In [8]:
```
#SQL CheatSheet
#https://www.stratascratch.com/blog/sql-cheat-sheet-technical-concepts-for-the
```

1. Most Profitable Companies

Forbes Medium ID 10354

Find the 3 most profitable companies in the entire world. Output the result along with the corresponding company name. Sort the result based on profits in descending order.

```
#方法1: with
WITH CTE AS(
SELECT
    company,
    profits,
    RANK() OVER(order by profits desc) as rnk
FROM
    forbes_global_2010_2014
)
SELECT
    company,
    profits
FROM CTE
WHERE rnk <= 3
```

```
#方法2: subquery
SELECT
    company,
    profit
FROM
    (SELECT
        *,
        rank() OVER(ORDER BY totol_profit DESC) as rank
    FROM
        (SELECT
            company,
            sum(profits) AS totol_profit
        FROM forbes_global_2010_2014
        GROUP BY 1) sq) sq2
WHERE rank <=3
```

```
#方法3: with改写的subquery
WITH CTE1 AS(
SELECT
    company,
    sum(profits) AS total_profit
FROM forbes_global_2010_2014
GROUP BY 1
),
CTE2 AS(
SELECT
    company,
```

```
    total_profit,
    RANK() OVER (ORDER BY total_profit DESC) AS rank
FROM CTE1
)
SELECT
    company,
    total_profit
FROM CTE2
WHERE rank <= 3
```

```
#方法4：python

# Explore Dataset
forbes_global_2010_2014.head()
forbes_global_2010_2014.sample(5)
forbes_global_2010_2014.info()

# Import your libraries
import pandas as pd
import numpy as np

# Group& sort columns
result = forbes_global_2010_2014.groupby('company')
['profits'].sum().reset_index().sort_values(by='profits', ascending = False)

# Rank the companies
result['rank'] = result['profits'].rank(method = 'min', ascending = False)

# Filter the dataset
result = result[result['rank']<=3][['company', 'profits']]

#Optimized Solution
forbes_global_2010_2014.sort_values(by = 'profits', ascending = False)
[['company', 'profits']].head(3)
```

2.Workers With The Highest Salaries

Interview Question Date: July 2021

Amazon DoorDash Easy ID 10353

You have been asked to find the job titles of the highest-paid employees.

Your output should include the highest-paid title or multiple titles with the same salary.

DataFrames: worker, titleExpected Output Type: pandas.DataFrame

```
-- --方法1：rank
-- WITH CTE AS(
-- SELECT
--     worker_title,
--     RANK() OVER(ORDER BY salary DESC) AS rnk
-- FROM worker w JOIN title t
-- ON w.worker_id = t.worker_ref_id
```

```
-- )
-- SELECT
--     worker_title
-- FROM CTE
-- WHERE rnk = 1
```

```
--方法2: case when
--先求max_salary; case when salary = max_salary then title; select * where
title is not null

SELECT *
FROM
    (SELECT CASE
        WHEN salary =
            (SELECT max(salary)
            FROM worker) THEN worker_title
    END AS best_paid_title
    FROM worker a
    INNER JOIN title b
    ON b.worker_ref_id = a.worker_id
    ORDER BY best_paid_title
    ) sq
WHERE best_paid_title IS NOT NULL
```

```
--方法3: select title; where salary = (select max(salary) from worker)

SELECT
    worker_title AS best_paid_title
FROM worker
JOIN title
ON work_id = worker_ref_id
WHERE salary = (SELECT MAX(salary) FROM worker)
```

3. Users By Average Session Time

Interview Question Date: July 2021

Meta/Facebook Medium ID 10352

Calculate each user's average session time. A session is defined as the time difference between a page_load and page_exit. For simplicity, assume a user has only 1 session per day and if there are multiple of the same events on that day, consider only the latest page_load and earliest page_exit, with an obvious restriction that load time event should happen before exit time event . Output the user_id and their average session time.

Table: facebook_web_log

```
--date(ts); ts::timestamp: 只取年月日
--timestamp: 2019-04-25 13:30:15
--date(timestamp): 2019-04-25
```

```
--timestamp::date : 2019-04-25
```

```
--方法1: self join
先创建cte表: self join取user_id, date, session(min(t1.time - t2.time));  where
中指定t1.action = 'page_load' t2.action = 'page_exit', t2.time > t1.time;
在表中选择user_id, avg(session)


WITH all_user_sessions AS(
SELECT
    t1.user_id,
           --user_id
    t1.timestamp::date as date,
           --date整数日; t1.timestamp::date = date(t1.timestamp)
    min(t2.timestamp) - max(t1.timestamp) as session_duration
                                                        --session = ealiest
page_exit - latest page_load: min(exit) - max(load)
FROM facebook_web_log t1 JOIN facebook_web_log t2
ON t1.user_id = t2.user_id
WHERE t1.action = 'page_load'
           --t1表为load
    AND t2.action = 'page_exit'
           --t2表为exit
    AND t2.timestamp > t1.timestamp
           --load在exit前: exit > load
GROUP BY 1, 2
)
SELECT user_id, avg(session_duration)
           --user_id, avg(session)
FROM all_user_sessions
GROUP BY user_id
```

```
--方法2:
--case when 求出page_load& page_exit timestamp; 再select 两者相减求
avg_session_time; 最后having is not null去空值

--题目条件:
--avg session time: page_load - page_exit
--latest page_load(max);  ealiest page_exit(min)
--load在exit前: (exit - load)
--user_id, avg session time

WITH min_max as
(
select
    user_id,
    date(timestamp),
    max(CASE
        WHEN action = 'page_load' then timestamp
        END) as pg_load,                                --latest page_load
    min(CASE
        WHEN action = 'page_exit' then timestamp
        END) as pg_exit                                 --ealiest page_exit
FROM facebook_web_log
GROUP BY 1,2
)
```

```
SELECT
    user_id,
    avg(pg_exit - pg_load) as avg_session_time        --avg(exit - load) =
avg_session_time
FROM min_max
GROUP BY 1
HAVING avg(pg_exit - pg_load) is not null            --去掉结尾空值
```

```
-- 方法3.
-- 创建exit表和load表 (user_id, day, exit_time/load_time) ;
-- 根据user_id和day去join两表; 取user_id, avg(exit - load);
-- 每个表在创立时要有day, 代表每天, 否则无法join成功

WITH exit as(
SELECT
    user_id,                                              --user_id
    date(timestamp) as day,                               --day
    min(timestamp) as exit                                --exit
FROM
    facebook_web_log
WHERE
    action = 'page_exit'
GROUP BY
    1,2
),
load as(
SELECT
    user_id,                                             --user_id
    date(timestamp) as day,                              --day

    max(timestamp) as load                               --load
FROM
    facebook_web_log
WHERE
    action = 'page_load'
GROUP BY
    1,2
)
SELECT
    e.user_id,                                           --user_id
    avg(exit - load)                                     --avg(exit - load)
FROM exit e JOIN load l
ON e.user_id = l.user_id
AND e.day = l.day
group by 1
```

```
#python:

# Import your libraries
import pandas as pd
import numpy as np
facebook_web_log.head()

# Extract page_load and page_exit action
loads = facebook_web_log.loc[facebook_web_log['action'] == 'page_load',
['user_id', 'timestamp']]
```

```
exits = facebook_web_log.loc[facebook_web_log['action'] == 'page_exit',
['user_id', 'timestamp']]
#Identify possible sessions of each user
sessions = pd.merge(loads, exits, how = 'inner', on = 'user_id', suffixes =
['_load', '_exit'])
#Filter valid sessions:
#page before page_exit
sessions = sessions[sessions['timestamp_load'] < sessions['timestamp_exit']]
#add a column with the date of a page_load timestamp
sessions['date_load'] = sessions['timestamp_load'].dt.date
#aggregate data by user_id and date, select latest page_load and ealiest
page_exit
sessions = sessions.groupby(['user_id',
'date_load']).agg({'timestamp_load':'max',
'timestamp_exit':'min'}).reset_index()
#calculate the duration of the session
sessions['duration'] = sessions['timestamp_exit'] -
sessions['timestamp_load']
sessions
#aggregate to get avg duration by user
result = sessions.groupby('user_id')['duration'].agg(lambda
x:np.mean(x)).reset_index()
```

4.Activity Rank

Interview Question Date: July 2021

Google Medium ID 10351

Find the email activity rank for each user. Email activity rank is defined by the total number of emails sent. The user with the highest number of emails sent will have a rank of 1, and so on. Output the user, total emails, and their activity rank. Order records by the total emails in descending order. Sort users with the same number of emails in alphabetical order. In your rankings, return a unique value (i.e., a unique rank) even if multiple users have the same number of emails. For tie breaker use alphabetical order of the user usernames.

Table: google_gmail_emails

```
--法1：cte
--total emails：from_user, count(to_user), gorup by 1
--先order by total email再order by 用户字母：row_number() OVER (ORDER BY
total_emails DESC, from_user ASC) as row_number
--最后再order一次

--审题：user, total_email, rank:total_email
--rank total emails desc
--users asc
--unique rank: row_number

--output: from user, total_emails, row_number

WITH CTE AS(
SELECT
    from_user,
```

```
    count(to_user) AS total_emails
FROM google_gmail_emails
GROUP BY 1
)
SELECT
    from_user,
    total_emails,
    row_number() OVER (ORDER BY total_emails DESC, from_user ASC) as
row_number
FROM CTE
ORDER BY
    total_emails DESC,
    from_user
```

```
--法2: count(*)与row_number可以在一个query中实现, group by 1。不用cte

SELECT
    from_user,
    count(*) as total_emails,
    row_number() OVER (order by count(*) desc, from_user asc)
FROM
    google_gmail_emails
GROUP BY
    from_user
ORDER BY
    total_emails DESC,
    from_user
```

5.Algorithm Performance

Interview Question Date: July 2021

Meta/Facebook Hard

Meta/Facebook is developing a search algorithm that will allow users to search through their post history. You have been assigned to evaluate the performance of this algorithm.

