Section 6:
Window Functions

Fanavaran Anisa
Iran Linux House
Linux & Open Source Training Center

Updates – Is it Correct?

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%
  - Write two **update** statements:

```
update instructor
        set salary = salary * 1.05
        where salary <= 100000;

update instructor
        set salary = salary * 1.03
        where salary > 100000;
```

  - Can we use the transactions ?

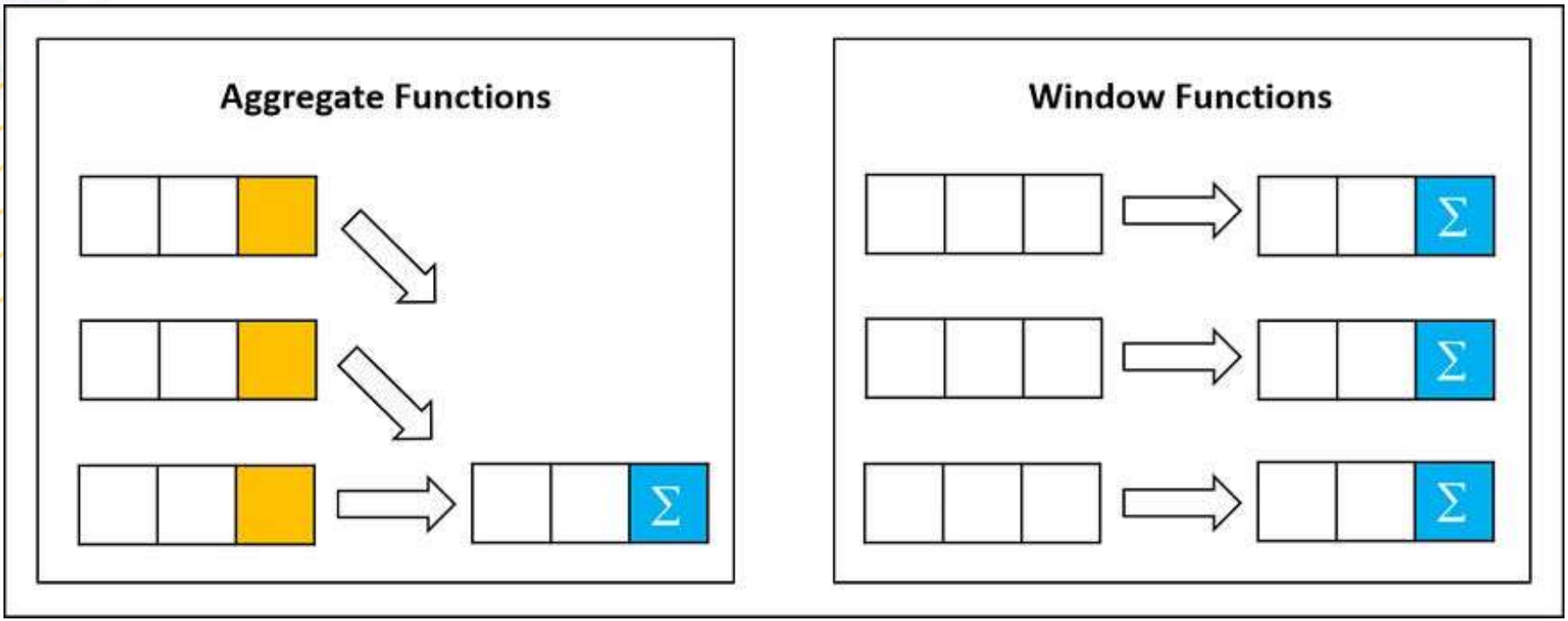
Case Statement for Conditional Updates

- Same query as before but with case statement

```
update instructor
    set salary = case
                    when salary <= 100000 then salary * 1.05
                    else salary * 1.03
                 end
```

Window Functions

A *window function* performs a calculation across a set of table rows that are somehow related to the current row.



