Assigning row numbers

WINDOW FUNCTIONS IN SNOWFLAKE

Jake Roach
Field Data Engineer

datacamp

Window functions

> Used to perform a calculation across a **"window" of rows** and return a value **for each row**

Ranking

- `RANK`, `DENSE_RANK`
- `FIRST_VALUE`, `LAST_VALUE`, `NTH_VALUE`
- `LAG`, `LEAD`

Aggregation

- `COUNT`, `SUM`, `AVG`

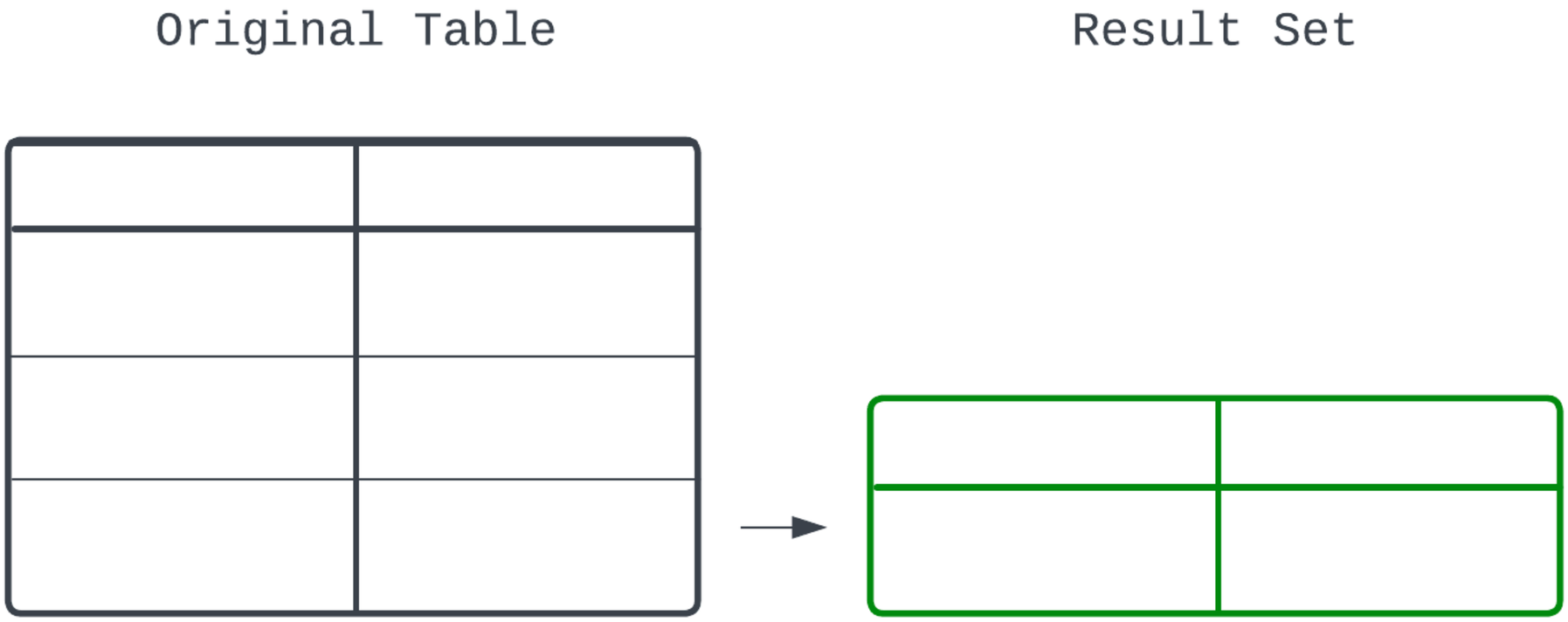
Window functions

Window functions help us answer new types of questions!

