PhUSE 2015

Paper CC05

Dynamic, Powerful, User-friendly SQL JOINS Made Easy with %VARLIST, a Generic SAS Macro Function

Jean-Michel Bodart, Business & Decision Life Sciences, Brussels, Belgium

ABSTRACT

PROC SQL is renowned as a powerful method to perform a variety of simple to complex dataset joins (or merges) with SAS®. However, writing and maintaining the SQL code quickly becomes a tedious task as soon as we depart from the simplest situations.

The macro-function %VARLIST() was developed as a generic utility function to retrieve lists of variables from one or more datasets, and process them according to specified criteria in order to dynamically generate SQL code fragments. Thus a few simple %VARLIST() calls inserted between additional SQL keywords will generate a full SQL join of arbitrary complexity.

This paper will present in a few examples the general syntax of SQL joins of increasing complexity applied to clinical data, show how %VARLIST() can be used to simplify SQL code generation and make it dynamic, and introduce the implementation of the necessary functionality as a macro-function.

INTRODUCTION

PROC SQL is renowned as a powerful method to perform a variety of simple to complex dataset joins (or merges) with SAS. However, writing and maintaining the SQL code quickly becomes a tedious task especially when not all variables cannot be handled all the same way (using *, or natural joins) and but must be listed explicitly.

SAS Macros are another well-known, powerful feature of the SAS system that can be used for automating, repeating and/or controlling a program execution. The macro language has its own syntax, and a limited set of statements and built-in functions.

Macro functions such as %INDEX(), %SUBSTR(),or %LOWCASE() compute a result based on specified or default arguments, and return (or resolve to) a value, which is then used inside other user-written macro definitions or in open code such as data step, global statements and procedure calls. Nesting macro functions is also possible, i.e. the result of an inner macro function becomes the argument of an outer macro function. For example the following nested macro functions can be used to retrieve the 3 characters code of the month in lowercase from a DATE9. Formatted date:

```
%put %LOWCASE(%SUBSTR(12OCT2015, 3, 3));
```

Which writes to the log:

```
oct
```

Moreover, SAS macro language allows users to create new macro functions and extend the available functionality.

USER-DEFINED MACRO-FUNCTIONS

User-written macro-functions such as %VARLIST() are composed of macro-code that is purely processed by the SAS Macro Language Processor; they cannot contain any DATA step or PROC step or global statement. Any text that remains “unprocessed” by the macro language processor becomes the return-value of the macro-function, i.e. what the macro-function resolves to. Here is an example macro-function that reformats the date passed as its argument.

```
%macro nicedate(date);
   %local datenum;
   %let datenum=%sysfunc(inputn(&date, ANYDTDTE.));
   %sysfunc(putn(&datenum, WEEKDATX.))
%mend;
%let datex=%nicedate(&sysdate9); %put I ran this example on &datex
[sysdate9=&sysdate9];
```

The log shows:

```
I ran this example on Wednesday, 2 September 2015 [sysdate9=02SEP2015]
```

In the example call, the current date from &sysdate9 is passed to macro %nicedate as macro-parameter &date, which is read by the nested function call %sysfunc(inputn()) into macro-variable &datenum as a numerical value using informat ANYDTDTE. Then &datenum value is formatted back to a character date with format WEEKDATX., by the nested function call %sysfunc(putn()). Note the returned value from this function call is not assigned to a macro variable within macro %nicedate, nor interpreted as a macro statement, and is also not followed by a semicolon; that

value is then left unchanged by the macro language processor, and becomes itself the return value of the %nicedate() macro-function. The result of the macro function %nicedate is assigned to macro-variable &datex, which is printed to the log using the %put macro statement.

