Twenty-five SQL practice exercises

These questions and example solutions will keep your skills sharp.

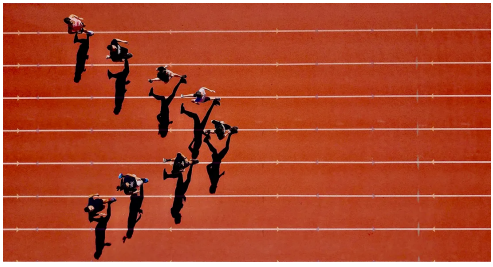


Photo credit: Unsplash (https://unsplash.com/photos/atSaEOeE8Nk)

Introduction

Structured query language (SQL) is used to retrieve and manipulate data stored in relational databases. Gaining working proficiency in SQL is an important prerequisite for many technology jobs and requires a bit of practice.

To complement SQL training resources (PGExercises (https://pgexercises.com/), LeetCode (https://leetcode.com/problemset/database/), HackerRank (https://www.hackerrank.com/domains/sql), Mode (https://mode.com/sql-tutorial/introduction-to-sql)) available on the web, I've compiled a list of my favorite questions that you can tackle by hand or solve with a PostgreSQL instance.

These questions cover the following critical concepts:

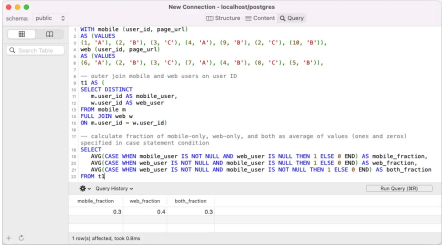
- **Basic retrieval** (SELECT, FROM)
- **Creating and aliasing** (WITH, AS, GENERATE_SERIES)
- **Filtering** (DISTINCT, WHERE, HAVING, AND, OR, IN, NOT IN)
- **Aggregation** (GROUP BY with COUNT, SUM, AVERAGE)

- **Joins** (INNER JOIN, LEFT JOIN, FULL OUTER JOIN on one or multiple (in)equalities, CROSS JOIN, UNION and UNION ALL)
- **Conditional statements** (CASE - WHEN - THEN - ELSE - END)
- **Window functions** (RANK, DENSE_RANK, ROW_NUMBER, SUM with PARTITION BY - ORDER BY)
- **Formatting** (LIMIT, ORDER BY, casting as an integer, float, or date, CONCAT, COALESCE)
- **Arithmetic operations and comparisons** (+, -, *, /, //, ^, <, >, =, !=)
- **Datetime operations** (EXTRACT(month/day/year))

Try it yourself

You can try these out yourself by downloading PostgreSQL (https://postgresapp.com/) and PSequel

(http://www.psequel.com/) (see this tutorial (https://www.youtube.com/watch?v=xaWlS9HtWYw) for a step-by-step installation guide) and then running the queries shown in the grey boxes in the text below. PSequel is only available for Mac — if you're using a PC, you can try one of these (https://alternativeto.net/software/psequel/?platform=windows) Windows alternatives.



Try these queries yourself using PSequel (http://www.psequel.com/) and the input tables provided below.

The first block of text in each query shown below establishes the input table and follows the format:

```
WITH input_table (column_1,
column_2)
AS (VALUES
(1, 'A'), (2, 'B'))
```

You can query against the input table using PSequel (shown above) and easily construct new tables for your own problems using this template.

Web-based SQL training resources fall short along a few dimensions. LeetCode, for instance, doesn't support the use of window functions and hides its most interesting questions behind a paywall. In addition, running SQL queries in your browser can be extremely slow — the data sets are large and retrieval speed is often throttled for non-premium users. Locally executing a query, on the other hand, is instantaneous and allows for rapid iteration through syntax bugs and intermediate tables. I've found this to be a more satisfying learning experience.

The questions outlined below include example solutions confirmed to work in PostgreSQL. Keep in mind there is usually more than one way to obtain the correct answer to a SQL problem. My preference is to use common table expressions (CTEs (https://www.sqlservertutorial.net/sql-server-basics/sql-server-cte/)) rather than nested subqueries — CTEs allow for a more linear illustration of the data wrangling sequence. Both approaches, however, can yield identical solutions. I also like to follow the convention (https://www.sqlstyle.guide/) of keeping SQL operators in all caps (SELECT, FROM, WHERE, etc.), column names in lowercase (user_id, date, etc.), and simple table aliasing (t1, t2, etc.) where possible.

The code snippets shown below can be run in PSequel as-is to yield the displayed result. Note one quirk of Postgres: fractions must be multiplied

by 1.0 to convert from integer to float format. This is not needed in other implementations of SQL and is not expected in interviews.

Feel free to leave your alternative answers in the comments!

Questions

1. Cancellation rates

From the following table of user IDs, actions, and dates, write a query to return the publication and cancellation rate for each user.

users

user_id	action	date
1	start	1-1-20
1	cancel	1-2-20
2	start	1-3-20
2	publish	1-4-20
3	start	1-5-20
3	cancel	1-6-20
4	start	1-7-20

Desired output

user_id	publish_rate	cancel_rate
1	0.5	0.5
2	1.0	0.0
3	0.0	1.0

```
WITH users (user_id, action, date)
AS (VALUES
(1,'start', CAST('01-01-20' AS
date)),
(1,'cancel', CAST('01-02-20' AS
date)),
(2,'start', CAST('01-03-20' AS
date)),
(2,'publish', CAST('01-04-20' AS
date)),
(3,'start', CAST('01-05-20' AS
date)),
(3,'cancel', CAST('01-06-20' AS
date)),
(1,'start', CAST('01-07-20' AS
date)),
(1,'publish', CAST('01-08-20' AS
date))),
-- retrieve count of starts,
cancels, and publishes for each
user1 AS (
SELECT
   user_id,
   SUM(CASE WHEN action = 'start'
THEN 1 ELSE 0 END) AS starts,
   SUM(CASE WHEN action = 'cancel'
THEN 1 ELSE 0 END) AS cancels,
   SUM(CASE WHEN action =
'publish' THEN 1 ELSE 0 END) AS
```

```
publishes
FROM users
GROUP BY 1
ORDER BY 1)-- calculate
publication, cancelation rate for
each user by dividing by number of
starts, casting as float by
multiplying by 1.0 (default floor
division is a quirk of some SQL
tools, not always needed)SELECT
   user_id,
   1.0*publishes/starts AS
publish_rate,
   1.0*cancels/starts AS
cancel_rate
FROM t1
```

2. Changes in net worth

From the following table of transactions between two users, write a query to return the change in net worth for each user, ordered by decreasing net change.

