Window Functions

Window Functions

Three common uses:

- Comparing an aggregate value with the original values
  - Eg. comparing a store’s sales numbers to the average sales numbers across all stores/all stores in a region

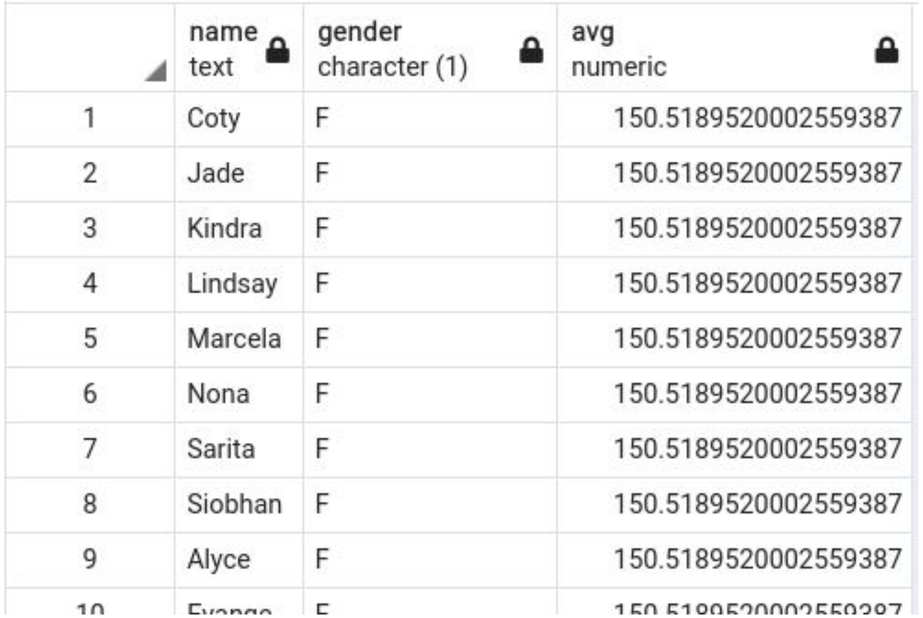
- Comparing observations within a group
  - Eg. Ranking stores within each region
  - Eg. comparing this year’s sales to last year’s sales

- Computing rolling or running averages
  - Eg. smoothing a time series by taking a rolling 7-day average

Window Functions

The big difference between an aggregate function and a window function is that an aggregate function will combine multiple rows into one, whereas a window function keeps all rows separate.

```
SELECT
      name, gender,
      AVG(num_registered) OVER(PARTITION BY gender)
FROM names;
```



	name text	gender character (1)	avg numeric
1	Coty	F	150.5189520002559387
2	Jade	F	150.5189520002559387
3	Kindra	F	150.5189520002559387
4	Lindsay	F	150.5189520002559387
5	Marcela	F	150.5189520002559387
6	Nona	F	150.5189520002559387
7	Sarita	F	150.5189520002559387
8	Siobhan	F	150.5189520002559387
9	Alyce	F	150.5189520002559387
10	Evange	F	150.5189520002559387

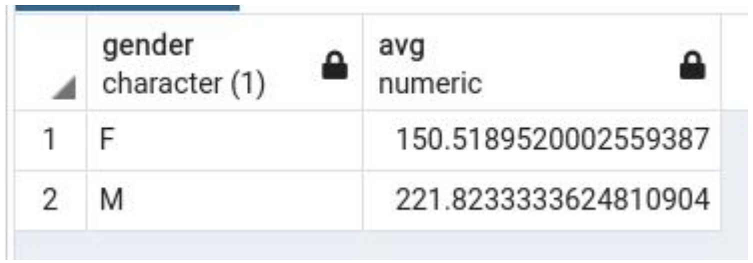
“PARTITION BY gender” says to calculate the average within the gender groups, but it does not collapse these groups into a single row.

Notice that we can SELECT name even though we are not PARTITIONing by name.