THE MACRO-FUNCTION %VARLIST

The macro-function %VARLIST() was developed as a generic utility function to retrieve lists of variables from one or more datasets, and process such lists according to specified criteria in order to dynamically generate code fragments. Here is a simple example of %VARLIST() used to retrieve (by default) all the variables names found in the specified dataset SASHELP.CLASS:

```
%let v = %VARLIST(data=SASHELP.CLASS);
%put v = &v;
```

Which displays in the log the list of variables found in SASHELP.CLASS, according to the dataset order:

```
v = Name Sex Age Height Weight
```

INGREDIENTS USED TO BUILD %VARLIST

%VARLIST() was created using SAS built-in macro-functions: %upcase(), %qupcase(), %symexist(), %eval(), %index(), %eval(), %scan(), %qscan(), %length(), %substr( ), %qsubstr(), %str(), %quote(), %bquote(), %superq(), %unquote(); as well as data step / SCL functions and call routines (most of which can be made available to the macro processor via the %sysfunc() and %qsysfunc() functions or the %syscall statement): open(), close(), attrn(), varnum(), varname(), varlen(), vartype(), varfmt(), varinfmt(), varlabel(), tranwrd(), index(), putn(), prxparse(), prxmatch(), prxchange(), prxfree(), quote(). The macro statements %let, %if .. %then, %else, %do .. %to, %do .. %while(), %do .. %until() are also used.
For more information about these elements, refer to SAS online help.

CONTROLLING THE VARIABLES TO BE INCLUDED IN THE RETURNED LIST

It is possible to return only the variables names matching those from a specified list, which must be space-separated. The following only returns variables in the list: “Weight Height BMI BSA” when found in the dataset SASHELP.CLASS:

```
%put %VARLIST(data=SASHELP.CLASS, var=Weight Height BMI BSA);
```

Which results in:

```
Weight Height
```

The matching of variables names is non case-sensitive. The keywords **#all#**, **#num#**, **#char#** can be used to refer respectively to all variables, the numerical variables, the character variables from the dataset. You can use constructs like **#2-4#** to refer to the second, third and fourth variables according to dataset order, or **#9#** to the ninth variable; **col05-07** will match variables “col05”, “Col6” and “COL07” (case-insensitive); use **B:** for variable names starting with “B” (or “b”); use **:ght** for variables ending with “ght”, **cm:dt** for variables starting with “cm” and ending with “dt”, **EXDOSE-EXDUR** to match all variables with names between EXDOSE and EXDUR in alphabetical order (e.g. EXDOSE, EXDOSFRM, EXDOSFRQ, EXDOSRGM, EXDOSTOT, EXDOSTXT, EXDOSU, EXDUR but not EXCAT nor EXELTM). **Sex--Height** matches all variables from Sex to Height in dataset order, e.g.:

```
%put %VARLIST(data=SASHELP.CLASS, var=Sex--Height);
Sex Age Height
```

Perl regular expressions like **/<pattern>/** can also be used to match variable names; e.g. **/^trt\d\dn?$/** matches any variable name starting with "trt", followed by two digits (from 00 to 99), followed optionally by "n", and without any additional characters. A case-insensitive match is applied.

Variable list operator **#not#** (or **#except#**) makes it possible to return only those variables names not matching those from a specified list. The following only returns numerical variables from dataset SASHELP.CLASS that are not in the list: “Weight Height BMI BSA”:

```
%put %VARLIST(data=SASHELP.CLASS, var=#num# #not# Weight Height BMI BSA);
```

Which results in:

```
Age
```

RETRIEVING VARIABLES FROM MULTIPLE DATASETS

It is possible to return variables names retrieved from a combination of multiple datasets. By default, specifying multiple datasets will return a unique list of variable names found in one or more of these datasets. The default dataset operator is **#UNION#** (or **#OR#)**. Other dataset operators are:

- **#INTERSECT#** (or **#AND#**), which requires the variable(s) to exist both in the dataset (or combination of datasets) on the left side of the operator, and in the (first) dataset on its right side.
- **#EXCEPT#** (or **#NOT#**), which requires the variable(s) to exist in the dataset (or combination of datasets) on the left side of the operator, but to be absent from the (first) dataset on its right side.
- **#XOR#** (exclusive OR), which requires the variable(s) to exist either in the dataset (or combination of datasets) on the left side of the operator, or in the (first) dataset on its right side, but they cannot exist in both.

Note that dataset operators are evaluated sequentially from left to right, and using parentheses to alter evaluation order is not supported.