Window Functions – Running Total

Running total mileage visual		
Day	**Miles Driving**	**Running Total**
Jan. 1	60	
Jan. 2	80	
Jan. 3	10	
Jan. 4	150	

Window Functions – 3 Day Average

Running Average Example		
Day	**Daily Revenue**	**3 Day Average**
1	39	
2	528	
3	39	
4	86	
5	86	
6	351	

Two-day average pedaling time of each cyclist

id	date	time
1	2019-07-05	22
1	2019-04-15	26
2	2019-02-06	28
1	2019-01-02	30
2	2019-08-30	20
2	2019-03-09	22

1 PARTITION BY id

id	date	time
1	2019-07-05	22
1	2019-04-15	26
1	2019-01-02	30

id	date	time
2	2019-02-06	28
2	2019-08-30	20
2	2019-03-09	22

2 ORDER BY date

id	date	time
1	2019-01-02	30
1	2019-04-15	26
1	2019-07-05	22

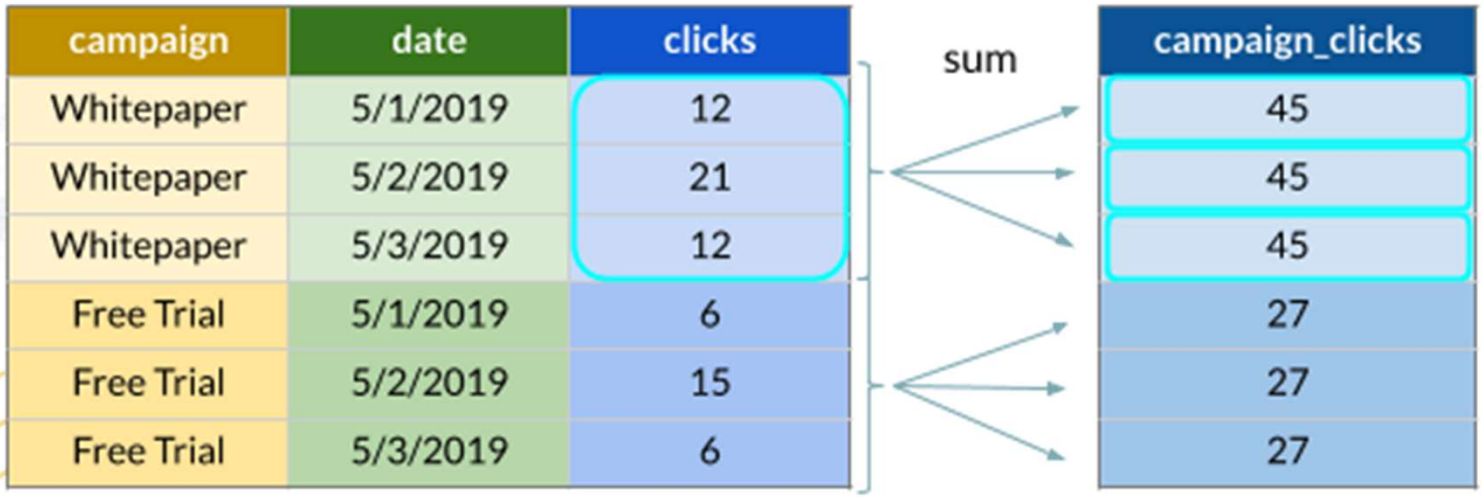
id	date	time
2	2019-02-06	28
2	2019-03-09	22
2	2019-08-30	20

