meta data for all the variables.

It looks slick, but the code introduces a nasty caveat, to be explained later.

```
proc sql noprint;
  %do i = 1 %to &nt.;
    create table &&tbl&i.. as
    select name, upcase(substr
      (type,1,1)) as &&tbl&i..
      label = "&&tbl&i..!(&&nobs&i..)"
      from dictionary.columns
      where libname eq "%upcase(&lib.)"
        and memname eq
          "%upcase(&&tbl&i..)"
      order by name;
  %end;
```

Thus far the `%xref` macro has determined the number of data sets in the data library, the name of each data set, and the name and data type of each variable in each data set. The next component simply creates a data set that uniquely identifies all the variables found in the data library. This data set serves as the base data set in the match merge. It's not needed for the merge, actually, but for the column that indicates the variables (attributes) in the report.

```
create table allattr as
select distinct name
from dictionary.columns
where libname eq "%upcase(&lib.)"
order by name;
quit;
```

The subsequent Data Step performs a match merge joining all the data sets needed to produce the cross-reference report. The ARRAY statement specifies the tables which allows the DO loop to assign a single dash for those instances when the element is missing, that is, the variable does not exist in that data set.

```
data rep;
  merge allattr
  %do i = 1 %to &nt.; &&tbl&i.. %end; ;
  by name;
  array tbls{*}$
  %do i = 1 %to &nt.; &&tbl&i.. %end; ;
  do i = 1 to dim(tbls);
    if tbls{i} eq ' '
      then tbls{i} = '-';
    end;
  run;
```

The final step, not surprisingly, produces the cross-reference using the Report procedure. The `%do` loop serves two purposes:

- enumerating the names of the variables in the COLUMN statement, denoting the names of the data sets in the data library; and,
- creating their respective DEFINE statements that includes a special use of the WIDTH option.

What if the number of observations in a data set, part of the variable label, requires more than the default width of eight bytes? Unless the WIDTH option stipulates otherwise, the text of the label is broken to fit the column width, thus, the report looks a bit ugly. In order to accommodate the real world of data analysis (where data sets contain thousands of observations), the `%xref` computes an adequate value for the WIDTH option of the Report procedure. How? By utilizing the `%length` and `%eval` Macro functions, along with appropriate logic. Thus, if the macro variable `&&nobs&i`, denoting the number of observations, contains a large (i.e., wide) number, including parentheses, the macro computes the needed width, accordingly, as shown below.

```
proc report data=rep headline headskip
  nowindows split='!';
  column name ('- Tables -'
  %do i = 1 %to &nt.; &&tbl&i.. %end;);
  define name / display width=10 'Attribute';
  %do i = 1 %to &nt.;
  define &&tbl&i. / display center width =
    %if %length(&&nobs&i..) gt
      %length(&&tbl&i..)
      %then %eval(%length(&&nobs&i..)+2);
      %else %eval(%length(&&tbl&i..)+2); ;
  title3 "Cross-Reference of Data Library";
  title4 "( %upcase(&lib.) )";
run;
```

An Enhancement

Notice that the `%xref` macro indicates only the existence of a variable in the several data sets by signifying its data type. Let's enhance the macro a bit by including the storage length of each variable. Doing this requires a modification of the SELECT statement in the appropriate query affecting the variable `&&tbl&i`. In short, the clause needs to include another piece of meta-data, that is, the LENGTH attribute, which is concatenated to the data type. Notice that the default limit of eight bytes is adequate length to store this information (e.g., C(200)).

```
proc sql noprint;
  %do i = 1 %to &nt.;
    create table &&tbl&i.. as
    select name, upcase(substr(type,1,1)) ||
      '(!|trim(left(put(length,3))) || ' )'
      as &&tbl&i..
      label = "&&tbl&i..!(&&nobs&i..)"
      from dictionary.columns
      where libname eq "%upcase(&lib.)"
        and memname eq "%upcase(&&tbl&i..)"
      order by name;
```

```
%end;
```

Another enhancement includes a wonderful, albeit undocumented, feature of the Report procedure. When processing large data library that contains numerous data sets and many variables, the Report procedure lists the item *name*, denoting attribute, only once in the report, thereby making it difficult to associate attributes with their respective tables, or vice versa. Using the ID option rather than the DISPLAY option of the DEFINE statement will list the item *name* on every page.

```
proc report data=rep headline headskip nowind;
  column name ('- Tables -'
    %do i = 1 %to &nt.; &&tbl&i.. %end;);
  define name / id width=10 'Attribute';
    :      :      :
quit;
```

A Dangerous Caveat

The `%xref` macro produces a cross-reference of any SAS data library. Well, almost. What happens when the `%xref` macro processes the WORK library? Answer: It annihilates the WORK library in a most peculiar way. It runs without error and produces a report, but it's wrong. In fact, the `%xref` replaces every data set with its respective meta-data version. Thus, of course, the `%xref` macro processes *only* permanent SAS data libraries. Finding where and why this happens poses a good exercise for the reader.

The %xref Macro

The `%xref` macro has only one positional parameter, *lib*, which denotes the SAS data library. It is shown below in its entirety, ready for use.

```
%macro xref(lib);
  options pageno=1;
  proc sql noprint;
    select count(memname) into :nt
      from dictionary.tables
      where libname eq "%upcase(&lib.)"
        and memtype eq "DATA";
  quit;
  %let nt = %left(&nt.);
  proc sql noprint;
    select memname into :tbl1 - :tbl&nt.
      from dictionary.tables
      where libname eq "%upcase(&lib.)"
        and memtype eq "DATA";
    select put(nobs,comma10.)
      into :nobs1 - :nobs&nt.
      from dictionary.tables
      where libname eq "%upcase(&lib.)"
        and memtype eq "DATA";
  quit;
  proc sql noprint;
  %do i = 1 %to &nt.;
    create table &&tbl&i.. as
    select name, upcase(substr(type,1,1)) ||
      ('||trim(left(put(length,3.))) || ')
      as &&tbl&i..
      label = "&&tbl&i..!(&nobs&i..)"
      from dictionary.columns
```

```
    where libname eq "%upcase(&lib.)"
      and memname eq "%upcase(&&tbl&i..)"
      order by name;
  %end;
  create table allattr as
  select distinct name
    from dictionary.columns
    where libname eq "%upcase(&lib.)"
      order by name;
  quit;
  data rep;
    merge allattr %do i = 1 %to &nt.;
      &&tbl&i.. %end; ;
    by name;
    array tbls{*}$ %do i = 1 %to &nt.;
      &&tbl&i.. %end; ;
    do i = 1 to dim(tbls);
      if tbls[i] eq ' '
        then tbls[i] = '-';
    end;
  run;
  proc report data=rep headline headskip
    nowindows split='!';
  column name ('- Tables -'
    %do i = 1 %to &nt.; &&tbl&i.. %end;);
  define name / id width=10 'Attribute';
  %do i = 1 %to &nt.;
    define &&tbl&i. / display center
      width = %if %length(&nobs&i..)
        gt %length(&&tbl&i..)
        %then %eval(%length(&nobs&i..)+2);
        %else %eval(%length(&&tbl&i..)+2); ;
    %end;
  title1 "Cross-Reference of Data Library";
  title2 "( %upcase(&lib.) )";
  run;
%mend xref;
```

Conclusion

SAS data libraries, consisting of several data sets that have common variables, often emulate a relational model. By employing meta-data made available through SAS Dictionary tables, it's possible to produce a concise report that offers useful information about a SAS data library.

The `%xref` macro represents a powerful tool that produces a report showing the organization and structure of a permanent SAS data library. Also, for those readers who wish to understand how that macro operates, the macro offers a good opportunity in learning more about meta-data and the Macro Language.

Author Information

John R. Gerlach
Maxim Group
Plymouth Meeting, PA
610.454.9967

Cindy Garra
IMS HEALTH
Plymouth Meeting, PA

SAS is a registered trademark of SAS Institute.

Acknowledgements

Thanks to Perry Watts, of IMS HEALTH, who suggested the idea of including the storage length in the Cross-reference report. And, thanks to SAS-L, where one learns even undocumented features of the SAS language.