We have a table with the user's search term, search result positions, and whether or not the user clicked on the search result.

Write a query that assigns ratings to the searches in the following way: • If the search was not clicked for any term, assign the search with rating=1 • If the search was clicked but the top position of clicked terms was outside the top 3 positions, assign the search a rating=2 • If the search was clicked and the top position of a clicked term was in the top 3 positions, assign the search a rating=3

As a search ID can contain more than one search term, select the highest rating for that search ID. Output the search ID and its highest rating.

Example: The search_id 1 was clicked (clicked = 1) and its position is outside of the top 3 positions (search_results_position = 5), therefore its rating is 2.

Table: fb_search_events

```
--法1:
--rating criteria: cte(case when)
-- 1.search not cliked, rating = 1: clicked = 0
-- 2.clicked, top position outsided the top 3, rating = 2: clicked = 1 and
search_results_position > 3
-- 3.clicked, top position in top 3, rating = 3: clicked = 1 and
search_results_position <= 3

--max rating for each user_id: max(rating)

--output: search_id, max_rating

WITH rating AS(
SELECT
    search_id,
    clicked,
    search_results_position,
CASE
    WHEN clicked = 0 THEN '1'
    WHEN clicked = 1 AND search_results_position > 3 THEN '2'
    WHEN clicked = 1 AND search_results_position <= 3 THEN '3'
    END AS rating
FROM fb_search_events
GROUP BY 1,2,3
)
SELECT
    search_id,
    max(rating) AS max_rating
FROM rating
GROUP BY 1
```

In [ ]:

```
--法2:
SELECT
    search_id,
    MAX(CASE WHEN clicked = 0 THEN 1
        WHEN search_results_position > 3 THEN 2
        ELSE 3 END) as max_rating
FROM fb_search_events
GROUP BY search_id
```

```
--#法3:
-- 建立一个矩阵，search_id, 3个case when filter
-- unnest(array[colum_a, colum_b, colum_c]) 是将3列并为一列，只显示有数字的那列
数。
-- max(rating)

WITH CTE AS(
    SELECT
        search_id,
        unnest(array[one, two, three]) AS rating
    FROM
    (SELECT
        search_id,
        CASE
            WHEN clicked = 0 THEN 1
```

```
            ELSE 0
        END AS one,
        CASE
            WHEN clicked = 1 AND search_results_position > 3 THEN 2
            ELSE 0
        END AS two,
        CASE
            WHEN clicked = 1 AND search_results_position <= 3 THEN 3
            ELSE 0
        END AS three
    FROM fb_search_events
    ) sq
    )
SELECT
    search_id,
    max(rating) as max_rating
FROM cte
GROUP BY 1
```

6.Distances Traveled

Interview Question Date: December 2020

Lyft Medium ID 10324

Find the top 10 users that have traveled the greatest distance. Output their id, name and a total distance traveled.

Tables: lyft_rides_log, lyft_users

```
--output: user_id, name, traveled_distance: total_distance_travled
--top 10; greatest distance

--方法1：cte
WITH CTE AS(
SELECT
    user_id,
    name,
    sum(distance) as traveled_distance,
    RANK() OVER (ORDER BY sum(distance) DESC) AS rnk
FROM
    lyft_rides_log l
JOIN
    lyft_users u
ON
    l.user_id = u.id
GROUP BY
    1, 2
)
SELECT
    user_id,
    name,
    traveled_distance
FROM
    CTE
```

```
WHERE
    rnk <= 10
```

```
--方法2：subquery：from后整体后tab一格，rank后order by另起tab
SELECT
    user_id,
    name,
    traveled_distance
FROM
    (SELECT
        lr.user_id,
        lu.name,
        SUM(lr.distance) AS traveled_distance,
        rank() OVER(
                    ORDER BY SUM(lr.distance) DESC) AS rank
    FROM lyft_users lu
    INNER JOIN lyft_rides_log lr ON lu.id = lr.user_id
    GROUP BY
        lr.user_id,
        lu.name
    ORDER BY
        traveled_distance DESC
    ) sq
WHERE rank <= 10
```

```
--方法3：
--rank() OVER(ODER BY traveled_distance DESC), rank<=10
--可被ORDER BY traveled_distance, limit 10替代

SELECT
    user_id,
    name,
    sum(distance) as traveled_distance
FROM
    lyft_rides_log l
JOIN
    lyft_users u
ON
    l.user_id = u.id
GROUP BY
    1, 2
ORDER BY
    traveled_distance DESC
LIMIT 10
```

7.Finding User Purchases【有个video可看】

Interview Question Date: December 2020

Amazon Medium ID 10322

Write a query that'll identify returning active users. A returning active user is a user that has made a second purchase within 7 days of any other of their purchases. Output a list of user_ids of these returning active users.

Table: amazon_transactions

```
#①替换写法
b.created_at - a.created_at BETWEEN 0 AND 7
b.created_at - a.created_at <=7
ABS(DATEDIFF(a.created_at, b.created_at)) <= 7

a.created_at <= b.created_at
b.created_at >= a.created_at

#②self join 要去重自身，需a.id != b.id

#③lead lag后，可用where next - current >= a 来filter out:
lead(X, 1) over(partition by Y order by X asc) as next【注意是asc】
lag(X, 1) over(partition by Y order by X ASC) as previous【注意也是asc】

#④2nd purchase within 7 days of any other purchase，包含第二单和第一单在同一天
b.created_at >= a.created_at
```

```
--return active user: 2nd purchase within 7 days of any other pur:
--包含第二次下单和第一次下单是同一天的情况，∴ b.created_at >= a.created_at
--output:user_id

--法1：self join；on user_id=, id<>, where b.created - a.created <= 7 and
a.created_at <=b.created

SELECT
distinct
a.user_id
FROM amazon_transactions a JOIN amazon_transactions b
ON a.user_id = b.user_id
AND a.id <> b.id                                          --排除同一用户的相同
购买记录进行比较:只比较不同的购买记录
WHERE
b.created_at - a.created_at BETWEEN 0 AND 7            --可以替换为
b.created_at - a.created_at <=7 ；也可以替换为ABS(DATEDIFF(a.created_at,
b.created_at)) <= 7
--DATEDIFF(a.created_at, b.created_at) <= 7               --为什么写这个不
对?
AND a.created_at <= b.created_at                          --【可以替换为
b.created_at >= a.created_at ；但必须包括=】
```

```
--法2：self join；on user_id；where .created_at - b.created_at BETWEEN 0 AND
7 AND a.id != b.id

SELECT
    DISTINCT(a.user_id)
FROM amazon_transactions a
JOIN amazon_transactions b
ON a.user_id = b.user_id
WHERE a.created_at - b.created_at BETWEEN 0 AND 7
    AND a.id != b.id
```

```
--法3：Lead建立新的一列，指前面变量的后一个值，再用where filter 相减 <=7即可
--lead(X, 1) over(partition by Y order by X asc) as next【注意是asc】
```

```
WITH next_transaction AS(
SELECT
    user_id,
    created_at,
    LEAD(created_at, 1) OVER(PARTITION BY user_id ORDER BY created_at ASC) AS
next_transaction
FROM amazon_transactions
)
SELECT
    DISTINCT user_id
FROM next_transaction
WHERE next_transaction - created_at <= 7

--LEAD(created_at, 1)表示获取在当前行之后的下一个行的created_at值。参数1表示获取
下一个行（偏移量为1）的created_at值。
```

```
--法4：lag建立新的一列，指前面变量的前一个值，再用where filter 相减 <=7即可
--lag(X, 1) over(partition by Y order by X ASC) as previous【注意也是asc】

WITH previous_transaction AS(
SELECT
    user_id,
    created_at,
    LAG(created_at, 1) OVER(PARTITION BY user_id ORDER BY created_at ASC) AS
previous_transaction
FROM amazon_transactions
)
SELECT
  DISTINCT user_id
FROM previous_transaction
WHERE created_at - previous_transaction <= 7
```

8.Monthly Percentage Difference

Interview Question Date: December 2020

Amazon Hard ID 10319

Given a table of purchases by date, calculate the month-over-month percentage change in revenue. The output should include the year-month date (YYYY-MM) and percentage change, rounded to the 2nd decimal point, and sorted from the beginning of the year to the end of the year. The percentage change column will be populated from the 2nd month forward and can be calculated as ((this month's revenue - last month's revenue) / last month's revenue)*100.