3 AVG(time)

ROWS BETWEEN 1 PRECEDING AND CURRENT ROW

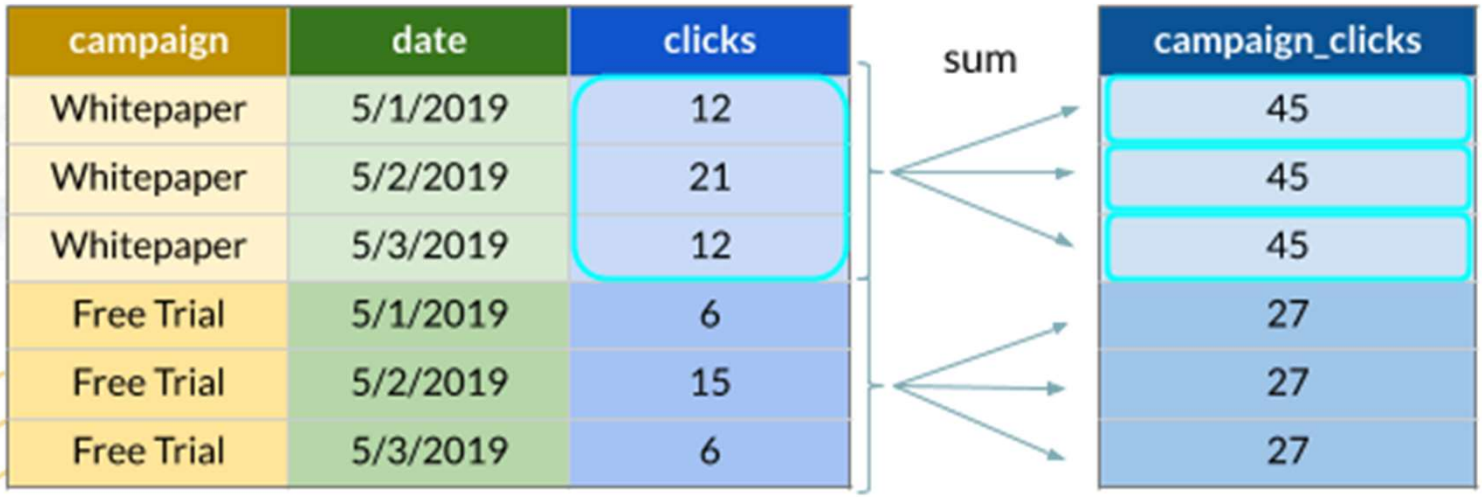
id	date	time	avg_time
1	2019-01-02	30	30
1	2019-04-15	26	28
1	2019-07-05	22	24
2	2019-02-06	28	28
2	2019-03-09	22	25
2	2019-08-30	20	21

Window Functions – The Classic Way




```
SELECT a.clicks
FROM campaigns c
 INNER JOIN (
   SELECT campaign, SUM(clicks) AS clicks FROM campaigns GROUP BY campaign
 ) a
 ON c.campaign = a.campaign
```

Window Functions




```
SELECT
 SUM(clicks) OVER (PARTITION BY campaign) AS campaign_clicks
FROM campaigns
```

Window Functions - Syntax

```
SELECT <column_1>, <column_2>,
  <window_function> OVER (
    PARTITION BY <...>
    ORDER BY <...>
    <window_frame>) <window_column_alias>
FROM <table_name>;
```

```
SELECT city, month,
  sum(sold) OVER (
    PARTITION BY city
    ORDER BY month
    RANGE UNBOUNDED PRECEDING) total
FROM sales;
```

Window Function - Over

```
SELECT city, month, sum(sold) OVER () AS
sum FROM sales;
```

Calculates Sum of Sold column (for all record – we have no where) and show it next to the city and Month column for all record in Sales table.

Window Function – Partition By

```
SELECT city, month, sum(sold) OVER
(PARTITION BY city) AS sum FROM sales;
```

month	city	sold
1	Rome	200
2	Paris	500
1	London	100
1	Paris	300
2	Rome	300
2	London	400
3	Rome	400

PARTITION BY city

month	city	sold	sum
1	Paris	300	800
2	Paris	500	800
1	Rome	200	900
2	Rome	300	900
3	Rome	400	900
1	London	100	500
2	London	400	500

Window Function – Order By

```
SELECT city, month, sum(sold) OVER
(PARTITION BY city ORDER BY month) sum
FROM sales;
```

