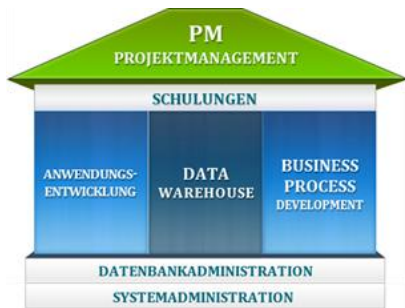


QuickReference v1.1

Oracle SQL Advanced Functions

-english-

author
Sven-Uwe Weller



syntegris information solutions GmbH
Hermannstr. 54-56
63263 Neu-Isenburg
Germany
Sven.Weller@syntegris.de
www.syntegris.de

Other available QuickReferences are:

- > Oracle SQL Basic Functions
- > Oracle Apex APIs

Regular expressions

Functions

regexp_count (source, pattern, position, match_params) (11.1)
regexp_instr (source, pattern, position, occur, return_option, match_params, subexpr)
regexp_like (source, pattern, match_params)
regexp_replace (source, pattern, replace, position, occur, match_params)
regexp_substr (source, pattern, position, occur, match_params, subexpr)

Parameters

source input string
 pattern search pattern
 replace replacement text
 backreferences allowed
 position starting position for the search
 occur number of occurrence for multiple matches
 return_option 0|1 which position to return
 0 first matching character
 1 position after the match
 match_params special settings for the search
 n "." includes newline chars
 m multi lines. ^ and \$ work for each line
 i case insensitive search
 c case sensitive search
 x ignore whitespaces
 subexpr 1..9 which () to return (11.1)

Regex Syntax (Posix)

. any character but newline
 * 0 or more greedy (10.2)
 + 1 or more greedy
 ? 0 or 1 makes other expr. non greedy
 | OR alternative expression
 \ backslash can be escape char
 ^ of the full string
 \$ end of the complete string
 () subexpression
 {n,m} n to m repetitions
 \n 1..9 backreference for ()
 [] any list match any char in this list
 [^] any not in list
 [=] equivalence class [=a=] matches a|A|á|À
 [::] character class
 [:digit:] 0-9 numerical chars
 [:xdigit:] A-Fa-f0-9 hexadecimal digits
 [:alpha:] a-zA-Z text
 [:alnum:] 0-9a-zA-Z no whitespaces|tabs
 [:space:] whitespaces, tabs
 [:ascii|blank|cntrl|graph|lower|upper|print|space|word:]

Meta characters (PERL syntax) (10.2)

\d single digit character
 \D non digit character
 \w word character
 \W non word character
 \s whitespace character
 \S non whitespace character
 \A beginning
 \Z end of the string
 ? after *,+,?, {n}, {n,}, {n,m} will make this non greedy

Back references

\1 references the first pair of parenthesis (). Up to 9 backreferences (\1 .. \9). Can be used in the pattern and the replace parameters. In subexpr just the number is used.

^Regex_(.*) examples

Examples

Count the number of words
 nvl(length(regexp_replace(str, '\s*\S*\s*', 'x')), 0)
 regexp_count(str, '(\w+)')
 regexp_count(str, '([[:alnum:]]+)')

str	result 1	result 2	result 3
This is just a test	5	5	5
_	1	1	0
two.words ?	2	2	1

Search for multiple strings (case insensitive)

regexp_like (str, 'ABC|xyz|999', 'i')

Extract only the numbers

regexp_substr(col, '[0-9]+') (until 10.1)
 regexp_replace(col, '[^0-9]') (all versions)
 regexp_replace(col, '\D+') (10.2)

Validate an email address

regexp_like('^[[:alnum:]]_%.+@[[:alnum:]]-]+\.[:alpha:]{2,4}\$')
 this allows only short domains

Validate an URL

regexp_like('http[s]?://[A-Za-z0-9-]+\.[A-Za-z]{2,5}/[A-Za-z0-9%_-.?=&@#]*\$')

Count dots in a string

regexp_count('This.is.my.website', '\.')

