



Regular Expression Functions

Function name	Description
<code>REGEXP_LIKE</code>	Similar to the LIKE operator but allows for the use of regular expressions in matching
<code>REGEXP_REPLACE</code>	Search and replace text using regular expression pattern
<code>REGEXP_INSTR</code>	Searches for a string using regular expression pattern and returns the position when match is found
<code>REGEXP_SUBSTR</code>	Searches for a string using regular expression pattern and returns the matched substring
<code>REGEXP_COUNT</code>	Returns the number of times a pattern appears in the string.

REGEXP_COUNT

Returns the number of times a pattern appears in the string.

```
REGEXP_COUNT( <subject> , <pattern> [ , <position> , <parameters> ] )
```

Optional:

- **position** Number of characters from the beginning of the string where the function starts searching for matches.
Default: `1` (the search for a match starts at the first character on the left)
- **parameters** String of one or more characters that specifies the parameters used for searching for matches. Supported values:

<code>c</code>	Enables case-sensitive matching.
<code>i</code>	Enables case-insensitive matching.
<code>m</code>	Enables multi-line mode (i.e. meta-characters <code>^</code> and <code>\$</code> mark the beginning and end of any line of the subject). By default, multi-line mode is disabled (i.e. <code>^</code> and <code>\$</code> mark the beginning and end of the entire subject).
<code>e</code>	Extracts sub-matches; applies only

	to <u>REGEXP_INSTR</u> , <u>REGEXP_SUBSTR</u> , <u>REGEXP_SUBSTR_ALL</u> , and the aliases for these functions.
S	Enables the POSIX wildcard character . to match \n . By default, wildcard character matching is disabled.

- Default: **C**

```
select regexp_count('It was the best of times,
it was the worst of times', '\\bwas\\b', 1) as "result" from dual;
```

```
-- OUTPUT
+-----+
| result |
+-----+
|      2 |
+-----+
```

```
select regexp_count('It was the best of times,
it was the worst of times', '\\bit\\b', 1) as "result" from dual;
```

```
-- OUTPUT
+-----+
| result |
+-----+
|      1 |
+-----+
```

```
select regexp_count('It was the best of times,
it was the worst of times', '\\bit\\b', 1, 'i') as "result" from dual;
```

```
-- OUTPUT
+-----+
| result |
+-----+
|      2 |
+-----+
```

```
create or replace table overlap (id number, a string);
insert into overlap values (1, 'abc,def,ghi,jkl,');
insert into overlap values (2, 'abc,,def,,ghi,,jkl,');
```

```
select id, regexp_count(a, '[:punct:][:alnum:]+[:punct:]', 1, 'i')
from overlap;
```

```
-- PATTERN
,abc, - is first count
def - not counted
,ghi, - 2nd counted
jkl - is not counted
so total 2 for 'abc,def,ghi,jkl,'
```

```
-- OUTPUT
+-----+
| ID | REGEXP_COUNT(A, '[:PUNCT:][:ALNUM:]+[:PUNCT:]', 1, 'I') |
+-----+
| 1 | 2 |
| 2 | 4 |
+-----+
```

```
SELECT REGEXP_COUNT('QQQABTERABABTBDF', '[Q]{2}'); -- OUTPUT: 1
-- BECAUSE IT IS COUNTING Q AS A PAIR 1, MEANS 'QQ' IS COUNT AS 1
-- AND THERE IS ONLY 1 'QQ' IN GROUP

SELECT REGEXP_COUNT('QQQABTERABABTBDF', '[Q]{1}'); -- OUTPUT: 3

SELECT REGEXP_COUNT('QQQABTERABABTBDF', '[Q]{3}'); -- OUTPUT: 1
-- 'QQQ' IS COUNT AS 1 AND THERE IS ONLY 1 'QQQ' IN GROUP

SELECT REGEXP_COUNT('QQQABTERABABTBDF', '[AB]{2}'); -- OUTPUT: 3
-- 'AB' IS COUNT AS 1 AND THERE IS ONLY 3 'AB' IN GROUP
```

More In-depth

```
SELECT REGEXP_COUNT('QQQABTERABABTBDF', '[AB]{2}'); -- 3
SELECT REGEXP_COUNT('QQQABTERABABTBDF', '[AB]{5}'); -- 0
SELECT REGEXP_COUNT('QQQABTERABABTBDF', '[AB]{4}'); -- 1
SELECT REGEXP_COUNT('QQQABTERABABTBDF', '[AB]{1}'); -- 7
```

- This query counts the occurrences of the pattern `[AB]{4}` in the string `'QQQABTERABABTBDF'`. The pattern `[AB]{4}` matches any consecutive occurrence of either 'A' or 'B' exactly four times. In this case, it will count the number of times 'AAAA', 'AAAB', 'AABA', 'ABAA', 'BAAA', 'AABB', 'ABAB', 'BABA', 'BBAA', 'ABBB', 'BAAB', or 'BBBA' appears consecutively in the string.

