Window Functions

By now you have noticed that when you return an aggregated value, every other value that is selected and ***not*** aggregated must appear in the GROUP BY clause.

```
SELECT year, country_id, SUM(bronze)
FROM winter_games
GROUP BY country_id;
```

Data Output | Explain | Messages | Notifications

```
ERROR:  column "winter_games.year" must appear in the GROUP BY clause or be used in an aggregate function
LINE 1: SELECT year, country_id, SUM(bronze)
               ^
SQL state: 42803
Character: 8
```

- Once the list of columns to GROUP BY becomes large, consider using a window function.

- Window functions will generate **result sets** in SQL

- You can generate the window and then perform aggregate calculations on them, including
  - running totals
  - rankings
  - moving averages

- Window functions are run after everything ***except*** the final ORDER BY clause

The OVER() function creates a **window** to apply an aggregate calculation to. Here the OVER() function has no arguments passed to it and will return the average *over all rows* in the table:

```
SELECT distinct county,
       avg(new_jobs) OVER() as average_new_jobs
FROM ecd;
```

	county text	average_new_jobs numeric
1	Claiborne	152.3558758314855876
2	Sequatchie	152.3558758314855876
3	Clay	152.3558758314855876
4	Cocke	152.3558758314855876
5	Pickett	152.3558758314855876
6	Dyer	152.3558758314855876
7	Houston	152.3558758314855876

You can create a partition inside of the OVER() function with the PARTITON BY command.

Let's say you want to create a partition for every county and year combination in the unemployment table in order to calculate the average unemployment in each county for each year.

Here is what one **window** partitioned by county and year would look like:

```
SELECT *
FROM unemployment
WHERE county = 'Anderson' and year = 2011;
```

	county text	period integer	period_name text	year integer	value numeric
1	Anderson	12	December	2011	7.6
2	Anderson	11	November	2011	7.6
3	Anderson	10	October	2011	8.0
4	Anderson	9	September	2011	8.6
5	Anderson	8	August	2011	8.9
6	Anderson	7	July	2011	8.9
7	Anderson	6	June	2011	9.5
8	Anderson	5	May	2011	8.7
9	Anderson	4	April	2011	8.8
10	Anderson	3	March	2011	9.3
11	Anderson	2	February	2011	9.5
12	Anderson	1	January	2011	9.7

Here we partition the unemployment table in the ecd database by county and year in order to find the average annual unemployment for each county in each year in the dataset.

```
SELECT distinct county,
                year,
                AVG(value) OVER(PARTITION BY county, year) as avg_annual_unemployment
FROM unemployment
ORDER BY year, county;
```

	county text	year integer	avg_annual_unemployment numeric
1	Anderson	2011	8.7583333333333333
2	Bedford	2011	11.2416666666666667
3	Benton	2011	11.3750000000000000
4	Bledsoe	2011	11.2916666666666667
5	Blount	2011	8.1750000000000000
6	Bradley	2011	8.9250000000000000
7	Campbell	2011	11.7000000000000000

Example 2: Using the unemployment table in the ecd database, write a window function to find the rolling average 12-month unemployment (look at the current month and the 11 preceding months) for each county in the dataset. Note that the period column refers to the numeric value for month (eg, March would be 3).

```
SELECT distinct county,
                value,
                year,
                period,
                AVG(value) OVER(PARTITION BY county ORDER BY year, period
                                ROWS BETWEEN 11 PRECEDING and CURRENT ROW)
                                as avg_12month_unemployment
FROM unemployment
ORDER BY county, year, period;
```

	county text	value numeric	year integer	period integer	avg_12month_unemployment numeric
1	Anderson	9.7	2011	1	9.7000000000000000
2	Anderson	9.5	2011	2	9.6000000000000000
3	Anderson	9.3	2011	3	9.5000000000000000
4	Anderson	8.8	2011	4	9.3250000000000000
5	Anderson	8.7	2011	5	9.2000000000000000
6	Anderson	9.5	2011	6	9.2500000000000000
7	Anderson	8.9	2011	7	9.2000000000000000

Example 3: Using the athletes table in the olympics database, use RANK() and a window function to order all athletes by BMI where a rank of 1 denotes the highest BMI.
- the calculation for BMI (given weight in kilograms and height in centimeters) is: **weight / (height/100)^2**

```
SELECT name,
       age,
       weight,
       height,
       (weight / (POWER((CAST(height as float)/100), 2)))  as bmi,
       RANK() OVER(ORDER BY (weight / (POWER((CAST(height as float)/100), 2))) DESC) as bmi_rank
FROM athletes
WHERE weight IS NOT NULL and height IS NOT NULL;
```

	name character varying (255)	age integer	weight integer	height integer	bmi double precision	bmi_rank bigint
1	Michelle Denee Carter	30	136	176	43.904958677685954	1
2	Ion Emilianov	39	165	202	40.43721203803548	2
3	Raven Saunders	20	109	165	40.03673094582186	3
4	Asmir Kolainac	31	140	187	40.0354599788384	4
5	Joseph Mathias "Joe" Kovacs	27	134	183	40.01313864253934	5
6	Darrell Hill	22	145	191	39.74671746936762	6
7	Tsanko Rosenov Arnaudov	24	155	198	39.53678196102439	7

Exercises

1. For each Tennessee county in the population table in the ecd database:
    a. create a partition to find the maximum population(max_pop) values in the table.
    b. Also find the minimum population (min_pop) for each county.

2. Use a window function in the olympics database to find the number of times each country had an athlete compete in the winter games. Be sure to return the country name along with the count of times that country had an athlete competing.