Window Functions

The big difference between an aggregate function and a window function is that an aggregate function will combine multiple rows into one, whereas a window function keeps all rows separate.

```
SELECT
	gender,
	AVG(num_registered)
FROM names
GROUP BY gender;
```



	gender character (1)	avg numeric
1	F	150.5189520002559387
2	M	221.8233333624810904

"GROUP BY gender" collapses all rows with the same gender down into a single row.

Window Functions

Useful keywords for comparing observations within groups:

- LAG/LEAD: compare an observations with the previous/next one within its group
- RANK: assign a rank within a group
    - Should be accompanied by an ORDER BY inside the accompanying OVER()
    - Can also use a PARTITION BY to generate rankings within groups

See here for a full list of window functions usable in PostgreSQL.

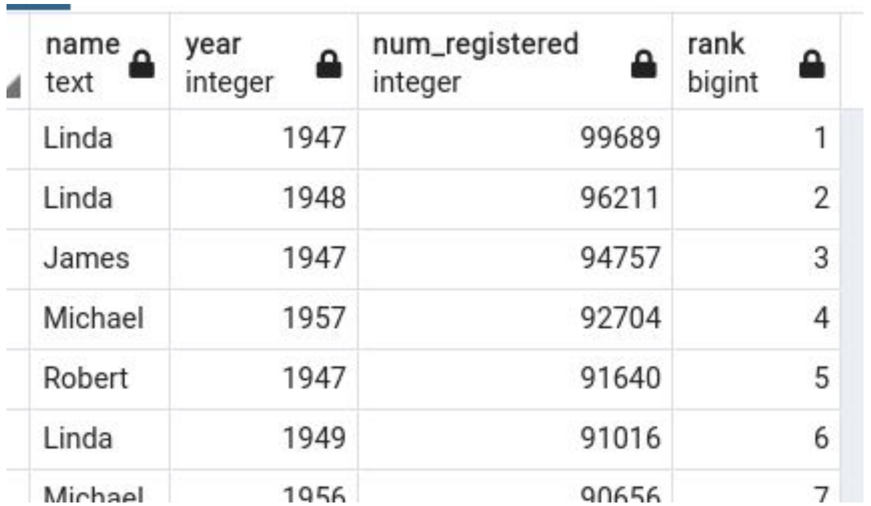
Window Functions

Useful keywords for comparing observations within groups:

- LAG/LEAD: compare an observations with the previous/next one within its group
- RANK: assign a rank within a group
    - Should be accompanied by an ORDER BY inside the accompanying OVER()
    - Can also use a PARTITION BY to generate rankings within groups

See here for a full list of window functions usable in PostgreSQL.

```
SELECT
    name, year, num_registered,
    RANK() OVER(ORDER BY num_registered DESC)
FROM names;
```



name text	year integer	num_registered integer	rank bigint
Linda	1947	99689	1
Linda	1948	96211	2
James	1947	94757	3
Michael	1957	92704	4
Robert	1947	91640	5
Linda	1949	91016	6
Michael	1956	90656	7

With no PARTITION BY, this ranks by num_registered across *all* rows.

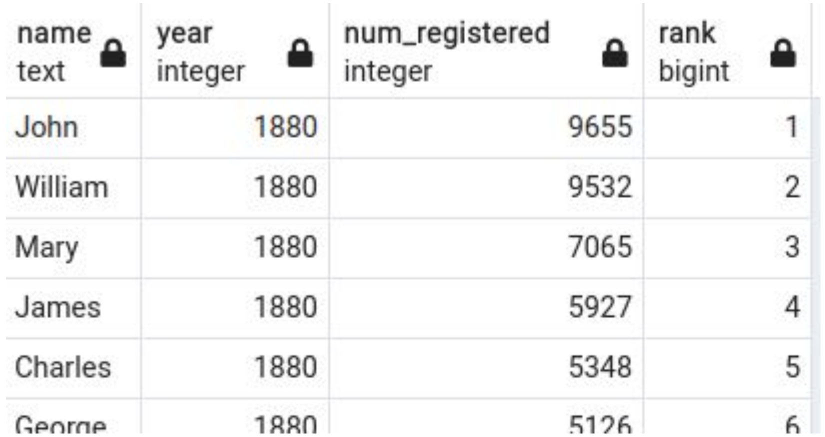
Window Functions

Useful keywords for comparing observations within groups:

- LAG/LEAD: compare an observations with the previous/next one within its group
- RANK: assign a rank within a group
  - Should be accompanied by an ORDER BY inside the accompanying OVER()
  - Can also use a PARTITION BY to generate rankings within groups

See here for a full list of window functions usable in PostgreSQL.

```
SELECT
    name, year, num_registered,
    RANK() OVER(
        PARTITION BY year
        ORDER BY num_registered DESC)
FROM names;
```



name text	year integer	num_registered integer	rank bigint
John	1880	9655	1
William	1880	9532	2
Mary	1880	7065	3
James	1880	5927	4
Charles	1880	5348	5
George	1880	5126	6

PARTITION BY year will generate separate rankings by num_registered for each year.

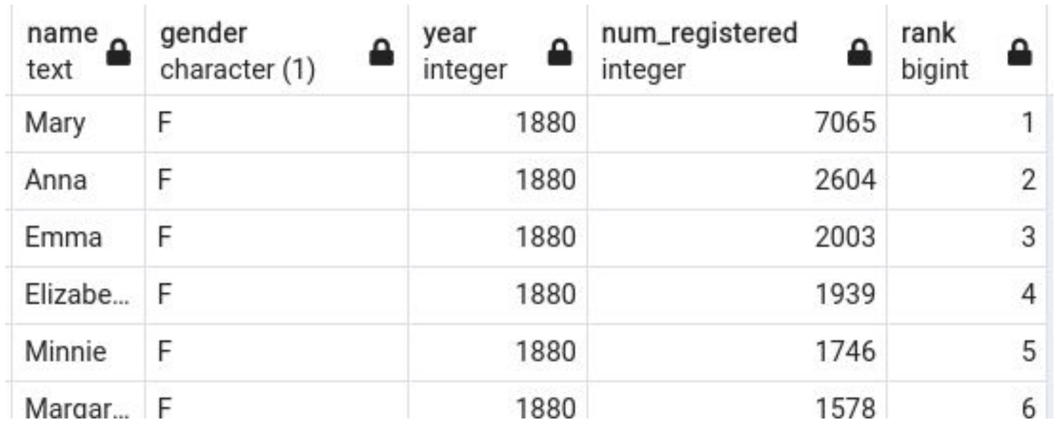
Window Functions

Useful keywords for comparing observations within groups:

- LAG/LEAD: compare an observations with the previous/next one within its group
- RANK: assign a rank within a group
    - Should be accompanied by an ORDER BY inside the accompanying OVER()
    - Can also use a PARTITION BY to generate rankings within groups

See here for a full list of window functions usable in PostgreSQL.

```
SELECT
    name, gender, year, num_registered,
    RANK() OVER(
        PARTITION BY year, gender
        ORDER BY num_registered DESC)
FROM names;
```



name text	gender character (1)	year integer	num_registered integer	rank bigint
Mary	F	1880	7065	1
Anna	F	1880	2604	2
Emma	F	1880	2003	3
Elizabe...	F	1880	1939	4
Minnie	F	1880	1746	5
Margar...	F	1880	1578	6

You can even PARTITION BY multiple columns, just like
you can GROUP BY multiple columns.

Window Functions

For rolling averages, see this excellent site for some great illustrations:
https://dataschool.com/how-to-teach-people-sql/how-window-functions-work/

Set a window size using ROWS and BETWEEN.

- Can be a fixed window size
  - Eg. ROWS BETWEEN 2 PRECEDING AND CURRENT ROW

- Or an expanding window size
  - Eg. ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW