SQL Top-20 Regular Expressions

Joachim Forster
Lisa Dräxlmaier GmbH
Vilsbiburg

Version 27.11.2017

What are Regular Expressions

With regular expressions you can find, replace, count, extract and check patterns in text

Regular Expression patterns are a sequences of characters

Some characters have special functionality and are called “metacharacters”, all others are “literals”

Regular expression are available in many programming environments since long time

Oracle supports REGEX features in SQL & PL/SQL starting with Version 10g (year 2003)

There exist different flavors of Regular Expressions, Oracle Syntax complies with POSIX standard

Oracle supports multilingual, multibyte character sets and sort orders

Instead of writing complex code write (sometimes complex) patterns

Traditional Character Functions in Oracle SQL and PL/SQL

FUNCTION NAME	RETURN TYPE	SOURCE STRING	LIKE PATTERN
LIKE	BOOLEAN	1	2

LIKE metacharacters:

'_' matches any character

'%' matches zero or more characters

FUNCTION NAME	RETURN TYPE	SOURCE STRING	SUB STRING	REPLACE STRING	START POSITION	OCCURRENCE	LENGTH
INSTR	NUMBER	1	2		3*	4*	
SUBSTR	STRING	1			2		3*
REPLACE	STRING	1	2	3*			
COUNT ?!?!	NUMBER	aggregate not a character function !!!					

* = optional

Regular Expression Functions and Parameters in Oracle SQL and PL/SQL

FUNCTION NAME	RETURN TYPE	SRC STR	REXP PATTERN	REPLACE STR	POSITION >= 1	OCCURRENCE >= 1(0)	RETURN PARAM (0,1)	MODIFIER icnmx	SUB EXPRESSION [0-9]
REGEXP_LIKE	BOOL	1	2					3*	
REGEXP_INSTR	INT	1	2		3*	4*	5*	6*	7*
REGEXP_SUBSTR	STR	1	2		3*	4* 0=err		5*	6*
REGEXP_REPLACE	STR	1	2	3*	4*	5* 0=all		6*	
REGEXP_COUNT	INT	1	2		3*			4*	

* = optional

Regular Expression metacharacters (1/2)

Meta char	Description
.	Matches any character excluding newline (including newline with match option n)
\	Escape character, treats the subsequent (meta)character as literal
^	Matches the begin of a string. (for begin of any line use match option m)
$	Matches the end of a string. (for end of any line whith match option m)
[]	matching list, try to match any one of the characters in the list, e.g.: [aeiou], [02468]
[^]	nonmatching list, try to match any character except the ones in the list. e.g.: [^,.!?]
[::]	Matches character classes. [:cntrl:],[:space:],[:blank:],[:print:],[:graph:],[:punct:], [:alnum:],[:alpha:],[:digit:],[:lower:],[:upper:],[:xdigit:]
\d \D	Matches a digit / nondigit character. equivalent [[:digit:]] / [^[:digit:]]
\w \W	Matches a word / nonword character. equivalent [[:alnum:]_] / [^[:alnum:]_]
\s \S	Matches a whitespace / non-whitespace character. equivalent [[:space:]] / [^[:space:]]
[==]	Matches equivalence classes.
[..]	Matches one collation element that can be more than one character.

Regular Expression metacharacters (2/2)

Meta char	Description
|	Used like an "OR" to specify more than one alternative.
( )	Used to group expressions as a subexpression.
\n	n is a number between 1 and 9. Matches the nth subexpression found within ( )

greedy	lazy	
*****	***?**	Matches 0 or more occurrences of the preceeding subexpression.
+	**+?**	Matches 1 or more occurrences of the preceeding subexpression.
?	**??**	Matches 0 or 1 occurrence of the preceeding subexpression.
{m}	**{m}?**	Matches exactly m occurences of the preceeding subexpression.
{m,}	**{m,}?**	Matches at least m occurences of the preceeding subexpression.
{m,n}	**{m,n}?**	Matches at least m but not more than n occ. of the prec.subexpression.

List of SQL Top-20 Regular Expressions (1/2)

TOP	Title
1.1:	Ascii Matrix (NSL_CHARACTERSET = WE8ISO8859P15)
2.1:	REGEXP_REPLACE (REPLACESTR = #)
2.2:	REGEXP_COUNT
2.3:	REGEXP_SUBSTR
2.4:	REGEXP_INSTR
2.5:	REGEXP_LIKE(_1) (boolean datatype not supported in Oracle)
2.6:	REGEXP_EXTRACT (PL/SQL ...) -> Top 19
3:	Count characters, words, sentences, lines, ... in test
4:	Extract Token by Position (including NULL Values)
5:	Extract Prefix and Suffix (nonmatching, greedy, lazy)
6:	Extract Pah, File and Extension Name
7:	Value for Key in KVP String
8:	Key Value Pairs to CSV Header and Data
9:	CSV Row to Table
10:	CSV Rows to Table

List of SQL Top-20 Regular Expressions (2/2)

TOP	Title
11.1:	OR_LIKE
11.2:	OR_LIKE_TO_REXP Pattern Translation function
12.1:	Is string upper or lower case
12.2:	Is upper ? ... Lots of errors can happen ... so test carefully
13:	Is numeric
14:	Check Valid Oracle Identifier (quoted and nonquoted)
15:	Remove consecutive duplicates
16:	Wrap Text
17:	Convert camelCase to UPPER_CASE code style and vice versa
18:	Remove Comments
19:	REGEXP_EXTRACT PL/SQL for extracting multiple substrings ...
20:	Compare Performance “Count” Traditional vs. Regexp

Top 1.1: Ascii Matrix (NSL_CHARACTERSET = WE8ISO8859P15) SQL

```
WITH character_set AS -- generate single byte character set
 (
 SELECT
 16 * floor((LEVEL - 1) / 16) y                                    -- upper 4 bit
,MOD((LEVEL - 1), 16)          x                                  -- lower 4 bit
,CASE WHEN regexp_like(chr(LEVEL - 1), '&pattern')
      THEN
           CASE  WHEN regexp_like(chr(LEVEL - 1), '[[:print:]]')
                 THEN chr(LEVEL - 1)            -- chr for printable characters
                 ELSE '--'                      -- '--' for nonprintable characters
           END
 END c
 FROM dual CONNECT BY LEVEL <= 256
)
SELECT * FROM character_set  -- pivot characters
pivot (MAX(c)
   FOR x IN(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15))
 ORDER BY y;
```

Top 1.1: Ascii Matrix for character class [[:space:]] RESULT

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32																
48																
64																
80																
96																
112																

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
128																
144																
160																
176																
192																
208																
224																
240																

Top 1.1: Ascii Matrix for character class [[:punct:]] RESULT

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48											:	;	<	=	>	?
64	@															
80												[	\	]	^	_
96	`															
112												{	|	}	~	

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
128																
144																
160		¡	¢	£	¿	¥		§		©	ª	«	¬	-	®	¯
176	°	±	²	³		µ	¶	·		¹	º	»				¿
192																
208								×								
224																
240								÷								

Top 1.1: Ascii Matrix for character class [[:digit:]] RESULT

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32																
48	0	1	2	3	4	5	6	7	8	9						
64																
80																
96																
112																

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
128																
144																
160																
176																
192																
208																
224																
240																

Top 1.1: Ascii Matrix for character class [[:upper:]] RESULT

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32																
48																
64		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z					
96																
112																

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
128																
144																
160							¿									
176					¿								¿		Y	
192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö		Ø	Ù	Ú	Û	Ü	Ý	Þ	
224																
240																

Top 1.1: Ascii Matrix for character class [[:lower:]] RESULT

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32																
48																
64																
80																
96		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z					

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
128																
144																
160									¿							
176									¿					¿		
192																
208																ß
224	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö		ø	ù	ú	û	ü	ý	þ	ÿ

Top 1.1: Ascii Matrix for character class [[:alpha:]] RESULT

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32																
48																
64		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z					
96		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z					

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
128																
144																
160							¿		¿							
176					¿				¿				¿	¿	Y	
192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö		Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
224	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö		ø	ù	ú	û	ü	ý	þ	ÿ

Top 1.1: Ascii Matrix for character class [[:alnum:]] RESULT

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32																
48	0	1	2	3	4	5	6	7	8	9						
64		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z					
96		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z					

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
128																
144																
160							¿		¿							
176					¿				¿				¿	¿	Y	
192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö		Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
224	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö		ø	ù	ú	û	ü	ý	þ	ÿ

Top 1.1: Ascii Matrix for character class \w RESULT

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32																
48	0	1	2	3	4	5	6	7	8	9						
64		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z					_
96		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z					

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
128																
144																
160							¿		¿							
176					¿				¿				¿	¿	Y	
192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö		Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
224	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö		ø	ù	ú	û	ü	ý	þ	ÿ

Top 1.1: Ascii Matrix for character class [[:print:]] RESULT

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
128																
144																
160		¡	¢	£	¿	¥	¿	§	¿	©	ª	«	¬	-	®	¯
176	°	±	²	³	¿	µ	¶	·	¿	¹	º	»	¿	¿	Y	¿
192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
224	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Top 1.1: Ascii Matrix for pattern [[:upper:][:digit:]_$#] RESULT

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32				#	$											
48	0	1	2	3	4	5	6	7	8	9						
64		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z					_
96																
112																

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
128																
144																
160							¿									
176					¿								¿		Y	
192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö		Ø	Ù	Ú	Û	Ü	Ý	Þ	
224																
240																

Top 1.1: Ascii Matrix for pattern [A-Z] (nls_sort = binary) RESULT

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32																
48																
64		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z					
96																
112																

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
128																
144																
160																
176																
192																
208																
224																
240																

```
alter session set nls_sort = binary;  -- default ok ...
```

Top 1.1: Ascii Matrix for pattern [A-Z] (nls_sort = german) RESULT

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32																
48																
64		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z					
96			b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z					

Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
128																
144																
160																
176															Y	
192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö		Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
224	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö		ø	ù	ú	û	ü	ý	þ	ÿ

```
alter session set nls_sort = german;  -- !!!!!
```

Top 2.1: REGEXP_REPLACE (replacestr = #, all occurrences = 0)

```
SELECT
--rexp_function(srcstr           , pattern      ,rep,po,occur,modifier) col_name     -- RESULT
--=============(=================,==============,===,==,=====,========)============ -- ==============
 regexp_replace('Eins,Zwei,Drei' , ','          ,'#'                   ) comma        -- Eins#Zwei#Drei
,regexp_replace('Eins,Zwei,Drei' , 'ei'         ,'#'                   ) ei           -- Eins,Zw#,Dr#
,regexp_replace('Eins,Zwei,Drei' , 'ei'         ,'#',10                ) ei_pos_10    -- Eins,Zwei,Dr#
,regexp_replace('Eins,Zwei,Drei' , 'ei'         ,'#',1 ,0     ,'i'     ) ei_ic        -- #ns,Zw#,Dr#
,regexp_replace('Eins,Zwei,Drei' , '^ei'        ,'#'                   ) ei_begin     -- Eins,Zwei,Drei
,regexp_replace('Eins,Zwei,Drei' , 'ei$'        ,'#'                   ) ei_end       -- Eins,Zwei,Dr#
,regexp_replace('Eins,Zwei,Drei' , 'ei|i'       ,'#'                   ) ei_or_i      -- E#ns,Zw#,Dr#
,regexp_replace('Eins,Zwei,Drei' , '[ei]'       ,'#'                   ) e_or_i       -- E#ns,Zw##,Dr##
,regexp_replace('Eins,Zwei,Drei' , '[^ei]'      ,'#'                   ) not_e_or_i   -- #i#####ei###ei
,regexp_replace('Eins.Zwei.Drei' , '.'          ,'#'                   ) any_char     -- ##############
,regexp_replace('Eins.Zwei.Drei' , '\.'         ,'#'                   ) esc_dot      -- Eins#Zwei#Drei
,regexp_replace('Eins.Zwei.Drei' , '[.]'        ,'#'                   ) class_dot    -- Eins#Zwei#Drei
,regexp_replace('Eins.Zwei.Drei' , '[[:lower:]]','#'                   ) class_lower  -- E###.Z###.D###
,regexp_replace('Eins.Zwei.Drei' , '\w'         ,'#'                   ) word_char    -- ####.####.####
,regexp_replace('Eins.Zwei.Drei' , '\w+'        ,'#'                   ) word_sting   -- #.#.#
,regexp_replace('Eins,Zwei,Drei' , 'i(,|$)'     ,'#'                   ) i_end_word   -- Eins,Zwe#Dre#
,regexp_replace('Eins,Zwei,Drei' , '(..).*\1'   ,'#'                   ) repeat_dup   -- Eins,Zw#
,regexp_replace('Drei,Fünf,Neun' , '[[=u=]]'    ,'#'                   ) equv_class   -- Drei,F#nf,Ne#n
,regexp_replace(''               , ''                                  ) null_values --
FROM dual;
```

Top 2.2: REGEXP_COUNT

```
SELECT
--rexp_function(srcstr           , pattern          ,posit    modifier) col_name    -- RESULT
--=============(=================,==================,=====,==========)============ -- ==============
 regexp_count  ('Eins,Zwei,Drei' , ','                                ) comma        --       2
,regexp_count  ('Eins,Zwei,Drei' , 'ei'                               ) ei           --       2
,regexp_count  ('Eins,Zwei,Drei' , 'ei'             ,10               ) ei_pos_10    --       1
,regexp_count  ('Eins,Zwei,Drei' , 'ei'             ,1        ,'i'    ) ei_ic        --       3
,regexp_count  ('Eins,Zwei,Drei' , '^ei'                              ) ei_begin     --       0
,regexp_count  ('Eins,Zwei,Drei' , 'ei$'                              ) ei_end       --       1
,regexp_count  ('Eins,Zwei,Drei' , 'ei|i'                             ) ei_or_i      --       3
,regexp_count  ('Eins,Zwei,Drei' , '[ei]'                             ) e_or_i       --       5
,regexp_count  ('Eins,Zwei,Drei' , '[^ei]'                            ) not_e_or_i   --       9
,regexp_count  ('Eins.Zwei.Drei' , '.'                                ) any_char     --      14
,regexp_count  ('Eins.Zwei.Drei' , '\.'                               ) esc_dot      --       2
,regexp_count  ('Eins.Zwei.Drei' , '[.]'                              ) class_dot    --       2
,regexp_count  ('Eins.Zwei.Drei' , '[[:lower:]]'                      ) class_lower  --       9
,regexp_count  ('Eins.Zwei.Drei' , '\w'                               ) word_char    --      12
,regexp_count  ('Eins.Zwei.Drei' , '\w+'                              ) word_sting   --       3
,regexp_count  ('Eins,Zwei,Drei' , 'i(,|$)'                           ) i_end_word   --       2
,regexp_count  ('Eins,Zwei,Drei' , '(..).*\1'                         ) repeat_dup   --       1
,regexp_count  ('Drei,Fünf,Neun' , '[[=u=]]'                          ) equv_class   --       2
,regexp_count  (''               , ''                                 ) null_values  --
FROM dual;
```

