PhUSE 2016

Paper CS05

Dictionary tables- a programmer's friend?

Marcin Pracz, Quintiles, Warsaw, Poland

ABSTRACT

Dictionary tables are read-only SAS views that contain a lot of information about SAS libraries, datasets, macros, formats and other features available in the current SAS session. For this reason, they are often referred to when preparing a drug submission or in any situation, where the aim is to develop a generic program that could search through or make changes to many datasets at once. Although widely used, in my opinion these views have much unrealized potential. For this reason, in this paper I will provide a short introduction about dictionary tables and will later proceed to show how they can be used to check for data issues in raw data transfers.

INTRODUCTION

There are 32 dictionary tables available in SAS version 9.4 (which means there are 3 new tables compared with SAS®9.2). Probably the easiest way to get a feeling of what they contain is by searching through a DICTIONARY.DICTIONARIES table- a kind of meta-table that contains information on other dictionary views and the variables available in every one of them. A simple PROC SQL (remember this is the only way to access dictionary tables) will set us on the way to explore the wealth of information available at our disposal.

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

It shows the information on available dictionary tables, columns in each of them, column labels, types, formats and positions, so is more or less an extended version of information displayed after execution of a 'describe table dictionary.xxx' statement.

As stipulated in the abstract, dictionary tables contain a great wealth of information. DICTIONARY.OPTIONS will show all SAS options and their values as set for the current session (a close equivalent to PROC OPTIONS procedure). DICTIONARY.MACROS contains all available macro variables (global, local and automatic), so it can serve as a quick reference point if macro variables are used for setting titles or footnotes in PROC REPORT to quickly see how they will resolve.

One of the more useful views are also DICTIONARY.TABLES and DICTIONARY.COLUMNS. Using a describe table statement, it can be noticed that the TABLES view contains some information frequently sought after in day-to-day coding, such as:

- the number of observations in datasets,
- creation dates for datasets.

A more detailed picture of the information contained in the TABLES view is presented below. It has been produced using a 'describe table' statement which can be seen as an alternative way of obtaining the information contained in DICTIONARY.DICTIONARIES. The main difference is that in case of 'describe table' the information is output in the log and it is not as detailed as in the DICTIONARIES view.

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

(Excerpt from log)

```
19    proc sql;
20      describe table dictionary.tables;
NOTE: SQL table DICTIONARY.TABLES was created like:

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',
   (…)
   num_character num label='Number of Character Variables',
   num_numeric num label='Number of Numeric Variables'
  );

21    quit;
```

Aside all this very helpful information, in my opinion the most interesting uses of dictionary tables relate, however, to data checks. As most of us have probably experienced, data received from data management are often dirty and it seems data managers use a different set of checks compared to biostatistics professionals which makes them incapable of detecting some of data irregularities. With dictionary tables, it is relatively easy to inspect whole libraries containing raw data for obvious problems. Having this in mind, in the remainder of the paper I would like to present some ideas that could be used to inspect data issues relating to dates and date variables.

FOCUS ON DATES

One of the most useful dictionary views is DICTIONARY.COLUMNS, which contains information on columns (and their attributes) present in all datasets available in any libname defined in a current SAS session. As the purpose of the paper is to show a solution for inspecting date variables- DICTIONARY.COLUMNS would be an indispensable tool.

Date variables could have very different variable names, but in a great majority of transferred data, they could be identified by the word 'Date' included in their label. Using the code below and restructuring the output a little bit, we could obtain a list of date variables in all datasets from the data transfer.

```
%macro get_date_vars(in_lib=,out_ds=);

  proc sql;
    create table pre_&out_ds. as
    select * from dictionary.columns
    where libname = upcase("&in_lib.")
    order by memname;
  quit;

  data &out_ds.;
    set pre_&out_ds.;
    where index(upcase(label),'DATE')>0;
  run;

%mend get_date_vars;
```

This list could look for example like Table 1 below.

Table 1. Extract from DICTIONARY.COLUMNS output showing only variables with 'Date' in variable label

	Library Name	Member Name	Column Name	Column Type	Column Length	Column Position	Column Number in Table	Column Label
1	SDTM	AE	AEENDTC	char	16	642	36	End Date/Time of Adverse Event
2	SDTM	AE	AESTDTC	char	16	626	35	Start Date/Time of Adverse Event
3	SDTM	CM	CMENDTC	char	10	548	17	End Date/Time of Medication
4	SDTM	CM	CMSTDTC	char	10	538	16	Start Date/Time of Medication
5	SDTM	DM	BRTHDTC	char	10	141	14	Date/Time of Birth
6	SDTM	DM	DMDTC	char	10	242	25	Date/Time of Collection
7	SDTM	DM	DTHDTC	char	1	136	11	Date/Time of Death
8	SDTM	DM	RFENDTC	char	16	68	6	Subject Reference End Date/Time
9	SDTM	DM	RFICDTC	char	10	116	9	Date/Time of Informed Consent
10	SDTM	DM	RFPENDTC	char	10	126	10	Date/Time of End of Participation
11	SDTM	DM	RFSTDTC	char	16	52	5	Subject Reference Start Date/Time

