

The Wealth of Information Found in the SASHELP Data Dictionary Views and how to Use It

Abstract

The SAS® data dictionary views that can be found in the SASHELP library provide a wealth of information that programmers can use to create code that reacts to its environment and the data available. This paper will give a description of those views and give several real world examples of code that makes use of this information.

Introduction to the Views

Those of us who work with relational database have become accustomed to using the data dictionary in those systems to obtain a wealth of information. SAS has a similar set of views but many SAS® programmers are not aware that they exist, or have not learned how they can greatly improve their programming. These views can be accessed two different ways. Within PROC SQL they can be found in a system defined library named "DICTIONARY". Outside of PROC SQL they are found in a much unexpected place, the SASHELP library along with the tutorial samples. To make things a little more confusing, the views in SASHELP have different names and sometimes different content than those found in the DICTIONARY library. The information in these views can be extremely useful, particularly when writing automated or data driven code. As we will explore later in this paper, they contain information about almost all aspects of the SAS environment, and the data storage objects that have been defined.

There are however a few down sides. Notice that these are called views and not tables. Their content is generated on-the-fly as the query is performed. This gives results that are up to the minute, but it makes for slow performance. In fact that performance can be considered very slow, especially if you have several libraries defined that contain many datasets. For this reason there are frequently two views for each purpose. One contains all of the available information. The other is a redacted view that runs more quickly.

The other noticeable issue is the severe lack of documentation. There are a few papers on the older of these views, but there is literally nothing about the newer. A search on support.sas.com for the name "VCHKCON", one of the newer views on relational integrity, returns only one hit. That hit is just the results of a sample program that returns a list of all of the tables in the DICTIONARY library! Hopefully this paper will help to alleviate this problem a bit. What follows breaks these views into a few different categories. These categories are creations of mine to aid in organizing this paper, and do not represent any construct in SAS. In each category the available tables will be listed along with a brief description of what they contain. In some cases an example will be given on how some of these views can be used.

The Fundamental Data Sources Views

The first set of views we will discuss are those that describe the defined libraries and the datasets they contain.

SASHELP View Name	SQL DICTIONARY View Name	Function
VCOLUMN	COLUMNS	Observation for every variable available in the session.
VTABLE	TABLES	Information for every SAS data set available in the session. Does not include data views.
VSLIB	LIBNAMES	Lists library names and the operating system locations they are assigned to.
VLIBNAM	LIBNAMES	Lists library names and the operating system locations they are assigned to.
VSTABLE	MEMBERS where memtype='DATA'	Lists data set names and the libraries they are located in. Smaller than VTABLE so faster.
VEXTFL	EXFILES	The file references assigned in the SAS session.
VVIEW	VIEWS	Lists the name and engine types of all views available in the session.
VSVIEW	MEMBERS where memtype='VIEW'	Includes the name and location of each view but not the type.

These views have been part of SAS for quite a while and are the best documented. Their uses are extensive. Unfortunately they will not work with all of the SAS Access engines. In my experience they will work on libraries connected to Microsoft® Access databases, but not on MySQL® databases.

The following is an example using the VCOLUMN view to generate a rename statement.

```

%macro ReadSF4;
  data SF4PCT142Hisp;
    set PovertySF4(rename=(%let dsid = %sysfunc(open(sashelp.vcolumn
                                (where =(libname = 'WORK' and
                                    memname = 'POVERTYSF4' and
                                    name like 'PCT%')),IS));
                                %if &dsid ne 0 %then
                                %do %while(%sysfunc(fetch(&dsid)) = 0);
                                  %let ColumnName= %sysfunc(getvarc(&dsid,
                                                                    %sysfunc(varnum(&dsid,NAME))));
                                  &ColumnName = &ColumnName.H
                                %end;
                                %else
                                  %put %STR(ERROR: Could not find columns in VCOLUMN;
                                %let rc = %sysfunc(close(&dsid));
                                ));
  run;
%mend;
%ReadSF4;

```

The result of running this program is a copy of the original PovertySF4 dataset with all of the columns whose name starts with "PCT" renamed by appending an H to the name. It was needed because I had two Census tables with hundreds of identical column names which I needed to merge without the columns interacting. By using this view, I avoided writing pages of code that could be subject to errors. You can see the use of the %SYSFUNC macro function to call the OPEN and FETCH IO functions. This is a very powerful way to utilize the data dictionary

information as it allows you to read a dataset or view without executing a procedure or dataset. Before the introduction of %SYSFUNC the uses of these views was much more cumbersome.