Top 2.3: REGEXP_SUBSTR

```
SELECT
--rexp_function(srcstr           , pattern          ,po,occur,mod,sube) col_name    -- RESULT
--=============(================,==================,==,=====,========)============ -- ==============
 regexp_substr ('Eins,Zwei,Drei' , ','                                ) comma       -- ,
,regexp_substr ('Eins,Zwei,Drei' , 'ei'                               ) ei          -- ei
,regexp_substr ('Eins,Zwei,Drei' , 'ei'             ,10               ) ei_pos_10   -- ei
,regexp_substr ('Eins,Zwei,Drei' , 'ei'             ,1 ,1     ,'i'    ) ei_ic       -- Ei
,regexp_substr ('Eins,Zwei,Drei' , '^ei'                              ) ei_begin    --
,regexp_substr ('Eins,Zwei,Drei' , 'ei$'                              ) ei_end      -- ei
,regexp_substr ('Eins,Zwei,Drei' , 'ei|i'                             ) ei_or_i     -- i
,regexp_substr ('Eins,Zwei,Drei' , '[ei]'                             ) e_or_i      -- i
,regexp_substr ('Eins,Zwei,Drei' , '[^ei]'                            ) not_e_or_i  -- E
,regexp_substr ('Eins.Zwei.Drei' , '.'                                ) any_char    -- E
,regexp_substr ('Eins.Zwei.Drei' , '\.'                               ) esc_dot     -- .
,regexp_substr ('Eins.Zwei.Drei' , '[.]'                              ) class_dot   -- .
,regexp_substr ('Eins.Zwei.Drei' , '[[:lower:]]'                      ) class_lower -- i
,regexp_substr ('Eins.Zwei.Drei' , '\w'                               ) word_char   -- E
,regexp_substr ('Eins.Zwei.Drei' , '\w+'                              ) word_sting  -- Eins
,regexp_substr ('Eins,Zwei,Drei' , 'i(,|$)'                           ) i_end_word  -- i,
,regexp_substr ('Eins,Zwei,Drei' , '(..).*\1'                         ) repeat_dup  -- ei,Drei
,regexp_substr ('Drei,Fünf,Neun' , '[[=u=]]'                          ) equv_class  -- ü
,regexp_substr (''               , ''                                 ) null_values --
FROM dual;
```

Top 2.4: REGEXP_INSTR

```
SELECT
--rexp_function(srcstr           , pattern          ,po,oc,re,mod,sube) col_name     -- RESULT
--=============(=================,==================,==,==,==,===,====)============ -- ==============
 regexp_instr  ('Eins,Zwei,Drei' , ','                                ) comma        --       5
,regexp_instr  ('Eins,Zwei,Drei' , 'ei'                               ) ei           --       8
,regexp_instr  ('Eins,Zwei,Drei' , 'ei'             ,10               ) ei_pos_10    --      13
,regexp_instr  ('Eins,Zwei,Drei' , 'ei'             ,1 ,1 ,0 ,'i'     ) ei_ic        --       1
,regexp_instr  ('Eins,Zwei,Drei' , '^ei'                              ) ei_begin     --       0
,regexp_instr  ('Eins,Zwei,Drei' , 'ei$'                              ) ei_end       --      13
,regexp_instr  ('Eins,Zwei,Drei' , 'ei|i'                             ) ei_or_i      --       2
,regexp_instr  ('Eins,Zwei,Drei' , '[ei]'                             ) e_or_i       --       2
,regexp_instr  ('Eins,Zwei,Drei' , '[^ei]'                            ) not_e_or_i   --       1
,regexp_instr  ('Eins.Zwei.Drei' , '.'                                ) any_char     --       1
,regexp_instr  ('Eins.Zwei.Drei' , '\.'                               ) esc_dot      --       5
,regexp_instr  ('Eins.Zwei.Drei' , '[.]'                              ) class_dot    --       5
,regexp_instr  ('Eins.Zwei.Drei' , '[[:lower:]]'                      ) class_lower  --       2
,regexp_instr  ('Eins.Zwei.Drei' , '\w'                               ) word_char    --       1
,regexp_instr  ('Eins.Zwei.Drei' , '\w+'                              ) word_sting   --       1
,regexp_instr  ('Eins,Zwei,Drei' , 'i(,|$)'                           ) i_end_word   --       9
,regexp_instr  ('Eins,Zwei,Drei' , '(..).*\1'                         ) repeat_dup   --       8
,regexp_instr  ('Drei,Fünf,Neun' , '[[=u=]]'                          ) equv_class   --       7
,regexp_instr  (''               , ''                                 ) null_values  --
FROM dual;
```

Top 2.5: REGEXP_LIKE1 (... as boolean not supported in SQL)

```
SELECT
--rexp_function(srcstr           , pattern                    ,modifier) col_name     -- RESULT
--=============(=================,============================,========)============ -- ==============
 regexp_like1   ('Eins,Zwei,Drei' , ','                                ) comma        --   TRUE
,regexp_like1   ('Eins,Zwei,Drei' , 'ei'                               ) ei           --   TRUE
,regexp_like1   ('Eins,Zwei,Drei' , 'ei'                               ) ei_pos_10    --   TRUE
,regexp_like1   ('Eins,Zwei,Drei' , 'ei'                  ,'i'         ) ei_ic        --   TRUE
,regexp_like1   ('Eins,Zwei,Drei' , '^ei'                              ) ei_begin     --  FALSE
,regexp_like1   ('Eins,Zwei,Drei' , 'ei$'                              ) ei_end       --   TRUE
,regexp_like1   ('Eins,Zwei,Drei' , 'ei|i'                             ) ei_or_i      --   TRUE
,regexp_like1   ('Eins,Zwei,Drei' , '[ei]'                             ) e_or_i       --   TRUE
,regexp_like1   ('Eins,Zwei,Drei' , '[^ei]'                            ) not_e_or_i   --   TRUE
,regexp_like1   ('Eins.Zwei.Drei' , '.'                                ) any_char     --   TRUE
,regexp_like1   ('Eins.Zwei.Drei' , '\.'                               ) esc_dot      --   TRUE
,regexp_like1   ('Eins.Zwei.Drei' , '[.]'                              ) class_dot    --   TRUE
,regexp_like1   ('Eins.Zwei.Drei' , '[[:lower:]]'                      ) class_lower  --   TRUE
,regexp_like1   ('Eins.Zwei.Drei' , '\w'                               ) word_char    --   TRUE
,regexp_like1   ('Eins.Zwei.Drei' , '\w+'                              ) word_sting   --   TRUE
,regexp_like1   ('Eins,Zwei,Drei' , 'i(,|$)'                           ) i_end_word   --   TRUE
,regexp_like1   ('Eins,Zwei,Drei' , '(..).*\1'                         ) repeat_dup   --   TRUE
,regexp_like1   ('Drei,Fünf,Neun' , '[[=u=]]'                          ) equv_class   --   TRUE
,regexp_like1   (''               , ''                                 ) null_values  --
FROM dual;
```

Top 2.5: REGEXP_LIKE1 PL/SQL Wrapper Function

```
CREATE OR REPLACE FUNCTION regexp_like1(srcstr   VARCHAR2
                                       ,pattern  VARCHAR2
                                       ,modifier VARCHAR2 DEFAULT NULL)
  -- wrapper function for regexp_like
  -- returns string 'TRUE', 'FALSE' and NULL
  -- only use for demo
  RETURN VARCHAR2 IS
  result_boolean BOOLEAN;
BEGIN
  -- PL_SQL supports boolean values
  result_boolean := regexp_like(srcstr   => srcstr
                               ,pattern  => pattern
                               ,modifier => modifier);
  RETURN(
  CASE WHEN result_boolean IS NULL THEN NULL
       WHEN result_boolean THEN 'TRUE'
                           ELSE 'FALSE'
         END);
END;
/
```

Top 2.6: REGEXP_EXTRACT (PL/SQL, REPL.STR = null, OCCURR. = 0) → Top 19

```
SELECT
--rexp_function(srcstr           , pattern      ,rep,po,occur,modifier) col_name    -- RESULT
--=============(=================,==============,===,==,=====,========)============ -- ==============
 regexp_extract('Eins,Zwei,Drei' , ','                                ) comma        -- ,,
,regexp_extract('Eins,Zwei,Drei' , 'ei'                               ) ei           -- eiei
,regexp_extract('Eins,Zwei,Drei' , 'ei'         ,'' ,10               ) ei_pos_10    -- ei
,regexp_extract('Eins,Zwei,Drei' , 'ei'         ,'' ,1 ,0     ,'i'    ) ei_ic        -- Eieiei
,regexp_extract('Eins,Zwei,Drei' , '^ei'                              ) ei_begin     --
,regexp_extract('Eins,Zwei,Drei' , 'ei$'                              ) ei_end       -- ei
,regexp_extract('Eins,Zwei,Drei' , 'ei|i'                             ) ei_or_i      -- ieiei
,regexp_extract('Eins,Zwei,Drei' , '[ei]'                             ) e_or_i       -- ieiei
,regexp_extract('Eins,Zwei,Drei' , '[^ei]'                            ) not_e_or_i   -- Ens,Zw,Dr
,regexp_extract('Eins.Zwei.Drei' , '.'                                ) any_char     -- Eins.Zwei.Drei
,regexp_extract('Eins.Zwei.Drei' , '\.'                               ) esc_dot      -- ..
,regexp_extract('Eins.Zwei.Drei' , '[.]'                              ) class_dot    -- ..
,regexp_extract('Eins.Zwei.Drei' , '[[:lower:]]'                      ) class_lower  -- insweirei
,regexp_extract('Eins.Zwei.Drei' , '\w'                               ) word_char    -- EinsZweiDrei
,regexp_extract('Eins.Zwei.Drei' , '\w+'                              ) word_sting   -- EinsZweiDrei
,regexp_extract('Eins,Zwei,Drei' , 'i(,|$)'                           ) i_end_word   -- i,i
,regexp_extract('Eins,Zwei,Drei' , '(..).*\1'                         ) repeat_dup   -- ei,Drei
,regexp_extract('Drei,Fünf,Neun' , '[[=u=]]'                          ) equv_class   -- üu
,regexp_extract(''               , ''                                 ) null_values --
FROM dual;
```