Find everything after the last dot

regexp_substr('This.is.my.website', '[^\.\+]') website

Third part (dot as delimiter) in a string

regexp_substr('This.is.my.website', '[^\.\+]', 1, 3) my

Get all strings starting with "ABC" from a list

Ltrim(regexp_replace(list, '(^|;)(ABC[[:alnum:]]+)', '\1\2'), ';')

list	result
ABC123;DEF123;ABC345	ABC123;ABC345
DEF123;ABCDEF	ABCDEF

Find and eliminate repeating strings

regexp_replace(str, '^(.+?)\1*\$', '\1')

str	result
ABCABC	ABC
333	3
A123A123A123A123	A123
ababX	ababX

get the next TD element from a html document after a search string

regexp_replace(htmlDoc, '^.*?<td>SearchStr</td>.*?<td>(w+)</td>.*\$', '\1', 1, 1, 'n') (10.2)

regexp_substr (htmlDoc, '^.*?<td>SearchStr</td>.*?<td>(w+)</td>.*\$', 1, 1, 'n', 1) (11.2)

{JSON} (12.1.0.2)

Functions

data is [not] json
json_exists(data, \$path on_error)
json_textcontains(column, \$path, searchstring)
json_query(data, \$path returning query_wrapper on_error)
json_table(data, \$path returning on_error COLUMNS columnlist)
json_value (data, \$path returning on_error)

hint: is json can be used in a check constraint to make sure that a column holds only valid json documents.

Syntax \$path (object or array)

```
$
[ { . { * | simplename | "complexname" } }
  |
  { [ { integer [TO integer]
    [, integer [TO integer]
    ]...
    }
  | *
  }
] ...
```

Parameters

data json varchar2, clob or a blob
 blobs need to add "FORMAT JSON"
 \$path json search path
 on_error action when return error happens
 ERROR|NULL|EMPTY|DEFAULT literal|TRUE|FALSE
 ON ERROR
 ERROR, NULL for all functions
 TRUE|FALSE for json_exists
 EMPTY for json_query
 DEFAULT for json_table, json_value
 returning return data type and formatting
 PRETTY ASCII

query_wrapper WITH|WITHOUT ARRAY WRAPPER
 columnlist json column list
 EXISTS PATH like json_exists
 PATH like json_query or json_value
 NESTED PATH single row unnesting
 FOR ORDINALITY row numbers

Examples

'{a:[5, 10, 15]}' is json true
 json_value('{a:[5, 10, 15]}', '\$.a[2]') 15
 json_value('{ "firstname": "Sven", "lastname": "Weller" }', '\$.lastname')
 default 'No last name found.' No last name found.

select * from json_table('{a:[5, 10, 15]}', '\$.a[*]')
 columns (RN for ordinality,
 VAL1 number path '\$');

-or-
 select * from json_table('{a:[5, 10, 15]}', '\$')
 columns (val2 varchar2(20) format json path '\$.a',
 nested path '\$.a' columns (val3 number path '\$[0]',
 val4 number path '\$[2]'));

RN	VAL1	VAL2	VAL3	VAL4
1	5	[5,10,15]	5	15
2	10			
3	15			

Analytic functions

Syntax

```
analytic_function([arguments])
OVER
  ([partition by ... ]
  [order by ... ]
  [ windowing_clause ])
```

Windowing clause:

```
{ ROWS | RANGE }
{ BETWEEN
  { UNBOUNDED PRECEDING
  | CURRENT ROW
  | value { PRECEDING | FOLLOWING }
  }
AND
  { UNBOUNDED FOLLOWING
  | CURRENT ROW
  | value { PRECEDING | FOLLOWING }
  }
| { UNBOUNDED PRECEDING
  | CURRENT ROW
  | value PRECEDING
  }
}
```