Table: sf_transactions

```
1. Date: timestamp转换为YYYY-MM：
②PostgreSQL：to_char(x, 'YYYY-MM')；to_char(date_trunc('month', x), 'YYYY-
MM')
to_char(created_at::date, 'YYYY-MM')
SUBSTRING(created_at::text, 1, 7)
LEFT(created_at::text, 7)
to_char(DATE_TRUNC('month', created_at), 'YYYY-MM')
```

①other SQL:切(left, substring); date_format(date_trunc('month'), x), '%Y-%m'

```
SUBSTRING(created_at, 1, 7)
LEFT(created_at, 7)
DATE_FORMAT(created_at, '%Y-%m')
DATE_FORMAT(DATE_TRUNC('MONTH', created_at), '%Y-%m')
```

2. :: 将 created_at 字段从其原始数据类型转换为其他类型，以便进行后续的操作。
3. 使用 DATE_TRUNC 函数将日期字段截断到月份级别。然后，使用 TO_CHAR 函数将截断后的日期字段格式化为 "YYYY-MM" 的形式。

4. 除法后保留两位小数：round(a/b * 100, 2)

5. 将lag中over后的日期单拿出来在最后写w，可简写lag(sum(value), 1) over (w)

```
--month by month pct change in rev
--output: date:YYYY-MM; % pct change 2nd decimal: (this month rev - last
month rev) / last month rev * 100

--法1：cte
With cte AS(
SELECT
    to_char(created_at::date, 'YYYY-MM') as year_month,
    sum(value) as revenue
FROM sf_transactions
GROUP BY 1
ORDER BY 1
)
SELECT
    year_month,
    -- revenue,
    -- LAG(revenue, 1) OVER(ORDER BY year_month) as last_month_revenue,
    round((revenue - LAG(revenue, 1) OVER(ORDER BY year_month))/ LAG(revenue,
1) OVER(ORDER BY year_month) * 100, 2) as revenue_diff_pct
FROM CTE

#可互相替代(substring, left, to_char, to_char(date_trunc)：
SUBSTRING(created_at::text, 1, 7) as year_month
LEFT(created_at::text, 7) as year_month
to_char(created_at::date, 'YYYY-MM') as year_month
to_char(DATE_TRUNC('month', created_at), 'YYYY-MM') as year_month
```

```
--法2.0：大牛写法：windows alias放在最后以保证code易读：
SELECT
    to_char(created_at::date, 'YYYY-MM') AS year_month,         --日期
    ROUND((sum(value) - lag(sum(value), 1) OVER (w)) * 100 /    --分子一行
    lag(sum(value), 1) OVER (w),2) as revenue_diff              --分母一行
FROM sf_transactions
GROUP BY 1
Window w as (ORDER BY to_char(created_at::date, 'YYYY-MM'))     --over后较长
的部分写成windows alias放在code最后
```

```
法2.1：将lag中over后的日期单拿出来在最后写w
SELECT
    to_char(created_at::date, 'YYYY-MM') as year_month,
```

```
    round(
        (
        sum(value) - lag(sum(value),1) over (w)
        )
        / lag(sum(value), 1) over (w) * 100
    ,2
    ) as revenue_diff
FROM sf_transactions
GROUP BY year_month
window w as (order by to_char(created_at::date, 'YYYY-MM'))
```

```
法2.2：直接将order by写全，不用写w
SELECT
    to_char(created_at::date, 'YYYY-MM') as year_month,
    round(
        (
        sum(value) - lag(sum(value),1) over (order by
to_char(created_at::date, 'YYYY-MM'))
        )
        / lag(sum(value), 1) over (order by to_char(created_at::date, 'YYYY-
MM')) * 100
    ,2
    ) as revenue_diff
FROM sf_transactions
GROUP BY year_month

--lag(sum(value),1) over (w) 表示对于每一行，它会获取与当前行相同窗口规范 w 中的
前一行的 sum(value) 值。
--窗口规范 w 被定义为 window w as (order by to_char(created_at::date, 'YYYY-
MM'))。
--它指定了按照 created_at 字段的日期部分进行排序，并且窗口函数将在该排序后的顺序中
进行计算。
```

9.New Products

Interview Question Date: December 2020

Salesforce Tesla Medium ID 10318

You are given a table of product launches by company by year. Write a query to count the net difference between the number of products companies launched in 2020 with the number of products companies launched in the previous year. Output the name of the companies and a net difference of net products released for 2020 compared to the previous year.

```
CASE WHEN:
①COUNT和SUM在CASE WHEN中互换：
COUNT(CASE WHEN year = 2019 THEN product_name END) AS prev_counts
SUM(CASE WHEN year = 2019 THEN 1 ELSE 0 END) AS prev_counts

②CASE WHEN中else可以省略，end不可省略
count(case when year = 2020 then 1 else null end) 等同于 count(case when year
= 2020 then product_name end)

③count/max(case when X then Y end) as Z: count/max/sum等放在case when括号外
```

case when不需要group by，count等aggregation function需要group by

④起名如果要起数字开头的名字，要加双引号"2019_counts"，包括运算中也要加，∴最好不要数字开头

⑤未免重复计算，count时看是否要加distinct：count(distinct a.brand_2020)

```
net diff # product 2020 vs 2019
company_name, net_products: 20 - 19: count(product_name) in 20

法1:
WITH CTE AS(
SELECT
    company_name,
    count(case when year = 2019 then product_name end) as prev_counts,
    count(case when year = 2020 then product_name end) as current_counts
FROM
    car_launches
GROUP BY 1
)
SELECT
    company_name,
    current_counts - prev_counts as net_products
FROM CTE
```

```
法2：直接减
SELECT
company_name,
--count(case when year = 2020 then 1 else null end) - count(case when year =
2019 then 1 else null end) as net_diff
count(case when year = 2020 then product_name end) - count(case when year =
2019 then product_name end) as net_diff
FROM car_launches
GROUP BY 1

count(case when year = 2020 then 1 else null end) 等同于 count(case when year
= 2020 then product_name end)
```

```
--法3：先用where分别选出2019年和20年product_name；再outer join on
company_name;
--最后取company_name, count(20) - count(19) as net_diff +group by

SELECT
    a.company_name,
    (count(distinct a.brand_2020) - count(distinct b.brand_2019)) as
net_products
FROM
    (SELECT
        company_name,
        product_name AS brand_2020
    FROM car_launches
    WHERE YEAR = 2020) a
FULL OUTER JOIN
    (SELECT
        company_name,
        product_name AS brand_2019
```

```
    FROM car_launches
    WHERE YEAR = 2019) b ON a.company_name = b.company_name
GROUP BY a.company_name
ORDER BY a.company_name
```

10. Cities With The Most Expensive Homes

Interview Question Date: December 2020

Zillow Medium ID 10315

Write a query that identifies cities with higher than average home prices when compared to the national average. Output the city names.

Table: zillow_transactions

```
output:city
city: > national avg home price
这个城市区域的平均值，大于国家区域的平均值
```

```
--正确答案：让城市平均值，select city，group by city，having avg(price) > 国家平均值
SELECT DISTINCT city
FROM zillow_transactions
GROUP BY city
HAVING avg(mkt_price) >
    (
    SELECT avg(mkt_price)
    FROM zillow_transactions
    )
```

```
错误答案，因为没有求city的平均值
SELECT DISTINCT city
FROM zillow_transactions
WHERE mkt_price >
    (SELECT avg(mkt_price)
    FROM zillow_transactions
    )
```

11.Revenue Over Time

Interview Question Date: December 2020

Amazon Hard ID 10314

Find the 3-month rolling average of total revenue from purchases given a table with users, their purchase amount, and date purchased. Do not include returns which are represented by negative purchase values. Output the year-month (YYYY-MM) and 3-month rolling average of revenue, sorted from earliest month to latest month.

A 3-month rolling average is defined by calculating the average total revenue from all user purchases for the current month and previous two months. The first two months will not be a true 3-month rolling average since we are not given data from last year. Assume each month

In [ ]:
```
#本月及前两个月的rolling avg:
#3 months rolling avg: avg total rev from all purchases for current and preivo
AVG(t.monthly_revenue) OVER(ORDER BY t.month ROWS BETWEEN 2 PRECEDING AND CURR
```

```
--法1:
--AVG(t.monthly_revenue) OVER (ORDER BY t.month ROWS BETWEEN 2 PRECEDING AND
CURRENT ROW) AS avg_revenue

SELECT
    t.month,
    AVG(t.monthly_revenue) OVER (
                                ORDER BY t.month ROWS BETWEEN 2 PRECEDING AND
CURRENT ROW) AS avg_revenue
FROM
    (SELECT
        to_char(created_at::date, 'YYYY-MM') AS month,
        sum(purchase_amt) AS monthly_revenue
    FROM amazon_purchases
    WHERE purchase_amt > 0
    GROUP BY 1
    ORDER BY 1
    ) t
ORDER BY 1
```

```
法2: i.a表算每个月的rev; ii.b表分别算本月, 上月, 上上月的rev; c表计算每个月有几个
rev; 用b和c表join求: avg_rev = sum(revenue) / count(rev)
最后grorup by month, avg_count

WITH rev AS(
SELECT
    to_char(created_at, 'YYYY-MM') as month,
    sum(purchase_amt) as revenue
FROM amazon_purchases
WHERE purchase_amt >= 1
GROUP BY 1
ORDER BY 1
),
final AS(
SELECT
    month,
    revenue,
    lag(revenue,1) OVER(ORDER BY month) as prev_month,
    lag(revenue,2) OVER(ORDER BY month) as prev_2_month
FROM rev
ORDER BY 1
),
month_count as(
SELECT
    month,
    (count(revenue) + count(prev_month) + count(prev_2_month)) as avg_count
FROM final
GROUP BY 1
```

```
ORDER BY 1
)
SELECT
f.month,
sum(COALESCE(f.revenue,0) + COALESCE(f.prev_month,0) +
COALESCE(f.prev_2_month,0)) / mc.avg_count AS avg_revenue
FROM final f
LEFT JOIN month_count mc
ON f.month = mc.month
GROUP BY f.month, mc.avg_count
ORDER BY 1
```

12.Naive Forecasting

Interview Question Date: December 2020

Uber Hard ID 10313

Some forecasting methods are extremely simple and surprisingly effective. Naïve forecast is one of them; we simply set all forecasts to be the value of the last observation. Our goal is to develop a naïve forecast for a new metric called "distance per dollar" defined as the (distance_to_travel/monetary_cost) in our dataset and measure its accuracy.