The following examples show the variables that are retrieved from various combinations of datasets SASHELP.CLASS and SASHELP.CLASSFIT:

```
%put One or Both: %VARLIST(data=SASHELP.CLASS #UNION# SASHELP.CLASSFIT).;
One or Both: Name Sex Age Height Weight predict lowermean uppermean lower upper.
%put Both: %VARLIST(data=SASHELP.CLASS #INTERSECT# SASHELP.CLASSFIT).;
Both: Name Sex Age Height Weight.
%put CLASS not CLASSFIT: %VARLIST(data=SASHELP.CLASS #NOT# SASHELP.CLASSFIT).;
CLASS not CLASSFIT:.
%put CLASSFIT not CLASS: %VARLIST(data=SASHELP.CLASSFIT #NOT# SASHELP.CLASS).;
CLASSFIT not CLASS: predict lowermean uppermean lower upper.
%put One not Both: %VARLIST(data=SASHELP.CLASS #XOR# SASHELP.CLASSFIT).;
One not Both: predict lowermean uppermean lower upper.
```

PROCESSING LISTS OF VARIABLES

The parameter *pattern*= is a kind of expression that controls how each variable retrieved or processed by %VARLIST() will appear in the returned value. In building that expression you can use keywords that will be replaced by the corresponding attribute of the variable, and non-keyword text that will appear unchanged. A separator specified by parameter *sep*= will be inserted between the processed pattern for one variable and the processed pattern for the next variable. E.g., the keywords **#var#**, **#vtyp#**, **#vlen#**, **#vfm#**, **#vinfmt#** and **#vlab#** can be used in parameter *pattern*= and will be replaced respectively by the variable name, its type (C or N), length, format, informat and label (not quoted). Keywords recognized as part of the *sep*= parameter are: **#space#** (or **#s#**), **#comma#** (or **#c#**), **#semicolon#** (or **#sc#**), replaced respectively by a single space character ( ), a comma (,) and a semicolon (;); the keywords **#and#** and **#or#** will be replaced respectively by the word " and " and " or " (both enclosed within spaces); **#cs#** and **#scs#** stand respectively for a comma and a semicolon, each of them followed by one space.

Using the default value of parameters *pattern*= (i.e. keyword **#var#**) and *sep*= (i.e. keyword **#space#**) will simply return the list of variable names separated by spaces.

Therefore, %VARLIST() could be used to return a specific attribute for a single variable, e.g.

```
%put %VARLIST(data=SASHELP.CLASS, var=Sex, pattern=#vtyp#);
C
%put %VARLIST(data=SASHELP.CLASS, var=Sex, pattern=#vlen#);
1
```

It could also be used to rename variables, e.g. by adding an underscore (_) as prefix, and a C (for character variables) or an N (for numeric variables) to the original variable name:

```
%put RENAME %VARLIST(data=SASHELP.CLASS, pattern=#var#=_#var##typ#);
RENAME Name=_NameC Sex=_SexC Age=_AgeN Height=_HeightN Weight=_WeightN
```

The following example retrieves multiple attributes of variables MEMNAME and CRDATE from dataset SASHELP.VTABLE, and demonstrates that the keywords #vfm# and #vinfmt# are replaced by an empty string for variables respectively without a format and an informat (such as MEMNAME)[i]:

```
%put %VARLIST(data=sashelp.vtable, var=memname crdate, sep=#cs#, pattern=#var#
length=#vlen# label="#vlab#" format=#vfmt# informat=#vinfmt#);
```

The log shows:

```
memname length=32 label="Member Name" format=  informat= ,
crdate length=8 label="Date Created" format=DATETIME. informat=DATETIME.
```

In this example, the returned string has a syntax compatible with items usable in PROC SQL SELECT clause (`column-name <AS alias> <column-modifier <... column-modifier>>`) for variables having both a format and an informat (such as CRDATE), but a syntax error (expecting a format name / an informat name) would be generated for variables (such as MEMNAME) without a format and/or informat. Additionally, the expression generated from "**#vlab#**" (or '**#vlab#**') may contain unmatched quotes if the label itself contains quotes.