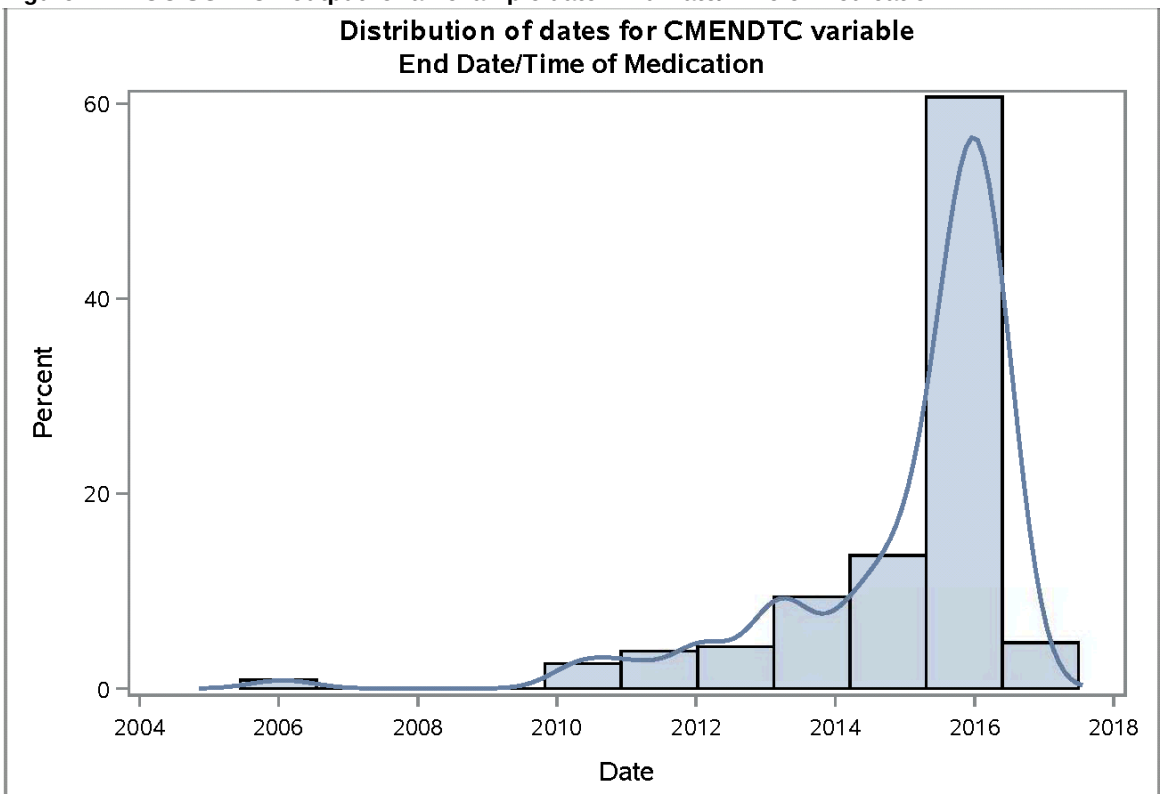
Seeing how easy it was to extract all date variables from the entire data transfer just by using variable labels, a number of useful checks could be imagined.

1. Firstly, we could construct histograms and densities for individual variables to see if they haven't got any obvious outliers.
2. As can be spotted in Table 1, some variables are in fact pairs comprising start and end dates for some events (such as adverse events). In such a case, obviously the start date could not come later than the end date

3. We could plot dates for individual subjects against the date they signed their informed consent to make sure no dates except for medical and disease history come before the consent date.
4. Finally, our analysis could even evolve towards a kind of risk-based monitoring solution, where we could analyze days of the week the data was obtained or try to detect patterns in times for reported outcomes (see if they don't show any suspicious regularities).

Going one by one through variables in Table 1, it is possible to extract values for every single one of them and use PROC SGPLOT to check if they haven't got any obvious outliers. Although the process is slightly more complicated than that, in essence it requires sourcing variable AEENDTC from the dataset AE (row 1 in Table 1), AESTDTC (row 2 in Table 1) and so on to inspect densities of all date variables from the DICTIONARY.COLUMNS output. Doing this in a do loop, we could construct plots for all date variables in our transfer. The resulting plot for one of them could be as in Figure 1.

Figure 1. PROC SGPLOT output for an example date- End Date/Time of Medication



As much as histogram bins could be worked on a little bit here (some data from the year 2016 appears to be in year 2017), we see that most data comes from years 2015-16. On top of that, we have a gap in data from year 2007 until 2009, so it seems as the earliest 'regular' data in our transfer come from the year 2010, but there are also some isolated observations from years 2005-06. Such cases would most likely be data issues.

Another thing easily noticed from Table 1 is that there are cases of start and end dates for given events. For example, AESTDTC and AEENDTC are respectively start and end dates for an adverse event. In this case obviously a start date should always be no later than the end date.