Another less obviously practical sample is the following macro:

```
%macro CopyTable(SourceLibname,SourceMemname,DestinationLibname,DestinationMemname);
proc sql noprint;
  select strip(Name) || " " || Type || '(' || strip(put(length,4.)) || ')' ||
    case label when ' ' then '' else ' label=' ||quote(strip(label)) end ||
    case format when ' ' then '' else ' format=' || strip(format) end ||
    case informat when ' ' then '' else ' informat=' || strip(informat) end
  into :CreateColumns separated by ','
  from dictionary.columns /*sashelp.vcolumn*/
  where memname = upcase("&SourceMemname")
    and libname = upcase("&SourceLibname")
    and memtype = "DATA";

  create table &DestinationLibname.&DestinationMemname(&CreateColumns);
quit;

%mend;
%CopyTable(SASHELP,CITIWK,work,junk);
```

This macro creates an empty copy of an existing dataset. It can put that dataset in a different library and can change its name. Notice that since it runs within PROC SQL, the DICTIONARY library was available and the COLUMNS view was used. The SASHELP library is also available within PROC SQL, so I will typically use it instead of DICTIONARY. This makes things a little easier to remember. Whereas this example could be replicated more easily using a DATA step or PROC COPY it is not terribly useful. The capabilities it represents are however powerful. Using this basic logic, one could create a program that would generate all of the create table and insert statements needed to recreate an existing table, or even an entire library. I have used it to make Oracle® copies of SAS tables while keeping control of the variable data types used.

The SAS Environment Views

These views have do not describe the datasets but instead the options, settings, and components of the SAS system environment itself.

SASHELP View Name	SQL DICTIONARY View Name	Function
VMACRO	MACROS	Names and values of all macro variables available to the session.
VOPTION	OPTIONS	Includes the name, description, current setting and scope for each SAS option except GOPTIONS.
VGOPT	GOPTIONS	Includes the name, description, current setting and scope for each SAS.
VALLOT	OPTIONS	List of all SAS options and their current setting
VTITLE	TITLES	Gives the location (type), number and value of each title and footnote currently available.
VSTYLE	STYLES	List of Available Styles
VENGINE	ENGINE	List of the available SAS ACCESS Engines (Not licensed Or Installed)

These can be particularly useful when writing macros that are designed to be reused in different programs. The current values of the macro variables and system options can be checked and changed as needed. Their values can also be stored for use at a later time, or printed for process documentation. The following example gives some ideas how the VMACRO view can be used:

```
%let TestMacroVar = This is a Test;
data CurrentMacros;
  set sashelp.VMACRO;
run;
%symdel TestMacroVar;

data _null_;
  set CurrentMacros(where = (scope ne 'AUTOMATIC'));
  call symput(name,value);
run;
%put TestMacroVar = & TestMacroVar;
```

This example sets a test macro variable and then stores all of the currently defined macro variables in a dataset. The test macro variable is then deleted. All of the macro variables are then re-established to their original values using the stored information. This is done in a DATA _NULL_ step using the CALL SYMPUT function. The last PUT statement will display the original value of the test macro variable, "This is a Test". Notice that only the variables that had a scope other than 'AUTOMATIC' were read into the data vector. This is because the automatic variables are controlled by SAS and cannot be changed. Without getting into a detailed discussion of macro variable scope, there are three scopes possible in this table, 'AUTOMATIC', 'GLOBAL', or some local scope. The variable we created in this example is a global variable. If it had been created inside a user defined macro procedure, it would be local. This is of course unless the code specifically declared as global using %GLOBAL. Local variables will have the name of the macro statement in which they are declared in the SCOPE variable.