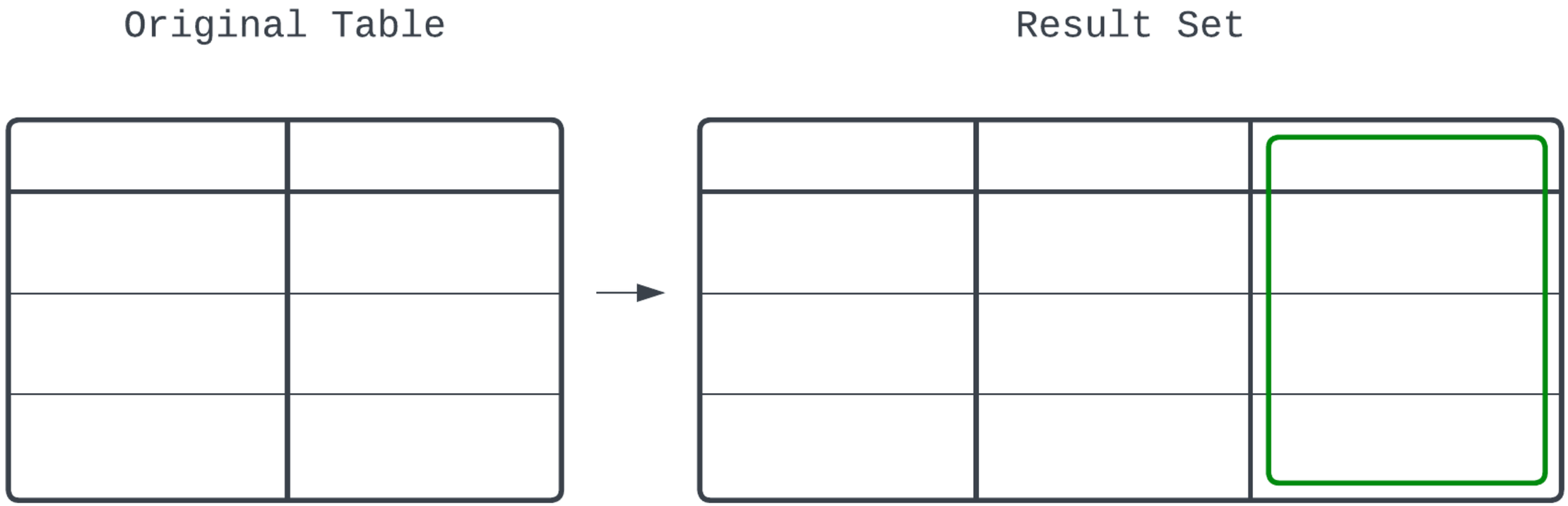
- What is the **third most popular concert** at each venue?
- How did sales for an item vary **day-to-day**?
- Is my **rolling average** of viewership trending up or down?

Traditional functions

Traditional aggregation function must have a `GROUP BY` clause.



Window functions



Do not require a `GROUP BY` !

Assigning row numbers

```
SELECT
    <other-fields>,

    ROW_NUMBER() OVER(
        ORDER BY <field>
    ) AS <alias>

FROM SCHEMA.table

...;
```

The `ROW_NUMBER()` function assigns a row number to each record in a result set

- `OVER()` defines the window
- Alias the resulting column
- No `GROUP BY` !

Concert attendance

```
SELECT
    customer_id,
    event_name,

    -- Assign a row number to each
    -- record in the result set

    ROW_NUMBER() OVER(
        ORDER BY time_spent_minutes
    ) AS row_num

FROM concerts.attendance;
```

customer_id	event_name	row_num
1931292	Music Fest	1
5462384	Elton John	2
7732891	Coachella	3
3124888	Porch Fest	4

Remember the syntax:

`ROW_NUMBER() OVER( ORDER BY ...)`!

Let's practice!

WINDOW FUNCTIONS IN SNOWFLAKE

Ranking window functions

WINDOW FUNCTIONS IN SNOWFLAKE

Jake Roach
Field Data Engineer

RANK()

```
SELECT
    user_id,
    event_name,
    distance_traveled,

    -- Find the closest attendees
    RANK() OVER(
        ORDER BY km_traveled
    ) AS closest_concert_goer

FROM CONCERTS.attendance;
```

`RANK()` is used to assign a "ranking" to records based on some field

- Similar to `ROW_NUMBER()`
- Handles ties
- Include an `ORDER BY`

RANK()

user_id	event_name	km_traveled	closest_attendees
user_81	Lunar Drift	0.5	1
user_02	Lunar Drift	1.1	2
user_33	Crimson Arc	8.6	3
user_33	Neon Prophet	8.6	3
user_15	VibeStorm	17	5
user_94	The Dusk Owls	41	6
user_47	Lunar Drift	61	7
user_56	Crimson Arc	116	8

RANK() with DESC

```
SELECT
    user_id,
    event_name,
    km_traveled,

    RANK() OVER(
        ORDER BY km_traveled DESC  -- Add DESC to ORDER BY
    ) AS furthest_concert_goer

FROM CONCERTS.attendance;
```

RANK() with DESC

user_id	event_name	km_traveled	closest_attendees -> furthest_attendees
user_81	Lunar Drift	0.5	1 -> 8
user_02	Lunar Drift	1.1	2 -> 7
user_33	Crimson Arc	8.6	3 -> 6
user_33	Neon Prophet	8.6	3 -> 6
user_15	VibeStorm	17	5 -> 4
user_94	The Dusk Owls	41	6 -> 3
user_47	Lunar Drift	61	7 -> 2
user_56	Crimson Arc	116	8 -> 1

FIRST_VALUE and LAST_VALUE

`FIRST_VALUE` and `LAST_VALUE` find the first-ranked and last-ranked value for a window, respectively