Analytic functions

avg (...) average
 count (*[distinct expr]) count
 dense_rank () rank with no gaps
 first_value (...) respect[ignore nulls] first value in set
hint: "respect[ignore nulls]" part is ansi conform since 11.2 (abbreviation=r/i)
 lag (... ,offset, default) r|i nulls previous row
 last_value (...) r|i nulls last value in set
 lead (... ,offset, default) r|i nulls next row
 listagg (... ,delimiter) within group (order by) (11.1)
 creates a list with delimiters
 max (...) maximum
 median (...) middle value
 min (...) minimum
 nth_value (... , n) from first|last r|i nulls value in nth row (11.2)

ntile (x) creates x buckets
 rank () ranking with gaps
 ratio_to_report (...) ratio of row to sum of set
 row_number () rownum for groups
 sum (...) sum

Analytic functions for statistics

corr|corrs_s|corr_k correlation functions
 covar_pop population covariance
 covar_samp sample covariance
 cume_dist cumulative distribution (>0 <=1)
 percent_rank cumulative distribution (>=0 <=1)
 first row = 0
 percentile_cont inverse distribution function
 percentile_disc inverse distribution function
 regr_... linear regression functions
 regr_slope|intercept|count|r2|avg|avg|sxx|syy|sxy
 stddev standard deviation (0 for 1 row)
 stddev_pop population standard deviation
 stddev_samp standard deviation (null for 1 row)
 var_pop population variance
 var_samp sample variance
 variance variance

Analytic functions for data mining (12.1.0.1)

```
cluster_details|distance|id|probability|set
highest probability cluster
feature_details|id|set|value
highest value feature (scoring)
prediction|cost|details|probability|set
a prediction based upon purpose
```

KEEP syntax

```
aggregate_function keep (dense_rank
first|last order by ...) [over (partition by
...)]
first|last rows in window, then aggregate
```

Example

Customer who bought the most
 select max(customer) keep (dense_rank first order by sum(itmcount) desc) as customer
 from orders o
 join orderitems oi on oi.orderid = o.id
 group by customer = John

Flashback

The flashback expression comes directly after the table expression just before the table alias.

Syntax

```
{ AS OF { SCN | TIMESTAMP }
| VERSIONS BETWEEN { SCN | TIMESTAMP }
| minvalue AND maxvalue
}
```

two query variations are possible
as of = Get the data how it was some time ago.
versions between = get all versions of the data for a specified period

Both versions can be applied to a **timestamp** value or to a **scn** number.

hint: flashback privilege on the table is needed to run a flashback query.

Pseudocolumns

VERSIONS_STARTSCN first scn of the row with this value
 VERSIONS_STARTTIME first time the row had this value
 VERSIONS_ENDSCN last scn of the row with this value
 VERSIONS_ENDTIME last time the row had this value
 VERSIONS_XID version number in raw
 VERSIONS_OPERATION type of change
 I after insert
 U during update
 D before delete

Examples

List of employees how they were yesterday

```
Select e.*
from emp as of timestamp systimestamp - interval '1' day e;
```

All changes during the last 10 minutes

```
Select versions_xid, versions_operation, e.*
from emp versions between
timestamp systimestamp - interval '10' minute
and systimestamp e;
```

This can return multiple results for a single row.

Subtotal grouping

Syntax

```
GROUP BY
  { expr
  | rollup
  | cube
  | grouping sets (grouping expression)
  }
```

Group clause

all versions calculate and add subtotal rows
rollup rolls up the dimensions from left to right
cube all combinations for all subtotals
grouping sets most flexible definition using sets
Hint: Master grouping sets, understand the others!

Transformations

```
rollup (a, b)
== cube (a, b)
== grouping sets ((a, b), (a), (b), ())
```

```
group by a, cube (b, c)
== group by grouping sets
((a, b, c), (a, b), (a, c), (b, c), ())
```

```
group by a, rollup (b, c)
== group by grouping sets
((a, b, c), (a, b), (a, c), (a), ())
```

Hint: () will add a line for a group over all rows.