The next example is a set of useful macro procedures. They act to completely turn off all printing of notes to the log, and then restore the settings to what they were originally. It comes in handy if you have a macro procedure that is used frequently and which clutters up the log unnecessarily:

```
%macro StartSilentMode;
  %if %symexist(_OriginalLogOptions) %then %do;
    options PRINTMSGLIST;
    %put WARNING: StartSilentMode was called while Silent Mode was already in;
    %put WARNING: effect. No action will be taken. The options stored;
    %put WARNING: from the previous run will continue to be stored.;
    %put WARNING: This prevents the loss of the earliest option settings.;
    options NOPRINTMSGLIST;
  %end;
  %else %do;
    %local BooleanTypeOptions;
    %global _OriginalLogOptions;
    proc sql noprint;
      select setting
        into :BooleanTypeOptions separated by ' '
        from sashelp.VOPTION
        where optname in('CMDMAC','IMPLMAC','MAUTOLOCDISPLAY','MEXECNOTE',
                        'MFILE','MLOGIC','MLOGICNEST','MPRINT','MPRINTNEST',
                        'NOTES','PRINTMSGLIST','SOURCE','SOURCE2','SYMBOLGEN',
                        'FULLSTIMER','STIMER');
    quit;
  %end;
end;
```

```

%let _OriginalLogOptions = OPTIONS &BooleanTypeOptions;
%put WARNING: Entering Silent Mode. No information except errors will be;
%put WARNING: displayed to the log.;
OPTIONS NOCMDMAC NOIMPLMAC NOMAUTOLOCDISPLAY NOMEXECNOTE NOMFILE NOMLOGIC
        NOMLOGICNEST NOMPRINT NOMPRINTNEST NONOTES NOPRINTMSGSLIST
        NOSOURCE NOSOURCE2 NOSYMBOLGEN NOFULLSTIMER NOSTIMER

%end;
%mend;
%macro EndSilentMode;
    &_OriginalLogOptions;
    %syndel _OriginalLogOptions;
    %put WARNING: Ending Silent Mode;
%mend;

%StartSilentMode;
DATA test; do x = 1 to 100; output; end; run;
%EndSilentMode;
DATA test; do x = 1 to 100; output; end; run;

```

The last four lines are a demonstration of the macro procedures being used. The first DATA step will write nothing to the log unless an error is encountered. The second will right to the log normally. The StartSilentMode macro accomplishes this by reading the VOPTION view to determine the current setting of the SAS options that control output to the log. Using this it generates a global macro variable that contains a SAS OPTIONS statement that reflects the current settings of these options. All of these options are then set so that output to the log is suppressed. The EndSilentMode executes that stored OPTIONS statement to restore these options to their original state. Notice that this code mentions Boolean type SAS options. There is a variable in this view named OPTTYPE which has one of three values, "char", "num", or "boolean". If the option is type "char" or "num" the option is set using a value such as "[OPTIONS PAGESIZE = 56;](#)". If it is Boolean, it is a toggle on or off such as "[OPTIONS MPRINT;](#)" or "[OPTIONS NOMPRINT;](#)". Those that are set to a value will have the current value listed in the SETTING variable of this view. The Boolean options however have the actual option statement in that variable. (eg. "[MPRINT](#)" or "[NOMPRINT](#)"). By chance, this example only modified options that were of type Boolean, so only that type was coded for.

The Relational Integrity and Indexing Views

There are a bunch of views many of which are relatively new to SAS and are very under documented. They describe the relational integrity and indexing of the datasets.

