Using Regular Expressions in Oracle

Everyday most of us deal with multiple string functions in Sql. May it be for truncating a string, searching for a substring or locating the presence of special characters.

The *regexp* functions available in Oracle 10g can help us achieve the above tasks in a simpler and faster way.

I have tried to illustrate the behavior of the **regexp** functions with common patterns and description of each. This document mainly focuses on the usage of patterns. The patterns can be used with any of the regular expression functions. But before we start let me include little <u>oracle documentation</u>.

I hope the documentation is good enough to start with. Okay let's start with our first case.

1. Validate a string for alphabets only

```
select case when
    regexp_like('Google' ,'^[[:alpha:]]{6}$')
        then 'Match Found'
    else 'No Match Found'
    end
as output from dual;
```

Output: Match Found

In the above example we tried to check if the source string contains only 6 alphabets and hence we got a match for "Google".

Now let's try to understand the pattern '^[[:alpha:]]{6}\$'

^ marks the start of the string[[:alpha:]] specifies alphabet class{6} specifies the number of alphabets\$ marks the end of the string

2. Validate a string for lower case alphabets only

```
select case when
    regexp_like('Terminator' ,'^([[:lower:]]{3,12})$')
        then 'Match Found'
    else 'No Match Found'
    end
as output from dual;
```

Output: No Match Found

If we change the input to "terminator", we get a match.

In the above example we tried to check if the source string contains only lower case alphabets with a string length varying from minimum of 3 to max of 12.

The pattern is similar as above but **{3,12}**.

{3,12} specifies at least 3 alphabets but not more than 12

3. Case Sensitive Search

```
select case when
    regexp_like('Republic Of India' ,'of', 'c')
        then 'Match Found'
    else 'No Match Found'
    end
as output from dual;
```

Output: No Match Found

"c" stands for case sensitive

In the above example we tried to search for the pattern ``of'' exactly matching the case and hence we didn't find a match

Let's now try a search for "of" in the source string irrespective of the case in the below example:

```
select case when
    regexp_like('Republic Of India' ,'of','i')
        then 'Match Found'
    else 'No Match Found'
    end
as output from dual;
```

Output: Match Found

"i" stands for case insensitive

4. Match nth character

```
select case when
    regexp_like('ter*minator' ,'^...[^[:alnum:]]')
        then 'Match Found'
        else 'No Match Found'
        end
        as output from dual;
```

Output: Match Found

In the above example we tried to search for a special character at the 4^{th} position of the input string "ter*minator"

Let's now try to understand the pattern '^...[^[:alnum:]]'

^ marks the start of the string
. a dot signifies any character (... 3 dots specify any three characters)
[^[:alnum:]] Non alpha-numeric characters (^ inside brackets specifies negation)

Note: The \$ is missing in the pattern. It's because we are not concerned beyond the 4th character and hence we need not mark the end of the string. (...[^[:alnum:]]\$ would mean any three characters followed by a special character and no characters beyond the special character)

5. Search and replace white spaces

select regexp_replace('Help Earth Stay Green','[[:blank:]]{2,8}',' ')
from dual;

Output: Help Earth Stay Green

In the above example we have replaced multiple white spaces (2 to 8) in the source string with single space.

Let's now try to understand the pattern [[:blank:]]{2,8}

Now that you are familiar with the patterns you may be wondering about the missing (^\$) anchors in the pattern. These have been skipped purposefully. In the source string we wanted to search for multiple spaces anywhere in the string. So we need not specify the start (^) and end (\$) anchors.

Note: $^{[:blank:]}{2,8}$ would mean a pattern consisting of min of 2 spaces to a max of 8 and no other characters and we won't find a match for the above input.

6. Search for control characters

```
select case when
    regexp_like('Super' || chr(13) || 'Star' ,'[[:cntrl:]]')
        then 'Match Found'
    else 'No Match Found'
    end
as output from dual;
```

Output: Match Found

In the above example we tried to search for presence of carriage return in our source string (13 is the ASCII value for carriage return).