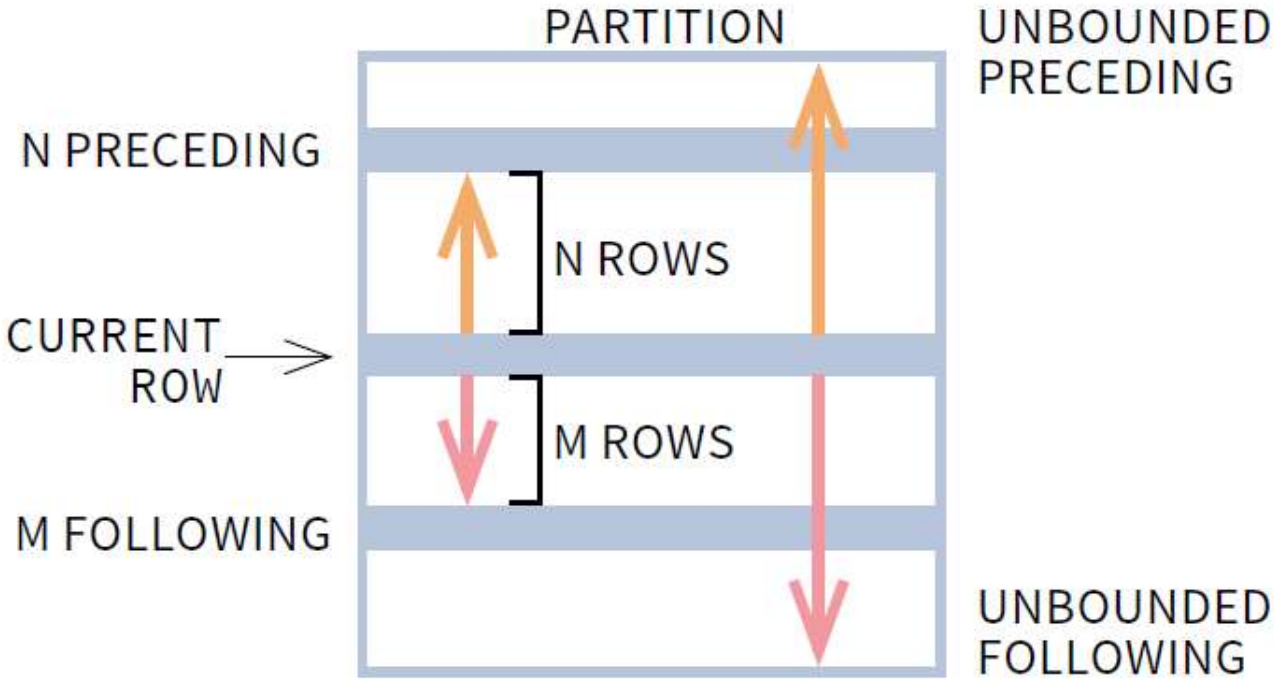
sold	city	month
200	Rome	1
500	Paris	2
100	London	1
300	Paris	1
300	Rome	2
400	London	2
400	Rome	3

PARTITION BY city ORDER BY month

sold	city	month
300	Paris	1
500	Paris	2
200	Rome	1
300	Rome	2
400	Rome	3
100	London	1
400	London	2

Window Function – Window Frame

A **window frame** is a set of rows that are somehow related to the current row. The window frame is evaluated separately within each partition.



