PAPER BB-09

The SAS® Magical Dictionary Tour

Linda Libeg, Westat, Rockville, MD

ABSTRACT

There are many instances when a large number of variables in a SAS® dataset need to be renamed, ordered, dropped, or just identified. If variable names are not assigned in chronological or positional order, the task of programming these names into your code can be laborious. A simple Proc SQL statement can be used to access the read-only SAS data dictionary tables and save you programming steps.

This paper presents numerous ways you can modify a Proc SQL statement that reads the SAS data dictionary tables and produce macro output to be used in later programming steps. So, "Roll-up" to the SAS Magical Dictionary Tour and learn how to extract various characteristics from your SAS dataset.

INTRODUCTION

SAS data dictionary tables are read-only views that contain Metadata about SAS data available in the current SAS session. Information pertaining to SAS libraries, datasets, macros, external files, and indexes are just some of the information that can be captured from SAS dictionary tables and assist in simplifying program code. There are twenty-nine (29) dictionary tables in SAS v9.2. The names and descriptions of each are listed in Appendix A. All of these dictionary tables can also be referenced using views from the SASHELP library. A crosswalk between the dictionary table names and the SASHELP view names is listed in Appendix B. Although the SASHELP library is very useful, the primary purpose of this paper is to discuss the procedures to access the information through PROC SQL using the DICTIONARY libref.

VIEWING DICTIONARY METADATA

Using the SELECT statement within PROC SQL, the SAS Metadata can be retrieved. A list of the 29 tables can be identified, the member data within a specific table can be retrieved, a description of the table structure can be viewed, or a SAS dataset of the dictionary information can be created.

A PROC SQL statement can return the 29 dictionary table member names plus information relevant to each table.

```
proc sql;
   select * from dictionary.dictionaries;
quit;
```

A PROC SQL statement can return only the dictionary table member names of the 29 tables.

```
proc sql;
   select unique memname from dictionary.dictionaries;
quit;
```

The content of a specific dictionary table can be viewed using the same PROC SQL statement, by replacing "dictionaries" with a table name. For Instance, the PROC SQL statement below returns the member names from the dictionary table, "members".

```
proc sql;
   select memname from dictionary.members;
quit;
```

A PROC SQL statement and the CREATE TABLE statements can be used to create a SAS dataset containing the information stored in the dictionary tables. For example, here we create the SAS dataset, "dmembers", containing the member names from the dictionary table,"members".

```
proc sql;
   create table dmembers as
   select memname from dictionary.members;
quit;
```

The PROC SQL “DESCRIBE TABLE” command can be used to view the structure of a dictionary table. This listing is output to the .log file.

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

The real magic of dictionary tables is the power to manipulate a PROC SQL statement. Using a PROC SQL query and the SQL keywords SELECT, INTO, SEPARATED BY, FROM and WHERE, the SAS dictionary tables can be read and information can be extracted and placed into a macro variable. The format for the PROC SQL code is:

```
PROC SQL;
    Select (attribute)
    INTO :(macro variable name) SEPARTED BY ‘ ‘
    FROM DICTIONARY.(dictionary name)
    WHERE libname = (quoted library name in upper case) and
          Memname = (quoted member name in upper case);
Quit;
```

The remainder of this paper will focus mainly on using this PROC SQL statement to extract information from the “Columns” dictionary table. The paper will briefly show that the PROC SQL statement can be used with other dictionary tables. Many examples will illustrate how this magical code can be used to simplify a program.

DICTIONARY.COLUMNS

Dictionary columns contain variable information such as name, type, length, and label, for all SAS datasets in the current SAS session. Attributes for each variable in the dictionary.columns view can be identified using the DESCRIBE command. Below is the listing generated in a .log file when submitting the PROC SQL code. Also listed below is a table describing the information contained in each variable of the columns table.

