Lecture 11-1: Statistical Functions in SQL

DATA 503: Fundamentals of Data Engineering

Lucas P. Cordova, Ph.D.

2026-03-16

This lecture covers statistical functions built into PostgreSQL. We explore correlation, linear regression, variance, standard deviation, ranking with window functions, rate calculations, and rolling averages. Based on Chapter 11 of Practical SQL, 2nd Edition.

Table of contents

1 Part 1: Setting Up

We need data before we can do statistics.

1.1 Loading the Chapter Database

1.1.1 Step 0: Get the data

Download the data from the Canvas assignment and save it to your computer. Copy the files to a safe/location on your machine; i.e. `/tmp` or `C:\Users\Public`

1.1.2 Step 1: Create a Fresh Database either using the command or in your GUI client

```
createdb stats_analysis
```

Connect to it: `\c stats_analysis` (or use your GUI client).

1.1.3 Step 2: Create and Load the Census Table

The American Community Survey (ACS) gives us county-level data on education, income, and commuting for 3,142 U.S. counties:

```
CREATE TABLE acs_2014_2018_stats (
    geoid text CONSTRAINT geoid_key PRIMARY KEY,
    county text NOT NULL,
    st text NOT NULL,
    pct_travel_60_min numeric(5,2),
    pct_bachelors_higher numeric(5,2),
    pct_masters_higher numeric(5,2),
    median_hh_income integer,
    CHECK (pct_masters_higher <= pct_bachelors_higher)
);
```

Notice the `CHECK` constraint. Why would we enforce that masters must be less than or equal to bachelors?

. . .

Because everyone with a master's degree *also* has a bachelor's degree. If `pct_masters` > `pct_bachelors`, something is deeply wrong with our data. Constraints catch problems that eyeballs miss.

1.1.4 Step 3: Import the CSV

```sql
copy acs_2014_2018_stats FROM '/path/to/acs_2014_2018_stats.csv' WITH (FORMAT CSV, HEADER);
```

Update the path to match your machine. Verify:

```sql
SELECT count(*) FROM acs_2014_2018_stats;
-- Should be 3,142
```

1.1.5 Quick Look at the Data

```sql
SELECT * FROM acs_2014_2018_stats
ORDER BY median_hh_income DESC
LIMIT 5;
```

Run this now. Which counties have the highest median household income? Any surprises?

2 Part 2: Measuring Correlation

Does education affect income? Let's ask the data.

2.1 The Pearson Correlation Coefficient

2.1.1 What Is Correlation?

The **Pearson correlation coefficient** (r) measures the strength and direction of a *linear* relationship between two variables. It answers the question: "When one variable increases, does the other tend to increase (or decrease) in a consistent, straight-line pattern?"

Formally, it is defined as:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \cdot \sum(y_i - \bar{y})^2}}$$

The numerator captures how X and Y move *together* (covariance). The denominator normalizes by how much each variable moves on its own (their standard deviations). The result is always between -1 and +1.

2.1.2 Interpreting the Correlation Coefficient

r value	Interpretation
+1.0	Perfect positive: as X goes up, Y goes up proportionally
+0.7 to +0.9	Strong positive
+0.4 to +0.69	Moderate positive
+0.1 to +0.39	Weak positive
0	No linear relationship
Negative values	Same scale, opposite direction

Key word: **linear**. Two variables can have a strong *curved* relationship and still show r near 0. Correlation only detects straight-line patterns. A parabolic relationship, for example, would produce r close to zero even though the variables are strongly related.

2.1.3 corr(Y, X) in PostgreSQL

`corr(Y, X)` is an **aggregate function** that computes the Pearson r across all rows. It takes two arguments: the dependent variable (Y) first, then the independent variable (X).

```sql
SELECT corr(median_hh_income, pct_bachelors_higher)
    AS bachelors_income_r
FROM acs_2014_2018_stats;
```

Run this. What do you get?

. . .

About **0.70**. That's a fairly strong positive correlation. Counties with higher percentages of bachelor's degree holders tend to have higher median household incomes. Not groundbreaking, but now you can *quantify* it.