SASHELP View Name	SQL DICTIONARY View Name	Function
VINDEX	INDEXES	The indexes available in the session. Contains column Information.
VCHKCON	CHECK_CONSTRAINTS	List of Check Constraints by Name with rules
VCNCOLU	CONSTRAINT_COLUMN_USAGE	List of Assignments of Constraints to Columns
VCNTABU	CONSTRAINT_TABLE_USAGE	List of Assignments of Constraints to Tables. Smaller version of VSTABCON
VREFCON	REFERENTIAL_CONSTRAINTS	List of the Referential Constraints
VSTABCON	CONSTRAINT_TABLE_USAGE	List of Assignments of Constraints to Tables. Larger version of VCNTABU

One use of these views would be to improve the functionality of the earlier example procedure that recreates a table. Using these views, the indexes and relational constraints could also be

recreated. The VINDEX view has been around for quite a while, and is very useful. When creating an index on a table, you will get an error if that index already exists. In batch processing this can cause the entire job to go into syntax checking mode. In any case it produces an error to the log which can be irritating at best. The next example is a macro procedure that when called checks to see if an index exists on the desired columns. If one does its name is returned in a global macro variable and a note is written to the log. If it doesn't, the index is created and the name of the new index is returned.

```
%macro CreateIndex(TableName,ByVars,IndexName);
  %local VindexDsid VindexIndexNameVarId VindexVarNameVarId PreviousIndex
    IndexPosition CurrentIndex IndexMatches MemberName LibraryName
    MatchingIndex;
  %let MemberName = %upcase(%scan(&TableName,-1,%str()));
  %let LibraryName = %upcase(%scan(&TableName,-2,%str()));
  %if &LibraryName = %then %let LibraryName = WORK;
  %let VindexDsid =
    %sysfunc(open(SASHELP.VINDEX(where = (UPCASE(memname) = "&MemberName" and
      UPCASE(libname) = "&LibraryName")),
      IS));
  %if &VindexDsid ne 0 %then %do;
    %let VindexIndexNameVarId = %sysfunc(varnum(&VindexDsid,indxname));
    %let VindexVarNameVarId = %sysfunc(varnum(&VindexDsid,name));
    %let PreviousIndex = ;
    %do %while(%sysfunc(fetch(&VindexDsid)) = 0);
      %let CurrentIndex =
        %upcase(%sysfunc(getvarc(&VindexDsid,&VindexIndexNameVarId)));
      %if &CurrentIndex = &PreviousIndex %then %do;
        %let IndexPosition = %eval( &IndexPosition + 1 );
      %end;
      %else %do;
        %if &IndexMatches = 1 %then %let MatchingIndex = &PreviousIndex;
        %let IndexPosition = 1;
        %let IndexMatches = 1;
      %end;
      %if %upcase(%scan(&ByVars,&IndexPosition)) ne
        %upcase(%sysfunc(getvarc(&VindexDsid,&VindexVarNameVarId))) and
        %scan(&ByVars,&IndexPosition) ne %str() %then
        %let IndexMatches = 0;
      %let PreviousIndex = &CurrentIndex;
    %end;
    %let rc = %sysfunc(close(&VindexDsid));
    %if &IndexMatches = 1 and %scan(&ByVars,%eval(&IndexPosition + 1)) = %str() %then
      %let MatchingIndex = &PreviousIndex;
    %if &MatchingIndex = %str() %then %do;
      %if &IndexName = %then
        %let IndexName = %sysfunc(subpad(%sysfunc(compress(&ByVars)),1,31));
      proc sql;
        create index &IndexName on
          &TableName(%sysfunc(translate(%sysfunc(strip(&ByVars)),%str(),%str( ))));
      quit;
      %let MatchingIndex = &CurrentIndex;
      %put NOTE: Index &IndexName created on table &TableName for (&ByVars);
    %end;
    %else %do;
      %put NOTE: Existing Index &MatchingIndex already indexes &TableName (&ByVars);
    %end;
  %end;
%else %put %str(ER)ROR: Could not open VINDEX;
```

```

%mend;

* Demonstrate the functionality;
data test;
  do x = 1 to 40;
    y = ranuni(3423);
    output;
  end;
run;
%CreateIndex(test,x y)
%CreateIndex(test,x)

```

The demonstration creates a test dataset. The first call to CreateIndex generates a new index named "XY" on the TEST dataset on the "X" and "Y" variables. The second call to CreateIndex returns a note saying "Existing Index XY already exists on table test for (x)".