- Comparing records in a column to that column's first/last value
- Takes the name of the field
- `ORDER BY` a field, these don't have to match

```
SELECT
    <other-fields>,

    -- FIRST_VALUE and LAST_VALUE
    -- both take a field

    [FIRST/LAST]_VALUE(<field>) OVER(
        ORDER BY <field>
    ) AS <alias>

FROM SCHEMA.table
...;
```

The Good and the Bad

```
SELECT
    user_id, event_name, satisfaction_score,

    FIRST_VALUE(satisfaction_score) OVER(
        ORDER BY satisfaction_score DESC
    ) AS most_satisfied,

    LAST_VALUE(satisfaction_score) OVER(
        ORDER BY satisfaction_score DESC
    ) AS least_satisfied

FROM CONCERTS.attendance;
```

The Good and the Bad

user_id	event_name	satisfaction_score	most_satisfied	least_satisfied
user_26	Pulse Theory	71	98	4
user_71	Echo Valley	9	98	4
user_26	Echo Valley	82	98	4
user_57	Nova Sway	4	98	4
user_39	Nova Sway	22	98	4
user_38	Bass Ritual	76	98	4
user_92	Pulse Theory	98	98	4
user_44	Nova Sway	62	98	4

Let's practice!

WINDOW FUNCTIONS IN SNOWFLAKE

Partitioning data in a window function

WINDOW FUNCTIONS IN SNOWFLAKE

Jake Roach
Field Data Engineer

Ranking data

Here, we're ranking all records in the result set!

user_id	event_name	km_traveled	closest_attendees
user_81	Lunar Drift	0.5	1
user_02	Lunar Drift	1.1	2
user_33	Crimson Arc	8.6	3
user_33	Neon Prophet	8.6	3
user_15	VibeStorm	17	5
user_94	The Dusk Owls	41	6
user_47	Lunar Drift	61	7
user_56	Crimson Arc	116	8

Ranking data with partitions

Now, we want to rank data for each specified window.

user_id	event_name	km_traveled	closest_attendees
user_81	Lunar Drift	0.5	1
user_02	Lunar Drift	1.1	2
user_47	Lunar Drift	61	3
user_33	Crimson Arc	8.6	1
user_56	Crimson Arc	116	2
user_33	Neon Prophet	8.6	1
user_15	VibeStorm	17	1
user_94	The Dusk Owls	41	1

PARTITION BY

```
SELECT
    user_id,
    event_name,
    distance_traveled,

    RANK() OVER(
        -- Create window by event_name
        PARTITION BY event_name
        ORDER BY km_traveled
    ) AS closest_concert_goer

FROM CONCERTS.attendance;
```

`PARTITION BY` helps us create windows of records to apply functions to

- `PARTITION BY` goes before `ORDER BY` in `OVER(...)`
- Similar to `GROUP BY`, but does not "collapse" records

Ranking data with partitions

```
SELECT
    level,
    price,

    RANK() OVER(
        PARTITION BY level
        ORDER BY price DESC
    ) AS price_rank

FROM CONCERTS.attendance;
```

- `PARTITION BY` creates windows

```
 level | price | price_rank
-------|-------|-----------
   100 |   765 |          1
   100 |   617 |          2
   100 |   490 |          3
   100 |   490 |          3

                ...

   200 |   212 |          1
   200 |   207 |          2

                ...
```

Generating summary metrics with FIRST_VALUE

```
FIRST_VALUE(<1>) OVER(
    PARTITION BY <2>
    ORDER BY <3>
) AS <alias>
```

`FIRST_VALUE` will help to find the first value in a window

`<1>` : which column in record to return

`<2>` : field to partition data by

`<3>` : field to determine first record

Generating summary metrics with AVG

```
AVG(<1>) OVER(
    PARTITION BY <2>
    -- No need to ORDER BY!
) AS <alias>
```

`AVG` will find the mean value of a field for each window

`<1>` : column to take the average of

`<2>` : field to partition data by

... no need for `ORDER BY` !

Customer satisfaction

```
SELECT
    user_id, event_name, satisfaction_score,

    FIRST_VALUE(satisfaction_score) OVER(
        PARTITION BY event_name   -- Satisfaction score for the closest concert-goer
        ORDER BY km_traveled
    ) AS first_score,

    -- Find the average satisfcation score for a "window" of records
    AVG(satisfaction_score) OVER(
        PARTITION BY event_name
    ) AS average_score

FROM CONCERTS.attendance;
```

Customer satisfaction

user_id	event_name	satisfaction_score	first_score	average_score
user_26	Pulse Theory	71	98	84.5
user_92	Pulse Theory	98	98	84.5
...				
user_57	Nova Sway	4	22	29.3
user_39	Nova Sway	22	22	29.3
user_44	Nova Sway	62	22	29.3
...				

Let's practice!

WINDOW FUNCTIONS IN SNOWFLAKE