The expected output of this query is `1`. Here's the breakdown of the matches:

- 'ABAB' at index 11-14

- This query counts the occurrences of the pattern `[AB]{2}` in the string `'QQQABTERABABTBDF'`. The pattern `[AB]{2}` matches any consecutive occurrence of either 'A' or 'B' exactly twice. In this case, it will count the number of times 'AB' or 'BA' appears consecutively in the string.

The expected output of this query is `3`. Here's the breakdown of the matches:

- 'AB' at index 4-5
- 'AB' at index 9-10
- 'AB' at index 11-12

EXAMPLE:

- DNA sequence of a mouse. Need to find the number of times the sequence of Cytosine, Adenine, Thymine (cat) proteins occur.

```
SELECT REGEXP_COUNT ('ccacctttccctccactcagttctcacctgtaaagcgccctccctcatc
cccatgcccccttacctgcagggtagagtaggctagaaaccagaga
gctccaagctccatctgtggagaggtgccatccttgggctgcgagaga
ggagaatttgccaaagctgcctgtttgaacgatggagacatgattgc
ccgtaaagggtcctgaatgcatgagatgtcttcgagagtaccggttac
gggttaaaaggtcatgagacttcgatcattacgatcgtggttaacacac
atatgagtatagagacacattggccaagagttgagattgagag', 'cat') as "cat count"
from dual;
```

◦ OUTPUT

	cat count
1	9

- All rows where there is an 'i' in the first name

```
SELECT first_name as "First Name"
FROM hr.employees
WHERE REGEXP_COUNT (first_name, 'i') > 0;
```

◦ OUTPUT

	First Name
1	David
2	Shelli
3	Amit
4	Elizabeth
5	David
6	Shelli
7	Amit
8	Elizabeth
9	David
10	Shelli
11	Amit
12	Elizabeth
13	David
14	Shelli
15	Amit
16	Elizabeth
17	David
18	Shelli
19	Amit
20	Elizabeth
21	David
22	Shelli
23	Amit
24	Elizabeth
25	David
26	Shelli
27	Amit
28	Elizabeth
29	David
30	Shelli
31	Amit
32	Elizabeth
33	Lindsey
34	William
35	Patrick
36	Winston

REGEXP_REPLACE

Returns the subject with the specified pattern (or all occurrences of the pattern) either removed or replaced by a replacement string. **If no matches are found, returns the original subject.**

```
REGEXP_REPLACE( <subject> , <pattern> [ , <replacement> ,
<position> , <occurrence> , <parameters> ] )
```

- **replacement**

String that replaces the substrings matched by the pattern. If an empty string is specified, the function removes all matched patterns and returns the resulting string.

Default: '' (empty string).

- **position**

Number of characters from the beginning of the string where the function starts searching for matches.

Default: 1 (the search for a match starts at the first character on the left)

EXAMPLE

- Reformat phone number from ###.###.#### to 1 (###)-###-####

```
SELECT ph_num, REGEXP_REPLACE(ph_num,
'([[:digit:]]{3})\.[[:digit:]]{3})\.[[:digit:]]{4})',
'1 (\1)-\2-\3') RESULT
FROM phone_number
```

	PH_NUM	RESULT
1	404.777.9311	1 (404)-777-9311
2	404.867.5309	1 (404)-867-5309
3	404.436.3566	1 (404)-436-3566
4	505.555.5555	1 (505)-555-5555

Meta Character	Description
[[:digit:]]{3}	Three digits (group 1)
\.	Then a '.' (Since the '.' is a META Character, we have to use the \ to 'escape' it)
[[:digit:]]{3}	Three digits (group 2)
\.	Then a '.'
[[:digit:]]{4}	Four digits (group 3)

- Get the username from email id of a user

```
SELECT REGEXP_REPLACE('adi.singh1069@gmai.com', '@.*\.(com)');
```

-- OUTPUT
adi.singh1069

-- HOW
basically the above code is like this
SELECT REGEXP_REPLACE('adi.singh1069@gmai.com', '@.*\.(com)', '');
so we are replacing @gmail.com from the string by a empty string.
As we know the default replacement is empty string.

REGEXP_LIKE

REGEXP_LIKE is similar to the **LIKE** function, but with POSIX extended regular expressions instead of SQL LIKE pattern syntax. It supports more complex matching conditions than LIKE.

```
REGEXP_LIKE( <subject> , <pattern> [ , <parameters> ] )
```

Returns

The data type of the returned value is BOOLEAN.

The function implicitly anchors a pattern at both ends (i.e. `' '` automatically becomes `'^$'`, and `'ABC'` automatically becomes `'^ABC$'`). To match any string starting with ABC, the pattern would be `'ABC.*'`.