Such issues can be prevented using alternative keywords: **#vfmtdef#** and **#vinfmftdef#** are similar to **#vfmt#** and **#vinfmt#** but return default (in)formats ($<length>. and Best12. respectively for character and numeric variables, where <length> is the length of the character variable), and **#vlabq#** returns an already quoted version of the variable label that caters for embedded quotes (or " " if no label is defined).

As such a long pattern could be of much use, a shorthand has been created: keyword **#vlenlabfmt#** will be replaced by the sequence: `<variable> [LENGTH=<length>] LABEL=<quoted label> [FORMAT=<format>] [INFORMAT=<informat>]` where parts between square brackets `[]` will only be present if the corresponding variable attribute could be retrieved from the associated dataset.

Finally, for a user looking for the first time at the contents of a SAS dataset, it can be quite handy to display both the label and the name of each variable at once (the functionality exists in SAS Viewer, but not within SAS ViewTable nor in PROC PRINT or PROC SQL output. Therefore the following *pattern*= keywords have been defined, which result in the dataset and variable name (within square brackets) being appended to the variable label: **#vlabsrc#**, **#vlabsrcq#**, **#vlenlabsrcfmt#** (as replacement for the same keywords without "src").
Let us illustrate this by looking at the first 5 variables (*var*=#1-5#) and first 2 records (inobs=2) in SASHELP.VCOLUMN:

```
proc sql inobs=2;
   select %VARLIST(data=sashelp.vcolumn, var=#1-5#, sep=#cs#,
                   pattern=#vlenlabsrcfmt#) from sashelp.vcolumn;
quit;
```

Library Name [SASHELP.VCOLUMN.LIBNAME]	Member Name [SASHELP.VCOLUMN.MEMNAME]	Member Type [SASHELP.VCOLUMN.MEMTYPE]	Column Name [SASHELP.VCOLUMN.NAME]	Column Type [SASHELP.VCOLUMN.TYPE]
SASHELP	AACOMP	DATA	locale	char
SASHELP	AACOMP	DATA	key	char

With the MPRINT option, the log shows how the code generated by %VARLIST() was integrated in the PROC SQL call, together with the length of the variables and the (absence of) format in their definition, and therefore clearly documents how the output was generated:

```
8821   proc sql inobs=2;
8822      select %VARLIST(data=sashelp.vcolumn, var=#1-5#, sep=#cs#,
8823                      pattern=#vlenlabsrcfmt#)
MPRINT(VARLIST):   libname length=8 label="Library Name [SASHELP.VCOLUMN.LIBNAME]",
memname length=32 label="Member Name [SASHELP.VCOLUMN.MEMNAME]",
memtype length=8 label="Member Type [SASHELP.VCOLUMN.MEMTYPE]",
name length=32 label="Column Name [SASHELP.VCOLUMN.NAME]",
type length=4 label="Column Type [SASHELP.VCOLUMN.TYPE]"
8824        from sashelp.vcolumn;
```

PROCESSING LISTS OF VARIABLES WITHOUT A SPECIFIED DATASET

%VARLIST() can also be used to process lists of variables without specified datasets to lookup variable attributes, when no variable keyword is specified that requires a dataset (such as: **#all#**, **#num#**, **#char#**, **#3#**, **#4-7#**, **<prefix>:**, **:<suffix>**, **<prefix>:<suffix>**, **<var1>-<var2>**, **<var1>--<var2>**, **/<pattern>/**).
This can be useful to expand some keyword constructs that do not require a lookup dataset, e.g. **col05-07** will be expanded as: `COL05 COL06 COL07`. Also, when macro-variables are used to refer to dynamic variable lists (space-separated), %VARLIST() can be used to return them separated by comas for use in PROC SQL calls.
E.g. a list of key variables may be space-separated to be used in a BY statement of a DATA or PROC step, but should be comma-separated for use in an SQL query:

```
%let keyvars=STUDYID USUBJID AVISITN PARAMCD;
proc sql;
   create table x2 as select * from x1
      order by %VARLIST(var=&keyvars, sep=#cs#);
quit;
```

With option MPRINT, the log shows:

```
8827   proc sql;
8828      create table x2 as select * from x1
8829         order by %VARLIST(var=&keyvars, sep=#cs#);
MPRINT(VARLIST):   STUDYID, USUBJID, AVISITN, PARAMCD
8830   quit;
```

