25 SQL practice problems with solutions:

https://towardsdatascience.com/twenty-five-sql-practice-exercises-5fc791e24082

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
1.0SELECT user_id, 1.0*publishes/starts AS publish_rate,
1.0*cancels/starts AS cancel_rate
FROM t1
```

1. With sumUsers
   (Select user_id,

Sum(case when action='start' then 1 else 0 end) as sumStart,
Sum(case when action='cancel' then 1 else 0 end) as sumCancel,
Sum(case when action='publish' then 1 else 0 end) as sumPublish
From users
)

Select user_id, sumCancel/sumStart as cancelRate, sumPublish/sumStart as publishRate from sumUsers

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 sender ),credits AS (
SELECT receiver, sum(amount) AS credited
FROM transactions
GROUP BY receiver )-- 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
```

2. select sender,receiver,sum(amount) over(partition by sender) as senderAmount,
sum(amount) over(partition by receiver) as receiverAmount,
(reveiverAmount-senderAmount) as netAmount
from transactions
order by netAmount

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
```

3. select date,item,count(item) as totalCount,dense_rank() over(parition by date order by totalCount desc) as mostFrequent from items
where mostFrequent=1
group by date,item

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 (everyone will
have a latest action, not everyone will have a second latest
action), 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
```

4. with lastAction as
(
select user_id,action_date,lead(action_date,1,null) over(partition by user_id order by action_date desc) as actionDate from users
order by action_date desc
)

with lastDate as
(
select user_id, unix_timestamp("action_date") as latestDate from lastAction
)

with secondLast as
(
select user_id, unix_timestamp("actionDate") as secondDate from lastAction
)

with finalResult(
select user_id,((lastDate-secondLast)/86400) as daysElapse
  from lastdate l
  left join secondLast s where l.user_id=s.user_id

)

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() function,
partitioning by user IDt1 AS (
SELECT *, row_number() OVER (PARTITION by user_id ORDER BY
transaction_date ASC) 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 datet3 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
```

5. with superUser as
(
select user_id,row_number() over(partition by user_id order by transaction_date) as totalTransaction
from users
)

select u.user_id , u.transaction_id from superUser s
Left join users u on u.user_id=s.user_id
where totalTransaction>=2
order by u.transaction_date

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.

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 ASC
```

6. user_table as --to know pages liked by friend
(
select f.user_id,f.friend,l.page_likes
from friends f
left join likes l on l.user_id=f.user_id
)

friend_table as --to know pages liked by the user's friend
(
select l.friend,l.page_likes
from user_table u
inner join likes l on l.user_id=l.friend and u.page_likes=l.page_likes
)
result_table as --to remove pages which are commonly liked by user and their friends based on which recommendation can be made to user
(
select u.user_id,u.page_likes
from user_table u
left join friend_table f on f.user_id=u.friend
where f.user_id is null

)

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),-- count mobile-only users as those
present in mobile but null in web, web-only users similarly, and
users of both as those not null in both mobile and web columns, and
```

```
total n-size with count(*)t2 AS (
SELECT sum(CASE WHEN mobile_user IS NOT NULL AND web_user IS NULL
THEN 1 ELSE 0 END ) AS n_mobile,
        sum(CASE WHEN web_user IS NOT NULL AND mobile_user IS NULL
THEN 1 ELSE 0 END ) AS n_web,
        sum(CASE WHEN web_user IS NOT NULL AND mobile_user IS NOT
NULL THEN 1 ELSE 0 END ) AS n_both,
        count(*) AS n_total
FROM t1 )-- calculate fraction of each, cast as float by
multiplying by 1.0SELECT 1.0*n_mobile/n_total AS mobile_fraction,
        1.0*n_web/n_total AS web_fraction,
        1.0*n_both/n_total AS both_fraction
FROM t2
```

7. a1 as
(
select m.user_id as mobileUser,w.user_id as webUser from mobiles m
full outer join web w on w.user_id=m.user_id
)

t1 as
(
sum(case when a1.mbileUser is not null then 1 else 0) as mobileCount,
sum(case when a1.webUser is not null then 1 else 0) as webCount,
sum(case when a1.mbileUser is not null and a1.webUser is not null then 1
else 0) as bothCount,
  count(*) from a1 as total

)

select t1.mobileCount/t1.total as mobileFraction,
t1.webCount/t1.total as webFraction,
t1.bothCount/t1.total as bothFraction
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 )-- divide the number of users with
upgrade time less than 30 days by the total number of feature 2
users, rounding to two decimals

SELECT round(1.0*sum(CASE WHEN upgrade_time < 30 THEN 1 ELSE 0
END)/count(*), 2) AS upgrade_rate
FROM t3
```

