# SQL Interview Prep Questions - 2 (16 - 30)

16) Write an SQL query to find the people who have the most friends and the most friends number.

Difficulty: Medium

Company : Facebook

Table: `RequestAccepted`

```
+----------------+---------+
| Column Name    | Type    |
+----------------+---------+
| requester_id   | int     |
| accepter_id    | int     |
| accept_date    | date    |
+----------------+---------+
(requester_id, accepter_id) is the primary key for this table.
This table contains the ID of the user who sent the request, the ID of the user who received the
request, and the date when the request was accepted.
```

**Input:**
RequestAccepted table:

```
+--------------+-------------+-------------+
| requester_id | accepter_id | accept_date |
+--------------+-------------+-------------+
| 1            | 2           | 2016/06/03  |
| 1            | 3           | 2016/06/08  |
| 2            | 3           | 2016/06/08  |
| 3            | 4           | 2016/06/09  |
+--------------+-------------+-------------+
```

**Output:**

```
+----+-----+
| id | num |
+----+-----+
| 3  | 3   |
+----+-----+
```

**Explanation:**
A user can be both a requester as well as an acceptor. So the friend count for a user should be accounted for when the user is on the requestor side or the accepter side.

**Step1**: Find the count of friends for each user when they are on the requestor side
**Step2**: Similarly find the count of friends when the users are on the acceptor side
**Step3**: Combine the count of users' friends when they were a requestor / accepter by taking a union of both the tables.

**Step4**: Find the total count by taking the sum of friends count by taking a group by on each users
**Step5**: Find the user with the maximum number of friends using the filters.

```
with requestor as (
  select requester_id as user_id, count(accepter_id) frnd_count from RequestAccepted
  group by requester_id
),

acceptor as (
  select accepter_id as user_id, count(requester_id) frnd_count from RequestAccepted
  group by accepter_id
),

all_frnd_count as
(
  select * from requestor union all select * from acceptor
),

uniq_user_frnd_cnt as (
  select user_id as id, sum(frnd_count) as num from all_frnd_count group by user_id
)

select * from uniq_user_frnd_cnt
where num = (select max(num) from uniq_user_frnd_cnt)
```

---

17) A quiet student is the one who took at least one exam and did not score the high or the low score.

Write an SQL query to report the students (student_id, student_name) being quiet in all exams. Do not return the student who has never taken any exam.

Return the result table ordered by student_id.

Difficulty: Hard

Company: Yahoo

Table: `Student`

```
+----------------------+---------+
| Column Name          | Type    |
+----------------------+---------+
| student_id           | int     |
| student_name         | varchar |
+----------------------+---------+
student_id is the primary key for this table.
student_name is the name of the student.
```

Table: `Exam`

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| exam_id       | int     |
| student_id    | int     |
| score         | int     |
+---------------+---------+
(exam_id, student_id) is the primary key for this table.
Each row of this table indicates that the student with student_id had a score points in the exam with id
exam_id.
```

**Explanation:**

**Step1:** Find all the students whose marks are the highest and lowest in any of the exam so they can be eliminated. This can be found by ranking the marks using the windows function in the exam table.

**Step2:** For finding the students who have scored the lowest marks partition by exam_id and sort marks in the ascending order so the students with the lowest mark are ranked 1.

**Step3:** Similarly create another column by partitioning on the exam_id and in the descending order of the marks so students with the highest marks will be ranked 1.

**Step4**: For a student to be silent there are two conditions given below which can be fed as subqueries in the filter condition.

1. Should have taken at least one test i.e. student_id should be in the exam table
2. Should not be highest or lowest scorer in any exam i.e. Should not be ranked 1 in step 2 or step 3.

```
Select  student_id,
        Student_name
         from student
        where student_id in (select student_id from exam) and student_id not in ( select
        student_id from( select student_id,
                                rank() over(partition by exam_id order by score asc) as asc_rank,
                                rank() over(partition by exam_id order by score desc) as
                                desc_rank from exam
```

```
                    ) tab1
        where asc_rank=1 or desc_rank=1
)
```

---

18) Write an SQL query to report the sum of all total investment values in 2016 tiv_2016, for all policyholders who:

- have the same tiv_2015 value as one or more other policyholders, and
- are not located in the same city like any other policyholder (i.e., the (lat, lon) attribute pairs must be unique).

Round tiv_2016 to two decimal places.

Difficulty: Medium

Company : Twitter

Table: `Insurance`

```
+-------------+-------+
| Column Name | Type  |
+-------------+-------+
| pid         | int   |
| tiv_2015    | float |
| tiv_2016    | float |
| lat         | float |
| lon         | float |
+-------------+-------+
pid is the primary key column for this table.
Each row of this table contains information about one policy where:
pid is the policyholder's policy ID.
tiv_2015 is the total investment value in 2015 and tiv_2016 is the total investment value in 2016.
lat is the latitude of the policy holder's city.
lon is the longitude of the policy holder's city.
```

**Input:**
Insurance table:

| pid | tiv_2015 | tiv_2016 | lat | lon |
|---|---|---|---|---|
| 1 | 10 | 5 | 10 | 10 |
| 2 | 20 | 20 | 20 | 20 |
| 3 | 10 | 30 | 20 | 20 |
| 4 | 10 | 40 | 40 | 40 |

**Output:**

| tiv_2016 |
|---|
| 45.00 |

**Explanation:**
The first record in the table, like the last record, meets both of the two criteria.
The tiv_2015 value 10 is the same as the third and fourth records, and its location is unique.

The second record does not meet any of the two criteria. Its tiv_2015 is not like any other policyholders and its location is the same as the third record, which makes the third record fail, too.
So, the result is the sum of tiv_2016 of the first and last record, which is 45.

**Explanation**:
**Step1**: For the first condition the investment value in 2015 should be the same i.e. it has to be duplicated or in other words unique investment values should not be considered. This can be done by taking a count of the tiv_2015 column and considering only those that have a count >1.
**Step2**: For the second condition every location should be unique, which means if location occurs more than once in the table, we should not consider it. This can be done by taking a count of the combination of lat and lon and considering only the count =1
**Step3**: Find the overall sum of tiv_2016 for records matching step1 and step2.

```
WITH
firstConditionCTE AS
        (
        SELECT tiv_2015 FROM Insurance GROUP BY tiv_2015
        HAVING COUNT(*) > 1
        ),

SecondConditionCTE AS
        (
        SELECT concat(lat, ' ', lon) as uniq_lat_lon FROM Insurance
        GROUP BY concat(lat, ' ', lon) HAVING COUNT(*) = 1
        )

SELECT
        ROUND(SUM(tiv_2016),2) AS tiv_2016
```

```
FROM Insurance
WHERE tiv_2015 IN (SELECT * FROM firstConditionCTE)
  AND
  concat(lat, ' ', lon) IN (SELECT uniq_lat_lon FROM SecondConditionCTE)
```

---

19) Write an SQL query to report all possible axis-aligned rectangles with a non-zero area that can be formed by any two points from the Points table.

Each row in the result should contain three columns (p1, p2, area) where:

- p1 and p2 are the id's of the two points that determine the opposite corners of a rectangle.
- area is the area of the rectangle and must be non-zero.

Return the result table ordered by area in descending order. If there is a tie, order them by p1 in ascending order. If there is still a tie, order them by p2 in ascending order.

Difficulty: Medium

Company: Twitter

Table: `Points`

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| id            | int     |
| x_value       | int     |
| y_value       | int     |
+---------------+---------+
id is the primary key for this table.
Each point is represented as a 2D coordinate (x_value, y_value).
```

**Explanation**:

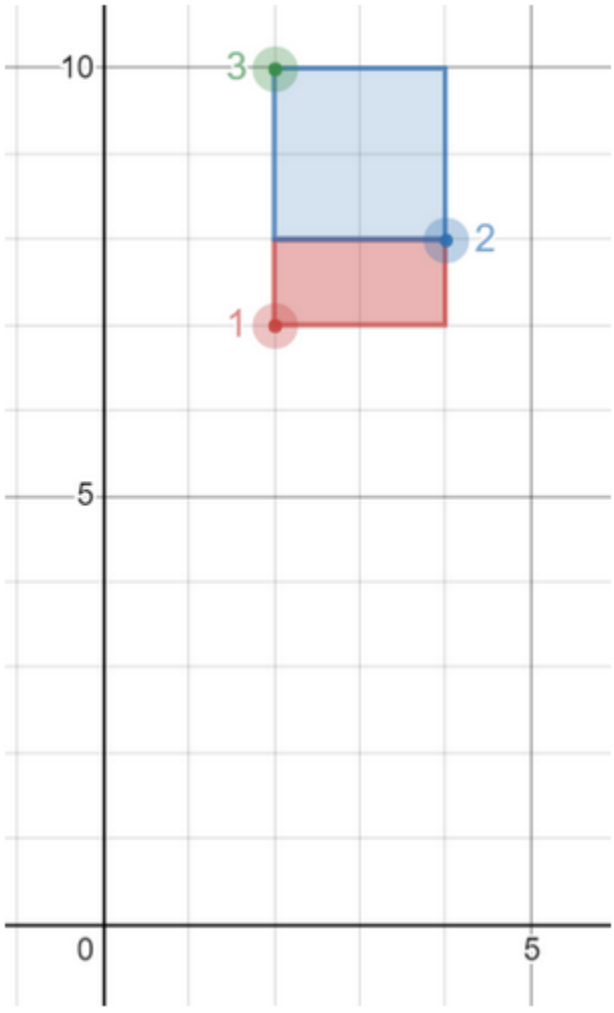
To find the area of a rectangle, we need to know the length and breadth i.e. 2 sides of the rectangle. For this we need 4 points(2 Xs and 2 Ys) to identify the corners of the rectangle. Each row in the Table contains one combination of x and y. So we will combine each row with other rows to get the 4 points.

**Step1**: Since we wanted to find all the possible rectangles with the points in the table, We will do a self join of the table. We use the < expression to join the two tables as we want to find the area for one combination of rectangle only once. I.e. Area of (p1, p2) is the same as (p2, p1).

**Step2**: Let's consider example points X1=4, X2=6, and X1 = 4, X2=1. As you can see in the first case X1 is smaller than X2 and in the second case X1 is greater than X2. Since we want only to find the distance between the two points we take the absolute difference between the two.The same goes with Y as well.

**Step3**: To avoid getting Area as 0, we filter out points with equal X values and equal Y values.

**Example 1:**




```
select
        l.id as P1,
        r.id as P2,
        abs(l.x_value-r.x_value) * abs(l.y_value-r.y_value) as AREA
        from points l
        join points r on l.id < r.id
```

```
where   l.x_value <> r.x_value
    and l.y_value <> r.y_value
ORDER BY AREA DESC, p1 ASC, p2 ASC;
```

---

20) Assume you are given the tables below about Facebook pages and page likes. Write a query to return the page IDs of all the Facebook pages that don't have any likes. The output should be in ascending order.

Difficulty: Easy

Company: Facebook

**pages Table:**

| Column Name | Type |
|---|---|
| page_id | integer |
| name | varchar |

**page_likes Table:**

| Column Name | Type |
|---|---|
| user_id | integer |
| page_id | integer |
| liked_date | datetime |

**Explanation:**

**Step1**: Since all the pages that have at least one like will be present in the page_likes Tables. We can write a subquery to get all those pages.

**Step2**: Write a query to find all the pages from the pages table which are not part of the page_likes.

```
SELECT page_id FROM pages
    where page_id not in (select distinct page_id from page_likes);
```

---

21) Assume you have an events table on app analytics. Write a query to get the app's click-through rate (CTR %) in 2022. Output the results in percentages rounded to 2 decimal places.

Notes:

- To avoid integer division, you should multiply the click-through rate by 100.0, not 100.
- Percentage of click-through rate = 100.0 * Number of clicks / Number of impressions

Company: Facebook

Difficulty: Easy

**`events` Table:**

| Column Name | Type |
|---|---|
| app_id | integer |
| event_type | string |
| timestamp | datetime |

**Explanation**: Event_type can be of either impression or click. We need to find the total number of impression and clicks and then use the formula
**Step1**: To get the number of clicks, create a new column 'clicks' filled as 1 whenever the event_type=click and 0 if not. Similarly create another column 'impressions' using the same logic.
**Step2**: Now that we have the 2 fields for calculating the CTR, we can group by on the app_id to find the sum of impressions and clicks for each app and calculate CTR using the formula. Filter for the required time period in the where clause.

```
WITH ctr_cte as
        (SELECT
         app_id,
            CASE WHEN event_type = 'click' THEN 1 ELSE 0 END AS clicks,
            CASE WHEN event_type = 'impression' THEN 1 ELSE 0 END AS impressions
        FROM events
        WHERE timestamp >= '2022-01-01' AND timestamp < '2023-01-01'
```

)

```
select app_id, ROUND(100.0*sum(clicks)/sum(impressions),2) as ctr
from ctr_cte
GROUP BY app_id
```

---

22) Assume that you are given the table below containing information on various orders made by eBay customers. Write a query to obtain the user IDs and number of products purchased by the top 3 customers; these customers must have spent at least $1,000 in total.

Output the user id and number of products in descending order. To break ties (i.e., if 2 customers both bought 10 products), the user who spent more should take precedence.

Company: Ebay

Difficulty: Easy

**`user_transactions` Table:**

| Column Name | Type |
|---|---|
| transaction_id | integer |
| product_id | integer |
| user_id | integer |
| spend | decimal |

**`user_transactions` Example Input:**

| transaction_id | product_id | user_id | spend |
|---|---|---|---|
| 131432 | 1324 | 128 | 699.78 |
| 131433 | 1313 | 128 | 501.00 |
| 153853 | 2134 | 102 | 1001.20 |
| 247826 | 8476 | 133 | 1051.00 |
| 247265 | 3255 | 133 | 1474.00 |
| 136495 | 3677 | 133 | 247.56 |

**Explanation**:
**Step1**: Find the count of products grouping by the customer_id
**Step2**: Since the minimum overall purchase amount of the customers should be 1000, filter those customers using the having clause.
**Step3**: As only the top 3 customers are required we can use limit 3.

```
select user_id,
        count(product_id) as product_num
from user_transactions
group by user_id having sum(spend)>=1000
order by count(product_id) desc, sum(spend) desc limit 3
```

---

23) The LinkedIn Creator team is looking for power creators who use their personal profile as a company or influencer page. If someone's LinkedIn page has more followers than the company they work for, we can safely assume that person is a power creator.

Write a query to return the IDs of these LinkedIn power creators ordered by the IDs.

Assumption:

- Each person with a LinkedIn profile in this database works at one company only.

Company: Linkedin

Difficulty: Easy

**`personal_profiles` Table:**

| Column Name | Type |
|---|---|
| profile_id | integer |
| name | string |
| followers | integer |
| employer_id | integer |

**`personal_profiles` Example Input:**

| profile_id | name | followers | employer_id |
|---|---|---|---|
| 1 | Nick Singh | 92,000 | 4 |
| 2 | Zach Wilson | 199,000 | 2 |
| 3 | Daliana Liu | 171,000 | 1 |
| 4 | Ravit Jain | 107,000 | 3 |
| 5 | Vin Vashishta | 139,000 | 6 |
| 6 | Susan Wojcicki | 39,000 | 5 |

**company_pages Table:**

| Column Name | Type |
|---|---|
| company_id | integer |
| name | string |
| followers | integer |

**company_pages Example Input:**

| company_id | name | followers |
|---|---|---|
| 1 | The Data Science Podcast | 8,000 |
| 2 | Airbnb | 700,000 |
| 3 | The Ravit Show | 6,000 |
| 4 | DataLemur | 200 |
| 5 | YouTube | 1,6000,000 |
| 6 | DataScience.Vin | 4,500 |

**Explanation:**

**Step1**: Join the personal_profile and the company_pages table using the employer_id and the company_id columns.

**Step2**: Now we can simply filter if the employee has more followers compared to the company

```
SELECT p.profile_id
FROM personal_profiles p JOIN company_pages c
        ON p.employer_id=c.company_id
WHERE p.followers > c.followers ORDER BY profile_id;
```

---

24) Your team at Accenture is helping a Fortune 500 client revamp their compensation and benefits program. The first step in this analysis is to manually review employees who are potentially overpaid or underpaid.

An employee is considered to be potentially overpaid if they earn more than 2 times the average salary for people with the same title. Similarly, an employee might be underpaid if they earn less than half of the average for their title. We'll refer to employees who are both underpaid and overpaid as compensation outliers for the purposes of this problem.

Write a query that shows the following data for each compensation outlier: employee ID, salary, and whether they are potentially overpaid or potentially underpaid (refer to Example Output below).

Company: Accenture

Difficulty: Medium

**`employee_pay` Table:**

| Column Name | Type |
|---|---|
| employee_id | integer |
| salary | integer |
| title | varchar |

**`employee_pay` Example Input:**

| employee_id | salary | title |
|---|---|---|
| 101 | 80000 | Data Analyst |
| 102 | 90000 | Data Analyst |
| 103 | 100000 | Data Analyst |
| 104 | 30000 | Data Analyst |
| 105 | 120000 | Data Scientist |

**Explanation:**

**Step1**: Since each employee's salary has to be compared with avg.salary for the same job title, it is ideal to use the window functions to find the avg salary partitioned by job title.

**Step2**: Now that we have the employee salary and the average salary in each row. We can do a simple comparison to find if they are 'underpaid'(salary/2 < avg.salary) or 'overpaid'(salary*2 > avg.salary) and assign it as 2 categories using the case statement. If the employees do not fall into either category then we can assign them as 'skip'

**Step3**: Filter out all the employees who are categorized as 'skip', since we want only underpaid and overpaid employees

```
select * from (
        SELECT employee_id, salary,
                CASE
                  WHEN (salary*2) < avg(salary) OVER(PARTITION by  title) THEN 'Underpaid'
                  WHEN (salary/2) > avg(salary) OVER(PARTITION by  title) THEN 'Overpaid'
                  ELSE 'skip'
                END as status
        FROM employee_pay
) a
where status <> 'skip'
```

---

25) When you log in to your retailer client's database, you notice that their product catalog data is full of gaps in the category column. Can you write a SQL query that returns the product catalog with the missing data filled in?

Assumptions

- Each category is mentioned only once in a category column.

- All the products belonging to the same category are grouped together.
- The first product from a product group will always have a defined category.
    - Meaning that the first item from each category will not have a missing category value.

Company: Accenture

Difficulty: Medium

**products Table:**

| Column Name | Type |
|---|---|
| product_id | integer |
| category | varchar |
| name | varchar |

**products Example Input:**

| product_id | category | name |
|---|---|---|
| 1 | Shoes | Sperry Boat Shoe |
| 2 | | Adidas Stan Smith |
| 3 | | Vans Authentic |
| 4 | Jeans | Levi 511 |
| 5 | | Wrangler Straight Fit |
| 6 | Shirts | Lacoste Classic Polo |
| 7 | | Nautica Linen Shirt |

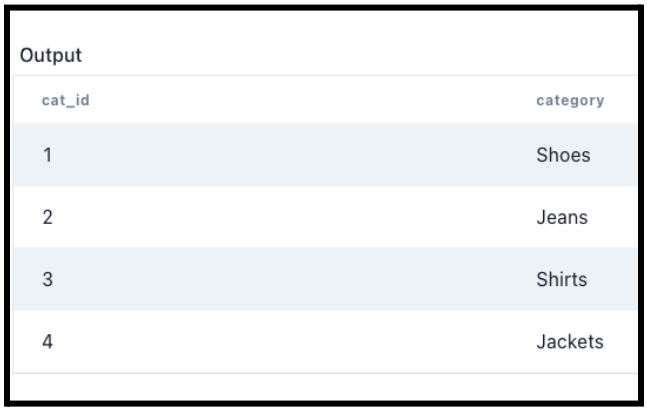
**Example Output:**

| product_id | category | name |
|---|---|---|
| 1 | Shoes | Sperry Boat Shoe |
| 2 | Shoes | Adidas Stan Smith |
| 3 | Shoes | Vans Authentic |
| 4 | Jeans | Levi 511 |
| 5 | Jeans | Wrangler Straight Fit |
| 6 | Shirts | Lacoste Classic Polo |
| 7 | Shirts | Nautica Linen Shirt |

**Explanation**:
As the data is to be filled in a sequence from top to bottom we have to find a method to group all the NULL values to a category starting from the top. i.e. in the example input screenshot above product_Ids 1 to 3 should be grouped by a common value. Similarly product_ids 4 and 5 should be grouped by a common value and 6,7 as a group.
**Step1**: We can assign a group(cat_id) by taking the count of the products in each category and order them by product_ids using window functions (t1 in the query). As we know if we use an aggregate function with ‘order by’ it will create a running aggregation(here running count). Since we did not use ‘partition by’, the entire table will be considered as one partition and everytime there is a new category the count is incremented by 1. Note: Since NULL means absence of value cat_id will be filled with the same value as the previous row. Sample Screenshot of t1 query output

| product_id | category | name | cat_id |
|---|---|---|---|
| 1 | Shoes | Sperry Boat Shoe | 1 |
| 2 | | Adidas Stan Smith | 1 |
| 3 | | Vans Authentic | 1 |
| 4 | Jeans | Levi 511 | 2 |
| 5 | | Wrangler Straight Fit | 2 |
| 6 | Shirts | Lacoste Classic Polo | 3 |
| 7 | | Nautica Linen Shirt | 3 |
| 8 | Jackets | Belstaff Blue Fit | 4 |
| 9 | | Uniqlo Comfort Jacket | 4 |

**Step2**: We can use the output of the t1 query, filter out the rows where the category value is NULL to get the category_id for each category column.  Sample Screenshot of t2 query output



Output

| cat_id | category |
|---|---|
| 1 | Shoes |
| 2 | Jeans |
| 3 | Shirts |
| 4 | Jackets |

**Step3**: Join the cat_if from t1 and t2 to get the final output

```
with t1 as( select
        product_id,  category, name,
        count(category) over (order by product_id) as cat_id
        from products)
, t2 as (
        select cat_id, category from t1
        where category is not null
        )
select t1.product_id, t2.category, t1.name
        from t1 left join t2 on t2.cat_id = t1.cat_id
```

---

26) You're given two tables on Spotify users' streaming data. songs_history table contains the historical streaming data and songs_weekly table contains the current week's streaming data.

Write a query to output the user id, song id, and cumulative count of song plays as of 4 August 2022 sorted in descending order.
Definitions:

- song_weekly table currently holds data from 1 August 2022 to 7 August 2022.
- songs_history table currently holds data up to 31 July 2022. The output should include the historical data in this table.

Assumption:

- There may be a new user or song in the songs_weekly table not present in the songs_history table.

Company: Spotify

Difficulty: Medium

**songs_history Table:**

| Column Name | Type |
|---|---|
| history_id | integer |
| user_id | integer |
| song_id | integer |
| song_plays | integer |

**songs_history Example Input:**

| history_id | user_id | song_id | song_plays |
|---|---|---|---|
| 10011 | 777 | 1238 | 11 |
| 12452 | 695 | 4520 | 1 |

song_plays : Refers to the historical count of streaming or song plays by the user.

**songs_weekly Table:**

| Column Name | Type |
|---|---|
| user_id | integer |
| song_id | integer |
| listen_time | datetime |

**songs_weekly Example Input:**

| user_id | song_id | listen_time |
|---|---|---|
| 777 | 1238 | 08/01/2022 12:00:00 |
| 695 | 4520 | 08/04/2022 08:00:00 |
| 125 | 9630 | 08/04/2022 16:00:00 |
| 695 | 9852 | 08/07/2022 12:00:00 |

**Explanation**:
**Step1**: The songs_weekly table contains only information from August 1 to 7, however we need only info up to 4th August. So we can filter out data using the date function in the where clause
**Step2**: Get the count of songs listened by each user by using group by between the dates 1 to 4th August in the songs_weekly table.
**Step3**: Since we need the overall information about the historical songs listened by each users along with the songs listened in the current week, we can union them together. Note: The column names and order should be in the matching
**Step4**: Now we have all the information in one single table, we can simply take a sum of all the songs listened grouped by each user_id

```
WITH filtered AS (
        SELECT user_id, song_id, song_plays FROM songs_history
```

```
    UNION ALL
SELECT user_id, song_id, COUNT(song_id) AS song_plays FROM songs_weekly
  WHERE DATE_PART('day', listen_time) < 5
  GROUP BY user_id, song_id
)

SELECT user_id, song_id, SUM(song_plays) AS song_plays FROM filtered
  GROUP BY user_id, song_id
  ORDER BY song_plays DESC;
```

---

27) Assume you are given the table below containing the information on the searches attempted and the percentage of invalid searches by country. Write a query to obtain the percentage of invalid searches.

Output the country in ascending order, total searches and overall percentage of invalid searches rounded to 2 decimal places.

Notes:

- num_search = Number of searches attempted; invalid_result_pct = Percentage of invalid searches.
- In cases where countries have search attempts but do not have a percentage of invalid searches in invalid_result_pct, it should be excluded, and vice versa.
- To find the percentages, multiply by 100.0 and not 100 to avoid integer division.

Difficulty: Medium

Company: Google

`search_category` **Table:**

| Column Name | Type |
|---|---|
| country | string |
| search_cat | string |
| num_search | integer |
| invalid_result_pct | decimal |

`search_category` **Example Input:**

| country | search_cat | num_search | invalid_result_pct |
|---|---|---|---|
| UK | home | null | null |
| UK | tax | 98000 | 1.00 |
| UK | travel | 100000 | 3.25 |

**Explanation**: The 'invalid_result_pct' is the percentage of invalid results for each of the search categories. However we need to find the overall invalid_result_pct which is not a simple addition. So we need to find the actual invalid search count first, add them up at country level and then find the percentage.
**Step1**: Find the actual invalid searches for each category by (num_search*invalid_result_pct)/100 and remove the searches where the num_searches is NULL in the where clause.
**Step2**: Now find the sum of overall search and also find the overall invalid search percentage by (summing the overall invalid searches) / (sum of overall searches) *100.0 by grouping by on the country

```
select country,sum(num_search) as total_search,
round(SUM(inval)*100.0/sum(num_search),2) as invalid_searches_pct
from
        (
        SELECT * ,
                (num_search*invalid_result_pct)/100 as inval
                FROM search_category
                where num_search is not null and invalid_result_pct is not null
        ) as c
group by country
ORDER BY country
```

---

28) For every customer that bought Photoshop, return a list of the customers, and the total spent on all the products except for Photoshop products.

Sort your answer by customer ids in ascending order

Difficulty: Medium

Company: Adobe

**adobe_transactions Table:**

| Column Name | Type |
|---|---|
| customer_id | integer |
| product | string |
| revenue | integer |

**adobe_transactions Example Input:**

| customer_id | product | revenue |
|---|---|---|
| 123 | Photoshop | 50 |
| 123 | Premier Pro | 100 |
| 123 | After Effects | 50 |
| 234 | Illustrator | 200 |
| 234 | Premier Pro | 100 |

**Explanation**

**Step1**: First step, find all the customers who have purchased 'Photoshop' i.e. inner query

**Step2**: Filter only those customers who have bought the photoshop product from the previous step from the same table. Now remove all the rows which correspond to transactions related to Photoshop.

**Step3**: Group by customer_id and find the sum of overall revenue which results in customer who have purchased photoshop and the overall revenue is from other products other than photoshop

```
SELECT customer_id,
        SUM(revenue) AS revenue
        FROM adobe_transactions
WHERE customer_id IN
                (SELECT customer_id
                FROM adobe_transactions
                WHERE product = 'Photoshop'
                )
AND product <> 'Photoshop'
GROUP BY customer_id;
```

---

29) You are given a table of PayPal payments showing the payer, the recipient, and the amount paid. A two-way unique relationship is established when two people send money back and forth. Write a query to find the number of two-way unique relationships in this data.

Assumption:
A payer can send money to the same recipient multiple times.

Company: Paypal

Difficulty: Medium

**payments Table:**

| Column Name | Type |
|---|---|
| payer_id | integer |
| recipient_id | integer |
| amount | integer |

**payments Example Input:**

| payer_id | recipient_id | amount |
|---|---|---|
| 101 | 201 | 30 |
| 201 | 101 | 10 |
| 101 | 301 | 20 |
| 301 | 101 | 80 |
| 201 | 301 | 70 |

**Explanation**:
Find the userA, userB combination. userA can be a sender and userB can be a receiver or vice versa. Both of these are considered as a single unique relationship between userA and userB..
**Step1**: To find if two users have been both a sender and receiver, self join the two tables by swapping the payer_id and receipient_id.
**Step2**: All the records that match are the ones which have a two way relationship.
**Step3**: To represent relationship between two users only once we filter receipeint_id > payer_id(can also be receipeint_id < payer_id). This will result in a two way relationship between users occurring only once.
**Step4**: Finally take a count to get the unique users relationship combination

```
select count(*) as comb
from(
        SELECT
                distinct a.payer_id,
                a.recipient_id
        FROM payments a join payments b
                on a.payer_id = b.recipient_id and
                b.payer_id = a.recipient_id
        Where a.amount is not null and b.amount is not null
        and  a.payer_id < a.recipient_id
) x
```

---

30) Assume you are given the table below containing information on user purchases. Write a query to obtain the number of users who purchased the same product on two or more different days. Output the number of unique users.

Company: Stitch fix

Difficulty: Medium

**`purchases` Table:**

| Column Name | Type |
|---|---|
| user_id | integer |
| product_id | integer |
| quantity | integer |
| purchase_date | datetime |

**`purchases` Example Input:**

| user_id | product_id | quantity | purchase_date |
|---|---|---|---|
| 536 | 3223 | 6 | 01/11/2022 12:33:44 |
| 827 | 3585 | 35 | 02/20/2022 14:05:26 |
| 536 | 3223 | 5 | 03/02/2022 09:33:28 |
| 536 | 1435 | 10 | 03/02/2022 08:40:00 |
| 827 | 2452 | 45 | 04/09/2022 00:00:00 |

**Explanation**:
**Step1**: To find if a customer has purchased the same product for more than 1 day, then there is going to be more than one transaction record for that product for that user with different dates.
**Step2**: With this logic we can group by user_id, product_id and find the count of unique purchase dates. If the purchase_dates are more than 1 which means the customer has purchased that product for more than 2 days.
**Step3**: Only select those unique customers who have had any purchase where the purchase date Count >1.

```
with cte as (
        Select  user_id,
                product_id,
                count(DISTINCT date(purchase_date)) as rep_purchase_count
        from purchases

        group by user_id, product_id
 )

select count(distinct (user_id)) as repeat_purchasers
from cte where rep_purchase_count>1
```

---