PharmaSUG 2013 - Poster PO12

# A SAS® Users Guide to Regular Expressions When the Data Resides in Oracle

Kunal Agnihotri, PPD, Inc., Morrisville, NC
Kenneth W. Borowiak, PPD, Inc., Morrisville, NC

## ABSTRACT

The popularity of the PRX functions and call routines has grown since they were introduced in SAS Version 9 due to the tremendous power they provide for matching patterns of text. Since the implementation of the regular expressions within these functions is rooted in the Perl-style syntax, there is portability outside of a SAS environment. It is not uncommon for SAS users to access data residing in an Oracle database. This paper explores the Oracle implementation of regular expressions by highlighting similarities and differences to the PRX implementation in series of queries using the PROC SQL Pass-Through facility against Oracle system tables.

## INTRODUCTION

A *regular expression* is a string that characterizes a pattern for matching and subsequent manipulation of text fields. The popularity of the PRX functions and call routines has grown since they were introduced in SAS Version 9 due to the tremendous power they provide for matching patterns of text. Those unfamiliar with PRX should refer to introductory papers by Borowiak [2008] and Cassell [2007], as concepts discussed there are used throughout this paper.

SAS users who want leverage the power of the Perl-style regular expressions via the PRX functions and call routines in their SAS environment have options when the data they want to access resides in an Oracle database[1]. They can access the data by establishing a pointer to the tables with a LIBNAME statement and using familiar PRX code. However, this results in pulling some, if not all, of the tables and records into the local SAS workspace so it can process the PRX statements. Another option is to the use the PROC SQL Pass-Through facility, where statements are processed in the Oracle space in its native language, and the resulting records brought into the SAS environment. While the syntax of the Oracle regular expressions is similar to that of PRX, there are differences in the way they are invoked. This paper explores the Oracle implementation of regular expressions, drawing on the similarities and differences to PRX using queries written against the Oracle system tables[2].
The implementation of regular expressions in Oracle began with their 10g release and can be characterized as following the POSIX Extended Regular Expressions (ERE) standard. To determine if your Oracle connection meets this requirement you can issue the SQL Pass-Through query in Figure 1.

[1] This requires licensing SAS/ACCESS to Oracle.

[2] Oracle system tables store metadata and are analogous to the SAS Dictionary tables and SASHELP views.

Figure 1 - Query to confirm version of the Oracle client

```
PROC SQL  _method feedback ;
    connect to oracle( path=&lesstaken. user=&me. pw=&myob. ) ;
     select *
     from    connection to oracle
             (  select    *
                from      v$version
                where     regexp_like( banner, 'oracle', 'i')  )
      ;
     disconnect from oracle ;
 quit ;
```

| BANNER |
| --- |
| Oracle Database 11g Enterprise Edition Release 11.2.0.2.0 - 64bit Production |

The values of the user id, password and path needed to make the connection to the Oracle tables will be embedded in the macro variables *me, myob*, and *lesstaken*, respectively. The actual query passed to the Oracle environment is found in the subquery embedded in the FROM clause of the PROC SQL statement.

## REGEXP_LIKE

The query in Figure 1 makes use of a regular expression to do a case-insensitive static string search via the REGEXP_LIKE function. This function is most closely related to the PRXMATCH function in SAS. The REGEXP_LIKE function returns a Boolean indicating whether the pattern matched in a given source string or not and takes the following form:

**REGEXP_LIKE (source_string , pattern ,  modifier)**

• source_string - the source where the pattern is to be searched.
• pattern – the regular expression.
• modifier – this changes the default behavior of the function while trying to match the pattern.

The arguments in the REGEXP_LIKE function will be common to all of the remaining REGEXP functions examined in this paper, although the location of the MODIFIER may vary due to the other functions having more arguments.

You may notice the following differences of REGEXP_LIKE vis-a-vis PRXMATCH:

- The source is listed in the first argument
- The pattern matching regular expression is in the second argument, rather than in the first argument.
- The regular expression is not embedded within a pair of delimiters.
- The regular expression modifiers are specified in the third argument, rather than directly following the delimiters that encapsulate the pattern.
- It does not provide information on where the match occurred.