2.1.4 Quick Check: Correlation Direction

If `corr()` returns **-0.65**, which of these is true?

As X increases, Y increases

As X increases, Y decreases

There is no relationship

The data is broken

. . .

B. A negative r means an inverse relationship. Strong, too: 0.65 is well into "moderate to strong" territory. The sign tells you direction; the magnitude tells you strength.

2.1.5 Checking Multiple Correlations

You can compute several correlations in one query:

```sql
SELECT
    round(
        corr(median_hh_income, pct_bachelors_higher)::numeric, 2
    ) AS bachelors_income_r,
    round(
        corr(pct_travel_60_min, median_hh_income)::numeric, 2
    ) AS income_travel_r,
    round(
        corr(pct_travel_60_min, pct_bachelors_higher)::numeric, 2
    ) AS bachelors_travel_r
FROM acs_2014_2018_stats;
```

Run this. Which pair has the weakest correlation? What does that tell you?

. . .

The commute-to-education correlation (`bachelors_travel_r`) is close to zero. Having a degree does not predict whether you will have a long commute. This makes sense: PhDs in Manhattan walk to work; PhDs in rural Montana drive 45 minutes.

2.1.6 Correlation Does NOT Mean Causation

A classic reminder:

- Ice cream sales correlate strongly with drowning deaths
- Does ice cream cause drowning? No. Both increase in summer.

There's an entire website dedicated to absurd correlations: tylervigen.com/spurious-correlations. The divorce rate in Maine correlates with per-capita margarine consumption. r = 0.99.

Always ask: **is there a plausible mechanism**, or is this just two things that happen to move together?

2.1.7 Class Participation: Education and Income

A state education board is debating whether to expand funding for graduate programs. Their argument: counties with higher rates of graduate degrees have significantly higher incomes, so investing in graduate education will boost the economy. Before the board votes, they want data.

Your task: Write a query to find the correlation between `pct_masters_higher` and `median_hh_income`. Round to 2 decimal places. Compare it to the bachelor's correlation we already computed (0.70). Is graduate education a stronger or weaker predictor of income than undergraduate education? What might explain the difference?

```sql
-- Your query here.  Hint: use the corr function, cast to a numeric type before rounding
```

. . .

```sql
SELECT round(
    corr(median_hh_income, pct_masters_higher)::numeric, 2
) AS masters_income_r
FROM acs_2014_2018_stats;
```

The master's correlation is slightly weaker (~0.66 vs ~0.70). This makes intuitive sense: the bachelor's degree represents the bigger jump from "no degree" to "degree holder." The bachelor's rate captures more of the variation in income because it includes everyone with *any* college degree, while the master's rate is a subset.

3 Part 3: Predicting with Regression

Correlation tells us *how strong* a relationship is. Regression tells us *what the relationship looks like* and lets us make predictions.

3.1 Linear Regression

3.1.1 The Regression Line

Simple linear regression fits a straight line through your data:

$\mathbf{Y = bX + a}$

- **b** = slope (how much Y changes for each unit increase in X)
- **a** = y-intercept (the value of Y when X is 0)

The line is calculated using the **least squares method**: it finds the line that minimizes the total squared distance between each data point and the line. Think of it as the line that is "closest" to all points simultaneously.

PostgreSQL gives us both components:

```sql
SELECT
    round(
        regr_slope(median_hh_income, pct_bachelors_higher)::numeric, 2
    ) AS slope,
    round(
        regr_intercept(median_hh_income, pct_bachelors_higher)::numeric, 2
    ) AS y_intercept
FROM acs_2014_2018_stats;
```

Run this. You should get a slope around **926.95** and an intercept around **27,901.15**.

