

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	1
2	Part 2: Measuring Correlation	3
3	Part 3: Predicting with Regression	5
4	Part 4: Variance and Standard Deviation	8
5	Part 5: Creating Rankings	9
6	Part 6: Rates for Meaningful Comparisons	13
7	Part 7: Rolling Averages	14
8	Part 8: What We Learned	16

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

```
1 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:

```
1 CREATE TABLE acs_2014_2018_stats (  
2     geoid text CONSTRAINT geoid_key PRIMARY KEY,  
3     county text NOT NULL,  
4     st text NOT NULL,  
5     pct_travel_60_min numeric(5,2),  
6     pct_bachelors_higher numeric(5,2),  
7     pct_masters_higher numeric(5,2),  
8     median_hh_income integer,  
9     CHECK (pct_masters_higher <= pct_bachelors_higher)  
10 );
```

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

```
1 \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:

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

1.1.4 Quick Look at the Data

```
1 SELECT * FROM acs_2014_2018_stats
2 ORDER BY median_hh_income DESC
3 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

```
1 SELECT corr(median_hh_income, pct_bachelors_higher)
2 AS bachelors_income_r
3 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:

```
1 SELECT
2     round(
3         corr(median_hh_income, pct_bachelors_higher)::numeric, 2
4     ) AS bachelors_income_r,
5     round(
6         corr(pct_travel_60_min, median_hh_income)::numeric, 2
7     ) AS income_travel_r,
8     round(
9         corr(pct_travel_60_min, pct_bachelors_higher)::numeric, 2
10    ) AS bachelors_travel_r
11 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?

```
1 -- Your query here
2
3 . . .
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

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:

$$Y = bX + a$$

- **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:

```
1 SELECT
2     round(
3         regr_slope(median_hh_income, pct_bachelors_higher)::numeric, 2
4     ) AS slope,
5     round(
6         regr_intercept(median_hh_income, pct_bachelors_higher)::numeric, 2
7     ) AS y_intercept
8 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.

```
1 SELECT round(  
2     regr_r2(median_hh_income, pct_bachelors_higher)::numeric, 3  
3 ) AS r_squared  
4 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?

```
1 -- Your query here  
  
...  
  
1 SELECT  
2     round(regr_slope(median_hh_income, pct_masters_higher)::numeric, 2) AS slope,  
3     round(regr_intercept(median_hh_income, pct_masters_higher)::numeric, 2) AS y_intercept,  
4     round(regr_r2(median_hh_income, pct_masters_higher)::numeric, 3) AS r_squared  
5 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
<code>var_pop()</code>	Population variance	Your data IS the entire population
<code>var_samp()</code>	Sample variance	Your data is a sample of a larger population
<code>stddev_pop()</code>	Population std dev	Your data IS the entire population
<code>stddev_samp()</code>	Sample std dev	Your data is a sample

4.1.2 Computing Standard Deviation

```
1 SELECT
2     round(stddev_pop(median_hh_income)::numeric, 2)
3     AS income_stddev_pop,
4     round(stddev_samp(median_hh_income)::numeric, 2)
5     AS income_stddev_samp
6 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

```
1 CREATE TABLE widget_companies (  
2     id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,  
3     company text NOT NULL,  
4     widget_output integer NOT NULL  
5 );  
6  
7 INSERT INTO widget_companies (company, widget_output)  
8 VALUES  
9     ('Dom Widgets', 125000),  
10    ('Ariadne Widget Masters', 143000),  
11    ('Saito Widget Co.', 201000),  
12    ('Mal Inc.', 133000),  
13    ('Dream Widget Inc.', 196000),  
14    ('Miles Amalgamated', 620000),  
15    ('Arthur Industries', 244000),  
16    ('Fischer Worldwide', 201000);
```

Run this now to create and populate the table.

5.1.2 rank() vs. dense_rank()

```
1 SELECT
2     company,
3     widget_output,
4     rank() OVER (ORDER BY widget_output DESC),
5     dense_rank() OVER (ORDER BY widget_output DESC)
6 FROM widget_companies
7 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

```
1 CREATE TABLE store_sales (  
2     store text NOT NULL,  
3     category text NOT NULL,  
4     unit_sales bigint NOT NULL,  
5     CONSTRAINT store_category_key PRIMARY KEY (store, category)  
6 );  
7  
8 INSERT INTO store_sales (store, category, unit_sales)  
9 VALUES  
10    ('Broders', 'Cereal', 1104),  
11    ('Wallace', 'Ice Cream', 1863),  
12    ('Broders', 'Ice Cream', 2517),  
13    ('Cramers', 'Ice Cream', 2112),  
14    ('Broders', 'Beer', 641),  
15    ('Cramers', 'Cereal', 1003),  
16    ('Cramers', 'Beer', 640),  
17    ('Wallace', 'Cereal', 980),  
18    ('Wallace', 'Beer', 988);
```

5.2.2 Ranking Within Categories

