SQL Window Functions Cheat Sheet

LearnSQL.com

LIST OF WINDOW FUNCTIONS

Aggregate Functions
- `avg()`
- `count()`
- `max()`
- `min()`
- `sum()`

Ranking Functions
- `row_number()`
- `rank()`
- `dense_rank()`

Distribution Functions
- `percent_rank()`
- `cume_dist()`

Analytic Functions
- `lead()`
- `lag()`
- `ntile()`
- `first_value()`
- `last_value()`
- `nth_value()`

AGGREGATE FUNCTIONS

- **`avg`***(expr)* – average value for rows within the window frame
- **`count`***(expr)* – count of values for rows within the window frame
- **`max`***(expr)* – maximum value within the window frame
- **`min`***(expr)* – minimum value within the window frame
- **`sum`***(expr)* – sum of values within the window frame

ORDER BY and `Window Frame`: Aggregate functions do not require an ORDER BY. They accept window frame definition (`ROWS`, `RANGE`, `GROUPS`).

RANKING FUNCTIONS

- **`row_number()`** – unique number for each row within partition, with different numbers for tied values
- **`rank()`** – ranking within partition, with gaps and same ranking for tied values
- **`dense_rank()`** – ranking within partition, with no gaps and same ranking for tied values

city	price	row_number	rank	dense_rank
		over(order by price)		
Paris	7	1	1	1
Rome	7	2	1	1
London	8.5	3	3	2
Berlin	8.5	4	3	2
Moscow	9	5	5	3
Madrid	10	6	6	4
Oslo	10	7	6	4

ORDER BY and Window Frame: `rank()` and `dense_rank()` require ORDER BY, but `row_number()` does not require ORDER BY. Ranking functions do not accept window frame definition (`ROWS`, `RANGE`, `GROUPS`).

DISTRIBUTION FUNCTIONS

- **`percent_rank()`** – the percentile ranking number of a row—a value in [0, 1] interval: (rank - 1) / (total number of rows - 1)
- **`cume_dist()`** – the cumulative distribution of a value within a group of values, i.e., the number of rows with values less than or equal to the current row's value divided by the total number of rows; a value in (0, 1] interval

`percent_rank() OVER(ORDER BY sold)`

city	sold	percent_rank
Paris	100	0
Berlin	150	0.25
Rome	200	0.5
Moscow	200	0.5
London	300	1

← without this row 50% of values are less than this row's value

`cume_dist() OVER(ORDER BY sold)`

city	sold	cume_dist
Paris	100	0.2
Berlin	150	0.4
Rome	200	0.8
Moscow	200	0.8
London	300	1

← 80% of values are less than or equal to this one

ORDER BY and Window Frame: Distribution functions require ORDER BY. They do not accept window frame definition (`ROWS`, `RANGE`, `GROUPS`).

ANALYTIC FUNCTIONS

- **`lead`***(expr, offset, default)* – the value for the row *offset* rows after the current; *offset* and *default* are optional; default values: *offset* = 1, *default* = NULL
- **`lag`***(expr, offset, default)* – the value for the row *offset* rows before the current; *offset* and *default* are optional; default values: *offset* = 1, *default* = NULL

`lead(sold) OVER(ORDER BY month)`

month	sold	
1	500	300
2	300	400
3	400	100
4	100	500
5	500	NULL

order by month

`lag(sold) OVER(ORDER BY month)`

month	sold	
1	500	NULL
2	300	500
3	400	300
4	100	400
5	500	100

order by month

`lead(sold, 2, 0) OVER(ORDER BY month)`

month	sold	
1	500	400
2	300	100
3	400	500
4	100	0
5	500	0

order by month; offset=2

`lag(sold, 2, 0) OVER(ORDER BY month)`

month	sold	
1	500	0
2	300	0
3	400	500
4	100	300
5	500	400

order by month; offset=2

- **`ntile`***(n)* – divide rows within a partition as equally as possible into *n* groups, and assign each row its group number.

`ntile(3)`

city	sold	
Rome	100	1
Paris	100	1
London	200	1
Moscow	200	2
Berlin	200	2
Madrid	300	2
Oslo	300	3
Dublin	300	3

ORDER BY and Window Frame: `ntile()`, `lead()`, and `lag()` require an ORDER BY. They do not accept window frame definition (`ROWS`, `RANGE`, `GROUPS`).

- **`first_value`***(expr)* – the value for the first row within the window frame
- **`last_value`***(expr)* – the value for the last row within the window frame

`first_value(sold) OVER (PARTITION BY city ORDER BY month)`

city	month	sold	first_value
Paris	1	500	500
Paris	2	300	500
Paris	3	400	500
Rome	2	200	200
Rome	3	300	200
Rome	4	500	200

`last_value(sold) OVER (PARTITION BY city ORDER BY month RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)`

city	month	sold	last_value
Paris	1	500	400
Paris	2	300	400
Paris	3	400	400
Rome	2	200	500
Rome	3	300	500
Rome	4	500	500

Note: You usually want to use RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING with `last_value()`. With the default window frame for ORDER BY, RANGE UNBOUNDED PRECEDING, `last_value()` returns the value for the current row.

- **`nth_value`***(expr, n)* – the value for the *n*-th row within the window frame; *n* must be an integer

```
nth_value(sold, 2) OVER (PARTITION BY city
  ORDER BY month RANGE BETWEEN UNBOUNDED
    PRECEDING AND UNBOUNDED FOLLOWING)
```

city	month	sold	nth_value
Paris	1	500	300
Paris	2	300	300
Paris	3	400	300
Rome	2	200	300
Rome	3	300	300
Rome	4	500	300
Rome	5	300	300
London	1	100	NULL

ORDER BY and Window Frame: `first_value()`, `last_value()`, and `nth_value()` do not require an ORDER BY. They accept window frame definition (`ROWS`, `RANGE`, `GROUPS`).