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

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',
   npos num label='Column Position',
   varnum num label='Column Number in Table',
   label char(256) label='Column Label',
   format char(49) label='Column Format',
   informat char(49) label='Column Informat',
   idxusage char(9) label='Column Index Type',
   sortedby num label='Order in Key Sequence',
   xtype char(12) label='Extended Type',
   notnull char(3) label='Not NULL?',
   precision num label='Precision',
   scale num label='Scale',
   transcode char(3) label='Transcoded?'
  );
quit;
```

VARIABLE	DESCRIPTION
libname	Contains the library name which must be reference in upper case. Temporary SAS datasets are referenced using “WORK”.
memname	Contains the member name referenced in upper case for SAS. Other engines such as ACCESS and EXCEL preserve the name as case sensitive.
memtype	Contains the member type. [DATA or VIEW]
name	Contains the variable names in the dataset. The case of the name is stored as it is in the dataset

	with the exception of transport datasets that are stored in upper-case.
type	Contains the variable type in lower case. [“num” for numeric and “char” for character].
length	Contains the length of the variable.
npos	Contains the offset of the variable position within the observation. [0, 2, 4, 10…]
varnum	Contains the number in the table. [1, 2, 3…]
label	Contains the variable label.
format	Contains the format name assigned to the variable.
informat	Contains the informat name assigned to the variable.
idxusage	Contains the index type in upper case. [SIMPLE, COMPOSITE, BOTH]
sortedby	Contains the key sequence order using numeric values starting with 0. Descending sort order is referenced with negative numbers.
xtype	Contains the extended type.
notnull	Contains references as to whether or not Null values are allowed. [“no” and “yes”].
precision	Contains precision information.
scale	Contains scale information.
transcode	Contains references as to whether or not transcoding was done. [“yes” and “no”]

EXAMPLE 1

A macro variable named, “varname”, is created that stores the names of all variables in the SAS temporary dataset, MYDATA. Using the PROC SQL INTO and SEPARATED BY clauses, a variable can be created and stored in a macro variable for use in another programming step. The value in the “separated by” clause can contain values other than a blank.

```
proc sql noprint;
    Select name
    Into :varname separated by ‘ ‘
    From DICTIONARY.COLUMNS
    Where LIBNAME eq “WORK” and MEMNAME = “MYDATA”;
quit;
```

EXAMPLE 2

You can expand the PROC SQL statement in Example 1 to subset the names in the macro variable to only those names needed. The PROC SQL statement in example 2 subsets the names of numeric variables on the dataset, MYDATA, that begin with ‘MP’; and, will store the names in the macro variable, “varname”. The ‘TYPE’ attribute “num” subsets the names in the macro variable to only variables containing numeric data. If variables with character data were desired, then the ‘TYPE’ attribute would be changed to “char”. The type attribute must be reference in the PROC SQL statement in quotes and lower case. Once the list of macro variables has been created, the list can be used in a number of ways. For instance, the macro variable can hold a list of variable names to include in a PROC FREQ statement or in a DROP or KEEP statement.

```
proc sql noprint;
    Select name
    Into :varname separated by ‘ ‘
    From DICTIONARY.COLUMNS
    Where LIBNAME eq “WORK” and MEMNAME = “MYDATA”
    And substr(strip(name),1,2) = ‘MP’ and strip(type) = “num”;
quit;

proc freq data=MYDATA;
    Tables &varname./missing;
run;

data MYDATA;
    Set MYDATA(drop=&varname);
run;
```

EXAMPLE 3

An expression can be used in the SELECT statement and stored in the macro variable. This PROC SQL statement will create a macro variable containing an expression that renames the character variable names beginning with ‘MP’ and having a length greater than 2 to begin with ‘T_’. The macro variables “varname” can then be used in a data statement to rename a list of variables in the SAS dataset.

```
proc sql noprint;
    Select strip(name) || ‘ = T_’ || trim(name)
    Into :varname separated by ‘ ‘
    From DICTIONARY.COLUMNS
    Where LIBNAME eq “WORK” and MEMNAME = “MYDATA”
    And substr(strip(name),1,2) = ‘MP’ and strip(type) = “char” and length > 2;
quit;

data MYDATA;
    Set MYDATA(rename=(&varname.));
run;
```

EXAMPLE 4

The next example creates an expression that contains a statement to convert variables from numeric to character and places the expression in the macro variable, “varname”. The macro variable containing this expression can be used in the data statement to convert the specified variables to character.

```
proc sql noprint;
    Select ‘T_’ || strip(name) || ‘ = put(’ || strip(name) || ‘,2.);’
    Into :varname separated by ‘ ‘
    From DICTIONARY.COLUMNS
    Where LIBNAME eq “WORK” and MEMNAME = “MYDATA”
    And substr(strip(name),1,2) = ‘MP’ and strip(type) = “num”;
quit;

data MYDATA2;
    Set MYDATA;
    &varname
run;
```

EXAMPLE 5

The next example assigns an “if” expression to the macro variable, “varname”. This “if” statement reassigns a value of ‘.’ to blank for all character variables on the SAS dataset, MYDATA, that begin with ‘MP’. The expression in the macro variable can be used in a data statement.

```
proc sql noprint;
    Select ‘if ‘ || strip(name) || ‘ = “.” Then ‘ || strip(name) || ‘ = “ “;’
    Into :varname separated by ‘ ‘
    From DICTIONARY.COLUMNS
    Where LIBNAME eq “WORK” and MEMNAME = “MYDATA”
    And substr(strip(name),1,2) = ‘MP’ and strip(type) = “char”;
quit;

data MYDATA2;
    Set MYDATA;
    &varname
run;
```

EXAMPLE 6

The information within the dictionary columns table can be used to return a Boolean value indicating whether or not a variable exists on the file. If the variable, MAGIC, exists in the SAS dataset, MYDATA, then the macro variable, “varexist”, will contains a value of 1, otherwise the macro variable, “varexist”, will contain a value of 0.

```
%macro magic;
    proc sql noprint;
        Select strip(put(count(*),8.))
        Into :varexist
        From DICTIONARY.COLUMNS
        Where LIBNAME eq “WORK” and MEMNAME = “MYDATA”
        And upcase(NAME) = ‘MAGIC’;
    quit;
    %if &varexist %then %do;
        Proc print data=MYDATA;
            Var MAGIC;
        run;
    %end;
%mend;
%magic
```

EXAMPLE 7

Two macro variables can be created in the same PROC SQL statement. The count of variables is stored in “numvars” and all identified variable names are stored in the macro variable “varnames”. If the number of character variables on the dataset MYDATA is 5, then “numvars” would contain the number ‘5’ and ”varnames” would contain a list of these 5 variable names.

```
%macro magic;
    proc sql noprint;
        Select count(*), strip(name)
        Into :numvars, :varnames separated by ‘ ‘
        From DICTIONARY.COLUMNS
        Where LIBNAME eq “WORK” and MEMNAME = “MYDATA”
        And type = “char”;
    quit;

    %if &numvars > 0 %then %do;
        %put ‘numvars=’ &numvars ‘varname=’ &varnames;

        proc freq data=MYDATA;
            Tables &varnames./missing;
        run;
    %end;

%mend;
%magic
```

EXAMPLE 8

A PROC SQL statement can use the FORMAT attribute to identify a specific format type. In this example, the date variables are being retrieved from the SAS dataset and placed in the macro variable, “datevars”.

```
proc sql noprint;
    Select name
    Into :datevars separated by ‘ ‘
    From DICTIONARY.COLUMNS
    Where LIBNAME eq “WORK” and MEMNAME = “MYDATA”
    and  (upcase(FORMAT) CONTAINS ‘DATE’
    or    upcase(FORMAT) CONTAINS ‘MMDD’
    or    upcase(FORMAT) CONTAINS ‘YYMM’);
quit;

%put ‘datevars=’ &datevars;

proc freq data=MYDATA;
    tables &datevars./missing;
run;
```

EXAMPLE 9

The PROC SQL statement above can be modified to find the names of variables with formats and also the format name associated with each identified variable. The macro variable “varname” will contain a list of all variables that are formatted and the macro variable “varfmt” will contain the variable name concatenated with the format name.

```
proc sql noprint;
    Select name, name || ‘ ‘ || format
    Into :varname separated by ‘ ‘, :varfmt separated by ‘ ‘
    From DICTIONARY.COLUMNS
    Where LIBNAME eq “WORK” and MEMNAME = “MYDATA”
    And FORMAT ^= “ “;
quit;

%put ‘varname=’ &varname ‘varfmt=’ &varfmt;

Proc freq data=MYDATA;
    tables &varname./missing;
    Format &varfmt;
run;
```

EXAMPLE 10

The next example uses parameters when calling the macro, Magic. A parameter for the dataset name and a second parameter for the "by" variable are sent in the macro call. The PROC SQL statement uses resolved values of these two parameters to assign the type ("char" or "num") of the 'by' variable to the macro variable, "byvartype". The macro variable can then be used in another SAS procedure. This example uses the type assignment in the macro variable to exclude the missing data for the specified "by" variable in a PROC FREQ statement.

In the first call to this macro, frequencies on the dataset are to be produced by GENDER. Assuming GENDER has the values 'M' for male and 'F' for female, the macro variable, "byvartype" will be assigned the value "char" by the PROC SQL statement. When the PROC FREQ statement is executed, the condition, 'where &byvar ne " "' is resolved.

In the second call to this macro, frequencies on the dataset are to be produced by REGION. Assuming that REGION contains the numeric values 1-4, the macro variable, "byvartype" will be assigned the value "num" by the PROC SQL statement. When the PROC FREQ statement is executed, the condition, 'where &byvar ne .' is resolved.

```
%macro magic(dsname,byvar);
    proc sql noprint;
        Select type
        Into :byvartype
        From DICTIONARY.COLUMNS
        Where LIBNAME eq “WORK” and MEMNAME = %upcase(“&dsname.”) and name =
                  %upcase(“&byvar.”);
    quit;

    proc sort data=&dsname;
        by &byvar;
    run;

    proc freq data=&dsname;
        by &byvar;
        %if &byvartype = char %then where &byvar ne “ “;
        %else where &byvar ne .;
        ;
        Tables _all_/missing;
    run;
%mend;
%magic(MYDATA,GENDER)
%magic(MYDATA,REGION)
```

EXAMPLE 11

The following example creates sequential macro variables containing the variables names in the SAS dataset. Although statement allows up to 9999 macro variables, macro variables are only created for the number of variables identified in the SAS dataset.

```
%macro Magic;

    proc sql noprint;
        Select name
```

```
          Into :var1 - :var9999
          From DICTIONARY.COLUMNS
          Where LIBNAME eq "WORK" and MEMNAME = "MYDATA";
      quit;

      proc sql noprint;
          Select count(*)
          Into :varcnt
          From DICTIONARY.COLUMNS
          Where LIBNAME eq "WORK" and MEMNAME = "MYDATA";
      quit;

      %do i = 1 %to &varcnt;
          %put "var&i=" &&var&i;
      %end;
  %mend;
  %magic
```

EXAMPLE 12

As mentioned in the "Viewing Dictionary Metadata" section, a SAS dataset can also be created that contains information extracted from a dictionary table. This example creates the SAS dataset "vartable" that contains the names of all variables in the SAS dataset, MYDATA.

```
  proc sql noprint;
      Create table vartable as
      Select name From DICTIONARY.COLUMNS
      Where LIBNAME eq "WORK" and MEMNAME = "MYDATA";
  quit;

  Proc print data=vartable;
      Title 'Listing of Variable Names in MYDATA File';
  run;
```

DICTIONARY.TABLES

Information can also be obtained from SAS dictionaries other than the column table. Dictionary tables contain information such as the library name, the member type, the creation date, the number of observations, and the number of variables. The information is available for every SAS dataset present in the current SAS session.

The PROC SQL "DESCRIBE TABLE" command can be used to view the structure of a dictionary table. This listing is output to the .log listing.

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

create table DICTIONARY.TABLES
  (
   libname char(8) label='Library Name',
   memname char(32) label='Member Name',
   memtype char(8) label='Member Type',
   dbms_memtype char(32) label='DBMS Member Type',
   memlabel char(256) label='Data Set Label',
   typemem char(8) label='Data Set Type',
   crdate num format=DATETIME informat=DATETIME label='Date Created',
   modate num format=DATETIME informat=DATETIME label='Date Modified',
   nobs num label='Number of Physical Observations',
   obslen num label='Observation Length',
   nvar num label='Number of Variables',
   protect char(3) label='Type of Password Protection',
   compress char(8) label='Compression Routine',
   encrypt char(8) label='Encryption',
   npage num label='Number of Pages',
   filesize num label='Size of File',
   pcompress num label='Percent Compression',
   reuse char(3) label='Reuse Space',
   bufsize num label='Bufsize',
   delobs num label='Number of Deleted Observations',
   nlobs num label='Number of Logical Observations',
   maxvar num label='Longest variable name',
   maxlabel num label='Longest label',
   maxgen num label='Maximum number of generations',
   gen num label='Generation number',
   attr char(3) label='Data Set Attributes',
   indxtype char(9) label='Type of Indexes',
   datarep char(32) label='Data Representation',
   sortname char(8) label='Name of Collating Sequence',
   sorttype char(4) label='Sorting Type',
   sortchar char(8) label='Charset Sorted By',
   reqvector char(24) format=$HEX48 informat=$HEX48 label='Requirements Vector',
   datarepname char(170) label='Data Representation Name',
   encoding char(256) label='Data Encoding',
   audit char(3) label='Audit Trail Active?',
   audit_before char(3) label='Audit Before Image?',
   audit_admin char(3) label='Audit Admin Image?',
   audit_error char(3) label='Audit Error Image?',
   audit_data char(3) label='Audit Data Image?',
   num_character num label='Number of Character Variables',
   num_numeric num label='Number of Numeric Variables'
  )
```