transactions

sender	receiver	amount	transaction_date
5	2	10	2-12-20
1	3	15	2-13-20
2	1	20	2-13-20
2	3	25	2-14-20
3	1	20	2-15-20
3	2	15	2-15-20
1	4	5	2-16-20

Desired output

user	net_change
1	20
3	5
4	5
5	-10
2	-20

```
WITH transactions (sender,
receiver, amount,
transaction_date)
AS (VALUES
(5, 2, 10, CAST('2-12-20' AS
date)),
(1, 3, 15, CAST('2-13-20' AS
date)),
(2, 1, 20, CAST('2-13-20' AS
date)),
(2, 3, 25, CAST('2-14-20' AS
date)),
(3, 1, 20, CAST('2-15-20' AS
date)),
(3, 2, 15, CAST('2-15-20' AS
date)),
(1, 4, 5, CAST('2-16-20' AS
date))),
-- sum amounts for each sender
(debits) and receiver
(credits)debits AS (
SELECT
   sender,
   SUM(amount) AS debited
FROM transactions
GROUP BY 1 ),credits AS (
SELECT
   receiver,
   SUM(amount) AS credited
```

```
FROM transactions
GROUP BY 1 )-- full (outer) join
debits and credits tables on user
id, taking net change as
difference between credits and
debits, coercing nulls to zeros
with coalesce()SELECT
   COALESCE(sender, receiver) AS
user,
   COALESCE(credited, 0) -
COALESCE(debited, 0) AS net_change
FROM debits d
FULL JOIN credits c
ON d.sender = c.receiver
ORDER BY 2 DESC
```

3. Most frequent items

From the following table containing a list of dates and items ordered, write a query to return the most frequent item ordered on each date. Return multiple items in the case of a tie.

items

date	item
1-1-20	apple
1-1-20	apple
1-1-20	pear
1-1-20	pear
1-2-20	pear
1-2-20	pear
1-2-20	pear
1-2-20	orange

Desired output

date	item
1-1-20	apple
1-1-20	pear
1-2-20	pear

```
WITH items (date, item)
AS (VALUES
(CAST('01-01-20' AS
date),'apple'),
(CAST('01-01-20' AS
date),'apple'),
(CAST('01-01-20' AS date),'pear'),
(CAST('01-01-20' AS date),'pear'),
(CAST('01-02-20' AS date),'pear'),
(CAST('01-02-20' AS date),'pear'),
(CAST('01-02-20' AS date),'pear'),
(CAST('01-02-20' AS
date),'orange')),-- add an item
count column to existing table,
grouping by date and item
columnst1 AS (
SELECT
   date,
   item,
   COUNT(*) AS item_count
FROM items
GROUP BY 1, 2
ORDER BY 1),-- add a rank column
in descending order, partitioning
by datet2 AS (
SELECT
   *,
   RANK() OVER (PARTITION BY date
ORDER BY item_count DESC) AS
```

```
    date_rank
    FROM t1)-- return all dates and
    items where rank = 1SELECT
       date,
       item
    FROM t2
    WHERE date_rank = 1
```

4. Time difference between latest actions

From the following table of user actions, write a query to return for each user the time elapsed between the last action and the second-to-last action, in ascending order by user ID.

users

user_id	action	action_date
1	Start	2-12-20
1	Cancel	2-13-20
2	Start	2-11-20
2	Publish	2-14-20
3	Start	2-15-20
3	Cancel	2-15-20
4	Start	2-18-20
1	Publish	2-19-20

Desired output

user_id	days_elapsed
1	6
2	3
3	0
4	NULL

```
WITH users (user_id, action,
action_date)
AS (VALUES
(1, 'start', CAST('2-12-20' AS
date)),
(1, 'cancel', CAST('2-13-20' AS
date)),
(2, 'start', CAST('2-11-20' AS
date)),
(2, 'publish', CAST('2-14-20' AS
date)),
(3, 'start', CAST('2-15-20' AS
date)),
(3, 'cancel', CAST('2-15-20' AS
date)),
(4, 'start', CAST('2-18-20' AS
date)),
(1, 'publish', CAST('2-19-20' AS
date))),
-- create a date rank column,
partitioned by user ID, using the
ROW_NUMBER() window function t1 AS
(
SELECT
   *,
   ROW_NUMBER() OVER (PARTITION BY
user_id ORDER BY action_date DESC)
AS date_rank
FROM users ),-- filter on date
```

```
rank column to pull latest and
next latest actions from this
tablelatest AS (
SELECT *
FROM t1
WHERE date_rank = 1 ),next_latest
AS (
SELECT *
FROM t1
WHERE date_rank = 2 )-- left join
these two tables, subtracting
latest from second latest to get
time elapsed SELECT
   l1.user_id,
   l1.action_date - l2.action_date
AS days_elapsed
FROM latest l1
LEFT JOIN next_latest l2
ON l1.user_id = l2.user_id
ORDER BY 1
```

5. Super users

A company defines its super users as those who have made at least two transactions. From the following table, write a query to return, for each user, the date when they become a super user,

ordered by oldest super users first. Users who are not super users should also be present in the table.

users

user_id	product_id	transaction_date
1	101	2-12-20
2	105	2-13-20
1	111	2-14-20
3	121	2-15-20
1	101	2-16-20
2	105	2-17-20
4	101	2-16-20
3	105	2-15-20

Desired output

user_id	superuser_date
1	2-14-20
3	2-15-20
2	2-17-20
4	NULL

```
WITH users (user_id, product_id,
transaction_date)
AS (VALUES
(1, 101, CAST('2-12-20' AS date)),
(2, 105, CAST('2-13-20' AS date)),
(1, 111, CAST('2-14-20' AS date)),
(3, 121, CAST('2-15-20' AS date)),
(1, 101, CAST('2-16-20' AS date)),
(2, 105, CAST('2-17-20' AS date)),
(4, 101, CAST('2-16-20' AS date)),
(3, 105, CAST('2-15-20' AS
date))),
-- create a transaction number
column using ROW_NUMBER(),
partitioning by user IDt1 AS (
SELECT
   *,
   ROW_NUMBER() OVER (PARTITION BY
user_id ORDER BY transaction_date)
AS transaction_number
FROM users),-- filter resulting
table on transaction_number = 2t2
AS (
SELECT
   user_id,
   transaction_date
FROM t1
WHERE transaction_number = 2 ),--
left join super users onto full
```

```
user table, order by date t3 AS (
SELECT DISTINCT user_id
FROM users )SELECT
   t3.user_id,
   transaction_date AS
superuser_date
FROM t3
LEFT JOIN t2
ON t3.user_id = t2.user_id
ORDER BY 2
```