Additional variable list operators may prove useful when processing dynamic lists of variables, passed as macro variables:

- **#intersect#** (or **#and#**): when present between two sublists of variables, this will resolve into one list of variables that are present in both of the sublists
- **#xor#**: when present between two sublists of variables, this will resolve into one list of variables that are present in either of the sublists but not both
- **#not#** (or **#except#**) (already mentioned above); when present between two sublists of variables, this will resolve into one list of variables that are present on the left of the operator but only if not present on the right

Note: variable list operators are processed sequentially from left to right, and using parentheses to alter evaluation order is not supported. A variable sublist is defined as all the variables between one variable list operator (or the start

of the list) and the next variable list operator (or the end of the list) specified as value for the *var*= parameter. Variable list operators process sublists in a case-insensitive way, so the result of: `age #not# AGE` is an empty list.

USING THE MACRO-FUNCTION %VARLIST IN SQL JOINS AND OTHER CLAUSES

Most SQL Joins used in the processing of clinical data can be considered as some particular case of an OUTER JOIN, which means combining two datasets based on specified matching rules (often an equijoin, i.e. observations from both dataset match when corresponding variable(s) have equal values), while keeping all observations from each dataset that do not have a match in the other dataset (=FULL JOIN), in addition to all observations that actually match between both datasets (and are therefore combined together). The particular case of not keeping observations that do not match corresponds to an INNER JOIN. The case of keeping non-matching observations from only one dataset in addition to the matching observations is a LEFT JOIN (if non-matching observations from the first dataset are kept), or a RIGHT JOIN (if non-matching observations from the second dataset are kept).
If we want to develop code that can be easily adapted to the most complex cases, we need to consider each case as potentially a FULL (OUTER) JOIN, although INNER and LEFT or RIGHT (OUTER) JOINS may have a simpler (but less easily adaptable) implementation. NATURAL joins offer the least possibility to control and adapt.

FULL OUTER JOIN EXAMPLE

Let's consider the need to merge two datasets:

- CC05.SiteID, with data about Sites involved in a Clinical Study
- CC05.Shipment, which records the Investigational Product Shipments to the Clinical Study Sites

We can use PROC SQL with %VARLIST() to show the contents of both datasets.
The log shows:

```
12455   proc sql;
12456      title "CC05.SiteID: Sites involved in Clinical Study";
12457      select %VARLIST(data=CC05.SiteID, pattern=#vlenlabsrcfmt#, sep=#cs#)
MPRINT(VARLIST):   SITEID length=8 label="Site ID [CC05.SITEID.SITEID]",
CITY length=12 label="City [CC05.SITEID.CITY]",
STATE length=12 label="State [CC05.SITEID.STATE]",
HOSPITAL length=61 label="Hospital [CC05.SITEID.HOSPITAL]"
12458         from CC05.SiteID;
```

CC05.SiteID: Sites involved in Clinical Study

Site ID [CC05.SITEID.SITEID	City [CC05.SITEID.CITY]	State [CC05.SITEID.STATE]	Hospital [CC05.SITEID.HOSPTAL]
1	Atlanta	Georgia	Children's Healthcare of Atlanta
2	Chicago	Illinois	Ann and Robert H. Lurie Children's Hospital of Chicago
3	Los Angeles	California	Children's Hospital Los Angeles
4	Houston	Texas	Texas Children's Hospital, Houston

```
12462   proc sql;
12463      title "CC05.Shipment: IP Shipments to Clinical Study Sites";
12464      select %VARLIST(data=CC05.Shipment, pattern=#vlenlabsrcfmt#, sep=#cs#)
MPRINT(VARLIST):   SHIPID length=8 label="Shipment ID [CC05.SHIPMENT.SHIPID]",
SITEID length=8 label="Site ID [CC05.SHIPMENT.SITEID]",
SHIPDATE length=8 label="Shipment Date [CC05.SHIPMENT.SHIPDATE]" format=DATE9.
12465         from CC05.Shipment;
```

