# Lecture 11-1: Statistical Functions in SQL

**DATA 351: Data Management with SQL**

Lucas P. Cordova, Ph.D.

2026-03-04

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. Shocking, I know.

## 1.1 Loading the Chapter Database

### 1.1.1 Step 1: Create a Fresh Database

```
CREATE DATABASE analysis;
```

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

### 1.1.2 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.

### 1.1.3 Step 3: Import the CSV

```
\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:

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

#### 1.1.4 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.

| r value | Interpretation |
|---|---|
| +1.0 | Perfect positive (as X goes up, Y goes up proportionally) |
| +0.6 to +0.9 | Strong positive |
| +0.3 to +0.59 | Moderate positive |
| +0.01 to +0.29 | 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.

#### 2.1.2 corr(Y, X) in PostgreSQL

```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.

### 2.1.3 Quick Quiz: Correlation Direction

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

A. As X increases, Y increases B. As X increases, Y decreases C. There is no relationship D. The data is broken

. . .

**B.** A negative r means an inverse relationship. Strong, too – -0.65 is well into "moderate to strong" territory.

### 2.1.4 Checking Multiple Correlations

You can compute several correlations in one query:

```
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 doesn't predict whether you'll have a long commute. Makes sense – PhDs in Manhattan walk to work; PhDs in rural Montana drive 45 minutes.

### 2.1.5 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?

### 2.1.6 Try It: Your Turn

Write a query to find the correlation between `pct_masters_higher` and `median_hh_income`. Round to 2 decimal places.

Is it stronger or weaker than the bachelor's correlation? Why might that be?

```
-- Your query here
```

. . .

```
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. This makes intuitive sense – a master's degree adds income, but the bachelor's is the bigger jump from "no degree" to "degree." The bachelor's rate captures more of the variation.

# 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)

PostgreSQL gives us both:

```
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 Does That Mean?

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%. (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 * 30 + 27,901.15 = $55,709.65**

You just did linear regression. In your head. With SQL. At 10 AM on a Wednesday.

### 3.1.4 Quick Quiz: 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?

A. $926.95 B. $9,269.50 C. $27,901.15 D. It depends on the county

. . .

**B.** The slope is the change per 1 percentage point. A 10-point increase = 10 * $926.95 = **$9,269.50**.

## 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.

```
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**.

That means ~49% of the variation in median income across counties is explained by the bachelor's degree rate. The other 51% comes from other factors (industry, cost of living, geography, etc.).

### 3.2.2 Try It: Predict and Evaluate

Write a query that computes the slope, intercept, and r-squared for `median_hh_income` vs. `pct_masters_higher`. Based on your $r^2$, is the master's rate a better or worse predictor of income than the bachelor's rate?

```
-- 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.

# 4 Part 4: Variance and Standard Deviation

How spread out is the data? Time for everyone's favorite statistics topic.

## 4.1 Measuring Spread

### 4.1.1 Variance and Standard Deviation

- **Variance** = average squared deviation from the mean. Hard to interpret because the units are squared.
- **Standard deviation** = square root of variance. Same units as the original data. Much more useful.

PostgreSQL has both population and sample versions:

| Function | Type | Use When |
|---|---|---|
| `var_pop()` | Population variance | Your data IS the entire population |
| `var_samp()` | Sample variance | Your data is a sample of a larger population |
| `stddev_pop()` | Population std dev | Your data IS the entire population |
| `stddev_samp()` | Sample std dev | Your data is a sample |

### 4.1.2 Computing Standard Deviation

```
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. The difference matters more with small samples.

#### 4.1.3 Quick Quiz: Population or Sample?

You have test scores for every student in DATA 351 this semester. Which function do you use?

A. `stddev_pop()` – because you have everyone B. `stddev_samp()` – because this semester is a sample of all possible students

. . .

**It depends on your question!** If you want the std dev of *this class*, use `pop`. If you want to estimate the std dev of *all students who might take this class*, use `samp`. Statistics is fun like that.