Select clause functions

Group_id 0|1 find duplicate groups
Grouping 0|1 extra (superaggregate) row
Grouping_id 0..x multiple grouping combo

Examples

Count orders per region and total count

```
select region, count(*)
from orders o
group by rollup(region);
```

region	count(*)
America	1
Europe	2
UK	1
	4

Total sum price for each order and region

```
select decode(grouping(o.region), 1,'-world-',o.region) region
, decode(grouping_id(o.order#,o.region), 2,'-total-',3,'-world-',to_char(o.order#)) orders
, o.customer customer
, sum(itmcount) items#
, sum(i.price*oi.itmcount) price
from orders o
join orderitems oi on oi.orderid = o.id
join items i on i.id = oi.itemid
group by grouping sets((o.order#, o.customer, o.region),(o.region), ())
order by o.region, o.order#;
```

region	orders	customer	items#	amount
America	173	John	6	2210
America	-total-		6	2210
Europe	240	Hans	3	1299
Europe	241	Francois	3	450
Europe	-total-		6	1749
UK	170	Paul	1	999
UK	-total-		1	999
-world-	-world-		13	4958

Subquery factoring

Syntax

```
WITH query_name
  ([column1, column2, ...]) (11.1)
AS (subquery)
[search_clause] [cycle_clause] (11.2)
[, query_name AS (subquery)]...
```

search_clause:

```
{ SEARCH
  { DEPTH|BREADTH
  FIRST BY column1 [, column2] ...
  [ASC|DESC][NULLS FIRST|LAST]
  } SET ordering_column }
```

cycle_clause:

```
{ CYCLE column1 [, column2]...
  SET cycle_mark_column TO value
  DEFAULT no_cycle_value }
```

Typical usages include

- reuse repeating subqueries
- test data creation
- temporarily store the data (db links)
- code densification and clarification
- recursive logic

sample data

The following virtual tables are used during the selects on this page. With clause subquery factoring allows to create this test data on the fly and to experiment with different data values. The column naming for the orders table is available since 11g. In 9 or 10g add the column names as it is done for the items and orderitems subqueries.

with

```
orders (id, order#, customer,region) as (11.1)
(select 1, 173, 'John', 'America' from dual union all
select 2, 170, 'Paul', 'UK' from dual union all
select 3, 240, 'Hans', 'Europe' from dual union all
select 4, 241, 'Francois', 'Europe' from dual)
```

items as (9)

```
(Select 1 id, 'TV' item, 2000 price from dual union all
Select 2 id, 'Radio' item, 150 price from dual union all
Select 3 id, 'IR control' item, 15 price from dual union all
Select 4 id, 'PC' item, 999 price from dual)
```

orderitems as

```
(select 1 orderid, 1 num, 1 itemid, 1 itmcount from dual union all
select 1 orderid, 2 num, 2 itemid, 1 itmcount from dual union all
select 1 orderid, 3 num, 3 itemid, 4 itmcount from dual union all
select 2 orderid, 1 num, 4 itemid, 1 itmcount from dual union all
select 3 orderid, 1 num, 4 itemid, 1 itmcount from dual union all
select 3 orderid, 2 num, 2 itemid, 2 itmcount from dual union all
select 4 orderid, 1 num, 2 itemid, 3 itmcount from dual union all
select 5 orderid, 1 num, 4 itemid, 4 itmcount from dual)
```

/* end of test data creation */

```
select o.order#, o.customer, o.region, i.item, i.price, oi.itmcount
from orders o
join orderitems oi on oi.orderid = o.id
join items i on i.id = oi.itemid
order by oi.orderid, oi.num;
```

orderid	customer	region	item	price	itmcount
173	John	America	TV	2000	1
173	John	America	Radio	150	1
173	John	America	IR control	15	4
170	Paul	UK	PC	999	1
240	Hans	Europe	PC	999	1
240	Hans	Europe	Radio	150	2
241	Francois	Europe	Radio	150	3