SCSUG 2019

Like, Learn to Love SAS® Like

Louise S. Hadden, Abt Associates Inc.

ABSTRACT

How do I like SAS®? Let me count the ways.... There are numerous instances where LIKE or LIKE statements can be used in SAS - and all of them are useful. This paper will walk through such uses of LIKE as: searches and joins with that smooth LIKE operator (and the NOT LIKE operator); the SOUNDS LIKE operator; using the LIKE condition to perform pattern-matching and create variables in PROC SQL; and PROC SQL CREATE TABLE LIKE.

INTRODUCTION

SAS provides numerous time and angst-saving techniques to make the SAS programmer's life easier. Among those techniques are the ability to search and select data using SAS functions and operators in the data step and PROC SQL, the ability to join data sets based on matches at various levels, the ability to create variables based on selecting values from other variables, and the ability to create empty "shells" of existing data sets using SAS metadata. This paper explores how LIKE is featured in each one of these techniques, and is suitable for all SAS practitioners. I hope that LIKE will become part of your SAS toolbox, too.

SMOOTH OPERATORS

SAS operators are used to perform a number of functions: arithmetic calculations, comparing or selecting variable values, or logical operations. Operators are loosely grouped as "prefix" (for example a sign before a variable) or "infix" which generally perform an operation BETWEEN two variables. Arithmetic operations using SAS operators may include exponentiation (**), multiplication (*), and addition (+), among others. Comparison operators may include greater than (>, GT) and equals (=, EQ), among others. Logical, or Boolean, operators include such operands as || or !!, AND, and OR, and serve the purpose of grouping SAS operations. Some operations that are performed by SAS operators have been formalized in functions. A good example of this is the concatenation operators (|| and !!) and the more powerful CAT functions which perform similar, but not identical, operations. Like operators are most frequently utilized in the data step and PROC SQL via a data step.

There is a category of SAS operators that act as comparison operators under special circumstances, generally in where statements in PROC SQL and the data step (and DS2) and subsetting if statements in the data step. These operators include the LIKE operator and the SOUNDS like operator, as well as the CONTAINS and the SAME-AND operators. It is beyond the scope of this short paper to discuss all the smooth operators, but they are definitely worth a look.

LIKE OPERATOR

Character operators are frequently used for "pattern matching", that is, evaluating whether a variable value equals / does not equal / sounds like a specified value or pattern. The LIKE operator is a case sensitive character operator that employs two special "wildcard" characters to specify a pattern: the percent sign (%) indicates any number of characters in a pattern, while the underscore (_) indicates the presence of a single character per underscore in a pattern. The LIKE operator is akin to the GREP utility available on Unix / Linux systems in terms of its ability to search strings.

Table 1: Boys' names
Troix
Trys
Troy
Troye
Tryst
Bryce

Table 2: Like Operator	
Like "Tr__"	Returns Trys and Troy
Like "Tro%"	Returns Troy and Troye
Like "Try%t"	Returns Tyrst
Like "_r%"	Returns Bryce, Troix, Trys, Troy, Troye and Tryst
Like "Tr___"	Returns Troix, Troye and Tryst

The like operator also includes an escape routine in case you need to use a string that includes a comparison operator such as the carat, the underscore or the percent sign, etc. An example of the escape routine syntax, when looking for a string containing a percent sign, is:

```
where yourvar like '100%' escape '%';
```

Additionally, SAS practitioners can use the NOT LIKE operator to select variables WITHOUT a given pattern. Please note that the LIKE statement is case sensitive. You can use the upcase / lowcase / propcase function to adjust input strings prior to using the LIKE statement. You may string multiple LIKE statements together with the AND or OR operators.

SOUNDS LIKE OPERATOR

The LIKE operator, described above, searches the actual spelling of operands to make a comparison. The SOUNDS LIKE operator uses phonetic values to determine whether character strings match a given pattern. As with the LIKE operator, the SOUNDS LIKE operator is useful for when there are misspellings and similar sounding names in strings to be compared. The SOUNDS LIKE operator is denoted with a short cut '-*'. SOUNDS LIKE is based on SAS's SOUNDEX algorithm. Strings are encoded by retaining the original first column, stripping all letters that are or act as vowels (A, E, H, I, O, U, W, Y), and then assigning numbers to groups: 1 includes B, F, P, and V; 2 includes C, G, J, K, Q, S, X, Z; 3 includes D and T; 4 includes L; 5 includes M and N; and 6 includes R. "Tristn" therefore becomes T6235, as does Tristan, Tristen, Tristian, and Tristin.