6. Content recommendation (hard)

Using the following two tables, write a query to return page recommendations to a social media user based on the pages that their friends have liked, but that they have not yet marked as liked. Order the result by ascending user ID. Source (https://www.glassdoor.com/Interview/Write-an-SQL-query-that-makes-recommendations-using-the-pages-that-your-friends-liked-Assume-you-have-two-tables-a-two-c-QTN_1413464.htm).

friends

user_id	friend
1	2
1	3
1	4
2	1
3	1
3	4
4	1
4	3

likes

user_id	page_likes
1	A
1	B
1	C
2	A
3	B
3	C
4	B

Desired output

user_id	recommended_page
2	B
2	C
3	A
4	A
4	C

```
WITH friends (user_id, friend)
AS (VALUES
(1, 2), (1, 3), (1, 4), (2, 1),
(3, 1), (3, 4), (4, 1), (4,
3)),likes (user_id, page_likes)
AS (VALUES
(1, 'A'), (1, 'B'), (1, 'C'), (2,
'A'), (3, 'B'), (3, 'C'), (4,
'B')),
-- inner join friends and page
likes tables on user_idt1 AS (
SELECT
   l.user_id,
   l.page_likes,
   f.friend
FROM likes l
JOIN friends f
ON l.user_id = f.user_id ),-- left
join likes on this, requiring user
= friend and user likes = friend
likes t2 AS (
SELECT
   t1.user_id,
   t1.page_likes,
   t1.friend,
   l.page_likes AS friend_likes
FROM t1
LEFT JOIN likes l
ON t1.friend = l.user_id
```

```
AND t1.page_likes = l.page_likes
)-- if a friend pair doesn't share
a common page like, friend likes
column will be null - pull out
these entries SELECT DISTINCT
   friend AS user_id,
   page_likes AS recommended_page
FROM t2
WHERE friend_likes IS NULL
ORDER BY 1
```

7. Mobile and web visitors

With the following two tables, return the fraction of users who only visited mobile, only visited web, and visited both.

mobile

user_id	page_url
1	A
2	B
3	C
4	A
9	B
2	C
10	B

web

user_id	page_url
6	A
2	B
3	C
7	A
4	B
8	C
5	B

Desired output

mobile_fraction	web_fraction	both_fraction
0.3	0.4	0.3

```
WITH mobile (user_id, page_url)
AS (VALUES
(1, 'A'), (2, 'B'), (3, 'C'), (4,
'A'), (9, 'B'), (2, 'C'), (10,
'B')),web (user_id, page_url)
AS (VALUES
(6, 'A'), (2, 'B'), (3, 'C'), (7,
'A'), (4, 'B'), (8, 'C'), (5,
'B')),
-- outer join mobile and web users
on user IDt1 AS (
SELECT DISTINCT
   m.user_id AS mobile_user,
   w.user_id AS web_user
FROM mobile m
FULL JOIN web w
ON m.user_id = w.user_id)--
calculate fraction of mobile-only,
web-only, and both as average of
values (ones and zeros) specified
in case statement conditionSELECT
   AVG(CASE WHEN mobile_user IS
NOT NULL AND web_user IS NULL THEN
1   ELSE 0 END) AS
mobile_fraction,
   AVG(CASE WHEN web_user IS NOT
NULL AND mobile_user IS NULL THEN
1 ELSE 0 END) AS web_fraction,
   AVG(CASE WHEN web_user IS NOT
```

```
NULL AND mobile_user IS NOT NULL
THEN 1 ELSE 0 END) AS
both_fraction
FROM t1
```

8. Upgrade rate by product action (hard)

Given the following two tables, return the fraction of users, rounded to two decimal places, who accessed feature two (type: F2 in events table) and upgraded to premium within the first 30 days of signing up.

users

user_id	name	join_date
1	Jon	2-14-20
2	Jane	2-14-20
3	Jill	2-15-20
4	Josh	2-15-20
5	Jean	2-16-20
6	Justin	2-17-20
7	Jeremy	2-18-20

events

user_id	type	access_date
1	F1	3-1-20
2	F2	3-2-20
2	P	3-12-20
3	F2	3-15-20
4	F2	3-15-20
1	P	3-16-20
3	P	3-22-20

Desired output

upgrade_rate
0.33

```
WITH users (user_id, name,
join_date)
AS (VALUES
(1, 'Jon', CAST('2-14-20' AS
date)),
(2, 'Jane', CAST('2-14-20' AS
date)),
(3, 'Jill', CAST('2-15-20' AS
date)),
(4, 'Josh', CAST('2-15-20' AS
date)),
(5, 'Jean', CAST('2-16-20' AS
date)),
(6, 'Justin', CAST('2-17-20' AS
date)),
(7, 'Jeremy', CAST('2-18-20' AS
date))),events (user_id, type,
access_date)
AS (VALUES
(1, 'F1', CAST('3-1-20' AS date)),
(2, 'F2', CAST('3-2-20' AS date)),
(2, 'P', CAST('3-12-20' AS date)),
(3, 'F2', CAST('3-15-20' AS
date)),
(4, 'F2', CAST('3-15-20' AS
date)),
(1, 'P', CAST('3-16-20' AS date)),
(3, 'P', CAST('3-22-20' AS
date))),
```

```
-- get feature 2 users and their
date of feature 2 accesst1 AS (
SELECT
   user_id,
   type,
   access_date AS f2_date
FROM events
WHERE type = 'F2' ),-- get premium
users and their date of premium
upgradet2 AS (
SELECT
   user_id,
   type,
   access_date AS premium_date
FROM events
WHERE type = 'P' ),-- for each
feature 2 user, get time between
joining and premium upgrade (or
null if no upgrade) by inner
joining full users table with
feature 2 users on user ID and
left joining premium users on user
ID, then subtracting premium
upgrade date from join datet3 AS (
SELECT t2.premium_date -
u.join_date AS upgrade_time
FROM users u
JOIN t1
ON u.user_id = t1.user_id
```

```
LEFT JOIN t2
ON u.user_id = t2.user_id )--
calculate fraction of users with
upgrade time less than 30 days as
average of values (ones and zeros)
specified in case statement
condition, rounding to two decimal
places

SELECT
   ROUND(AVG(CASE WHEN
upgrade_time < 30 THEN 1 ELSE 0
END), 2) AS upgrade_rate
FROM t3
```

9. Most friended

Given the following table, return a list of users and their corresponding friend count. Order the result by descending friend count, and in the case of a tie, by ascending user ID. Assume that only unique friendships are displayed (i.e., [1, 2] will not show up again as [2, 1]). From LeetCode (https://leetcode.com/problems/friend-requests-ii-who-has-the-most-friends/).

friends

user1	user2
1	2
1	3
1	4
2	3

Desired output

user_id	friend_count
1	3
2	2
3	2
4	1

```
WITH friends (user1, user2)
AS (VALUES (1, 2), (1, 3), (1, 4),
(2, 3)),
-- compile all user appearances
into one column, preserving
duplicate entries with UNION ALL
t1 AS (
SELECT user1 AS user_id
FROM friends
UNION ALL
SELECT user2 AS user_id
FROM friends)-- grouping by user
ID, count up all appearances of
that userSELECT
   user_id,
   COUNT(*) AS friend_count
FROM t1
GROUP BY 1
ORDER BY 2 DESC
```

10. Project aggregation (hard)

The projects table contains three columns: task_id, start_date, and end_date. The difference between end_date and start_date is 1 day for each row in the table. If task end dates are consecutive they are part of the same project. Projects do not overlap.