7. Extract sub strings from source string

i) select regexp_substr('1PSN/231_3253/ABc','[[:alnum:]]+') from dual;

Output: 1PSN

[[:alnum:]]+ One or more number of alphanumeric characters (The "+" sign stands for one or more occurrences)

Note: I didn't include the "^" and "\$" anchors as I wanted to search for the matching pattern anywhere in the source string. By default the first matching pattern is returned.

ii) select regexp_substr('1PSN/231_3253/ABc','[[:alnum:]]+',1,2) from dual;

Output: 231

Note the extra parameters in the expression compared to the first example.

"1" specifies that the search needs to start from the first character in the source string

"2" specifies the second occurrence of the matching pattern which is 231 in the source string "1PSN/231_3253/ABc"

iii) select regexp_substr('@@/231_3253/ABc','@*[[:alnum:]]+') from dual;

Output: 231

@* Search for zero or more occurrences of @ ("*" stands for zero or more occurrences)

[[:alnum:]]+ followed by one or more occurrences of alphanumeric characters

Note: In the above example oracle looks for @ (zero times or more) immediately followed by alphanumeric characters.

Since a '/' comes between @ and 231 the output is: 0 occurrences of @ + one or more occurrences of alphanumeric characters.

iv) select regexp_substr('1@/231_3253/ABc','@+[[:alnum:]]*') from dual;

Output: @

@+ one or more occurrences of @
[[:alnum:]]* followed by 0 or more occurrences of alphanumeric characters

v) select regexp_substr('1@/231_3253/ABc','@+[[:alnum:]]+') from dual;

Output: Null

@+ one or more occurrences of @
[[:alnum:]]+ followed by one or more occurrences of alphanumeric characters

In the above example, there is no matching pattern in the source string and hence the output is null.

vi) select regexp_substr('@1PSN/231_3253/ABc125','[[:digit:]]+\$') from dual;

Output: 125

[[:digit:]]+ one or more occurrences of digits only
\$ at the end of the string

In the above example we tried to extract the digits at the end of the source string.

vii) select regexp_substr('@1PSN/231_3253/ABc','[^[:digit:]]+\$') from dual;

Output: /ABc

[^[:digit:]]+\$ one or more occurrences of non-digit literals at the end of the string *Note:* "^" *inside square brackets marks the negation of the class*

viii) select regexp_substr('Tom_Kyte@oracle.com','[^@]+') from dual;

Output: Tom_Kyte

[^@]+ one or more occurrences of characters which are not "@"

The above example extracts the name part from email address.

ix) select regexp_substr('1PSN/231_3253/ABc','[[:alnum:]]*',1,2) from dual;

Output: Null

[[:alnum:]]* zero or more number of alphanumeric characters

We looked for the second occurrence of alpha numeric characters in the source string which is 231. Don't we find the output misleading here? In fact it isn't.

If you carefully look at the pattern, it says; second occurrence of zero **or** more number of alphanumeric characters. The word "or" is stressed here. We have a "/" followed by 1PSN. This accounts for zero occurrence of alphanumeric characters and hence the output "Null".

Now let's move to our next case.

8. Validate email

```
select case when
REGEXP_LIKE('tom_kyte@oracle.com',
```

```
'^([[:alnum:]]+(_?|\.))[[:alnum:]]*@[[:alnum:]]+(\.([[:alnum:]]+)){1,2}$') then
'Match Found'
    else 'No Match Found'
    end
as output from dual;
```

Output: Match Found

Let's try to understand the pattern
^([[:alnum:]]+(_?|\.))[[:alnum:]]*@[[:alnum:]]+(\.([[:alnum:]]+)){1,2}\$
in parts.

You are aware that the "^" marks the start of the string. Now let's break ([[:alnum:]]+(_?|\.)) into two parts: [[:alnum:]]+ and (_?|\.)