To develop this forecast, sum "distance to travel" and "monetary cost" values at a monthly level before calculating "distance per dollar". This value becomes your actual value for the current month. The next step is to populate the forecasted value for each month. This can be achieved simply by getting the previous month's value in a separate column. Now, we have actual and forecasted values. This is your naïve forecast. Let's evaluate our model by calculating an error matrix called root mean squared error (RMSE). RMSE is defined as sqrt(mean(square(actual - forecast)). Report out the RMSE rounded to the 2nd decimal spot.

Table: uber_request_logs

```
square: power(X,2)
mean: avg(Y)
sqrt: sqrt(Z)
2 decimal: round(M::decimal, 2)：如果M不是小数格式的话
sum(distance_to_travel) / sum(monetary_cost) <=>
avg((distance_to_travel/monetary_cost)
```

```
#法1.1:
-- 将预测值定为上一个月的实际值，求预测值和实际值的误差MRSE:
-- "distance per dollar" defined as the (distance_to_travel/monetary_cost)
-- month: sum "distance to travel"  and "monetary cost" values
-- populate the forecasted value for each month. This can be achieved simply
by getting the previous month's value in a separate column
-- calculating an error matrix called root mean squared error (RMSE).
-- sqrt(mean(square(actual - forecast))


WITH CTE AS(
SELECT
    to_char(request_date, 'YYYY-MM') as month,
```

```
    sum(distance_to_travel) as distance_to_travel_sum,
    sum(monetary_cost) as monetary_cost_sum
FROM uber_request_logs
GROUP BY 1
),
CTE2 AS(
SELECT
    month,
    distance_to_travel_sum/ monetary_cost_sum as actual_value,
    lag(distance_to_travel_sum/ monetary_cost_sum, 1) over(order by month) as
forecasted_value
FROM CTE
ORDER BY 1
)
SELECT
    ROUND(SQRT(AVG(POWER((actual_value - forecasted_value),2)))::DECIMAL,2)
AS mrse
FROM CTE2
```

```
#法1.2：先求sum(A)/sum(B)，和先写出来sum(A)，sum(B),下一个cte再相除，效果是一样
的：

WITH monthly_actuals AS(
SELECT
    to_char(request_date, 'YYYY-MM') as month,
    sum(distance_to_travel) / sum(monetary_cost) as actual_value
FROM uber_request_logs
GROUP BY 1
),
forecast AS(
SELECT
    *,
    LAG(actual_value, 1) OVER (ORDER BY month) as forecasted_value
FROM monthly_actuals
)
SELECT
        ROUND(
            SQRT(
                AVG(
                    POWER((actual_value - forecasted_value),2)
                    )
            )::DECIMAL
        ,2) AS mrse
FROM forecast
```

```
#法3：
WITH avg_monthly_dist_per_dollar AS
(SELECT
    to_char(request_date::date, 'YYYY-MM') as request_mnth,
    sum(distance_to_travel) / sum(monetary_cost) AS monthly_dist_per_dollar
FROM uber_request_logs
GROUP BY 1
ORDER BY 1
),
    naive_forecast AS
(
```

```
SELECT
    request_mnth,
    monthly_dist_per_dollar,
    lag(monthly_dist_per_dollar,1) over (
                                        order by request_mnth)
                                        as previous_monthly_dist_per_dollar
FROM avg_monthly_dist_per_dollar
),
    power AS(
SELECT
    request_mnth,
    monthly_dist_per_dollar,
    previous_monthly_dist_per_dollar,
    POWER(previous_monthly_dist_per_dollar - monthly_dist_per_dollar, 2) AS
power
FROM naive_forecast
GROUP BY 1,2,3
ORDER BY 1
)
SELECT round(sqrt(avg(power))::decimal, 2) FROM power
```

13.Class Performance

Interview Question Date: December 2020

Box Medium ID 10310

You are given a table containing assignment scores of students in a class. Write a query that identifies the largest difference in total score of all assignments. Output just the difference in total score (sum of all 3 assignments) between a student with the highest score and a student with the lowest score.

Table: box_scores

```
#法1:
SELECT
max(assignment1 + assignment2 + assignment3) - min(assignment1 + assignment2
+ assignment3) as difference_in_scores
FROM box_scores
```

```
#法2: subquery
SELECT
    max(score) - min(score) AS difference_in_scores
FROM
    (SELECT
        student,
        sum(assignment1 + assignment2 + assignment3) AS score
    FROM box_scores
    GROUP BY 1
    ) t
```

14. Salaries Differences

Interview Question Date: November 2020

LinkedIn Dropbox Easy ID 10308

Write a query that calculates the difference between the highest salaries found in the marketing and engineering departments. Output just the absolute difference in salaries.

Tables: db_employee, db_dept

```
ABS(max(CASE WHEN X THEN Y END) - min(CASE WHEN M THEN N END)) AS diff
SELECT ABS((SELECT A FROM B) - (SELECT C FROM D)) AS diff
WHERE A in ('X', 'Y'): 在A中filter出 X 和 Y

max(case when A then B else end): 不能错写成case when a then max(sth)
max window function最后的end不能丢
```

```
#法1: ABS(max(CASE WHEN X THEN Y END) - min(CASE WHEN M THEN N END))

SELECT
-- max(CASE WHEN department = 'marketing' THEN salary END) AS mrt_max,
-- max(CASE WHEN department = 'engineering' THEN salary END) AS eng_max,
    ABS(max(CASE WHEN department = 'marketing' THEN salary END) - max(CASE
WHEN department = 'engineering' THEN salary END)) AS salary_difference
FROM db_employee a
JOIN db_dept b
ON a.department_id = b.id
```

```
#法2: where 分别filter出mrt和eng的最大值，在最外面用SELECT ABS(A - B) AS diff 去求，最外面两者相减时不需要from。

SELECT ABS(
    (SELECT
        max(salary)
    FROM db_employee emp
    JOIN db_dept dpt
    ON emp.department_id = dpt.id
    WHERE department = 'marketing'
)
-
(
    SELECT
        max(salary)
    FROM db_employee emp
    JOIN db_dept dpt
    ON emp.department_id = dpt.id
    WHERE department = 'engineering'
)
)AS salary_difference
```

```
#法3: 先单出一列mrt和eng的最大值，再让两者中的max - min 就得了非负整数diff，此处没有用abs:
--WHERE A in ('X', 'Y'): 在A中filter出 X 和 Y

WITH CTE AS(
```

```
SELECT
    department_id,
    max(salary) as highest_salary
FROM db_employee e
JOIN db_dept d
ON d.id = e.department_id
WHERE d.department in ('marketing', 'engineering')
GROUP BY 1
)
SELECT
    max(highest_salary) - min(highest_salary) AS salary_diff
FROM CTE
```

15.Risky Projects

Interview Question Date: November 2020

LinkedIn Medium ID 10304

Identify projects that are at risk for going overbudget. A project is considered to be overbudget if the cost of all employees assigned to the project is greater than the budget of the project.

You'll need to prorate the cost of the employees to the duration of the project. For example, if the budget for a project that takes half a year to complete is 10K, then the total half-year salary of all employees assigned to the project should not exceed $10K. Salary is defined on a yearly basis, so be careful how to calculate salaries for the projects that last less or more than one year.

Output a list of projects that are overbudget with their project name, project budget, and prorated total employee expense (rounded to the next dollar amount).

HINT: to make it simpler, consider that all years have 365 days. You don't need to think about the leap years.

Tables: linkedin_projects, linkedin_emp_projects, linkedin_employees

```
1.如果结果只返回0，不返回0后小数，可以使分子或分母变为float，比如365.00，或者
+0.0，即可返回小数点后位数了

2.ROUND(X,2)，CEILING(X)，FLOOR(X)，ceiling和floor没有,2的设置：
ROUND UP（向上取整）：CEILING
CEILING(3.14) 返回 4，CEILING(5.6) 返回 6
ROUND DOWN（向下取整）：FLOOR
FLOOR(3.14) 返回 3，FLOOR(5.6) 返回 5
ROUND（四舍五入）
ROUND(3.14) 返回 3，ROUND(5.6) 返回 6

3.计算天数差值 or 除法：DATE_PART('day', X - Y)，但不能在postgre sql中用
DATE_PART('day', end_date - start_date)
EXTRACT(DAY FROM end_date - start_date) AS duration_days
ROUND(COALESCE(CAST(field1 AS DOUBLE), 0)/field2, 2) FROM TB

4.--SELECT后的部分可以在having中筛选条件使用
```

