PharmaSUG 2021 - Paper QT-098

Wildcarding in Where Clauses

David B. Horvath, MS, CCP

ABSTRACT

This mini-paper (and the resulting short session) is a focused discussion of wildcarding within the Where clause and related functionality within SAS®. There is wildcarding allowed within where clause like expressions by using special characters. But if you want to search for those specific characters, you have to override them. If you forget to escape the character, you'll get unexpected matches.

Topics include:

- proc sql; select x from y where x like %a/_b escape /;
- _ matches 1 character
- % matches any characters
- / escapes the next character (says next is actual character, not special meaning)
- set sorted_ddl (obs=max where=(name like '%/_ORD%' escape '/'));

INTRODUCTION

Wildcarding within the where clause has been available within SQL since the 1980s. SAS has supported that capability for almost that long. In addition to use within the where clause, SAS provides additional methods of wildcard text searches.

WHERE CLAUSE WILDCARDING BASICS

The general format of wildcarding within the where clause is:

```
Where variable like "%val%" escape "/";
```

In this case, '%' matches zero to many characters while '_' matches to exactly one character. The 'escape' clause is optional (and is normally set to '/'). Note that there is no default escape character within SAS. If you want to match to a '%' or '_' within the string, you will need to escape them by prefixing with the escape character:

```
Where variable like "%string/_with/_underscore" escape "/";
```

YOU MUST USE CORRECT SYNTAX

You do have to be careful when coding escape sequences. For instance if you make a mistake like:

```
Where product_group like "%/%" escape "%";
```

Or

```
Where product_group like "%Kid/s%" escape "/";
```

You'll see

```
ERROR: Like pattern is not a valid pattern due to the invalid use of the
escape pattern.
```

USABILITY

The 'Like' statement has a lot of functionality. Of course, it can be used in PROC SQL. Because it is part of ANSI SQL, it can be passed through to your database engine. Why do you care? Because the search will take place on the database server with only resulting rows being returned – not tying up the network or SAS Server for the matching.

In addition, the database may make use of any relevant indexes. Typically, the index will be scanned based on the string up to the first %, then the result filtered based on the rest of the string. And, importantly, processing load is usually better than a full table scan with filter

'Like' can be used in DATA step 'where' as well. Both stand-alone and as part of the set statement are allowed. It will not work as part of an 'if' statement though.

One nice feature is that SAS Indexes will be used whenever possible – DATA and PROC SQL both!

SOME EXAMPLES

Given the Initial List, the following are a series of strings in a like clause and the matching result. The best way to understand it is to review the list, the like string, and which examples actually match.

Initial List
Diana
Diane
Dianna
Dianthus
Dyan
David
Dav%id
Dav_id
Dav/id
Dav/%id

Table 1 List of Search Candidates

like_string	Result
David	David
like_string	name
Dav%	David
Dav%	Dav%id
Dav%	Dav_id
Dav%	Dav/id
Dav%	Dav/%id
like_string	name
Dav_%	David
Dav_%	Dav%id
Dav_%	Dav_id
Dav_%	Dav/id
Dav_%	Dav/%id
like_string	name
Dav/%	Dav/id
Dav/%	Dav/%id
like_string	name
Dav/%id	Dav/id
Dav/%id	Dav/%id

Table 2 First Set of Results

like_string	name
D_a%	Diana
D_a%	Diane
D_a%	Dianna
D_a%	Dianthus
D_a%	Dyan
like_string	name
%a	Diana
%a	Dianna
like_string	name
%nn%	Dianna
like_string	name
Dav/% escape	NONE
like_string	name
Dav_	NONE

Table 3 Second Set of Results

like_string	name
Dav/%id escape	Dav%id
like_string	name
Dav/%% escape	Dav%id
like_string	name
Dav/id	Dav/id
like_string	name
Dav//id escape	Dav/id
like_string	name
Dia__	Diana
Dia__	Diane
like_string	name
Di% or Dy%	Diana
Di% or Dy%	Diane
Di% or Dy%	Dianna
Di% or Dy%	Dianthus
Di% or Dy%	Dyan

Table 4 Third Set of Results

There are many places you can use the like clause itself within SAS. Some code examples include:
Like can be used in many contexts

It can be used in any data= where clause (with or without an escape):

```
proc print data=list (where=(name like 'David')); run;

proc print data=list (where=(name like 'Dav//id' escape '/')); run;
```

It can be used on the set where clause in a data statement. Escape is optional – you only need to include it if you need to escape a value (in this case, searching for the string “Dav%” followed by any other characters.

```
data list2;
      set list (where=(name like 'Dav/%%' escape '/'));
run;
```

And, it can be used as a standalone where.

```
data list2;
      set list;
      where name like 'Dav/%%' escape '/';
run;
```

ALTERNATIVES TO ‘LIKE’

SAS has multiple alternatives to using ‘like’. Some provide even more flexibility.

Within a where clause, you can use ‘contains’ which is equivalent to a like “%string%” statement. As with ‘like’, it will use SAS indexes but it is unlikely to pass through to a database engine.

The prxmatch() function allows the use of extremely complex search strings coded as Regular Expressions. Although this function provides extreme flexibility, it will make use of SAS Indexes nor pass through to a database engine.

There is also the limited functionality with EQ:. The following if/eq: is equivalent to name like “Dav/%%” escape “/”. It matches characters in name up to the length of the comparison string (effectively wildcarding everything to the right.

```
if name eq: 'Dav%' ;
```

The importance of SAS Indexes are that they allow SAS to search for strings by reading through the highly search-efficient structure of the index rather than reading in every field on every record. The importance of database pass-through is that the searches take place on the database engine and only the matching rows are returned to the SAS server. The advantage of making use of database indexes is similar to that of SAS – more efficient than reading raw data rows.

CAUTION

In any commands in this paper, the single and double quotation marks should be simple, not the “smart quotes” forced my Microsoft. The same applies to dashes or minus signs – they should not be “em dashes” (- versus –)

You can safely ignore this warning:

```
where product_group like "%Kid's%";
WARNING: Apparent invocation of macro KID not resolved.
```

Because of the ‘%’ one part of the SAS interpreter thinks there should be a macro here.

CONCLUSION

SAS provides many capabilities for string searching within fields that allow for wildcarding or fuzzy matches. ‘like’ is just one of many!

REFERENCES

SAS® 9.4 SQL Procedure User’s Guide, Fourth Edition:
https://documentation.sas.com/?docsetId=acreldb&docsetTarget=n05b4mygsvt845n1vnr6r5kchbjf.htm&docsetVersion=9.4&locale=en

The Basics of Using SAS® Indexes
https://support.sas.com/resources/papers/proceedings/proceedings/sugi30/247-30.pdf

ACKNOWLEDGMENTS

I want to thank the organizers of this great conference – I would have preferred to present in person but certainly understand the need for the conference to be virtual. I will miss the interaction with other speakers and attendees. I also want to thank my employer for their past willingness to allow me to expand my horizons by attending conferences. Lastly but certainly not least, I want to thank my spouse Mary who doesn’t complain when I spend so much time at the keyboard working on documents like this.

CONTACT INFORMATION

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

David B. Horvath, CCP
+1-610-859-8826
dhorvath@cobs.com
http://www.cobs.com
LinkedIn: https://www.linkedin.com/in/dbhorvath/