3.1.2 What the Regression Functions Compute

Function	What It Returns	Meaning
`regr_slope(Y, X)`	The slope b	For every 1-unit increase in X, Y changes by b
`regr_intercept(Y, X)`	The y-intercept a	The predicted Y when X is 0
`regr_r2(Y, X)`	The coefficient of determination	Fraction of Y's variance explained by X
`regr_avgx(Y, X)`	Mean of X values	Average of the independent variable
`regr_avgy(Y, X)`	Mean of Y values	Average of the dependent variable
`regr_count(Y, X)`	Count of non-null pairs	Number of rows used in the calculation

The slope of ~927 means: **for every 1 percentage point increase in bachelor's degree holders, the median household income increases by about $927**.

The intercept of ~$27,901 is the predicted income when the bachelor's rate is 0%. This is a theoretical county with zero college graduates.

3.1.3 Making a Prediction

If a county has **30%** of residents with bachelor's degrees, what income does our model predict?

. . .

Y = 926.95 x 30 + 27,901.15 = $55,709.65

You just did linear regression in your head with SQL on a Wednesday, in the evening.

3.1.4 Interpreting the Slope

Our regression shows a slope of 926.95 for income vs. bachelor's rate. If a county's bachelor's rate increases from 25% to 35%, how much does our model predict income will change?

$926.95

$9,269.50

$27,901.15

It depends on the county

. . .

B. The slope is the change per 1 percentage point. A 10-point increase = 10 x $926.95 = **$9,269.50**. The slope is constant across the entire line, so it does not depend on which county you start from.

3.2 r-Squared: How Good Is the Model?

3.2.1 The Coefficient of Determination

r-squared (r^2) tells you what **percentage of the variation in Y** is explained by X. It is literally the Pearson r value squared, but its interpretation is more concrete.

- An r^2 of 0.49 means 49% of the variation in income across counties can be predicted from the bachelor's degree rate alone.
- The remaining 51% comes from factors not in the model (industry, cost of living, geography, etc.).

```sql
SELECT round(
    regr_r2(median_hh_income, pct_bachelors_higher)::numeric, 3
) AS r_squared
FROM acs_2014_2018_stats;
```

Run this. You should get about **0.490**.

An r^2 closer to 1.0 means the model explains most of the variation; closer to 0 means it explains very little. In social science, 0.49 is quite high for a single predictor.

3.2.2 Class Participation: Real Estate Prediction Model

A real estate investment firm is evaluating rural counties for development. They hypothesize that counties with more graduate degree holders have higher incomes, and they want a predictive model to estimate a county's median income from its graduate education rate.

Your task: Write a query that computes the slope, intercept, and r-squared for `median_hh_income` vs. `pct_masters_higher`. Based on your results: if a county has 12% of residents with master's degrees or higher, what income does the model predict? Is the master's rate a better or worse predictor of income than the bachelor's rate ($r^2 = 0.49$)?

```
-- Your query here
```

. . .

```
SELECT
    round(regr_slope(median_hh_income, pct_masters_higher)::numeric, 2) AS slope,
    round(regr_intercept(median_hh_income, pct_masters_higher)::numeric, 2) AS y_intercept,
    round(regr_r2(median_hh_income, pct_masters_higher)::numeric, 3) AS r_squared
FROM acs_2014_2018_stats;
```

The r^2 for master's is lower (~0.37 vs ~0.49). Bachelor's rate is the better predictor. The master's slope is steeper (each percentage point is worth more dollars), but the model explains less of the total variation because the master's rate has a narrower range of values and more noise.

4 Part 4: Variance and Standard Deviation

How spread out is the data? These functions measure the dispersion of values around the mean.

4.1 Measuring Spread

4.1.1 What Variance Measures

Variance measures how far data points are from the mean, on average. The calculation:

1. Find the mean of all values
2. Subtract the mean from each value (these are the "deviations")
3. Square each deviation (to eliminate negative signs)
4. Average the squared deviations

The result is in **squared units**, which makes it hard to interpret directly. If your data is in dollars, variance is in "dollars squared." Not very intuitive.

4.1.2 What Standard Deviation Measures

Standard deviation is the square root of variance. It brings the measurement back into the **original units** of the data. A standard deviation of $12,000 on income data means the typical county's income is about $12,000 away from the average.