Write a query to return the start and end dates of each project, and the number of days it took to complete. Order by ascending project duration, and ascending start date in the case of a tie. From HackerRank (https://www.hackerrank.com/challenges/sql-projects/problem).

projects

task_id	start_date	end_date
1	10-01-2020	10-02-2020
2	10-02-2020	10-03-2020
3	10-03-2020	10-04-2020
4	10-13-2020	10-14-2020
5	10-14-2020	10-15-2020
6	10-28-2020	10-29-2020
7	10-30-2020	10-31-2020

Desired output

start_date	end_date	project_duration
10-28-2020	10-29-2020	1
10-30-2020	10-31-2020	1
10-13-2020	10-15-2020	2
10-01-2020	10-04-2020	3

```
WITH projects (task_id,
start_date, end_date)
AS (VALUES
(1, CAST('10-01-20' AS date),
CAST('10-02-20' AS date)),
(2, CAST('10-02-20' AS date),
CAST('10-03-20' AS date)),
(3, CAST('10-03-20' AS date),
CAST('10-04-20' AS date)),
(4, CAST('10-13-20' AS date),
CAST('10-14-20' AS date)),
(5, CAST('10-14-20' AS date),
CAST('10-15-20' AS date)),
(6, CAST('10-28-20' AS date),
CAST('10-29-20' AS date)),
(7, CAST('10-30-20' AS date),
CAST('10-31-20' AS date))),
-- get start dates not present in
end date column (these are “true”
project start dates) t1 AS (
SELECT start_date
FROM projects
WHERE start_date NOT IN (SELECT
end_date FROM projects) ),-- get
end dates not present in start
date column (these are “true”
project end dates) t2 AS (
SELECT end_date
FROM projects
```

```
WHERE end_date NOT IN (SELECT
start_date FROM projects) ),--
filter to plausible start-end
pairs (start < end), then find
correct end date for each start
date (the minimum end date, since
there are no overlapping
projects)t3 AS (
SELECT
   start_date,
   MIN(end_date) AS end_date
FROM t1, t2
WHERE start_date < end_date
GROUP BY 1 )SELECT
   *,
   end_date - start_date AS
project_duration
FROM t3
ORDER BY 3, 1
```

11. Birthday attendance

Given the following two tables, write a query to return the fraction of students, rounded to two decimal places, who attended school
(attendance = 1) on their birthday.

[Source (http://orhancanceylan.com/facebook/data/science/interview/2020/05/17/sql_facebook_student-attendance.html).](http://orhancanceylan.com/facebook/data/science/interview/2020/05/17/sql_facebook_student-attendance.html)

attendance

student_id	school_date	student_id	attendance
1	4-3-20	1	0
2	4-3-20	2	1
3	4-3-20	3	1
1	4-4-20	1	1
2	4-4-20	2	1
3	4-4-20	3	1
1	4-5-20	1	0
2	4-5-20	2	1
3	4-5-20	3	1
4	4-5-20	4	1

students

student_id	school_id	grade_level	date_of_birth
1	2	5	4-3-12
2	1	4	4-4-13
3	1	3	4-5-14
4	2	4	4-3-13

Desired output

birthday_attendance
0.67

```
WITH attendance (student_id,
school_date, attendance)
AS (VALUES
(1, CAST('2020-04-03' AS date),
0),
(2, CAST('2020-04-03' AS date),
1),
(3, CAST('2020-04-03' AS date),
1),
(1, CAST('2020-04-04' AS date),
1),
(2, CAST('2020-04-04' AS date),
1),
(3, CAST('2020-04-04' AS date),
1),
(1, CAST('2020-04-05' AS date),
0),
(2, CAST('2020-04-05' AS date),
1),
(3, CAST('2020-04-05' AS date),
1),
(4, CAST('2020-04-05' AS date),
1)),students (student_id,
school_id, grade_level,
date_of_birth)
AS (VALUES
(1, 2, 5, CAST('2012-04-03' AS
date)),
(2, 1, 4, CAST('2013-04-04' AS
```

```
date)),
(3, 1, 3, CAST('2014-04-05' AS
date)),
(4, 2, 4, CAST('2013-04-03' AS
date)))
-- join attendance and students
table on student ID, and day and
month of school day = day and
month of birthday, taking average
of attendance column values and
roundingSELECT
ROUND(AVG(attendance), 2) AS
birthday_attendance
FROM attendance a
JOIN students s
ON a.student_id = s.student_id
AND EXTRACT(MONTH FROM
school_date) = EXTRACT(MONTH FROM
date_of_birth)
AND EXTRACT(DAY FROM school_date)
= EXTRACT(DAY FROM date_of_birth)
```

12. Hacker scores

Given the following two tables, write a query to return the hacker ID, name, and total score (the sum of maximum scores for each challenge completed) ordered

by descending score, and by ascending hacker ID in the case of score tie. Do not display entries for hackers with a score of zero. From HackerRank (https://www.hackerrank.com/contests/simply-sql-the-sequel/challenges/full-score).

hackers

hacker_id	name
1	John
2	Jane
3	Joe
4	Jim

submissions

submission_id	hacker_id	challenge_id	score
101	1	1	10
102	1	1	12
103	2	1	11
104	2	1	9
105	2	2	13
106	3	1	9
107	3	2	12
108	3	2	15
109	4	1	0