This macro works by opening a cursor on the VINDEXT view sub setting for indexes on the passed table. This view contains a record for every variable in the index and would be better named VINDEXCOL, but it isn't. The cursor returns a list of indexes and their indexed variables. Fortunately, this view returns records in order of INDEXNAME which identifies the index name and INDEXPOS which designates the order in which variables in a compound index are indexed. The program need only identify any index whose variables match those passed in the order they were passed. If the existing index contains the correct variables and some extras later in the sequence, that index will serve the purpose and is also considered a match as we see in this demonstration. If a match is found the value of the global macro variable MatchingIndex is set to the name of the index. If not, an index name is generated by appending the index variable names, and trimming the results to the maximum index name length of 31. A CREATE INDEX SQL statement is then generated and executed. The index name is generated in this fashion because single variable index names must match the name of the variable being indexed. This code has a small flaw in that it can not handle multiple compound indexes whose first 31 characters match. Code can be written to handle this situation, but it complicated the logic to much to be used as an example.

The Other Data Sources Views

The data source views discussed earlier are only a subset of those available. There are views for the currently defined formats. There are also view for the other possible library member types Catalogs, and Access.

SASHELP View Name	SQL DICTIONARY View Name	Function
VFORMAT	FORMAT	List of the format names. More Detailed version of VFORMAT
VCFORMAT		List of the format names. Smaller version of VFORMAT
VMEMBER	MEMBERS	Lists all data sets, catalogs, views, and multidimensional databases available in the session.
VSACCES	MEMBERS where memtype='ACCESS'	Lists SAS/ACCESS views available in the session.
VSTABVW	MEMBERS where memtype='VIEW' or memtype='DATA'	Includes both data sets and views. Has fewer variables than VTABLE so it is faster.
VCATALOG	CATALOGS	Information for each SAS catalog.
VSCATLG	MEMBERS where memtype='CATALOG'	Quick list of catalogs and the libraries they are located in. Smaller than VCATLG so faster.

The VFORMAT view could be used to test if a format has been defined. It could also be used to generate a list of available formats for an AF application.

The Miscellaneous Views

These last two views didn't quite fit into the other categories.

SASHELP View Name	SQL DICTIONARY View Name	Function
VDCTNRY	DICTIONARIES	DICTIONARY tables and their columns
VREMEMB	REMEMBER	Remembered Information

The VDCTNRY view contains a list of the columns in the SQL Dictionary views. The VREMEMB view is a mystery. I found no documentation it, nor could I find much that referred to "Remembered Information" in the SAS documentation. Every time I have looked at this view it has been empty. So, if there is anybody out there who knows what this might contain, please contact me. Maybe you should have written this paper!

Tips for Using These Views

As I mentioned at the start, these view can be slow. One way to mitigate this is to close any libraries you do not need using the "`LIBNAME MyLibraryName CLEAR;`" statement. Another way to speed performance is to use the smallest of the views that meet your needs. There are frequently two views for the same information, one that is complete, and one that is redacted for performance.

Getting comfortable with %SYSFUNC and the OPEN statements will make use of these tables much less daunting. After a while it will become easier to write a macro that reads all of the variables in a dataset than it would be to type them all out by hand. This also makes the code more forgiving when variables are added or removed.

A good source of additional information on the data dictionary views can be found at: <http://www2.sas.com/proceedings/sugi29/249-29.pdf>

Conclusion

The data dictionary views are a tool that SAS programmers should know more about in order to create more robust and data sensitive code. A little time invested in learning their uses will be well worth the effort.

Biography

Curtis currently works for Looking Glass Analytics as a SAS consultant and GIS service manager. Prior to that, he worked for the Washington State Department of Social & Health Services Division of Research and Data Analysis, and the US Census Bureau. He has worked extensively with SAS for fifteen years. He has expertise in Geographic Information Systems, database design/programming, and demographic analysis. He holds a bachelors degree in geography from the University of Washington.

Curtis Mack
Looking Glass Analytics
Curtis.Mack@lgan.com
www.LGAN.com

Acknowledgements

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