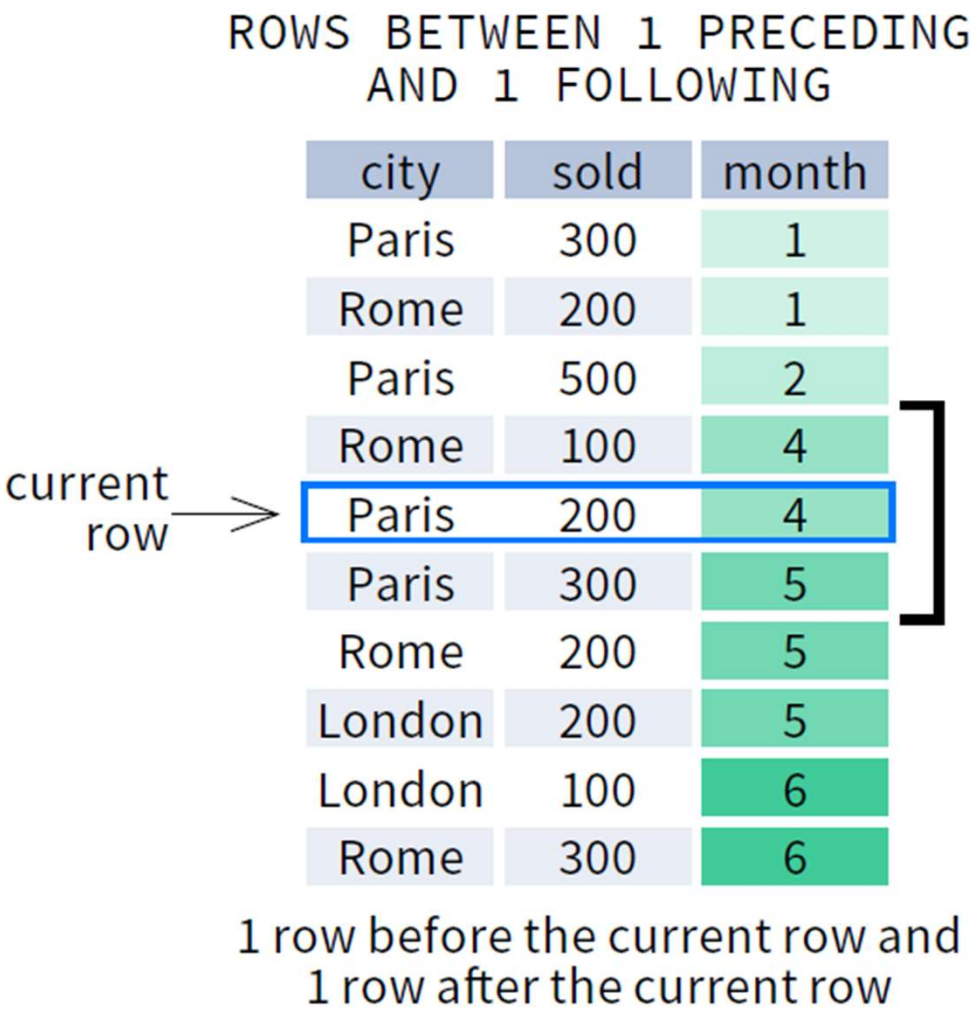
Window Function – Window Frame

ROWS | RANGE | GROUPS BETWEEN lower_bound AND upper_bound

The bounds can be any of the five options:

- UNBOUNDED PRECEDING
- n PRECEDING
- CURRENT ROW
- n FOLLOWING
- UNBOUNDED FOLLOWING

Window Function – Window Frame



Window Function – Window Frame

```
RANGE BETWEEN 1 PRECEDING
AND 1 FOLLOWING
```