5.如果不写::decimal 或 ::float可能会导致最后一位不准

```
overbudget: cost of all employees > budget of project

budget: 0.5 yr ~ 10k, then total 0.5 yr salary of all employees < 10k
salary yrly basis
365 days/ yr

output: projects overbudget: name, budget, prorated total employee expense(nt
dollar)
title, budget, prorated_employee_expense

prorated_employee_expense = duration in yr * (sum(salary/ yr))
outbudget: budget - pro  < 0

#法1:  预计花费 = 每天salary * 天数 = 起终时间相减得天数；把该项目所有人的salary加
起来，再÷ 365 得每天salary
WITH prorated_expense AS(
SELECT
    title,
    budget,
    (end_date - start_date) AS duration,
    SUM(salary) / 365 AS salary_per_day,
    (end_date::date - start_date::date) * (SUM(salary) / 365) AS
prorated_employee_expense
FROM linkedin_projects a
JOIN linkedin_emp_projects b
ON a.id = b.project_id
JOIN linkedin_employees c
ON b.emp_id = c.id
GROUP BY 1, 2, 3
ORDER BY 1, 2
)
SELECT
    title,
    budget,
    CEILING(prorated_employee_expense) AS prorated_employee_expense
FROM prorated_expense
WHERE budget < prorated_employee_expense
ORDER BY 1
```

```
#法2: 预计花费 = SUM(年salary * 年数) = SUM(年salary * (END - START)/365 )
ROUNDUP: CEILING(X)
SELECT后的部分可以在having中筛选条件使用
如果不写::decimal 或 ::float就会导致最后一位不准

SELECT
    title,
    budget,
    CEILING(SUM(salary * (end_date - start_date)::decimal / 365 )) AS
prorated_employee_expense
FROM
    linkedin_projects AS p JOIN linkedin_emp_projects AS ep ON p.id =
ep.project_id
                           JOIN linkedin_employees AS e ON ep.emp_id = e.id
GROUP BY
```

```
    title,
    budget
HAVING budget < CEILING(SUM(salary * (end_date - start_date)::float / 365 ))
ORDER BY 1
```

16.Top Percentile Fraud

Interview Question Date: November 2020

Google Netflix Hard ID 10303

ABC Corp is a mid-sized insurer in the US and in the recent past their fraudulent claims have increased significantly for their personal auto insurance portfolio. They have developed a ML based predictive model to identify propensity of fraudulent claims. Now, they assign highly experienced claim adjusters for top 5 percentile of claims identified by the model. Your objective is to identify the top 5 percentile of claims from each state. Your output should be policy number, state, claim cost, and fraud score.

Table: fraud_score

```
top 5 percentile:
NTILE(100) OVER (PARTITION BY STATE ORDER BY fraud_score DESC) AS
percentile; percentile <= 5
```

```
SELECT
    policy_num,
    state,
    claim_cost,
    fraud_score
FROM
    (
    SELECT
        *,
        NTILE(100) OVER (PARTITION BY STATE ORDER BY fraud_score DESC) AS
percentile
    FROM fraud_score
    ) t
WHERE percentile <= 5
```

17. Distance Per Dollar

Interview Question Date: November 2020

Uber Hard ID 10302

You're given a dataset of uber rides with the traveling distance ('distance_to_travel') and cost ('monetary_cost') for each ride. For each date, find the difference between the distance-per-dollar for that date and the average distance-per-dollar for that year-month. Distance-per-dollar is defined as the distance traveled divided by the cost of the ride.

The output should include the year-month (YYYY-MM) and the absolute average difference in distance-per-dollar (Absolute value to be rounded to the 2nd decimal). You should also count both success and failed request_status as the distance and cost values are populated for all ride requests. Also, assume that all dates are unique in the dataset. Order your results by earliest request date first.

```
求每天ratio 与 月均ratio 的运算:
i.所有, ratio, month
ii. date, month, ratio, avg(ratio) over (partition by month) as month_ratio
iii. distinct month, round(abs(ratio - month_ratio),2)
```

```
#法1: 求每天ratio 与 月均ratio 的运算:
-- i.所有, ratio, month
-- ii. date, month, ratio, avg(ratio) over (partition by month) as
month_ratio
-- iii. distinct month, round(abs(ratio - month_ratio),2)

SELECT
    DISTINCT
    b.request_mnth,
    ROUND(ABS(b.dist_to_cost - b.avg_dist_to_cost)::DECIMAL, 2) AS
mean_deviation
    FROM (
    SELECT
        a.request_date,
        a.request_mnth,
        a.dist_to_cost,
        AVG(a.dist_to_cost) OVER (PARTITION BY request_mnth) AS
avg_dist_to_cost
    FROM (
    SELECT
        *,
        (distance_to_travel / monetary_cost) AS dist_to_cost,
        to_char(request_date::date, 'YYYY-MM') AS request_mnth
    FROM uber_request_logs) a
    )b
ORDER BY b.request_mnth
```

```
#法2:
WITH avg_distance_per_dollar AS(
SELECT
    to_char(request_date, 'YYYY-MM') as request_mnth,
    AVG((distance_to_travel) / (monetary_cost)) AS avg_distance_per_dollar
FROM uber_request_logs
GROUP BY 1
),
distance_per_dollar AS(
SELECT
    request_date,
    to_char(request_date, 'YYYY-MM') as request_mnth,
    distance_to_travel / monetary_cost AS distance_per_dollar
FROM uber_request_logs
),
diff AS(
SELECT
```

```
    request_date,
    avg_distance_per_dollar,
    distance_per_dollar,
    ABS(avg_distance_per_dollar - distance_per_dollar) AS difference
FROM distance_per_dollar a
LEFT JOIN avg_distance_per_dollar b
ON a.request_mnth = b.request_mnth
),
final AS(
SELECT
    DISTINCT
    to_char(request_date, 'YYYY-MM') AS year_month,
    ROUND(difference::DECIMAL,2)
FROM diff
)
SELECT * FROM final
ORDER BY 1
```

```
#法3:

SELECT
    request_mnth,
    ROUND(AVG(mean_deviation), 2) AS difference
FROM
    (
    SELECT
        request_mnth,
        ABS(dist_to_cost - monthly_dist_to_cost)::DECIMAL AS mean_deviation
    FROM
    (
        SELECT
            to_char(request_date::date, 'YYYY-MM') AS request_mnth,
            distance_to_travel / monetary_cost AS dist_to_cost,
            SUM(distance_to_travel) OVER (PARTITION BY
to_char(request_date::date, 'YYYY-MM'))
            / SUM(monetary_cost) OVER (PARTITION BY
to_char(request_date::date, 'YYYY-MM')) AS monthly_dist_to_cost
        FROM uber_request_logs
    ) t
    ) t2
GROUP BY 1
ORDER BY 1
```

18. Expensive Projects

Interview Question Date: November 2020

Microsoft Medium ID 10301

Given a list of projects and employees mapped to each project, calculate by the amount of project budget allocated to each employee . The output should include the project title and the project budget rounded to the closest integer. Order your list by projects with the highest budget per employee first.

Tables: ms_projects, ms_emp_projects

```
1.互换:
CEILING(budget / COUNT(emp_id)::DECIMAL)
ROUND(budget / COUNT(emp_id)::FLOAT)::numeric, 0)

2.如果最后一位差一点，试试::float/ ::decimal/ ::numeric
3.ORDER BY 后可直接用alis name
```

```
--budget to employee
--output: project, budget_emp_ratio: round to closest int: ceiling
--order by ratio desc

SELECT
    title AS project,
    CEILING(budget / COUNT(emp_id)::DECIMAL) AS budget_emp_ratio
--    ROUND((budget / COUNT(emp_id)::FLOAT)::numeric, 0) AS budget_emp_ratio
FROM ms_projects a
JOIN ms_emp_projects b
ON a.id = b.project_id
GROUP BY title, budget
ORDER BY budget_emp_ratio DESC
```

19.Premium vs Freemium

Interview Question Date: November 2020

Microsoft Hard ID 10300

Find the total number of downloads for paying and non-paying users by date. Include only records where non-paying customers have more downloads than paying customers. The output should be sorted by earliest date first and contain 3 columns date, non-paying downloads, paying downloads.

Tables: ms_user_dimension, ms_acc_dimension, ms_download_facts

```
1.CASE WHEN 后的END不要忘
2.string: yes, no 需要加''
3.having 放在group by 后，不可以用alis_name
4.case后换行，when接在case后地方，end换行和case对齐
```

```
--total number of downloads for pay and non-pay by date
--include only download(non-pay) > download(pay)

--output: date, non_paying's downloads, paying's downloads:sum(downloads)
--sort by date asc

SELECT
    date,
    SUM(CASE
            WHEN paying_customer = 'no' THEN downloads
        END) AS non_paying,
    SUM(CASE
            WHEN paying_customer = 'yes' THEN downloads
```

```
        END) AS paying
FROM ms_user_dimension mud
JOIN ms_acc_dimension mac
ON mud.acc_id = mac.acc_id
JOIN ms_download_facts mdf
ON mud.user_id = mdf.user_id
GROUP BY date
HAVING SUM(CASE WHEN paying_customer = 'no' THEN downloads END) > SUM(CASE
WHEN paying_customer = 'yes' THEN downloads END)
ORDER BY date
```

20. Finding Updated Records

Interview Question Date: November 2020

Microsoft Easy

We have a table with employees and their salaries, however, some of the records are old and contain outdated salary information. Find the current salary of each employee assuming that salaries increase each year. Output their id, first name, last name, department ID, and current salary. Order your list by employee ID in ascending order.

Table: ms_employee_salary

```
SELECT
    id,
    first_name,
    last_name,
    department_id,
    max(salary) AS max
FROM ms_employee_salary
GROUP BY 1,2,3,4
ORDER BY 1
```

21.Comments Distribution

Interview Question Date: November 2020

Meta/Facebook Hard ID 10297

Write a query to calculate the distribution of comments by the count of users that joined Meta/Facebook between 2018 and 2020, for the month of January 2020.

The output should contain a count of comments and the corresponding number of users that made that number of comments in Jan-2020. For example, you'll be counting how many users made 1 comment, 2 comments, 3 comments, 4 comments, etc in Jan-2020. Your left column in the output will be the number of comments while your right column in the output will be the number of users. Sort the output from the least number of comments to highest.

To add some complexity, there might be a bug where an user post is dated before the user join date. You'll want to remove these posts from the result.