Desired output

hacker_id	name	total_score
2	Jane	24
3	Joe	24
1	John	12

```
WITH hackers (hacker_id, name)
AS (VALUES
(1, 'John'),
(2, 'Jane'),
(3, 'Joe'),
(4, 'Jim')),submissions
(submission_id, hacker_id,
challenge_id, score)
AS (VALUES
(101, 1, 1, 10),
(102, 1, 1, 12),
(103, 2, 1, 11),
(104, 2, 1, 9),
(105, 2, 2, 13),
(106, 3, 1, 9),
(107, 3, 2, 12),
(108, 3, 2, 15),
(109, 4, 1, 0)),
-- from submissions table, get
maximum score for each hacker-
challenge pairt1 AS (
SELECT
   hacker_id,
   challenge_id,
   MAX(score) AS max_score
FROM submissions
GROUP BY 1, 2 )-- inner join this
with the hackers table, sum up all
maximum scores, filter to exclude
```

```
hackers with total score of zero,
and order result by total score
and hacker IDSELECT
   t1.hacker_id,
   h.name,
   SUM(t1.max_score) AS
total_score
FROM t1
JOIN hackers h
ON t1.hacker_id = h.hacker_id
GROUP BY 1, 2
HAVING SUM(max_score) > 0
ORDER BY 3 DESC, 1
```

13. Rank without RANK (hard)

Write a query to rank scores in the following table without using a window function. If there is a tie between two scores, both should have the same rank. After a tie, the following rank should be the next consecutive integer value. From LeetCode (https://leetcode.com/problems/rank-scores/).

scores

id	score
1	3.50
2	3.65
3	4.00
4	3.85
5	4.00
6	3.65

Desired output:

score	score_rank
4.00	1
4.00	1
3.85	2
3.65	3
3.65	3
3.50	4

```
WITH scores (id, score)
AS (VALUES
(1, 3.50),
(2, 3.65),
(3, 4.00),
(4, 3.85),
(5, 4.00),
(6, 3.65))
-- self-join on inequality
produces a table with one score
and all scores as large as this
joined to it, grouping by first id
and score, and counting up all
unique values of joined scores
yields the equivalent of
DENSE_RANK()   [check join output
to understand]SELECT
   s1.score,
   COUNT(DISTINCT s2.score) AS
score_rank
FROM scores s1
JOIN scores s2
ON s1.score <= s2.score
GROUP BY s1.id, s1.score
ORDER BY 1 DESC
```

14. Cumulative salary sum

The following table holds monthly salary information for several employees. Write a query to get, for each month, the cumulative sum of an employee's salary over a period of 3 months, excluding the most recent month. The result should be ordered by ascending employee ID and month. From LeetCode (https://leetcode.com/problems/find-cumulative-salary-of-an-employee/).

employee

id	pay_month	salary
1	1	20
2	1	20
1	2	30
2	2	30
3	2	40
1	3	40
3	3	60
1	4	60
3	4	70

Desired output

id	pay_month	salary	cumulative_sum
1	1	20	20
1	2	30	50
1	3	40	90
2	1	20	20
3	2	40	40
3	3	60	100

```
WITH employee (id, pay_month,
salary)
AS (VALUES
(1, 1, 20),
(2, 1, 20),
(1, 2, 30),
(2, 2, 30),
(3, 2, 40),
(1, 3, 40),
(3, 3, 60),
(1, 4, 60),
(3, 4, 70)),
-- add column for descending month
rank (latest month = 1) for each
employeet1 AS (
SELECT *,
   RANK() OVER (PARTITION BY id
ORDER BY pay_month DESC) AS
month_rank
FROM employee )-- filter to
exclude latest month and months
5+, create cumulative salary sum
using SUM() as window function,
order by ID and monthSELECT
   id,
   pay_month,
   salary,
   SUM(salary) OVER (PARTITION BY
id ORDER BY month_rank DESC) AS
```

```
cumulative_sum
FROM t1
WHERE month_rank != 1
AND month_rank <= 4
ORDER BY 1, 2
```

15. Team standings

Write a query to return the scores of each team in the teams table after all matches displayed in the matches table. Points are awarded as follows: zero points for a loss, one point for a tie, and three points for a win. The result should include team name and points, and be ordered by decreasing points. In case of a tie, order by alphabetized team name.

teams

team_id	team_name
1	New York
2	Atlanta
3	Chicago
4	Toronto
5	Los Angeles
6	Seattle

matches

match_id	host_team	guest_team	host_goals	guest_goals
1	1	2	3	0
2	2	3	2	4
3	3	4	4	3
4	4	5	1	1
5	5	6	2	1
6	6	1	1	2

Desired output

team_name	total_points
Chicago	6
New York	6
Los Angeles	4
Toronto	1
Atlanta	0
Seattle	0

```
WITH teams (team_id, team_name)
AS (VALUES
(1, 'New York'),
(2, 'Atlanta'),
(3, 'Chicago'),
(4, 'Toronto'),
(5, 'Los Angeles'),
(6, 'Seattle')),matches (match_id,
host_team, guest_team, host_goals,
guest_goals)
AS (VALUES
(1, 1, 2, 3, 0),
(2, 2, 3, 2, 4),
(3, 3, 4, 4, 3),
(4, 4, 5, 1, 1),
(5, 5, 6, 2, 1),
(6, 6, 1, 1, 2)),
-- add host points and guest
points columns to matches table,
using case-when-then to tally up
points for wins, ties, and
lossest1 AS (
SELECT
   *,
   CASE WHEN host_goals >
guest_goals THEN 3
        WHEN host_goals =
guest_goals THEN 1
        ELSE 0 END AS host_points,
```

```
   CASE WHEN host_goals <
guest_goals THEN 3
   WHEN host_goals = guest_goals
THEN 1
   ELSE 0 END AS guest_points
FROM matches )-- join result onto
teams table twice to add up for
each team the points earned as
host team and guest team, then
order as requestedSELECT
   t.team_name,
   a.host_points + b.guest_points
AS total_points
FROM teams t
JOIN t1 a
ON t.team_id = a.host_team
JOIN t1 b
ON t.team_id = b.guest_team
ORDER BY 2 DESC, 1
```

16. Customers who didn't buy a product

From the following table, write a query to display the ID and name of customers who bought products A and B, but didn't

buy product C, ordered by ascending customer ID.

customers

id	name
1	Daniel
2	Diana
3	Elizabeth
4	John

orders

order_id	customer_id	product_name
1	1	A
2	1	B
3	2	A
4	2	B
5	2	C
6	3	A
7	3	A
8	3	B
9	3	D