CC05.Shipment: IP Shipments to Clinical Study Sites

Shipment ID [CC05.SHIPMENT.SHIPID]	Site ID [CC05.SHIPMENT.SITEID]	Shipment Date [CC05.SHIPMENT.SHIPDATE]
1	3	15JAN2012
2	1	21JAN2012
3	4	20MAY2012
4	1	22MAY2012
5	5	03JUN2012

In a full outer equijoin, the datasets have to be merged by matching on equal values of corresponding variables. The matching conditions are specified in an ON clause. Generally when variables with same name and type exist in both datasets (variable SITEID in our example) they should be used as part of the matching key, although there might be exceptions – key variables may have different names, or require type conversion, and in case many variables have common names and types some may need to be excluded from the matching key. This may be because they are redundant, have values that don't match exactly (e.g. open fields such as comments, verbatim drug name, Adverse event or medical condition) or that have been updated more recently in one dataset than in the other. Alternatively they may have incompatible types that cannot be converted into each other, or simply represent distinct concepts.
In our example we need to match records having the same SHIPID value in both dataset, which we may want to code:

```
ON CC05.SiteID.SITEID = CC05.Shipment.SITEID
```

However SQL only supports one- or two-level variable names, the libname part cannot be included, so we must use:

```
ON SiteID.SITEID = Shipment.SITEID
```

The SQL SELECT clause needs to list all variables to be retrieved from the JOIN in the desired order for the resulting dataset. When the same variable name exists in both datasets being joined, we must specify from which dataset the variable shall be taken, otherwise the first dataset containing that variable will be assumed.
But if the resulting dataset has to include non-matching observations from both dataset, choosing to retrieve a variable from a single dataset will result in missing values for the non-matching observations originating from the other dataset.
To avoid this, we can use the COALESCE() function, which uses the value from its first argument having a non-missing value, if any. All its arguments must have the same type. The results of the COALESCE() function do not inherit the variable attributes (name, label, format, informat, length) from the source variable(s), only the variable type is kept and default attribute values are assigned. So in order to retain the relevant attributes from the source variable they must be explicitly specified after the COALESCE function.
Therefore we can use:

```
SELECT COALESCE(SiteID.SITEID, Shipment.SITEID) as SITEID length=8 label="Site ID"
```

This piece of code can be generated for all variables in common to both datasets with:

```
SELECT %VARLIST(data=CC05.SiteID #INTERSECT# CC05.Shipment, sep=#cs#,
pattern=COALESCE(SiteID.#var#, Shipment.#var#) as #vlenlabfmt#)
```

In order to also include in the SELECT clause the list of variables that exist in one dataset but not both, we can add:

```
, %VARLIST(data=CC05.SiteID #XOR# CC05.Shipment, sep=#cs#)
```

With default *pattern*= value, the variables are listed without specifying their source dataset, which is acceptable for those variables that only exist in a single dataset. However we could have it completely specified (in the log) with `pattern=#dsa#.#var#` (or if both datasets were specified as one-level, with `pattern=#dsn#.#var#`).
Indeed the *pattern*= keyword **#dsn#** resolves as the dataset name (including libname if specified), but **#dsa#** resolves either as the dataset alias (if specified), or (if not) as the bare dataset name (without libname).
To specify a dataset alias in the *data*= parameter, we can use the syntax: `data=[libname.]<dataset>:<alias>`.
In that case, the same dataset alias must also be specified in the SQL FROM clause:

```
FROM [libname.]<dataset> [as] <alias>.
```

It is possible to combine and simplify the two %VARLIST() calls above, as well as generalize them for the case where more than 2 datasets are joined together successively, as:

```
SELECT %VARLIST(data=CC05.SiteID CC05.Shipment, sep=#cs#, pattern=#autoCOAL#)
```

where pattern keyword **#autoCOAL#** generates either the appropriate COALESCE() function call for all variables that exist in more than one dataset specified in *data*= parameter, or the equivalent to `pattern=#dsa#.#var#` for variables found in a single of these datasets.
The type of JOIN (FULL, LEFT, RIGHT or INNER), the second dataset (with alias if applicable) and the SQL ON clause must then be specified after the first dataset on the FROM clause. When the syntax for a FULL join is used from the beginning, changing the type of JOIN is just a matter of replacing the keyword FULL with the desired type.
In our example (full equijoin matching on all variables in common and without assigning dataset aliases), we could have:

```
FROM CC05.Shipment FULL JOIN CC05.SiteID
   ON %VARLIST(data=CC05.SiteID #INTERSECT# CC05.Shipment, sep=#and#,
               pattern=SiteID.#var# eq Shipment.#var#)
```