Top 2.6: REGEXP_EXTRACT (PL/SQL, REPL.STR = '#', OCCURR. = 0) → Top 19

```
SELECT
--rexp_function(srcstr           , pattern      ,rep,po,occur,modifier) col_name     -- RESULT
--=============(=================,==============,===,==,=====,========)============ -- ==============
 regexp_extract('Eins,Zwei,Drei' , ','          ,'#'                   ) comma        -- ##
,regexp_extract('Eins,Zwei,Drei' , 'ei'         ,'#'                   ) ei           -- ##
,regexp_extract('Eins,Zwei,Drei' , 'ei'         ,'#',10                ) ei_pos_10    -- #
,regexp_extract('Eins,Zwei,Drei' , 'ei'         ,'#',1 ,0     ,'i'     ) ei_ic        -- ###
,regexp_extract('Eins,Zwei,Drei' , '^ei'        ,'#'                   ) ei_begin     --
,regexp_extract('Eins,Zwei,Drei' , 'ei$'        ,'#'                   ) ei_end       -- #
,regexp_extract('Eins,Zwei,Drei' , 'ei|i'       ,'#'                   ) ei_or_i      -- ###
,regexp_extract('Eins,Zwei,Drei' , '[ei]'       ,'#'                   ) e_or_i       -- #####
,regexp_extract('Eins,Zwei,Drei' , '[^ei]'      ,'#'                   ) not_e_or_i   -- #########
,regexp_extract('Eins.Zwei.Drei' , '.'          ,'#'                   ) any_char     -- ##############
,regexp_extract('Eins.Zwei.Drei' , '\.'         ,'#'                   ) esc_dot      -- ##
,regexp_extract('Eins.Zwei.Drei' , '[.]'        ,'#'                   ) class_dot    -- ##
,regexp_extract('Eins.Zwei.Drei' , '[[:lower:]]','#'                   ) class_lower  -- #########
,regexp_extract('Eins.Zwei.Drei' , '\w'         ,'#'                   ) word_char    -- ############
,regexp_extract('Eins.Zwei.Drei' , '\w+'        ,'#'                   ) word_sting   -- ##
,regexp_extract('Eins,Zwei,Drei' , 'i(,|$)'     ,'#'                   ) i_end_word   -- ##
,regexp_extract('Eins,Zwei,Drei' , '(..).*\1'   ,'#'                   ) repeat_dup   -- #
,regexp_extract('Drei,Fünf,Neun' , '[[=u=]]'    ,'#'                   ) equv_class   -- ##
,regexp_extract(''               , ''                                  ) null_values  --
FROM dual;
```

Top 3: Count characters, words, sentences, lines, ... in test

```
WITH t AS (SELECT
'Einmal saß Kasperls Großmutter auf der Bank vor ihrem Häuschen in der Sonne und
mahlte Kaffee. Kasperl und sein Freund Seppel hatten ihr zum Geburtstag eine neue
Kaffeemühle geschenkt, die hatten sie selbst erfunden. Wenn man daran kurbelte,
spielte sie "Alles neu macht der Mai", das war Großmutters Lieblingslied.'
s FROM dual)
SELECT                                                          -- RESULT
 length      (s                        ) length_text            --    315
,vsize       (s                        ) vsize_text             --    315
,regexp_count(s, '.'                   ) characters             --    312
,regexp_count(s, '.', 1, 'n'           ) characters_nl          --    315
,regexp_count(s, '\w+'                 ) words                  --     49
,regexp_count(s, '\.'                  ) sentences              --      3
,regexp_count(s, '[ÄÖÜäöüß]'           ) german_characters      --      5
,regexp_count(s, chr(10)               ) lines                  --      3
,regexp_count(s, 'Kaffee'              ) substr1                --      2
,regexp_count(s, '\bKaffee\b'          ) word_boundary_not_supported --  0
,regexp_count(s, '(^|\W)Kaffee(\W|$)') word_boundary_workaround --      1
FROM t;
```

Top 4: Extract Token by Position (including NULL Values)

```
WITH t       (id , csv               ) AS
 (      SELECT 1 , 'Eins,Zwei,Drei' FROM dual
  UNION SELECT 2 , ',Zwei,Drei'     FROM dual
  UNION SELECT 3 , 'Eins,,Drei'     FROM dual
  UNION SELECT 4 , ',Zwei,'         FROM dual
 )
SELECT csv
      ,regexp_substr(csv, '([^,]*)(,|$)', 1, 1, null, 1) token_1
      ,regexp_substr(csv, '([^,]*)(,|$)', 1, 2, null, 1) token_2
      ,regexp_substr(csv, '([^,]*)(,|$)', 1, 3, null, 1) token_3
  FROM t ORDER BY id;
```

```
CSV             | TOKEN_1    | TOKEN_2    | TOKEN_3
--------------- | ---------- | ---------- | ----------
Eins,Zwei,Drei  | Eins       | Zwei       | Drei
,Zwei,Drei      |            | Zwei       | Drei
Eins,,Drei      | Eins       |            | Drei
,Zwei,          |            | Zwei       |
```

Top 5.1: Extract Prefix (greedy ?!?, lazy ?, lazy+subexp)

```
WITH t   (id,  str                    ) AS
 ( SELECT 1 , 'Prefix.Suffix'         FROM dual UNION
   SELECT 2 , 'Prefix.Middle.Suffix' FROM dual UNION
   SELECT 3 , '.Middle.'             FROM dual UNION
   SELECT 4 , 'Middle'               FROM dual )
SELECT str
,regexp_substr(str, '.*\.'                    ) pref_greedy      -- greedy !?!
,regexp_substr(str, '.*?\.'                   ) pref_lazy        -- lazy   ?
,regexp_substr(str, '(.*?)\.'    , 1, 1, null, 1) pref_lazy_subexp -- lazy, subexp.
FROM t ORDER BY id;

STR                | PREF_GREEDY    | PREF_LAZY | PREF_LAZY_SUBEXP
------------------ | -------------- | --------- | ----------------
Prefix.Suffix      | Prefix.        | Prefix.   | Prefix
Prefix.Middle.Suffix | Prefix.Middle. | Prefix. | Prefix
.Middle.           | .Middle.       | .         |
Middle             |                |           |
```

Top 5.2: Extract Prefix (nonmatching)

```
WITH t   (id,  str                      ) AS
 ( SELECT 1 , 'Prefix.Suffix'           FROM dual UNION
   SELECT 2 , 'Prefix.Middle.Suffix'    FROM dual UNION
   SELECT 3 , '.Middle.'                FROM dual UNION
   SELECT 4 , 'Middle'                  FROM dual )
SELECT str
,regexp_substr(str, '[^.]*'                      ) pref  -- nonmatching
FROM t ORDER BY id;
```

```
STR                  | PREF
-------------------- | -------
Prefix.Suffix        | Prefix
Prefix.Middle.Suffix | Prefix
.Middle.             |
Middle               | Middle
```