```
1 SELECT  
2     category,  
3     store,  
4     unit_sales,  
5     rank() OVER (  
6         PARTITION BY category  
7         ORDER BY unit_sales DESC  
8     )  
9 FROM store_sales  
10 ORDER BY category, rank() OVER (  
11     PARTITION BY category ORDER BY unit_sales DESC  
12 );
```

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.

```
1 -- Your query here
2
3 . . .
4
5 SELECT
6     county,
7     st,
8     median_hh_income,
9     dense_rank() OVER (ORDER BY median_hh_income DESC) AS income_rank
10 FROM acs_2014_2018_stats
11 ORDER BY income_rank
12 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.

```
1 -- Your query here
2
3 . . .
4
5 WITH ranked AS (
6     SELECT
7         county,
8         st,
9         median_hh_income,
10        dense_rank() OVER (
11            PARTITION BY st
12            ORDER BY median_hh_income DESC
13        ) AS state_rank
14    FROM acs_2014_2018_stats
15 )
16 SELECT county, st, median_hh_income, state_rank
17 FROM ranked
18 WHERE state_rank = 1
19 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

```
1 CREATE TABLE cbp_naics_72_establishments (  
2     state_fips text,  
3     county_fips text,  
4     county text NOT NULL,  
5     st text NOT NULL,  
6     naics_2017 text NOT NULL,  
7     naics_2017_label text NOT NULL,  
8     year smallint NOT NULL,  
9     establishments integer NOT NULL,  
10    CONSTRAINT cbp_fips_key PRIMARY KEY (state_fips, county_fips)  
11 );  
12  
13 \copy cbp_naics_72_establishments FROM '/path/to/cbp_naics_72_establishments.csv' WITH (FORM
```

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:

```
1 SELECT
2     cbp.county,
3     cbp.st,
4     cbp.establishments,
5     round(
6         (cbp.establishments::numeric / pop.pop_est_2018) * 1000, 1
7     ) AS estabs_per_1000
8 FROM cbp_naics_72_establishments cbp
9 JOIN us_counties_pop_est_2019 pop
10    ON cbp.state_fips = pop.state_fips
11    AND cbp.county_fips = pop.county_fips
12 WHERE pop.pop_est_2018 >= 50000
13 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

```
1 CREATE TABLE us_exports (  
2     year smallint,  
3     month smallint,  
4     citrus_export_value bigint,  
5     soybeans_export_value bigint  
6 );  
7  
8 \copy us_exports FROM '/path/to/us_exports.csv' WITH (FORMAT CSV, HEADER);
```

7.1.2 The Raw Data Is Noisy

```
1 SELECT year, month, citrus_export_value  
2 FROM us_exports  
3 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

```
1 SELECT year, month, citrus_export_value,  
2     round(  
3         avg(citrus_export_value)  
4         OVER(ORDER BY year, month  
5             ROWS BETWEEN 11 PRECEDING AND CURRENT ROW), 0  
6     ) AS twelve_month_avg  
7 FROM us_exports  
8 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.

```
1 -- Your query here
. . .
1 SELECT year, month, soybeans_export_value,
2     round(
3         avg(soybeans_export_value)
4             OVER(ORDER BY year, month
5                 ROWS BETWEEN 5 PRECEDING AND CURRENT ROW), 0
6     ) AS six_month_avg
7 FROM us_exports
8 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
<code>corr(Y, X)</code>	Pearson correlation coefficient (-1 to +1)
<code>regr_slope(Y, X)</code>	Slope of the regression line
<code>regr_intercept(Y, X)</code>	Y-intercept of the regression line
<code>regr_r2(Y, X)</code>	Coefficient of determination (0 to 1)
<code>var_pop()</code> / <code>var_samp()</code>	Population / sample variance
<code>stddev_pop()</code> / <code>stddev_samp()</code>	Population / sample standard deviation

Function	What It Does
<code>rank()</code>	Ranking with gaps on ties
<code>dense_rank()</code>	Ranking without gaps on ties
<code>PARTITION BY</code>	Rank within groups
<code>ROWS BETWEEN ... AND ...</code>	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.** $\text{slope} * X + \text{intercept} = \text{predicted } Y$.
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>