Tables: fb_users, fb_comments

```
①算 count(comments)~ count(users)
i. user_id, count(comments) as cmt_count, group by 1
ii. cmt_count, count(user_id) as user_count, group by 1

②COUNT(*) 与 COUNT(X_id) 在一些时候效果相同可替换

③BETWEEN 'A' AND 'B'::DATE

④考虑是否有等于号=：joined <= created ： 考虑join当天发帖的人

⑤先join两表，列出最细level的变量，再继续整理计算
```

```
--comments by the # users joined between 18 and Jan 20; time:for the month of
January 2020.
--output: comment_cnt：#com ；  user_cnt#users
--a count of comments and # of users join 18-20 that made that # of comments
in Jan 2020
    -- join date of 18 and 20; comment date of jan 20
--eg. #users~1comment, 2 comments, 3 comments, 4, etc.--
--sort #com asc
--bug: user post date < user join date: remove


--法1：cte
WITH comment_cnt AS(
SELECT
    b.user_id,
    COUNT(b.created_at) AS comment_cnt                                  --
COUNT(*)也可以
    FROM fb_users a
    JOIN fb_comments b
    ON a.id = b.user_id
    WHERE (b.created_at BETWEEN '2020-01-01' AND '2020-01-31'::DATE)      --
BETWEEN 'A' AND 'B'::DATE
    AND (a.joined_at BETWEEN '2018-01-01' AND '2020-12-31'::DATE)
    AND joined_at <= created_at                                         --
joined <= created ： 考虑join当天发帖的人
    GROUP BY 1
    ORDER BY 2
),
    user_cnt AS(
    SELECT
        comment_cnt,
        COUNT(user_id) AS user_cnt
    FROM comment_cnt
    GROUP BY 1
    ORDER BY 1
    )
SELECT *
FROM user_cnt
```

```
法2：subquery
subquery写法：写新FROM，在from后(，在上次code句末)，将上次code整体tab
```

```
subquery写法：先join两表，把最细level的变量都挑出来，再count comment，再count
user
SELECT
    comment_cnt,
    COUNT(id) AS user_cnt
FROM (
    SELECT
        t.id,
        COUNT(t.user_id) as comment_cnt
    FROM (
        SELECT
            a.id,
            a.joined_at,
            b.user_id,
            b.created_at
        FROM fb_users a
        JOIN fb_comments b
        ON a.id = b.user_id
        WHERE (b.created_at BETWEEN '2020-01-01' AND '2020-01-31')
        AND (a.joined_at BETWEEN '2018-01-01' AND '2020-12-31')
        AND joined_at <= created_at) t
    GROUP BY t.id ) c
GROUP BY comment_cnt
ORDER BY comment_cnt ASC
```

22.Most Active Users On Messenger

Interview Question Date: November 2020

Meta/Facebook Medium ID 10295

Meta/Facebook Messenger stores the number of messages between users in a table named 'fb_messages'. In this table 'user1' is the sender, 'user2' is the receiver, and 'msg_count' is the number of messages exchanged between them. Find the top 10 most active users on Meta/Facebook Messenger by counting their total number of messages sent and received. Your solution should output usernames and the count of the total messages they sent or received

Table: fb_messages

```
0. union and union all:
此题应该用UNION ALL，
因为如果union user1 and user2 后出现相同记录，但代表的含义不同的需要全部保留
如union后出现两条记录：UserA，5，分别表示sendA发了5条信息和receiverA收了5条信息
union all是全部保留，用union则distinct记录

1理解：msg_count:既是user1发的数字，也是user2收的数字。
2如果想要进行的数值列在名字列之前，依然可以照常计算：user2，sum(msg_count)；
user_name，sum(msg_count)

3union法：
i. 求user1发的数:user1，msg_count
ii.求user2收的数:user2，msg_count
iii.把user1 发的数UNION user2收的数，是每个user收发的数明细
iv.user_name，sum(count) 是每个user收发总数
```

注：可以先sum 再union；也可以先union再sum

4self join法：
i.先求发过&收过消息的所有人名单表：distinct u1 union distinct u2 as user_name
ii.再将原表 a 和新表 b join，on a.user1 = b.user_name OR a.user2 = b.user_name，
得所有人名单收发明细：加新列命名user_name，出：user1，user2，user1和user1，user2，user2

id	date	user1	user2	msg_count	user_name
1	2020-08-02	kpena	scottmartin	2	kpena
1	2020-08-02	kpena	scottmartin	2	scottmartin

iii. 取user_name，sum(msg_count)，group by 1，得所有人，收发总数

5limit和rank互换法：
rank需要多写一个query，但对于数值相同的记录处理稳妥，不会少。partition可以省略。
limit适用于没有数值相同的记录。

```
#messages IN users table:fb_messages
TOP 10 most active user by counting total #messages sent & received
output: usernames, total_msg_count: count of total mess sent & received

法1:
SELECT user_name, SUM(msg_count) AS total_messages
FROM
(
    SELECT user1 AS user_name, msg_count
    FROM fb_messages
    UNION ALL
    SELECT user2 AS user_name, msg_count
    FROM fb_messages
) AS subquery
GROUP BY user_name
ORDER BY total_messages DESC
LIMIT 10
```

```
法2.1: 先SUM再UNION, limit
SELECT *
FROM(
    SELECT
        user1 AS user_name,
        SUM(msg_count) AS s_r_sum
    FROM fb_messages
    GROUP BY user1
    UNION
    SELECT
        user2 AS user_name,
        SUM(msg_count) AS s_r_sum
    FROM fb_messages
    GROUP BY user2
    ) t
ORDER BY 2 DESC
LIMIT 10
```

```
法2.2: 先SUM再UNION, rank
SELECT
    user_name,
```

```
    total_msg_count
    FROM(
    SELECT
        user_name,
        SUM(s_r_sum) AS total_msg_count,
        RANK() OVER (ORDER BY SUM(s_r_sum) DESC) AS rnk
    FROM(
        SELECT
            user1 AS user_name,
            SUM(msg_count) AS s_r_sum
        FROM fb_messages
        GROUP BY user1
        UNION
        SELECT
            user2 AS user_name,
            SUM(msg_count) AS s_r_sum
        FROM fb_messages
        GROUP BY user2
        ) t
    GROUP BY user_name,s_r_sum) t2
    WHERE rnk <= 10
    ORDER BY total_msg_count DESC
```

法3：self join
i.先求发过&收过消息的所有人名单表：distinct u1 union distinct u2 as user_name
ii.再将原表 a 和新表 b join，on a.user1 = b.user_name OR a.user2 = b.user_name，
得所有人名单收发明细：加新列命名user_name，出：user1，user2，user1和user1，user2，user2

```
    id  date    user1   user2   msg_count   user_name
    1   2020-08-02  kpena   scottmartin 2   kpena
    1   2020-08-02  kpena   scottmartin 2   scottmartin
```

iii. 取user_name，sum(msg_count)，group by 1，得所有人，收发总数

```
SELECT
    u.user_name, SUM(m.msg_count) AS total_messages
FROM fb_messages m
JOIN (
    SELECT DISTINCT user1 AS user_name FROM fb_messages
    UNION
    SELECT DISTINCT user2 AS user_name FROM fb_messages
) u ON m.user1 = u.user_name OR m.user2 = u.user_name
GROUP BY u.user_name
ORDER BY total_messages DESC
LIMIT 10
```

```
WITH user_name AS(
    SELECT DISTINCT user1 AS user_name FROM fb_messages
    UNION
    SELECT DISTINCT user2 AS user_name FROM fb_messages
),
    final AS(
    SELECT *
    FROM fb_messages a
    JOIN user_name b
    ON a.user1 = b.user_name OR a.user2 = user_name
```

```
    ORDER BY 1
    )
SELECT
    user_name AS username,
    SUM(msg_count) AS total_msg_count
FROM final
GROUP BY 1
ORDER BY 2 DESC
LIMIT 10
```

23.SMS Confirmations From Users

Interview Question Date: November 2020

Meta/Facebook Medium

Meta/Facebook sends SMS texts when users attempt to 2FA (2-factor authenticate) into the platform to log in. In order to successfully 2FA they must confirm they received the SMS text message. Confirmation texts are only valid on the date they were sent.

Unfortunately, there was an ETL problem with the database where friend requests and invalid confirmation records were inserted into the logs, which are stored in the 'fb_sms_sends' table. These message types should not be in the table.

Fortunately, the 'fb_confirmers' table contains valid confirmation records so you can use this table to identify SMS text messages that were confirmed by the user.

Calculate the percentage of confirmed SMS texts for August 4, 2020. Be aware that there are multiple message types, the ones you're interested in are messages with type equal to 'message'.

Tables: fb_sms_sends, fb_confirmers

```
求部分占整体的比率:
i. 求部分的query:可以多放一些细节level
ii. 求整体的query
iii. 整体 left join 部分 ON 整体1 = 部分1 AND 整体2 = 部分2: 出现以整体为准，部分与之match的表格，可能出现match部分为null的情况
iv. count(部分.id) / count(整体.id) ::float : 得出比率 (此处可以填写两个表的任意变量，结果不变)
```

```
WITH send AS(
    SELECT
        ds,
        phone_number,
        type
    FROM fb_sms_sends
    WHERE type = 'message'
    ),
    confirmed AS(
    SELECT
        date,
        phone_number
```

```
    FROM fb_confirmers
    )
SELECT
    COUNT(b.phone_number) / COUNT(a.phone_number)::float * 100 AS perc
FROM send a
LEFT JOIN confirmed b
ON a.ds = b.date AND a.phone_number = b.phone_number
WHERE a.ds = '08-04-2020'
```

24. Acceptance Rate By Date

Interview Question Date: November 2020

Meta/Facebook Medium ID 10285

What is the overall friend acceptance rate by date? Your output should have the rate of acceptances by the date the request was sent. Order by the earliest date to latest.

Assume that each friend request starts by a user sending (i.e., user_id_sender) a friend request to another user (i.e., user_id_receiver) that's logged in the table with action = 'sent'. If the request is accepted, the table logs action = 'accepted'. If the request is not accepted, no record of action = 'accepted' is logged.

```
--friend acc rate by date
--output: date: request sent date, percentage_acceptance: acc rate
--order by date

--(sent, accepted) / sent

SELECT
a.date,
COUNT(b.user_id_receiver) / COUNT(a.user_id_sender)::float AS acceptance_rate
--b.后写其他b表变量也可；a.后写其他a表变量也可
FROM (
SELECT
    user_id_sender,
    user_id_receiver,
    date,
    action
FROM fb_friend_requests
WHERE action = 'sent') a
LEFT JOIN (
    SELECT
        user_id_sender,
        user_id_receiver,
        date,
        action
    FROM fb_friend_requests
    WHERE action = 'accepted') b
ON b.user_id_sender = a.user_id_sender                          -- ① 所有变量:
send left join acc on sender_id = sender id and receiver_id = receiver_id
AND b.user_id_receiver = a.user_id_receiver
GROUP BY 1                                                      -- ② group by
date, count(acc.receiver_id) / count(send.sender_id) = acceptance rate
```

```
ORDER BY 1 DESC
```

25. Popularity Percentage

Interview Question Date: November 2020

Meta/Facebook Hard ID 10284

Find the popularity percentage for each user on Meta/Facebook. The popularity percentage is defined as the total number of friends the user has divided by the total number of users on the platform, then converted into a percentage by multiplying by 100. Output each user along with their popularity percentage. Order records in ascending order by user id. The 'user1' and 'user2' column are pairs of friends.

Table: facebook_friends

```
①用UNION让第一列包含所有用户
第一列是所有用户名，第二列是他的朋友的总表：
SELECT u1, u2 FROM facebook_friends
UNION
SELECT u2, u1 FROM facebook_friends

②在第一列维度下，第二列在一行内实现 部分与整体 的比率
第一列是用户名，第二列是流行率：他的朋友数 / 平台用户总数
SELECT
user1,
COUNT(user2) / (SELECT COUNT(DISTINCT user1) FROM total)::float * 100
FROM total
user1, count(*) or user1, count(user_2)是在算每个user的朋友数;
distinct user1 from total 是平台所有用户数
```

```
WITH total AS(
SELECT
    DISTINCT
    user1,
    user2
    FROM facebook_friends
    UNION
    SELECT
    user2,
    user1
    FROM facebook_friends
    ORDER BY 1
),
friends AS(
    SELECT
    user1,
    COUNT(user2) / (SELECT COUNT(DISTINCT user1) FROM total)::float * 100 AS
popularity_percent
    FROM total
    GROUP BY 1
    ORDER BY 1
 )
```

```
SELECT * FROM friends
```

```
WITH total AS(
    SELECT
        DISTINCT
        user1 AS user
        FROM facebook_friends
        UNION
        SELECT
        user2 AS user
        FROM facebook_friends
        ORDER BY 1
    ),
    total_count AS(
    SELECT *                                        --求出总数
    FROM total),
    friends AS(
        SELECT
        user1,
        COUNT(user2) AS friend                      --第一列的为user, count后面的
user数
        FROM facebook_friends
        GROUP BY 1
        UNION                                        --UNION
        SELECT
        user2,                                       --第二列的为user, count前面
的user数
        COUNT(user1) AS friend
    FROM facebook_friends
    GROUP BY 1
    ),
    friends_count AS(
    SELECT
        user1,
        SUM(friend) AS friends_count                --求出每个user的friend数
    FROM friends
    GROUP BY 1
    ORDER BY 1
    ),
    final AS(
    SELECT
        user1,
        friends_count / (SELECT COUNT(*) FROM total_count) * 100 AS
popularity_percent          --每个user的friend数/total数: total可以用select *
from total
    FROM friends_count
    )
SELECT * FROM final
```

26.Find the top-ranked songs for the past 20 years.

Spotify Medium ID 10283

Find all the songs that were top-ranked (at first position) at least once in the past 20 years

Table: billboard_top_100_year_end

```
过去N年:date_part要加'', extract不加''
DATE_PART('year', CURRENT_DATE) - year <= N
EXTRACT(year from CURRENT_DATE) - year <= 20
```

```
SELECT
    DISTINCT
    song_name
FROM billboard_top_100_year_end
WHERE DATE_PART('year', CURRENT_DATE) - year <= 20
--WHERE extract(year from current_date) - year <= 20
--WHERE year BETWEEN 2003 AND 2023
AND year_rank = 1
```

27.Find all inspections which are part of an inactive program

City of Los Angeles Easy ID 10277

Find all inspections which are part of an inactive program.

```
WHERE filter 条件:
ilike '%inactive'
= 'INACTIVE'
```

```
SELECT *
FROM los_angeles_restaurant_health_inspections
WHERE program_status = 'INACTIVE'
```

28. Find the total number of available beds per hosts' nationality

Airbnb Medium ID 10187

Find the total number of available beds per hosts' nationality. Output the nationality along with the corresponding total number of available beds. Sort records by the total available beds in descending order.

Tables: airbnb_apartments, airbnb_hosts

```
当两张表中都有看起来需要的变量时，多留意下从哪张表取:
第一张表中的country是room所在country；要host nationality要在第二张表中找

/*
auto_comment_out_words
```

```
*/
```

```
SELECT
    b.nationality,
    SUM(n_beds) AS total_beds_available
    -- country AS nationality, -- 这是room的country, ∴ ×
FROM
    airbnb_apartments a
INNER JOIN
    airbnb_hosts b
ON
    a.host_id = b.host_id
GROUP BY
    nationality
ORDER BY
    total_beds_available DESC
```

29.Order all countries by the year they first participated in the Olympics

ESPN Easy ID 10184

Order all countries by the year they first participated in the Olympics.

Output the National Olympics Committee (NOC) name along with the desired year.

Sort records by the year and the NOC in ascending order.

Table: olympics_athletes_events

```
求AB为1组，每组的第一个值：
RANK 法：RANK() OVER (PARTITION BY noc ORDER BY year ASC) AS rnk
MIN法：A, MIN(B) + group by 1
```

```
/*
year first in Olympic, countries
output: noc, first_time_year
sort year, NOC in asc
*/

-- RANK 法：RANK() OVER (PARTITION BY noc ORDER BY year ASC) AS rnk
WITH rnk AS(
SELECT
    noc,
    year,
    RANK() OVER (PARTITION BY noc ORDER BY year ASC) AS rnk
FROM olympics_athletes_events
)
SELECT
    DISTINCT
    noc,
    year AS first_time_year
FROM rnk
```

```
WHERE rnk = 1
```

```
--MIN法: A, MIN(B) + group by 1 求AB为1组, 每组的第一个值
SELECT
    noc,
    MIN(year) AS first_time_year
FROM olympics_athletes_events
GROUP BY 1
ORDER BY 1,2
```

30.Total Cost Of Orders

Interview Question Date: July 2020

Amazon Etsy Easy ID 10183

Find the total cost of each customer's orders. Output customer's id, first name, and the total order cost. Order records by customer's first name alphabetically.

Tables: customers, orders

```
--total cost of each customer's orders
--output: id: customer's id; first_name, sum:  total order cost
--order by first name asc

SELECT
    c.id,
    c.first_name,
    SUM(o.total_order_cost) AS sum
FROM customers c
JOIN orders o
ON c.id = o.cust_id
GROUP BY c.id,
        c.first_name
ORDER BY c.first_name ASC
```

31.Find the lowest score for each facility in Hollywood Boulevard

Interview Question Date: July 2020

City of Los Angeles City of San Francisco Tripadvisor Medium ID 10180

Find the lowest score per each facility in Hollywood Boulevard. Output the result along with the corresponding facility name. Order the result based on the lowest score in descending order and the facility name in the ascending order.

Table: los_angeles_restaurant_health_inspections

```
前后模糊搜索: ILIKE '%HOLLYWOOD BLVD%'
```

```
--'%hollywood%b%l%v%d%'
```

```
--lowest score per facility in HB
--output: facility_name, min_score
--order lowest score desc, facility name in asc

SELECT
    facility_name,
    MIN(score) AS min_score
FROM los_angeles_restaurant_health_inspections
WHERE facility_address ILIKE '%HOLLYWOOD BLVD%'
GROUP BY facility_name
ORDER BY min_score DESC, facility_name ASC
```

32.Businesses Open On Sunday

Yelp Medium ID 10178

Find the number of businesses that are open on Sundays. Output the slot of operating hours along with the corresponding number of businesses open during those time slots. Order records by total number of businesses opened during those hours in descending order.

Tables: yelp_business_hours, yelp_business

```
SELECT
    DISTINCT
    a.Sunday,
    COUNT(is_open) AS total_business       --COUNT(*)也可以
FROM yelp_business_hours a
JOIN yelp_business b
ON a.business_id = b.business_id
WHERE is_open = 1 AND sunday IS NOT NULL
GROUP BY 1
ORDER BY 2 DESC
```

33.Bikes Last Used

Lyft DoorDash Easy ID 10176

Find the last time each bike was in use. Output both the bike number and the date-timestamp of the bike's last use (i.e., the date-time the bike was returned). Order the results by bikes that were most recently used.