Table 3: Sounds Like Operator	
Trystan	T6235
Trysten	T6235
Trystian	T6235
Trystin	T6235
Trystn	T6235
Troye	T6

Sounds Like operator:

https://documentation.sas.com/?docsetId=sqlproc&docsetTarget=p0a62rd151ctown1x38ihdpjozyv.htm&docsetVersion=9.4&locale=en

JOINS WITH THE LIKE OPERATOR

It is possible to select records with the LIKE operator in PROC SQL with a where statement, including with joins. For example, the code below selects records from the SASHELP.ZIPCODE file that are in the state of Massachusetts and are for a city that begins with "SPR".

```
proc sql;
   CREATE TABLE TEMP1 AS
   select
          a.City ,
          a.countynm  , a.city2 ,
          a.statename , a.statename2
      from sashelp.zipcode as a
      where upcase(a.city) like 'SPR%' and
upcase(a.statename)='MASSACHUSETTS' ;
quit;
```

The test print of table TEMP1 shows only cases for Springfield, Massachusetts.

CITY	*COUNTYNM*	*CITY2*	*STATENAME*	*STATENAME2*
Springfield	Hampden	SPRINGFIELD	Massachusetts	MASSACHUSETTS
Springfield	Hampden	SPRINGFIELD	Massachusetts	MASSACHUSETTS
Springfield	Hampden	SPRINGFIELD	Massachusetts	MASSACHUSETTS
Springfield	Hampden	SPRINGFIELD	Massachusetts	MASSACHUSETTS
Springfield	Hampden	SPRINGFIELD	Massachusetts	MASSACHUSETTS

.The code below joins SASHELP.ZIPCODE and a copy of the same file with a renamed key column (city->geocity), again selecting records for the join that are in the state of Massachusetts and are for a city that begins with "SPR".

```
proc sql;
   CREATE TABLE TEMP2 AS
   select
          a.City , b.geocity,
          a.countynm  ,
          a.statename , b.statecode,
          a.x, a.y
      from sashelp.zipcode as a, zipcode2 as b
      where a.city = b.geocity and upcase(a.city) like 'SPR%' and b.statecode
= 'MA' ;
quit;
```

The test print of table TEMP2 shows only cases for Springfield, Massachusetts with additional variables from the joined file.

CITY	*geocity*	*COUNTYNM*	*STATENAME*	*STATECODE*	*X*	*Y*
Springfield	Springfield	Hampden	Massachusetts	MA	-72.589584	42.099922
Springfield	Springfield	Hampden	Massachusetts	MA	-72.589584	42.099922
Springfield	Springfield	Hampden	Massachusetts	MA	-72.538580	42.116054
Springfield	Springfield	Hampden	Massachusetts	MA	-72.538580	42.116054
Springfield	Springfield	Hampden	Massachusetts	MA	-72.538580	42.116054

THE LIKE "CONDITION"

The LIKE operator is sometimes referred to as a "condition", generally in reference to character comparisons where the prefix of a string is specified in a search. LIKE "conditions" are restricted to the data step, as the colon modifier is not supported in PROC SQL. The syntax for the LIKE "condition" is:

```
where firstname=: 'Tr';
```

This statement would select all first names in Table 2 above. To accomplish the same goal in PROC SQL, the LIKE operator can be used with a trailing % in a where statement.

PROC SQL CASE LIKE

The LIKE operator in PROC SQL can be used in conjunction with the CASE WHEN statement to create variables. The code snippets below feature using the LIKE operator with OR operators and CASE WHEN to create both a categorical variable (DXTYPE) and binaries (CANCERDX, ARTHDX, BCKPNDX, RHEUMDX, NONMSTDX, and OVERDOSE) from selected diagnosis codes with varying prefix lengths. *(Please note that the list of diagnosis codes for each category or binary is incomplete – this code snippet is for demonstration purposes only.)*

```
PROC SQL;
    CREATE TABLE out1 AS
    SELECT DYAD_OPIOID_ID,
           analelig,
           DIAG,
           CASE
           WHEN DIAG LIKE '14%'
           OR diag  LIKE 'D45%'
           THEN 'CANCERDX'
           WHEN diag  LIKE 'M00%'
           OR diag  LIKE '71%'
           THEN 'ARTHDX'
           WHEN DIAG LIKE '720%'
           OR diag  LIKE 'M6788%'
           THEN 'BCKPNDX'
           WHEN DIAG LIKE '725%'
           OR diag  LIKE 'R2989%' THEN 'RHEUMDX'
           WHEN DIAG LIKE '3382%'
           OR diag  LIKE 'T50%' THEN 'OVERDOSE'
```

```
           ELSE ''
           END AS DXTYPE,
           CASE
           WHEN DIAG LIKE '14%'
            OR diag  LIKE 'D45%'
           THEN 1
           ELSE 0
           END AS CANCERDX,
           CASE
           WHEN DIAG LIKE '71%'
           OR diag  LIKE 'M7964%' THEN 1
           ELSE 0
           END AS ARTHDX,
           CASE
           WHEN DIAG LIKE '720%'
           OR diag  LIKE 'M6788%' THEN 1
           ELSE 0
           END AS BCKPNDX,
           CASE
           WHEN DIAG LIKE '725%'
           OR diag  LIKE 'R2989%' THEN 1
           ELSE 0
           END AS RHEUMDX,
          CASE
           WHEN DIAG LIKE '3382%'
           OR diag like 'G89%'
           THEN 1
           ELSE 0
           END AS NONMSTDX,
            CASE
            WHEN DIAG LIKE'9650%'
            OR diag  LIKE 'T50%'
            THEN 1
            ELSE 0
            END AS OVERDOSE
            FROM in1
            ORDER BY dyad_opioid_id;
QUIT;
////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
PROC SQL;
    CREATE TABLE dd.flags AS
    SELECT distinct dyad_opioid_id
    ,analelig
     ,MAX(CANCERDXAS DXCANCER
     ,MAX(RHEUMDX) AS DXRHEUM
     ,MAX(BCKPNDX) AS DXBCKPN
     ,MAX(NONMSTDX) AS DXNONMST
     ,MAX(ARTHDX) AS DXARTH
      ,MAX(OVERDOSE) AS DXOVERDOSE
     FROM out1
      WHERE analelig=1
     GROUP BY dyad_opioid_id
     ORDER BY dyad_opioid_id;
QUIT;
////////////////////////////////////////////////////////////////////////////////////////////////////
```