In a normal distribution:

- About 68% of values fall within 1 standard deviation of the mean
- About 95% fall within 2 standard deviations
- About 99.7% fall within 3 standard deviations

This is the "68-95-99.7 rule" and it is one of the most useful rules of thumb in statistics.

4.1.3 Population vs. Sample: Why Two Versions?

PostgreSQL provides two versions of each function. The difference comes down to what you divide by:

Function	Divides By	Use When
`var_pop(x)`	N (total count)	Your data IS the entire population
`var_samp(x)`	N - 1	Your data is a sample from a larger population
`stddev_pop(x)`	N	Your data IS the entire population
`stddev_samp(x)`	N - 1	Your data is a sample

Why N - 1 for samples? This is called **Bessel's correction**. When you estimate the mean from a sample, you "use up" one degree of freedom. Dividing by N would systematically underestimate the true spread. Dividing by N - 1 corrects for this bias. The effect is small for large samples (3,142 counties) but significant for small ones (20 students).

4.1.4 Computing Standard Deviation

```sql
SELECT
    round(stddev_pop(median_hh_income)::numeric, 2)
        AS income_stddev_pop,
    round(stddev_samp(median_hh_income)::numeric, 2)
        AS income_stddev_samp
FROM acs_2014_2018_stats;
```

Run this. The population and sample versions are very close because we have 3,142 data points. With N that large, dividing by N vs. N - 1 makes almost no difference. Try it with a table of 10 rows and the gap widens significantly.

4.1.5 Quick Check: Population or Sample?

You have test scores for every student in DATA 503 this semester. Which function do you use?

stddev_pop() because you have everyone

stddev_samp() because this semester is a sample of all possible students

. . .

It depends on your question. If you want the standard deviation of *this specific class*, use `stddev_pop()` because you have the full population. If you want to estimate the standard deviation of *all students who might ever take this class*, use `stddev_samp()` because this class is a sample of that larger population. The question you are answering determines the function.

5 Part 5: Creating Rankings

This is where window functions enter the picture. They compute values across a set of rows *without collapsing them* the way aggregate functions do.

5.1 Understanding the Three Ranking Functions

5.1.1 Setting Up the Demo Table

```sql
CREATE TABLE widget_companies (
    id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    company text NOT NULL,
    widget_output integer NOT NULL
);

INSERT INTO widget_companies (company, widget_output)
VALUES
    ('Dom Widgets', 125000),
    ('Ariadne Widget Masters', 143000),
    ('Saito Widget Co.', 201000),
    ('Mal Inc.', 133000),
    ('Dream Widget Inc.', 196000),
    ('Miles Amalgamated', 620000),
    ('Arthur Industries', 244000),
    ('Fischer Worldwide', 201000);
```

Run this now to create and populate the table.

5.1.2 row_number(), rank(), and dense_rank()

PostgreSQL provides three ranking functions. They all assign positions based on an `ORDER BY`, but they differ in how they handle **ties** (rows with the same value).

```
SELECT
    company,
    widget_output,
    row_number() OVER (ORDER BY widget_output DESC) AS row_num,
    rank()       OVER (ORDER BY widget_output DESC) AS rank,
    dense_rank() OVER (ORDER BY widget_output DESC) AS dense_rank
FROM widget_companies
ORDER BY widget_output DESC;
```

Run this. Pay close attention to what happens at Saito Widget Co. and Fischer Worldwide, which both produced 201,000 widgets.

5.1.3 How Each Function Handles Ties

Here is the output from that query:

Company	Output	row_number	rank	dense_rank
Miles Amalgamated	620,000	1	1	1
Arthur Industries	244,000	2	2	2
Saito Widget Co.	201,000	3	3	3
Fischer Worldwide	201,000	4	3	3
Dream Widget Inc.	196,000	5	**5**	**4**

Company	Output	row_number	rank	dense_rank
Ariadne Widget Masters	143,000	6	6	5
Mal Inc.	133,000	7	7	6
Dom Widgets	125,000	8	8	7

At the tie (201,000):

- **`row_number()`:** 3 and 4. Unique per row; order within the tie is **arbitrary** (non-deterministic).
- **`rank()`:** both 3, then **skips to 5**. Counts rows ranked above you: Dream Widget has four rows above it, so rank 5.
- **`dense_rank()`:** both 3, then **4**. Counts *distinct* values above you: three tiers (620k, 244k, 201k), so Dream Widget is 4.