Table: dc_bikeshare_q1_2012

```
--last time each bike in use
--output: bike_number: bike number; last_used: date last use
--order by most recent used

--1.RANK法:
WITH rnk AS(
```

```
SELECT
    bike_number,
    end_time AS last_used,
    RANK() OVER (PARTITION BY bike_number ORDER BY end_time DESC) AS rnk
FROM dc_bikeshare_q1_2012
GROUP BY bike_number, end_time                          --rank()前group by所有变量
)
SELECT
    bike_number,
    last_used
FROM rnk
WHERE rnk = 1
```

```
2.MAX法:
SELECT
     bike_number,
     MAX(end_time) AS last_used
FROM dc_bikeshare_q1_2012
GROUP BY bike_number
ORDER BY 2 DESC
```

34.Days At Number One

Spotify Hard ID 10173

Find the number of days a US track has stayed in the 1st position for both the US and worldwide rankings. Output the track name and the number of days in the 1st position. Order your output alphabetically by track name.

If the region 'US' appears in dataset, it should be included in the worldwide ranking.

Tables: spotify_daily_rankings_2017_us, spotify_worldwide_daily_song_ranking

```
法1：整体left join部分
世界；US；世界left join US: date = date and name = name; filter 世界榜1US榜1交集：a.date is not null and b.date is not null；name, count(*)

法2.1：inner join; count(*)
us inner join world; date = date name = name：既在US也在世界榜上出现的记录
where filter us.position = 1 AND world.position = 1，就是既在US也在世界榜1了，直接COUNT(*)即可

法2.2：计算既在总表也在部分表出现的次数：总表inner join部分表；两个表中选变量完成sum() over (partition by) + case when；name, max()
```

```
-- days a US track 1st for both US and World
-- region'US' appear -> include in world ranking
-- output: trackname, n_days_on_n1_position
-- order name asc

--法1：整体left join部分
```

```
--世界; US; 世界left join US: date = date and name = name; filter 世界榜1US榜1
交集: a.date is not null and b.date is not null; name, count(*)
WITH world AS(
    SELECT
    date,
    position,
    trackname
    FROM spotify_worldwide_daily_song_ranking
    WHERE position = 1
    ),
    us AS(
    SELECT
    date,
    position,
    trackname
    FROM spotify_daily_rankings_2017_us
    WHERE position = 1
    )
SELECT
    a.trackname,
    COUNT(*) AS n_days_on_n1_position
FROM world a LEFT JOIN us b
ON a.date = b.date AND a.trackname = b.trackname
WHERE a.date IS NOT NULL and b.date IS NOT NULL
GROUP BY 1
```

```
法2.1: inner join; count(*)
us inner join world; date = date name = name: 既在US也在世界榜上出现的记录
where filter us.position = 1 AND world.position = 1, 就是既在US也在世界榜1了,
直接COUNT(*)即可

SELECT
    us.trackname,
    COUNT(*) AS n_days_on_n1_position
FROM spotify_daily_rankings_2017_us us
INNER JOIN spotify_worldwide_daily_song_ranking world
ON world.trackname = us.trackname AND world.date = us.date
WHERE us.position = 1 AND world.position = 1
GROUP BY 1
```

```
法2.2: 计算既在总表也在部分表出现的次数: 总表inner join部分表; 两个表中选变量完成
sum() over (partition by) + case when; name, max()
① US榜1: US与world join: date = date and name = name; us.position = 1
② US世界榜1天数: us.name, SUM(case when world.position = 1 then 1 else 0 end)
OVER (PARTITION BY us.name) AS days【COUNT也可以】
此处用us.name, count(*) 不行, 因为也许存在us第一但不是world第一的情况。但本题没有
出现特例。
③ distinct选出歌名和天数: name, max(days)
us INNER JOIN world ; date = date AND name = name:  同一天既在us榜单也在world榜
单出现, US榜1的歌
SUM(CASE WHEN world.position = 1 THEN 1 ELSE 0 END) OVER (PARTITION BY
us.trackname) AS n_days:
对于us榜1的歌来说, 如果也在世界榜1就计算为1次, sum/count有多少次, 就是both在us和世
界榜1的天数。

SELECT
```

```
trackname,
MAX(n_days_on_n1_position) AS n_days_on_n1_position
FROM
    (
    SELECT
    us.trackname,
    SUM(CASE                                                                    --
此处用sum/count均可
            WHEN world.position = 1 THEN 1
            ELSE 0
        END) OVER (PARTITION BY us.trackname) AS n_days_on_n1_position
    FROM spotify_daily_rankings_2017_us us
    INNER JOIN spotify_worldwide_daily_song_ranking world
    ON world.trackname = us.trackname AND world.date = us.date
    WHERE us.position = 1
    ) tmp
GROUP BY trackname
ORDER BY trackname
```

35.Best Selling Item

Interview Question Date: July 2020

Amazon Ebay Best Buy Hard ID 10172

Find the best selling item for each month (no need to separate months by year) where the biggest total invoice was paid. The best selling item is calculated using the formula (unitprice * quantity). Output the description of the item along with the amount paid.

Table: online_retail

```
WITH total AS(
SELECT
to_char(invoicedate, 'YYYY-MM') AS month,
description,
unitprice * quantity AS total_paid
FROM online_retail
GROUP BY to_char(invoicedate, 'YYYY-MM'), description, unitprice, quantity
),
rank AS(
SELECT
    month,
    description,
    total_paid,
    RANK() OVER (PARTITION BY month ORDER BY total_paid DESC) AS rnk
FROM total
)
SELECT
    month,
    description,
    total_paid
FROM rank
```

```
WHERE rnk = 1
```

36.Best Selling Item

Interview Question Date: July 2020

Amazon Ebay Best Buy Hard ID 10172

Find the best selling item for each month (no need to separate months by year) where the biggest total invoice was paid. The best selling item is calculated using the formula (unitprice * quantity). Output the description of the item along with the amount paid.

Table: online_retail

```
1. SUM(unitprice * quantity)与unitprice * quantity的区别:
sum能一个月内购买相同产品的的不同记录加总，不加sum只能算出一条记录
sum时只需要group by sum前的变量，不加sum时，需要group by sum前的变量和sum内的变量

2. 将SUM(P*N)与 RANK放在一个Query里更有效率
```

```
法1:
WITH total AS(
SELECT
EXTRACT(month from invoicedate) AS month,
description,
SUM(unitprice * quantity) AS total_paid
FROM online_retail
GROUP BY month, description
--, unitprice, quantity                        - 不加sum需要Group by所有变量
),
rank AS(
SELECT
    month,
    description,
    total_paid,
    DENSE_RANK() OVER (PARTITION BY month ORDER BY total_paid DESC) AS rnk
FROM total
)
SELECT
    month,
    description,
    total_paid
FROM rank
WHERE rnk = 1
```

```
法2: 将SUM(P*N)与 RANK放在一个Query里更有效率

SELECT
    MONTH,
    description,
    total_paid
FROM
    (SELECT
```

```
        date_part('month', invoicedate) AS MONTH,
        description,
        SUM(unitprice * quantity) AS total_paid,
        RANK() OVER (PARTITION BY date_part('month', invoicedate)
                     ORDER BY SUM(unitprice * quantity) DESC) AS rnk
    FROM online_retail
    GROUP BY MONTH, description ) tmp
WHERE rnk = 1
```

37.Find the genre of the person with the most number of oscar winnings

Netflix Hard ID 10171

Find the genre of the person with the most number of oscar winnings. If there are more than one person with the same number of oscar wins, return the first one in alphabetic order based on their name. Use the names as keys when joining the tables.

Tables: oscar_nominees, nominee_information

In [ ]:
```
求user的count, count最大值记录对应的属性:
i.求每个演员得奖几次，对应的排名：name, count(*), rank() over (order by COUNT(*)
ii.求genre让名字从刚才的query中取rank1：name IN (select nominess from wins where

nominee, COUNT(*) OVER (PARTITION BY nominee)  = nominee, COUNT(*), R + GROUP
```

```
-- person with most # of oscar 's genre
-- >1 person same # of oscar wins, return 1st based on name asc
-- names as keys join table
--output: top_genre

法1:
i.求每个演员得奖几次，对应的排名：name, count(*), rank() over (order by COUNT(*)
DESC);
ii.求genre让名字从刚才的query中取rank1：name IN (select nominess from wins
where rnk = 1)

WITH wins AS(
SELECT
    a.nominee,
    COUNT(*) AS wins,
    RANK() OVER (ORDER BY COUNT(*) DESC) AS rnk
FROM oscar_nominees a
JOIN nominee_information b
ON a.nominee = b.name
WHERE winner = TRUE
GROUP BY 1
)
SELECT DISTINCT top_genre
FROM nominee_information
```

```
WHERE name IN (SELECT nominee FROM wins WHERE rnk = 1)
```

```
法2:
nominee, COUNT(*) OVER (PARTITION BY nominee)  = nominee, COUNT(*), R + GROUP
BY 1

SELECT
    DISTINCT top_genre
FROM nominee_information info
INNER JOIN (
    SELECT
        nominee,
        n_winnings
    FROM (
        SELECT
            nominee,
            COUNT(*) OVER (PARTITION BY nominee) AS n_winnings
        FROM oscar_nominees
        WHERE winner = true
        ) tmp
    WHERE n_winnings = 2
    ORDER BY 2 DESC, 1 ASC
    ) tmp
ON tmp.nominee = info.name
```