PROC SQL CREATE TABLE LIKE

There are several methods of creating an empty data set. PROC SQL CREATE TABLE LIKE is one of the most efficient ones when creating a shell from an existin g data set, as it automatically uses the metadata from the existing SAS data set. The code below takes advantage of the fact that OPTIONS DLCREATEDIR enables the creation of a library / folder if one does not already exist (DLCREATEDIR is benign, and will not overwrite existing folders or data within

those folders). The CREATE TABLE LIKE method is NOT benign, and will overwrite an existing data set. Therefore, the program uses conditional logic and %SYSFUNC to determine if the file to be written already exists, and does not overwrite the file, instead producing an error note in the log.

```
Options dlcreatedir;
run;

/* set years for old and new files */

%let prevyear=2018;
%let fileyear=2019;

%let base=S:\Projects\yourproject\;

libname prevann    "&base.Analysis&prevyear";
libname annual "&base.Analysis&fileyear";

/*******************************************
Program:   Writeemanddontweep.sas
Purpose:   (1) Build empty shell for annual longitudinal file with
  ratings data
           (2) Build empty shell for annual longitudinal file with
  Health
               Inspection Cutpoints

NOTE: Analysis2019 folder / library automatically (and benignly)
created using DLCREATEDIR option

Input Data:
(1) annual longitudinal provider/month ratings FOR PRIOR YEAR:
    allratingsYYYY_long.sas7bdat
(2) annual longitudinal (state/month- HI cutpoints FOR PRIOR YEAR:
(3) allHIcuts<yyyy>_long.sas7bdat

Output Data:
(1) EMPTY SHELL of annual longitudinal provider/month ratings:
     allratingsYYYY_long.sas7bdat
(2) EMPTY SHELL of annual longitudinal (state/month- HI cutpoints
     allHIcuts<yyyy>_long.sas7bdat

 *******************************************/
run;

title1 "Compile Annual Longitudinal files for rating and health
inspection cutpoints" ;
footnote1 "%sysfunc(getoption(sysin)) - &sysdate - &systime";
run;
```

```
** ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ** ;
** Part 1: Ratings
** ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ** ;

title2 'Ratings';

  %if %sysfunc(exist(annual.allratings&fileyear._long))=0 %then
  %do;

  PROC SQL;
    CREATE TABLE annual.allratings&fileyear._long
            LIKE prevann.allratings&prevyear._long;
    DESCRIBE TABLE annual.allratings&fileyear._long;
  QUIT;

  %end;

  %else %do;

  %put ERROR: Data Set ALLRATINGS&FILEYEAR._LONG already exists!;

  run;

  %end;
```

CONCLUSION

SAS provides practitioners with several useful techniques using LIKE statements: the smooth LIKE OPERATOR / CONDITION in both the data step and PROC SQL, CASE WHEN LIKE in PROC SQL, and CREATE TABLE LIKE in PROC SQL. There's definitely reason to like LIKE in SAS programming.

REFERENCES

Gilsen, Bruce. September 2001. "SAS® Program Efficiency for Beginners." *Proceedings of the Northeast SAS Users Group Conference*, Baltimore, MD.

Roesch, Amanda. September 2011. "Matching Data Using Sounds-Like Operators and SAS® Compare Functions." *Proceedings of the Northeast SAS Users Group Conference*, Portland, ME.

Shankar, Charu. June 2019. "The Shape of SAS® Code." *Proceedings of PharmaSUG 2019 Conference*, Philadelphia, PA..

CONTACT INFORMATION

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

Louise S. Hadden
Abt Associates Inc.
617-349-2385
Louise_hadden@abtassoc.com
abtassociates.com

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.