5.1.4 The Math Behind row_number()

`row_number()`: Simply assigns 1, 2, 3, … sequentially. Every row gets a unique number regardless of ties. Think of it as numbering the rows in the sorted result set.

- Formula: position in the sorted output (1-indexed)
- Ties get different numbers (order within ties is arbitrary)
- Maximum value always equals the total row count

5.1.5 The Math Behind rank()

`rank()`: For each row, counts the number of rows that come **strictly before** it in the ordering, then adds 1. If two rows tie, they both have the same number of rows before them, so they get the same rank.

- Formula: 1 + (number of rows with a strictly better value)
- Gaps appear after ties: if 3 rows tie at rank 2, the next rank is 5 (not 3)
- Maximum value always equals the total row count

5.1.6 The Math Behind dense_rank()

`dense_rank()`: For each row, counts the number of **distinct values** that are strictly better, then adds 1. Ties get the same rank, and the next distinct value gets the very next integer.

- Formula: 1 + (number of distinct values strictly better than this row's value)
- No gaps: ranks are always consecutive integers (1, 2, 3, …)
- Maximum value equals the number of distinct values, not the row count

5.1.7 When to Use Each Function

Function	Use When	Example
`row_number()`	You need a unique identifier for each row, regardless of ties	Pagination, assigning sequential IDs, picking one row per group
`rank()`	You want Olympic-style ranking where ties cause gaps	Competition rankings ("two golds, no silver, then bronze")
`dense_rank()`	You want consecutive ranking labels with no gaps	Top-N analysis where you want exactly N distinct tiers

The Olympic analogy: In the Olympics, if two athletes tie for gold, both get gold and the next athlete gets bronze (rank 3, skipping silver). That is `rank()`. If you instead want to say "there are two first-place finishers and one second-place finisher," that is `dense_rank()`.

5.1.8 Quick Check: rank() with Three-Way Ties

If three companies are tied for 2nd place using `rank()`, what rank does the next company get?

3

4

5

It depends

. . .

C. Using `rank()`, three companies at rank 2 means positions 2, 3, and 4 in the row count are "used" by the tied rows. The formula says: 1 + (number of rows strictly better) = 1 + 4 = 5. The next company gets rank 5.

With `dense_rank()`, the answer would be **3** because only 2 distinct values (rank 1 and rank 2) come before it.

5.2 Ranking Within Groups: PARTITION BY

5.2.1 Setting Up Store Sales

```
CREATE TABLE store_sales (
    store text NOT NULL,
    category text NOT NULL,
    unit_sales bigint NOT NULL,
    CONSTRAINT store_category_key PRIMARY KEY (store, category)
);

INSERT INTO store_sales (store, category, unit_sales)
VALUES
    ('Broders', 'Cereal', 1104),
    ('Wallace', 'Ice Cream', 1863),
    ('Broders', 'Ice Cream', 2517),
    ('Cramers', 'Ice Cream', 2112),
    ('Broders', 'Beer', 641),
    ('Cramers', 'Cereal', 1003),
    ('Cramers', 'Beer', 640),
    ('Wallace', 'Cereal', 980),
    ('Wallace', 'Beer', 988);
```

5.2.2 Ranking Within Categories

`PARTITION BY` divides the data into groups, and the ranking restarts within each group. Think of it like running a separate ranking query for each category, then stitching the results together.

```
SELECT
    category,
    store,
    unit_sales,
    rank() OVER (
        PARTITION BY category
```

```sql
        ORDER BY unit_sales DESC
    ) AS category_rank
FROM store_sales
ORDER BY category, category_rank;
```

Run this. The ranking resets for each category. Broders might be number 1 in Ice Cream but number 2 in Beer. `PARTITION BY` creates separate ranking "lanes."