EXAMPLE 1

The number of observations in a dataset can be retrieved by selecting the “nobs” attribute from the Dictionary.tables. In this example, the macro variable, ”nobs”, is created that stores the number of observations in the SAS dataset, MYDATA.

```
proc sql noprint;
    Select strip(put(nobs,8.))
    into :nobs
    From DICTIONARY.TABLES
    Where LIBNAME = ‘WORK’ and MEMNAME = ‘MYDATA’;
quit;

%put &nobs;
```

EXAMPLE 2

The number of variables is a dataset can be retrieved by selecting the “nvar” attribute from the Dictionary.tables. In this example, the macro variable, “nvar”, is created that stores the number of variables in the SAS dataset, MYDATA.

```
proc sql noprint;
    Select strip(put(nvar,8.))
    into :nvar
    From DICTIONARY.TABLES
    Where LIBNAME = ‘WORK’ and MEMNAME = ‘MYDATA’;
quit;

%put &nvar;
```

EXAMPLE 3

The dataset creation date can be retrieved by selecting the “crdate” attribute from Dictionary.tables. The macro variable, “vardate”, is created that stores the creation date of the SAS dataset, MYDATA. The date value is returned in the format, “11JUL11:13:23:49”.

```
proc sql noprint;
    Select crdate
    into :vardate
    From DICTIONARY.TABLES
    Where LIBNAME = ‘WORK’ and MEMNAME = ‘MYDATA’;
quit;

%put &vardate;
```

EXAMPLE 4

This example creates a macro variable named, “vardate”, which contains the names of all files available in the current SAS session created in 2011.

```
Proc sql noprint;
    Select MEMNAME
    Into :vardate separated by ‘ ‘
    From DICTIONARY.TABLES
    Where YEAR(DATEPART(CRDATE))EQ 2011;
Quit;

&put  &vardate;
```

DICTIONARY.MEMBERS

Useful information can also be obtained from the Members table, which contains information about member types such as tables, views and catalogs.

The PROC SQL “DESCRIBE MEMBERS” command can be used to view the structure of a dictionary member just as it was used to describe the structure of the dictionary.columns table and the dictionary.tables table.

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

create table DICTIONARY.MEMBERS
  (
   libname char(8) label='Library Name',
   memname char(32) label='Member Name',
   memtype char(8) label='Member Type',
   dbms_memtype char(32) label='DBMS Member Type',
   engine char(8) label='Engine Name',
   index char(3) label='Indexes',
   path char(1024) label='Pathname'
  );
```

EXAMPLE 1

Similar to dictionary.tables, the members table can also return the member names contained within a library. The macro variable, “memname”, is created and stores the member names identified in the library “WORK”.

```
proc sql noprint;
    Select memname
    into :memname separated by ‘ ‘
    From DICTIONARY.MEMBERS
    Where LIBNAME = ‘WORK’;
quit;

%put member names: &memname;
```

EXAMPLE 2

In the example below, the “path” attribute of the dictionary members table will be extracted. The paths of all datasets available in the SAS Session will be assigned to the macro variable “mempath”.

```
proc sql noprint;
    Select path
    into :mempath separated by ‘ ‘
    From DICTIONARY.MEMBERS
    Where LIBNAME = 'WORK';
quit;