Top 5.3: Extract Suffix (nonmatching)

```
WITH t   (id,  str                    ) AS
 ( SELECT 1 , 'Prefix.Suffix'         FROM dual UNION
   SELECT 2 , 'Prefix.Middle.Suffix' FROM dual UNION
   SELECT 3 , '.Middle.'             FROM dual UNION
   SELECT 4 , 'Middle'               FROM dual )
SELECT str
,regexp_substr(str, '\.([^.]*$)', 1, 1, null, 1) suff  -- nonmatching, subexp.
,regexp_substr(str, '[^.]*$'                   ) suff2 -- nonmatching
FROM t ORDER BY id;
```

```
STR                 | SUFF   | SUFF2
------------------- | ------ | -------
Prefix.Suffix       | Suffix | Suffix
Prefix.Middle.Suffix | Suffix | Suffix
.Middle.            |        |
Middle              |        | Middle
```

Top 6: Extract Path, File and Extension Name

```
WITH t        ( id , path_filename            ) AS
 (       SELECT  1 , 'file.sql'               FROM dual
  UNION  SELECT  2 , 'file'                   FROM dual
  UNION  SELECT  3 , 'dir\file.sql'           FROM dual
  UNION  SELECT  4 , 'dir\file.bak.sql'       FROM dual
  UNION  SELECT  5 , 'dir\sdir\file.bak.sql' FROM dual
  UNION  SELECT  6 , 'dir\'                   FROM dual
)
SELECT path_filename
,regexp_substr (path_filename , '([^\]*)[\]' , 1 , 1 , '' , 1         ) root
,regexp_substr (path_filename , '(.*)\\'     , 1 , 1 , '' , 1         ) path
,regexp_substr (path_filename , '[^\]*$'                              ) filename_ext
,regexp_replace(regexp_substr (path_filename , '[^\]*$'), '\.[^.]*$'  ) filename_ext2
,regexp_substr (regexp_substr (path_filename , '[^\]*$')    , '[^.]*' ) filename
,regexp_substr (path_filename , '[.]([^.]*)$', 1 , 1 , '' , 1         ) ext
  FROM t ORDER BY id;
```

PATH_FILENAME	ROOT	PATH	FILENAME_EXT	FILENAME_EXT2	FILENAME	EXT
file.sql			file.sql	file	file	sql
file			file	file	file	
dir\file.sql	dir	dir	file.sql	file	file	sql
dir\file.bak.sql	dir	dir	file.bak.sql	file.bak	file	sql
dir\sdir\file.bak.sql	dir	dir\sdir	file.bak.sql	file.bak	file	sql
dir\	dir	dir				

Top 7: Value for Key in KVP String — SQL/RESULT

```
WITH weather (
id, kvps                                          ) AS            (SELECT
 1, 'temp_max=22 C; temp_min=12 C'                FROM dual UNION SELECT -- measure max min temperature
 2, 'sun=8.9 h; temp_max=26 C; temp_min=14 C'     FROM dual UNION SELECT -- additionally sunshine hours
 3, 'sun=3.7 h; temp_min=16 C; temp_max=19 C'     FROM dual UNION SELECT -- temp_min before temp_max
 4, 'sun=6.7 h; temp_min=18 C; temp_max=23 C'     FROM dual UNION SELECT -- =
 5, 'sun=6.3 h; temp_min=; temp_max='             FROM dual              -- temp. measurement failed
)
SELECT id, kvps -- src,pattern      (key)                    ,pos ,occ,mod ,subex
,regexp_substr(kvps   ,'(;|^| )' || 'sun'      || '=([^;]*)', 1   ,1   ,'i' , 2    ) sun
,regexp_substr(kvps   ,'(;|^| )' || 'temp_min' || '=([^;]*)', 1   ,1   ,'i' , 2    ) temp_min
,regexp_substr(kvps   ,'(;|^| )' || 'temp_max' || '=([^;]*)', 1   ,1   ,'i' , 2    ) temp_max
FROM weather ORDER BY id;
```

ID	KVPS	SUN	TEMP_MIN	TEMP_MAX
1	temp_max=22 C; temp_min=12 C		12 C	22 C
2	sun=8.9 h; temp_max=26 C; temp_min=14 C	8.9 h	14 C	26 C
3	sun=3.7 h; temp_min=16 C; temp_max=19 C	3.7 h	16 C	19 C
4	sun=6.7 h; temp_min=18 C; temp_max=23 C	6.7 h	18 C	23 C
5	sun=6.3 h; temp_min=; temp_max=	6.3 h		

Top 8: Key Value Pairs to CSV Header and Data SQL/RESULT

```
WITH weather (
id, kvps                                        ) AS            (SELECT
 1, 'temp_max=22 C; temp_min=12 C'              FROM dual UNION SELECT -- measure max min temperature
 2, 'sun=8.9 h; temp_max=26 C; temp_min=14 C'   FROM dual UNION SELECT -- additionally sunshine hours
 3, 'sun=3.7 h; temp_min=16 C; temp_max=19 C'   FROM dual UNION SELECT -- temp_min before temp_max
 4, 'sun=6.7 h; temp_min=18 C; temp_max=23 C'   FROM dual UNION SELECT -- =
 5, 'sun=6.3 h; temp_min=; temp_max='           FROM dual              -- temp. measurement failed
)
SELECT id, kvps -- src,pattern       , replstr
,regexp_replace(kvps, '=.*?(;|$)'     , '\1') keys_from_key_val_pairs -- search / remove values, keep
,regexp_replace(kvps, '(^|;).*?(=|$)', '\1') vals_from_key_val_pairs -- search / remove keys, keep
FROM weather ORDER BY id;
```

```
  ID | KVPS                                    | KEYS_FROM_KEY_VAL_PAIRS    | VALS_FROM_KEY_VAL_PAIRS
---- | --------------------------------------- | -------------------------- | -------------------------
   1 | temp_max=22 C; temp_min=12 C            | temp_max; temp_min         | 22 C;12 C
   2 | sun=8.9 h; temp_max=26 C; temp_min=14 C | sun; temp_max; temp_min    | 8.9 h;26 C;14 C
   3 | sun=3.7 h; temp_min=16 C; temp_max=19 C | sun; temp_min; temp_max    | 3.7 h;16 C;19 C
   4 | sun=6.7 h; temp_min=18 C; temp_max=23 C | sun; temp_min; temp_max    | 6.7 h;18 C;23 C
   5 | sun=6.3 h; temp_min=; temp_max=         | sun; temp_min; temp_max    | 6.3 h;;
```

Top 9: CSV Row to Table

```
WITH t (id  , csv_col          ) AS
 (SELECT  1 ,'Eins,Zwei,Drei' FROM dual)
SELECT id    y
      ,level x
      ,regexp_substr(csv_col, '([^,]*)(,|$)', 1, LEVEL, '', 1) token
  FROM t
CONNECT BY LEVEL <= regexp_count(csv_col, ',') + 1
ORDER BY y, x;
```

```
   Y |    X | TOKEN
---- | ---- | ----------
   1 |    1 | Eins
   1 |    2 | Zwei
   1 |    3 | Drei
```

Top 10: CSV Rows to Table

```
WITH t       (id,  csv_col          ) AS
 (      SELECT 1 , 'Eins,Zwei,Drei' FROM dual
  UNION SELECT 2 , 'Vier,,Sechs'    FROM dual
  UNION SELECT 3 , ',Acht'          FROM dual )
SELECT id       y, l.lev    x, l.token
  FROM t
      ,lateral (SELECT LEVEL lev
                     ,regexp_substr(t.csv_col,'([^,]*)(,|$)',1,LEVEL,NULL,1) token
                FROM dual
               CONNECT BY LEVEL <= regexp_count(t.csv_col, ',') + 1) l
ORDER BY y, x;


   Y |    X | TOKEN
---- | ---- | ----------
   1 |    1 | Eins
   1 |    2 | Zwei
   1 |    3 | Drei
   2 |    1 | Vier
   2 |    2 |
   2 |    3 | Sechs
   3 |    1 |
   3 |    2 | Acht
```