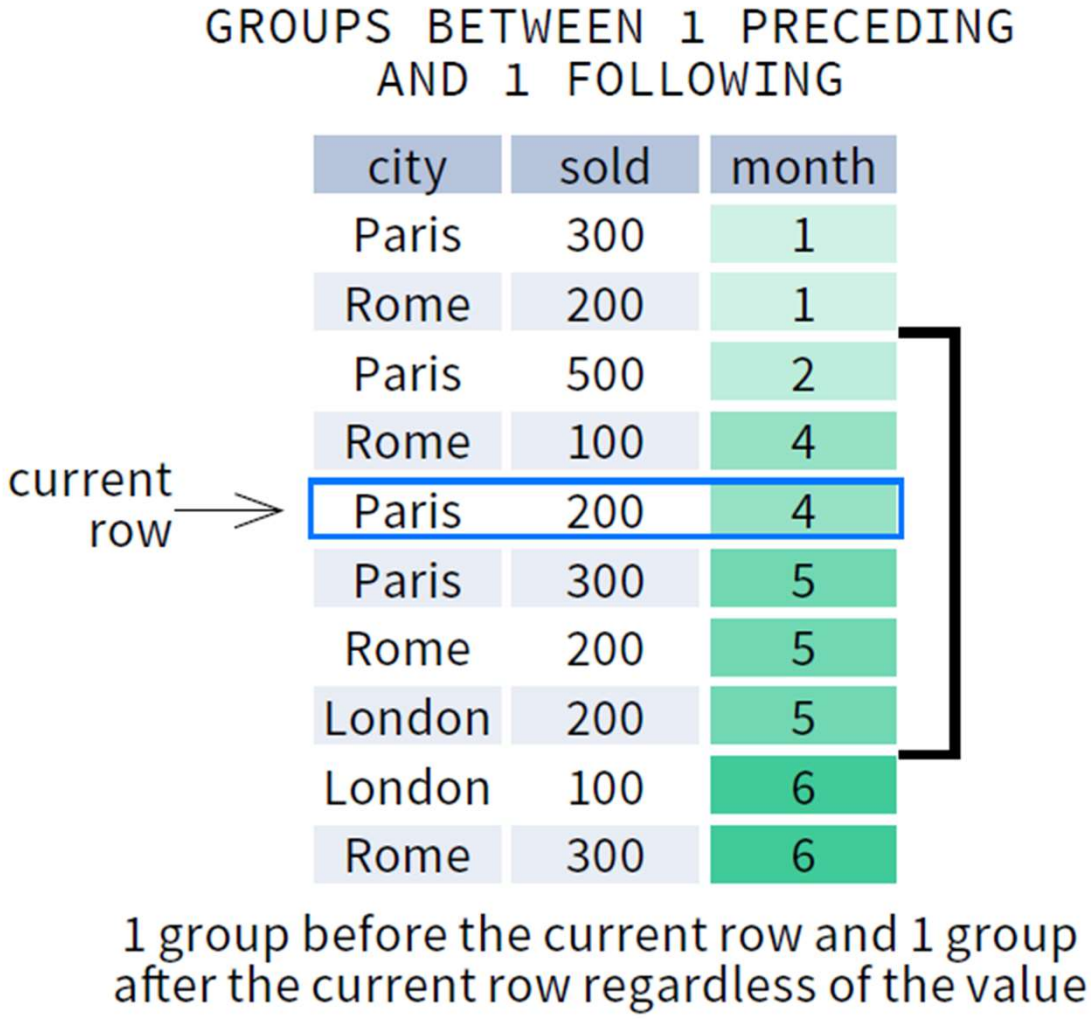
city	sold	month
Paris	300	1
Rome	200	1
Paris	500	2
Rome	100	4
Paris	200	4
Paris	300	5
Rome	200	5
London	200	5
London	100	6
Rome	300	6

current row →

cu

values in the range between 3 and 5
ORDER BY must contain a single expression

Window Function – Window Frame



Window Frame Abbrevations

Abbreviation	Meaning
`UNBOUNDED PRECEDING`	`BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`
`n PRECEDING`	`BETWEEN n PRECEDING AND CURRENT ROW`
`CURRENT ROW`	`BETWEEN CURRENT ROW AND CURRENT ROW`
`n FOLLOWING`	`BETWEEN AND CURRENT ROW AND n FOLLOWING`
`UNBOUNDED FOLLOWING`	`BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING`

Default Window Frame

- If ORDER BY is specified, then the frame is RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.
- Without ORDER BY, the frame specification is ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

List of Window Functions

- 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()`
  - `count()`
  - `max()`
  - `min()`
  - `sum()`

Ranking Functions

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

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

`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

`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

Analytic Functions

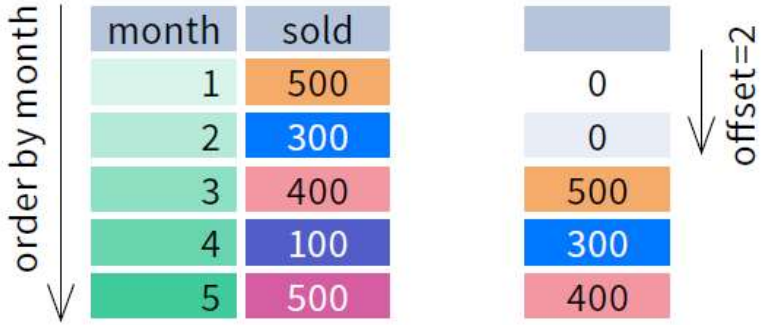
- lead(expr, offset, default) - the value for the row offset rows after the current; offset and default are optional; default values: offset = 1, default = NULL
- lag(expr, offset, default) - the value for the row offset rows before the current; offset and default are optional; default values: offset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