[[:alnum:]]+ represents one or more occurrences of alphanumeric characters
(_?|\.) represents zero or one occurrence of underscore or dot. The "?" symbol here
stands for zero or one occurrence; and "|" symbol represents OR. The expression
(_?|\.) represents for a sub expression as a whole representing an optional
underscore or dot. The parentheses help in grouping expressions together to
represent one sub expression.

So now, ([[:alnum:]]+(_?|\.)) as a whole represents one or more occurrences of alphanumeric characters optionally followed by an underscore or dot.

Now that we have understood sub expressions we can move ahead with the rest of the pattern.

[[:alnum:]]* followed by zero or more occurrences of alphanumeric characters @ followed by @

[[:alnum:]]+ followed by one or more occurrences of alphanumeric characters
(\.([[:alnum:]]+)){1,2} followed by the sub expression: dot followed by one or more
occurrences of alphanumeric characters; from one to a max of two occurrences of the
sub expression (*Example: .com or .co.in*)

Input: tom.kyte@oracle.com Output: Match Found

Input: tom-kyte@oracle.co.uk Output: No Match Found *Note: Did you notice the backslash before the dot? The backslash here is used as escape sequence for dot. Take extra care while testing patterns that include dot literals, as a Dot (.) alone stands for any character*

9. Validate SSN

```
select case when
    regexp_like('987-65-4321' ,'^[0-9]{3}-[0-9]{2}-[0-9]{4}$')
    then 'Match Found'
    else 'No Match Found'
    end
as output from dual;
```

Output: Match Found

Input: 987-654-3210 Output: No match found

start of the string
[0-9]{3} three occurrences of digits from 0-9
followed by hyphen
[0-9]{2} two occurrences of digits from 0-9
followed by hyphen
[0-9]{4} four occurrences of digits from 0-9
end of the string

The above pattern can also be used to validate phone numbers with little customization.

10.Consecutive Occurrences

i) Let's try to search for two consecutive occurrences of letters from a-z in the following example.

select regexp_substr('Republicc Of Africaa' ,'([a-z])\1', 1,1,'i') from dual;

Output: cc

([a-z]) character set a-z
\1 consecutive occurrence of any character in the class [a-z]
1 starting from 1st character in the string
1 First occurrence
i stands for case insensitive

ii) Now let's try to search for three consecutive occurrences digits from 6 to 9 in the following example.

select case when
 regexp_like('Patch 10888 applied' ,'([6-9])\1\1')
 then 'Match Found'
 else 'No Match Found'
 end
as output from dual;

Output: Match Found

Note: I didn't specify the position or occurrence in the above sql query. By default, oracle will search for the first occurrence of the matching pattern from the start of the string and will stop when it encounters appropriate match. If no match is found it stops at the end of the string

11.Formatting Strings

FROM dual;

Output: (040) 9966-1234

We tried to format a phone number in the above example. Let's understand the match pattern and replacing string.

Our match pattern is ^([[:digit:]]{3})([[:digit:]]{4})\$

([[:digit:]]{3}) 3 digits
([[:digit:]]{4}) followed by 4 digits
([[:digit:]]{4}) followed by 4 digits

Why did I group the digits into sub expressions using the parentheses? I could have simply searched for [[:digit:]]{11} as my input string comprises of 11 digits only. To understand the reason, let's look at the replacing string (\1) 2-3

- (includes a opening parenthesis
- \1 represents the first sub group expression in our match pattern i.e. 040
-) includes a closing parenthesis
- **\2** represents the second sub group i.e. 9966
- includes a hyphen
- **\3** represents the third sub group i.e. 1234

We'll see some more formatting in our next example.

FROM dual;

Output: +91-40-9966-1234

In the next example, we shall include a space between every character.

SELECT REGEXP_REPLACE('YAHOO', '(.)', '\1 ') as Output FROM dual;

Output: Y A H O O

12.Some more patterns

In the below example let's look for http:// followed by a substring of one or more alphanumeric characters and optionally, a period (.)