Much of the regular expression syntax of PRX familiar to the SAS user is valid in REGEXP_LIKE and the other REGEXP functions. This will be demonstrated in the query in Figure 2 and throughout the paper.

Figure 2 - Portability of regular expression syntax between SAS and Oracle implementations

```
PROC SQL  _method feedback ;
    connect to oracle( path=&lesstaken. user=&me. pw=&myob. ) ;
        select   *
        from     connection to oracle
                 ( select  column_name
                   from    sys.all_ind_columns
                   where   regexp_like(column_name, '^COORD_(AXIS|SYS)','c') )
        ;
        disconnect from oracle  ;
 quit ;
```

| COLUMN_NAME |
|---|
| COORD_SYS_ID |
| COORD_AXIS_NAME |
| COORD_AXIS_NAME_ID |
| COORD_SYS_TYPE |
| COORD_SYS_NAME |
| COORD_SYS_ID |

The beginning of the string anchor ^ is used to match the text COORD_. This is followed by a pair of grouping parentheses containing uses a pipe separator for *alternation*, which acts like an OR logical operator, to match the text AXIS or SYS. Although the default setting is for the regex to be case-sensitive, this modifier argument can be explicitly set to c. This modifier value is not valid in PRX statements. In the event that both of the i and c modifiers are specified in the call then the one that appears last will be honored.

## REGEXP_INSTR

The Oracle function that is most closely related to the PRXMATCH function is REGEXP_INSTR, where the default behavior is to return the location of the match. It can give information on the beginning or ending position of the match depending on the return_option argument. The syntax for this function is similar to the REGEXP_LIKE function, but has three additional optional arguments:

**REGEXP_INSTR (source_string,pattern , position , occurrence,return_option, modifier)**

- position - Location in the string to begin the pattern search. The default value is 1 and can be omitted if the match is to performed from position 1.
- occurrence - Identify which occurrence of the pattern in the source to match. The default value is 1 and can be omitted if the match is looking for the first occurrence.
- Return_option - Determines if it should return the position of the first character in the match or the last character. The default value is 0 which gives the position of the first character of the occurrence. A value of 1 returns the position that follows the last character in the occurrence.

The example in Figure 3 returns the starting position where the TABLE_NAME field ends with an underscore followed by one or more digits.

Figure 3 - Find location where the pattern match begins

```
PROC SQL  _method feedback ;
    connect to oracle( path=&lesstaken. user=&me. pw=&myob. ) ;
     select   *
     from    connection to oracle
             (  select   table_name
                       , regexp_instr( table_name, '_\d+$') match_pos
                from     sys.all_tables
               where    regexp_like( table_name, '_\d+$')   )
      ;
      disconnect from oracle ;
 quit ;
```

| TABLE_NAME | MATCH_POS |
|---|---|
| KU$_LIST_FILTER_TEMP_2 | 21 |
| KU$_DATAPUMP_MASTER_11_2 | 23 |
| KU$_DATAPUMP_MASTER_11_1_0_7 | 27 |
| KU$_DATAPUMP_MASTER_11_1 | 23 |
| KU$_DATAPUMP_MASTER_10_1 | 23 |

The end of the string anchor $ in Oracle can be used to match the last non-whitespace character. This differs from PRXMATCH, where a functionally equivalent regular expression would need to account for any whitespace to fill up the allotted character bytes ( e.g. prxmatch( ‘m/_\d+**\s***$’, table_name) ).

The query in Figure 4 below is similar to that in Figure 3, except the ending position of the first occurrence is sought. Since the fifth argument controls this behavior with a value of 1, the optional third and fourth arguments must also be provided in the function call. Since the function will return the position following the end of the match the result will need to be decremented by 1.

Figure 4 - Find location where the pattern match ends

```
PROC SQL  _method feedback ;
   connect to oracle( path=&lesstaken. user=&me. pw=&myob. ) ;
    select   *
    from     connection to oracle
             (  select   table_name
                       , regexp_instr( table_name, '_\d+$', 1, 1, 1 )-1
                           end_match_pos
                from     sys.all_tables
                where    regexp_like( table_name, '_\d+$')   )
          disconnect from oracle
    ;
quit ;
```

| TABLE_NAME | END_MATCH_POS |
|---|---|
| KU$_LIST_FILTER_TEMP_2 | 22 |
| KU$_DATAPUMP_MASTER_11_2 | 24 |
| KU$_DATAPUMP_MASTER_11_1_0_7 | 28 |
| KU$_DATAPUMP_MASTER_11_1 | 24 |
| KU$_DATAPUMP_MASTER_10_1 | 24 |

## REGEXP_SUBSTR

Another powerful function available to SAS programmers from the regexp arsenal is REGEXP_SUBSTR. This goes a step further from REGEXP_LIKE and gives the user a substring from the given source string. Apart from the arguments present in REGEXP_LIKE, two more arguments make up REGEXP_SUBSTR:

**REGEXP_SUBSTR(source_string , pattern , position, occurrence , modifier )**

- position - A positive integer indicating where the operator should begin the search. The default is 1.
- occurrence – This indicates which occurrence the operator should look for in the source string. The function looks for the regular expression from the first position in the source string. The default is 1.

The example in Figure 5 returns the third occurrence of a substring where the COLUMN_NAME field has text embedded in a pair of quotation marks.

Figure 5 - Query to extract a substring from a source

```
PROC SQL  _method feedback ;
    connect to oracle( path=&lesstaken. user=&me. pw=&myob. ) ;
     select   *
     from    connection to oracle
               (  select  column_name
                        , regexp_substr(column_name, '"[^"]+"' ,1 ,3 ,'i')
                             match_str
                   from    sys.all_ind_columns
                    where   regexp_like(column_name,'"[^"]+"', 'i')  )
      ;
     disconnect from oracle ;
     quit ;
```

| COLUMN_NAME | MATCH_STR |
|---|---|
| "XMLDATA"."SCHEMA_URL" | |
| "XMLDATA"."PROPERTY"."GLOBAL" | "GLOBAL" |
| "XMLDATA"."PROPERTY"."NAME" | "NAME" |
| "XMLDATA"."PROPERTY"."PROP_NUMBER" | "PROP_NUMBER" |

The regular expression in this function call looks for " in the source string followed by a non-double quote, defined by the *negated character class* [^"], then followed by a " . The + *quantifier* is used to search for multiple non-quote occurrences of the character class in the source string. Also take notice that the first result for MATCH_STR is blank as there is no third occurrence of the regular expression in the source string. The ability to extract the nth occurrence in the string with a single call to REGEXP_SUBSTR is an advantage over PRX, which usually takes 2 or 3 functions call (e.g PRXPARSE-PRXMATCH-PRXPOSN[3], CALL PRXSUBSTR-SUBSTR, or PRXPARSE-CALL PRXNEXT-SUBSTR in a DO loop ).

## REGEXP_REPLACE

The last function in the REGEXP family is REGEXP_REPLACE, which is analogous to PRXCHANGE. This function looks for the occurrence of a regular expression in the source string and replaced with literal text or text from a capture buffer. The form of the REGEXP_REPLACE is:

**REGEXP_REPLACE (source_string, pattern, replacement, position, occurrence, modifier)**

In the example in Figure 6 the REGEXP_REPLACE function is used to replace digits and the $ sign at the end of the source string with literal text.

[3] An ambitious PRXer could create a user-defined function with PROC FCMP to encapsulate these calls, which would then mimic the functionality REGEXP_SUBSTR.

Figure 6 - Query to replace a regular expression with literal text

```
proc sql  _method feedback ;
       connect to oracle( path=&lesstaken. user=&me. pw=&myob. ) ;
        select   *
        from     connection to oracle
                 (  select   column_name
                         ,  regexp_replace(column_name, '\d{3,4}\$$','_TABLE')
                              mat_replace
                    from     sys.all_ind_columns
                    where regexp_like(column_name,'\d{3,4}\$$','i')
                 )
        ;
       disconnect from oracle ;
      quit ;
```

\d in the above example is used to match any digits characters (0 through 9) in the source string. The *quantifier* {3,4} succeeding \d indicates the function to look for at least 3 digits but to a maximum of 4. This is followed by \$, an escaped reference made to match the $ as literal text in the source. And finally the $ which asks the function to look for the regular expression at the end of the source. The \ differentiates between the 2 $ signs - one as literal text and the other as the match the end of line character. The matched digits followed by the $ are then replaced by the string _TABLE in the new variable MAT_REPLACE.

| COLUMN_NAME | MAT_REPLACE |
|---|---|
| SYS_NC00004$ | SYS_NC0_TABLE |
| SYS_NC00007$ | SYS_NC0_TABLE |
| SYS_NC00006$ | SYS_NC0_TABLE |
| SYS_NC00075$ | SYS_NC0_TABLE |
| SYS_NC00179$ | SYS_NC0_TABLE |

## METACHARACTERS AND CHARACTER CLASSES

Figure 7 below shows some of the other common syntax for character classes and metacharacters between the PRX or Oracle implementation of regular expressions that were not used any of the previous examples.

Figure 7 - Other Common Character Classes and Metacharacter

| Operator | Description |
|---|---|
| * | Matches 0 or more occurrences of the preceding expression |
| ? | Matches 0 or 1 occurrences of the preceding expression |
| \d | Digit characters, equivalent to [0-9]. There appears to be a bug where \d is not honored when used within a user-defined character class. |
| \D | Non-digit characters |
| [:digit:] | POSIX bracketed expression for matching digit characters - equivalent to \d |
| [:alpha:] | POSIX bracketed expression for matching alphabet characters - equivalent to [a-zA-Z] |
| \b | Word boundary |
| \B | Non-word boundary |
| \s | Any whitespace character, including tabs |
| (?: ) | Non-capturing buffers **are not** supported in REGEXP functions |
| (?=), (?!), (?<=), (?<! ) | Positive and negative look-arounds **are not** supported in REGEXP functions |

## CONCLUSION

The queries written in this paper use very basic and simple examples of the REGEXP functions in Oracle with PROC SQL Pass-Through facility. The user is encouraged to mix and match these functions and make use of the plethora of combinations that can be made up with the use of metacharacters at their disposal. It does take some time to familiarize oneself with the various options at hand, but is more efficient than other options available when querying for data residing in Oracle. PRX users will definitely recognize the similarities and differences with the REGEXP functions and welcome this new addition to their knowledge base.

## RERERENCES

Borowiak, Kenneth W. (2004), “Effectively Using the Indices in an Oracle® Database with SAS®”. Proceedings of the Seventeenth Annual Northeast SAS Users Group Conference, USA.
http:/www.nesug.org/proceedings/nesug04/po/po12.pdf

Borowiak, Kenneth W. (2012), “PRXCHANGE: Accept No Substitutions”. Proceedings of the 21st Annual Southeast SAS Users Group Conference, USA.

Borowiak, Kenneth W. (2008), “PRX Functions and Call Routines: There is Hardly Anything Regular About Them!”. Proceedings of the Twenty First Annual Northeast SAS Users Group Conference, USA.
http://www.nesug.org/proceedings/nesug08/bb/bb11.pdf

Cassell, David L., ”The Basics of the PRX Functions” SAS Global Forum 2007
http://www2.sas.com/proceedings/forum2007/223-2007.pdf

Friedl, Jeffrey E.F., *Mastering Regular Expressions 3rd Edition*

Introducing Oracle Regular Expressions. An Oracle® White Paper. ORACLE®. September 2003.
http://www.oracle.com/technetwork/database/focus-areas/application-development/twp-regular-expressions-133133.pdf

## ACKNOWLEDGEMENTS

The authors would like to thank Jenni Borowiak and Jim Worley for their insightful comments on this paper.



## DISCLAIMER

The content of this paper are the works of the authors and do not necessarily represent the opinions, recommendations, or practices of PPD, Inc.

## CONTACT INFORMATION

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

Kunal Agnihotri
3900 Paramount Parkway
Morrisville NC 27560

**kunal.agnihotri@ppdi.com**
**agnihotrikunal@gmail.com**

Ken Borowiak
3900 Paramount Parkway
Morrisville NC 27560

**ken.borowiak@ppdi.com**
**ken.borowiak@gmail.com**