Analytic Functions

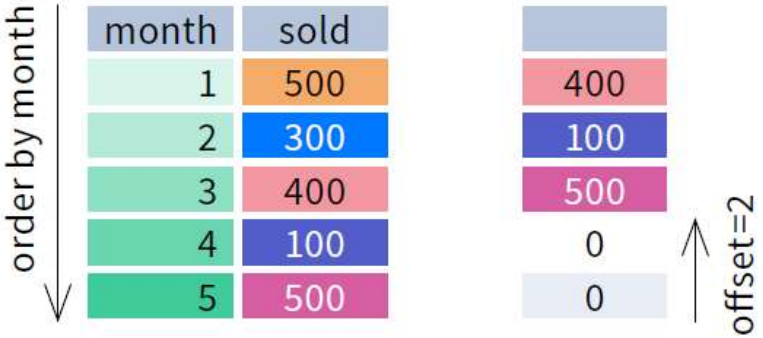
`lag(sold) OVER(ORDER BY month)`

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

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

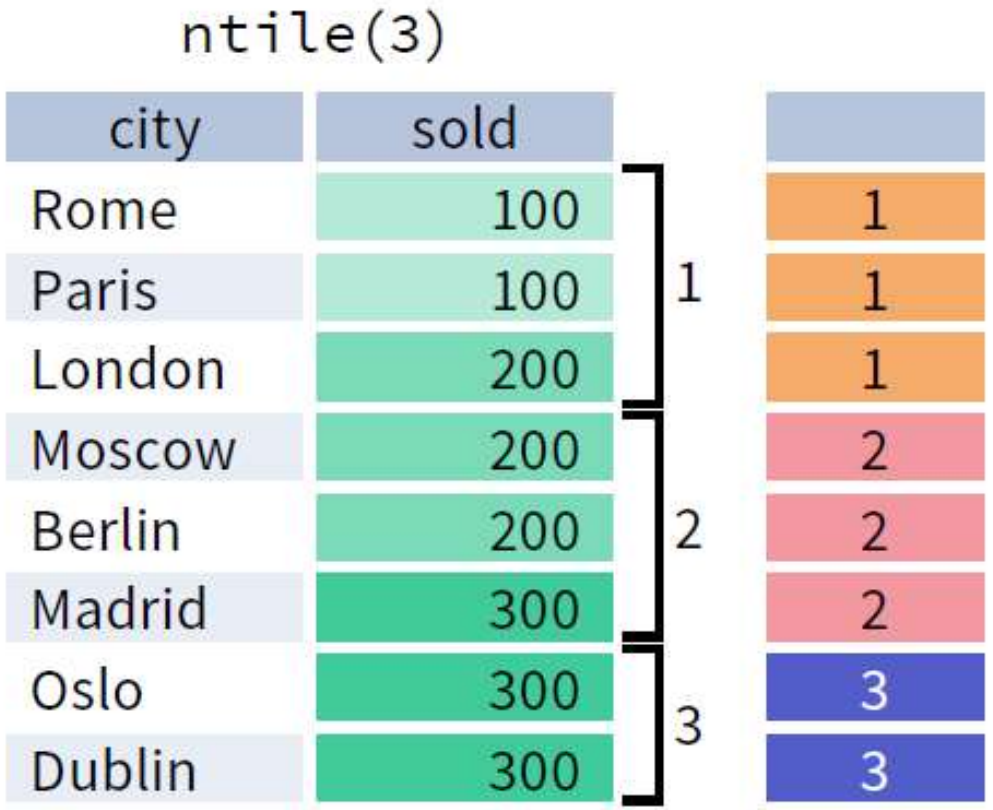


`lead(sold, 2, 0) OVER(ORDER BY month)`



Analytic Functions

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



Analytic Functions

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

Analytic Functions

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

Analytic Functions

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.

Analytic Functions

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)
```

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