Without `PARTITION BY`, the window function treats the entire result set as one group. With it, each partition gets its own independent ranking.

5.2.3 Class Participation: National Income Rankings

The U.S. Census Bureau wants to publish a county income ranking report. They need two outputs:

1. A clean national ranking of the top 10 counties by median household income, using `dense_rank()` so there are no confusing gaps if counties tie.
2. A per-state ranking identifying the wealthiest county in each state, so that each state's "top county" can be highlighted.

Task 1: Write a query that ranks all counties by `median_hh_income` (highest first) using `dense_rank()`. Show the county name, state, income, and rank. Limit to the top 10.

```sql
-- Your query here
```

. . .

```sql
SELECT
    county,
    st,
    median_hh_income,
    dense_rank() OVER (ORDER BY median_hh_income DESC) AS income_rank
FROM acs_2014_2018_stats
ORDER BY income_rank
LIMIT 10;
```

5.3 Common Table Expressions (CTEs)

5.3.1 The Problem: Filtering on Window Functions

You might try to write this:

```sql
SELECT county, st, median_hh_income,
    dense_rank() OVER (PARTITION BY st ORDER BY median_hh_income DESC) AS state_rank
FROM acs_2014_2018_stats
WHERE state_rank = 1;  -- ERROR!
```

This fails. PostgreSQL evaluates `WHERE` **before** window functions, so `state_rank` does not exist yet when the filter runs. You need a way to compute the rank first, then filter the results. That is what a CTE does.

5.3.2 What Is a CTE?

A **Common Table Expression** (CTE) is a temporary, named result set that exists only for the duration of a single query. Think of it as creating a throwaway view that you can immediately query against.

The syntax uses the `WITH` keyword:

```sql
WITH cte_name AS (
    -- any SELECT query
    SELECT ...
)
SELECT ...
FROM cte_name
WHERE ...;
```

The query inside the parentheses runs first and produces a result set. The outer query then treats `cte_name` as if it were a regular table.

5.3.3 CTE Example: Step by Step

Here is a simple example. Suppose we want to find counties where income is above the national average:

```sql
WITH national_avg AS (
    SELECT avg(median_hh_income) AS avg_income
    FROM acs_2014_2018_stats
)
SELECT county, st, median_hh_income
FROM acs_2014_2018_stats, national_avg
WHERE median_hh_income > national_avg.avg_income
ORDER BY median_hh_income DESC
LIMIT 10;
```

Step 1: The CTE (`national_avg`) computes the average income across all counties.

Step 2: The outer query references that result and filters counties above the average.

Without the CTE, you would need a subquery in the `WHERE` clause. CTEs are often easier to read, especially when the inner query is complex.

5.3.4 Why CTEs Matter for Window Functions

CTEs solve the window function filtering problem cleanly:

1. **Inner query (the CTE):** Compute the window function (rank, row_number, etc.)
2. **Outer query:** Filter on the computed column

This two-step pattern is one of the most common uses of CTEs, and you will use it frequently.

5.3.5 Class Participation: Wealthiest County Per State

Task 2: A federal grant program awards funding to the single highest-income county in each state. Your job is to determine which county in each state would receive the grant.

Modify the previous query to rank counties **within each state** using `PARTITION BY`. Then filter to show only the top-ranked county per state. Order the final results by income descending to see which states have the wealthiest top counties.

Hint: You cannot `WHERE` on a window function in the same query where it is defined. You will need a subquery or CTE.

```
-- Your query here
```

. . .

```
WITH ranked AS (
    SELECT
        county,
        st,
        median_hh_income,
        dense_rank() OVER (
            PARTITION BY st
            ORDER BY median_hh_income DESC
        ) AS state_rank
    FROM acs_2014_2018_stats
)
SELECT county, st, median_hh_income, state_rank
FROM ranked
```

```sql
WHERE state_rank = 1
ORDER BY median_hh_income DESC;
```

This uses a CTE (Common Table Expression). You cannot filter on `state_rank` in the same `SELECT` where it is defined because window functions execute after `WHERE`. The CTE computes the ranks first, then the outer query filters.

