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

```
平方：square：power(X,2)
平均值：mean：avg(Y)
开根：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

```
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.

```
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

```
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
```

36.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
```

37.Gender With Most Doctor Appointments

HealthTap Natera Easy ID 10170

Find the gender that has made the most number of doctor appointments. Output the gender along with the corresponding number of appointments.

Table: medical_appointments

```
--gender most # doc app
--output:gender, n_appointments

--MOST: order by desc
```

```
--aggreagte: group by

SELECT
    gender,
    COUNT(appointmentid) AS n_appointments
FROM medical_appointments
GROUP BY gender
ORDER BY n_appointments DESC
LIMIT 1
```

38.Highest Total Miles

Interview Question Date: July 2020

Uber Medium ID 10169

You're given a table of Uber rides that contains the mileage and the purpose for the business expense. You're asked to find business purposes that generate the most miles driven for passengers that use Uber for their business transportation. Find the top 3 business purpose categories by total mileage.

Table: my_uber_drives

```
-- RANK() OVER(ORDER BY X DESC) AS rnk + where rnk <=3;  LIMIT 3  可以互换
```

```
--Uber rides: mileage and the purpose for the biz expense
-- most miles
-- top 3 biz purpose categories by total mileage.
--output: purpose, miles_sum

SELECT
    purpose,
    miles_sum
FROM
    (SELECT
        DISTINCT
        purpose,
        SUM(miles) AS miles_sum,
        RANK() OVER (ORDER BY SUM(miles) DESC) AS rnk
    FROM my_uber_drives
    WHERE category = 'Business'
    GROUP BY purpose
    ORDER BY miles_sum DESC) t
WHERE rnk <= 3
```

39.Number Of Records By Variety

Microsoft Linux Easy ID 10168

Find the total number of records that belong to each variety in the dataset. Output the variety along with the corresponding number of records. Order records by the variety in ascending order.

Table: iris

```
-- # total belong to each variety
--output: variety, n_total_varieties: variety along with the corresponding
number of records
--order records asc
SELECT
    variety,
    COUNT(*) AS n_total_varieties
FROM iris
GROUP BY 1
ORDER BY 2 ASC
```

40.Total Number Of Housing Units

Airbnb Zillow Easy ID 10167

Find the total number of housing units completed for each year. Output the year along with the total number of housings. Order the result by year in ascending order.

Note: Number of housing units in thousands.

Table: housing_units_completed_us

```
-- each year total # housing units completed
--output: year, n_units: total number of housings.
--order by year asc

--Number of housing units in thousands

SELECT
    year,
    SUM(south + west + midwest + northeast) AS n_units
FROM housing_units_completed_us
GROUP BY year
ORDER BY year
```

41.Reviews of Hotel Arena

Airbnb Expedia Easy ID 10166

Find the number of rows for each review score earned by 'Hotel Arena'. Output the hotel name (which should be 'Hotel Arena'), review score along with the corresponding number of rows with that score for the specified hotel.

Table: hotel_reviews

```
--# of rows for each review score ~ Hotel Arena
--output: hotel_name: Hotel Arena; reviewer_score; count
```

```
SELECT
    hotel_name,
    reviewer_score,
    COUNT(*) AS count
FROM hotel_reviews
WHERE hotel_name = 'Hotel Arena'
GROUP BY hotel_name, reviewer_score
```

42.Total AdWords Earnings

Interview Question Date: July 2020

Google Easy ID 10164

Find the total AdWords earnings for each business type. Output the business types along with the total earnings.

Table: google_adwords_earnings

```
--total ADwords earnings for each biz type
--output: business_type,  earnings: total earnings.

SELECT
    business_type,
    SUM(adwords_earnings) AS earnings
FROM google_adwords_earnings
GROUP BY business_type
```

43.Product Transaction Count

Microsoft Nvidia Medium ID 10163

Find the number of transactions that occurred for each product. Output the product name along with the corresponding number of transactions and order records by the product id in ascending order. You can ignore products without transactions.

Tables: excel_sql_inventory_data, excel_sql_transaction_data

```
order by的变量，如果没有在select中出现过，则需要在group by中出现一次
```

```
--# transactions for each product
--output: product_name, count: number of transactions
--order records by the product id in ascending order
--ignore products without transactions.

SELECT
    i.product_name,
    COUNT(*) AS count
FROM excel_sql_transaction_data t
LEFT JOIN excel_sql_inventory_data i
```

```
ON t.product_id = i.product_id
GROUP BY product_name,t.product_id
ORDER BY t.product_id
```

44.Number Of Acquisitions

Crunchbase Easy ID 10162

Find the number of acquisitions that occurred in each quarter of each year. Output the acquired quarter in YYYY-Qq format along with the number of acquisitions and order results by the quarters with the highest number of acquisitions first.

Table: crunchbase_acquisitions

```
--# acquisition occ in each quarter of each year

--output: acquired_quarter: 2013-Q4;  cnt_acq:  the number of acquisitions
--order by cntacq DESCC: quarters with highest # acquisition first

SELECT
    acquired_quarter,
    COUNT(*) AS cnt_acq
FROM crunchbase_acquisitions
GROUP BY acquired_quarter
ORDER BY cnt_acq DESC
```

45.Ranking Hosts By Beds

Interview Question Date: July 2020

Airbnb Medium ID 10161

Rank each host based on the number of beds they have listed. The host with the most beds should be ranked 1 and the host with the least number of beds should be ranked last. Hosts that have the same number of beds should have the same rank but there should be no gaps between ranking values. A host can also own multiple properties. Output the host ID, number of beds, and rank from highest rank to lowest.

```
--have the same number of beds should have the same rank but there should be
no gaps between ranking values.

DENSE_RANK：同一数量的赋值相同的rank数；rank每等之间没有gap
```

```
SELECT
    host_id,
    SUM(n_beds) AS number_of_beds,
    DENSE_RANK() OVER (ORDER BY SUM(n_beds) DESC) AS rnk
FROM airbnb_apartments
```

```
GROUP BY host_id
```

46.Rank guests based on their ages

Airbnb Easy ID 10160

Rank guests based on their ages. Output the guest id along with the corresponding rank. Order records by the age in descending order.

Table: airbnb_guests

```
RANK() OVER 前的变量，不需要group by
```

```
--rank guest ages
--output: guest id, rank
--order records by age desc

SELECT
    guest_id,
    RANK() OVER (ORDER BY age DESC) AS rank
FROM airbnb_guests
```

47.Ranking Most Active Guests

Airbnb Medium ID 10159

Rank guests based on the number of messages they've exchanged with the hosts. Guests with the same number of messages as other guests should have the same rank. Do not skip rankings if the preceding rankings are identical. Output the rank, guest id, and number of total messages they've sent. Order by the highest number of total messages first.

Table: airbnb_contacts

```
rank也可以放在第一列
```

```
--rank guest ; # messages eachanged with hosts
--dense_rank: guests same # messages ~ same rank; not skip rank if preceding
ranks are identical
--output:ranking,   id_guest,   sum_n_messages



SELECT
    DENSE_RANK() OVER(ORDER BY SUM(n_messages) DESC) AS ranking,
    id_guest,
    SUM(n_messages) AS sum_n_messages
FROM
    airbnb_contacts
GROUP BY
```

```
    id_guest
```

48.Number Of Units Per Nationality

Airbnb Medium ID 10156

Find the number of apartments per nationality that are owned by people under 30 years old.

Output the nationality along with the number of apartments.

Sort records by the apartments count in descending order.

Tables: airbnb_hosts, airbnb_units

```
apartment count是count(distinct unit_id)
因为airbnb_hosts表中有很多重复的host_id = 1的记录，所以和airbnb_units JOIN之后，
有很多重复记录，此时不能直接COUNT(*)，要COUNT(DISTINCT unit_id)
```

```
--# apt per nationality owned by people age <30
--output: nationality, apartment_count
--sort by apt count desc

SELECT
    h.nationality,
    COUNT(DISTINCT u.unit_id) AS apartment_count
FROM airbnb_hosts h
JOIN airbnb_units u
ON h.host_id = u.host_id
WHERE h.age < 30
AND unit_type = 'Apartment'
GROUP BY h.nationality
ORDER BY apartment_count DESC
```

49.Find the number of Yelp businesses that sell pizza

Yelp Easy ID 10153

Find the number of Yelp businesses that sell pizza.

Table: yelp_business

```
LOWER(X) LIKE '%A%' = X ILIKE '%A%'

lower(categories) like '%pizza%' = categories ILIKE '%Pizza%'
```

```
SELECT
    COUNT(*) AS count
FROM yelp_business
```

```
WHERE categories ILIKE '%Pizza%'
```

50. Workers With The Highest And Lowest Salaries

Amazon Siemens Medium ID 10152

You have been asked to find the employees with the highest and lowest salary.

Your output should include the employee's ID, salary, and department, as well as a column salary_type that categorizes the output by:

'Highest Salary' represents the highest salary

'Lowest Salary' represents the lowest salary

Tables: worker, title

```
1. case when可以全命名
2. case when也可以只命名最高最低
3. 正反rank法:
i.赋值正反rank: lowest_sal, highest_sal
ii. case when highest_sal = 1 THEN 'A' ELSE 'B': 赋值最高rank为A, 其他rank为B
iii. where highest_sal = 1 OR lowest_sal = 1: 只挑出最高和最低
4. '' AS X 命名 + where select max/min, 分别求出最高和最低UNION ALL

5. case when写法:
CASE
    WHEN salary = (SELECT MAX(salary) FROM worker) THEN 'Highest Salary'
    WHEN salary = (SELECT MIN(salary) FROM worker) THEN 'Lowest Salary'
ELSE 'N/A'
END AS salary_type


6. 两种case方法可以替换:
①
CASE
    WHEN salary = (SELECT max(salary) FROM total) THEN 'Highest Salary'
    WHEN salary = (SELECT min(salary) FROM total) THEN 'Lowest Salary'
ELSE NULL
END

②
CASE
    WHEN salary = (SELECT max(salary) FROM total THEN 'Highest Salary')
ELSE 'Lowest Salary'
END
```

```
--find employees with highest and lowest salary
--salary_type : Highest Salary, Lowest Salary
--output: worker_id, salary, department, salary_type

--法1: 只命名最高最低, where 取出 rnk 在 最高和最低 中的 rnk值(union)
WITH total AS(
```

```
SELECT
    worker_id,
    salary,
    department,
    RANK() OVER (ORDER BY salary DESC) AS rnk
FROM worker
)
SELECT
    worker_id,
    salary,
    department,
    -- CASE
    --     WHEN salary = (SELECT max(salary) FROM total) THEN 'Highest
Salary'
    --     WHEN salary = (SELECT min(salary) FROM total) THEN 'Lowest Salary'
    -- ELSE NULL
    -- END AS salary_type
    CASE
        WHEN salary = (SELECT max(salary) FROM total) THEN 'Highest Salary'
    ELSE 'Lowest Salary'
    END AS salary_type
FROM total
WHERE rnk IN (SELECT min(rnk) FROM total UNION SELECT min(rnk) FROM total)
```

```
--法2：全命名，where只取出<> 'N/A'的记录，即最大和最小的记录
WITH total AS(
SELECT
    worker_id,
    salary,
    department,
    CASE
        WHEN salary = (SELECT MAX(salary) FROM worker) THEN 'Highest Salary'
        WHEN salary = (SELECT MIN(salary) FROM worker) THEN 'Lowest Salary'
        ELSE 'N/A'
        END AS salary_type
FROM worker
)
SELECT * FROM total
WHERE salary_type <> 'N/A'
```

```
--法3：正反rank法：
-- i.赋值正反rank：lowest_sal, highest_sal
-- ii. case when highest_sal = 1 THEN 'A' ELSE 'B'：赋值最高rank为A，其他rank
为B
-- iii. where highest_sal = 1 OR lowest_sal = 1：只挑出最高和最低

WITH CTE AS(
SELECT
    *,
    RANK() OVER (ORDER BY salary) AS lowest_sal,
    RANK() OVER (ORDER BY salary DESC) AS highest_sal
FROM worker
)
SELECT
    worker_id,
    salary,
```

```
    department,
    -- lowest_sal,
    -- highest_sal,
    CASE
        WHEN highest_sal = 1 THEN 'Highest Salary'
        ELSE 'Lowest Salary'
        END AS salary_type
FROM CTE
WHERE highest_sal = 1 OR lowest_sal = 1
```

```
--法4: '' AS X 命名 + where select max/min, 分别求出最高和最低UNION ALL
with highest_salary as(
select
    worker_id,
    salary,
    department,
    'Highest Salary' as salary_type
from worker
where salary = (select max(salary) from worker)
),
lowest_salary as(
select
    worker_id,
    salary,
    department,
    'Lowest Salary' as salary_type
from worker
where salary = (select min(salary) from worker)
)
select * from highest_salary
union all
select * from lowest_salary
```

51.Gender With Generous Reviews

Interview Question Date: June 2020

Airbnb Easy ID 10149

Write a query to find which gender gives a higher average review score when writing reviews as guests. Use the from_type column to identify guest reviews. Output the gender and their average review score.

Tables: airbnb_reviews, airbnb_guests

```
注意: airbnb_reviews中的from_user同时包含guest_id, user_id
①需要先Where挑选出from_type = 'guest', 才能得出guest_id的记录, 再与airbnb_guests去join
②也可以两个表先join一下, 再只where filter出 from_type = 'guest'的记录就可以了
```

```
-- find which gender gives a higher avg review score when as guests
-- output: gender, avg_score
```

```
SELECT
    gender,
    AVG(review_score) AS avg_score
FROM
    (
    SELECT
        from_user,
        review_score
    FROM airbnb_reviews r
    WHERE from_type = 'guest'
    ) t
    JOIN airbnb_guests g
    ON t.from_user = g.guest_id
GROUP BY gender
ORDER BY avg_score DESC
LIMIT 1
```

```
SELECT
    g.gender,
    AVG(review_score) AS avg_score
FROM airbnb_reviews r
JOIN  airbnb_guests g
ON r.from_user = g.guest_id
WHERE r.from_type = 'guest'
GROUP BY g.gender
ORDER BY avg_score DESC
LIMIT 1
```

52.Find the top 5 cities with the most 5 star businesses

Yelp Medium ID 10148

Find the top 5 cities with the most 5-star businesses. Output the city name along with the number of 5-star businesses. In the case of multiple cities having the same number of 5-star businesses, use the ranking function returning the lowest rank in the group and output cities with a rank smaller than or equal to 5.

Table: yelp_business

```
limit 5:只展示5个记录
rank <=5：将rank = 5个以里的所有记录挑出
```

```
-- top 5 cities most 5-star biz
-- DENSE_RANK: rank return lowest rank in the group; rank <=5
-- output: city, count_of_5_stars

--  'In the case of multiple cities having the same number of 5-star
businesses, use the ranking function returning the lowest rank in the group'
-- It means that you should use:

-- method = 'min' if you're using Python and Pandas
-- rank() if you use PostgreSQL (minimum method is default to it)
```

```
-- It's just a way of making it clear what rank function should be used.


SELECT
    city,
    count_of_5_stars
FROM (
SELECT
    city,
    COUNT(*) AS count_of_5_stars,
    RANK() OVER (ORDER BY COUNT(*) DESC) as rnk
FROM yelp_business
WHERE stars = 5
GROUP BY city) t
WHERE rnk <= 5
```

53.Find countries that are in winemag_p1 dataset but not in winemag_p2

Interview Question Date: June 2020

Wine Magazine Medium ID 10147

Find countries that are in winemag_p1 dataset but not in winemag_p2. Output distinct country names. Order records by the country in ascending order.

Tables: winemag_p1, winemag_p2

```
# countries in p1 but not in p2
# output country: distinct country
# order by country asc

SELECT country FROM winemag_p1
WHERE country NOT IN (SELECT country FROM winemag_p2)
ORDER BY country ASC
```

54.Make a pivot table to find the highest payment in each year for each employee

City of San Francisco Hard ID 10145

Make a pivot table to find the highest payment in each year for each employee. Find payment details for 2011, 2012, 2013, and 2014. Output payment details along with the corresponding employee name. Order records by the employee name in ascending order

Table: sf_public_salaries

```
题中提到pivot <=> case when: eg.MAX(case when) + group by
else 0 把其余值赋为0
```

MAX(pay_year) 去找到这年最高的totalpay，因为可能一年有两个total pay

```
-- select * from sf_public_salaries;

-- PV to find each yr, each employee, highest payment
--2011 - 2014
--output: payment, employee name
--employeename, pay_2011, pay_2012, pay_2013, pay_2014
--order name asc

SELECT
    employeename,
    -- year,
    --totalpay,
    MAX(CASE WHEN year = 2011 THEN totalpay ELSE 0 END) AS pay_2011,
    MAX(CASE WHEN year = 2012 THEN totalpay ELSE 0 END) AS pay_2012,
    MAX(CASE WHEN year = 2013 THEN totalpay ELSE 0 END) AS pay_2013,
    MAX(CASE WHEN year = 2014 THEN totalpay ELSE 0 END) AS pay_2014
    -- totalpaybenefits
FROM sf_public_salaries
GROUP BY employeename
ORDER BY employeename
```

55.Average Weight of Medal-Winning Judo

ESPN Medium ID 10144

Find the average weight of medal-winning Judo players of each team with a minimum age of 20 and a maximum age of 30. Consider players at the age of 20 and 30 too. Output the team along with the average player weight.

Table: olympics_athletes_events

```
SELECT
    team,
    AVG(weight) AS average_player_weight
FROM olympics_athletes_events
WHERE sport = 'Judo' AND age BETWEEN 20 AND 30 AND medal IS NOT NULL
GROUP BY team
```

```
SELECT
    team,
    avg(weight) AS average_player_weight
FROM olympics_athletes_events
WHERE sport = 'Judo'
AND medal IS NOT NULL
GROUP BY 1
HAVING min(age) >= 20 and max(age) <= 30
ORDER BY1
```

56.Find players who participated in the Olympics representing more than one team

ESPN Easy ID 10143

Find players who participated in the Olympics representing more than one team. Output the player name, team, games, sport, and the medal.

Table: olympics_athletes_events

```
Find players who participated in the Olympics representing more than one team


ESPN
Easy
ID 10143

Find players who participated in the Olympics representing more than one
team.
Output the player name, team, games, sport, and the medal.

Table: olympics_athletes_events
```

```
X column has A: X ILIKE '%A%'
每次代表多于一个team：在team中，包含/的值：X ILIKE '%/%'
```

```
--players:OLYMPIC, more than 1 team
--output: name, team, games, sport, medal

--X column has A: X ILIKE '%A%'
--每次代表多于一个team：在team中，包含/的值：X ILIKE '%/%'

SELECT
    name,
    team,
    games,
    sport,
    medal
FROM olympics_athletes_events
WHERE team ILIKE '%/%'
```

57.Apple Product Counts

Google Apple Medium ID 10141

Find the number of Apple product users and the number of total users with a device and group the counts by language. Assume Apple products are only MacBook-Pro, iPhone 5s, and iPad-air. Output the language along with the total number of Apple users and users with any device. Order your results based on the number of total users in descending order.

Tables: playbook_events, playbook_users

```
1.ILIKE '%macbook pro%' AND '%iphone 5s%' AND 'ipad air')
2.order by 可以用产生的alias
3.both count distinct user id
```

```
SELECT
    language,
    COUNT(DISTINCT CASE
                        WHEN device IN('macbook pro',
                                       'iphone 5s',
                                       'ipad air') THEN a.user_id
                    END) AS n_apple_users,
    COUNT(DISTINCT a.user_id) AS n_total_users
FROM playbook_events a
JOIN playbook_users b
ON a.user_id = b.user_id
WHERE device IS NOT NULL
GROUP BY language
ORDER BY n_apple_users DESC
```

58. MacBook Pro Events

Google Apple Medium ID 10140

Find how many events happened on MacBook-Pro per company in Argentina from users that do not speak Spanish. Output the company id, language of users, and the number of events performed by users.

Tables: playbook_events, playbook_users

```
Spanish Not English
```

```
--  evens on macbook-pro in argentina from users not speak Spanish
-- output:company_id, language: language of users, n_macbook_pro_events: the
number of events performed by users.


SELECT
    u.company_id,
    u.language,
    COUNT(*) AS n_macbook_pro_events
FROM playbook_events e
JOIN playbook_users u
ON e.user_id = u.user_id
WHERE e.location = 'Argentina'
AND e.device = 'macbook pro'
AND u.language != 'spanish'
GROUP BY u.company_id, u.language
```

59.Number of Speakers By Language

Google Apple Medium ID 10139

Find the number of speakers of each language by country. Output the country, language, and the corresponding number of speakers. Output the result based on the country in ascending order.

Tables: playbook_events, playbook_users

```
--country, language, #speaker
--output: location, language, n_speakers
--order country asc

SELECT
    e.location,
    u.language,
    COUNT(DISTINCT u.user_id)
FROM playbook_events e
JOIN playbook_users u
ON e.user_id = u.user_id
GROUP BY e.location, u.language
ORDER BY e.location
```

60.Even-numbered IDs Hired in June

Amazon Bosch Easy ID 10137

Find employees who started in June and have even-numbered employee IDs.

Table: worker

```
1.奇偶数表达
偶数 even:
num % 2 = 0
mod(num, 2) = 0
奇数 odd:
num % 2 != 0
mod(num, 2) != 0

2.月份表达
EXTRACT(MONTH FROM X): EXTRACT(MONTH FROM joining_date) AS month
DATE_PART('MONTH', X): DATE_PART('MONTH', joining_date) AS month

3.没在变量中出现的，也可以直接where filter出
WHERE EXTRACT(MONTH FROM joining_date) = 6
```

```
--#法1:
SELECT * FROM worker
WHERE mod(worker_id, 2) = 0
AND EXTRACT(MONTH FROM joining_date) = 6
```

```
--#法2:
WITH odd_num_worker AS(
SELECT
```

```
    *,
    EXTRACT(MONTH FROM joining_date) AS month
    --DATE_PART('MONTH', joining_date) AS month
FROM worker
WHERE worker_id % 2 = 0
)
SELECT
    worker_id,
    first_name,
    last_name,
    salary,
    joining_date,
    department
FROM odd_num_worker
WHERE month = 6
```

61.Odd-numbered ID's Hired in February

Amazon Bosch Easy

Find employees who started in February and have odd-numbered employee IDs.

Table: worker

```
SELECT * FROM worker
WHERE mod(worker_id, 2) != 0
AND EXTRACT(MONTH FROM joining_date) = 2
```

62.Spam Posts

Interview Question Date: June 2020

Meta/Facebook Medium ID 10134

Calculate the percentage of spam posts in all viewed posts by day. A post is considered a spam if a string "spam" is inside keywords of the post. Note that the facebook_posts table stores all posts posted by users. The facebook_post_views table is an action table denoting if a user has viewed a post.

Tables: facebook_posts, facebook_post_views

求部分与整体比值:

```
1. COUNT(CASE WHEN X THEN ID END) / COUNT(ID) + JOIN
2. SUM(CASE WHEN X THEN 1 ELSE 0 END) * 100 / COUNT(*) + JOIN
3. 整体 left join 部分; SELECT 部分/整体求比率
```

```
--percentage spam posts in all viewed posts by day
--spam: post = 'spam'
--facebook_posts stores all posts by user; view: vew post
```

```
--output: post_date, spam_share

--#法1:
WITH total AS(
SELECT
    p.post_date,
    COUNT(v.post_id) AS total_viewed_posts,
    COUNT(CASE
            WHEN p.post_keywords ILIKE '%spam%' THEN v.post_id
          END) AS spam_posts
FROM facebook_posts p
JOIN facebook_post_views v
ON p.post_id = v.post_id
GROUP BY post_date
)
SELECT
    post_date,
    spam_posts / total_viewed_posts::float * 100 AS spam_share
FROM total
```

```
--#法2: 部分与整体比值: SUM(CASE WHEN X THEN 1 ELSE 0 END) * 100 / COUNT(*) +
JOIN
SELECT
    post_date,
    SUM(CASE WHEN post_keywords ILIKE '%spam%' THEN 1 ELSE 0 END) * 100 /
COUNT(*)
FROM facebook_posts p
JOIN facebook_post_views v
ON p.post_id = v.post_id
GROUP BY post_date
```

```
--#法3:  整体 left join 部分; SELECT 部分/整体求比率

SELECT
    spam_summary.post_date,
    n_spam / n_posts::FLOAT * 100 AS spam_share
FROM
    (SELECT
    post_date,
    SUM(CASE
            WHEN v.viewer_id IS NOT NULL THEN 1
            ELSE 0
        END) AS n_posts
    FROM facebook_posts p
    JOIN facebook_post_views v
    ON p.post_id = v.post_id
    GROUP BY post_date) posts_summary
LEFT JOIN
    (SELECT
    post_date,
    SUM(CASE
            WHEN v.viewer_id IS NOT NULL THEN 1
            ELSE 0
        END) AS n_spam
    FROM facebook_posts p
    JOIN facebook_post_views v
```

```
    ON p.post_id = v.post_id
    WHERE post_keywords ILIKE '%spam%'
    GROUP BY post_date) spam_summary
ON posts_summary.post_date = spam_summary.post_date
```

63.Requests Acceptance Rate

Airbnb Medium ID 10133

Find the acceptance rate of requests which is defined as the ratio of accepted contacts vs all contacts. Multiply the ratio by 100 to get the rate.

Table: airbnb_contacts

In [ ]:
```
整体/部分比值:
1. COUNT(ts.部分) / COUNT(整体)
2. SUM(CASE WHEN X THEN 1 ELSE 0 END) / COUNT(*)
3. COUNT(CASE WHEN X THEN A) / COUNT(*)
```

```
--#法1: COUNT(ts.部分) / COUNT(整体)
SELECT
    COUNT(ts_accepted_at) / COUNT(ts_contact_at)::FLOAT * 100 AS
acceptance_rate
FROM airbnb_contacts
```

```
--#法2: SUM(CASE WHEN X THEN 1 ELSE 0 END) / COUNT(*)
SELECT
    100 * SUM(CASE
                  WHEN ts_accepted_at IS NOT NULL THEN 1
                  ELSE 0
              END) / COUNT(*) AS acceptance_rate
FROM airbnb_contacts
```

```
--#法3: COUNT(CASE WHEN X THEN A) / COUNT(*)
SELECT
    COUNT(CASE
              WHEN ts_accepted_at IS NOT NULL THEN ts_accepted_at
          END) / COUNT(*)::FLOAT * 100 AS acceptance_rate
FROM airbnb_contacts
```

64. Highest Crime Rate

City of San Francisco Easy ID 10132

Find the number of crime occurrences for each day of the week. Output the day alongside the corresponding crime count.

Table: sf_crime_incidents_2014_01

```
--# crime occurrences for each day of the week
```

```
--day_of_week; n_occurences

SELECT
    day_of_week,
    COUNT(*) AS n_occurences
FROM sf_crime_incidents_2014_01
GROUP BY
    day_of_week
ORDER BY
    n_occurences DESC
```

65. Business Name Lengths

Interview Question Date: June 2020

City of San Francisco Hard ID 10131

Find the number of words in each business name. Avoid counting special symbols as words (e.g. &). Output the business name and its count of words.

Table: sf_restaurant_health_violations

```
1.regexp_replace(business_name, '[^a-zA-Z0-9 ]', '', 'g'):去除除字母以外的字符
将源文本中所有匹配上述正则表达式的字符都移除，从而得到一个只包含字母、数字和空格的新字符串。
'[^a-zA-Z0-9 ]' 是一个正则表达式模式，用于匹配任何不是字母、数字或空格的字符。
'' 是一个空字符串，表示将匹配到的文本部分替换为空，即移除匹配到的字符。
'g' 是一个标志，表示全局匹配，会替换所有匹配到的字符，而不仅仅是第一个匹配项。


2.array_length(regexp_split_to_array(b_name, '\s+'), 1) : 计算词组数量
将会计算 b_name 经过空格字符拆分后的数组的长度，即计算 b_name 中包含的单词数量。
array_length() 是一个函数，它用于获取数组的长度（即数组中元素的数量）
'\s+' 是一个正则表达式模式，表示匹配一个或多个连续的空格字符
```

```
SELECT
    DISTINCT business_name,
    array_length(regexp_split_to_array(b_name, '\s+'), 1) AS word_count
FROM
    (SELECT
        business_name,
        regexp_replace(business_name, '[^a-zA-Z0-9 ]', '', 'g') AS b_name
    FROM sf_restaurant_health_violations) AS sfr
```

66.Find the number of inspections for each risk category by inspection type

Interview Question Date: June 2020

City of San Francisco Medium ID 10130 25

Data Engineer Data Scientist BI Analyst Data Analyst Find the number of inspections that resulted in each risk category per each inspection type. Consider the records with no risk category value belongs to a separate category. Output the result along with the corresponding inspection type and the corresponding total number of inspections per that type. The output should be pivoted, meaning that each risk category + total number should be a separate column. Order the result based on the number of inspections per inspection type in descending order.

```
法1: COUNT(CASE WHEN X THEN id END)
法2: SUM(CASE WHEN X THEN 1 ELSE 0 END)
```

```
#法1: COUNT(CASE WHEN X THEN id END)
SELECT
    inspection_type,
    COUNT(CASE WHEN risk_category IS NULL THEN inspection_id END) AS
no_risk_results,
    COUNT(CASE WHEN risk_category = 'Low Risk' THEN inspection_id END) AS
low_risk_results,
    COUNT(CASE WHEN risk_category = 'Moderate Risk' THEN inspection_id END)
AS medium_risk_results,
    COUNT(CASE WHEN risk_category = 'High Risk' THEN inspection_id END) AS
high_risk_results,
    COUNT(inspection_id) AS total_inspections
FROM sf_restaurant_health_violations
GROUP BY inspection_type
ORDER BY total_inspections DESC
```

```
#法2: SUM(CASE WHEN X THEN 1 ELSE 0 END)
SELECT
inspection_type,

SUM(CASE
        WHEN risk_category IS NULL THEN 1 ELSE 0
    END) AS no_risk_results,

SUM(CASE
        WHEN risk_category = 'Low Risk' THEN 1 ELSE 0
    END) AS low_risk_results,

SUM(CASE
        WHEN risk_category = 'Moderate Risk' THEN 1 ELSE 0
    END) AS medium_risk_results,

SUM(CASE
        WHEN risk_category = 'High Risk' THEN 1 ELSE 0
    END) AS high_risk_results,

COUNT(*) AS total_inspections

FROM
    sf_restaurant_health_violations

GROUP BY
    inspection_type
```

```
ORDER BY
    total_inspections DESC
```

67. Count the number of movies that Abigail Breslin nominated for oscar

Google Netflix Easy ID 10128

Count the number of movies that Abigail Breslin was nominated for an oscar.

Table: oscar_nominees

```
去重重复的记录：COUNT(DISTINCT X)
```

```
SELECT
    COUNT(DISTINCT movie) AS n_movies_by_abi
FROM oscar_nominees
WHERE nominee = 'Abigail Breslin'
```

68. Calculate Samantha's and Lisa's total sales revenue

Amazon Groupon Salesforce Easy ID 10127

What is the total sales revenue of Samantha and Lisa?

Table: sales_performance

```
去AB两个对应的值：
X IN ('A', 'B')
X = 'A' OR 'B'

WHERE salesperson IN ('Samantha', 'Lisa')
WHERE salesperson = 'Samantha' OR salesperson = 'Lisa'
```

```
SELECT
    SUM(sales_revenue) AS total_revenue
FROM sales_performance
WHERE salesperson IN ('Samantha', 'Lisa')
--WHERE salesperson = 'Samantha' OR salesperson = 'Lisa'
```

69.Bookings vs Non-Bookings

Interview Question Date: May 2020

Airbnb Medium ID 10124

Display the average number of times a user performed a search which led to a successful booking and the average number of times a user performed a search but did not lead to a booking. The output should have a column named action with values 'does not book' and 'books' as well as a 2nd column named average_searches with the average number of searches per action. Consider that the booking did not happen if the booking date is null. Be aware that search is connected to the booking only if their check-in dates match.

整体（有book的和没有book的） LEFT JOIN 部分 （book的） ON guest book的条件
整体：做了case when: ts_book不为空且ds.check-in相等 为'book', 否则'does not book', AVG(n_searches)
部分：所有ts_book不为空的记录
ON: id_user = id_guest AND ds_checkin = ds_checkin：把部分（book的记录）与整体（book&unbook的记录）match上

```
--avg # times a user search ~ succ booking & avg # times a user ~ not booking
--output: action：books, does not book , average_searches：  average number of searches per action.
-- book not happen if book date is null; book happen only if check-in dates match

法1:

SELECT
    CASE
          WHEN c.ts_booking_at IS NOT NULL AND c.ds_checkin = s.ds_checkin THEN 'books'
          ELSE 'does not book'
      END AS action,
    AVG(n_searches) AS average_searches
FROM airbnb_searches s
LEFT JOIN
(SELECT *
    FROM airbnb_contacts
WHERE ts_booking_at IS NOT NULL) c
ON s.id_user = c.id_guest
--ON s.id_user = c.id_guest AND s.ds_checkin = c.ds_checkin 效果与ON s.id_user = c.id_guest  一样
GROUP BY 1
```

```
法2：全 left join 部分
WITH c as
(select
    *
FROM airbnb_contacts
WHERE ts_booking_at is not null    --book的
)
SELECT                              --book+not book的
CASE WHEN c.ts_booking_at IS NOT NULL AND c.ds_checkin = s.ds_checkin THEN 'books'  --search ~ booking: ts_checkin date matches
     ELSE 'does not book'
END AS action,
AVG(n_searches) AS average_searches
FROM airbnb_searches s LEFT JOIN c --用全的left join部分的
ON s.id_user = c.id_guest          --顾客id = 用户id
```

```
GROUP BY 1

-- ※此处s是有book的，有没book的；c是全book的
-- s left join c
```

70.Find the total number of searches for houses Westlake neighborhood with a TV

Airbnb Easy ID 10122

Find the total number of searches for houses in Westlake neighborhood with a TV among the amenities.

Table: airbnb_search_details

```
SELECT
    COUNT(*)
FROM airbnb_search_details
WHERE neighbourhood = 'Westlake'
AND amenities ILIKE '%TV%'
AND property_type = 'House'
```

71.Number Of Custom Email Labels

Google Medium ID 10120

Find the number of occurrences of custom email labels for each user receiving an email. Output the receiver user id, label, and the corresponding number of occurrences.

Tables: google_gmail_emails, google_gmail_labels

```
--# custom eail labels for each user receive email
--output:  user_id: receiver user id, label, n_occurences: # occurence

SELECT
    to_user AS user_id,
    label,
    COUNT(*) AS n_occurences
FROM google_gmail_emails e
JOIN google_gmail_labels l
ON e.id = l.email_id
WHERE label ILIKE '%Custom%'
GROUP BY to_user, label
```

72.User Exile

Meta/Facebook Easy ID 10091

Find the number of relationships that user with id == 1 is not part of.

Table: facebook_friends

```
user1!=1 AND user2!=1
1 NOT IN (user1, user2)
```

```
SELECT
    COUNT(*) AS user1_not_in_relationship
FROM facebook_friends
WHERE 1 NOT IN (user1, user2)
```

```
SELECT
    COUNT(*) AS user1_not_in_relationship
FROM facebook_friends
WHERE user1!=1 AND user2!=1
```

73.Find the percentage of shipable orders

Google Amazon Medium ID 10090

Find the percentage of shipable orders. Consider an order is shipable if the customer's address is known.

Tables: orders, customers

```
1.COUNT(CASE WHEN X THEN Y END) / COUNT(*)::FLOAT AS pct
2.CASE WHEN X THEN FALSE ELSE TRUE END AS is_shipable 建立T/F列
SUM(CASE WHEN is_shipable THEN 1 ELSE 0 END) / COUNT(*)::NUMERIC AS pct
3.COUNT(A) * 100 / COUNT(*)
```

```
法1:

COUNT(CASE WHEN X THEN Y END) / COUNT(*)::FLOAT AS pct
SELECT
    COUNT(CASE WHEN c.address IS NOT NULL THEN o.id END) / COUNT(*)::FLOAT *
100 AS percent_shipable
FROM orders o
JOIN customers c
ON o.cust_id = c.id
```

```
法2:

CASE WHEN X THEN FALSE ELSE TRUE END AS is_shipable 建立T/F列
SUM(CASE WHEN is_shipable THEN 1 ELSE 0 END) / COUNT(*)::NUMERIC AS pct

WITH is_shipable AS(
SELECT
    o.id,
    CASE WHEN address IS NULL THEN FALSE ELSE TRUE END AS is_shipable
FROM orders o
JOIN customers c
ON o.cust_id = c.id
)
```

```
SELECT
    SUM(CASE WHEN is_shipable THEN 1 ELSE 0 END)::NUMERIC / COUNT(*) * 100 AS
percent_shipable
FROM is_shipable
```

```
--法3: shippable = #orders with address/#orders

SELECT
    COUNT(c.address) * 100 / COUNT(*) AS percent_shipable
FROM orders o
JOIN customers c
ON o.cust_id = c.id
```

74. Find the number of customers without an order

Google Amazon Medium ID 10089

Find the number of customers without an order.

Tables: orders, customers

```
韦恩图不在求法:
X NOT IN SELECT (B FROM C)
LEFT JOIN + X IS NULL
```

```
SELECT
    COUNT(c.id) AS n_customers_without_orders
FROM customers c
LEFT JOIN orders o
ON o.cust_id = c.id
--WHERE c.id NOT IN (SELECT cust_id FROM orders)
WHERE o.cust_id IS NULL
```

75.Liked' Posts

Meta/Facebook Medium ID 10088

Find the number of posts which were reacted to with a like.

Tables: facebook_reactions, facebook_posts

```
--法1: 整体 left join部分 + 创建like列 T/F, SUM(CASE WHEN like_post = 'TRUE'
THEN 1 ELSE 0 END)
--法2: 算每个post有多少like: SUM(CASE WHEN X THE 1 ELSE 0 END); COUNT(*) +
num_like_each_post !=0
--法3: COUNT(DISTINCT ID) + WHERE reaction = 'like'
```

```
--法1: 整体 left join部分 + 创建like列 T/F, SUM(CASE WHEN like_post = 'TRUE'
THEN 1 ELSE 0 END)
```

```
WITH like_post AS(
SELECT
    DISTINCT
    p.post_id,
    CASE WHEN r.reaction = 'like' THEN TRUE ELSE FALSE END AS like_posts
FROM facebook_posts p
LEFT JOIN facebook_reactions r
ON p.post_id = r.post_id
)
SELECT
    SUM(CASE WHEN like_posts = 'TRUE' THEN 1 ELSE 0 END) AS
n_posts_with_a_like
FROM like_post
```

```
--法2：算每个post有多少like：SUM(CASE WHEN X THE 1 ELSE 0 END)；COUNT(*) +
num_like_each_post !=0
WITH num_like_each_post AS(
SELECT
    DISTINCT p.post_id,
    SUM(CASE WHEN reaction = 'like' THEN 1 ELSE 0 END) AS num_like_each_post
FROM facebook_posts p
LEFT JOIN facebook_reactions r
ON p.post_id = r.post_id
GROUP BY 1
)
SELECT
    COUNT(*)
FROM num_like_each_post
WHERE num_like_each_post != 0
```

```
--法3：COUNT(DISTINCT ID) + WHERE reaction = 'like'
SELECT
    COUNT(DISTINCT p.post_id) AS n_posts_with_a_like
FROM facebook_posts p
LEFT JOIN facebook_reactions r
ON p.post_id = r.post_id
WHERE r.reaction = 'like'
```

76.Find all posts which were reacted to with a heart

Meta/Facebook Easy ID 10087

Find all posts which were reacted to with a heart. For such posts output all columns from facebook_posts table.

Tables: facebook_reactions, facebook_posts

```
去除重复记录： SELECT DISTINCT table.*
```

```
--# posts reacted with a heart
--output all columns
```

```
SELECT
    DISTINCT p.*
FROM
    facebook_posts p
JOIN
    facebook_reactions r
ON
    p.post_id = r.post_id
WHERE
    r.reaction = 'heart'
```

77.Email Details Based On Sends

Google Medium ID 10086

Find all records from days when the number of distinct users receiving emails was greater than the number of distinct users sending emails

Table: google_gmail_emails

```
法1: COUNT(A), COUNT(B) + HAVING COUNT(A) < COUNT(B)
法2: 整体 JOIN 部分 (COUNT(A) / COUNT(B): 等于在原表后加了一列ratio; filter
ratio < 1
```

```
法1: COUNT(A), COUNT(B) + HAVING COUNT(A) < COUNT(B)
WITH base AS(
SELECT
    day,
    COUNT(DISTINCT from_user) AS users_sending_emails,
    COUNT(DISTINCT to_user) AS users_receiving_emails
FROM google_gmail_emails
GROUP BY 1
HAVING COUNT(DISTINCT from_user) < COUNT(DISTINCT to_user)
)
SELECT *
FROM google_gmail_emails
WHERE day IN (SELECT day FROM base)
```

```
法2: 整体 JOIN 部分 (COUNT(A) / COUNT(B): 等于在原表后加了一列ratio; filter
ratio < 1

SELECT
    g.*
FROM google_gmail_emails g
JOIN
    (SELECT
        day,
        COUNT(DISTINCT from_user):: NUMERIC /
        COUNT(DISTINCT to_user) AS sent_received_ratio
    FROM
        google_gmail_emails
    GROUP BY
        day) base
ON g.day = base.day
```

```
AND base.sent_received_ratio < 1
```

78.Meta/Facebook Matching Users Pairs

Meta/Facebook Medium ID 10085

Find matching pairs of Meta/Facebook employees such that they are both of the same nation, different age, same gender, and at different seniority levels. Output ids of paired employees.

Table: facebook_employees

```
法1: self join on e1.id < e2.id for non-repeat pairs
法2: self join:将所有条件做在join里面 + id is not null
```

```
--法1: self join on e1.id < e2.id for non-repeat pairs
SELECT
    a.id AS employee_1,
    b.id AS employee_2
FROM facebook_employees a
JOIN facebook_employees b
ON a.id !=b.id
WHERE a.location = b.location
AND a.age != b.age
AND a.gender = b.gender
AND a.is_senior != b.is_senior
```

```
--法2: self join:将所有条件做在join里面 + id is not null

SELECT
    e1.id AS employee_1,
    e2.id AS employee_2
FROM
    facebook_employees e1
JOIN
    facebook_employees e2
ON
    e1.location = e2.location AND
    e1.age != e2.age AND
    e1.gender = e2.gender AND
    e1.is_senior != e2.is_senior
WHERE
    e1.id IS NOT NULL AND
    e2.id IS NOT NULL
```

79. Cum Sum Energy Consumption

Interview Question Date: April 2020

Meta/Facebook Hard ID 10084

Calculate the running total (i.e., cumulative sum) energy consumption of the Meta/Facebook data centers in all 3 continents by the date. Output the date, running total energy consumption, and running total percentage rounded to the nearest whole number.

```
1.Running Total 和 All Days Total
i.total表：先把三个地区的表 union all一起（如有同一天的相同数字记录会保留）
ii.by date表：date, sum(c) AS day_total求每天的消费明细
iii. running total：date, sum(day_total) over (order by date) 求截止该日为止的消费总和明细
iv. all days total：sum(day_total)是所有日的消费总和
v. pct_of_total: running_toal / all days total = sum(day_total) over (order by date) / sum(day_total))

2.保留整数：round(X / Y, 0)
```

```
-- running total (cum sum) energy consump in all 3 continents by date
--output date, cumulative_total_energy：running total,
percentage_of_total_energy：running total pct rounded to nearest whole number

WITH total AS(
    SELECT * FROM fb_eu_energy
    UNION ALL
    SELECT * FROM fb_na_energy
    UNION ALL
    SELECT * FROM fb_asia_energy
    ORDER BY 1
),
energy_by_date AS(
    SELECT
        date,
        SUM(consumption) AS total_energy
    FROM total
    GROUP BY date
    ORDER BY date
)
SELECT
    date,
    SUM(total_energy) OVER (ORDER BY DATE) AS cumulative_total_energy,
    -- (SELECT SUM(total_energy) FROM energy_by_date) AS
all_days_total_energy,
    ROUND(SUM(total_energy) OVER(ORDER BY DATE) * 100 / (SELECT
SUM(total_energy) FROM energy_by_date), 0) AS percentage_of_total_energy --
cum / all_days_total
FROM energy_by_date
```

80.Start Dates Of Top Drivers

Lyft Medium ID 10083

Find contract starting dates of the top 5 most paid Lyft drivers. Consider only drivers who are still working with Lyft.

Table: lyft_drivers

```
--contract starting dates of top 5 most paid drivers
--only drivers still work with lyft: end_date IS NULL: end_date是空值

--output: start_date

SELECT start_date
FROM
    (SELECT
        *,
        RANK() OVER (
                    ORDER BY yearly_salary DESC) AS rnk
    FROM lyft_drivers
    WHERE end_date IS NULL) t
WHERE rnk <= 5
```

81. Find the number of employees who received the bonus and who didn't

Microsoft Dell Hard ID 10081

Find the number of employees who received the bonus and who didn't. Bonus values in employee table are corrupted so you should use values from the bonus table. Be aware of the fact that employee can receive more than bonus. Output value inside has_bonus column (1 if they had bonus, 0 if not) along with the corresponding number of employees for each.

Tables: employee, bonus

```
--  # employees receive the bonus and who did not
--  bonus values in employee table ×; use bonus values from the bonus table
-- can receive bonus > 1
--  output: has_bonus: has_bonus (1,0) , n_employees: # employess


SELECT
CASE WHEN b.bonus_amount IS NOT NULL THEN 1
     ELSE 0
     END AS has_bonus,
COUNT(DISTINCT e.id)
FROM
    employee e
LEFT JOIN
    bonus b
ON
    b.worker_ref_id = e.id
GROUP BY 1
```

82. Find matching hosts and guests in a way that they are both of the same gender and nationality

Airbnb Medium ID 10078

Find matching hosts and guests pairs in a way that they are both of the same gender and nationality. Output the host id and the guest id of matched pair.

Tables: airbnb_hosts, airbnb_guests

```
DISTINCT 去除ID PAIR中重复的记录
```

```
SELECT
    DISTINCT
    h.host_id,
    g.guest_id
FROM airbnb_hosts h
JOIN airbnb_guests g
ON h.nationality = g.nationality AND h.gender = g.gender
```

83.Income By Title and Gender

City of San Francisco Medium ID 10077

Find the average total compensation based on employee titles and gender. Total compensation is calculated by adding both the salary and bonus of each employee. However, not every employee receives a bonus so disregard employees without bonuses in your calculation. Employee can receive more than one bonus. Output the employee title, gender (i.e., sex), along with the average total compensation.

Tables: sf_employee, sf_bonus

```
--存在同一个人同一个salary对应多个bonus的情况:
--需要先将第二个表，将一个worker_id，对应一个total_bonus
--再将两表join起来，salary + total_bonus作为total_comp，再avg()

with cte as(
select
    worker_ref_id,
    sum(bonus) as total_bonus
from sf_bonus
group by 1
)
select
employee_title,
sex,
avg(a.salary + b.total_bonus) as avg_total_com
from sf_employee a join cte b
on a.id = b.worker_ref_id
group by 1,2
```

84.Find the average age of guests reviewed by each host

Airbnb Medium ID 10074

Find the average age of guests reviewed by each host. Output the user along with the average age.

Tables: airbnb_reviews, airbnb_guests

```
-- avg age of guests reviewed by each host
--output: from_user: HOST, average_age: guests

--1.题意是每个host 去评价的 客户的平均年龄: from_type = 'host'; from_user AS
host_id; avg(age) AS avg_guest_age

--2.在应该保留全部记录的题目中，不要使用distinct。
--一个host可以给同一个guest多次打分，而且存在多次打分中仍然打一样的分。
--这样导致如果total使用distinct，则有不该被省略的记录被省略。
```

```
WITH total AS(
SELECT
    *
FROM airbnb_reviews r
JOIN airbnb_guests g
ON r.to_user = g.guest_id
WHERE from_type = 'host'
ORDER BY 1,2
)
SELECT
    from_user,
    AVG(age) AS average_age
FROM total
GROUP BY 1
ORDER BY 1
```

85. Favorite Host Nationality

Interview Question Date: April 2020

Airbnb Medium ID 10073

For each guest reviewer, find the nationality of the reviewer's favorite host based on the guest's highest review score given to a host. Output the user ID of the guest along with their favorite host's nationality. In case there is more than one favorite host from the same country, list that country only once (remove duplicates).

Both the from_user and to_user columns are user IDs.

Tables: airbnb_reviews, airbnb_hosts

```
稳妥的方法:
i. join两表开新表total时，保留全部记录*, rnk
ii. 在计算时，取出所需变量，视情况看是否需要distinct
```

```
--each guest reviewer ~ nationality of the reviewer's favorite host ~ guest's
highest review score
```

```
--output: from_user: user ID of the guest; nationality: favorite host

--in case > 1 favorite host from the same country, list country only
once(remove duplicate)

WITH total AS(
SELECT
DISTINCT
    *,
    -- from_user,
    -- to_user,
    -- nationality,
    -- review_score,
    RANK() OVER (PARTITION BY from_user ORDER BY review_score DESC) as rnk
FROM airbnb_reviews r
JOIN airbnb_hosts h
ON r.to_user = h.host_id
WHERE from_type = 'guest'
ORDER BY 1
)
SELECT
    DISTINCT
    from_user,
    nationality
FROM total
WHERE rnk = 1
```

86.Guest Or Host Kindness

Interview Question Date: April 2020

Airbnb Easy ID 10072

Find whether hosts or guests give higher review scores based on their average review scores. Output the higher of the average review score rounded to the 2nd decimal spot (e.g., 5.11).

Table: airbnb_reviews

```
order by中可以有新产生的变量名
```

```
--是H or G 给了高分，依据他们的平均分 ==> 留一个最高平均分的记录

SELECT from_type,
       ROUND(AVG(review_score), 2) winner
FROM airbnb_reviews
GROUP BY from_type
ORDER BY winner DESC
LIMIT 1
```

```
SELECT
    from_type,
    ROUND(av,2)
FROM(
```

```
    SELECT
        from_type,
        AVG(review_score) AS av,
        DENSE_RANK() OVER (ORDER BY AVG(review_score) DESC) AS rank
    FROM airbnb_reviews
GROUP BY from_type) m
WHERE rank = 1
```

87.Hosts' Abroad Apartments

Airbnb Medium ID 10071

Find the number of hosts that have accommodations in countries of which they are not citizens.

Tables: airbnb_hosts, airbnb_apartments

```
--# of hosts that have accomadations in countries of which host are not
citizens
--output: count

SELECT
    COUNT(DISTINCT a.host_id)
FROM airbnb_hosts h
JOIN airbnb_apartments a
ON h.host_id = a.host_id
WHERE h.nationality != a.country
```

88.DeepMind employment competition

Google Medium ID 10070 14

Data Engineer Data Scientist BI Analyst Data Analyst Find the winning teams of DeepMind employment competition. Output the team along with the average team score. Sort records by the team score in descending order.

Tables: google_competition_participants, google_competition_scores

```
 sum(b.member_score) / count(a.member_id) = avg(b.member_score)
```

```
SELECT
    team_id,
    AVG(member_score) AS team_score
FROM google_competition_participants p
JOIN google_competition_scores s
ON p.member_id = s.member_id
GROUP BY 1
ORDER BY 2 DESC
```

89.Correlation Between E-mails And Activity Time

Interview Question Date: April 2020

Google Hard ID 10069

There are two tables with user activities. The google_gmail_emails table contains information about emails being sent to users. Each row in that table represents a message with a unique identifier in the id field. The google_fit_location table contains user activity logs from the Google Fit app. Find the correlation between the number of emails received and the total exercise per day. The total exercise per day is calculated by counting the number of user sessions per day.

Tables: google_gmail_emails, google_fit_location

```
i.每个用户每天收#邮件：取to_user, day, count(*)，从google_gmail_emails
ii. 每个用户每天exercise：取user_id, day, count(distinct session_id)
【为什么distinct?】：在google_fit_location中，因为step_id不同，会出现同一user_id
同一天同一session_id的不同记录，所以要distinct(session_id)
iii. join两个表：email和session，on两个条件day and user_id 【为什么也on
user_id?】
vi. COALESCE (A,0) ：A值缺少，则取0
v. CORR(X,Y)：XY两变量间correlation
```

```
-- google_gmail_emails: emails sent to users; each row ~ unique identifier ~
a message
-- google_fit_location: user activity logs from google fit app

--relationship between # emails received and total exercise per day:counting
the number of user sessions per day
--corr

--写法1：CTE
WITH SESSIONS AS(
SELECT
    user_id,                                                          --要
取user_id
    day,
    COUNT(DISTINCT session_id) AS session                             --必
须distinct session_id
FROM google_fit_location
GROUP BY 1,2
),
EMAIL AS(
SELECT
    to_user,                                                          --要
取user_id
    day,
    COUNT(id) AS email_received
FROM google_gmail_emails
GROUP BY 1,2
)
SELECT
```

```
    CORR(COALESCE(email_received,0), COALESCE(session,0)) AS corr       --要
COALESCE(X,0)
FROM EMAIL e
FULL JOIN SESSIONS s                                                    --必
须full outter join
ON e.day = s.day AND user_id = to_user                                  --on
两个条件:user_id; day
```

```
写法2：subquery
SELECT
    corr(COALESCE(n_emails:: NUMERIC, 0), COALESCE(total_exercise, 0))   --
corr(X,Y); COALESCE(A,B)
FROM
(
    SELECT                                                               --每
个用户每天收#邮件
        to_user,
        day,
        COUNT(*) AS n_emails
    FROM google_gmail_emails
    GROUP BY 1,2) mail_base
FULL OUTER JOIN
(                                                                        --每
个用户每天exercise
    SELECT
        user_id,
        day,
        COUNT(DISTINCT session_id) AS total_exercise
    FROM google_fit_location
    GROUP BY 1,2
    ) total_exercise
ON mail_base.to_user = total_exercise.user_id
AND mail_base.day = total_exercise.day                                  --ON
to_user = user_id AND day = day
JOIN google_fit_location f
ON g.to_user = f.user_id
GROUP BY 1
```

90. User Email Labels

Interview Question Date: April 2020

Google Medium ID 10068

Find the number of emails received by each user under each built-in email label. The email labels are: 'Promotion', 'Social', and 'Shopping'. Output the user along with the number of promotion, social, and shopping mails count,.

Tables: google_gmail_emails, google_gmail_labels Hints Expected Output All required columns and the first 5 rows of the solution are shown

```
COUNT(case when A then 1 else null end) = SUM(case when A then 1 else 0 end)
```

```
case when 后的end别忘记
```

```
-- # emails received by each user ~ each built-in email label
-- labels:  'Promotion', 'Social', and 'Shopping'.
-- output: user ~ # of p, s, s mails count
-- output: to_user, promotion_count, social_count, shopping_count

SELECT
    to_user,
    COUNT(CASE WHEN label = 'Promotion' THEN id END) AS promotion_count,
    COUNT(CASE WHEN label = 'Social' THEN id END) AS social_count,
    COUNT(CASE WHEN label = 'Shopping' THEN id END) AS shopping_count
FROM google_gmail_emails e
JOIN google_gmail_labels l
ON e.id = l.email_id
GROUP BY 1
```

91. Google Fit User Tracking

92. Fans vs Opposition

Interview Question Date: March 2020

Meta/Facebook Hard ID 10062

Meta/Facebook is quite keen on pushing their new programming language Hack to all their offices. They ran a survey to quantify the popularity of the language and send it to their employees. To promote Hack they have decided to pair developers which love Hack with the ones who hate it so the fans can convert the opposition. Their pair criteria is to match the biggest fan with biggest opposition, second biggest fan with second biggest opposition, and so on. Write a query which returns this pairing. Output employee ids of paired employees. Sort users with the same popularity value by id in ascending order.

Duplicates in pairings can be left in the solution. For example, (2, 3) and (3, 2) should both be in the solution.

Table: facebook_hack_survey

```
求一高一低PAIR ID:
用row_number保持唯一性
法1:
i.love_rnk, hate_rnk
ii. ID PAIR: id1, id2 + self join on love_rnk = hate_rnk
法2:
i. 分别建立两个表：各取id, love/hate rnk
ii.ID PAIR: id1, id2 + join on love_rnk = hate_rnk
```

```
法1:
i.love_rnk, hate_rnk
ii. ID PAIR: id1, id2 + self join on love_rnk = hate_rnk
WITH total AS(
```

```
SELECT
    *,
    RANK() OVER (ORDER BY popularity DESC, employee_id ASC) AS love_rnk,
    RANK() OVER (ORDER BY popularity, employee_id ASC) AS hate_rnk
FROM facebook_hack_survey
)
SELECT
    t1.employee_id,
    t2.employee_id
FROM total t1
JOIN total t2
ON t1.love_rnk = t2.hate_rnk
```

```
法2:
i. 分别建立两个表：各取id, love/hate rnk
ii.ID PAIR: id1, id2 + join on love_rnk = hate_rnk
WITH love_rnk AS(
    SELECT
        employee_id,
        ROW_NUMBER() OVER (ORDER BY popularity DESC, employee_id ASC) AS
love_rnk
    FROM facebook_hack_survey),
    hate_rnk AS(
    SELECT
        employee_id,
        ROW_NUMBER() OVER (ORDER BY popularity, employee_id ASC) AS hate_rnk
    FROM facebook_hack_survey
    )
SELECT
    l.employee_id AS employee_fan_id,
    h.employee_id AS employee_opposition_id
FROM love_rnk l
JOIN hate_rnk h
ON l.love_rnk = h.hate_rnk
```

93. Find the number of reviews received by Lo-Lo's Chicken & Waffles for each star

Yelp Easy ID 10058

Find the number of reviews received by Lo-Lo's Chicken & Waffles for each star. Output the number of stars along with the corresponding number of reviews. Sort records by stars in ascending order.

Table: yelp_reviews

```
--ILIKE 'Lo-Lo_s Chicken & Waffles'
--like 'Lo-Lo%s Chicken & Waffles'
--如果名字中 有撇 '，可以把' 写成_或者%；活用ILIKE
```

```
--# reviews received for each star
--output:stars, n_reviews:  # stars and # reviews
--order by stars asc
```

```
SELECT
    stars,
    COUNT(*) AS n_reviews
FROM yelp_reviews
WHERE business_name ILIKE 'Lo-Lo_s Chicken & Waffles'
GROUP BY 1
ORDER BY 1
```

94.Popularity of Hack

Interview Question Date: March 2020

Meta/Facebook Easy ID 10061

Meta/Facebook has developed a new programing language called Hack.To measure the popularity of Hack they ran a survey with their employees. The survey included data on previous programing familiarity as well as the number of years of experience, age, gender and most importantly satisfaction with Hack. Due to an error location data was not collected, but your supervisor demands a report showing average popularity of Hack by office location. Luckily the user IDs of employees completing the surveys were stored. Based on the above, find the average popularity of the Hack per office location. Output the location along with the average popularity.

Tables: facebook_employees, facebook_hack_survey

95. Popularity of Hack

Interview Question Date: March 2020

Meta/Facebook Easy ID 10061

Meta/Facebook has developed a new programing language called Hack.To measure the popularity of Hack they ran a survey with their employees. The survey included data on previous programing familiarity as well as the number of years of experience, age, gender and most importantly satisfaction with Hack. Due to an error location data was not collected, but your supervisor demands a report showing average popularity of Hack by office location. Luckily the user IDs of employees completing the surveys were stored. Based on the above, find the average popularity of the Hack per office location. Output the location along with the average popularity.

Tables: facebook_employees, facebook_hack_survey

```
--error location data not collect
--avg pop of hack by office
--avg pop of the hack per office location
--output: location, avg_popularity

SELECT
```

```
    location,
    AVG(popularity) AS avg_popularity
FROM facebook_employees e
JOIN facebook_hack_survey s
ON e.id = s.employee_id
GROUP BY 1
```

Top Cool Votes

Interview Question Date: March 2020

Yelp Medium

Find the review_text that received the highest number of 'cool' votes. Output the business name along with the review text with the highest numbef of 'cool' votes.

Table: yelp_reviews

```
--review_text receive highest # cool votes
--output: business_name, review_text

--法1：rank cte做法：
WITH rnk AS(
SELECT
    business_name,
    review_text,
    cool,
    RANK() OVER (ORDER BY cool DESC) AS rnk
FROM yelp_reviews
)
SELECT
    business_name,
    review_text
FROM rnk
WHERE rnk = 1
```

```
--法2：max subquery做法：
SELECT
    business_name,
    review_text
FROM yelp_reviews
WHERE cool =
    (
    SELECT
    MAX(cool) AS max_cool
    FROM yelp_reviews
    )
```

Find the number of reviews received by Lo-Lo's Chicken & Waffles for each star

Yelp Easy ID 10058

Find the number of reviews received by Lo-Lo's Chicken & Waffles for each star. Output the number of stars along with the corresponding number of reviews. Sort records by stars in ascending order.

Table: yelp_reviews

```
ILIKE 'Lo-Lo_s Chicken & Waffles'
like 'Lo-Lo%s Chicken & Waffles'
如果名字中 有撇 '，可以把' 写成_或者%；活用ILIKE
```

```
--# reviews received for each star
--output:stars, n_reviews:  # stars and # reviews
--order by stars asc


SELECT
    stars,
    COUNT(*) AS n_reviews
FROM yelp_reviews
WHERE business_name ILIKE 'Lo-Lo_s Chicken & Waffles'
GROUP BY 1
ORDER BY 1
```

Most Checkins

Yelp Medium ID 10053

Find the top 5 businesses with the most check-ins. Output the business id along with the number of check-ins.

Table: yelp_checkin

```
--top 5 biz with most check-ins
--output biz id with # check-ins
--output: business_id, n_checkins

--rank法求top 5:
WITH checkins_rank AS(
SELECT
    business_id,
    SUM(checkins) AS n_checkins,
    RANK() OVER (ORDER BY SUM(checkins) DESC) AS rnk
FROM yelp_checkin
GROUP BY 1
)
SELECT
    business_id,
    n_checkins
FROM checkins_rank
```

```
WHERE rnk <= 5
```

```
--limit法求top 5:
SELECT
    business_id,
    SUM(checkins) AS counts
FROM yelp_checkin
GROUP BY 1
ORDER BY 2 DESC
LIMIT 5
```

Find the average number of stars for each state

Yelp Easy ID 10052

Find the average number of stars for each state. Output the state name along with the corresponding average number of stars.

Table: yelp_business

```
--select * from yelp_business;

SELECT
    state,
    AVG(stars) AS average_stars
FROM yelp_business
GROUP BY 1
```

Find the number of open businesses

Yelp Easy ID 10051

Find the number of open businesses.

Table: yelp_business

```
--select * from yelp_business;

SELECT
    COUNT(*) AS business_open
FROM yelp_business
WHERE is_open = 1
```

Find the review count for one-star businesses from yelp

Yelp Easy ID 10050

Find the review count for one-star businesses from yelp. Output the name along with the corresponding review count.

Table: yelp_business

```
--select * from yelp_business;
SELECT
    name,
    review_count
FROM yelp_business
WHERE stars = 1
```

Reviews of Categories

Interview Question Date: March 2020

Yelp Medium ID 10049

Find the top business categories based on the total number of reviews. Output the category along with the total number of reviews. Order by total reviews in descending order.

Table: yelp_business

```
--top biz categories ~ total # reviews
--output: category, review_cnt:
--order by total reviews desc

--UNNEST(STRING_TO_ARRAY(A, ';')): 把A中用; 分隔开的每个词 分解取出


SELECT
    UNNEST(STRING_TO_ARRAY(categories, ';')) AS category,
    SUM(review_count) AS review_cnt
FROM yelp_business
GROUP BY 1
ORDER BY 2 DESC
```

Top Businesses With Most Reviews

Yelp Medium ID 10048

Find the top 5 businesses with most reviews. Assume that each row has a unique business_id such that the total reviews for each business is listed on each row. Output the business name along with the total number of reviews and order your results by the total reviews in descending order.

Table: yelp_business

In [ ]:
```
--top 5 biz with most reviews
--unique biz_id ~ total reviews for each biz
--output: name, review_count:  business name along with the total number of re
--order by total reviews desc

--LIMIT 5 不好, rank 求top 5才最精准:
WITH business_rank AS(
SELECT
    name,
    review_count,
    RANK() OVER (ORDER BY review_count DESC) AS rnk
FROM yelp_business
)
SELECT
    name,
    review_count
FROM business_rank
WHERE rnk <= 5
```

Top 5 States With 5 Star Businesses

Interview Question Date: March 2020

Yelp Hard ID 10046

Find the top 5 states with the most 5 star businesses. Output the state name along with the number of 5-star businesses and order records by the number of 5-star businesses in descending order. In case there are ties in the number of businesses, return all the unique states. If two states have the same result, sort them in alphabetical order.

Table: yelp_business

```
-- top 5 states with most 5 star biz
-- output: state,  n_businesses:
-- order by # 5 star biz in desc order,  sort state asc
-- in case ties in # biz, return all unique states

--求top 5, 用limit 5会不精准-因为会漏项, 还是rank精准

WITH biz_rank AS(
SELECT
    state,
    COUNT(business_id) AS n_businesses,
    RANK() OVER (ORDER BY COUNT(business_id) DESC) AS rnk
FROM yelp_business
WHERE stars = 5
GROUP BY 1
ORDER BY 2 DESC, 1 ASC
)
SELECT
    state,
    n_businesses
```

```
FROM biz_rank
WHERE rnk <= 5
```

Highest Priced Wine In The US

Interview Question Date: March 2020

Wine Magazine Medium ID 10044 13

Data Engineer Data Scientist BI Analyst Data Analyst Find the highest price in US country for each variety produced in English speaking regions, but not in Spanish speaking regions, with taking into consideration varieties that have earned a minimum of 90 points for every country they're produced in. Output both the variety and the corresponding highest price.

Let's assume the US is the only English speaking region in the dataset, and Spain, Argentina are the only Spanish speaking regions in the dataset. Let's also assume that the same variety might be listed under several countries so you'll need to remove varieties that show up in both the US and in Spanish speaking countries.

Table: winemag_p1

```
--highest price in US ~ each variety produced in English regions but not
spanish regions
--√ varieties earned mini 90 for every country produced
--output: variety, max: variety and highest price
--US ~ only English speaking region; Spain, Argentina ~ Spanish only
--remove varieties show up in both US & SPain

--问题：一个variety种类，可以有多个points；一个variety种类，可以有多个价格


WITH remove AS(
SELECT
    variety
FROM winemag_p1
WHERE country IN ('Spain', 'Argentina')
),
residue AS(
SELECT
    variety
FROM winemag_p1
WHERE variety NOT IN (SELECT variety FROM remove)
GROUP BY 1
HAVING MIN(points) >= 90
)
SELECT
    variety,
    MAX(price)
FROM winemag_p1
WHERE variety IN (SELECT variety FROM residue)
AND country = 'US'
```

```
GROUP BY 1
```

Median Price Of Wines

Interview Question Date: March 2020

Wine Magazine Hard ID 10043

Find the median price for each wine variety across both datasets. Output distinct varieties along with the corresponding median price.

Tables: winemag_p1, winemag_p2

```
①当两个dataset列数不一致时，可以选出需要的列进行union操作；如果列数一直，可以直接select * 进行union操作
②此处应该UNION ALL而不是UNION，选择两个dataset的全部
③median：
i. percentile_cont(0.5) within GROUP (ORDER BY A) AS median_price
ii. avg(A) + WHERE row_number_asc IN (row_number_desc, row_number_desc - 1, row_number_desc + 1 );
如果是奇数个记录，则选两个row_num相等的是中位数(eg. n = 5, median = asc = desc = 3)的记录选出来求平均值，即这个记录的值
如果是偶数个记录，则将asc = desc - 1 和asc = desc + 1的两个记录选出来求平均值
```

```
--法1：percentile_cont(0.5) within GROUP (ORDER BY A)
WITH total AS(
    SELECT
        variety,
        price::float
    FROM winemag_p1
    UNION ALL
    SELECT
        variety,
        price::float
    FROM winemag_p2
)
SELECT
    variety,
    percentile_cont(0.5) within GROUP (ORDER BY price) AS median_price
FROM total
GROUP BY 1
```

```
--法2：row_number：avg(A) + WHERE row_number_asc IN (row_number_desc, row_number_desc - 1, row_number_desc + 1 )
WITH total AS(
    SELECT
        variety,
        price
    FROM winemag_p1
    WHERE price IS NOT NULL
    UNION ALL
    SELECT
        variety,
        price
```

```
    FROM winemag_p2
    WHERE price IS NOT NULL
),
W1 AS(
SELECT
    variety,
    price,
    ROW_NUMBER() OVER (PARTITION BY variety ORDER BY price ASC) AS rnk_asc,
    ROW_NUMBER() OVER (PARTITION BY variety ORDER BY price DESC) AS rnk_desc

FROM total
)
SELECT
    variety,
    AVG(price)
FROM W1
WHERE rnk_asc IN (rnk_desc, rnk_desc - 1, rnk_desc + 1)
GROUP BY 1
```

Top 3 Wineries In The World

Interview Question Date: March 2020

Wine Magazine Hard ID 10042

Find the top 3 wineries in each country based on the average points earned. In case there is a tie, order the wineries by winery name in ascending order. Output the country along with the best, second best, and third best wineries. If there is no second winery (NULL value) output 'No second winery' and if there is no third winery output 'No third winery'. For outputting wineries format them like this: "winery (avg_points)"

Table: winemag_p1

```
①连接括号写成(C)的形式：CASE WHEN A THEN B || ' (' || C || ')' ELSE NULL END
AS D
②ROW_NUMBER() OVER (PARTITION BY A ORDER BY B, C) AS row_num
③max(A) 取一个值
④如果没有A，就写成B：COALESCE(max(A), 'B')
```

```
-- top 3 wineries in each country ~ avg points earned
-- order wineries by winery name in sac
-- output:country, top_winery, second_winery, third_winery：winery
(avg_points)
-- If there is no second winery (NULL value) output 'No second winery' and if
there is no third winery output 'No third winery'

SELECT country,
       max(top_winery) AS top_winery,
       COALESCE(max(second_winery), 'No second winery') AS second_winery,
       COALESCE(max(third_winery), 'No third winery') AS third_winery
FROM
  (SELECT country,
          CASE
```

```
            WHEN POSITION = 1 THEN winery || ' (' || round(avg_points) ||
')'
            ELSE NULL
        END AS top_winery,
        CASE
            WHEN POSITION = 2 THEN winery || ' (' || round(avg_points) ||
')'
            ELSE NULL
        END AS second_winery,
        CASE
            WHEN POSITION = 3 THEN winery || ' (' || round(avg_points) ||
')'
            ELSE NULL
        END AS third_winery
   FROM
     (SELECT country,
             winery,
             ROW_NUMBER() OVER (PARTITION BY country
                                ORDER BY avg_points DESC, winery ASC) AS
POSITION,
                               avg_points
      FROM
        (SELECT country,
                winery,
                avg(points) AS avg_points
         FROM winemag_p1
         WHERE country IS NOT NULL
         GROUP BY country,
                  winery) tmp1) tmp2
   WHERE POSITION <= 3) tmp3
GROUP BY country
```

Most Expensive And Cheapest Wine

Interview Question Date: March 2020

Wine Magazine Hard ID 10041

Find the cheapest and the most expensive variety in each region. Output the region along with the corresponding most expensive and the cheapest variety. Be aware that there are 2 region columns, the price from that row applies to both of them.

Note: The results set contains no ties.

Table: winemag_p1

```
①total: SELECT * FROM A UNION ALL SELECT * FROM B
②计算A中，C在B最大/小时的值：
cte1: A, ROW_NUMBER() OVER (ORDER BY B DESC) AS rnk1, ROW_NUMBER() OVER
(ORDER BY B ASC) AS rnk2,
cte2: A, MAX(CASE WHEN rnk1 = 1 THEN C END) AS max_price_B; MIN(CASE WHEN
rnk1 = 1 THEN C END) AS min_price_B + GROUP BY A
```

③当遇到case when创建后，本应放在一行的记录，放成了几行，中间可能有空格。解决方法是A, MAX() + group by 1，可以将其并为一行

```
WITH total AS(
    SELECT
        region_1 AS region,
        price,
        variety
    FROM winemag_p1
    WHERE region_1 IS NOT NULL
    UNION ALL
    SELECT
        region_2 AS region,
        price,
        variety
    FROM winemag_p1
    WHERE region_2 IS NOT NULL
),
result AS (
    SELECT
        *,
        ROW_NUMBER() OVER (PARTITION BY region ORDER BY price DESC) AS
expensive_rank,
        ROW_NUMBER() OVER (PARTITION BY region ORDER BY price ASC) AS
cheap_rank
    FROM total
    WHERE price IS NOT NULL
    AND region IS NOT NULL
    ORDER BY region, price DESC
)
SELECT
    region,
    MAX(CASE
        WHEN expensive_rank = 1 THEN variety
        ELSE NULL
    END) AS most_expensive_variety,
    MIN(CASE
        WHEN cheap_rank = 1 THEN variety
        ELSE NULL
    END) AS cheapest_variety
FROM result
GROUP BY 1
```

Find all wines from the winemag_p2 dataset which are produced in countries that have the highest sum of points in the winemag_p1 dataset

Wine Magazine Hard ID 10040

Find all wines from the winemag_p2 dataset which are produced in the country that have the highest sum of points in the winemag_p1 dataset.

Tables: winemag_p1, winemag_p2

```
--all wines from winmag_p2 produced in the country ~ highest sum of points in
1_p1

--select * from winemag_p1;
WITH sum_points_country AS(
SELECT
    country,
    SUM(points) AS sum_points,
    RANK() OVER (ORDER BY SUM(points) DESC) AS rnk
FROM winemag_p1
GROUP BY country
)
SELECT
    *
FROM winemag_p2
WHERE country IN (SELECT country FROM sum_points_country WHERE rnk = 1)
```

Macedonian Vintages

Wine Magazine Medium ID 10039

Find the vintage years of all wines from the country of Macedonia. The year can be found in the 'title' column. Output the wine (i.e., the 'title') along with the year. The year should be a numeric or int data type.

Table: winemag_p2

```
--select * from winemag_p2;

--vintage years of all wines from Macedonia
--year ~ title
--output: title, year

SELECT
    title,
    substring(title, '[0-9]{4}'):: integer AS year
--    NULLIF(regexp_replace(title, '\D','','g'), ''):: NUMERIC AS year
--    split_part(title, ' ', 2) :: NUMERIC AS year
FROM winemag_p2
WHERE country = 'Macedonia'
```

Find all provinces which produced more wines in 'winemag_p1' than they did in 'winemag_p2'

Interview Question Date: March 2020

Wine Magazine Medium ID 10038

Find all provinces which produced more wines in 'winemag_p1' than they did in 'winemag_p2'. Output the province and the corresponding wine count. Order records by the wine count in descending order.

```
-- all provinces produce more wines in p1 > p2
-- output: province, cnt_1
-- order by cnt desc

SELECT
    a.province,
    a.cnt_1
FROM
    (SELECT
        province,
        COUNT(winery) AS cnt_1         --COUNT(*)也可
    FROM winemag_p1
    GROUP BY province) a
    JOIN
    (SELECT
        province,
        COUNT(winery) AS cnt_2
    FROM winemag_p2
    GROUP BY province) b
ON a.province = b.province
WHERE cnt_1 - cnt_2 > 0              --ON两个条件也可
ORDER BY cnt_1 DESC
```

Find Favourite Wine Variety

Interview Question Date: March 2020

Wine Magazine Hard ID 10037

Find each taster's favorite wine variety. Consider that favorite variety means the variety that has been tasted by most of the time. Output the taster's name along with the wine variety.

Table: winemag_p2

```
--each taster's fa wine variety
--tasted by most of the time
--output: taster_name, variety:
-- select * from winemag_p2
-- WHERE taster_name = 'Joe Czerwinski'
--AND variety = 'Champagne Blend'

--#算favarite 是算一个taster_name, variety 有多少个记录: count(*)
--#修好：有一个没有年份的漏算记录；有重复年份的按多个记录算

WITH total AS(
SELECT
    taster_name,
    variety,
    substring(title, '[0-9]{4}')::integer AS year
```

```
FROM winemag_p2
WHERE taster_name IS NOT NULL
--AND substring(title, '[0-9]{4}') IS NOT NULL
),
year_rank AS(
SELECT
    taster_name,
    variety,
    COUNT(*) AS years,
    RANK() OVER (PARTITION BY taster_name ORDER BY COUNT(*) DESC) AS rnk
FROM total
GROUP BY 1, 2
)
SELECT
    taster_name,
    variety
FROM year_rank
WHERE rnk = 1
```

Find the number of wines with and without designations per country

Interview Question Date: March 2020

Wine Magazine Medium ID 10035

Find the number of wines with and without designations per country. Output the country along with the total without designations, total with designations, and the final total of both.

Table: winemag_p2

```
--# wines with and without designations per country
--output: country, total_without_designation, total_with_designation,
grand_total

--该值为空 IS NULL
--当计算该值有多少个时：SUM(CASE WHEN A THEN 1 ELSE 0 END) ；不能将SUM改为
COUNT，因为如用COUNT(CASE WHEN A THEN 1 ELSE 0 END)代表共计多少个 = COUNT(*)

SELECT
    country,
    SUM(CASE WHEN designation IS NULL THEN 1 ELSE 0 END) AS
total_without_designation,
    SUM(CASE WHEN designation IS NOT NULL THEN 1 ELSE 0 END) AS
total_with_designation,
    COUNT(*) AS grand_total
FROM winemag_p2
GROUP BY country
```

```
--# wines with and without designations per country
--output: country, total_without_designation, total_with_designation,
grand_total

--该值为空 IS NULL
```

```
--当计算该值有多少个时：SUM(CASE WHEN A THEN 1 ELSE 0 END) ；不能将SUM改为
COUNT，因为如用COUNT(CASE WHEN A THEN 1 ELSE 0 END)代表共计多少个 = COUNT(*)

SELECT
    country,
    SUM(CASE WHEN designation IS NULL THEN 1 ELSE 0 END) AS
total_without_designation,
    SUM(CASE WHEN designation IS NOT NULL THEN 1 ELSE 0 END) AS
total_with_designation,
    COUNT(*) AS grand_total
FROM winemag_p2
GROUP BY country
```

Wine Variety Revenues

Interview Question Date: February 2020

Wine Magazine Medium ID 10033

Find the total revenue made by each region from each variety of wine in that region. Output the region, variety, and total revenue.

Take into calculation both region_1 and region_2. Remove the duplicated rows where region, price and variety are exactly the same.

Table: winemag_p1

```
-- total rev made by each rgion ~ each variety of wine in region
-- output: region, variety, sum：  total revenue
-- remove duplicated rows where region, price, variety are the same


--一个区域内，一个variety，可以对应多个price，把所有price相加一起，就是revenue

WITH total AS(
    (SELECT
        region_1 AS region,
        price,
        variety
    FROM
    winemag_p1
    )
    UNION
    (SELECT
        region_2 AS region,
        price,
        variety
    FROM
    winemag_p1
    )
)
SELECT
    DISTINCT
    region,
```

```
    variety,
    SUM(price) AS sum
FROM total
WHERE region IS NOT NULL AND price IS NOT NULL         --此句不可少
GROUP BY region, variety
ORDER BY SUM(price) DESC
```

Best Wines By Points-To-Price

Interview Question Date: February 2020

Wine Magazine Medium ID 10032

Find the wine with the highest points to price ratio. Output the title, points, price, and the corresponding points-to-price ratio.

Table: winemag_p2

```
--①如果项不是数字格式，要换为数字：  ::NUMERIC
--②找最大值的两种写法：
-- i. rank = 1
-- ii. A = SELECT MAX(A) FROM B
```

```
WITH ratio AS(
    SELECT
        title,
        points,
        price,
        points:: NUMERIC / price:: NUMERIC AS points_price_ratio,
        RANK() OVER (ORDER BY points / price DESC) AS rnk
    FROM winemag_p2
    WHERE points / price IS NOT NULL
)
SELECT
        title,
        points,
        price,
        points_price_ratio
FROM ratio
--WHERE rnk = 1
WHERE points_price_ratio = (SELECT max(points_price_ratio) FROM ratio)
```

Find the number of Bodegas outside of Spain that produce wines with the blackberry taste

Interview Question Date: February 2020

Wine Magazine Medium ID 10031

Find the number of Bodegas (wineries with "bodega" pattern inside the name) outside of Spain that produce wines with the blackberry taste (description contains blackberry string). Group the count by country and region. Output the country, region along with the number of bodegas.

```
①一个地区可能有多个相同名称的winery，此时应该用count(distinct winery) 去重：
②将region_1和region_2合并union格式，并进行去重：
SELECT
    *
FROM
(SELECT a FROM A
UNION
SELECT b FROM B
WHERE a != b) t
```

```
--# of B(wineries) outside of Spain that produce wines with balackberry
taste(description)
--group the count by country and region
--output: country,  region, n_bodegas

SELECT
    country,
    region,
    COUNT(DISTINCT winery) AS n_bodegas
FROM
    (SELECT
        country,
        region_1 AS region,
        winery,
        description
    FROM winemag_p1
    UNION ALL
    SELECT
        country,
        region_2 AS region,
        winery,
        description
    FROM winemag_p1
    WHERE region_1 != region_2) t         --去除掉Region为空的部分，以及两个
region名称相同的部分
WHERE winery ILIKE '%bodega%'
AND country != 'Spain'
AND description ILIKE '%blackberry%'
GROUP BY 1,2
ORDER BY 3 DESC
```

Price Of Wines In Each Country

Interview Question Date: February 2020

Wine Magazine Medium ID 10029

Find the minimum, average, and maximum price of all wines per country. Assume all wines listed across both datasets are unique. Output the country name along with the corresponding minimum, maximum, and average prices.

①看清是否需要合并两个数据集，如果数据集的记录没有重叠，无需去重，直接使用union all；如果有需要去重的需要使用union
②转换格式：::NUMERIC, ::TEXT
③求最大平均最小值：min, max, avg

```
--min, avg, max price of all wines per country
--all wines unique
--output: country, min_price, avg_price, max_price

SELECT
    country,
    MIN(price) AS min_price,
    AVG(price) AS avg_price,
    MAX(price) AS max_price
FROM
    (SELECT
        country:: TEXT,
        price:: NUMERIC
    FROM winemag_p1
    UNION ALL
    SELECT
        country:: TEXT,
        price:: NUMERIC
    FROM winemag_p2) t
GROUP BY 1
ORDER BY 1
```

Find the number of wines each taster tasted within the variation

Interview Question Date: February 2020

Wine Magazine Medium ID 10028

Find the number of wines each taster tasted within the variation. Output the tester's name, variety, and the number of tastings. Order records by taster name and the variety in ascending order and by the number of tasting in descending order.

Table: winemag_p2

```
--# wines ~ taster ~ winthin variation
--output: taster_name, variety, n_tastings
--order by taster_name, variety ASCC, n_tastings DESC

SELECT
    taster_name,
    variety,
    COUNT(*) AS n_tastings
FROM winemag_p2
```

```
WHERE taster_name IS NOT NULL
GROUP BY
    taster_name,
    variety
ORDER BY
    taster_name, variety ASC,
    COUNT(*) DESC
```

Find the number of US-based wineries that have expensive wines (price >= 200)

Wine Magazine Easy ID 10027

Find the number of US-based wineries that have expensive wines. A wine is considered to be expensive if its price is $200 or more.

Table: winemag_p1

```
当记录中，有重复名字的A，使用COUNT(*)无法去重，应该使用COUNT(DISTINCT A)
```

```
--# US based wineries ~ expensive wines
--wine expensive if price >= 200
--output: n_wineries

SELECT
    COUNT(DISTINCT winery) AS n_wineries
FROM winemag_p1
WHERE price >= 200
AND country = 'US'
```

Find all wineries which produce wines by possessing aromas of plum, cherry, rose, or hazelnut

Wine Magazine Medium ID 10026

Find all wineries which produce wines by possessing aromas of plum, cherry, rose, or hazelnut. To make it more simple, look only for singular form of the mentioned aromas. HINT: if one of the specified words is just a substring of another word, this should not be a hit, but a miss.

Example Description: Hot, tannic and simple, with cherry jam and currant flavors accompanied by high, tart acidity and chile-pepper alcohol heat. Therefore the winery Bella Piazza is expected in the results.

Table: winemag_p1

```
lower(a) ~ '\y(A|B|C|D)\y': a中仅包含ABCD的记录，不包含ABCD的衍生词
因为description中包含的不是关键词，而是含有关键词的词，应该被去除：eg. prosecco
(rose), plump (plum), plummy (plum)
```

用ILIKE无法实现去除功能，'\y(A|B|C|D)\y': 确保我们匹配的词汇是一个完整的单词，不是部分匹配。

```
SELECT DISTINCT winery
FROM winemag_p1
WHERE lower(description) ~ '\y(plum|cherry|rose|hazelnut)\y'
```

```
--ILIKE错误方法，无法排除 specified words is just a substring of another word 的情况：Carpene Malvolti, Finca El Origen, La Fiammenga
-- SELECT
--     DISTINCT winery
-- --    description
-- FROM winemag_p1
-- WHERE description LIKE '%plum%'
-- or description LIKE '%cherry%'
-- or description LIKE '%rose%'
-- or description LIKE '%hazelnut%'


--错误原因：
-- Carpene Malvolti：  Prosecco
-- Finca El Origen：plump
-- La Fiammenga：plummy
-- select * from winemag_p1
-- where winery IN ('Carpene Malvolti', 'Finca El Origen', 'La Fiammenga')
```

Find all possible varieties which occur in either of the winemag datasets

Wine Magazine Medium ID 10025

Find all possible varieties which occur in either of the winemag datasets. Output unique variety values only. Sort records based on the variety in ascending order.

Tables: winemag_p1, winemag_p2

```
--all possible varieties  ~ in either winemag
--output: variety：unique variety values
--sort by variety asc
--将variety从每个表中选出，再distinct 效果等同于distinct variety再union
```

```
SELECT
    DISTINCT variety
FROM
    (SELECT
        variety
    FROM winemag_p1
    UNION ALL
    SELECT
        variety
    FROM winemag_p2) t
```

```
ORDER BY variety
```

Wine varieties tasted by 'Roger Voss'

Wine Magazine Easy ID 10024

Find wine varieties tasted by 'Roger Voss' and with a value in the 'region_1' column of the dataset. Output unique variety names only.

Table: winemag_p2

```
SELECT
    DISTINCT variety
FROM winemag_p2
WHERE taster_name = 'Roger Voss'
AND region_1 IS NOT NULL
```

Find all wine varieties which can be considered cheap based on the price

Wine Magazine Easy ID 10022

Find all wine varieties which can be considered cheap based on the price. A variety is considered cheap if the price of a bottle lies between 5 to 20 USD. Output unique variety names only.

Table: winemag_p1

```
--all wine varieties ~ cheap ~ price
--a vareity cheap: 5 < price < 20
--output: unique variety names


SELECT
    DISTINCT variety
FROM winemag_p1
WHERE price BETWEEN 5 AND 20
```

Find all top-rated wineries based on points

Wine Magazine Easy ID 10021

Find all top-rated wineries based on points. Consider a top-rated winery has been awarded points more or equal than 95.

Table: winemag_p1

```
--all top rated wineries ~ points
--top rated winery awarded points more or equal than 95
```

```
SELECT
    DISTINCT winery
FROM winemag_p1
WHERE points >= 95
```

Find prices for Spanish, Italian, and French wines

Wine Magazine Easy ID 10020

Find prices for Spanish, Italian, and French wines. Output the price.

Table: winemag_p1

```
SELECT
    price
FROM winemag_p1
WHERE country IN ('Spain', 'Italy', 'France')
```

Find the fraction of rides for each weather and the hour

Interview Question Date: February 2020

Lyft Hard ID 10019

Find the fraction (percentage divided by 100) of rides each weather-hour combination constitutes among all weather-hour combinations. Output the weather, hour along with the corresponding fraction.

Table: lyft_rides

①计算部分与整数的比值:
法1: i.共计数cte ii. part数cte, part left join total ON TRUE, 即可以直接使用所有变量
法2: i.共计数cte, ii. part数cte, 在part cte中如用total数, 可以直接写select A from total
②算probablity遇到都是0的情况, ::NUMERIC 或 ::DECIMAL 可以转换为小数

```
WITH total AS(
SELECT
    COUNT(*) AS all_counts
FROM lyft_rides
),
part AS (
SELECT
    weather,
    hour,
    COUNT(*) AS part_counts
FROM lyft_rides
```

```
GROUP BY 1,2
)
--法1:
SELECT
    weather,
    hour,
    part_counts / all_counts :: DECIMAL
FROM part LEFT JOIN total
ON TRUE
ORDER BY 1,2
--法2:
-- SELECT
--     weather,
--     hour,
--     part_counts / (SELECT all_counts FROM total)::DECIMAL AS probability
-- FROM part
-- ORDER BY 1,2
```

Churn Rate Of Lyft Drivers

Lyft Medium ID 10016 17

Data Engineer Data Scientist BI Analyst Data Analyst Find the global churn rate of Lyft drivers across all years. Output the rate as a ratio.

Table: lyft_drivers

```
churn rate = end_record / all_record
```

```
--global churn rate ~ lyft drivers all years
--output: global_churn_rate: rate

--select * from lyft_drivers;

SELECT
    COUNT(end_date) / COUNT(*) ::DECIMAL AS global_churn_rate
FROM lyft_drivers
```

Lyft Driver Salary And Service Tenure

Interview Question Date: February 2020

Lyft Hard ID 10018

Find the correlation between the annual salary and the length of the service period of a Lyft driver.

Table: lyft_drivers

```
① 现在的日期/ A日期 减去 B日期 算时长:
 i. (COALESCE(CURRENT_DATE, A::DATE) - B::DATE)::DECIMAL AS duration
 ii. DATEDIFF(COALESCE(CURRENT_DATE, A::DATE), B::DATE) AS duration
```

```
② 两者关系corr: CORR(A, B)
```

```
--correlation bt annual salary and the length of the service period of a
driver
--output: corr
```

```
WITH CTE AS(
select
    yearly_salary,
    COALESCE(end_date, CURRENT_DATE) - start_date AS Duration
    --DATEDIFF(COALESCE(end_date, current_date), start_date) AS duration
from lyft_drivers
)
SELECT
CORR(yearly_salary, duration)
FROM CTE
```

Year Over Year Churn

Interview Question Date: February 2020

Lyft Hard ID 10017

Find how the number of drivers that have churned changed in each year compared to the previous one. Output the year (specifically, you can use the year the driver left Lyft) along with the corresponding number of churns in that year, the number of churns in the previous year, and an indication on whether the number has been increased (output the value 'increase'), decreased (output the value 'decrease') or stayed the same (output the value 'no change').

Table: lyft_drivers

```
①计算当年及前一年的值（LAG&COLEASE的替换用法）：
i. COUNT(*) AS curr, COALESCE(LAG(A, 1) OVER (ORDER BY B), 0) AS prev
ii. COUNT(*) AS curr, LAG(COUNT(*), 1, '0') OVER (ORDER BY B) AS prev

②年份提取:
i. EXTRACT(YEAR FROM A)
ii. DATE_PART('YEAR', A)

③CASE WHEN中 ELSE 与 WHEN替换写法:
CASE WHEN a > b THEN c
     WHEN a < b THEN d
     ELSE e
END
CASE WHEN a > b THEN c
     WHEN a < b THEN d
     WHEN a = b THEN e
END
```

```
--法1: COUNT(*) AS curr, COALESCE(LAG(A, 1) OVER (ORDER BY B), 0) AS prev
WITH year_churn AS(
SELECT
    DATE_PART('YEAR', end_date::DATE) AS year_driver_churned,
```

```
    COUNT(*) AS n_churned,
    LAG(COUNT(*), 1, '0') OVER (
                                ORDER BY DATE_PART('YEAR', end_date::DATE))
AS n_churned_prev
FROM lyft_drivers
WHERE end_date IS NOT NULL
GROUP BY 1
ORDER BY 1
)
SELECT
    *,
      CASE
          WHEN n_churned > n_churned_prev THEN 'increase'
          WHEN n_churned < n_churned_prev THEN 'decrease'
          ELSE 'no change'
      END
FROM year_churn
```

```
--法2: COALESCE(LAG(A, 1) OVER (ORDER BY B), 0):
WITH year_churn AS(
SELECT
    EXTRACT(YEAR FROM end_date) AS year_driver_churned,
    COUNT(*) AS n_churned
FROM lyft_drivers
WHERE end_date IS NOT NULL
GROUP BY 1
ORDER BY 1
),
prev_churn AS(
SELECT
    year_driver_churned,
    n_churned,
    COALESCE(LAG(n_churned, 1) OVER (ORDER BY year_driver_churned), 0) AS
n_churned_prev
FROM year_churn
)
SELECT
    *,
    CASE
        WHEN n_churned - n_churned_prev > 0 THEN 'increase'
        WHEN n_churned - n_churned_prev = 0 THEN 'no change'
        WHEN n_churned - n_churned_prev < 0 THEN 'decrease'
END AS case
FROM prev_churn
```

Positive Ad Channels

Uber Hard ID 10013

Find the advertising channel with the smallest maximum yearly spending that still brings in more than 1500 customers each year.

Table: uber_advertising

```
--AVD CHANNEL ~ SMALLEST MAX YEARLY SPENDING BRINGS > 1500 customers
--output: advertising_channel: tv
--select * from uber_advertising;

--法1:
WITH total AS(
SELECT
    advertising_channel,
    SUM(money_spent) AS yearly_spending
FROM uber_advertising
GROUP BY 1
HAVING MIN(customers_acquired) > 1500                                    --每年有不同
channel, 不同channel每年获客数也不同, ∴min(A) > a
)
SELECT
    advertising_channel
FROM total
WHERE yearly_spending = (SELECT MIN(yearly_spending) FROM total)
```

```
--法2:
WITH CTE AS(
SELECT
    advertising_channel,
    MAX(money_spent) AS max_money_spent
FROM uber_advertising
WHERE customers_acquired > 1500
GROUP BY 1
),
CTE2 AS(
SELECT
    advertising_channel,
    max_money_spent,
    DENSE_RANK() OVER (ORDER BY max_money_spent ASC) AS rnk
FROM CTE
)
SELECT
    advertising_channel
FROM CTE2
WHERE rnk = 1
```

Find all number pairs whose first number is smaller than the second one and the product of two numbers is larger than 11

Uber Delta Airlines Medium ID 10011

Find all number pairs whose first number is smaller than the second one and the product of two numbers is larger than 11. Output both numbers in the combination.

Table: transportation_numbers

```
①self join不需要自身pair: SELECT * FROM A JOIN B ON A.a != B.a
```

②self join后注意去重

```
--all num pairs ~ 1st num < 2nd; product of 2 numbers > 11
--output: num1, num2: both numbers


WITH total AS(
SELECT
    a.number AS num1,
    b.number AS num2
FROM transportation_numbers a
JOIN transportation_numbers b
ON a.index != b.index
)
SELECT
DISTINCT *
--num1 * num2 AS product
FROM total
WHERE num1 < num2
AND num1 * num2 > 11
ORDER BY 1
```

Advertising Channel Effectiveness

Uber Medium ID 10012

Find the effectiveness of each advertising channel in the period from 2017 to 2018 (both included). The effectiveness is calculated as the ratio of total money spent to total customers aquired.

Output the advertising channel along with corresponding effectiveness. Sort records by the effectiveness in ascending order.

Table: uber_advertising

```
filter出两个值:
i. a IN ('A', 'B')
ii. a BETWEEN A AND B
```

```
--effectiveness of each adv channel 2017 - 2018
--effectiveness ~ ratio total money / total customers acquired

--output: advertising_channel, avg_effectiveness
--select * from uber_advertising;
WITH CTE AS(
SELECT
    advertising_channel,
    SUM(money_spent) AS total_money_spent,
    SUM(customers_acquired) AS  total_customers_aquired
FROM uber_advertising
--WHERE year IN ('2017', '2018')
WHERE year BETWEEN 2017 AND 2018
GROUP BY 1
```

```
)
SELECT
    advertising_channel,
    total_money_spent / total_customers_aquired ::NUMERIC AS
avg_effectiveness--::FLOAT --:: DECIMAL
FROM CTE
ORDER BY 2
```

Find The Combinations

Uber Lyft Medium ID 10010

Find all combinations of 3 numbers that sum up to 8. Output 3 numbers in the combination but avoid summing up a number with itself.

Table: transportation_numbers

In [ ]:
```
①3个元素self join且不能出现3个元素自身重复：
join时不能用index，因为有一个index对应多个num的情况
而要用a.num != b.num and b.num != c.num and a.num != c.num

②sum up to 8: 相加得8
```

```
--all combi 3 nums ~ sum up to 8
--output 3 nums; no sum up a num with itself

WITH total AS(
SELECT
    a.number AS num_1,
    b.number AS num_2,
    c.number AS num_3
FROM transportation_numbers a
JOIN transportation_numbers b
ON a.number != b.number
JOIN transportation_numbers c
ON b.number != c.number
AND a.number != c.number
)
SELECT
DISTINCT
    *
FROM total
WHERE num_1 + num_2 + num_3 = 8
ORDER BY 1, 2
```

Find the total costs and total customers acquired in each year

Uber Easy ID 10009

Find the total costs and total customers acquired in each year. Output the year along with corresponding total money spent and total acquired customers.

Table: uber_advertising

```
--total cost and customers acquired each yr
--output: year, total_money_spent, total_customers_acquired

--select * from uber_advertising;

SELECT
    year,
    SUM(money_spent) AS total_money_spent,
    SUM(customers_acquired) AS total_customers_acquired
FROM uber_advertising
GROUP BY year
```

Sum Of Numbers

Uber Tesla Medium ID 10008

Find the sum of numbers whose index is less than 5 and the sum of numbers whose index is greater than 5. Output each result on a separate row.

Table: transportation_numbers

```
法1：一行两列换为一列两行： case when -> union (all)
法2：直接sum + where + union all
```

```
--法1：一行两列换为一列两行： sum + case when  -> union (all)
WITH total AS(
SELECT
    SUM(CASE WHEN index < 5 THEN number END) AS sum_1,
    SUM(CASE WHEN index > 5 THEN number END) AS sum_2
FROM transportation_numbers
)
SELECT
    sum_1 AS sum
FROM total
UNION ALL
SELECT
    sum_2 AS sum
FROM total
```

```
--法2：直接sum + where + union all
SELECT
    SUM(number) AS sum
FROM transportation_numbers
WHERE index < 5
UNION ALL
SELECT
    SUM(number) AS sum
FROM transportation_numbers
```

```
WHERE index > 5
```

Average Cost Of Each Request

Uber Easy ID 10007

Find the average cost of each request status. Request status can be either 'success' or 'fail'. Output the request status along with the average cost.

Table: uber_ride_requests

```
-- avg cost of each request status
-- request status ~ 'success' or 'fail'
-- output: request_status, average_cost

--select * from uber_ride_requests;
SELECT
    request_status,
    AVG(monetary_cost)
FROM uber_ride_requests
GROUP BY request_status
```

Find the average distance traveled in each hour

Lyft Easy ID 10006

Find the average distance traveled in each hour. Output the hour along with the corresponding average traveled distance. Sort records by the hour in ascending order.

Table: lyft_rides

```
--select * from lyft_rides;
SELECT
    hour,
    AVG(travel_distance) AS average_distance_traveled
FROM lyft_rides
GROUP BY hour
ORDER BY hour
```

Hour Of Highest Gas Expense

Lyft Easy ID 10005

Find the hour with the highest gasoline cost. Assume there's only 1 hour with the highest gas cost.

Table: lyft_rides

In [ ]:
```sql
--法1: select max
SELECT
    hour
FROM lyft_rides
WHERE gasoline_cost = (SELECT MAX(gasoline_cost) FROM lyft_rides)
```

```sql
--法2: order by + limit 1
SELECT
    hour
FROM lyft_rides
ORDER BY gasoline_cost DESC
LIMIT 1
```

Find all Lyft rides which happened on rainy days before noon

Lyft Easy ID 10004

Find all Lyft rides which happened on rainy days before noon.

Table: lyft_rides

```sql
SELECT
    *
FROM lyft_rides
WHERE weather = 'rainy'
AND hour < 12
```

Lyft Driver Wages

Lyft Easy ID 10003

Find all Lyft drivers who earn either equal to or less than 30k USD or equal to or more than 70k USD. Output all details related to retrieved records.

Table: lyft_drivers

```sql
SELECT
    *
FROM lyft_drivers
WHERE yearly_salary < 30000 OR yearly_salary >= 70000
```

Find the advertising channel where Uber spent more than 100k USD in 2019

Uber Easy ID 10002

Find the advertising channel where Uber spent more than 100k USD in 2019.

Table: uber_advertising

```
SELECT
    DISTINCT
    advertising_channel
FROM uber_advertising
WHERE money_spent > 100000
AND year = '2019'
```

Find the cost per customer for advertising via public transport

Uber Easy ID 10001

Find the cost per customer for each advertising channel and year combination . Include only channels that are advertised via public transport (advertising channel includes "bus" substring). The cost per customer is equal to the total spent money divided by the total number of acquired customers through that advertising channel. Output advertising channel and its cost per customer.

Table: uber_advertising

```
SELECT
    advertising_channel,
    year,
    SUM(money_spent) / SUM(customers_acquired) AS cost_per_customer
FROM uber_advertising
WHERE advertising_channel ILIKE '%bus%'
GROUP BY advertising_channel, year
```

Find the cost per customer for advertising via public transport

Uber Easy ID 10001

Find the cost per customer for each advertising channel and year combination . Include only channels that are advertised via public transport (advertising channel includes "bus" substring). The cost per customer is equal to the total spent money divided by the total number of acquired customers through that advertising channel. Output advertising channel and its cost per customer.

Table: uber_advertising

```
SELECT
    advertising_channel,
    year,
    SUM(money_spent) / SUM(customers_acquired) AS cost_per_customer
FROM uber_advertising
WHERE advertising_channel ILIKE '%bus%'
```

```
GROUP BY advertising_channel, year
```

Find the year that Uber acquired more than 2000 customers through celebrities

Uber Easy ID 10000

Find the year that Uber acquired more than 2000 customers through advertising using celebrities.

Table: uber_advertising

```
SELECT
    year
FROM uber_advertising
WHERE customers_acquired > 2000
AND advertising_channel = 'celebrities'
```

Find songs that are ranked between 8-10

Spotify Easy ID 9999

Find songs that are ranked between 8-10. Output the track name along with the corresponding position, ordered ascendingly.

Table: spotify_worldwide_daily_song_ranking

```
--BETWEEN AND两端的值也包括在内
-- A BETWEEN 8 AND 10
-- A IN (8,9,10)
```

```
SELECT
    trackname,
    position
FROM spotify_worldwide_daily_song_ranking
WHERE position BETWEEN 8 AND 10
--WHERE position IN (8,9,10)
ORDER BY position ASC
```

Top 100 Ranked Songs

Spotify Easy ID 9997

Find the total number of streams for the top 100 ranked songs.

Table: spotify_worldwide_daily_song_ranking

```
SELECT
    SUM(streams) AS n_streams
```

```
FROM spotify_worldwide_daily_song_ranking
WHERE position <= 100
```

Find the average number of streams across all songs

Spotify Easy ID 9996

Find the average number of streams across all songs.

Table: spotify_worldwide_daily_song_ranking

```
SELECT
    AVG(streams)
FROM spotify_worldwide_daily_song_ranking;
```

Top 10 Ranked Songs

Spotify Easy ID 9995

Find the top 10 ranked songs by position. Output the track name along with the corresponding position and sort records by the position in descending order and track name alphabetically, as there are many tracks that are tied for the same position.

Table: spotify_worldwide_daily_song_ranking

```
去重:
i. distinct
ii. group by
在一个只选出两个变量的query中，也可以使用group by 1,2
```

```
-- TOP 10 ranked songs by position
-- output: trackname, position
-- order by position dec, name asc
-- !songs in same position

SELECT
    DISTINCT
    trackname,
    position
FROM spotify_worldwide_daily_song_ranking
WHERE position <= 10
--GROUP BY trackname, position
ORDER BY position DESC, trackname ASC
```

Find songs with less than 2000 streams

Spotify Easy ID 9994

Find songs with less than 2000 streams. Output the track name along with the corresponding streams. Sort records by streams in descending order. There is no need to group rows with same track name

Table: spotify_worldwide_daily_song_ranking

```
--songs < 2000 streams
--output: trackname, streams
--order by streams desc

--select * from spotify_worldwide_daily_song_ranking;

SELECT
    trackname,
    streams
FROM spotify_worldwide_daily_song_ranking
WHERE streams < 2000
ORDER BY streams DESC
```

Find artists with the highest number of top 10 ranked songs over the years

Spotify Medium ID 9993

Find artists with the highest number of top 10 ranked songs over the years. Output the artist along with the corresponding number of top 10 rankings.

Table: spotify_worldwide_daily_song_ranking

```
highest top 10:
因为一位artist这些年，多首top 10的歌可能是重复的，所以需要distinct去重
i. COUNT(DISTINCT *), RANK() OVER (ORDER BY COUNT(DISTINCT *) DESC); rnk = 1
ii. COUNT(DISTINCT *) as num; WHERE num = SELECT MAX(num) FROM total
```

```
--法1：

--select * from spotify_worldwide_daily_song_ranking;
--ARTISTIS ~ HIGHEST # top 10 ranked songs
--output: artist, no_top10

WITH total AS(
SELECT
    artist,
    COUNT(DISTINCT trackname) AS no_top10,
    RANK() OVER (ORDER BY COUNT(DISTINCT trackname) DESC) AS rnk
FROM spotify_worldwide_daily_song_ranking
WHERE position <= 10
GROUP BY 1
)
SELECT
    artist,
    no_top10
FROM total
```

```
WHERE rnk = 1
```

```
--法2:
WITH total AS(
SELECT
    artist,
    COUNT(DISTINCT trackname) AS no_top10
FROM spotify_worldwide_daily_song_ranking
WHERE position <= 10
GROUP BY 1
)
SELECT
    artist,
    no_top10
FROM total
WHERE no_top10 = (SELECT MAX(no_top10) FROM total)
```

Find how many times each artist appeared on the Spotify ranking list

Spotify Easy ID 9992

Find how many times each artist appeared on the Spotify ranking list Output the artist name along with the corresponding number of occurrences. Order records by the number of occurrences in descending order.

Table: spotify_worldwide_daily_song_ranking

```
--times ~ artist appeared on ranking
--output: artist, n_occurences
--order by # cc desc


SELECT
    artist,
    COUNT(*) AS n_occurences
FROM spotify_worldwide_daily_song_ranking
GROUP BY artist
ORDER BY COUNT(*) DESC
```

Top Ranked Songs

Spotify Medium ID 9991

Find songs that have ranked in the top position. Output the track name and the number of times it ranked at the top. Sort your records by the number of times the song was in the top position in descending order.

Table: spotify_worldwide_daily_song_ranking

```
--songs ~ ranked in the top position
```

```
--output: trackname, times_top1:  track name, the number of times it ranked
at the top
--order times_top1 desc

SELECT
    trackname,
    COUNT(*) AS times_top1
FROM spotify_worldwide_daily_song_ranking
WHERE position = 1
GROUP BY trackname
ORDER BY COUNT(*) DESC
```

Find songs that have more than 3 million streams

Spotify Easy ID 9990

Find songs that have more than 3 million streams. Output the track name, artist, and the corresponding streams. Sort records based on streams in descending order.

Table: spotify_worldwide_daily_song_ranking

```
SELECT
    trackname,
    artist,
    streams
FROM spotify_worldwide_daily_song_ranking
WHERE streams > 3000000
ORDER BY streams DESC
```

Highest Paid City Employees

City of San Francisco Hard ID 9989 15

Data Engineer Data Scientist BI Analyst Data Analyst Find the top 2 highest paid City employees for each job title. Output the job title along with the corresponding highest and second-highest paid employees.

Table: sf_public_salaries

①多个列多个行希望将结果并在一行时：max
②只要top2结果，用row_number，不能用rank/dense_rank，因为在同一rnk下会有多个结果，无法确认每次只选出top2
③totalpaybenefits和totalpay不确定用哪个变量时，可以都用一下，看哪个可以过
④case when中要有else null，因为有的分类下可能不一定有第二个结果

```
--top 2 highest paid city employees ~ job title
--output: jobtitle, best, second_best

WITH total AS(
SELECT
```

```
    jobtitle,
    employeename,
    totalpaybenefits,
    ROW_NUMBER() OVER (PARTITION BY jobtitle ORDER BY totalpaybenefits DESC)
AS row_num
FROM sf_public_salaries
)
SELECT
    jobtitle,
    MAX(CASE WHEN row_num = 1 THEN employeename ELSE NULL END) AS best,
    MAX(CASE WHEN row_num = 2 THEN employeename ELSE NULL END) AS second_best
FROM total
WHERE row_num <= 2
GROUP BY jobtitle
```

Find the top 5 least paid employees for each job title

City of San Francisco Hard ID 9986

Find the top 5 least paid employees for each job title. Output the employee name, job title and total pay with benefits for the first 5 least paid employees. Avoid gaps in ranking.

Table: sf_public_salaries

```
--top 5 least paid employees ~ job title
--output: employeename, jobtitle, totalpaybenefits

--select * from sf_public_salaries;
WITH total AS(
SELECT
    employeename,
    jobtitle,
    totalpaybenefits,
    ROW_NUMBER() OVER (PARTITION BY jobtitle ORDER BY totalpaybenefits ASC)
AS row_num
FROM sf_public_salaries
)
SELECT
    employeename,
    jobtitle,
    totalpaybenefits
FROM total
WHERE row_num <= 5
```

Overtime Pay

City of San Francisco Medium ID 9987

Find the employee who earned most from working overtime. Output the employee name.

Table: sf_public_salaries

```
求最问题:
i.法1: cte + max: SELECT * FROM A WHERE B = (SELECT MAX(B) FROM A)
ii.法2: cte rnk + rnk = 1
```

```
--法1: cte + max: SELECT * FROM A WHERE B = (SELECT MAX(B) FROM A)
WITH most AS(
SELECT
    employeename,
    overtimepay
FROM sf_public_salaries
)
SELECT
    employeename
FROM most
WHERE overtimepay = (SELECT max(overtimepay) FROM most)
```

```
--法2: cte rnk + rnk = 1
WITH most AS(
SELECT
    employeename,
    overtimepay,
    RANK() OVER (ORDER BY overtimepay DESC) AS rnk
FROM sf_public_salaries
)
SELECT
    employeename
FROM most
WHERE rnk = 1
```

Find the top 3 jobs with the highest overtime pay rate

City of San Francisco Easy ID 9988

Get the job titles of the 3 employees who received the most overtime pay Output the job title of selected records.

Table: sf_public_salaries

```
--job titles ~ 3 employees received most ot pay
--output: jobtitle
--select * from sf_public_salaries;
```

```
SELECT
    jobtitle
FROM sf_public_salaries
WHERE
    overtimepay IS NOT NULL AND overtimepay <> 0
ORDER BY overtimepay DESC
LIMIT 3
```

Above Average But Not At The Top

City of San Francisco Hard ID 9985

Find all people who earned more than the average in 2013 for their designation but were not amongst the top 5 earners for their job title. Use the totalpay column to calculate total earned and output the employee name(s) as the result.

Table: sf_public_salaries

```
--all ppl earned > avg in 2013 for designation, but not top 5 earners for job
title
--totalpay to calculate total earned, outt employee names
--output: employeename

--法1：portion_avg：AVG() OVER (PARTITION BY A)，可与portition_rank同用
WITH total AS(
SELECT
    jobtitle,
    employeename,
    totalpay,
    AVG(totalpay) OVER (PARTITION BY jobtitle) AS avg_totalpay,
    RANK() OVER (PARTITION BY jobtitle ORDER BY totalpay DESC) AS rnk
FROM sf_public_salaries
WHERE year = 2013
)
SELECT
employeename
FROM total
WHERE totalpay > avg_totalpay
AND rnk > 5
```

```
--法2：CTE：3个大逻辑：rank -> portion_not_top5; portion_avg ->
portion_avg_join_all_and_portion_not_top5; above_avg
WITH rank_total AS(
SELECT
    jobtitle,
    employeename,
    RANK() OVER (PARTITION BY jobtitle ORDER BY totalpay DESC) AS rnk
FROM sf_public_salaries
WHERE year = 2013
GROUP BY jobtitle, employeename, totalpay
),
not_top_5 AS(
SELECT
```

```
    jobtitle,
    employeename
FROM rank_total
WHERE rnk > 5
),
avg_totalpay_cte AS(
SELECT
    jobtitle,
    AVG(totalpay) AS avg_totalpay
FROM sf_public_salaries
WHERE year = 2013
GROUP BY jobtitle
),
join_not_top_5 AS(
SELECT
    employeename,
    totalpay,
    avg_totalpay
FROM sf_public_salaries a
JOIN avg_totalpay_cte b
ON a.jobtitle = b.jobtitle
WHERE employeename IN (SELECT employeename FROM not_top_5)
)
SELECT
    employeename
FROM join_not_top_5
WHERE totalpay > avg_totalpay
```

```
--法3：subquery写法，逻辑同法2，看起来更简洁
-- part1: avg; part2: main join avg; part3: rnk > 5
--part1:
SELECT
    employeename
FROM sf_public_salaries main
JOIN
    (SELECT
        jobtitle,
        avg(totalpay) AS avg_pay
    FROM sf_public_salaries
    WHERE year = 2013
    GROUP BY jobtitle) aves
ON main.jobtitle = aves.jobtitle
AND main.totalpay > aves.avg_pay
--part3:
WHERE main.employeename IN
--part2:
(SELECT employeename
FROM (
    SELECT
        employeename,
        jobtitle,
        totalpay,
        RANK() OVER (PARTITION BY jobtitle ORDER BY totalpay DESC) AS rnk
    FROM sf_public_salaries
    WHERE YEAR = 2013
) sq
```

```
WHERE rnk > 5)
```

Highest And Lowest Paying Jobs

City of San Francisco Medium ID 9984

Find the ratio and the difference between the highest and lowest total pay for each job title. Another condition is to remove rows total pay equal to zero from the calculation. Output the job title along with the corresponding difference, ratio, highest total pay, and the lowest total pay. Sort records based on the ratio in descending order.

Table: sf_public_salaries

```
min/ max可直接在query中加减乘除操作
```

```
--ratio and diff bt highest and lowest toal pay ~ job title
--remove total pay = 0
--output:jobtitle, difference, ratio: max/min, max_totalpay, min_totalpay
--order by ratio desc

--法1：CTE
WITH total AS(
SELECT
    jobtitle,
    MAX(totalpay) AS max_totalpay,
    MIN(totalpay) AS min_totalpay
FROM sf_public_salaries
WHERE totalpay != 0
GROUP BY jobtitle
)
SELECT
    jobtitle,
    max_totalpay - min_totalpay AS difference,
    max_totalpay / min_totalpay AS ratio,
    max_totalpay,
    min_totalpay
FROM total
ORDER BY ratio DESC
```

```
--法2：min/ max可直接在query中加减乘除操作

SELECT
    jobtitle,
    max(totalpay) - min(totalpay) AS difference,
    max(totalpay) / min(totalpay) AS ratio,
    max(totalpay) AS max_totalpay,
    min(totalpay) AS min_totalpay
FROM sf_public_salaries
WHERE
    totalpay > 0
GROUP BY
    jobtitle
ORDER BY
```

```
    ratio DESC
```

Median Job Salaries

City of San Francisco Hard ID 9983

Find the median total pay for each job. Output the job title and the corresponding total pay, and sort the results from highest total pay to lowest.

Table: sf_public_salaries

```
①求中位数median:
法1:
percentile_cont(0.5) within GROUP (ORDER BY totalpay) AS median_pay
法2:
ROW_NUMBER ASC ~ rnk_asc DESC ~ rnk_desc:
AVG(unit) WHERE rnk_asc IN (rnk_desc, rnk_desc - 1, rnk_desc + 1)
法3: ROW_NUMBER() ASC, COUNT(*); AVG(unit) + row_num IN ((total_row + 1) / 2,
(total_row + 2) / 2)

②ORDER BY A时，如果A是复合变量要写全，否则报错需要group by，group by出现新变量导致
结果错误
```

```
-- 法1: percentile_cont(0.5) within GROUP (ORDER BY totalpay) AS median_pay
SELECT
    jobtitle,
    percentile_cont(0.5) within GROUP (ORDER BY totalpay) AS median_pay
FROM sf_public_salaries
GROUP BY jobtitle
ORDER BY percentile_cont(0.5) within GROUP (ORDER BY totalpay) DESC
```

```
--法2:
--ROW_NUMBER ASC ~ rnk_asc DESC ~ rnk_desc:
--AVG(unit) WHERE rnk_asc IN (rnk_desc, rnk_desc - 1, rnk_desc + 1)

--ORDER BY A时，如果A是复合变量要写全，否则报错需要group by，group by出现新变量导
致结果错误
WITH total AS(
SELECT
    jobtitle,
    totalpay,
    ROW_NUMBER() OVER (PARTITION BY jobtitle ORDER BY totalpay ASC) AS
rnk_asc,
    ROW_NUMBER() OVER (PARTITION BY jobtitle ORDER BY totalpay DESC) AS
rnk_desc
FROM sf_public_salaries
)
SELECT
    jobtitle,
    AVG(totalpay)
FROM total
WHERE rnk_asc IN (rnk_desc, rnk_desc - 1, rnk_desc + 1)
GROUP BY jobtitle
```

```
ORDER BY AVG(totalpay) DESC
```

```
--法3: ROW_NUMBER() ASC, COUNT(*); AVG(unit) + row_num IN ((total_row + 1) /
2,  (total_row + 2) / 2)

SELECT
jobtitle, AVG(totalpay) AS median
FROM (SELECT jobtitle, totalpay,
             ROW_NUMBER() OVER(PARTITION BY jobtitle ORDER BY totalpay) AS
row_num,
             COUNT(*) OVER(PARTITION BY jobtitle) AS total_row
FROM sf_public_salaries) AS tmp
WHERE row_num IN ((total_row + 1) / 2,  (total_row + 2) / 2)
GROUP BY 1
ORDER BY 1,2 DESC;
```

Employees Without Benefits

City of San Francisco Hard ID 9981

Find the ratio between the number of employees without benefits to total employees. Output the job title, number of employees without benefits, total employees relevant to that job title, and the corresponding ratio. Order records based on the ratio in ascending order.

Table: sf_public_salaries

```
①SUM(CASE WHEN A THEN 1 ELSE 0 END): 算某个值是A即算一个，否则不算时用SUM。如果
用COUNT会出现和COUNT(*)相同的情况。
②without A要考虑两种情况: A = 0 OR A IS NULL
③算ratio要转换成小数格式: ::NUMERIC, ::DECIMAL, ::FLOAT
```

```
--ratio bt # ppl without benefit to total ppl
--output: jobtitle, no_employees_without_benefits, total_people, rto
--order by ratio asc

WITH total AS(
SELECT
    jobtitle,
    SUM(CASE WHEN benefits = 0 OR benefits IS NULL THEN 1 ELSE 0 END) AS
no_employees_without_benefits,
    COUNT(*) AS total_people
FROM sf_public_salaries
GROUP BY jobtitle
)
SELECT
    jobtitle,
    no_employees_without_benefits,
    total_people,
    no_employees_without_benefits / total_people::NUMERIC AS rto
FROM total
```

```
ORDER BY rto ASC
```

Employee With Lowest Pay

City of San Francisco Medium ID 9980

Find the employee who earned the lowest total payment with benefits from a list of employees who earned more from other payments compared to their base pay. Output the first name of the employee along with the corresponding total payment with benefits.

Table: sf_public_salarie

```
取第n个，用空格隔开的词组：LOWER(SPLIT_PART(A, ' ', n))：
```

In [ ]:
```
--employee ~ lowest total payment with benefits from a list of employees earne
-- output: lower, totalpaybenefits
--select * from sf_public_salaries

WITH tpb AS(
SELECT
    MIN(totalpaybenefits) AS min_tpb
FROM sf_public_salaries
WHERE otherpay > basepay
)
SELECT
    LOWER(SPLIT_PART(employeename, ' ', 1)),
    totalpaybenefits
FROM sf_public_salaries
WHERE totalpaybenefits = (SELECT min_tpb FROM tpb)
```

Find the top 5 highest paid and top 5 least paid employees in 2012

City of San Francisco Hard ID 9979

Find the top 5 highest paid and top 5 least paid employees in 2012. Output the employee name along with the corresponding total pay with benefits. Sort records based on the total payment with benefits in ascending order.

Table: sf_public_salaries

```
--①求top5和least5：
--(ORDER BY A DESC, LIMIT 5) UNION (ORDER BY A ASC, LIMIT 5)
--RANK() OVER (ORDER BY A DESC), RANK() OVER (ORDER BY A ASC); UNION + rnk
<=5 / BETWEEN 1 AND 5
--②UNION如需order by,仅在最底行写一次：这个ORDER BY 只在UNION后生效
```

```
--top 5 and least 5 paid in 2012
--output: employeename, totalpaybenefits
```

```
--order by totalpaybenefits asc

WITH total AS(
SELECT
    employeename,
    totalpaybenefits,
    RANK() OVER (ORDER BY totalpaybenefits DESC) AS rnk_desc,
    RANK() OVER (ORDER BY totalpaybenefits ASC) AS rnk_asc
FROM sf_public_salaries
WHERE year = 2012
)
SELECT
    employeename,
    totalpaybenefits
FROM total
WHERE rnk_desc BETWEEN 1 AND 5
UNION ALL
SELECT
    employeename,
    totalpaybenefits
FROM total
WHERE rnk_asc BETWEEN 1 AND 5
-- 这个ORDER BY 只在UNION后生效，且仅用写一次
ORDER BY totalpaybenefits ASC
```

Find employees who earned the highest and the lowest total pay without any benefits

City of San Francisco Medium ID 9978

Find employees who earned the highest and the lowest total pay without any benefits. Output the employee name along with the total pay. Order records based on the total pay in descending order.

Table: sf_public_salaries

```
UNION和WHERE OR 作用互换：SELECT * FROM a WHERE A UNION SELECT * FROM b WHERE B 和 WHERE A OR B 可以互换
```

```
-- employees earned highest and lowest total pay without benefits
-- output: employeename, totalpay
--order by total pay desc

--法1：
WITH CTE AS(
SELECT
    employeename,
    totalpay,
    RANK() OVER (ORDER BY totalpay DESC) AS rnk_desc,
    RANK() OVER (ORDER BY totalpay ASC) AS rnk_asc
FROM sf_public_salaries
WHERE benefits IS NULL OR benefits = 0                      --total pay without any benefits.
```

```
)
SELECT
    employeename,
    totalpay
FROM CTE
WHERE rnk_desc = 1
UNION
SELECT
    employeename,
    totalpay
FROM CTE
WHERE rnk_asc = 1
ORDER BY totalpay DESC
```

```
--法2:
WITH CTE AS(
SELECT
    employeename,
    totalpay,
    RANK() OVER (ORDER BY totalpay DESC) AS rnk_desc,
    RANK() OVER (ORDER BY totalpay ASC) AS rnk_asc
FROM sf_public_salaries
WHERE benefits IS NULL OR benefits = 0                      --total pay
without any benefits.
)
SELECT
    employeename,
    totalpay
FROM CTE
WHERE rnk_desc = 1 OR rnk_asc = 1
ORDER BY totalpay DESC
```

Find the number of police officers, firefighters, and medical staff employees

City of San Francisco Hard ID 9977

Find the number of police officers (job title contains substring police), firefighters (job title contains substring fire), and medical staff employees (job title contains substring medical) based on the employee name. Output each job title along with the corresponding number of employees.

Table: sf_public_salaries

```
重点是如果create的case when中有null值需要去除，再开一个cte，让where case when的变量名IS NOT NULL即可
```

```
--# police officers (job title has police), firefighters (job title has
firefighters), medical staff employees (job title has medical)
--~ employee name
--output: company, n_employees

--法1：先case when + count cte，再filter out null值数据
```

```
WITH total AS(
SELECT
    CASE
        WHEN jobtitle ILIKE '%fire%' THEN 'Firefighter'
        WHEN jobtitle ILIKE '%police%' THEN 'Police'
        WHEN jobtitle ILIKE '%medical%' THEN 'Medical'
    END AS company,
    COUNT(employeename) AS n_employees
FROM sf_public_salaries
GROUP BY company
)
SELECT
    company,
    n_employees
FROM total
WHERE company IS NOT NULL
```

```
--法2：先union所需数据，再case when + count
WITH total AS(
    SELECT
    *
    FROM sf_public_salaries
    WHERE jobtitle ILIKE '%police%'
    UNION ALL
    SELECT
    *
    FROM sf_public_salaries
    WHERE jobtitle ILIKE '%fire%'
    UNION ALL
    SELECT
    *
    FROM sf_public_salaries
    WHERE jobtitle ILIKE '%medical%'
)
SELECT
    CASE
        WHEN jobtitle ILIKE '%fire%' THEN 'Firefighter'
        WHEN jobtitle ILIKE '%police%' THEN 'Police'
        WHEN jobtitle ILIKE '%medical%' THEN 'Medical'
    END AS company,
    COUNT(employeename) AS n_employees
FROM total
GROUP BY company
```

METROPOLITAN TRANSIT AUTHORITY' Employees

City of San Francisco Easy ID 9975

Find all employees with a job title that contains 'METROPOLITAN TRANSIT AUTHORITY' and output the employee's name along with the corresponding total pay with benefits.

Table: sf_public_salaries

```
SELECT
    employeename,
    totalpaybenefits
FROM sf_public_salaries
WHERE jobtitle ILIKE '%METROPOLITAN TRANSIT AUTHORITY%'
```

Benefits Of Employees Called Patrick

City of San Francisco Easy ID 9974

Find benefits that people with the name 'Patrick' have. Output the full employee name along with the corresponding benefits.

Table: sf_public_salaries

```
SELECT
    employeename,
    benefits
FROM sf_public_salaries
WHERE employeename ILIKE '%Patrick%'
```

Quarterback With The Longest Throw

ESPN Hard ID 9966

Find the quarterback who threw the longest throw in 2016. Output the quarterback name along with their corresponding longest throw.

The 'lg' column contains the longest completion by the quarterback.

Table: qbstats_2015_2016

```
-- 提取变量中的数字:
-- i. SUBSTRING(A FROM '[0-9]+')::int
-- ii. SUBSTRING(A, '[0-9]+')::int
-- iii. SUBSTRING(A, '\d+')::int
-- iv. REGEXP_REPLACE(A, '[^0-9]+', '')::int
```

```
-- QB threw longest throw in 2016
-- output: qb, lg_num

WITH total AS(
SELECT
    qb,
    REGEXP_REPLACE(lg, '[^0-9]+', '')::int AS lg_num
FROM qbstats_2015_2016
WHERE year = '2016'
)
SELECT
    *
FROM total
```

```
WHERE lg_num = (SELECT MAX(lg_num) FROM total)
```

Top Teams In The Rio De Janeiro 2016 Olympics

ESPN Hard ID 9960

Find the top 3 medal-winning teams by counting the total number of medals for each event in the Rio De Janeiro 2016 olympics. In case there is a tie, order the countries by name in ascending order. Output the event name along with the top 3 teams as the 'gold team', 'silver team', and 'bronze team', with the team name and the total medals under each column in format "{team} with {number of medals} medals". Replace NULLs with "No Team" string.

Table: olympics_athletes_events

```
① 连接 变量 和 文字 或 变量和变量，在两者之间插入：||
②定义新position创建新列题思路：
i. 求medal team pair, count(*)
ii. ROW_NUMBER唯一性定义position，ORDER BY 后即按数字排序也按字母排序，使同一排名下的记录也有唯一性：ROW_NUMBER() OVER (PARTITION BY A ORDER BY B DESC, C ASC) AS position
iii. 定义不同列，合并项：CASE WHEN position = 1/2/3 THEN A || 'B' || C || 'D' ELSE NULL ；  winery || ' (' || round(avg_points) || ')' ELSE NUL
iv.让属于同一种类下，不同行的记录放在一行，无值赋值为no：A, MAX(B), COALESCE(MAX(C), 'no'), COALESCE(MAX(D), 'no') + GROUP BY A
```

```
--top 3 medal winning teams ~ total # medal each even in RDJ 2016 olympics
--in case tie, order countries by name asc.
--output:event, gold_team, silver_team, bronze_team:  {team} with {number of medals} medals

WITH total AS(
    SELECT
        event,
        team,
        COUNT(medal) AS cnt,
        ROW_NUMBER() OVER (PARTITION BY event ORDER BY COUNT(medal) DESC, team ASC) AS position
    FROM olympics_athletes_events
    WHERE city = 'Rio de Janeiro' AND year = '2016'
    AND medal IS NOT NULL
    GROUP BY event, team
    ORDER BY event, team
    ),
    total_2 AS(
    SELECT
        event,
        CASE
        WHEN position = 1 THEN  team || ' with ' || cnt || ' medals'
        ELSE NULL
        END AS gold_team,
        CASE
        WHEN position = 2 THEN  team || ' with ' || cnt || ' medals'
```

```
        ELSE NULL
        END AS silver_team,
        CASE
        WHEN position = 3 THEN  team || ' with ' || cnt || ' medals'
        ELSE NULL
        END AS bronze_team
    FROM total
    )
SELECT
    event,
    MAX(gold_team),
    COALESCE(MAX(silver_team), 'No team'),
    COALESCE(MAX(bronze_team), 'No team')
FROM total_2
GROUP BY event
```

Olympic Medals By Chinese Athletes

ESPN Hard ID 9959

Find the number of medals earned in each category by Chinese athletes from the 2000 to 2016 summer Olympics. For each medal category, calculate the number of medals for each olympic games along with the total number of medals across all years. Sort records by total medals in descending order.

Table: olympics_athletes_events

```
①一行中既有总数也有分列项：
法1：一步算出每年和所有年份总和：CASE WHEN A THEN COUNT(B) ELSE 0 END, COUNT(*)
AS total
CASE WHEN year THEN COUNT(medal) ELSE 0 END AS medal_year, COUNT(*) AS
total_medals
法2：行变列：先算出每年的奖牌数，再列出每年的奖牌数和总奖牌数
a, COUNT(*) AS cnt; SUM(CASE WHEN A THEN B ELSE 0 END), SUM(cnt) AS total
i. year, COUNT(*) AS n_medals
ii. SUM(CASE WHEN year THEN n_medals ELSE 0 END) AS medals_year,
SUM(n_medals)

②包含收尾时间点写法：BETWEEN AND, IN
BETWEEN 2000 AND 2016
IN (2000, 2004, 2008, 2012, 2016)

③SUM + CASE WHEN不同写法(注意group by不同)：
a, SUM(CASE WHEN A THEN B ELSE 0 END) AS C + GROUP BY a
a, CASE WHEN A THEN SUM(B) ELSE 0 END AS C + GROUP BY a, A

④ORDER BY 后可以接新创建的变量名
```

```
--# medals earned ~ category ~ Chinese athletes 2000 - 2016 summer olympics
--output: medal medals_2000 medals_2004 medals_2008 medals_2012 medals_2016
total_medals：
--each medal category, # medals ~ each olympic games, total # medals
--order by total medals desc
```

```
-- 法1：一步算出每年和所有年份总和：CASE WHEN A THEN COUNT(B) ELSE 0 END,
COUNT(*) AS total
--CASE WHEN year THEN COUNT(medal) ELSE 0 END AS medal_year, COUNT(*) AS
total_medals

SELECT
    medal,
    CASE WHEN year = 2000 THEN COUNT(medal) ELSE 0 END AS medals_2000,
    CASE WHEN year = 2004 THEN COUNT(medal) ELSE 0 END AS medals_2004,
    CASE WHEN year = 2008 THEN COUNT(medal) ELSE 0 END AS medals_2008,
    CASE WHEN year = 2012 THEN COUNT(medal) ELSE 0 END AS medals_2012,
    CASE WHEN year = 2016 THEN COUNT(medal) ELSE 0 END AS medals_2016,
    COUNT(*) AS total_medals
FROM olympics_athletes_events
WHERE team = 'China'
AND year BETWEEN 2000 AND 2016
GROUP BY medal, year
ORDER BY COUNT(*) DESC
```

```
--法2：行变列：先算出每年的奖牌数，再列出每年的奖牌数和总奖牌数
a, COUNT(*) AS cnt; SUM(CASE WHEN A THEN B ELSE 0 END), SUM(cnt) AS total
i. year, COUNT(*) AS n_medals
ii. SUM(CASE WHEN year THEN n_medals ELSE 0 END) AS medals_year,
SUM(n_medals)

WITH total AS(
SELECT
    year,
    medal,
    COUNT(*) AS n_medals
FROM olympics_athletes_events
WHERE team = 'China'
AND year IN (2000, 2004, 2008, 2012, 2016) AND medal IS NOT NULL
GROUP BY year, medal
)
SELECT
    medal,
    SUM(CASE WHEN year = 2000 THEN n_medals ELSE 0 END) AS medals_2000,
    SUM(CASE WHEN year = 2004 THEN n_medals ELSE 0 END) AS medals_2004,
    SUM(CASE WHEN year = 2008 THEN n_medals ELSE 0 END) AS medals_2008,
    SUM(CASE WHEN year = 2012 THEN n_medals ELSE 0 END) AS medals_2012,
    SUM(CASE WHEN year = 2016 THEN n_medals ELSE 0 END) AS medals_2016,
    SUM(n_medals) AS total_medals
FROM total
GROUP BY medal
ORDER BY total_medals DESC
```

Find how the average male height changed between each Olympics from 1896 to 2016

ESPN Hard ID 9957

Find how the average male height changed between each Olympics from 1896 to 2016. Output the Olympics year, average height, previous average height, and the corresponding average height difference. Order records by the year in ascending order.

If avg height is not found, assume that the average height of an athlete is 172.73

```
--1. 清洗数据：保证每个人的唯一性，要保留id；因为数据有重复，要保证每个人只计算一次，用DISTINCT
--2.填充数据：COALESCE(AVG(), A)：如果AVG没有，就替换为A；
--3. 计算数据：算prev, diff
--4. 两个COALESCE 可以在创建的cte中相减
```

```
--1. 清洗数据：保证每个人的唯一性，要保留id；因为数据有重复，要保证每个人只计算一次，用DISTINCT
WITH total AS(
SELECT
    DISTINCT
    year,
    id,
    height
FROM olympics_athletes_events
WHERE year BETWEEN 1896 AND 2016 AND sex = 'M'
),
--2.填充数据：COALESCE(AVG(), A)：如果AVG没有，就替换为A；计算数据：算prev, diff
--两个COALESCE 可以在创建的cte中相减
prev_height AS(
SELECT
    year,
    COALESCE(AVG(height), 172.73) AS avg_height,
    COALESCE(LAG(AVG(height), 1) OVER (ORDER BY year ASC), 172.73) AS prev_avg_height,
    COALESCE(AVG(height), 172.73) - COALESCE(LAG(AVG(height), 1) OVER (ORDER BY year ASC), 172.73) AS avg_height_diff
FROM total
GROUP BY year
)
SELECT *
FROM prev_height
```

Norwegian Alpine Skiers

ESPN Hard ID 9955

Find all Norwegian alpine skiers who participated in 1992 but didn't participate in 1994. Output unique athlete names.

Table: olympics_athletes_events

```
除去某个部分：
--法1：NOT IN
```

```
--法2：除去某个部分：EXCEPT：SELECT A FROM B WHERE C EXCEPT SELECT A FROM D
WHERE E
```

```
--法1：除去某个部分：NOT IN
WITH not_in AS(
SELECT
    *
FROM olympics_athletes_events
WHERE year = 1994
)
SELECT
    DISTINCT name
FROM olympics_athletes_events
WHERE year = 1992
    AND sport = 'Alpine Skiing'
    AND team = 'Norway'
AND name NOT IN (SELECT name FROM not_in)
ORDER BY name DESC
```

```
--法2：除去某个部分：EXCEPT：SELECT A FROM B WHERE C EXCEPT SELECT A FROM D
WHERE E
SELECT
    DISTINCT name
FROM olympics_athletes_events
WHERE year = 1992
    AND sport = 'Alpine Skiing'
    AND team = 'Norway'
EXCEPT
SELECT name
FROM olympics_athletes_events
WHERE year = 1994
```

Olympics Gender Ratio

ESPN Hard ID 9953

Find the gender ratio between the number of men and women who participated in each Olympics. Output the Olympics name along with the corresponding number of men, women, and the gender ratio. If there are Olympics with no women, output a NULL instead of a ratio.

Table: olympics_athletes_events

```
①原数据库有重复数据影响COUNT，一定先DISTINCT去重，
i. SELECT DISTINCT 所需变量
ii. SELECT DISTINCT A, COUNT(DISTINCT CASE WHEN B THEN C END) + GROUP BY A
②SUM(CASE WHEN A THEN 1 ELSE 0 END) AS cnt
③A / B出现0.X显示不出来：A::NUMERIC, A::float, A * 1.00
④分母B有0无法除的情况：A::NUMERIC / NULLIF(B, 0)
```

```
-- gender ratio BT # men and women in each olympics
-- output: games    total_men   total_women gender_ratio
-- no women, output a NULL instead of a ratio.
```

```
WITH distinct_values AS(
SELECT
DISTINCT
    games,
    id,
    sex
FROM olympics_athletes_events
),
values AS(
SELECT
    games,
    SUM(CASE WHEN sex = 'M' THEN 1 ELSE 0 END) AS total_men,
    SUM(CASE WHEN sex = 'F' THEN 1 ELSE 0 END) AS total_women
FROM distinct_values
GROUP BY games
)
SELECT
    games,
    total_men,
    total_women,
    total_men::NUMERIC / NULLIF(total_women, 0) AS gender_ratio
FROM values
```

Name to Medal Connection

ESPN Hard ID 9952

Find the connection between the number of letters in the athlete's first name and the number of medals won for each type for medal, including no medals. Output the length of the name along with the corresponding number of no medals, bronze medals, silver medals, and gold medals.

Table: olympics_athletes_events

```
取第n个，用空格隔开的词组：LOWER(SPLIT_PART(A, ' ', n))
数第1个，用空格隔开的词组，有几个字母：LENGTH(SPLIT_PART(A, ' ', 1)) AS
length_of_name,
```

```
-- # letters in first name AND # medals for each type for medal(including no
medals)
-- output: length_of_name   no_medals   bronze_medals   silver_medals
gold_medals

SELECT
    LENGTH(SPLIT_PART(name, ' ', 1)) AS length_of_name,
    SUM(CASE
            WHEN medal IS NULL THEN 1
            ELSE 0
        END) AS no_medals,
    SUM(CASE
            WHEN medal = 'Bronze' THEN 1
            ELSE 0
        END) AS bronze_medals,
    SUM(CASE
            WHEN medal = 'Silver' THEN 1
```

```
            ELSE 0
        END) AS silver_medals,
    SUM(CASE
            WHEN medal = 'Gold' THEN 1
        ELSE 0
        END) AS gold_medals
FROM olympics_athletes_events
GROUP BY 1
```

Unique Highest Salary

Interview Question Date: May 2019

Salesforce LinkedIn Hard ID 9919

Find the highest salary among salaries that appears only once.

Table: employee

```
求只出现一次的值中最大的值:
法1: cte: A, COUNT(*), GROUP BY A; MAX(A) + cnt = 1
法2: subquery: A, GROUP BY 1, HAVING COUNT(A) = 1; MAX(A)
法3: A, GROUP BY A, HAVING COUNT(*) = 1, LIMIT 1
法4: MAX(A) FROM (SELECT DISTINCT A FROM B) t
```

```
--法1: cte: A, COUNT(*), GROUP BY A; MAX(A) + cnt = 1
WITH cnt_cte AS(
SELECT
    salary,
    COUNT(*) AS cnt
FROM employee
GROUP BY salary
)
SELECT
    MAX(salary) AS max_salary
FROM cnt_cte
WHERE cnt = 1
```

```
--法2: subquery: A, GROUP BY 1, HAVING COUNT(A) = 1; MAX(A)
SELECT
    MAX(salary) AS max_salary
FROM
(
SELECT
    salary
FROM employee
GROUP BY salary
HAVING COUNT(salary) = 1
) AS t
```

```
--法3: A, GROUP BY A, HAVING COUNT(*) = 1, LIMIT 1
SELECT
    salary
```

```
FROM employee
GROUP BY salary
HAVING COUNT(*) = 1
ORDER BY salary DESC
LIMIT 1
```

```
--法4: MAX(A) FROM (SELECT DISTINCT A FROM B) t
SELECT
MAX(salary)
FROM (SELECT DISTINCT salary FROM employee) t
```

Median Salary

Interview Question Date: April 2019

Walmart The Honest Company Twitter Hard ID 9900

Find the median employee salary of each department. Output the department name along with the corresponding salary rounded to the nearest whole dollar.

Table: employe

```
中位数:
--法1: A, PENCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY B) + GROUP BY A
--法2: row_desc IN (row_asc, row_asc + 1, row_asc - 1)
--法3: ROW_NUMBER() OVER (PARTITION BY A ORDER BY B) AS row_num, COUNT(*)
OVER (PARTITION BY A) AS total_row: WHERE row_num IN ((total_row + 1)/2,
(total_row + 2)/2): 奇数个: 取中间1个; 偶数个: 取中间相邻2个
--法4: ABS(asc_n - desc_n) <= 1: 奇数个: row_num相等; 偶数个: row_num差1
```

```
--median employee salary ~ department
--output: department, median_salary: rounded to the nearest whole dollar.

--法1: A, PENCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY B) + GROUP BY A
SELECT
    department,
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY salary) AS median_sal
FROM employee
GROUP BY department
```

```
--法2: row_desc IN (row_asc, row_asc + 1, row_asc - 1)
WITH total AS(
SELECT
    department,
    salary,
    ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC) AS
row_desc,
    ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary ASC) AS
row_asc
FROM employee
)
SELECT
    department,
```

```
    ROUND(AVG(salary)) AS median_salary
FROM total
WHERE row_desc IN (row_asc, row_asc + 1, row_asc - 1)
GROUP BY department
```

```
--法3: ROW_NUMBER() OVER (PARTITION BY A ORDER BY B) AS row_num, COUNT(*)
OVER (PARTITION BY A) AS total_row: WHERE row_num IN ((total_row + 1)/2,
(total_row + 2)/2): 奇数个: 取中间1个; 偶数个: 取中间相邻2个
WITH total AS(
SELECT
    department,
    salary,
    ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary) AS row_num,
    COUNT(*) OVER (PARTITION BY department) AS total_row
FROM employee
)
SELECT
    department,
    ROUND(AVG(salary)) AS avg_sal
FROM total
WHERE row_num IN ((total_row + 1) / 2, (total_row + 2) / 2)
GROUP BY department
```

```
--法4: ABS(asc_n - desc_n) <= 1: 奇数个: row_num相等; 偶数个: row_num差1
WITH meds AS
(
    SELECT department,
        salary,
        ROW_NUMBER() OVER(PARTITION BY department ORDER BY salary ASC) as
asc_n,
        ROW_NUMBER() OVER(PARTITION BY department ORDER BY salary DESC) as
desc_n
    FROM employee
)
SELECT
    department,
    ROUND(AVG(salary)) as salary
FROM meds
WHERE ABS(asc_n - desc_n) <= 1
GROUP BY 1
```

Distinct Salaries

Interview Question Date: April 2019

Twitter Hard ID 9898

Find the top three distinct salaries for each department. Output the department name and the top 3 distinct salaries by each department. Order your results alphabetically by department and then by highest salary to lowest.

Table: twitter_employee

```
--top 3 distinct salaries ~ department
--output: distinct department, salary top 3
--order by departmetn asc, salary desc
WITH total AS(
SELECT
    DISTINCT
    department,
    salary
FROM twitter_employee

),
total_row_num AS(
SELECT
    department,
    salary,
    ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC) AS
row_num   --DISTINCT后, ROW_NUMBER与RANK作用相同
FROM total
)
SELECT
    department,
    salary
FROM total_row_num
WHERE row_num <= 3
ORDER BY department ASC, salary DESC
```

Find the oldest survivor per passenger class

Google Hard ID 9883

Find the oldest survivor of each passenger class. Output the name and the age of the survivor along with the corresponding passenger class. Order records by passenger class in ascending order.

Table: titanic

```
--oldest suvivor ~ passenger class
--output:pclass name    age
--order passenger class asc

--法1: RANK
WITH total AS(
SELECT
    pclass,
    name,
    age,
    RANK() OVER (PARTITION BY pclass ORDER BY age DESC) AS rnk
FROM titanic
WHERE age IS NOT NULL AND survived = 1
)
SELECT
    pclass,
    name,
```

```
    age
FROM total
WHERE rnk  = 1
```

```
--法2: SELECT B, C FROM A t JOIN (SELECT B, MAX(C) + GROUP BY B) tmp ON t.B =
tmp.B AND t.C = tmp.C
SELECT
    t.pclass,
    t.name,
    t.age
FROM titanic t
INNER JOIN
(
SELECT
    pclass,
    MAX(age) AS oldest_survivor_age
FROM titanic
WHERE survived = 1
GROUP BY pclass
) tmp
ON t.pclass = tmp.pclass
AND t.age = tmp.oldest_survivor_age
WHERE survived = 1
```

Common Letters

Interview Question Date: February 2019

Google Hard ID 9823

Find the top 3 most common letters across all the words from both the tables (ignore filename column). Output the letter along with the number of occurrences and order records in descending order based on the number of occurrences.

Tables: google_file_store, google_word_lists

```
--取用空格分开的词组: LOWER(UNNEST(STRING_TO_ARRAY(A, ' ')))
--取用逗号分开的词组: LOWER(UNNEST(STRING_TO_ARRAY(A, ',')))
--取连在一起的字母: UNNEST(REGEXP_SPLIT_TO_ARRAY(A, ''))
--算字母, 字母个数: A, COUNT(*)
```

```
--top 3 most common letters for all words from both tables
--output: letter, n_occurences

WITH total AS(
SELECT
    LOWER(UNNEST(STRING_TO_ARRAY(contents, ' '))) AS word
FROM google_file_store
UNION ALL
SELECT
    LOWER(UNNEST(STRING_TO_ARRAY(words1, ','))) AS word
FROM google_word_lists
UNION ALL
```

```
SELECT
    LOWER(UNNEST(STRING_TO_ARRAY(words2, ','))) AS word
FROM google_word_lists
),
word_to_letter AS(
SELECT
UNNEST(REGEXP_SPLIT_TO_ARRAY(word, '')) AS letter
FROM total
)
SELECT
    letter,
    COUNT(*) AS n_occurences
FROM word_to_letter
GROUP BY letter
ORDER BY n_occurences DESC
LIMIT 3
```

Find the average number of friends a user has

Google Hard ID 9822

Find the average number of friends a user has.

Table: google_friends_network

```
① 计算朋友数量的平均数:
i.计算所有的user和朋友: u1,u2 union u2, u1
每个user的朋友数: u1, count(*) AS cnt
平均数: AVG(cnt)
ii.计算所有的user和朋友: u1,u2 union u2, u1
平均数: COUNT(*) * 1.0 / COUNT(DISTINCT user_1) AS avg_fr
② 平均数逻辑: A, COUNT(B) AS C; AVG(C) =  SUM(C) / COUNT(A)
```

```
--法1:
WITH total AS(
SELECT
    user_id AS user_1,
    friend_id AS user_2
FROM google_friends_network
UNION
SELECT
    DISTINCT
    friend_id AS user_1,
    user_id AS user_2
FROM google_friends_network
ORDER BY user_1
),
friend_cnt_cte AS (
SELECT
    user_1,
    COUNT(user_2) AS friend_cnt
FROM total
GROUP BY user_1
```

```
)
SELECT
    AVG(friend_cnt) AS avg_fr
--    SUM(friend_cnt) / COUNT(user_1) AS avg_fr
FROM friend_cnt_cte
```

```
# --法2:
-- 计算所有的user和朋友: u1,u2 union u2, u1
-- 平均数: COUNT(*) * 1.0 / COUNT(DISTINCT user_1) AS avg_fr

WITH total AS(
SELECT
    user_id AS user_1,
    friend_id AS user_2
FROM google_friends_network
UNION
SELECT
    DISTINCT
    friend_id AS user_1,
    user_id AS user_2
FROM google_friends_network
ORDER BY user_1
)

SELECT
    -- COUNT(*) AS friend_cnt_sum,
    -- COUNT(DISTINCT user_1) AS user_cnt,
    COUNT(*) * 1.0 / COUNT(DISTINCT user_1) AS avg_fr
FROM total
```

Common Friends Friend

Interview Question Date: February 2019

Google Hard ID 9821

Find the number of a user's friends' friend who are also the user's friend. Output the user id along with the count.

Table: google_friends_network

In [ ]:
```
user朋友的朋友也是user的朋友的数量:

t1: user_id, friend_id
t2: user_id, friend_id
t3: user_id, friend_id

user_id, COUNT(DISTINCT friend_id)
distinct t1.user_id, t3.user_id AS friend_id
t1 join t2 on t1.friend_id = t1.user_id
   join t3 on t2.friend_id = t3.user_id
```

```
--# user's friends' friend is user's friend
--output: user_id   n_friends
```

```
WITH relationship AS(
SELECT
    user_id,
    friend_id
FROM google_friends_network
UNION
SELECT
    friend_id AS user_id,
    user_id AS friend_id
FROM google_friends_network
),
friend_friend AS(
SELECT
    DISTINCT
    a.user_id,
    c.user_id AS friend_id
FROM relationship a
JOIN relationship b ON a.friend_id = b.user_id
JOIN relationship c ON b.friend_id = c.user_id
AND c.friend_id = a.user_id
)
SELECT
    user_id,
    COUNT(DISTINCT friend_id) AS n_friends
FROM friend_friend
GROUP BY user_id
```

File Contents Shuffle

Interview Question Date: February 2019

Google Hard ID 9818

Sort the words alphabetically in 'final.txt' and make a new file named 'wacky.txt'. Output the file contents in one column and the filename 'wacky.txt' in another column. Lowercase all the words. To simplify the question, there is no need to remove the punctuation marks.

If coding in python, the file contents should be contained in a list.

Table: google_file_store

```
①创建一个名称：SELECT 'ABCD' AS name FROM E
②取用空格分开词组：LOWER(UNNEST(STRING_TO_ARRAY(A, ' ')))
③合并用空格分开的词组：ARRAY_TO_STRING(ARRAY_AGG(LOWER(A)), ' ')
```

```
--sort final.txt ASC -> wacky.txt
--output file contents in 1 column and filename wacky.txt
--lowercase all words
--no remove punc

WITH words AS(
SELECT
LOWER(UNNEST(STRING_TO_ARRAY(contents, ' '))) AS words
```

```
FROM google_file_store
WHERE filename = 'final.txt'
ORDER BY words ASC
)
SELECT
'wacky.txt' AS filename,
--ARRAY_AGG(LOWER(words)), ' '
ARRAY_TO_STRING(ARRAY_AGG(LOWER(words)), ' ') AS contents
FROM words
```

Find the list of intersections between both word lists

Google Hard ID 9816

Find the list of intersections between both word lists.

Table: google_word_lists

```
找到intersect共有部分:
法1: SELECT A FROM B WHERE A IN (SELECT C FROM D)
法2: SELECT A FROM B INTERSECT SELECT C FROM D
```

```
--法1: SELECT A FROM B WHERE A IN (SELECT C FROM D)
WITH total AS(
SELECT
LOWER(UNNEST(STRING_TO_ARRAY(words1, ','))) AS words1_sep,
LOWER(UNNEST(STRING_TO_ARRAY(words2, ','))) AS words2_sep
FROM google_word_lists
)
SELECT
    words1_sep AS word
FROM total
WHERE words1_sep IN (SELECT words2_sep FROM total)
ORDER BY words1_sep
```

```
--法2: SELECT A FROM B INTERSECT SELECT C FROM D
SELECT
    LOWER(UNNEST(STRING_TO_ARRAY(words1, ','))) AS word
FROM google_word_lists
INTERSECT
SELECT
    LOWER(UNNEST(STRING_TO_ARRAY(words2, ','))) AS word
FROM google_word_lists
```

Price Of A Handyman

Interview Question Date: January 2019

Google Hard ID 9815

Find the price that a small handyman business is willing to pay per employee. Get the result based on the mode of the adword earnings per employee distribution. Small businesses are considered to have not more than ten employees.

Table: google_adwords_earnings

```
①A / B 和 A / B :: FLOAT的区别是有无小数，更加精确的结果是::FLOAT，可能会在Mode结果中有区别
②找到A中的众数：MODE() WITHIN GROUP (ORDER A)
③adword earnings per employee distribution：adwords_earnings / n_employees
```

```
--price small handyman biz to pay per employee
--mode of the adword earnings per employee distribution
--small biz are considered to have <= 10 employees

SELECT
    MODE() WITHIN GROUP (ORDER BY adwords_earnings / n_employees::FLOAT) AS
price_willing_to_pay_per_employee
FROM google_adwords_earnings
WHERE business_type = 'handyman'
AND n_employees <= 10
```

Counting Instances in Text

Google Hard ID 9814

Software Engineer Find the number of times the words 'bull' and 'bear' occur in the contents. We're counting the number of times the words occur so words like 'bullish' should not be included in our count. Output the word 'bull' and 'bear' along with the corresponding number of occurrences.

Table: google_file_store

```
WITH total AS(
SELECT
    LOWER(UNNEST(STRING_TO_ARRAY(contents, ' '))) AS word
FROM google_file_store
)
SELECT
    word,
    COUNT(word)
FROM total
WHERE word IN ('bull', 'bear')
--    word ILIKE 'bull' or word ILIKE 'bear'
GROUP BY word
```

Average Time Between Steps

Meta/Facebook Hard ID 9793

Find the average time (in seconds), per product, that needed to progress between steps. You can ignore products that were never used. Output the feature id and the average time.

Table: facebook_product_features_realizations

①计算每个产品两个相邻阶段间的时间差平均值:
i. lag_cte: feature_id, user_id, 每相邻的两个step的时间点记录:
timestamp, prev_timestamp: timestamp::TIMESTAMP AS timestamp, LAG(timestamp, 1) OVER (PARTITION BY A, B ORDER BY C ASC):: TIMESTAMP AS prev_timestamp
ii. time_diff: feature_id, user_id, 每相邻的两个step的时间点差异: EXTRACT (EPOCH FROM timestamp - prev_timestamp)
iii. avg_time_per_user: feature_id, user_id, AVG(time_diff) AS avg_elapsed_time
iv. avg_time_per_product: feature_id, AVG(avg_elapsed_time)
②ignore products that were never used: WHERE prev_timestamp IS NOT NULL

③有B参与的求A平均值: 可以先计算有A,B, AVG(time) AS avg, 再计算A, AVG(avg)

```
-- the average time(seconds) it takes for a user to progress between steps
for each product
-- ignore products were never used
-- output: feature_id, avg: feature id, avg time

WITH lag_cte AS(
SELECT
    feature_id,
    user_id,
    step_reached,
    timestamp::TIMESTAMP AS timestamp,
    LAG(timestamp, 1) OVER (PARTITION BY feature_id, user_id ORDER BY
step_reached ASC):: TIMESTAMP AS prev_timestamp
FROM facebook_product_features_realizations
),
time_diff AS(
SELECT
    feature_id,
    user_id,
    timestamp,
    prev_timestamp,
    EXTRACT (EPOCH FROM timestamp - prev_timestamp) AS elapsed_time   --计算
两个timestamp间的秒数
FROM lag_cte
WHERE prev_timestamp IS NOT NULL                                      --
ignore products that were never used.
),
avg_time_per_user AS(
SELECT
    feature_id,
    user_id,
    AVG(elapsed_time) AS avg_elapsed_time
FROM time_diff
GROUP BY feature_id, user_id
)
SELECT
    feature_id,
    AVG(avg_elapsed_time)
FROM avg_time_per_user
```

```
GROUP BY feature_id
```

Views Per Keyword

Interview Question Date: January 2019

Meta/Facebook Hard ID 9791

Create a report showing how many views each keyword has. Output the keyword and the total views, and order records with highest view count first.

Tables: facebook_posts, facebook_post_views

```
①取含有[] # 且有逗号分隔的词组清洗数据：
UNNEST(STRING_TO_ARRAY(BTRIM(post_keywords, '[]#'), ',')) AS word

A：[#spam#], [basketball,lebron_james,nba]
BTRIM(A, '[]#')：spam, basketball,lebron_james,nba
string_to_array(BTRIM(A, '[]#'), ',')：["spam"],
["basketball","lebron_james","nba"]
unnest(string_to_array(BTRIM(A, '[]#'), ','))：spam, basketball,
lebron_james, nba

regexp_replace(business_name, '[^a-zA-Z0-9 ]', '', 'g')
array_length(regexp_split_to_array(b_name, '\s+'), 1)

②当数值没有时，填充没有数值为0：COALESCE(n_views, 0) AS n_views

③求每个post被view了几次：
i.法1：
连接post和view表，left join：不能用inner join，因为要保留没有在view表里出现的
post，他们的view_cnt为0
word, COUNT(*) AS n_views
ii. 法2：
先创建cte在views表中算每个post的view数：post_id, COUNT(*) AS n_views
然后将post表与cte表left join：word, COALESCE(n_views, 0)
最后要算每个word总view数：word, SUM(n_views)
```

```
--report showing how many views each keyword has
--output: keyword, total_views
--order by 2 desc

-- 法1：
-- 连接post和view表，left join：不能用inner join，因为要保留没有在view表里出现的
post，他们的view_cnt为0
-- word, COUNT(*) AS n_views

WITH total AS(
SELECT
    *,
    UNNEST(STRING_TO_ARRAY(BTRIM(post_keywords, '[]#'), ',')) AS word
FROM facebook_posts p
LEFT JOIN facebook_post_views v
ON p.post_id = v.post_id
```

```
)
SELECT
    word,
    COUNT(viewer_id) AS cnt
FROM total
GROUP BY 1
ORDER BY cnt DESC
```

```
-- 法2:
-- 先创建cte在views表中算每个post的view数: post_id, COUNT(*) AS n_views
-- 然后将post表与cte表left join: word, COALESCE(n_views, 0)
-- 最后要算每个word总view数: word, SUM(n_views)
WITH base AS(
SELECT
    post_id,
    COUNT(*) AS n_views
FROM facebook_post_views
GROUP BY post_id
),
base2 AS(
SELECT
    UNNEST(STRING_TO_ARRAY(BTRIM(post_keywords, '[]#'), ','))  AS keyword,
    COALESCE(n_views, 0) AS n_views
FROM facebook_posts posts
LEFT JOIN base post_views
ON posts.post_id = post_views.post_id
)
SELECT
    keyword,
    SUM(n_views) AS total_views
FROM base2
GROUP BY keyword
ORDER BY total_views DESC
```

Find the number of processed and not-processed complaints of each type

Meta/Facebook Hard ID 9790

Find the number of processed and non-processed complaints of each type. Replace NULL values with 0s. Output the complaint type along with the number of processed and not-processed complaints.

Table: facebook_complaints

In [ ]:
```
计算A在B = True or False下的数量:
法1: A, SUM(CASE WHEN B = 'TRUE' THEN 1 ELSE 0 END) AS n_B, SUM(CASE WHEN B =
法2:
i. total cte: A, B, count(*) AS cnt
ii. A, MAX(CASE WHEN B THEN cnt ELSE 0 END) AS n_B, MAX(CASE WHEN NOT B THEN c
法3: A, COUNT(*) FILTER (WHERE B = 'TRUE') AS n_B, COUNT(*) FILTER (WHERE B =
```

```
--# processed and non-processed complaints ~ each type
--replaace null -> 0s
--output: type, n_complaints_processed, n_complaints_not_processed

--法1: A, SUM(CASE WHEN B = 'TRUE' THEN 1 ELSE 0 END) AS n_B, SUM(CASE WHEN B
= 'FALSE' THEN 1 ELSE 0 END) AS n_not_B + GROUP BY A
SELECT
    type,
    SUM(CASE WHEN processed = 'TRUE' THEN 1 ELSE 0 END) AS
n_complaints_processed,
    SUM(CASE WHEN processed = 'FALSE' THEN 1 ELSE 0 END) AS
n_complaints_not_processed
FROM facebook_complaints
GROUP BY type
```

```
--法2:
i. total cte: A, B, count(*) AS cnt
ii. A, MAX(CASE WHEN B THEN cnt ELSE 0 END) AS n_B, MAX(CASE WHEN NOT B THEN
cnt ELSE 0 END) AS n_not_B + GROUP BY A

WITH total AS(
SELECT
    type,
    processed,
    COUNT(*) AS n_entries
FROM facebook_complaints
GROUP BY type, processed
)
SELECT
    type,
    MAX(CASE WHEN processed THEN n_entries ELSE 0 END) AS
n_complaints_processed,
    MAX(CASE WHEN NOT processed THEN n_entries ELSE 0 END) AS
n_complaints_not_processed
FROM total
GROUP BY type
```

```
--法3: A, COUNT(*) FILTER (WHERE B = 'TRUE') AS n_B, COUNT(*) FILTER (WHERE B
= 'FALSE') AS n_not_B
SELECT
    type,
    COUNT(*) FILTER (WHERE processed = 'True') AS n_complaints_processed,
    COUNT(*) FILTER (WHERE processed = 'False') AS n_complaints_not_processed
FROM facebook_complaints
```

```
GROUP BY type
```

Time Between Two Events

Interview Question Date: July 2018

Meta/Facebook Hard ID 9784

Meta/Facebook's web logs capture every action from users starting from page loading to page scrolling. Find the user with the least amount of time between a page load and their first scroll down. Your output should include the user id, page load time, first scroll down time, and time between the two events in seconds.

Table: facebook_web_log

```
①两个ts相减出秒：
CAST(ts1 - ts2 AS TIME) AS time_diff
(ts1 - ts2::TIME) AS time_diff

找到同一个user，初次load和初次scroll down，最少的时间差：

法1：
i. cte:
a: A, ROW_NUMBER() OVER (PARTITION BY A ORDER BY B) AS row_num WHERE C =
'c1', D
b: A, ROW_NUMBER() OVER (PARTITION BY A ORDER BY B) AS row_num WHERE C =
'c2', D
ii. A, D1 - D2 FROM a join b ON a.row_num = b.row_num AND a.A = b.A WHERE D2
> D1


法2：将取两个不同性质的表self join一起，rnk = 1取duration最小值：
A, t1.A AS load_time, t2.A AS scroll_time, t2.A - t1.A AS duration
t1 join t2 ON t1.B = t2.B WHERE t1.C = 'c1' AND t2.C = 'c2' AND t2.A > t1.A

法3：LEAD(A) OVER (ORDER BY A); LEAD(B) OVER (ORDER BY C)
```

```
--user least amount of time BT page load and 1st scroll down
--output: user_id,  load_time,  scroll_time,    duration


法1：
i. cte:
a: A, ROW_NUMBER() OVER (PARTITION BY A ORDER BY B) AS row_num WHERE C =
'c1', D
b: A, ROW_NUMBER() OVER (PARTITION BY A ORDER BY B) AS row_num WHERE C =
'c2', D
ii. A, D1 - D2 FROM a join b ON a.row_num = b.row_num AND a.A = b.A WHERE D2
> D1


WITH pl AS(
    SELECT
        user_id,
```

```
        ROW_NUMBER () OVER (PARTITION BY user_id ORDER BY timestamp) AS
row_num,
        timestamp AS page_load_ts
    FROM facebook_web_log
    WHERE action = 'page_load'
),
sd AS(
    SELECT
        user_id,
        ROW_NUMBER () OVER (PARTITION BY user_id ORDER BY timestamp) AS
row_num,
        timestamp AS scroll_down_ts
    FROM facebook_web_log
    WHERE action = 'scroll_down'
)
SELECT
    pl.user_id,
    pl.page_load_ts,
    sd.scroll_down_ts,
    (sd.scroll_down_ts - pl.page_load_ts)::TIME AS time_diff
--    CAST(sd.scroll_down_ts - pl.page_load_ts AS time) AS time_diff
FROM pl JOIN sd
ON pl.row_num = sd.row_num AND pl.user_id = sd.user_id
--找到同一个user, 初次load和初次scroll down, 二次load和二次scroll down
WHERE scroll_down_ts >= page_load_ts
ORDER BY time_diff
LIMIT 1
```

法2: 将取两个不同性质的表self join一起, rnk = 1取duration最小值:

```
A, t1.A AS load_time, t2.A AS scroll_time, t2.A - t1.A AS duration
t1 join t2 ON t1.B = t2.B WHERE t1.C = 'c1' AND t2.C = 'c2' AND t2.A > t1.A

-- RANK() OVER (ORDER BY duration) AS rnk
-- rnk = 1

WITH cte AS(
SELECT
    t1.user_id,
    t1.timestamp AS load_time,
    t2.timestamp AS scroll_time,
    (t2.timestamp - t1.timestamp)::TIME AS duration
FROM facebook_web_log t1
JOIN facebook_web_log t2 ON t1.user_id = t2.user_id
WHERE t1.action = 'page_load'
AND t2.action = 'scroll_down'
AND t2.timestamp > t1.timestamp
),
rank_cte AS(
SELECT
    *,
    RANK() OVER (ORDER BY duration) AS rnk
FROM cte
)
SELECT
    user_id,
    load_time,
    scroll_time,
```

```
    duration
FROM rank_cte
WHERE rnk = 1
```

```
法3: LEAD(A) OVER (ORDER BY A); LEAD(B) OVER (ORDER BY C)
WITH cte AS(
SELECT
    *,
    LEAD(timestamp) OVER (ORDER BY timestamp) AS ts2,  --将后面一个ts拿过来
    LEAD(action) OVER (ORDER BY timestamp) AS action2  --将后面一个action拿过
来
FROM facebook_web_log
ORDER BY user_id, timestamp
)
SELECT
    user_id,
    timestamp AS page_load_time,
    ts2 AS first_scroll_time,
    CAST(ts2 - timestamp AS time) AS time_difference   --page load and 1st
scroll down
FROM cte
WHERE action = 'page_load' AND action2 = 'scroll_down'   --第一个动作load, 第
二个动作scroll_down
ORDER BY time_difference
LIMIT 1
```

Common Interests Amongst Users

Meta/Facebook Hard ID 9776

Count the subpopulations across datasets. Assume that a subpopulation is a group of users sharing a common interest (ex: Basketball, Food). Output the percentage of overlapping interests for two posters along with those poster's IDs. Calculate the percentage from the number of poster's interests. The poster column in the dataset refers to the user that posted the comment.

Table: facebook_posts

```
①求A1 A2相同部分重叠百分比overlap pct:
A是poster, B是keyword
法1:
i. 算每个A里共有几个B: total
A, B, COUNT(*) OVER (PARTITION BY A) AS total
ii. overlap = 每对A有几个相同的B / 其中一个最长的A总B : all join total ON 相同的
关键词, poster号不同
a.A, b.A, COUNT(a.B) / MAX(total)::DECIMAL; FROM all JOIN total; ON a.B = b.B
AND a.A <> b.A + GROUP BY a.A, b.A

法2:
i.split_word
ii.total: A, COUNT(*) + GROUP BY A【COUNT(*)出total】
```

```
iii. intersection: a.A, b.A, COUNT(*), A a JOIN A b ON a.B = b.B WHERE a.A <>
b.A  + GROUP BY 1,2【self join出共有intersection】
iv. overlap: a.A, b.A, intersection::DECIMAL / total ; FROM total a LEFT JOIN
intersection b ; ON a.A = b.A【left join出overlap】

②转换为小数: CAST(A AS DECIMAL) = A::DECIMAL
```

```
法1:
i. 算每个A里共有几个B: total
A, B, COUNT(*) OVER (PARTITION BY A) AS total
ii. overlap = 每对A有几个相同的B / 其中一个最长的A总B ： all join total ON 相同的
关键词, poster号不同
a.A, b.A, COUNT(a.B) / MAX(total)::DECIMAL; FROM all JOIN total; ON a.B = b.B
AND a.A <> b.A + GROUP BY a.A, b.A

WITH split_word AS(
SELECT
    poster,
    UNNEST(STRING_TO_ARRAY(BTRIM(post_keywords, '[]#'), ',')) AS keyword
FROM facebook_posts
),
poster_total AS(
SELECT
    poster,
    keyword,
    COUNT(*) OVER (PARTITION BY poster) AS total
--算每个poster里有几个词
FROM split_word
)
SELECT
    a.poster,
    b.poster,
    -- COUNT(a.keyword),                                                    -
-每对poster 有几个相同的词
    -- MAX(b.total),                                                       -
-每对poster 相同的词数 / 其中一个最长的poster总次数 = overlap
    COUNT(a.keyword) / MAX(b.total)::DECIMAL AS overlap
FROM split_word a JOIN poster_total b
ON a.keyword = b.keyword
AND a.poster <> b.poster
GROUP BY a.poster, b.poster
```

```
法2:
A是poster, B是keyword
i.split_word
ii.total: A, COUNT(*) + GROUP BY A【COUNT(*)出total】
iii. intersection: a.A, b.A, COUNT(*), A a JOIN A b ON a.B = b.B WHERE a.A <>
b.A  + GROUP BY 1,2【self join出共有intersection】
iv. overlap: a.A, b.A, intersection::DECIMAL / total FROM total a LEFT JOIN
intersection b ON a.A = b.A【left join出overlap】


WITH split_word AS
(SELECT
    poster,
    UNNEST(STRING_TO_ARRAY(BTRIM(post_keywords, '[]'), ',')) AS keyword
```

```
FROM facebook_posts
ORDER BY 1,2),
poster_total AS(
SELECT
    poster,
    COUNT(*) AS total                    --每个poster共有几个词
FROM split_word
GROUP BY poster
),
poster_intersection AS(
SELECT
    a.poster,
    b.poster AS poster2,
    COUNT(*) AS intersection             --每对poster有几个相同的词：看每对
poster出现了几个记录：1  basketball  2   basketball；1    food    2   food；1
  nba 2   nba：1 ~ 2 共 3个词重叠
FROM split_word a
JOIN split_word b
ON a.keyword = b.keyword                 --self join keyword相同
WHERE a.poster <> b.poster               --确定每对poster都是自己和别人
GROUP BY a.poster, b.poster
)
SELECT
    a.poster AS poster1,
    b.poster2 AS poster2,
    intersection::DECIMAL / total AS overlap
--    CAST(intersection AS DECIMAL)/total AS overlap
FROM poster_total a LEFT JOIN poster_intersection b
ON a.poster = b.poster
WHERE a.poster is NOT NULL AND b.poster is NOT NULL
```

Most Popular Room Types

Airbnb Hard ID 9763

Find the room types that are searched by most people. Output the room type alongside the number of searches for it. If the filter for room types has more than one room type, consider each unique room type as a separate row. Sort the result based on the number of searches in descending order.

Table: airbnb_searches

```
①UNNEST(STRING_TO_ARRAY(LTRIM(A, ','), ',')：取出去掉左边逗号，并以逗号分隔的词
②将一列里的词解构出来，算每个词对应的总搜索量：
SELECT DISTINCT *, UNNEST(STRING_TO_ARRAY(LTRIM(A, ','), ',')) AS
cleaned_filter：
--注意LTRIM(A, ',')：让变量左边是逗号的情况去掉，避免null值产生
--注意*，否则会有记录缺失情况；DISTINCT 因为原纪录待解构变量中可能有重复值
cleaned_filter, SUM(n_searches) AS sum_searches
```

```
-- room types ~ searched by most ppl
-- output: cleaned_filter, sum_searches：the room type alongside the number
of searches
```

```
-- filter for room types has more than one room type, consider each unique
room type as a separate row.
-- order by # searches DESC

WITH total AS(
SELECT
    DISTINCT   --在filter_room_types中，在一个格子里也存在重复值，eg"Entire
home/apt,Entire home/apt,Private room"，这样3个单词也只能有distinct2个词出来
    *,
--    UNNEST(STRING_TO_ARRAY(filter_room_types, ',')) AS cleaned_filter
      --有null值
    UNNEST(STRING_TO_ARRAY(LTRIM(filter_room_types, ','), ',')) AS
cleaned_filter   --没有null值
FROM airbnb_searches
)
SELECT
    cleaned_filter,
    SUM(n_searches) AS sum_searches
FROM total
GROUP BY cleaned_filter
ORDER BY sum_searches DESC
```

Inspection Scores For Businesses

Interview Question Date: May 2018

City of San Francisco Hard ID 9741

Find the median inspection score of each business and output the result along with the business name. Order records based on the inspection score in descending order. Try to come up with your own precise median calculation. In Postgres there is percentile_disc function available, however it's only approximation.

Table: sf_restaurant_health_violations

```
得到每个A的共计数，以及B在A中排序：
A,
B,
ROW_NUMBER() OVER (PARTITION BY A ORDER BY B) AS rnk,
COUNT(*) OVER (PARTITION BY A) AS cnt

Median中位数：
法1：percentile_cont(0.5) within GROUP (ORDER BY A)

法2：
A, B, row_number, cnt
A, AVG(B) WHERE row_mum IN ((cnt + 1 / 2), (cnt + 2 / 2)) + GROUP BY A, B

法3：
A, AVG(B), WHERE row_desc IN (row_asc, row_asc + 1, row_asc - 1) + GROUP BY A
```

```
--法1：percentile_cont(0.5) within GROUP (ORDER BY A)
SELECT
    business_name,
```

```
    percentile_cont(0.5) within GROUP (ORDER BY inspection_score) AS
median_inspection_score
FROM sf_restaurant_health_violations
WHERE inspection_score IS NOT NULL
GROUP BY business_name
ORDER BY median_inspection_score DESC
```

```
--法2:
-- A, B, row_number, cnt
-- A, AVG(B) WHERE row_mum IN ((cnt + 1 / 2), (cnt + 2 / 2)) + GROUP BY A, B

WITH cte AS(
SELECT
    business_name,
    inspection_score,
    ROW_NUMBER() OVER (PARTITION BY business_name ORDER BY inspection_score)
AS row_num,
    COUNT(*) OVER (PARTITION BY business_name) AS cnt
FROM sf_restaurant_health_violations
WHERE inspection_score IS NOT NULL
)
SELECT
    business_name,
    AVG(inspection_score) AS median_inspection_score
FROM cte
WHERE row_num IN ((cnt + 1) / 2, (cnt + 2) / 2 )
GROUP BY business_name, cnt
ORDER BY median_inspection_score DESC
```

Worst Businesses

Interview Question Date: May 2018

City of San Francisco Hard ID 9739

For every year, find the worst business in the dataset. The worst business has the most violations during the year. You should output the year, business name, and number of violations.

Table: sf_restaurant_health_violations

```
取年份:
DATE_PART('year', inspection_date) AS year
EXTRACT(YEAR FROM inspection_date) AS year

WHERE 中不能有aggregate function, 所以如果RANK() OVER()中含有复合函数, 就要再开一
个cte
```

```
WITH total AS(
SELECT
    DATE_PART('year', inspection_date) AS year,
--    EXTRACT(YEAR FROM inspection_date) AS year,
    business_name,
```

```
    COUNT(violation_id) AS violation_count
FROM sf_restaurant_health_violations
WHERE risk_category IS NOT NULL
GROUP BY 1,2
),
rank_cte AS(
SELECT
    year,
    business_name,
    violation_count,
    RANK() OVER (PARTITION BY year ORDER BY violation_count DESC) AS rnk
FROM total
)
SELECT
    year,
    business_name,
    violation_count
FROM rank_cte
WHERE rnk = 1
```

Highest Number Of High-risk Violations

Interview Question Date: May 2018

City of San Francisco Hard ID 9736 9

Data Engineer Data Scientist BI Analyst Data Analyst ML Engineer Find details of the business with the highest number of high-risk violations. Output all columns from the dataset considering business_id which consist 'high risk' phrase in risk_category column.

Table: sf_restaurant_health_violations

```
在总表中，挑出满足一个选出的部分：

法1：  a = (SElECT a FROM cte WHERE rnk = 1)
cte: A, COUNT(*), RANK() OVER (ORDER BY B DESC) AS rnk
SELECT * FROM A WHERE a = (SElECT a FROM cte WHERE rnk = 1)

法2：原表join cte ON id WHERE rnk = 1
cte: A, COUNT(*), RANK() OVER (ORDER BY B DESC) AS rnk
```

```
法1：  a = (SElECT a FROM cte WHERE rnk = 1)
cte: A, COUNT(*), RANK() OVER (ORDER BY B DESC) AS rnk
SELECT * FROM A WHERE a = (SElECT a FROM cte WHERE rnk = 1)
WITH cte AS(
SELECT
    business_id,
--    risk_category,
    COUNT(business_id) AS risk_cnt,
    RANK() OVER (ORDER BY COUNT(business_id) DESC) AS rnk
FROM sf_restaurant_health_violations
WHERE risk_category = 'High Risk'
GROUP BY business_id, risk_category
)
```

```
SELECT
    *
FROM sf_restaurant_health_violations
WHERE business_id  = (SELECT business_id FROM cte WHERE rnk = 1)
```

```
法2：原表join cte ON id WHERE rnk = 1
cte: A, COUNT(*), RANK() OVER (ORDER BY B DESC) AS rnk
SELECT A.* FROM A JOIN cte B ON A.a = B.a WHERE rnk = 1
WITH cte AS(
SELECT
    business_id,
--    risk_category,
    COUNT(business_id) AS risk_cnt,
    RANK() OVER (ORDER BY COUNT(business_id) DESC) AS rnk
FROM sf_restaurant_health_violations
WHERE risk_category = 'High Risk'
GROUP BY business_id, risk_category
)
SELECT
    A.*
FROM sf_restaurant_health_violations A
JOIN cte B ON A.business_id = B.business_id
WHERE rnk = 1
```

Number Of Inspections By Zip

Interview Question Date: May 2018

City of San Francisco Hard ID 9734

Find the number of inspections that happened in the municipality with postal code 94102 during January, May or November in each year. Output the count of each month separately.

Table: sf_restaurant_health_violations

```
①CASE WHEN + COUNT/SUM用法：
法1：
A, CASE WHEN B THEN COUNT(C) ELSE 0 END AS C_cnt + GROUP BY A, B
A, MAX(C_cnt) AS C_cnt + GROUP BY A

法2：A, SUM(CASE WHEN A THEN 1 ELSE 0 END) + GROUP BY A

②EXTRACT (YEAR FROM A :: DATE) AS year
EXTRACT可以直接在当前Query中使用，也可以放在CASE WHEN中

③MAX中无法写aggregate复合函数，可以再开一个cte使用复合函数命名的变量即可
```

```
法1：
A, CASE WHEN B THEN COUNT(C) ELSE 0 END AS C_cnt + GROUP BY A, B
A, MAX(C_cnt) AS C_cnt + GROUP BY A
WITH total AS(
SELECT
    *,
    EXTRACT(year FROM inspection_date::DATE) AS year,
```

```
    EXTRACT(month FROM inspection_date::DATE) AS month
FROM sf_restaurant_health_violations
WHERE business_postal_code = '94102'
),
cte2 AS(
SELECT
    year,
    CASE WHEN month = 1 THEN COUNT(inspection_id) ELSE 0 END AS
january_counts,
    CASE WHEN month = 5 THEN COUNT(inspection_id) ELSE 0 END AS may_counts,
    CASE WHEN month = 11 THEN COUNT(inspection_id) ELSE 0 END AS
november_counts
FROM total
GROUP BY year, month
)
SELECT
    year,
    MAX(january_counts) AS january_counts,
    MAX(may_counts) AS may_counts,
    MAX(november_counts) AS november_counts
FROM cte2
GROUP BY year
ORDER BY year
```

```
法2: A, SUM(CASE WHEN A THEN 1 ELSE 0 END) + GROUP BY A
WITH total AS(
SELECT
    *,
    EXTRACT(year FROM inspection_date::DATE) AS year,
    EXTRACT(month FROM inspection_date::DATE) AS month
FROM sf_restaurant_health_violations
WHERE business_postal_code = '94102'
)
SELECT
    year,
    SUM(CASE WHEN month = 1 THEN 1 ELSE 0 END) AS january_counts,
    SUM(CASE WHEN month = 5 THEN 1 ELSE 0 END) AS may_counts,
    SUM(CASE WHEN month = 11 THEN 1 ELSE 0 END) AS november_counts
FROM total
GROUP BY year
```

Dates Of Inspection

Interview Question Date: April 2018

City of Los Angeles Hard ID 9714

Find the latest inspection date for the most sanitary restaurant(s). Assume the most sanitary restaurant is the one with the highest number of points received in any inspection (not just the last one). Only businesses with 'restaurant' in the name should be considered in your analysis. Output the corresponding facility name, inspection score, latest inspection date, previous inspection date, and the difference between the latest and previous inspection dates. And order the records based on the latest inspection date in ascending order.

Table: los_angeles_restaurant_health_inspections

```
法1：rnk
A, B, C, LAG(C,1) OVER (PARTITION BY A ORDER BY C) AS prev_C, rnk
WHERE C = SELECT MAX(C) FROM table
SELECT * FROM total WHERE rnk = 1

法2：
匹配一个全集total，和一个只有两个变量构成的filter cte：
i. total：A, B, C
ii. cte: A, MAX(B) AS max_B + GROUP BY A
iii. total a JOIN cte b ON a.A = b.A WHERE B = max_B
```

```
-- latest inspection date ~ most sanitary resturants
-- assume most sanitary resturant is one with highest # of points ~ received
in inspection ~ not just last one
-- only biz with 'resturant' in name
-- output: facility_name, score: inspection score, activity_date: latest
inspection date, prev_activity_date,
number_of_days_between_high_scoring_inspections：difference between the
latest and previous inspection dates
-- order the records ~ latest inspection date asc

法1：rnk
A, B, C, LAG(C,1) OVER (PARTITION BY A ORDER BY C) AS prev_C, rnk
WHERE C = SELECT MAX(C) FROM table
SELECT * FROM total WHERE rnk = 1

WITH total AS(
SELECT
    facility_name,
    score,
    activity_date,
    LAG(activity_date, 1) OVER (PARTITION BY facility_name ORDER BY
activity_date) AS prev_activity_date,
    RANK() OVER (PARTITION BY facility_name ORDER BY activity_date DESC) AS
rnk
FROM los_angeles_restaurant_health_inspections
WHERE facility_name ILIKE '%RESTAURANT%'
AND score = (SELECT MAX(score) FROM
los_angeles_restaurant_health_inspections)
ORDER BY facility_name
)
SELECT
    facility_name,
    score,
    activity_date,
    prev_activity_date,
    activity_date - prev_activity_date AS
number_of_days_between_high_scoring_inspections
FROM total
WHERE rnk = 1
```

```
法2：
匹配一个全集total，和一个只有两个变量构成的filter cte：
i. total：A, B, C
```

```
ii. cte: A, MAX(B) AS max_B + GROUP BY A
iii. total a JOIN cte b ON a.A = b.A WHERE B = max_B

WITH total AS(
SELECT
    facility_name,
    score,
    activity_date,
    LAG(activity_date, 1) OVER (PARTITION BY facility_name ORDER BY
activity_date) AS prev_activity_date,
    activity_date - LAG(activity_date, 1) OVER (PARTITION BY facility_name
ORDER BY activity_date) AS number_of_days_between_high_scoring_inspections
FROM los_angeles_restaurant_health_inspections
),
max_date_cte AS(
SELECT
    facility_name,
    MAX(activity_date) AS max_date
FROM los_angeles_restaurant_health_inspections
GROUP BY facility_name
)
SELECT
    a.facility_name,
    score,
    activity_date,
    prev_activity_date,
    number_of_days_between_high_scoring_inspections
FROM total a JOIN max_date_cte b
ON a.facility_name = b.facility_name
WHERE a.facility_name ILIKE '%RESTAURANT%'
AND activity_date = max_date
AND score = (SELECT MAX(score) FROM
los_angeles_restaurant_health_inspections)
```

Find the scores of 4 quartiles of each company

Interview Question Date: April 2018

City of Los Angeles Hard ID 9713

Find the scores of 4 quartiles of each company

Output the company name along with the corresponding score of each quartile. Order records based on the average score of all quartiles in ascending order.

Table: los_angeles_restaurant_health_inspections

```
①percentile:
percentile_cont(0.25) within GROUP (ORDER BY score) AS q1
percentile_cont(0.5) within GROUP (ORDER BY score) AS q2
percentile_cont(0.75) within GROUP (ORDER BY score) AS q3
percentile_cont(1) within GROUP (ORDER BY score) AS q4
```

```
②平均数:
i. AVG(A)
ii. 0.25 * (q1 + q2 + q3 + q4)

③order by 后可接复合函数aggregate function
```

```
--scores ~ 4 quartiles ~ each company
--output:owner_name q1  q2  q3  q4
--order by avg score of all quartiles asc

--法1:
WITH total AS(
SELECT
    owner_name,
    percentile_cont(0.25) within GROUP (ORDER BY score) AS q1,
    percentile_cont(0.5) within GROUP (ORDER BY score) AS q2,
    percentile_cont(0.75) within GROUP (ORDER BY score) AS q3,
    percentile_cont(1) within GROUP (ORDER BY score) AS q4
FROM los_angeles_restaurant_health_inspections
GROUP BY  owner_name
)
SELECT
    *
FROM total
ORDER BY 0.25 * (q1 + q2 + q3 + q4) ASC
```

```
--法2:
SELECT
    owner_name,
    percentile_cont(0.25) within GROUP (ORDER BY score) AS q1,
    percentile_cont(0.5) within GROUP (ORDER BY score) AS q2,
    percentile_cont(0.75) within GROUP (ORDER BY score) AS q3,
    percentile_cont(1) within GROUP (ORDER BY score) AS q4
FROM los_angeles_restaurant_health_inspections
GROUP BY owner_name
ORDER BY AVG(score) ASC
```

Facilities With Lots Of Inspections

Interview Question Date: April 2018

City of Los Angeles Hard ID 9711

Find the facility that got the highest number of inspections in 2017 compared to other years. Compare the number of inspections per year and output only facilities that had the number of inspections greater in 2017 than in any other year. Each row in the dataset represents an inspection. Base your solution on the facility name and activity date fields.

Table: los_angeles_restaurant_health_inspections

```
算2017比其他年份检查数都高方法:

法1: (年份2017年排第一 rnk=1, 且只有这一个年份排第一 n_rows = 1)
```

```
i. 每个设施，按年检查数排序：A, B, COUNT(*) AS cnt, RANK() OVER (PARTITION BY A
ORDER BY COUNT(*) DESC)
ii. 算年检查数第一时，有几个年份： A, B, COUNT(*) OVER (PARTITION BY A) AS
n_rows + WHERE rnk = 1
iii. 算年份2017年，且只有一个年份排第一，即2017比其他年份都高：A + WHERE n_rows =
1 AND year = 2017


法2：算出cnt_2017 > cnt_not_2017
total, cnt_2017, cnt_not_2017
3个表left join: coalesce(cnt, 0)
+ where cnt_2017 > cnt_not_2017


取年份：
to_char(activity_date, 'YYYY') AS year
EXTRACT(year FROM activity_date) AS year
DATE_PART('year', activity_date) AS year

算有共有几个记录：A, B, COUNT(*) OVER (PARTITION BY A) AS n_rows + GROUP BY A,
B

不能A, EXTRACT(year FROM B) + GROUP BY A, B后直接COUNT(*),
需要A, EXTRACT(year FROM B) + GROUP BY A, EXTRACT(year FROM B)
因为GROUP BY A, B后，即使year相同，date不同，也算作不同记录，即cnt不准
故可以：
i.再新建个cte算cnt
ii. A, EXTRACT(year FROM B) + GROUP BY A, EXTRACT(year FROM B)才可以
```

```
-- 法1：（年份2017年排第一，且只有这一个年份排第一）
-- i. 每个设施，按年检查数排序：A, B, COUNT(*) AS cnt, RANK() OVER (PARTITION
BY A ORDER BY COUNT(*) DESC)
-- ii. 算年检查数第一时，有几个年份： A, B, COUNT(*) OVER (PARTITION BY A) AS
n_rows + WHERE rnk = 1
-- iii. 算年份2017年，且只有一个年份排第一，即2017比其他年份都高：A + WHERE
n_rows = 1 AND year = 2017

WITH rnk_cte AS(
SELECT
    facility_name,
--    DATE_PART('year', activity_date) AS year,
    to_char(activity_date, 'YYYY') AS year,
    COUNT(*) AS cnt,
    RANK() OVER (PARTITION BY facility_name ORDER BY COUNT(*)DESC) AS rnk
FROM los_angeles_restaurant_health_inspections
GROUP BY facility_name, to_char(activity_date, 'YYYY')
),
cte2 AS(
SELECT
    facility_name,
    year,
    COUNT(*) OVER (PARTITION BY facility_name) AS n_rows   --n_row是有几个年份
排在rnk = 1
FROM rnk_cte
WHERE rnk = 1
GROUP BY facility_name, year
)
```

```
SELECT
    facility_name
FROM cte2
WHERE n_rows = 1  --因为题目要求2017要greater than other years, 所以如果n_row = 2代表有2个年份都排名第一，需要排除
AND year = '2017'
```

```
-- 法2: 算出cnt_2017 > cnt_not_2017
-- total, cnt_2017, cnt_not_2017
-- 3个表left join: coalesce
-- + where cnt_2017 > cnt_not_2017

WITH total AS(
SELECT
    facility_name,
    EXTRACT(year FROM activity_date) AS year,
    record_id
FROM los_angeles_restaurant_health_inspections
),
total_cnt AS(
SELECT
    facility_name,
    year,
    COUNT(*) AS cnt
FROM total
GROUP BY facility_name, year
),
year_2017 AS(
SELECT
    facility_name,
    cnt AS cnt_2017
FROM total_cnt
WHERE year = 2017
),
year_not_2017 AS(
SELECT
    facility_name,
    MAX(cnt) AS cnt_not_2017
FROM total_cnt
WHERE year != 2017
GROUP BY facility_name
),
cte AS(
SELECT
    DISTINCT
    a.facility_name,
    COALESCE(cnt_2017, 0) AS cnt_2017_2,
    COALESCE(cnt_not_2017, 0) AS cnt_not_2017_2
FROM total_cnt a
LEFT JOIN year_2017 b
ON a.facility_name = b.facility_name
LEFT JOIN year_not_2017 c
ON b.facility_name = c.facility_name
)
SELECT
    DISTINCT facility_name
FROM cte
```

```
WHERE cnt_2017_2 > cnt_not_2017_2
ORDER BY facility_name
```

```
检查:
当除17年外也有记录时:
SELECT * FROM los_angeles_restaurant_health_inspections
WHERE facility_name IN ('CARNICERIA LA OAXAQUENA','MORI SUSHI', 'QWENCH 9TH
AND FLOWER')
ORDER BY facility_name, activity_date
15      16           17
         1           2
1        2           3
         1           2

n_row = 2的记录:
SELECT * FROM los_angeles_restaurant_health_inspections
WHERE facility_name IN ('DRINK COFFEE + TEA', 'GUATEMALTECA BAKERY', 'QWENCH
JUICE BAR', 'ROBEKS JUICE')
ORDER BY facility_name, activity_date
```

Find the variance and the standard deviation of scores that have grade A

City of Los Angeles Hard ID 9708

Find the variance of scores that have grade A using the formula AVG((X_i - mean_x) ^ 2). Output the result along with the corresponding standard deviation.

Table: los_angeles_restaurant_health_inspections

```
平方: square: power(X,2) or X * X
平均值: mean: avg(Y)
开根: sqrt: sqrt(Z)

求方差，标准差:
i. cte: AVG(A) AS avg_A
ii. X - X^: A - (SELECT avg_A FROM cte)
iii. (X - X^) ^2: POWER(A - (SELECT avg_A FROM cte) , 2)
iv. variance = AVG((X - X^) ^2): AVG(POWER(A - (SELECT avg_A FROM cte) , 2))
v. std = SQRT(AVG(POWER(A - (SELECT avg_A FROM cte) , 2)),2)
```

```
--variance of scores ~ grade A: AVG((x_i - mean_x)^2)
--output: variance  std

WITH mean_x_cte AS(
SELECT
    AVG(score) AS mean_x
FROM los_angeles_restaurant_health_inspections
WHERE grade = 'A'
),
mean_minus_cte AS(
SELECT
    score - (SELECT mean_x FROM mean_x_cte) AS mean_minus
```

```
FROM los_angeles_restaurant_health_inspections
WHERE grade = 'A'
)
SELECT
    AVG(POWER(mean_minus,2)) AS variance,
--    AVG(mean_minus * mean_minus),
    SQRT(AVG(POWER(mean_minus,2))) AS std
FROM mean_minus_cte
```

3rd Most Reported Health Issues

Interview Question Date: April 2018

City of Los Angeles Hard ID 9701

Each record in the table is a reported health issue and its classification is categorized by the facility type, size, risk score which is found in the pe_description column.

If we limit the table to only include businesses with Cafe, Tea, or Juice in the name, find the 3rd most common category (pe_description). Output the name of the facilities that contain 3rd most common category.

Table: los_angeles_restaurant_health_inspections

```
--WHERE A AND (B OR C OR D): WHERE 中 如有并列关系，含有OR的要加括弧：
--需要将WHERE中3个ILIKE加括弧，否则变成平行关系，出来的结果一部分不满足第一个条件

--A ILKIE '%a%' OR A ILIKE '%b%' OR A ILIKE '%c%'

取部分和整体的部分：
法1：SELECT A FROM total WHERE B IN (SELECT B FROM cte)
法2：JOIN: A, total a JOIN cte b ON a.B = b.B
```

```
-- limit table on only has biz with Cafe, Tea, Juice in the name
-- 3rd common category(pe_description)
-- output: facility_name

取部分和整体的部分：
法1：SELECT A FROM total WHERE B IN (SELECT B FROM cte)

WITH common_category_cte AS(
SELECT
    pe_description,
    COUNT(record_id) AS cnt,
    RANK() OVER (ORDER BY COUNT(record_id) DESC) AS rnk
FROM los_angeles_restaurant_health_inspections
WHERE facility_name ILIKE '%CAFE%'
or facility_name ILIKE '%TEA%'
or facility_name ILIKE '%JUICE%'
GROUP BY pe_description
)
SELECT
    DISTINCT facility_name
FROM los_angeles_restaurant_health_inspections
```

```
WHERE pe_description IN (SELECT pe_description FROM common_category_cte WHERE
rnk = 3)
AND
(facility_name ILIKE '%CAFE%'
or facility_name ILIKE '%TEA%'
or facility_name ILIKE '%JUICE%')
ORDER BY facility_name
```

```
PE: RESTAURANT (0-30) SEATS MODERATE RISK, 不属于target pe_description
SELECT DISTINCT pe_description
FROM los_angeles_restaurant_health_inspections
WHERE facility_name IN ('DRINK COFFEE + TEA', 'QWENCH JUICE BAR', 'ROBEKS
JUICE', 'TROPICANA JUICE BAR & DELI' )

-- pe_description
-- RESTAURANT (0-30) SEATS LOW RISK
-- RESTAURANT (31-60) SEATS MODERATE RISK
```

Growth of Airbnb

Interview Question Date: February 2018

Airbnb Hard ID 9637

Estimate the growth of Airbnb each year using the number of hosts registered as the growth metric. The rate of growth is calculated by taking ((number of hosts registered in the current year - number of hosts registered in the previous year) / the number of hosts registered in the previous year) * 100. Output the year, number of hosts in the current year, number of hosts in the previous year, and the rate of growth. Round the rate of growth to the nearest percent and order the result in the ascending order based on the year. Assume that the dataset consists only of unique hosts, meaning there are no duplicate hosts listed.

Table: airbnb_search_details

```
ROUND(((A - B) * 100 / B) )
ROUND(((A - B) / (CAST(B AS numeric)) )  *100)

LAG(X, 1) OVER (ORDER BY Y)
```

```
-- growth of airbnb each yr ~ # hosts registered
-- rate of growth: (# hosts registered current yr - # hosts registered prev
yr) / # hosts reg prev yr * 100
-- output: year current_year_host   prev_year_host  estimated_growth:the rate
of growth
-- round rate to pct
-- order yr asc
-- no duplicate hosts


WITH host_cnt AS(
    SELECT
        to_char(host_since, 'YYYY') AS year,
        COUNT(*) AS current_year_host
```

```
    FROM airbnb_search_details
    GROUP BY to_char(host_since, 'YYYY')
),
    prev_cnt AS(
    SELECT
        year,
        current_year_host,
        LAG(current_year_host, 1) OVER (ORDER BY year) AS prev_year_host
    FROM host_cnt
    )
SELECT
    *,
    ROUND(((current_year_host - prev_year_host) * 100 / (prev_year_host)) )
AS estimated_growth
--    ROUND(((current_year_host - prev_year_host)       /
(CAST(prev_year_host AS numeric)) )  *100) estimated_growth
FROM prev_cnt
ORDER BY year
```

City With Most Amenities

Interview Question Date: February 2018

Airbnb Hard ID 9633

You're given a dataset of searches for properties on Airbnb. For simplicity, let's say that each search result (i.e., each row) represents a unique host. Find the city with the most amenities across all their host's properties. Output the name of the city.

Table: airbnb_search_details

```
①先算出每个此条的A_cnt，再算每个城市，A_cnt_sum

②取出以逗号分隔的词：STRING_TO_ARRAY(A, ',')
算array中有多少个词：ARRAY_LENGTH(A, 1)
{TV,"Wireless Internet","Air conditioning","Smoke detector","Carbon monoxide
detector",Essentials,"Lock on bedroom door",Hangers,Iron}

③选最大值：
法1：amenties_cnt; rank = 1
法2：SELECT A FROM B WHERE A = (SELECT MAX(A) FROM B)
```

```
法1：amenties_cnt; rank = 1
WITH total AS(
SELECT
    city,
    ARRAY_LENGTH(STRING_TO_ARRAY(amenities, ','), 1) AS amenities_cnt
FROM airbnb_search_details
),
rnk_cte AS(
SELECT
    city,
    SUM(amenities_cnt) AS amenities_cnt_sum,
    RANK() OVER (ORDER BY SUM(amenities_cnt) DESC) AS rnk
```

```
FROM total
GROUP BY city
)
SELECT
    city
FROM rnk_cte
WHERE rnk = 1
```

```
法2: SELECT A FROM B WHERE A = (SELECT MAX(A) FROM B)
WITH total AS(
SELECT
    city,
    ARRAY_LENGTH(STRING_TO_ARRAY(amenities, ','), 1) AS amenities_cnt
FROM airbnb_search_details
),
sum_cte AS(
SELECT
    city,
    SUM(amenities_cnt) AS total_amenities
FROM total
GROUP BY city
)
SELECT
    DISTINCT city
FROM sum_cte
WHERE total_amenities = (SELECT MAX(total_amenities) FROM sum_cte)
```

Host Popularity Rental Prices

Interview Question Date: February 2018

Airbnb Hard ID 9632

You’re given a table of rental property searches by users. The table consists of search results and outputs host information for searchers. Find the minimum, average, maximum rental prices for each host’s popularity rating. The host’s popularity rating is defined as below: 0 reviews: New 1 to 5 reviews: Rising 6 to 15 reviews: Trending Up 16 to 40 reviews: Popular more than 40 reviews: Hot

Tip: The id column in the table refers to the search ID. You'll need to create your own host_id by concating price, room_type, host_since, zipcode, and number_of_reviews.

Output host popularity rating and their minimum, average and maximum rental prices.

Table: airbnb_host_searches

```
①创建unique id: CONCAT(A, B); A || B; md5(A || B)
i. CONCAT(price, room_type, host_since, zipcode, number_of_reviews) AS
host_id,
409.43Private room2013-03-19100307
ii. A || B || C || D || E
270.81Shared room2016-03-279000635
iii. md5(A || B || C || D || E)
```

```
90922e43d7d3fb9c055abdb401549b36

②需要注意distinct，因为截取出的信息，会有host重复出现，如不去重，影响平均值准确性
```

```
table rental property searches by users
mini, avg, max rental prices for each host's pop rating
id: search id
create your own host_id by concating price, room_type, host_since, zipcode,
and number_of_reviews.
output: host_pop_rating, min_price, avg_price,  max_price

WITH host_info AS(
SELECT
    DISTINCT
    CONCAT(price, room_type, host_since, zipcode, number_of_reviews) AS
host_id,
    price,
    number_of_reviews
FROM airbnb_host_searches
),
host_type AS(
SELECT
    *,
    CASE
        WHEN number_of_reviews = 0 THEN 'New'
        WHEN number_of_reviews BETWEEN 1 AND 5 THEN 'Rising'
        WHEN number_of_reviews BETWEEN 6 AND 15 THEN 'Trending Up'
        WHEN number_of_reviews BETWEEN 16 AND 40 THEN 'Popular'
        WHEN number_of_reviews > 40 THEN 'Hot'
    END AS host_pop_rating
FROM host_info
)
SELECT
    host_pop_rating,
    MIN(price) AS min_price,
    AVG(price) AS avg_price,
    MAX(price) AS max_price
FROM host_type
GROUP BY host_pop_rating
```

Keywords From Yelp Reviews

Yelp Hard ID 9612

Find Yelp food reviews containing any of the keywords: 'food', 'pizza', 'sandwich', or 'burger'. List the business name, address, and the state which satisfies the requirement.

Tables: yelp_business, yelp_reviews

```
BTRIM(A , '"')
STRPOS(A, 'B') <> 0

多个相似：
A similar to '%(B|C|D|E)%'
```

```
STRPOS(A, 'B') <> 0 OR STRPOS(A, 'C')
A ILIKE '%B%' OR A ILIKE '%C%'

法1：不用JOIN方法：SELECT A FROM B WHERE C IN (SELECT C FROM B WHERE ILIKE)

法2：用JOIN方法
```

```
-- 法1：不用JOIN方法：SELECT A FROM B WHERE C IN (SELECT C FROM B WHERE ILIKE)
SELECT
    name,
    address,
    state
FROM yelp_business
WHERE name in (SELECT
    DISTINCT business_name
FROM yelp_reviews
WHERE review_text ILIKE '%food%'
    OR review_text ILIKE '%pizza%'
    OR review_text ILIKE '%sandwich%'
    OR review_text ILIKE '%burger%')
```

```
法2：用JOIN方法
WITH keyword AS(
SELECT
    business_name
FROM yelp_reviews
WHERE
    STRPOS(review_text, 'food') <> 0 OR
    STRPOS(review_text, 'sandwich') <> 0 OR
    STRPOS(review_text, 'pizza') <> 0 OR
    STRPOS(review_text, 'burger') <> 0
)
SELECT
    DISTINCT
    business_name,
    address,
    state
FROM keyword a JOIN yelp_business b
ON BTRIM(b.name, '"') = a.business_name
--ON b.name = a.business_name
```

Exclusive Amazon Products

Amazon Hard ID 9608

Find products which are exclusive to only Amazon and therefore not sold at Top Shop and Macy's. Your output should include the product name, brand name, price, and rating.

Two products are considered equal if they have the same product name and same maximum retail price (mrp column).

Tables: innerwear_macys_com, innerwear_topshop_com, innerwear_amazon_com

合并多个变量：
CONCAT(A, B)
A || '|' || B

多个变量形成的变量组在不同表中关系表达：
法1：CONCAT(A, B) AS A_B使其变成一个变量，SELECT A FROM B WHERE A NOT IN (SELECT A FROM C)

法2：
i. not_in_cte: A, B
ii. 将WHERE后(A, B)括起来filter：SELECT C,D,E WHERE (A,B) NOT IN (SELECT A, B FROM not_in_cte)

```
products ~ only Amazon and not sold at top shop and Macy's
output: product_name    brand_name  price   rating
2 equal products ~ same product name and same max retail price


法1：CONCAT(A, B) AS A_B使其变成一个变量，SELECT A FROM B WHERE A NOT IN (SELECT A FROM C)
WITH amazon AS(
    SELECT
        CONCAT(product_name, mrp) AS product_name_mrp,
        product_name,
        brand_name,
        price,
        rating
    FROM innerwear_amazon_com
),
    macys AS(
    SELECT
        CONCAT(product_name, mrp) AS product_name_mrp
    FROM innerwear_macys_com
    ),
    topshop AS (
    SELECT
        CONCAT(product_name, mrp) AS product_name_mrp
    FROM innerwear_topshop_com
    )
SELECT
    product_name,
    brand_name,
    price,
    rating
FROM amazon
WHERE product_name_mrp NOT IN (SELECT product_name_mrp FROM macys UNION ALL
SELECT product_name_mrp FROM topshop)
```

```
法2：
i. not_in_cte: A, B
ii. 将WHERE后(A, B)括起来filter：SELECT C,D,E WHERE (A,B) NOT IN (SELECT A, B FROM not_in_cte)
WITH not_in AS(
SELECT
    product_name,
    mrp
```

```
FROM innerwear_macys_com
UNION ALL
SELECT
    product_name,
    mrp
FROM innerwear_topshop_com
)
SELECT
    product_name,
    brand_name,
    price,
    rating
FROM innerwear_amazon_com
WHERE (product_name, mrp) NOT IN (SELECT product_name, mrp FROM not_in)
```

Differences In Movie Ratings

Google Netflix Hard ID 9606

Calculate the average lifetime rating and rating from the movie with second biggest id across all actors and all films they had acted in. Remove null ratings from the calculation. Role type is "Normal Acting". Output a list of actors, their average lifetime rating, rating from the film with the second biggest id (use id column), and the absolute difference between the two ratings.

Tables: nominee_filmography, nominee_informatio

```
算A的平均值，及A在所有作品中-第二大ID用户给的评分，两者绝对值：

法1：JOIN
i. life_time_rating: A, AVG(B) + GROUP BY A
ii. 2nd_rating: A, B, ROW_NUMBER() OVER (PARTITION BY A ORDER BY id DESC) AS
rnk：对每个Actor来说，给他们评价过的第二大用户id给的分数
iii. 2nd_rating JOIN life_time_rating ON name WHERE rnk = 2

法2：AVG(B) OVER (PARTITION BY A)可以和其他RANK一起使用
A, B, RANK() OVRE (PARTITION BY A ORDER BY id DESC) AS rnk, AVG(B) OVER
(PARTITION BY A) AS rating_life
rnk = 2
```

```
WITH life_time_rating AS(
SELECT
    name,
    AVG(rating) AS lifetime_rating
FROM nominee_filmography
WHERE role_type = 'Normal Acting' AND rating IS NOT NULL
GROUP BY name
),
second_last_rating_cte AS(
SELECT
    name,
    rating,
    ROW_NUMBER() OVER (PARTITION BY name ORDER by id DESC) AS rnk
FROM nominee_filmography
```

```
WHERE role_type = 'Normal Acting' AND rating IS NOT NULL                --
rating is not null和role type filter要保留
)
SELECT
    a.name,
    a.rating AS second_last_rating,
    b.lifetime_rating,
    ABS(b.lifetime_rating - a.rating) AS variance
FROM second_last_rating_cte a
JOIN life_time_rating b
ON a.name = b.name
WHERE rnk = 2
```

```
法2: AVG(B) OVER (PARTITION BY A)可以和其他RANK一起使用
A, B, RANK() OVRE (PARTITION BY A ORDER BY id DESC) AS rnk, AVG(B) OVER
(PARTITION BY A) AS rating_life
rnk = 2

WITH cte AS (
SELECT name
    ,rating
    ,rank() OVER (partition by name order by id desc ) AS rkn
    ,AVG(rating) OVER (partition by name) AS ratingLife
FROM nominee_filmography
WHERE rating is not null and role_type = 'Normal Acting'
)

SELECT name
    ,rating
    ,ratingLife
    ,abs(rating - ratingLife) AS variance
FROM cte
WHERE rkn = 2
```

Completed Trip within 168 Hours

Interview Question Date: October 2022

Uber Hard ID 2134

An event is logged in the events table with a timestamp each time a new rider attempts a signup (with an event name 'attempted_su') or successfully signs up (with an event name of 'su_success').

For each city and date, determine the percentage of successful signups in the first 7 days of 2022 that completed a trip within 168 hours of the signup date. HINT: driver id column corresponds to rider id column

Tables: signup_events, trip_details

```
两个Timestamp相减得出小时数:
HOUR = EXTRACT(EPOCH FROM ts1 - ts2) / 3600
```

```
the percentage of successful signups in the first 7 days of 2022 that
completed a trip within 168 hours of the signup date
COUNT(1号到7号成功注册且168h内从注册到完成司机数) / COUNT(1号到7号成功注册司机数)
```

```
events table with a timestamp each time a new rider attempts a
signup(attempted_su)
or succ sign up (su_success)

city, date, pct succ signups first 7 days of 2022 completed trip within 168
hours of signup date
rider_id = driver_id

output: city_id date    percentage
168 hours  = 7 days


WITH signups AS(
SELECT
    city_id,
    timestamp,
    rider_id
FROM signup_events
WHERE event_name ILIKE 'su_success'
AND timestamp::DATE BETWEEN '2022-01-01' AND '2022-01-07'
),
first_trips_in_168_hours AS(
SELECT
    DISTINCT
    driver_id
FROM signup_events
JOIN trip_details
ON rider_id = driver_id
WHERE status LIKE 'completed'
AND EXTRACT(EPOCH FROM actual_time_of_arrival - timestamp) / 3600 <= 168
)
SELECT
    city_id,
    timestamp::date AS date,
    COUNT(driver_id) * 100  / COUNT(rider_id) :: FLOAT AS percentage
FROM signups
LEFT JOIN first_trips_in_168_hours
ON rider_id = driver_id
GROUP BY 1,2
```

Consecutive Days

Interview Question Date: July 2021

Salesforce Netflix Hard ID 2054

Find all the users who were active for 3 consecutive days or more.

Table: sf_events

①取B后第一个值：LEAD(B, 1) OVER (PARTITION BY A ORDER BY B)
②取B后第二个值：LEAD(B, 2) OVER (PARTITION BY A ORDER BY B)
③DATEDIFF(A, B) = 1: A B 日期之间差值为1

④求连续3天或以上
法1：
i. cte1: DISTINCT A, B FROM table
ii. cte2: A, B, B - ROW_NUMBER() OVER (PARTITION BY A ORDER BY B) AS B_r FROM cte
iii. DISTINCT A FROM cte2 GROUP BY A, B_r HAVING COUNT(*) >= n
因为如果date是连续的，则B - ROW_NUMBER() OVER (ORDER BY B) 会是相同的，
算有几个user_id, date_h_r相同的记录(GROUP BY 1,2)，就是连续几天。

⑤满足3天连续：
法1：
i.cte: A, B, LEAD(B, 1) OVER (PARTITION BY A ORDER BY B) AS next_day, LEAD(B,2) OVER (PARTITION BY A ORDER BY B) AS next_2_day
ii. WHERE datediff(next_day, date) = 1 AND datediff(next_2_day, date) = 2
法2：A a JOIN A b ON a.date + INTERVAL '1 day' = b.date JOIN A c ON b.date + INTERVAL '1 day' = c.date

求连续3天或以上
法1：
i. cte1: DISTINCT A, B FROM table
ii. cte2: A, B, B - ROW_NUMBER() OVER (PARTITION BY A ORDER BY B) AS B_r FROM cte
iii. DISTINCT A FROM cte2 GROUP BY A, B_r HAVING COUNT(*) >= n

因为如果date是连续的，则B - ROW_NUMBER() OVER (ORDER BY B) 会是相同的，
算有几个user_id, date_h_r相同的记录(GROUP BY 1,2)，就是连续几天。

```
WITH distinct_days AS(
SELECT
DISTINCT
    user_id,
    date AS date_h
FROM sf_events
ORDER BY user_id ASC, date_h ASC
),
consecutive_days AS(
SELECT
    user_id,
    date_h,
    ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY date_h),
    date_h - ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY date_h)::int AS date_h_r
FROM distinct_days
)
SELECT
    DISTINCT
    user_id
    -- user_id,
    -- date_h_r
FROM consecutive_days
GROUP BY user_id, date_h_r
```

```
HAVING COUNT(*) >= 3
```

满足3天连续：

法1：
i.cte: A, B, LEAD(B, 1) OVER (PARTITION BY A ORDER BY B) AS next_day,
LEAD(B,2) OVER (PARTITION BY A ORDER BY B) AS next_2_day
ii. WHERE datediff(next_day, date) = 1 AND datediff(next_2_day, date) = 2

```
WITH CTE AS (
SELECT
    user_id,
    date,
    LEAD(DATE, 1) OVER(PARTITION BY USER_ID ORDER BY DATE) AS NEXT_DAY,
    LEAD(DATE, 2) OVER(PARTITION BY USER_ID ORDER BY DATE) AS NEXT_2_DAY
FROM sf_events)
SELECT
    DISTINCT user_id
FROM CTE
WHERE DATEDIFF(next_day, date)= 1
AND DATEDIFF(NEXT_2_DAY, date)= 2
```

法2：A a JOIN A b ON a.date + INTERVAL '1 day' = b.date JOIN A c ON b.date +
INTERVAL '1 day' = c.date

```
SELECT
a.user_id
FROM sf_events a
JOIN sf_events b
ON a.date + INTERVAL '1 day' = b.date
JOIN sf_events c
ON b.date + INTERVAL '1 day' = c.date
```

Sales Percentage Week's Beginning and End

Interview Question Date: May 2023

Facebook Hard ID 2165

The sales department has given you the sales figures for the first two months of 2023.

You've been tasked with determining the percentage of weekly sales on the first and last day of every week.

In your output, include the week number, percentage sales for the first day of the week, and percentage sales for the last day of the week. Both proportions should be rounded to the nearest whole number.

Table: early_sales

①把每周分为第几周：DATE_PART('WEEK', TO_TIMESTAMP(invoicedate, 'DD/MM/YYYY'))
②把每天分为每周第几天：DATE_PART('ISODOW', TO_TIMESTAMP(invoicedate,
'DD/MM/YYYY')) AS day_of_week

```
DATE(DATE_TRUNC('week', TO_TIMESTAMP(invoicedate, 'DD/MM/YYYY HH24:MI')))


③求每周，周一销售额，周日销售额
i. cte: week_no, day_of_week, sales
ii. week_no, (SUM(CASE WHEN day_of_week = 1 THEN sales ELSE 0 END) /
SUM(sales) * 100)::INTEGER AS start_week_pc,
(SUM(CASE WHEN day_of_week = 7 THEN sales ELSE 0 END) / SUM(sales) *
100)::INTEGER AS start_week_pc
+ GROUP BY 1
```

```
WITH sales AS(
SELECT
    DATE_PART('WEEK', TO_TIMESTAMP(invoicedate, 'DD/MM/YYYY')) AS week_no,

    DATE_PART('ISODOW', TO_TIMESTAMP(invoicedate, 'DD/MM/YYYY')) AS
day_of_week,
    quantity::FLOAT * unitprice AS sales
FROM early_sales
)
SELECT
    week_no,
    (SUM(CASE WHEN day_of_week = 1 THEN sales ELSE 0 END) / SUM(sales) *
100)::INTEGER AS start_week_pc,
    (SUM(CASE WHEN day_of_week = 7 THEN sales ELSE 0 END) / SUM(sales) *
100)::INTEGER AS end_week_pc
FROM sales
GROUP BY 1
ORDER BY 1
```

```
法2:
WITH first_week_day AS(
SELECT
    DATE(TO_TIMESTAMP(invoicedate, 'DD/MM/YYYY HH24:MI')) AS first_day,
    --DATE(DATE_TRUNC('week', TO_TIMESTAMP(invoicedate, 'DD/MM/YYYY
HH24:MI'))),
    SUM(quantity * unitprice) AS first_day_sales
FROM early_sales
WHERE DATE(TO_TIMESTAMP(invoicedate, 'DD/MM/YYYY HH24:MI')) =
DATE(DATE_TRUNC('week', TO_TIMESTAMP(invoicedate, 'DD/MM/YYYY HH24:MI')))
GROUP BY 1
),
last_week_day AS(
SELECT
    DATE(TO_TIMESTAMP(invoicedate, 'DD/MM/YYYY HH24:MI')) AS last_day,
    SUM(quantity * unitprice) AS last_day_sales
FROM early_sales
WHERE DATE(TO_TIMESTAMP(invoicedate, 'DD/MM/YYYY HH24:MI')) =
DATE(DATE_TRUNC('week', TO_TIMESTAMP(invoicedate, 'DD/MM/YYYY HH24:MI'))) +
interval '1 week' - interval '1 day'
GROUP BY 1),
full_week AS(
SELECT
    EXTRACT(WEEK FROM TO_TIMESTAMP(invoicedate, 'DD/MM/YYYY HH24:MI')) AS
week_no,
    SUM(quantity * unitprice) AS week_sales
```

```
FROM early_sales
GROUP BY 1
)
SELECT
    week_no,
    COALESCE(ROUND(100 * (first_day_sales / week_sales)), 0) AS
start_week_pc,
    COALESCE(ROUND(100 * (last_day_sales / week_sales)), 0) AS end_week_pc
FROM first_week_day f
FULL JOIN last_week_day l
                  --[FULL JOIN]
ON CAST(first_day AS date) = CAST(last_day AS DATE) - 6
                  --一周的周一和周日
JOIN full_week fw
ON fw.week_no = EXTRACT(WEEK FROM first_day) OR fw.week_no = EXTRACT(WEEK
FROM last_day)         --周一的周 = 周日的周
ORDER BY 1
```

The Most Popular Client_Id Among Users Using Video and Voice Calls

Interview Question Date: March 2021

Microsoft Apple Hard ID 2029

Select the most popular client_id based on a count of the number of users who have at least 50% of their events from the following list: 'video call received', 'video call sent', 'voice call received', 'voice call sent'.

Table: fact_events

```
满足B是或C或D: CASE WHEN B IN ('C', 'D') THEN 1 ELSE 0 END

两者等价:
A, SUM(CASE WHEN B IN ('C', 'D') THEN 1 ELSE 0 END) / COUNT(*)::DECIMAL AS
pct + GROUP BY A
A, AVG(CASE WHEN B IN ('C', 'D') THEN 1 ELSE 0 END) AS pct + GROUP BY A

找到最多用户数的客户，客户50%活动来自一个list:
法1:
在pct_cte中，第二个变量应该写AVG(CASE WHEN)的pct省略没写，放在HAVING中直接>=0.5可
以少些一个cte
在rank_cte中，少些一个COUNT(*)变量，放在RANK() OVER (ORDER BY COUNT(*))可以直接
按数量排序，只是要多加一个GROUP BY A

法2:
i. pct_cte: A, SUM(CASE WHEN B IN ('C', 'D') THEN 1 ELSE 0 END) /
COUNT(*)::DECIMAL AS pct + GROUP BY A
ii.client_cte: A, RANK() OVER (ORDER BY COUNT(*)DESC) AS rnk WHERE rnk = 1
```

```
most pop client_id based on a count of the number of users who have >= 50%
events
video call received, video call sent, voice call received, voice call sent
```

找到最多用户数的客户，客户50%活动来自一个list:
法1:
在pct_cte中，第二个变量应该写AVG(CASE WHEN)的pct省略没写，放在HAVING中直接>=0.5可以少些一个cte
在rank_cte中，少些一个COUNT(*)变量，放在RANK() OVER (ORDER BY COUNT(*))可以直接按数量排序，只是要多加一个GROUP BY A

```
WITH pct_cte AS(
SELECT
    user_id
FROM fact_events
GROUP BY user_id
HAVING AVG(CASE
                WHEN event_type in ('video call received', 'video call sent',
'voice call received', 'voice call sent') THEN 1
                ELSE 0
            END) >= 0.5
),
rank_cte AS(
SELECT
    client_id,
    RANK() OVER (ORDER BY COUNT(*) DESC) AS rnk
FROM fact_events
WHERE user_id IN (SELECT user_id FROM fact_events)
GROUP BY client_id
)
SELECT
    client_id
FROM rank_cte
WHERE rnk = 1
```

法2:
i. pct_cte: A, SUM(CASE WHEN B IN ('C', 'D') THEN 1 ELSE 0 END) / COUNT(*)::DECIMAL AS pct + GROUP BY A
ii.client_cte: A, RANK() OVER (ORDER BY COUNT(*)DESC) AS rnk WHERE rnk = 1

```
WITH pct_cte AS(
SELECT
user_id,
SUM(CASE WHEN event_type IN ('video call received', 'video call sent', 'voice
call received', 'voice call sent') THEN 1 ELSE 0 END) / COUNT(*)::DECIMAL AS
pct
FROM fact_events
GROUP BY 1
),
client_cte AS(
SELECT
    client_id,
--    COUNT(client_id) AS client_cnt,
    RANK() OVER (ORDER BY COUNT(client_id) DESC) AS rnk
FROM fact_events
WHERE user_id IN (SELECT user_id FROM pct_cte WHERE pct >= 0.5)
GROUP BY client_id
)
SELECT
    client_id
```

```
FROM client_cte
WHERE rnk = 1
```

Cookbook Recipes

Interview Question Date: February 2022

Amazon Ebay Hard ID 2089 33

Data Engineer Data Scientist BI Analyst Data Analyst You are given the table with titles of recipes from a cookbook and their page numbers. You are asked to represent how the recipes will be distributed in the book. Produce a table consisting of three columns: left_page_number, left_title and right_title. The k-th row (counting from 0), should contain the number and the title of the page with the number 2×k in the first and second columns respectively, and the title of the page with the number 2×k+1 in the third column. Each page contains at most 1 recipe. If the page does not contain a recipe, the appropriate cell should remain empty (NULL value). Page 0 (the internal side of the front cover) is guaranteed to be empty.

Table: cookbook_titles

In [ ]:
```
类似于FB 两个抓手，左边抓名字，右边也抓名字的题

要给记录命名一列数字排序，可以用ROW_NUMBER() OVER (ORDER BY A)

不能直接在k_cte中选，只能是在一行page_number = left_page(2k)，就会少掉left page是
需要将k_cte先与原表join，on left_page = page number，获得所有left_page = 2k时，应
再与原表join一次，on right_page = page_number，获得所有right_page = 2k + 1，应该
最后取left_page_number，left_title，right_title即可
```

```
WITH k_cte AS(
SELECT
    *,
    ROW_NUMBER() OVER (ORDER BY page_number) - 1 AS k,
    2 * (ROW_NUMBER() OVER (ORDER BY page_number) - 1) AS left_page,
    2 * (ROW_NUMBER() OVER (ORDER BY page_number) - 1) + 1 AS right_page
FROM cookbook_titles
)
SELECT
--    *,
    left_page,
    b.title AS left_title,
    c.title AS rigt_title
FROM k_cte a LEFT JOIN cookbook_titles b
ON a.left_page = b.page_number
LEFT JOIN cookbook_titles c
ON a.right_page = c.page_number
```

```
WITH k_cte AS(
SELECT
    ROW_NUMBER() OVER (ORDER BY page_number) - 1 AS k,                    --
k
```

```
    2 * (ROW_NUMBER() OVER (ORDER BY page_number) - 1) AS left_page,       -
-2k
    2 * (ROW_NUMBER() OVER (ORDER BY page_number) - 1) + 1 AS right_page   -
-2k + 1
FROM cookbook_titles
)
SELECT
    left_page,
    CASE WHEN a.left_page = b.page_number THEN b.title ELSE NULL END AS
left_title,
    CASE WHEN a.right_page = c.page_number THEN c.title ELSE NULL END AS
right_title
FROM k_cte a
LEFT JOIN cookbook_titles b
ON a.left_page = b.page_number
LEFT JOIN cookbook_titles c
ON a.right_page = c.page_number
```

Customer Tracking

Interview Question Date: October 2022

Amazon Shopify Hard ID 2136

Given the users' sessions logs on a particular day, calculate how many hours each user was active that day.

Note: The session starts when state=1 and ends when state=0.

Table: cust_tracking

```
如果不是timestamp格式，可以化为::TIME格式进行相减。

两个Timestamp相减得出小时数：
HOUR = EXTRACT(EPOCH FROM ts1 - ts2) / 3600
HOUR = SUM(ts1:TIME - ts2::TIME) / 3600

LAG(A, -1)和LEAD(A, 1)两者作用相同：
LAG(timestamp, -1) OVER (PARTITION BY cust_id ORDER BY timestamp) AS
next_event
LEAD(timestamp, 1) OVER (PARTITION BY cust_id ORDER BY timestamp) AS end_time

已知上线下线日志，求每个用户上线总时长：将用户每次上下线时间对应起来：
法1：
i.id, ts, LAG(ts, -1) OVER (PARTITION BY id ORDER BY ts) AS next_event,
LAG(ts, -1) OVER (PARTITION BY id ORDER BY ts)::TIME - ts::TIME AS hours
ii. cust_id, SUM(hours) + WHERE state = 1 + GROUP BY 1

法2：
i. id, state, timestamp AS start_time, LEAD(A, 1) OVER (PARTITION BY id ORDER
BY ts) AS end_time
ii. id, EXTRACT(EPOCH FROM end_time::TIME - start_time::TIME) / 3600 AS hour
+ WHERE state = 1
```

法3：ROW_NUMBER JOIN：分别算出开始row_num，结束row_num，两个cte join ON id AND row_num
i. sess_start: id, state, ts, ROW_NUMBER() OVER (PARTITION BY id ORDER BY ts) AS row_num + WHERE state = 1
ii.sess_end: id, state, ts, ROW_NUMBER() OVER (PARTITION BY id ORDER BY ts) AS row_num + WHERE state = 0
iii. id, SUM(b.ts:TIME - a.ts::TIME) / 3600 FROM sess_start a JOIN sess_end b ON a.id = b.id AND a.row_num = b.row_num + GROUP BY id

```
WITH time_between_events AS
(
SELECT
    *,
    LAG(timestamp, -1) OVER (PARTITION BY cust_id ORDER BY timestamp) AS
next_event,
    EXTRACT(EPOCH FROM LAG(timestamp, -1) OVER (PARTITION BY cust_id ORDER BY
timestamp)::TIME - timestamp::TIME) / 3600 AS hours_to_next_event
FROM cust_tracking
)
SELECT
    cust_id,
    SUM(hours_to_next_event) AS total_hours
FROM time_between_events
WHERE state = 1
GROUP BY cust_id
```

```
WITH cte AS(
SELECT
    cust_id,
    state,
    timestamp AS start_time,
    LEAD(state, 1) OVER (PARTITION BY cust_id ORDER BY timestamp) AS state_2,
    LEAD(timestamp, 1) OVER (PARTITION BY cust_id ORDER BY timestamp) AS
end_time
FROM cust_tracking
),
hour_cte AS(
SELECT
    cust_id,
    EXTRACT(EPOCH FROM end_time::TIME - start_time::TIME) / 3600 AS hour
FROM cte
WHERE state = 1
)
SELECT
    cust_id,
    SUM(hour) AS total_hours
FROM hour_cte
GROUP BY cust_id
ORDER BY cust_id
```

```
WITH sess_start AS(
SELECT
    cust_id,
    state,
    timestamp,
    ROW_NUMBER() OVER (PARTITION BY cust_id ORDER BY timestamp) AS row_num
```

```
FROM cust_tracking
WHERE state = 1
),
sess_end AS(
SELECT
    cust_id,
    state,
    timestamp,
    ROW_NUMBER() OVER (PARTITION BY cust_id ORDER BY timestamp) AS row_num
FROM cust_tracking
WHERE state = 0
)
SELECT
    a.cust_id,
    SUM(b.timestamp::TIME - a.timestamp::TIME) / 3600 AS total_hours
FROM sess_start a
JOIN sess_end b
ON a.cust_id = b.cust_id
AND a.row_num = b.row_num
GROUP BY a.cust_id
ORDER BY a.cust_id
```

Delivering and Placing Orders

Interview Question Date: May 2021

DoorDash Hard ID 2037 10

Data Engineer Data Scientist BI Analyst Data Analyst You have been asked to investigate whether there is a correlation between the average total order value and the average time in minutes between placing an order and having it delivered per restaurant.

You have also been told that the column order_total represents the gross order total for each order. Therefore, you'll need to calculate the net order total.

The gross order total is the total of the order before adding the tip and deducting the discount and refund.

Table: delivery_details

```
minutes = EXTRACT (EPOCH FROM ts1 - ts2) / 60
minutes = DATEDIFF('minute', ts1, ts2)

求一个id下平均时长和平均价值：
id, AVG(minutes), AVG(value) + GROUP BY id
```

```
investigate correlation bt avg total order value and avg time in minutes bt
place order and delivered per rest
order_total represents the gross order total
need to calculate the net order total
gross order total is the total of the order before adding the tip and
deducting the discount and refund.

--法1：
```

```
WITH cte AS(
SELECT
    restaurant_id,
    order_total + tip_amount - (discount_amount + refunded_amount) AS
final_order_value,
    customer_placed_order_datetime,
    delivered_to_consumer_datetime
FROM delivery_details
),
stats AS
(
SELECT
    restaurant_id,
    AVG(EXTRACT (EPOCH FROM delivered_to_consumer_datetime -
customer_placed_order_datetime) / 60) AS delivery_time_minutes,
    AVG(final_order_value) AS avg_order_value
FROM cte
GROUP BY restaurant_id
)
SELECT
CORR(delivery_time_minutes, avg_order_value)
FROM stats
```

```
--法2:
WITH total AS(
SELECT
    restaurant_id,
    AVG(order_total + tip_amount - (discount_amount + refunded_amount)) AS
final_order_value,
    AVG(EXTRACT (EPOCH FROM delivered_to_consumer_datetime -
customer_placed_order_datetime) / 60) AS delivery_time_minutes
FROM delivery_details
GROUP BY 1
)
SELECT
CORR(final_order_value,delivery_time_minutes) AS corr
FROM total
```

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A_cnt下有多少B_cnt:
i. B, COUNT(*) AS B_cnt + GROUP BY B
ii. B_cnt, COUNT(A) AS A_cnt + GROUP BY B_cnt

②YYYY-MM-DD格式在一个日期之中:
datetime BETWEEN 'XXXX-XX-XX' AND 'XXXX-XX-XX'
datetime >='XXXX-XX-XX' AND datetime <= 'XXXX-XX-XX'

③YYYY-MM-DD格式取月份:
TO_CHAR(datetime, 'YYYY-MM') = 'XXXX-XX'

④bug where ts1 is dated before ts2: ts2 <= ts1
```

```
WITH comment_cnt_cte AS(
SELECT
    a.id,
    COUNT(*) AS comment_cnt
FROM fb_users a JOIN fb_comments b
ON a.id = b.user_id
WHERE a.joined_at <= b.created_at
AND joined_at BETWEEN '2018-01-01' AND '2020-12-31'
AND created_at BETWEEN '2020-01-01' AND '2020-01-31'
GROUP BY id
)
SELECT
    comment_cnt,
    COUNT(id) AS user_cnt
FROM comment_cnt_cte
GROUP BY comment_cnt
ORDER BY comment_cnt
```

Find fare differences on the Titanic using a self join

Google LinkedIn Hard ID 9603

Find the average absolute fare difference between a specific passenger and all passengers that belong to the same pclass, both are non-survivors and age difference between two of them is 5 or less years. Do that for each passenger (that satisfy above mentioned coniditions). Output the result along with the passenger name.

Table: titanic

```
fare difference between a specific passenger and all passengers that belong
to the same pclass
a.name,
AVG(ABS(price1 - price2))
FROM A JOIN B ON A.id != B.id
```

```
WHERE a.pclass = b.pclass
GROUP BY name
```

```
--avg abs fare diff bt specific passenger and all passenger belong to same
pclass
--both non-survivors and age difference bt 2 of them is 5 or 5- years
-- output: name1, avg_fare

SELECT
    a.name,
    AVG(ABS(a.fare - b.fare)) AS avg_fare
FROM titanic a JOIN titanic b
ON a.passengerid != b.passengerid
WHERE a.pclass = b.pclass
AND a.survived = 0 AND b.survived = 0
AND ABS(a.age - b.age) <= 5
GROUP BY a.name
ORDER BY a.name
```

Department Manager and Employee Salary Comparison

Interview Question Date: January 2023

Oracle Hard ID 2146

Oracle is comparing the monthly wages of their employees in each department to those of their managers and co-workers.

You have been tasked with creating a table that compares an employee's salary to that of their manager and to the average salary of their department.

It is expected that the department manager's salary and the average salary of employee's from that department are in their own separate column.

Order the employee's salary from highest to lowest based on their department. Your output should contain the department, employee id, salary of that employee, salary of that employee's manager and the average salary from employee's within that department rounded to the nearest whole number.

Note: Oracle have requested that you not include the department manager's salary in the average salary for that department in order to avoid skewing the results. Managers of each department do not report to anyone higher up; they are their own manager.

Table: employee_o

```
①求B中的A平均值方法：
AVG(A) OVER (PARTITION BY B)
AVG(a.salary) OVER (PARTITION BY a.department)：限定左侧a无employee信息

②即有employee本身，部门平均（无经理），经理信息做法：
让一个表，左侧全是employee(无经理)，右侧都是匹配的经理
```

```
i. SELECT a.A, a.id, a.B, b.B, AVG(a.B) OVER (PARTITION BY A)
A JOIN A ON a.manager_id = b.id AND a.id != b.id

③不像:
NOT ILIKE 'A' = NOT LIKE '%A%'
```

```
monthly wages of each depart to those of their managers and co-workers
table: employee's salary ~ manager and avg sal of their department
output: department, employee_id, employee_salary, manager_salary,
avg_employee_salary: ROUND
order employe salary highest to lowest ~ department
not include department manager's salary in avg sal for department

法1:
让一个表，左侧全是employee(无经理)，右侧都是匹配的经理
i. SELECT a.A, a.id, a.B, b.B, AVG(a.B) OVER (PARTITION BY A)
A JOIN A ON a.manager_id = b.id AND a.id != b.id

SELECT
    a.department,
    a.id AS employee_id,
    a.salary AS employee_salary,
    b.salary AS manager_salary,
    ROUND(AVG(a.salary) OVER(PARTITION BY a.department)) AS
avg_employee_salary
FROM employee_o a
JOIN employee_o b ON a.manager_id = b.id
AND a.id <> b.id
```

```
法2:
i. department_avg_cte a: A, AVG(B) ON id != id AND A = A AND b.title NOT
ILIKE 'manager'
ii. manager_cte b: A, MAX(CASE WHEN title = 'Manager' THEN B ELSE 0 END) AS
manager_salary
iii. all join on department: 个人，部门，经理: a JOIN b JOIN employee_o ON A
AND c.title != 'Manager'

WITH department_avg AS(
SELECT
    a.department,
    ROUND(AVG(b.salary)) AS avg_employee_salary
FROM employee_o a JOIN employee_o b
ON a.id != b.id
WHERE a.department = b.department
AND b.employee_title NOT ILIKE 'Manager'
GROUP BY a.department
),
manager_salary_cte AS(
SELECT
    department,
    MAX(CASE WHEN employee_title = 'Manager' THEN salary ELSE 0 END) AS
manager_salary
FROM employee_o
GROUP BY department
)
SELECT
```

```
    a.department,
    c.id AS employee_id,
    c.salary AS employee_salary,
    manager_salary,
    avg_employee_salary
FROM department_avg a
JOIN manager_salary_cte b
ON a.department = b.department
JOIN employee_o c
ON b.department = c.department
WHERE c.employee_title != 'Manager'
```

Find The Most Profitable Location

Interview Question Date: April 2021

Uber Noom Hard ID 2033

Find the most profitable location. Write a query that calculates the average signup duration and average transaction amount for each location, and then compare these two measures together by taking the ratio of the average transaction amount and average duration for each location.

Your output should include the location, average duration, average transaction amount, and ratio. Sort your results from highest ratio to lowest.

Tables: signups, transactions

```
因为会出现一个signup_id会有多个重复的记录，所以直接AVG(duration)不准，要
AVG(DISTINCT duration)：
You should calculate mean duration before you join it with the signups. There
can be more than one row from transaction table that matches with signups (or
none), which will lead to incorrect mean duration calculation.
```

```
most profitable location
calculate avg signup duration and avg transaction amount ~ each location

compare 2 measures by taking ratio of the avg transaction amount and avg
duration for each location
output: location, mean_duration, mean_revenue, ratio
order ratio high to low

WITH total AS(
SELECT
    a.signup_id,
    location,
    signup_stop_date - signup_start_date AS duration,
    amt AS revenue
FROM signups a
JOIN transactions b
ON a.signup_id = b.signup_id
)
SELECT
    location,
    AVG(DISTINCT duration) AS mean_duration,
```

```
    AVG(revenue) AS mean_revenue,
    AVG(revenue) /AVG(duration) AS ratio
FROM total
GROUP BY location
ORDER BY ratio DESC
```

First Day Retention Rate

Interview Question Date: February 2022

Amazon Hard ID 2090

Calculate the first-day retention rate of a group of video game players. The first-day retention occurs when a player logs in 1 day after their first-ever log-in. Return the proportion of players who meet this definition divided by the total number of players.

Table: players_logins

```
①两个时间天数差值为1：DATEDIFF(datetime1, datetime2) = 1

②给时间加上一天：datetime + INTERVAL '1 day'
CASE WHEN first_log + INTERVAL '1 day' = login_date THEN 1 ELSE 0 END AS
retention

③求部分和整体的比值：
法1：
i. cte: A, MIN(B) AS first_B + GROUP BY A
ii.COUNT(DISTINCT id) / (SELECT COUNT(DISTINCT id) FROM C) FROM C JOIN cte ON
id WHERE DATEDIFF(D, E) = 1

法2：
i. total: id, date, LEAD(date, 1) OVER (PARTITION BY id ORDER BY date) AS
next, ROW_NUMBER() OVER (PARTITION BY id ORDER BY date)
ii. COUNT(DISTINCT id) / (SELECT COUNT(*) FROM A) FROM total WHERE row_num =
1 AND next_login_date - login_date <= 1

法2.2：
i. total: id, date, LEAD(date, 1) OVER (PARTITION BY id ORDER BY date) AS
next, ROW_NUMBER() OVER (PARTITION BY id ORDER BY date)
ii. SUM(CASE WHEN datetime1 - datetime2  <= 1 THEN 1 ELSE 0 END) AS retention
FROM total WHERE row_num = 1
iii. SUM(retention) / COUNT(*)::DECIMAL AS retention_rate

法3：
i. first_log_cte: id, date_time, MIN(datetime) OVER (PARTITION BY id) AS
first_log
ii. re_cte: *, CASE WHEN first_log + INTERVAL '1 day' = login_date THEN 1
ELSE 0 END AS retention FROM first_log_cte
iii. SUM(retention) / COUNT(DISTINCT id) FROM re_cte
```

```
--法1：
求部分和整体的比值：
i. cte: A, MIN(B) AS first_B + GROUP BY A
```

```
ii.COUNT(DISTINCT id) / (SELECT COUNT(DISTINCT id) FROM C) FROM C JOIN cte ON
id WHERE DATEDIFF(D, E) = 1

WITH first_days AS
(
SELECT
    player_id,
    MIN(login_date) AS first_day
FROM players_logins
GROUP BY player_id
)
SELECT
    COUNT(DISTINCT a.player_id) / (SELECT COUNT(DISTINCT player_id) FROM
players_logins) ::FLOAT AS retention_rate
FROM first_days a
JOIN players_logins b
ON a.player_id = b.player_id
WHERE datediff(login_date, first_day) = 1
```

法2:

```
i. total: id, date, LEAD(date, 1) OVER (PARTITION BY id ORDER BY date) AS
next, ROW_NUMBER() OVER (PARTITION BY id ORDER BY date)
ii. COUNT(DISTINCT id) / (SELECT COUNT(*) FROM A) FROM total WHERE row_num =
1 AND next_login_date - login_date <= 1

WITH total AS(
SELECT
    player_id,
    login_date,
    LEAD(login_date, 1) OVER (PARTITION BY player_id ORDER BY login_date) AS
next_login_date,
    ROW_NUMBER() OVER (PARTITION BY player_id ORDER BY login_date) AS row_num
FROM players_logins
)
SELECT
    -- player_id,
    -- login_date,
    -- next_login_date,
    -- next_login_date - login_date AS log_in_diff,
    COUNT(DISTINCT player_id) / (SELECT COUNT(DISTINCT player_id) FROM
players_logins)::DECIMAL AS retention_rate
FROM total
WHERE row_num = 1
AND next_login_date - login_date <= 1
```

法2.2:

```
i. total: id, date, LEAD(date, 1) OVER (PARTITION BY id ORDER BY date) AS
next, ROW_NUMBER() OVER (PARTITION BY id ORDER BY date)
ii. SUM(CASE WHEN datetime1 - datetime2  <= 1 THEN 1 ELSE 0 END) AS retention
FROM total WHERE row_num = 1
iii. SUM(retention) / COUNT(*)::DECIMAL AS retention_rate

WITH total AS(
SELECT
    player_id,
```

```
    login_date,
    LEAD(login_date, 1) OVER (PARTITION BY player_id ORDER BY login_date) AS
next_login_date,
    ROW_NUMBER() OVER (PARTITION BY player_id ORDER BY login_date) AS row_num
FROM players_logins
),
retention_cte AS(
SELECT
    -- player_id,
    -- login_date,
    -- next_login_date,
    -- next_login_date - login_date AS log_in_diff,
    SUM(CASE WHEN next_login_date - login_date <= 1 THEN 1 ELSE 0 END) AS
retention
FROM total
WHERE row_num = 1
GROUP BY player_id,login_date,next_login_date
)
SELECT
    SUM(retention) / COUNT(*):: DECIMAL AS retention_rate
FROM retention_cte
```

```
# 法3:
i. first_log_cte: id, date_time, MIN(datetime) OVER (PARTITION BY id) AS
first_log
ii. re_cte: *, CASE WHEN first_log + INTERVAL '1 day' = login_date THEN 1
ELSE 0 END AS retention FROM first_log_cte
iii. SUM(retention) / COUNT(DISTINCT id) FROM re_cte

WITH first_log_cte AS(
    SELECT
        player_id,
        login_date,
        MIN(login_date) OVER (PARTITION BY player_id) AS first_log
    FROM players_logins
    ORDER BY player_id, login_date
),
retention_cte AS(
    SELECT
        DISTINCT
        *,
        CASE WHEN first_log + INTERVAL '1 day' = login_date THEN 1 ELSE 0 END
AS retention
    FROM first_log_cte
)
SELECT
    SUM(retention) / COUNT(DISTINCT player_id)::FLOAT AS retention_rate
FROM retention_cte
```

From Microsoft to Google

Interview Question Date: December 2021

LinkedIn Hard ID 2078

Consider all LinkedIn users who, at some point, worked at Microsoft. For how many of them was Google their next employer right after Microsoft (no employers in between)?

Table: linkedin_users

```
LEAD(A, 1) OVER (PARTITION BY id ORDER BY date) AS next
LOWER(A) LIKE LOWER('abc')
```

```
-- all linkedin users worked at M
-- # was Google their next employer after M

WITH total AS(
SELECT
    user_id,
    employer,
    LEAD(employer, 1) OVER (PARTITION BY user_id ORDER BY start_date) AS
next_employer,
    start_date,
    end_date
FROM linkedin_users
)
SELECT
    COUNT(*) AS n_employees
FROM total
WHERE employer = 'Microsoft' AND next_employer = 'Google'
--WHERE LOWER(employer) LIKE LOWER('microsoft') AND LOWER(next_employer) LIKE
LOWER('google')
```

Highest Earning Merchants

Interview Question Date: February 2022

DoorDash Hard ID 2094

Each day, you have been asked to find a merchant who earned more money the previous day.

Before comparing totals between merchants, round the total amounts to the nearest 2 decimals places.

Your output should include the date in the format 'YYYY-MM-DD' and the merchant's name, but only for days where data from the previous day is available.

Note: In the case of multiple merchants having the same highest shared amount, your output should include all the names in different rows.

Tables: order_details, merchant_details

```
①ROUND(SUM(A::DECIMAL),2):FLOAT不能被SUM, DECIMAL才可以被SUM

⑦两种变换格式的表达方式:
CAST(SUM(A) AS DECIMAL)
SUM(A::DECIAML)
```

```
②时间加1天：ts + INTERVAL '1 day'
③只取年月日 YYYY-MM-DD：CAST(ts AS date)
④只取年月日及时间加1天：CAST(d.today + INTERVAL '1 day' AS date)

⑤得每天各家明细后有每天的最高额那家的信息：
每天各家明细 JOIN 每天最高额明细 ON date = date AND value = max_value
daily_cte JOIN daily_max_cte ON date = date AND value = max_value

⑥思路：
i. daily_total_earned：每天每个商户营业额
ii.highest_totals: 每天营业额最高的商户
iii. 取后一天的时间，因为算的是prev day的营业额，取名：
CAST(d.today + INTERVAL '1 day' AS date), name
daily_total_earned a JOIN highest_totals b ON d.today = h.today AND
d.daily_total = h.highest_total
JOIN 原表 ON merchant_id
```

```
WITH daily_total_earned AS
    (SELECT
        order_timestamp::DATE AS today,
        merchant_id,
        ROUND(SUM(total_amount_earned::DECIMAL), 2) AS daily_total
    FROM order_details
    GROUP BY today, merchant_id                                          --此处不
能group by order_timestamp，否则影响结果准确性
    ),
    highest_totals AS
    (SELECT
        today,
        MAX(daily_total) AS highest_total
    FROM daily_total_earned
    GROUP BY today
    )
SELECT
    CAST(d.today + INTERVAL '1 day' AS date),
    name
FROM daily_total_earned d
JOIN highest_totals h
ON d.today = h.today AND d.daily_total = h.highest_total   --ON date AND
value
JOIN merchant_details m
ON m.id = d.merchant_id
```

Lowest Revenue Generated Restaurants

Interview Question Date: May 2021

DoorDash Hard ID 2036

Write a query that returns a list of the bottom 2% revenue generating restaurants. Return a list of restaurant IDs and their total revenue from when customers placed orders in May 2020.

You can calculate the total revenue by summing the order_total column. And you should calculate the bottom 2% by partitioning the total revenue into evenly distributed buckets.

Table: doordash_delivery

```
①取年月日：CAST(ts AS date)
②取年月：
TO_CHAR(ts, 'YYYY-MM')
DATE_PART('year', ts)  AND DATE_PART('month', ts)
③取日：EXTRACT(year from ts)


④算percentile:
bottome 2%:
NTILE(100) OVER (ORDER BY A ASC) AS percentile + WHERE percentile <= 2
NTILE(50) OVER (ORDER BY A ASC) AS percentile + WHERE percentile = 1
PERCENT_RANK() OVER (ORDER BY A ASC) AS rnk + WHERE rnk <= 0.02
RANK() OVER (ORDER BY A) as rk + WHERE rk <= (SELECT CEIL(MAX(rk) * 0.02)
from cte)
```

```
returns a list of the bottom 2% revenue resturant
a list of resturant IDs and revenue when customers place order in 2020.05
output: restaurant_id, total_order

total revenue by sum(order_total)
calculate the bottom 2% by partition the total rev into evenly distributed
buckets

WITH total AS(
SELECT
    CAST(customer_placed_order_datetime AS date) AS
customer_placed_order_time,
    restaurant_id,
    SUM(order_total) AS total_order,
    NTILE(100) OVER (ORDER BY SUM(order_total) ASC) AS percentile
FROM doordash_delivery
WHERE CAST(customer_placed_order_datetime AS date) BETWEEN '2020-05-01' AND
'2020-05-31'
GROUP BY CAST(customer_placed_order_datetime AS date), restaurant_id
)
SELECT
    restaurant_id,
    total_order
FROM total
WHERE percentile <= 2
```

Maximum Number of Employees Reached

Interview Question Date: June 2021

Uber Hard ID 2046

Write a query that returns every employee that has ever worked for the company. For each employee, calculate the greatest number of employees that worked for the company during their tenure and the first date that number was reached. The termination date of an employee should not be counted as a working day.

Your output should have the employee ID, greatest number of employees that worked for the company during the employee's tenure, and first date that number was reached.

Table: uber_employees

从旧到新每个时间点，计算running total总数，：SUM(COUNT) OVER (ORDER BY date) AS max
在保证员工任职最大值找到最早日期：ROW_NUMBER() OVER (PARTITION BY id ORDER BY maxemp DESC, dateq ASC)
在任职期间：FROM table a JOIN sum_cte b ON b.dateq BETWEEN a.date_1 AND COALESCE(a.date_2, CURRENT_DATE)

i. 将入职离职日期归为一列，入职赋值+1，离职赋值-1：SELECT date_1 AS dateq, 1 AS COUNT UNION ALL date_2, -1
ii. 每个入职离职点，计算公司总人数：sum_cte: SUM(COUNT) OVER (ORDER BY dateq) AS maxemp
iii.在保证员工最大值找到最早日期：id, dateq, maxemp, ROW_NUMBER() OVER (PARTITION BY id ORDER BY maxemp DESC, dateq ASC) AS rnk
FROM sum_cte JOIN 原table + rnk = 1

```
-- every employee has ever worked for the company
-- each employee ~ calculate the greatest # employees worked for the company during tenure and 1st date @ was reached
-- termination date not count as working day
-- output: id   maxemp  dateaq

WITH cte1 AS
(
SELECT
    hire_date AS dateq,
    1 AS COUNT
FROM uber_employees
UNION ALL
SELECT
    termination_date,
    -1 AS COUNT
FROM uber_employees
WHERE termination_date IS NOT NULL
),
sum_cte AS(
SELECT
    dateq,
    SUM(COUNT) OVER (ORDER BY dateq) AS maxemp        --在每个日期点，公司总人数
FROM cte1
),
result_cte AS(
SELECT
    a.id,
    b.dateq,
    b.maxemp,
    ROW_NUMBER() OVER (PARTITION BY a.id ORDER BY b.maxemp DESC, b.dateq ASC) AS rnk  --分类靠id, order by 人数desc, 日期 asc
FROM uber_employees a
JOIN sum_cte b
```

```
ON b.dateq BETWEEN a.hire_date AND COALESCE(a.termination_date, CURRENT_DATE)
  --时间限制在员工入职后，和离职日期/现在时间
)
SELECT
    id,
    maxemp,
    dateq
FROM result_cte
WHERE rnk = 1    --在保证任期内公司员工最大值同时，找到最早的日期
```