Desired output

id	name
1	Daniel
3	Elizabeth

```
WITH customers (id, name)
AS (VALUES
(1, 'Daniel'),
(2, 'Diana'),
(3, 'Elizabeth'),
(4, 'John')),orders (order_id,
customer_id, product_name)
AS (VALUES
(1, 1, 'A'),
(2, 1, 'B'),
(3, 2, 'A'),
(4, 2, 'B'),
(5, 2, 'C'),
(6, 3, 'A'),
(7, 3, 'A'),
(8, 3, 'B'),
(9, 3, 'D'))
-- join customers and orders
tables on customer ID, filtering
to those who bought both products
A and B, removing those who bought
product C, returning ID and name
columns ordered by ascending
IDSELECT DISTINCT
   id,
   name
FROM orders o
JOIN customers c
ON o.customer_id = c.id
```

```
WHERE customer_id IN (SELECT
customer_id
                      FROM orders
                      WHERE
product_name = 'A')
AND customer_id IN (SELECT
customer_id
                    FROM orders
                    WHERE
product_name = 'B')
AND customer_id NOT IN (SELECT
customer_id
                        FROM
orders
                        WHERE
product_name = 'C')
ORDER BY 1
```

17. Median latitude (hard)

Write a query to return the median latitude of weather stations from each state in the following table, rounding to the nearest tenth of a degree. Note that there is no MEDIAN() function in SQL!
From HackerRank

(https://www.hackerrank.com/challenges/weather-observation-station-20/problem).

stations

id	city	state	latitude	longitude
1	Asheville	North Carolina	35.6	82.6
2	Burlington	North Carolina	36.1	79.4
3	Chapel Hill	North Carolina	35.9	79.1
4	Davidson	North Carolina	35.5	80.8
5	Elizabeth City	North Carolina	36.3	76.3
6	Fargo	North Dakota	46.9	96.8
7	Grand Forks	North Dakota	47.9	97.0
8	Hettinger	North Dakota	46.0	102.6
9	Inkster	North Dakota	48.2	97.6

Desired output

state	median_latitude
North Carolina	35.9
North Dakota	47.4

```
WITH stations (id, city, state,
latitude, longitude)
AS (VALUES
(1, 'Asheville', 'North Carolina',
35.6, 82.6),
(2, 'Burlington', 'North
Carolina', 36.1, 79.4),
(3, 'Chapel Hill', 'North
Carolina', 35.9, 79.1),
(4, 'Davidson', 'North Carolina',
35.5, 80.8),
(5, 'Elizabeth City', 'North
Carolina', 36.3, 76.3),
(6, 'Fargo', 'North Dakota', 46.9,
96.8),
(7, 'Grand Forks', 'North Dakota',
47.9, 97.0),
(8, 'Hettinger', 'North Dakota',
46.0, 102.6),
(9, 'Inkster', 'North Dakota',
48.2, 97.6)),
-- assign latitude-ordered row
numbers for each state, and get
total row count for each statet1
AS (
SELECT
   *,
   ROW_NUMBER() OVER (PARTITION BY
state ORDER BY latitude ASC) AS
```

```
row_number_state,
            count(*) OVER
(PARTITION BY state) AS row_count
FROM stations )-- filter to middle
row (for odd total row number) or
middle two rows (for even total
row number), then get average
value of those, grouping by
stateSELECT
   state,
   AVG(latitude) AS
median_latitude
FROM t1
WHERE row_number_state >=
1.0*row_count/2
AND row_number_state <=
1.0*row_count/2 + 1
GROUP BY 1
```

18. Maximally-separated cities

From the same table in question 17, write a query to return the furthest-separated pair of cities for each state, and the corresponding distance (in degrees, rounded to 2 decimal places)

between those two cities. From HackerRank (https://www.hackerrank.com/challenges/weather-observation-station-19/problem).

stations

id	city	state	latitude	longitude
1	Asheville	North Carolina	35.6	82.6
2	Burlington	North Carolina	36.1	79.4
3	Chapel Hill	North Carolina	35.9	79.1
4	Davidson	North Carolina	35.5	80.8
5	Elizabeth City	North Carolina	36.3	76.3
6	Fargo	North Dakota	46.9	96.8
7	Grand Forks	North Dakota	47.9	97.0
8	Hettinger	North Dakota	46.0	102.6
9	Inkster	North Dakota	48.2	97.6

Desired output

state	city_1	city_2	distance
North Carolina	Asheville	Elizabeth City	6.34
North Dakota	Grand Forks	Hettinger	5.91

```
WITH stations (id, city, state,
latitude, longitude)
AS (VALUES
(1, 'Asheville', 'North Carolina',
35.6, 82.6),
(2, 'Burlington', 'North
Carolina', 36.1, 79.4),
(3, 'Chapel Hill', 'North
Carolina', 35.9, 79.1),
(4, 'Davidson', 'North Carolina',
35.5, 80.8),
(5, 'Elizabeth City', 'North
Carolina', 36.3, 76.3),
(6, 'Fargo', 'North Dakota', 46.9,
96.8),
(7, 'Grand Forks', 'North Dakota',
47.9, 97.0),
(8, 'Hettinger', 'North Dakota',
46.0, 102.6),
(9, 'Inkster', 'North Dakota',
48.2, 97.6)),
-- self-join on matching states
and city < city (avoids identical
and double-counted city pairs),
pulling state, city pair, and
latitude/longitude coordinates for
each city
t1 AS (
SELECT
   s1.state,
```

```
   s1.city AS city1,
   s2.city AS city2,
   s1.latitude AS city1_lat,
   s1.longitude AS city1_long,
   s2.latitude AS city2_lat,
   s2.longitude AS city2_long
FROM stations s1
JOIN stations s2
ON s1.state = s2.state
AND s1.city < s2.city ),-- add a
column displaying rounded
Euclidean distance t2 AS (
SELECT *,
ROUND(( (city1_lat - city2_lat)^2
+ (city1_long - city2_long)^2 ) ^
0.5, 2) AS distance
FROM t1 ),-- rank each city pair
by descending distance for each
statet3 AS (
SELECT *, RANK() OVER (PARTITION
BY state ORDER BY distance DESC)
AS dist_rank
FROM t2 )-- return the city pair
with maximium separationSELECT
   state,
   city1,
   city2,
   distance
```

```
FROM t3
WHERE dist_rank = 1
```

19. Cycle time

Write a query to return the average cycle time across each month. Cycle time is the time elapsed between one user joining and their invitees joining. Users who joined without an invitation have a zero in the “invited by” column.

users

user_id	join_date	invited_by
1	01-01-20	0
2	01-10-20	1
3	02-05-20	2
4	02-12-20	3
5	02-25-20	2
6	03-01-20	0
7	03-01-20	4
8	03-04-20	7