Top 11.1: OR_LIKE SQL

```
SELECT NAME
-- combination of multiple or-like conditions ------------------------------------------
,CASE WHEN NAME =     'Josef'                    -- Josef
        OR NAME LIKE 'Paul%'                     -- Paul, Paula
        OR NAME LIKE 'Fr__z'                     -- Franz, Fritz
        OR NAME =     'Anika'                    -- Anika
        OR NAME =     'Annika'                   -- Annika
        OR NAME LIKE '___'                       -- Ben
        OR NAME LIKE 'Hans\_Martin' ESCAPE '\' -- 'Hans_Martin (not Hans-Martin, Hans Martin)
        OR NAME LIKE 'John F.'                   -- John F.
      THEN 'X'  END or_like
-- reusable, one (bind) variable position ----------------------------------------------
,CASE WHEN regexp_like(NAME, '^(Josef|Paul.*|Fr..z|An{1,2}ika|...|Hans_Martin|John F\.)$')
      THEN 'X' END regexp_like_c
  FROM names
 ORDER BY NAME;
```

Top 11.1: OR_LIKE RESULT

NAME	OR_LIKE	REGEXP_LIKE_C
Anika	X	X
Anna		
Annika	X	X
Ben	X	X
Franz	X	X
Fritz	X	X
Hans		
Hans Martin		
Hans-Martin		
Hans_Martin	X	X
JOSEF		
John F.	X	X
Josef	X	X
Maria		
Paul	X	X
Paula	X	X
Xaver Franz		
josef		

Top 11.2: OR_LIKE_TO_REXP Pattern Translation function (... Idea)

```
WITH FUNCTION or_like_to_rexp (or_like VARCHAR2 ) RETURN VARCHAR2
    -- translate OR_LIKE Pattern to REGEXP Pattern
    IS
BEGIN
     return
     '^('                -- add begin anchor
     ||replace (replace (replace (replace (replace (replace ( or_like
     , '\_', '####')     -- save \_
     , '\%', '@@@@')     -- save \%
     , '_', '.')         -- like _ will become regexp_like .
     , '%', '.*')        -- like % will become regexp_like .*
     , '####', '_')      -- saved \_ will be _ regexp_like literal
     , '@@@@', '%')      -- saved \% will be % regexp_like literal
     ||')$'              -- add end anchor
     ;
END;
SELECT OR_LIKE_TO_REXP('Josef|Paul%|Fr__z|Anika|Annika|___|Hans\_Martin|John F\.') OR_LIKE_TO_REXP
  FROM dual
/

OR_LIKE_TO_REXP
--------------------------------------------------------------------------------
^(Josef|Paul.*|Fr..z|Anika|Annika|...|Hans_Martin|John F\.)$
```

Top 12.1: Is string upper or lower case — SQL

```
WITH t ( id, s)  AS
 (SELECT  1, 'UPPERCASE'    FROM dual UNION
  SELECT  2, 'UPPER CASE'   FROM dual UNION
  SELECT  3, 'UPPER_CASE2'  FROM dual UNION
  SELECT  4, 'ÜPPERCASE'    FROM dual UNION
  SELECT 11, 'lowercase'    FROM dual UNION
  SELECT 12, 'lower case'   FROM dual UNION
  SELECT 13, 'lower_case2'  FROM dual UNION
  SELECT 14, 'löwercase'    FROM dual UNION
  SELECT 21, 'Mixed Case'   FROM dual UNION
  SELECT 22, 'MixedCase'    FROM dual)
SELECT s
,case when upper(s) = s                          then '  X' end is_upper_sql
,case when lower(s) = s                          then '  X' end is_lower_sql
,case when regexp_like(s,'^[^[:lower:]]*$')  then '  X' end is_upper_rexp
,case when regexp_like(s,'^[^[:upper:]]*$')  then '  X' end is_lower_rexp
  FROM t
 ORDER BY id;
```

Top 12.1: Is string upper or lower case RESULT

S	IS_UPPER_SQL	IS_LOWER_SQL	IS_UPPER_REXP	IS_LOWER_REXP
UPPERCASE	X		X	
UPPER CASE	X		X	
UPPER_CASE2	X		X	
ÜPPERCASE	X		X	
lowercase		X		X
lower case		X		X
lower_case2		X		X
löwercase		X		X
Mixed Case				
MixedCase				

Top 12.2: Is upper ? Lots of errors can happen !!! SQL

```
WITH t ( id, s)  AS
 (SELECT  1, 'UPPERCASE'    FROM dual UNION
  SELECT  2, 'UPPER CASE'   FROM dual UNION
  SELECT  3, 'UPPER_CASE2'  FROM dual UNION
  SELECT  4, 'ÜPPERCASE'    FROM dual UNION
  SELECT 11, 'lowercase'    FROM dual UNION
  SELECT 12, 'lower case'   FROM dual UNION
  SELECT 13, 'lower_case2'  FROM dual UNION
  SELECT 14, 'löwercase'    FROM dual UNION
  SELECT 21, 'Mixed Case'   FROM dual UNION
  SELECT 22, 'MixedCase'    FROM dual)
SELECT s
,case when regexp_like(s,'UPPER'                    ) then '  X' end "UPPER"      -- UPPER substring
,case when regexp_like(s,'[UPPER]'                  ) then '  X' end "[UPPER]"    -- once any of the chars in [UPPER]
,case when regexp_like(s,'[:UPPER:]'                ) then '  X' end "[:UPPER:]" -- once any of the chars in [:UPPER:]
,case when regexp_like(s,'[:upper:]'                ) then '  X' end "[:upper:]" -- once any of the chars in [:upper:]
--,case when regexp_like(s,'[[:UPPER:]]') then '  X  ' end error5  ORA-12729: invalid character class in regular expression
,case when regexp_like(s,'[[:upper:]]'              ) then '  X' end "[[:upper:]]" -- once one upper char
,case when regexp_like(s,'[[:upper:]]*'             ) then '  X' end "[[:upper:]]*" -- once zero or more upper char
,case when regexp_like(s,'^[[:upper:]]*$'           ) then '  X' end "^[[:upper:]]*$" -- only upper chars from ^ to $
,case when regexp_like(s,'^[[:upper:] -_]*'         ) then '  X' end "^[[:upper:] -_]*$" -- and the range from ' ' to '_'
,case when regexp_like(s,'^[[:upper:] _-]*$'        ) then '  X' end "^[[:upper:] _-]*$" -- and chars ' ', '_', '-'
,case when regexp_like(s,'^[[:upper:][:digit:] _-]*$') then '  X' end "^[[:upper:][:digit:] _-]*$" -- and the digits
,case when regexp_like(s,'^[^[:lower:]]*$'          ) then '  X' end "^[^[:lower:]]*$" -- ... think nonmatching !!!
  FROM t
ORDER BY id;
```

Top 12.2: Is upper ? Lots of errors can happen !!! RESULT

S	UPPER	[UPPER]	[:UPPER:]	[:upper:]	[[:upper:]]	[[:upper:]]*	^[[:upper:]]*$
UPPERCASE	X	X	X		X	X	X
UPPER CASE	X	X	X		X	X	
UPPER_CASE2	X	X	X		X	X	
ÜPPERCASE		X	X		X	X	X
lowercase				X		X	
lower case				X		X	
lower_case2				X		X	
löwercase				X		X	
Mixed Case				X	X	X	
MixedCase				X	X	X	

S	^[[:upper:] -_]*$	^[[:upper:] _-]*$	^[[:upper:][:digit:] _-]*$	^[^[:lower:]]*$
UPPERCASE	X	X	X	X
UPPER CASE	X	X	X	X
UPPER_CASE2	X		X	X
ÜPPERCASE	X	X	X	X
lowercase	X			
lower case	X			
lower_case2	X			
löwercase	X			
Mixed Case	X			
MixedCase	X			

Top 13: Is numeric (NLS_NUMERIC_CHARACTERS = '.,') SQL

```
WITH t (id, num_as_str)  AS (  SELECT
 1 , '1234'         FROM dual UNION SELECT
 2 , '12.34'        FROM dual UNION SELECT
 3 , '1234.'        FROM dual UNION SELECT
 4 , '+555'         FROM dual UNION SELECT
 5 , '-4444'        FROM dual UNION SELECT
 6 , '-1234.1'      FROM dual UNION SELECT
 7 , ' 1234 '       FROM dual UNION SELECT
 8 , '.0'           FROM dual UNION SELECT
 9 , '.99'          FROM dual UNION SELECT
10 , '-.23.23'      FROM dual UNION SELECT
11,  '- 4444'       FROM dual UNION SELECT
12 , '12 34'        FROM dual UNION SELECT
13 , '1,234.56'     FROM dual UNION SELECT
14 , 'drei'         FROM dual)
SELECT -- NLS_NUMERIC_CHARACTERS = '.,'
 id,num_as_str
,CASE WHEN regexp_like( num_as_str,'^(\ )*(-|\+)?[0-9]*\.?[0-9]*(\ )*$')
      THEN to_number(num_as_str)
 END to_num
FROM t
ORDER BY id;
```