%put member path: &mempath;
```

CONCLUSION

The use of Dictionary tables can perform magic in a SAS environment. Along with a SAS macro statement and PROC SQL, valuable metadata information can be retrieved from dictionary tables. Custom made lists and expressions that enable new variables to be assigned or variables to be renamed can be created and used in a SAS program. This paper is only an introduction to SAS Dictionary tables; and, there is a wealth of additional information available in SAS dictionary tables. Hopefully, this paper will inspire SAS programmers to investigate other features of SAS dictionary tables that will enhance their SAS programs.

REFERENCES

Dilorio, Frank & Abolafia, Jeff (2004). “Dictionary Tables and Views: Essential Tools for Serious Applications” SAS Conference Proceedings: SUGI 29, Paper 237-29.

Eberhardt, Peter. “How Do I Look it Up if I cannot Spell It: An Introduction to SAS® Dictionary Tables”. SAS Global Forum 2007, Paper 235-2007.

Varney, Brian. “Using Metadata and Project Data for Data-Driven Programming”. SAS Conference Proceedings: SUGI 31, Paper 045-31.

Zeng, Xu. “Variable Names Don’t Begin with the Same Characters? No Problem: How to Create Variable List Without Copying, Pasting or Excel Intervention”. SAS Global Forum 2009, Paper 053-2009.

SAS Support http://support.sas.com

ACKNOWLEDGMENTS

I would like to thank Michael Raithel and Mike Rhoads for their guidance in helping me to prepare and present this paper.

RECOMMENDED READING

A paper by Frank Dilorio has excellent examples of reading the dictionary.macros table and applying the extracted information.

Dilorio, Frank. “%whatChanged: A tool for the Well-Behaved Macro” SAS Conference Proceedings: NESUG 2010, Programming Beyond the Basics.

DISCLAIMER

The contents of this paper are the work of the author(s) and do not necessarily represent the opinions, recommendations, or practices of Westat.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Linda Libeg
Email: LindaLibeg@westat.com

TRADEMARKS

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are trademarks of their respective companies.

APPENDIX A: SAS DICTIONARY TABLE NAMES AND DESCRIPTIONS

Table in SAS 9.2	Description
CATALOGS	Contains SAS Session Catalog information
CHECK_CONSTRAINTS	Contains SAS Session Check Constraint information
COLUMNS	Contains SAS Session Table Information
CONSTRAINT_COLUMN_USAGE	Contains column information referenced by integrity constraints
CONSTRAINT_TABLE_USAGE	Contains table information for tables with defined integrity constraints
DATAITEMS	Contains information about known data items
DESTINATIONS	Contains information about known ODS destinations
DICTIONARIES	Contains information on DICTIONARY tables and their columns
ENGINES	Contains information on available SAS engines
EXTFILES	Contains External Files information
FILTERS	Contains information about known filters
FORMATS	Contains information on available formats
FUNCTIONS	Contains information about known functions
GOPTIONS	Contains information on SAS/Graph options
INDEXES	Contains SAS Session Index information
INFOMAPS	Contains information about information maps
LIBNAMES	Contains LIBNAME information
MACROS	Contains SAS Session Macro information
MEMBERS	Contains information about objects currently defined in SAS libraries
OPTIONS	Contains Current Session options
PROMPTS	Contains information about all known SAS/GRAPH prompts
PROMPTSXML	Contains information about all know XML prompts
REFERENTIAL_CONSTRAINTS	Contains information on Referential constraints
REMEMBER	Contains information about remembered text
STYLES	Contains ODS Styles
TABLE_CONSTRAINTS	Contains information on Table constraints
TABLES	Contains information about tables and datasets
TITLES	Contains Titles and Footnote information
VIEWS	Contains SAS Session information about Views

APPENDIX B: SAS DICTIONARY TABLES AND SASHELP VIEWS CROSSWALK

Tables in SAS 9.2	SASHELP VIEW
CATALOGS	VCATALG
CHECK_CONSTRAINTS	VCHKCON
COLUMNS	VCOLUMN
CONSTRAINT_COLUMN_USAGE	VCNCOLU
CONSTRAINT_TABLE_USAGE	VCNTABU
DATAITEMS	VDATAIT
DESTINATIONS	VDEST
DICTIONARIES	VDCTNRY
ENGINES	VENGINE
EXTFILES	VEXTFL
FILTERS	VFILTER
FORMATS	VFORMAT
FUNCTIONS	VFUNC
GOPTIONS	VGOPT
INDEXES	VINDEX
INFOMAPS	VINFOMP
LIBNAMES	VLIBNAM
MACROS	VMACRO
MEMBERS	VMEMBER
OPTIONS	VOPTION
PROMPTS	VPROMPT
PROMPTSXML	VPRMXML
REFERENTIAL_CONSTRAINTS	VREFCON
REMEMBER	VREMEMB
STYLES	VSTYLE
TABLE_CONSTRAINTS	VTABLE
TABLES	VTABCON
TITLES	VTITLE
VIEWS	VVIEW