## 5 Part 5: Creating Rankings

This is where window functions enter the picture. And they are *cool*.

### 5.1 rank() and dense_rank()

#### 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 rank() vs. dense_rank()

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

**Run this.** Notice what happens at Saito Widget Co. and Fischer Worldwide – they both produced 201,000 widgets.

. . .

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

`rank()` skips position 4 (goes 3, 3, 5). `dense_rank()` doesn't skip (goes 3, 3, 4). Use `dense_rank()` when you don't want gaps.

### 5.1.3 Quick Quiz: rank() Output

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

A. 3 B. 4 C. 5 D. It depends

. . .

**C.** Three companies at rank 2 means positions 2, 3, 4 are "used." The next company gets rank 5.

## 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

```
SELECT
    category,
    store,
    unit_sales,
    rank() OVER (
        PARTITION BY category
        ORDER BY unit_sales DESC
    )
FROM store_sales
ORDER BY category, rank() OVER (
    PARTITION BY category ORDER BY unit_sales DESC
);
```

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

### 5.2.3 Try It: Rank the Counties

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.

```
-- Your query here
```

. . .

```
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;
```

### 5.2.4 Try It: Rank Within States

Now modify the query to rank counties **within each state** using `PARTITION BY`. Show only rank 1 per state – the richest county in each state.

Hint: you'll need a subquery or CTE because you can't `WHERE` on a window function.

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

This uses a CTE! You cannot filter on `state_rank` in the same `SELECT` where it's defined, so we wrap it and filter in the outer query.

# 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**.

### 6.1.2 Loading the Business Patterns Data

```
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, great. If not, we can compute rates using the ACS data. For now, here's the pattern:

```
SELECT
    cbp.county,
    cbp.st,
    cbp.establishments,
    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 Quiz: Why Cast to Numeric?

What does `5 / 2` return in PostgreSQL?

A. 2.5 B. 2 C. 3 D. An error

. . .

**B.** Integer division truncates. `5 / 2 = 2`. To get 2.5, cast one side: `5::numeric / 2`. This trips people up *constantly*.

## 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

```
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

```
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 exports growing over time? Hard to tell from the raw numbers.

### 7.1.3 12-Month Rolling Average

```
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's 12 rows total – one full year of monthly data.

The first 11 rows won't have a full window (there aren't 11 preceding rows yet), so the average is computed over whatever is available.

### 7.1.5 Try It: Soybean Rolling Average

Write a query that computes a 6-month rolling average for `soybeans_export_value`. Show year, month, the raw value, and the rolling average.

```
-- Your query here
```

. . .

```
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.

# 8 Part 8: What We Learned

## 8.1 Summary

### 8.1.1 Key Functions

| Function | What It Does |
|---|---|
| `corr(Y, X)` | Pearson correlation coefficient (-1 to +1) |
| `regr_slope(Y, X)` | Slope of the regression line |
| `regr_intercept(Y, X)` | Y-intercept of the regression line |
| `regr_r2(Y, X)` | Coefficient of determination (0 to 1) |
| `var_pop()` / `var_samp()` | Population / sample variance |
| `stddev_pop()` / `stddev_samp()` | Population / sample standard deviation |

| Function | What It Does |
|---|---|
| `rank()` | Ranking with gaps on ties |
| `dense_rank()` | Ranking without gaps on ties |
| `PARTITION BY` | Rank within groups |
| `ROWS BETWEEN ... AND ...` | Define a window frame for rolling calculations |

### 8.1.2 The Big Ideas

1. **Correlation measures linear relationships.** Strong r does not mean causation.
2. **Regression predicts.** slope * X + intercept = predicted Y.
3. **r-squared tells you how much** of the variation your model explains.
4. **Use rates, not raw counts** when comparing differently-sized populations.
5. **Window functions don't collapse rows.** They compute across a "window" of related rows.
6. **Rolling averages smooth noise.** Essential for time-series analysis.
7. **Always cast to numeric** before dividing integers. `5 / 2 = 2` in PostgreSQL. Don't learn this the hard way.

### 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