Top 13: Is numeric (NLS_NUMERIC_CHARACTERS = '.,') RESULT

ID	NUM_AS_STR	TO_NUM
1	1234	1234
2	12.34	12.34
3	1234.	1234
4	+555	555
5	-4444	-4444
6	-1234.1	-1234.1
7	1234	1234
8	.0	0
9	.99	.99
10	-.23.23	
11	- 4444	
12	12 34	
13	1,234.56	
14	drei	

Top 14: Check Valid Oracle Identifier (quoted and nonquoted) SQL(1/2)

```
WITH t (id, ora_id                          ) AS
( SELECT
 -- valid nonquoted identifiers (stored upper case)
  1, 'V$VALID_IDENTIFIER_#10'               FROM dual UNION SELECT -- upper, digits + special char $_#
  2, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ_$#0'       FROM dual UNION SELECT -- max 30 characters
  3, 'ÄÖÜ'                                  FROM dual UNION SELECT -- german mutated vowels
  4, 'TABLE'                                FROM dual UNION SELECT -- reserved words cannot be checked !!!
 -- valid "quoded identifiers" (stored case sensitive)
 11, '"_$#.,- aA0123"'                      FROM dual UNION SELECT -- punct, upper, lower, digit char
 10, '"abcdefghijklmnopqrstuvwxyz_$#0"'     FROM dual UNION SELECT -- max 30 characters
 13, '"$()*+.?[\^{|"'                       FROM dual UNION SELECT -- regex characters as identifier
 14, '"TABLE"'                              FROM dual UNION SELECT -- reserved word TABLE
 15, '"Table"'                              FROM dual UNION SELECT -- reserved word Table
 16, '" "'                                  FROM dual UNION SELECT -- even blank
 -- invalid identifiers
 20, '9A'                                   FROM dual UNION SELECT -- not starting with digit
 21, '_'                                    FROM dual UNION SELECT -- not starting with _
 22, '$'                                    FROM dual UNION SELECT -- not starting with $
 26, 'b'                                    FROM dual UNION SELECT -- no lower chars (stored upper)
 27, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ_$#01'      FROM dual UNION SELECT -- nonquoted too long
 28, '"abcdefghijklmnopqrstuvwxyz_$#01"'    FROM dual              -- quoted too long
)
```

Top 14: Check Valid Oracle Identifier (quoted and nonquoted) SQL(2/2)

```
SELECT
  id
, ora_id
, CASE WHEN regexp_like(ora_id, '^[[:upper:]][[:upper:][:digit:]_$#]{0,29}$')
       THEN '  X'  END chk_nonquot
, CASE WHEN regexp_like(ora_id, '^"[^"]{1,30}"$')
       THEN '  X'  END chk_quot
, CASE WHEN regexp_like(ora_id, '^(([[:upper:]][[:upper:][:digit:]_$#]{0,29})|(^"[^"]{1,30}"))$')
       THEN '  X'  END chk_valid
FROM t
ORDER BY id;
```

Top 14: Check Valid Oracle Identifier (quoted and nonquoted) RESULT

ID	ORA_ID	CHK_NONQUO	CHK_QUOT	CHK_VALID
1	V$VALID_IDENTIFIER_#10	X		X
2	ABCDEFGHIJKLMNOPQRSTUVWXYZ_$#0	X		X
3	ÄÖÜ	X		X
4	TABLE	X		X
10	"abcdefghijklmnopqrstuvwxyz_$#0"		X	X
11	"_$#.,- aA0123"		X	X
13	"$()*+.?[\^{|"		X	X
14	"TABLE"		X	X
15	"Table"		X	X
16	" "		X	X
20	9A			
21	_			
22	$			
26	b			
27	ABCDEFGHIJKLMNOPQRSTUVWXYZ_$#01			
28	"abcdefghijklmnopqrstuvwxyz_$#01"			

Top 14: Create Table with valid identifiers ?!?! RESULT

```
create table VALID_IDENTIFIERS
(
 V$VALID_IDENTIFIER_#10                number,
 ABCDEFGHIJKLMNOPQRSTUVWXYZ_$#0        number,
 ÄÖÜ                                   number,
 "abcdefghijklmnopqrstuvwxyz_$#0"      number,
 "_$#.,- aA0123"                       number,
 "$()*+.?[\^{|"                        number,
 "TABLE"                               number,
 "Table"                               number,
 " "                                   number,
 b                                     number
)
/
```

Table created

Top 15: Remove consecutive duplicates

```
SELECT regexp_replace('AB,AB,CD,EF,GH,IJ,IJ,IJ,AB'
                      ,'([^,]*)(,\1)+($|,)'
                      ,'\1\3') ab_ab_cd_ef_gh_ij_ij_ij_ab
  FROM dual;


AB_AB_CD_EF_GH_IJ_IJ_IJ_AB
---------------------------
AB,CD,EF,GH,IJ,AB
```

Top 16: Wrap Text (&line_size) SQL

```
with t as (select
'Kasperl und sein Freund Seppel hatten ihr zum Geburtstag eine
neue Kaffeemühle geschenkt, die hatten sie selbst erfunden.'
text from dual)
select --text
 regexp_replace(
        replace(text,chr(10),' ')
        ,'(.{1,&line_size})','\1'||chr(10))                    wrap_char
,regexp_replace(
        replace(text,chr(10),' ')
        ,'(.{1,&line_size})(\W|$)','\1'||chr(10))              wrap_word
,regexp_replace(
        replace(text,chr(10),' ')
        ,'(.{1,&line_size})(\W|$)|(.{1,&line_size})','\1\3'||chr(10)) wrap_word_cut
from t;
```

Top 16: Wrap Text (line_size=20) RESULT

```
WRAP_CHAR            | WRAP_WORD            | WRAP_WORD_CUT
-------------------- | -------------------- | --------------------
Kasperl und sein Fre | Kasperl und sein     | Kasperl und sein
und Seppel hatten ih | Freund Seppel hatten | Freund Seppel hatten
r zum Geburtstag ein | ihr zum Geburtstag   | ihr zum Geburtstag
e neue Kaffeemühle g | eine neue            | eine neue
eschenkt, die hatten | Kaffeemühle          | Kaffeemühle
 sie selbst erfunden | geschenkt, die       | geschenkt, die
.                    | hatten sie selbst    | hatten sie selbst
                     | erfunden.            | erfunden.
```

Top 16: Wrap Text (line_size=10) RESULT

```
WRAP_CHAR            | WRAP_WORD            | WRAP_WORD_CUT
-------------------- | -------------------- | --------------------
Kasperl un           | Kasperl              | Kasperl
d sein Fre           | und sein             | und sein
und Seppel           | Freund               | Freund
 hatten ih           | Seppel               | Seppel
r zum Gebu           | hatten ihr           | hatten ihr
rtstag ein           | zum                  | zum
e neue Kaf           | Geburtstag           | Geburtstag
feemühle g           | eine neue            | eine neue
eschenkt,            | Kaffeemühle          | Kaffeemühl
die hatten           | geschenkt,           | e
 sie selbs           | die hatten           | geschenkt,
t erfunden           | sie selbst           | die hatten
.                    | erfunden.            | sie selbst
                     |                      | erfunden.
```

Top 17: Convert camelCase to UPPER_CASE code style and vice versa

```
SELECT
 upper(
 regexp_replace(
 'CamelToUpperUnderscore', '([[:upper:]])', '_\1', 2)) "CamelToUpperUnderscore"
,REPLACE(
 initcap(
 'UPPER_UNDERSCORE_TO_CAMEL'), '_')                    "UPPER_UNDERSCORE_TO_CAMEL"
FROM dual;


CamelToUpperUnderscore     | UPPER_UNDERSCORE_TO_CASE
-------------------------- | ------------------------
CAMEL_TO_UPPER_UNDERSCORE  | UpperUnderscoreToCamel
```

Top 18: Remove Comments SQL

```
WITH t AS (SELECT
 q'<SELECT /* this select shows how to remove multiline
          and single line comments
          limitation: don't use comments within strings */
regexp_replace(regexp_replace(s  -- text
,'\/\*.*?\*\/', '', 1, 0, 'n')   -- first remove multiline comment
,'\-\-.*$'    , '', 1, 0, 'm')   -- then remove single line comment
please_comment_regexes
FROM t>' s
    FROM dual)
SELECT /* this select shows how to remove multiline
          and single line comments
          limitation: don't use comments within strings */
regexp_replace(regexp_replace(s -- text
,'\/\*.*?\*\/', '', 1, 0, 'n')  -- first remove multiline comment
,'\-\-.*$'    , '', 1, 0, 'm')  -- then remove single line comment
please_comment_regexes
FROM t;
```

Top 18: Remove Comments RESULT

```
PLEASE_COMMENT_REGEXES
----------------------------------------
SELECT
regexp_replace(regexp_replace(s
,'\/\*.*?\*\/', '', 1, 0, 'n')
,'\-\-.*$'    , '', 1, 0, 'm')
please_comment_regexes
FROM t
```

Top 19: REGEXP_EXTRACT PL/SQL for extracting multiple matches

```
CREATE OR REPLACE FUNCTION regexp_extract(srcstr     VARCHAR2
                                         ,pattern    VARCHAR2
                                         ,replacestr VARCHAR2 DEFAULT NULL
                                         ,position   NUMBER DEFAULT 1
                                         ,occurrence NUMBER DEFAULT 0
                                         ,modifier   VARCHAR2 DEFAULT NULL)
 -- extract multiple matches
 -- based on regexp_replace removes everything that is not matched (and optionally replaced)
 -- replacestr = null returns complete match without modification
 -- overcomes limitation of regexp_substr ( ... occurrence => 0) argument 0 is out of range
 -- limitation: backreference \9 cannot be used
 RETURN VARCHAR2 IS
  c VARCHAR2(1) := chr(1); -- any never used character
  v_pattern    VARCHAR2(4000) := pattern;
  v_replacestr VARCHAR2(4000) := replacestr;
BEGIN
  IF replacestr IS NULL THEN
    v_pattern    :=  '(' || REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(pattern,
                          '\9'),'\8','\9'),'\7','\8'),'\6','\7'),'\5','\6'),'\4','\5'),'\3','\4'),'\2','\3'),'\1','\2')|| ')';
    v_replacestr := '\1';
  END IF;
  -- find/replace and enclose substring by c, enclose the whole string by c, remove everything enclosed by c
  RETURN   regexp_replace(c || regexp_replace(srcstr, v_pattern, c || v_replacestr || c, position, occurrence, modifier) || c
                         ,c || '.*?' || c);
END;
```

Top 20.1: Compare Performance "Count" Traditional vs. Regexp Setup

```
-- Create table books ...
create table books
( author      varchar2(20)
 ,title       varchar2(25)
 ,text_v2     varchar2(4000) -- preface
 ,text_clob   clob           -- text
);

-- Insert 500 books ...
insert into books ....

-- Check/Select count, sum(length(text ...) books ...
SELECT  COUNT(1)                cnt_books
       ,sum(length(text_v2))    sum_length_text_v2
       ,sum(length(text_clob)) sum_length_text_clob
  FROM books;

CNT_BOOKS | SUM_LENGTH_TEXT_V2 | SUM_LENGTH_TEXT_CLOB
-----------------------------------------------------
      500 |          1,000,000 |           10,000,000
```

Top 20.2: Performance Traditional Workaround (text_v2)

```
SELECT SUM(length(text_v2) - length(REPLACE(text_v2,'Hotzenplotz'))) / length('Hotzenplotz') cnt_str
  FROM books where rownum <= 500;
   CNT_STR
----------
      1000
Executed in 0.29 seconds

SELECT SUM(length(text_v2) - length(REPLACE(text_v2, '.'))) cnt_dot
  FROM books where rownum <= 500;
   CNT_DOT
----------
     10000
Executed in 0.37 seconds

SELECT SUM(length(text_v2)) sum_length
  FROM books where rownum <= 500;
SUM_LENGTH
----------
   1000000
Executed in 0.418 seconds
```

Top 20.3: Performance Regexp_count (text_v2)

```
SELECT SUM(regexp_count(text_v2,'Hotzenplotz' )) cnt_str
  FROM books where rownum <= 500;
   CNT_STR
----------
      1000
Executed in 0.613 seconds

SELECT SUM(regexp_count(text_v2,'\.')) cnt_dot
  FROM books where rownum <= 500;
   CNT_DOT
----------
     10000
Executed in 0.578 seconds

SELECT SUM(regexp_count(text_v2,'.',1,'n')) sum_length
  FROM books where rownum <= 500;
SUM_LENGTH
----------
   1000000
Executed in 0.475 seconds
```

Top 20.4: Performance Traditional Workaround (text_clob)

```
SELECT SUM(length(text_clob) - length(REPLACE(text_clob,'Hotzenplotz'))) / length('Hotzenplotz') cnt_str
  FROM books where rownum <= 50;
   CNT_STR
----------
      1000
Executed in 0.381 seconds

SELECT SUM(length(text_clob) - length(REPLACE(text_clob, '.'))) cnt_dot
  FROM books where rownum <= 50;
   CNT_DOT
----------
     10000
Executed in 0.455 seconds

SELECT SUM(length(text_clob)) sum_length
  FROM books where rownum <= 50;
SUM_LENGTH
----------
   1000000
Executed in 0.367 seconds
```

Top 20.5: Performance Regexp_count (text_clob)

```
SELECT SUM(regexp_count(text_clob,'Hotzenplotz' )) cnt_str
  FROM books where rownum <= 50;
   CNT_STR
----------
      1000
Executed in 1.911 seconds

SELECT SUM(regexp_count(text_clob,'\.')) cnt_dot
  FROM books where rownum <= 50;
   CNT_DOT
----------
     10000
Executed in 11.3 seconds

SELECT SUM(regexp_count(text_clob,'.',1,'n')) sum_length
  FROM books where rownum <= 50;
SUM_LENGTH
----------
   1000000
Executed in 505.302 seconds
```

Top 20.5: Compare Performance "Count" Traditional vs. Regexp RESULT

Pattern	Datatype	'Hotzenplotz'	'\.'	'.' (n)
sum matches		1,000	10,000	1,000,000
Traditional "workaround" count	varchar2	0.4 s	0.4 s	0.4 s
REGEXP_COUNT	varchar2	0.6 s	0.6 s	0.5 s
Traditional "workaround" count	clob	0.4 s	0.5 s	0.4 s
REGEXP_COUNT	clob	1.9 s	11.6 s	505.3 s

… there seems to be a performance problem (bug ?!?) with REGEXP for CLOBS

… for simple patterns but lot of matches !?!

… further (more sophisticated) analysis has to be done …

Literature / Tools

*** Oracle Database Development Guide**

docs.oracle.com/database/122/ADFNS/regexp.htm#ADFNS1013

*** Oracle Regular Expressions Pocket Reference by Jonathan Gennick, O'Reilly & Associates**

2003 First Edition, Oracle Version 10, no updated edition to cover Oracle Version 11, 12 !!!

*** Mastering Regular Expressions by Jeffrey E. F. Friedl, O'Reilly & Associates**

2006 Third Edition, very detailed, coveres many languages, flavors

regex.info/book.html

*** Regular Expression Cookbook: Detailed Solution in Eight Programming Languages**

2012 Second Edition

regular-expressions.info/cookbook.html

*** Online Regexp test tools**

regexr.com	online, free
regex101.com	online, free

Vielen Dank für Ihre Aufmerksamkeit!

Joachim Forster

Lisa Dräxlmaier GmbH

Landshuter Straße 100

D-84137 Vilsbiburg

E-Mail joachim.forster@draexlmaier.com

Internet: www.draexlmaier.com