Note *sep*=**#and#** in case more than one variable are common to both datasets, since the generated ON clause must be like: `ON (SiteID.<var1>=Shipment.<var1>) and (SiteID.<var2>=Shipment.<var2>) [and ..]`
In this last %VARLIST() call, the pattern can also be simplified and generalized for the case where more than 2 datasets specified in *data*= parameter are joined together successively using keyword **#autoEQ#**, as:

```
%VARLIST(data=CC05.SiteID #INTERSECT# CC05.Shipment [#INTERSECT# ..], sep=#and#,
pattern=#autoEQ#)
```

The complete code for this JOIN (after adding a CREATE TABLE clause and an optional ORDER BY clause) would show the following in the log:

```
13988   proc sql;
```

```
13989      create table SiteShip2 as
13990         select %VARLIST(data=SiteID Shipment, sep=#cs#, pattern=#autoCOAL#)
MPRINT(VARLIST):   coalesce(SiteID.SITEID, Shipment.SITEID) as SITEID length=8
label="Site ID", SiteID.CITY, SiteID.STATE, SiteID.HOSPITAL, Shipment.SHIPID,
Shipment.SHIPDATE
13991            from  SiteID
13992               full join
13993                  Shipment
13994                  on %VARLIST(data=CC05.SiteID #INTERSECT# CC05.Shipment,
sep=#and#, pattern=#autoEQ#)
MPRINT(VARLIST):   (SiteID.SITEID = Shipment.SITEID)
13995               order by SiteID, SHIPID
13996                  ;
NOTE: Table WORK.SITESHIP2 created, with 6 rows and 6 columns.

13979      title "CC05.SiteShip: Investigational Product Shipment by Site";
13980      select * from CC05.SiteShip;
```

CC05.SiteShip: Investigational Product Shipment by Site

Site ID	City	State	Hospital	Shipment ID	Shipment Date
1	Atlanta	Georgia	Children's Healthcare of Atlanta	2	21JAN2012
1	Atlanta	Georgia	Children's Healthcare of Atlanta	4	22MAY2012
2	Chicago	Illinois	Ann and Robert H. Lurie Children's Hospital of Chicago	.	.
3	Los Angeles	Califor-	Children's Hospital Los Angeles	1	15JAN2012
4	Houston	Texas	Texas Children's Hospital, Houston	3	20MAY2012
5				5	03JUN2012

Note a record with Site ID = 2 was not found in dataset CC05.Shipment, therefore corresponding variables Shipment ID and Shipment date remained empty. This record would have been dropped in a LEFT JOIN.
A record with Site ID = 5 was not found in dataset CC05.SiteID, therefore corresponding variables City, State and Hospital have missing values. This record would have been dropped in a RIGHT JOIN.
With an INNER JOIN, both non-matching records (with Site IDs 2 and 5) would have been dropped.
The Site ID (SITEID) variable which was in common to both datasets always has a value even in non-matching records thanks to the COALESCE() function.

GENERALIZED SQL JOIN SYNTAX

Instead of only generating the contents of one ON clause with *pattern*=#autoEQ#, it is possible to automatically generate the contents of a complete FROM clause (including JOINS between 2 or more datasets), based on the following assumptions:

- The dataset(s) listed in *data*= parameter must be included (with corresponding alias(es) if defined)
- If multiple datasets are present, they must be joined successively using equijoin(s), which can be specified using the specific keywords **#autoFULLEQ#**, **#autoLEFTEQ#**, **#autoRIGHTEQ#** and **#autoINNEREQ#** respectively as either FULL, LEFT, RIGHT or INNER joins
- The ON clause must specify the equality of all variables in common between the left-hand dataset (or intermediate datasets resulting from preceding merges) to be merged and the (next) right hand dataset. If no variables are in common, the ON clause will be specified as the condition (1 EQ 1), which is always true and results in a Cartesian Product or CROSS JOIN