8. with a1 as
(
select u.user_id,distinct count(u.*) as
totalCount,unix_timestamp(u.join_date) as
joinDate,unix_timestamp(e.access_date) as accessDate
from users u
left join events e on e.user_id=u.user_id
where e.type='F2'
)

a2 as
(
select

```
sum(case when ((a1.accessDate-a1.joinDate)<86400*30) then 1 else 0) as
upgradeCount
from a1
)

select a2.upgradeCount/a1.totalCount
```

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.

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, 1 ASC
```

9. a1 as
(
select user1 as user_id, count(user1) as friend_count from friends group by user1

)

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 descending start date in the case of a tie.
From HackerRank.

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 start_date )SELECT *, end_date - start_date AS
project_duration
FROM t3
ORDER BY project_duration ASC, start_date ASC
```

```
10.a1 as
  (
  select unix_timestamp(start_date),unix_timestamp(end_date) from projects
     where start_date not in end_date and
     end_date not in start_date
     order by start_date,end_date
  )

  a2 as
  (
  select a1.start_date,a1.end_start,(a1.end_date-a1.start_date)/86400 as
  projectDays
     from a1
  )
```

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.

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, summing ones in
attendance column, dividing by total number of entries, and
roundingSELECT round(1.0*sum(attendance)/count(*), 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)
```

11.
```
a1 as
(
select a.student_id,date_format(s.date_of_birth,'dd-mm') as birthDate
   from attendance a
   left join students s on s.student_id=a.student_id
   and birthDate=time_format(a.school_date, 'dd-mm')

)
select count(*) as totalCount from students
a2 as
(
  select count(a1.*)/totalCount as fractionValue from a1,students s
   where a1.student_id=s.student_id
)
```

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.

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 hacker_id, challenge_id )-- 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 ASC
```

12.
```
a1 as
(
select hacker_id,challenge_id,max(score) from submissions
   group by hacker_id,challenge_id
)

a2 as
(
select a1.hacker_id,sum(a1.score) as maxScore from a1
   group by a1.hacker_id

)

a3 as
(
select a2.hacker_id,h.name, a1.maxScore from a2
   left join hackers h on h.hackers_id=a2.hackers_id
  group by a2.hacker_id,h.name
  order by maxScore desc,hacker_id
)
```

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.

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
fully]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.

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 )-- create cumulative salary sum using sum() as
window function, filter to exclude latest month and months 5+,
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	2
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 total_points DESC, team_name ASC
```

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 ASC
```

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.

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 state
```

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.

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 cityt1 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 dist
FROM t1 ),-- rank each city pair by descending distance for each
statet3 AS (
```

```
SELECT *, rank() OVER (PARTITION BY state ORDER BY dist DESC) AS
dist_rank
FROM t2 )-- return the city pair with maximium separationSELECT
state, city1, city2, dist
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
ORDER BY 1 )-- 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.

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)),
-- create row numbers to get handle on consecutive days, 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.

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 1.0*sum(outcome)/count(*) AS avg_outcome,
       stddev(outcome) AS std_dev,
       count(*) AS group_size
FROM study
WHERE assignment = 0 ),treatment AS (
SELECT 1.0*sum(outcome)/count(*) 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 ASC
```

24. Taxi cancellation 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.

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	0.50
2020-10-02	0.33

```
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,
rounding as requested and filtering to first two days of the
monthSELECT request_date, round(1 - 1.0*sum(CASE WHEN status =
'completed' THEN 1 ELSE 0 END)/count(*), 2) 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 request_date
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_date), 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
```