Desired output

month	avg_cycle_time
1	27.0
2	12.5
3	3.0

```
WITH users (user_id, join_date,
invited_by)
AS (VALUES
(1, CAST('01-01-20' AS date), 0),
(2, CAST('01-10-20' AS date), 1),
(3, CAST('02-05-20' AS date), 2),
(4, CAST('02-12-20' AS date), 3),
(5, CAST('02-25-20' AS date), 2),
(6, CAST('03-01-20' AS date), 0),
(7, CAST('03-01-20' AS date), 4),
(8, CAST('03-04-20' AS date), 7)),
-- self-join on invited by = user
ID, extract join month from
inviter join date, and calculate
cycle time as difference between
join dates of inviter and
inviteet1 AS (
SELECT
   CAST(EXTRACT(MONTH FROM
u2.join_date) AS int) AS month,
   u1.join_date - u2.join_date AS
cycle_time
FROM users u1
JOIN users u2
ON u1.invited_by = u2.user_id )--
group by join month, take average
of cycle times within each
monthSELECT
   month,
```

```
   AVG(cycle_time) AS
cycle_time_month_avg
FROM t1
GROUP BY 1
ORDER BY 1
```

20. Three in a row

The attendance table logs the number of people counted in a crowd each day an event is held. Write a query to return a table showing the date and visitor count of high-attendance periods, defined as three consecutive entries (not necessarily consecutive dates) with more than 100 visitors. From LeetCode (https://leetcode.com/problems/human-traffic-of-stadium/solution/).

attendance

event_date	visitors
01-01-20	10
01-04-20	109
01-05-20	150
01-06-20	99
01-07-20	145
01-08-20	1455
01-11-20	199
01-12-20	188

Desired output

event_date	visitors
01-07-20	145
01-08-20	1455
01-11-20	199
01-12-20	188

```
WITH attendance (event_date,
visitors)
AS (VALUES
(CAST('01-01-20' AS date), 10),
(CAST('01-04-20' AS date), 109),
(CAST('01-05-20' AS date), 150),
(CAST('01-06-20' AS date), 99),
(CAST('01-07-20' AS date), 145),
(CAST('01-08-20' AS date), 1455),
(CAST('01-11-20' AS date), 199),
(CAST('01-12-20' AS date), 188)),
-- add row numbers to identify
consecutive entries, since date
column has some gapst1 AS (
SELECT *,
   ROW_NUMBER() OVER (ORDER BY
event_date) AS day_num
FROM attendance ),-- filter this
to exclude days with > 100
visitorst2 AS (
SELECT *
FROM t1
WHERE visitors > 100 ),-- self-
join (inner) twice on offset = 1
day and offset = 2 dayst3 AS (
SELECT
   a.day_num AS day1,
   b.day_num AS day2,
   c.day_num AS day3
```

```
FROM t2 a
JOIN t2 b
ON a.day_num = b.day_num - 1
JOIN t2 c
ON a.day_num = c.day_num - 2 )--
pull date and visitor count for
consecutive days surfaced in
previous tableSELECT
   event_date,
   visitors
FROM t1
WHERE day_num IN (SELECT day1 FROM
t3)
   OR day_num IN (SELECT day2 FROM
t3)
   OR day_num IN (SELECT day3 FROM
t3)
```

21. Commonly purchased together

Using the following two tables, write a query to return the names and purchase frequency of the top three pairs of products most often bought together. The names of both products should

appear in one column. Source (https://www.careercup.com/question?id=5759072822362112).

orders

order_id	customer_id	product_id
1	1	1
1	1	2
1	1	3
2	2	1
2	2	2
2	2	4
3	1	5

products

id	name
1	A
2	B
3	C
4	D
5	E

Desired output

product_pair	purchase_freq
A B	2
A D	1
B D	1

```
WITH orders (order_id,
customer_id, product_id)
AS (VALUES
(1, 1, 1),
(1, 1, 2),
(1, 1, 3),
(2, 2, 1),
(2, 2, 2),
(2, 2, 4),
(3, 1, 5)),products (id, name)
AS (VALUES
(1, 'A'),
(2, 'B'),
(3, 'C'),
(4, 'D'),
(5, 'E')),
-- get unique product pairs from
same order by self-joining orders
table on order ID and product ID <
product ID (avoids identical and
double-counted product pairs)t1 AS
(
SELECT
   o1.product_id AS prod_1,
   o2.product_id AS prod_2
FROM orders o1
JOIN orders o2
ON o1.order_id = o2.order_id
AND o1.product_id < o2.product_id
```

```
),-- join products table onto this
to get product names, concatenate
to get product pairs in one
columnt2 AS (
SELECT CONCAT(p1.name, ' ',
p2.name) AS product_pair
FROM t1
JOIN products p1
ON t1.prod_1 = p1.id
JOIN products p2
ON t1.prod_2 = p2.id )-- grouping
by product pair, return top 3
entries sorted by purchase
frequencySELECT *,
   COUNT(*) AS purchase_freq
FROM t2
GROUP BY 1
ORDER BY 2 DESC
LIMIT 3
```

22. Average treatment effect (hard)

From the following table summarizing the results of a study, calculate the average treatment effect as well as

upper and lower bounds of the 95% confidence interval. Round these numbers to 3 decimal places.

study

participant_id	assignment	outcome
1	0	0
2	1	1
3	0	1
4	1	0
5	0	1
6	1	1
7	0	0
8	1	1
9	1	1

Desired output

point_estimate	lower_bound	upper_bound
0.300	-0.338	0.988

```
WITH study (participant_id,
assignment, outcome)
AS (VALUES
(1, 0, 0),
(2, 1, 1),
(3, 0, 1),
(4, 1, 0),
(5, 0, 1),
(6, 1, 1),
(7, 0, 0),
(8, 1, 1),
(9, 1, 1)),
-- get average outcomes, standard
deviations, and group sizes for
control and treatment
groupscontrol AS (
SELECT
   AVG(outcome) AS avg_outcome,
   STDDEV(outcome) AS std_dev,
   COUNT(*) AS group_size
FROM study
WHERE assignment = 0 ),treatment
AS (
SELECT
   AVG(outcome) AS avg_outcome,
   STDDEV(outcome) AS std_dev,
   COUNT(*) AS group_size
FROM study
WHERE assignment = 1 ),-- get
```

```
average treatment effect
sizeeffect_size AS (
SELECT t.avg_outcome -
c.avg_outcome AS effect_size
FROM control c, treatment t ),--
construct 95% confidence interval
using z* = 1.96 and magnitude of
individual standard errors [ std
dev / sqrt(sample size)
]conf_interval AS (
SELECT 1.96 * (t.std_dev^2 /
t.group_size
             + c.std_dev^2 /
c.group_size)^0.5 AS conf_int
FROM treatment t, control c
)SELECT round(es.effect_size, 3)
AS point_estimate,
        round(es.effect_size -
ci.conf_int, 3) AS lower_bound,
        round(es.effect_size +
ci.conf_int, 3) AS upper_bound
FROM effect_size es, conf_interval
ci
```