To identify such pairs (without any knowledge about the variables included in our data transfer), we could once again relate to variable labels. In most data transfers, the labels for event beginning dates contain either the word 'Start' or 'First' (such as 'Subject Reference Start Date/Time'). The end dates, on the other hand, have the words 'End', 'Stop' or 'Last' in their labels.

If we search our structure dataset from Table 1 for pairs of variables where the label includes these words and differs only by the 'Start'/'End' component, we could identify all start/end date pairs. This could be done by transposing the dataset and applying a condition: `if mod_var_lbl&i. = mod_var_lbl&j.` for all pairs of variables &i. and &j. (after modifying the labels to remove the words 'Start', 'First', 'End' etc. which can be done using, for example, the TRANWRD function). After identifying all pairs of 'Start' and 'End' dates, we could proceed to

list only those where the start date is greater than the end date or a start date is missing where the end date is not- getting in this way the list of dates suspected of data issues. Such a list could look like the one in Table 2.

Of course, someone could argue the condition could be applied on every dataset separately (AE, CM, MH etc.) and a similar result could be obtained. But the advantage of using dictionary tables is that we could build a macro that does it automatically for all datasets in a data transfer, without any *a priori* knowledge about variables they contain. Searching through Table 2, we indeed get at least 2 data issues in rows 23 and 24 that could be reported to data management either manually or by exporting this list to an Excel spreadsheet structured like a data issue log.

Table 2. Example output for data issues where end dates are before start dates (or start dates are missing and end dates are not)

	DSET_NAME	VAR_NAMES	START_DATE	END_DATE	Unique Subject Identifier
23	CM	CMSTDTC CMENDTC	2016-03-09	2015-03-10	02005880024040024
24	CM	CMSTDTC CMENDTC	2016-03-09	2015-03-12	02005880024040024
25	CM	CMSTDTC CMENDTC		2016-03-01	02005880024050006
26	CM	CMSTDTC CMENDTC		2016-03-01	02005880024050006
27	CM	CMSTDTC CMENDTC		2016-04-08	02005880024050013

Using this framework of a structure dataset containing all date variables a number of other checks could be imagined, such as:

- making sure date sequences are valid with informed consent date going first, followed by randomization date, first dose date etc.
- elements of risk-based monitoring, like searching for suspicious clusters of measurement times or days of the week.

Especially the 2nd bullet above is a very wide topic so I hope it will be forgiven that these are just mentioned here and they could be a subject of another paper in the future…

CONCLUSION

In this paper I have presented some features of dictionary tables, together with some ideas for their uses in checking data issues. The list of checks could of course be extended and is limited only by a programmer's imagination and practical examples of issues most often found in data transfers, which often depend on corporate environment, the list of checks used by the data management etc.

The most distinctive advantage of using dictionary tables in such a setting is that the checks could be performed automatically, without any *a priori* knowledge of the data transfer and eCRF forms, on practically any project. This would allow the programmer to very quickly check the data we work on are free of the most frequently spotted issues.

REFERENCES

K. P. Lafler, Exploring Dictionary Tables and Views, SUGI 30, Paper 070-30, http://www2.sas.com/proceedings/sugi30/070-30.pdf (accessed Sept 2016)

L. Libeg, The SAS® Magical Dictionary Tour, SESUG 2011, http://analytics.ncsu.edu/sesug/2011/BB09.Libeg.pdf (accessed Sept 2016)

M. Davies, You Could Look It Up: An Introduction to SASHELP Dictionary Views, http://www.ats.ucla.edu/stat/sas/library/nesug00/bt3004.pdf (accessed Sept 2016)

CONTACT INFORMATION

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

Marcin Pracz
Quintiles
1, Gottlieba Daimlera street
02-460, Warsaw
Work Phone: (+48) 22 4303160
Email: Marcin.Pracz@quintiles.com

Appendix 1. Full list of dictionary tables available with SAS® 9.4

CATALOGS	Catalogs and catalog-specific information
CHECK_CONSTRAINTS	Check constraints
COLUMNS	Columns from every table
CONSTRAINT_COLUMN_USAGE	Constraint column usage
CONSTRAINT_TABLE_USAGE	Constraint table usage
DATAITEMS	Information Map Data Items
DESTINATIONS	Open ODS Destinations
DICTIONARIES	DICTIONARY tables and their columns
ENGINES	Available engines
EXTFILES	Files defined in FILENAME statements, or implicitly
FILTERS	Information Map Filters
FORMATS	Available formats
FUNCTIONS	Available functions
GOPTIONS	SAS/GRAPH options
INDEXES	Indexes
INFOMAPS	Information Maps
LIBNAMES	LIBNAME information
LOCALES	Available Locales
MACROS	Defined macros
MEMBERS	Tables, catalogs, and views
OPTIONS	SAS options
PROMPTS	Information Map Prompts
PROMPTSXML	Information Map Prompts XML
REFERENTIAL_CONSTRAINTS	Referential constraints
REMEMBER	Remembered information?
STYLES	Styles?
TABLES	Tables and table-specific information
TABLE_CONSTRAINTS	Table constraints
TITLES	TITLE statements
VIEWS	Views and view-specific information
VIEW_SOURCES	Sources Referenced by View
XATTRS	Extended Attributes