6 Part 6: Rates for Meaningful Comparisons

Raw counts lie. Rates tell the truth.

6.1 Rates Per Thousand

6.1.1 The Problem with Raw Counts

Los Angeles County has 31,000 restaurants. Teton County, Wyoming has 234.

Does LA love food more? No. LA has 10 million people. Teton has 23,000. To compare fairly, we need a **rate**: a count normalized by population size. Rates let us compare apples to apples across groups of different sizes.

6.1.2 Loading the Business Patterns Data

```sql
CREATE TABLE cbp_naics_72_establishments (
    state_fips text,
    county_fips text,
    county text NOT NULL,
    st text NOT NULL,
    naics_2017 text NOT NULL,
    naics_2017_label text NOT NULL,
    year smallint NOT NULL,
    establishments integer NOT NULL,
    CONSTRAINT cbp_fips_key PRIMARY KEY (state_fips, county_fips)
);

copy cbp_naics_72_establishments FROM '/path/to/cbp_naics_72_establishments.csv' WITH (FORMA
```

This table has county-level counts of "Accommodation and Food Services" businesses (NAICS code 72).

6.1.3 Computing Establishments Per 1,000 People

We need a population table. If you have `us_counties_pop_est_2019` from Chapter 5, this query joins the two:

```sql
SELECT
    cbp.county,
    cbp.st,
    cbp.establishments,
    pop.pop_est_2018,
    round(
        (cbp.establishments::numeric / pop.pop_est_2018) * 1000, 1
    ) AS estabs_per_1000
FROM cbp_naics_72_establishments cbp
JOIN us_counties_pop_est_2019 pop
    ON cbp.state_fips = pop.state_fips
    AND cbp.county_fips = pop.county_fips
WHERE pop.pop_est_2018 >= 50000
ORDER BY estabs_per_1000 DESC;
```

The `::numeric` cast is critical. Without it, integer division truncates everything to 0.

6.1.4 Quick Check: Why Cast to Numeric?

What does `5 / 2` return in PostgreSQL?

2.5

2

3

An error

. . .

B. Integer division truncates. `5 / 2 = 2`. To get 2.5, cast one side: `5::numeric / 2`. This trips people up *constantly*. Any time you divide integers and expect a decimal result, cast first.

7 Part 7: Rolling Averages

Monthly data is noisy. Rolling averages smooth it out.

7.1 Moving Averages with Window Functions

7.1.1 Loading Export Data

```sql
CREATE TABLE us_exports (
    year smallint,
    month smallint,
    citrus_export_value bigint,
    soybeans_export_value bigint
);

copy us_exports FROM '/path/to/us_exports.csv' WITH (FORMAT CSV, HEADER);
```

7.1.2 The Raw Data Is Noisy

```sql
SELECT year, month, citrus_export_value
FROM us_exports
ORDER BY year, month;
```

Run this. Citrus exports spike in certain months and drop in others. Seasonal patterns make it hard to see the trend. Is citrus export growing over time? Hard to tell from the raw numbers.

7.1.3 12-Month Rolling Average

A rolling (or moving) average replaces each data point with the average of itself and its neighbors. This smooths out short-term fluctuations and reveals the underlying trend.

```sql
SELECT year, month, citrus_export_value,
    round(
        avg(citrus_export_value)
            OVER(ORDER BY year, month
                 ROWS BETWEEN 11 PRECEDING AND CURRENT ROW), 0
    ) AS twelve_month_avg
FROM us_exports
ORDER BY year, month;
```

Run this. The rolling average smooths out the seasonal spikes. Now you can see whether the overall trend is up, down, or flat.

7.1.4 How the Window Frame Works

`ROWS BETWEEN 11 PRECEDING AND CURRENT ROW` means: "average this row plus the 11 before it." That is 12 rows total, which is one full year of monthly data.