SELECT REGEXP_SUBSTR('Go to http://www.oracle.com/products and click on database','http://([[:alnum:]]+\.?){3,4}/?') RESULT

FROM dual;

Output: http://www.oracle.com

Input: Go to http://www.oracle.co.uk/products and click on database Output: http://www.oracle.co.uk

http:// The characters http://
([[:alnum:]]+\.?) represents the sub expression: one or more occurrences of
alphanumeric characters followed by a dot optionally
{3,4} minimum 3 occurrences of the above sub expression to a max of 4
/? followed by forward slash optionally

Let's now try to extract the third value from a csv string.

select regexp_substr('1101,Yokohama,Japan,1.5.105','[^,]+',1,3)as Output from
dual;

Output: Japan

[^,]+ one or more occurrences of non comma characters
1 specifies the starting position
3 Third match

Let us assume we have a source string as "Why does a kid enjoy kidding with kids only?" and we want to search for either kid or kids or kidding in the source string.

select case when
 regexp_like('Why does a kid enjoy kidding with kids only?','kid(s|ding)*', 'i')
 then 'Match Found'
 else 'No Match Found'
 end
as output from dual;

Output: Match Found

kid characters kid (s|ding)* followed by zero or more occurrences of s or ding (| stands for OR) i represents case insensitive

I have tried to accommodate few common and regular patterns which we may find useful for our work. For more examples and information on regular expressions you may visit the following links.

- 1. <u>http://www.oracle.com/technology/obe/obe10gdb/develop/regexp/regexp.htm</u>
- 2. <u>http://www.psoug.org/reference/regexp.html</u>

Oracle Documentation

Available regular Expression Functions:

Function Name	Description
REGEXP_LIKE	Similar to the LIKE operator, but performs regular expression

	matching instead of simple pattern matching
REGEXP_INSTR	Searches for a given string for a regular expression pattern and returns the position were the match is found
REGEXP_REPLACE	Searches for a regular expression pattern and replaces it with a replacement string
REGEXP_SUBSTR	Searches for a regular expression pattern within a given string and returns the matched substring

Syntax

REGEXP_LIKE(srcstr, pattern [,match_option])

REGEXP_INSTR(srcstr, pattern [, position [, occurrence [, return_option [, match_option]]])

REGEXP_SUBSTR(srcstr, pattern [, position [, occurrence [, match_option]]])

REGEXP_REPLACE(srcstr, pattern [,replacestr [, position [, occurrence [, match_option]]])

Metacharacters

You can use several predefined metacharacter symbols in the pattern matching with the functions.

Symbol	Description
*	Matches zero or more occurrences
1	Alternation operator for specifying alternative matches
^/\$	Matches the start of line and the end of line
[]	Bracket expression for a matching list matching any one of the expressions represented in the list
[^exp]	If the caret is inside the bracket, it negates the expression.
{m}	Matches exactly <i>m</i> times
{m,n}	Matches at least <i>m</i> times but no more than <i>n</i> times
[: :]	Specifies a character class and matches any character in that class
١	Can have four different meanings: (1) stand for itself; (2) quote the next character; (3) introduce an operator; (4) do nothing
+	Matches one or more occurrences
?	Matches zero or one occurrence
•	Matches any character in the supported character set (except NULL)
()	Grouping expression (treated as a single subexpression)

\n	Backreference expression
[==]	Specifies equivalence classes
[]	Specifies one collation element (such as a multicharacter element)

Match Options

Option	Description
С	Case sensitive matching
i	Case insensitive matching

Character Classes

Option	Description
[:alnum:]	Alphanumeric characters
[:alpha:]	Alphabetic characters
[:blank:]	Blank Space Characters
[:cntrl:]	Control characters (nonprinting)
[:digit:]	Numeric digits
[:lower:]	Lowercase alphabetic characters
[:space:]	Space characters (nonprinting), such as carriage return, newline, vertical tab, and form feed
[:upper:]	Uppercase alphabetic characters