EXAMPLE

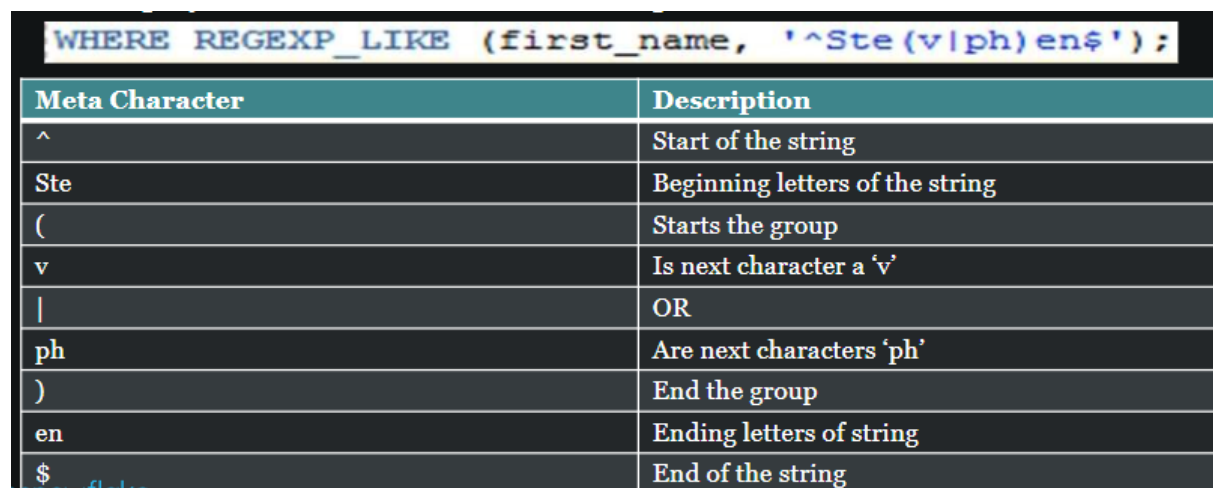
All employees first name of Steven or Stephen



```
SELECT first_name as "First Name"
, last_name as "Last Name"
, hire_date as "Hire Date"
FROM hr.employees
WHERE REGEXP_LIKE (first_name, '^Ste(v|ph)en$');
```

	First Name	Last Name	Hire Date
1	Steven	King	17-JUN-03
2	Steven	Markle	08-MAR-08
3	Stephen	Stiles	26-OCT-05

BREAKDOWN



```
WHERE REGEXP_LIKE (first_name, '^Ste(v|ph)en$');
```

Meta Character	Description
<code>^</code>	Start of the string
<code>Ste</code>	Beginning letters of the string
<code>(</code>	Starts the group
<code>v</code>	Is next character a 'v'
<code> </code>	OR
<code>ph</code>	Are next characters 'ph'
<code>)</code>	End the group
<code>en</code>	Ending letters of string
<code>\$</code>	End of the string

REGEXP_SUBSTRING

Returns the substring that matches a regular expression within a string. If no match is found, returns NULL.

```
REGEXP_SUBSTR( <subject> , <pattern> [ , <position>
[ , <occurrence> [ , <regex_parameters> [ , <group_num> ] ] ] )
```

Returns

The function returns a value of type VARCHAR that is the matching substring.

EXAMPLE

```
select id, regexp_substr(string1, 'the\\W+\\w+') as "RESULT"
  from demo2
 order by id;
+-----+
| ID | RESULT          |
+-----+
| 2 | the best        |
| 3 | the string      |
| 4 | NULL            |
+-----+
```

EXAMPLE 2 - WORK OF 'e' parameter

```
SELECT regexp_substr('@This is a @test string.',
 '@(\\w+)\\s+(\\w+)', 1, 1);

-- OUTPUT
@This is

SELECT regexp_substr('@This is a @test string.',
 '@(\\w+)\\s+(\\w+)', 1, 1, 'e'); -- here after e, the value of group is
-- 1 by default.

-- OUTPUT
This

SELECT regexp_substr('@This is a @test string.',
```



```
'@(\w+)\s+(\w+)', 1, 1, 'e', 2); -- 2 represent which group to
-- extract, here in group 1 - "this" & in group 2 - "is"

-- OUTPUT
is
```

'e' - Extract the sub match - here `@(\w+)\s+(\w+)` we got output like this first without `e` `@This is`, but after `e` we get `This`. So it takes sub match from the pattern where there is no word or num like `@, the etc`, it takes sub match from the pattern `(\w+)\s+(\w+)` and it only takes one word.

REGEXP_INSTR

Returns the position of the specified occurrence of the regular expression pattern in the string subject. If no match is found, returns 0.

```
REGEXP_INSTR( <subject> , <pattern> [ , <position>
[ , <occurrence> [ , <option> [ , <regexp_parameters>
[ , <group_num> ] ] ] ] )
```

Arguments

Required:

- **subject** The string to search for matches. **pattern** Pattern to match.

Optional:

- **position** Number of characters from the beginning of the string where the function starts searching for matches.
Default: `1` (the search for a match starts at the first character on the left)
Positions are 1-based, not 0-based. For example, the position of the letter "M" in "MAN" is 1, not 0.
- **occurrence** Specifies which occurrence of the pattern to match. The function skips the first `occurrence - 1` matches.
Default: `1`
- **option** Specifies whether to return the offset of the first character of the match (`0`) or the offset of the first character following the end of the match (`1`).
Default: `0`

- **regexp_parameters** String of one or more characters that specifies the regular expression parameters used for searching for matches. The supported values are:
 - **c**: case-sensitive.
 - **i**: case-insensitive.
 - **m**: multi-line mode.
 - **e**: extract sub-matches.
 - **s**: the '.' wildcard also matches newline.

EXAMPLE

- Search for a matching string. In this case, the string is “nevermore” followed by a single decimal digit (e.g. “nevermore1”):

```
select id, string1,
       regexp_substr(string1, 'nevermore\d') AS "SUBSTRING",
       regexp_instr( string1, 'nevermore\d') AS "POSITION"
from demo1
order by id;
```

ID	STRING1	SUBSTRING	POSITION
1	nevermore1, nevermore2, nevermore3.	nevermore1	1

- Search for a matching string, but starting at the 5th character in the string, rather than at the 1st character in the string:

```
select id, string1,
       regexp_substr(string1, 'nevermore\d', 5) AS "SUBSTRING",
       regexp_instr( string1, 'nevermore\d', 5) AS "POSITION"
from demo1
order by id;
```

ID	STRING1	SUBSTRING	POSITION
1	nevermore1, nevermore2, nevermore3.	nevermore2	13

Breakdown

1. The input string is **'nevermore1, nevermore2, nevermore3.'**

2. The regular expression pattern `'nevermore\\d'` matches the substring `'nevermore1'` within the input string. The pattern consists of the characters "nevermore" followed by a single digit.
3. The `regexp_instr` function starts searching for the pattern from the fifth position in the input string.
4. The function finds a match for the pattern `'nevermore\\d'` starting from position 13 in the input string.
5. Therefore, the result of `regexp_instr('nevermore1, nevermore2, nevermore3.', 'nevermore\\d', 5)` is 13, indicating that the pattern `'nevermore\\d'` is found at position 13 in the input string when searching from the fifth position.
 - Counting of Position is Starting from 1.

- Search for a matching string, but look for the 3rd match rather than the 1st match:

```
select id, string1,
       regexp_substr(string1, 'nevermore\\d', 1, 3) AS "SUBSTRING",
       regexp_instr( string1, 'nevermore\\d', 1, 3) AS "POSITION"
from demo1
order by id;
```

ID	STRING1	SUBSTRING	POSITION
1	nevermore1, nevermore2, nevermore3.	nevermore3	25

- This query is nearly identical the previous query, but this shows how to use the `option` parameter to indicate whether you want the position of the matching expression, or the position of the first character after the matching expression:

```
select id, string1,
       regexp_substr(string1, 'nevermore\\d', 1, 3) AS "SUBSTRING",
       regexp_instr( string1, 'nevermore\\d', 1, 3, 0) AS "START_POSITION",
       regexp_instr( string1, 'nevermore\\d', 1, 3, 1) AS "AFTER_POSITION"
from demo1
order by id;
```

```
-- OUTPUT
STARTING_POSITION - 25
AFTER_POSITION - 35 -- position of .
```

- This query shows that if you search for an occurrence beyond the last actual occurrence, the position returned is 0:

```
select id, string1,
       regexp_substr(string1, 'nevermore', 1, 4) AS "SUBSTRING",
       regexp_instr( string1, 'nevermore', 1, 4) AS "POSITION"
from demo1
order by id;
```

ID	STRING1	SUBSTRING	POSITION
1	nevermore1, nevermore2, nevermore3.	NULL	0

Example

looks for:

- the word “the”
- followed by one or more non-word characters (for example, the whitespace separating words)
- followed by one or more word characters.

“Word characters” include not only the letters a-z and A-Z, but also the underscore (“_”) and the decimal digits 0-9, but not whitespace, punctuation, etc.

```
select string1,
       regexp_substr(string1, 'the\\W+\\w+') as "SUBSTRING",
       regexp_instr(string1, 'the\\W+\\w+') as "POSITION"
from demo2
order by id;
```

OUTPUT

STRING	SUBSTRING	POSITION
It was the best of times, it was the worst of times.	the best	8
In the string the extra spaces are redundant.	the string	7
A thespian theater is nearby.	NULL	0