The first 11 rows will not have a full window (there are not 11 preceding rows yet), so the average is computed over whatever rows are available. Row 1 averages only itself; row 2 averages rows 1-2; and so on until row 12, when the full 12-month window kicks in.

Other common window frame options:

Frame Clause	What It Means
`ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING`	5-row centered average
`ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`	Cumulative (running) average
`ROWS BETWEEN 5 PRECEDING AND CURRENT ROW`	6-row trailing average

7.1.5 ROWS vs. RANGE

There are two ways to define a window frame:

- **ROWS**: counts a fixed number of physical rows, regardless of values
- **RANGE**: includes all rows whose value falls within a logical range of the current row

For most time-series work with evenly spaced data, `ROWS` is what you want. `RANGE` is useful when you have gaps in your data and want to include all rows within a value-based range.

7.1.6 Class Participation: Soybean Trade Analysis

An agricultural trade analyst at the USDA suspects that soybean exports have been declining, but the monthly figures are too volatile to draw a conclusion. The analyst needs a smoothed view of the data to present to Congress.

Your task: Write a query that computes a **6-month rolling average** for `soybeans_export_value`. Show the year, month, raw export value, and the rolling average. Based on the smoothed output, does the trend appear to be increasing, decreasing, or flat?

```sql
-- Your query here
```

. . .

```sql
SELECT year, month, soybeans_export_value,
    round(
        avg(soybeans_export_value)
            OVER(ORDER BY year, month
                 ROWS BETWEEN 5 PRECEDING AND CURRENT ROW), 0
    ) AS six_month_avg
FROM us_exports
ORDER BY year, month;
```

`ROWS BETWEEN 5 PRECEDING AND CURRENT ROW` = 6 rows total (current row + 5 before it).

8 Part 8: What We Learned

8.1 Summary

8.1.1 Key Functions Reference

Function	What It Computes
`corr(Y, X)`	Pearson correlation coefficient (-1 to +1)
`regr_slope(Y, X)`	Slope of the least-squares regression line
`regr_intercept(Y, X)`	Y-intercept of the regression line
`regr_r2(Y, X)`	Coefficient of determination (0 to 1)
`var_pop(x)` / `var_samp(x)`	Population / sample variance (divides by N vs. N-1)
`stddev_pop(x)` / `stddev_samp(x)`	Population / sample standard deviation
`row_number()`	Sequential numbering, every row unique, ties are arbitrary
`rank()`	Same rank for ties, gaps after ties (Olympic-style)
`dense_rank()`	Same rank for ties, no gaps (consecutive integers)
`PARTITION BY`	Restarts the window function for each group
`ROWS BETWEEN ... AND ...`	Defines a window frame for rolling calculations

8.1.2 The Big Ideas

1. **Correlation measures linear relationships.** A strong r does not mean causation. Always ask whether there is a plausible mechanism.
2. **Regression predicts.** slope x X + intercept = predicted Y. The slope tells you the rate of change per unit.
3. **r-squared tells you how much** of the variation your model explains. The rest is from factors not in the model.

4. **Population vs. sample** matters for variance and standard deviation. Use N for populations, N-1 for samples (Bessel's correction).
5. **Use rates, not raw counts** when comparing groups of different sizes.
6. **Window functions do not collapse rows.** They compute across a "window" of related rows while keeping every row in the output.
7. **Rolling averages smooth noise.** Essential for time-series analysis. Choose a window size that matches your data's periodicity.
8. **Always cast to numeric** before dividing integers. `5 / 2 = 2` in PostgreSQL.

8.1.3 References

1. DeBarros, A. (2022). *Practical SQL: A Beginner's Guide to Storytelling with Data* (2nd ed.). No Starch Press. Chapter 11.
2. PostgreSQL Window Functions: https://www.postgresql.org/docs/current/functions-window.html
3. PostgreSQL Aggregate Functions: https://www.postgresql.org/docs/current/functions-aggregate.html
4. Spurious Correlations: https://www.tylervigen.com/spurious-correlations