23. Rolling sum salary

The following table shows the monthly salary for an employee for the first nine months in a given year. From this, write a query to return a table that displays, for each month in the first half of the year, the rolling sum of the employee's salary for that month and the following two months, ordered chronologically.

salary

month	salary
1	2000
2	3000
3	5000
4	4000
5	2000
6	1000
7	2000
8	4000
9	5000

Desired output

month	salary_3mos
1	10000
2	12000
3	11000
4	7000
5	5000
6	7000

```
WITH salaries (month, salary)
AS (VALUES
(1, 2000),
(2, 3000),
(3, 5000),
(4, 4000),
(5, 2000),
(6, 1000),
(7, 2000),
(8, 4000),
(9, 5000))
-- self-join to match month n with
months n, n+1, and n+2, then sum
salary across those months, filter
to first half of year, and
sortSELECT
   s1.month,
   SUM(s2.salary) AS salary_3mos
FROM salaries s1
JOIN salaries s2
ON s1.month <= s2.month
AND s1.month > s2.month - 3
GROUP BY 1
HAVING s1.month < 7
ORDER BY 1
```

24. Taxi cancel rate

From the given trips and users tables for a taxi service, write a query to return the cancellation rate in the first two days in October, rounded to two decimal places, for trips not involving banned riders or drivers. From LeetCode (https://leetcode.com/problems/trips-and-users/description/).

trips

trip_id	rider_id	driver_id	status	request_date
1	1	10	completed	2020-10-01
2	2	11	cancelled_by_driver	2020-10-01
3	3	12	completed	2020-10-01
4	4	10	cancelled_by_rider	2020-10-02
5	1	11	completed	2020-10-02
6	2	12	completed	2020-10-02
7	3	11	completed	2020-10-03

users

user_id	banned	type
1	no	rider
2	yes	rider
3	no	rider
4	no	rider
10	no	driver
11	no	driver
12	no	driver

Desired output

request_date	cancel_rate
2020-10-01	0.0
2020-10-02	0.5

```
WITH trips (trip_id, rider_id,
driver_id, status, request_date)
AS (VALUES
(1, 1, 10, 'completed',
CAST('2020-10-01' AS date)),
(2, 2, 11, 'cancelled_by_driver',
CAST('2020-10-01' AS date)),
(3, 3, 12, 'completed',
CAST('2020-10-01' AS date)),
(4, 4, 10, 'cancelled_by_rider',
CAST('2020-10-02' AS date)),
(5, 1, 11, 'completed',
CAST('2020-10-02' AS date)),
(6, 2, 12, 'completed',
CAST('2020-10-02' AS date)),
(7, 3, 11, 'completed',
CAST('2020-10-03' AS date))),users
(user_id, banned, type)
AS (VALUES
(1, 'no', 'rider'),
(2, 'yes', 'rider'),
(3, 'no', 'rider'),
(4, 'no', 'rider'),
(10, 'no', 'driver'),
(11, 'no', 'driver'),
(12, 'no', 'driver'))
-- filter trips table to exclude
banned riders and drivers, then
calculate cancellation rate as 1 -
```

```
fraction of trips completed,
filtering to first two days of the
month
SELECT
   request_date,
   1 - AVG(CASE WHEN status =
'completed' THEN 1 ELSE 0 END) AS
cancel_rate
FROM trips
WHERE rider_id NOT IN (SELECT
user_id
                       FROM users
                       WHERE
banned = 'yes' )
AND driver_id NOT IN (SELECT
user_id
                      FROM users
                      WHERE banned
= 'yes' )
GROUP BY 1
HAVING EXTRACT(DAY FROM
request_date) <= 2
```

25. Retention curve (hard)

From the following user activity table, write a query to return the fraction of users who are retained (show some

activity) a given number of days after joining. By convention, users are considered active on their join day (day 0).

users

user_id	action_date	action
1	01-01-20	Join
1	01-02-20	Access
2	01-02-20	Join
3	01-02-20	Join
1	01-03-20	Access
3	01-03-20	Access
1	01-04-20	Access

Desired output:

day_no	n_total	n_active	retention
0	3	3	1.00
1	3	2	0.67
2	3	1	0.33
3	1	1	1.00

```
WITH users (user_id, action_date,
action)
AS (VALUES
(1, CAST('01-01-20' AS date),
'Join'),
(1, CAST('01-02-20' AS date),
'Access'),
(2, CAST('01-02-20' AS date),
'Join'),
(3, CAST('01-02-20' AS date),
'Join'),
(1, CAST('01-03-20' AS date),
'Access'),
(3, CAST('01-03-20' AS date),
'Access'),
(1, CAST('01-04-20' AS date),
'Access')),
-- get join dates for each
userjoin_dates AS (
SELECT
   user_id,
   action_date AS join_date
FROM users
WHERE action = 'Join' ),-- create
vector containing all dates in
date rangedate_vector AS (
SELECT
CAST(GENERATE_SERIES(MIN(action_da
te), MAX(action_date),
```

```
            '1 day'::interval) AS
date) AS dates
FROM users ),-- cross join to get
all possible user-date
combinationsall_users_dates AS (
SELECT DISTINCT
   user_id,
   d.dates
FROM users
CROSS JOIN date_vector d ),-- left
join users table onto all user-
date combinations on matching user
ID and date (null on days where
user didn't engage), join onto
this each user's signup date,
exclude user-date combinations
falling before user signupt1 AS (
SELECT
   a.dates - c.join_date AS
day_no,
   b.user_id
FROM all_users_dates a
LEFT JOIN users b
ON a.user_id = b.user_id
AND a.dates = b.action_date
JOIN join_dates c
ON a.user_id = c.user_id
WHERE a.dates - c.join_date >= 0
)-- grouping by days since signup,
```

```
count (non-null) user IDs as
active users, total users, and the
quotient as retention rateSELECT
   day_no,
   COUNT(*) AS n_total,
   COUNT(DISTINCT user_id) AS
n_active,
   ROUND(1.0*COUNT(DISTINCT
user_id)/COUNT(*), 2) AS retention
FROM t1
GROUP BY 1
```

Appendix

A note on one common problem: if you see a syntax error when using CTEs, check that you have commas between CTEs and no comma after the last CTE.

```
WITH input_table (column_1,
column_2)
AS (VALUES
(1, 'A'), (2, 'B')),     -- comma
between CTEst1 AS (
SELECT *
FROM input_table
WHERE column_2 = 'A')    -- no
comma after last CTESELECT *
FROM t1
```