One of these 4 specific keywords can be specified either as the full contents of the *pattern*= parameter, so that the %VARLIST() call returns the code of the FROM clause, or as the full contents of parameter *from*=, so that it returns the entire code of the SELECT clause, followed by the SQL keyword FROM and the full contents of the FROM clause.
Let's consider the additional dataset Randolist with the following variables:

```
%put %VARLIST(data=CC05.RandoList);
TRTN TRTCD TRT SHIPID BLKID PACKID RANDOMORDER
```

We can now JOIN this dataset together with the previous two, via a FULL equijoin based on the variables in common, using dataset aliases and excluding (e.g.) the variable RANDOMORDER as follows (with a single %VARLIST() call):

```
proc sql;
  create table CC05.SiteShipRand3 as select
```

```
    %VARLIST(data=CC05.SiteID:a CC05.Shipment:b RandoList:c
          , var=#all# #not# :ORDER, sep=#cs#, pattern=#autoCOAL#, from=#autoFULLEQ#);
```

The log shows:

```
3571     create table CC05.SiteShipRand3 as select
3572     %VARLIST(data=CC05.SiteID:a CC05.Shipment:b RandoList:c
         , var=#all# #not# :ORDER, sep=#cs#, pattern=#autoCOAL#, from=#autoFULLEQ#);
MPRINT(VARLIST):   coalesce(a.SITEID, b.SITEID) as SITEID length=8 label="Site ID"
             , a.CITY, a.STATE, a.HOSPITAL
             , coalesce(b.SHIPID, c.SHIPID) as SHIPID length=8 label="Shipment ID"
             , b.SHIPDATE, c.TRTN, c.TRTCD, c.TRT, c.BLKID, c.PACKID
          FROM CC05.SiteID as a
               FULL JOIN CC05.Shipment as b ON (a.SITEID = b.SITEID)
               FULL JOIN RandoList as c ON (b.SHIPID = c.SHIPID)
NOTE: Table CC05.SITESHIPRAND3 created, with 50 rows and 11 columns.
```

THE %VARLIST MACRO CODE

The full macro code (and additional examples of use) are available at
http://www.phusewiki.org/wiki/index.php?title=SAS_macro-function_%25VARLIST

CONCLUSION

Carefully built %VARLIST() calls, combined with additional SQL keywords and possibly referring to variables list passed as macro-variables can be used to consistently generate dynamic code for one or more powerful SQL joins of low to high complexity, precisely documented in the log. Consequently, it seems this macro can provide a good programming basis and even prove to be time-saving and prevent errors in the somewhat cumbersome process of coding SQL complex joins.

REFERENCES

Foster, Edward. "Create your own Functions using SAS/MACRO and SCL", PhUSE 2006, CS06.

Hendrickx, John. "Dequote me on that: Using the dequote function to add some friendliness to SAS macros", PhUSE 2014, CC02.

Mendez, Lisa and Dunn, Toby. "A Propaedeutics for PROC SQL Joins", SAS Global Forum 2009

SAS Institute, "SAS 9.3 Macro Language: Reference"

SAS Institute, "SAS 9.3 Functions and Call Routines: Reference"

SAS Institute, "SAS 9.3 SQL Procedure: User's Guide"

Williams, Christianna S. "Queries, Joins, and WHERE Clauses, Oh My!! Demystifying PROC SQL", SAS Global Forum 2012

ACKNOWLEDGEMENTS

The author's thanks go to Olena Andriyevska, Geert Peel and Dirk Spruck for their support.

CONTACT INFORMATION

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

Jean-Michel Bodart
Business & Decision Life Sciences
Rue Saint-Lambert 141
1200 Brussels (Belgium)
Work Phone: +32 (0)2 774 11 00
Fax: +32 (0)2 774 11 99
Email: jean-michel.bodart@businessdecision.com
Web: www.businessdecision-lifesciences.com

[i] The same applies to **#vlab#** for variables without a label