# SQL Interview Prep Questions Repository

1) Given we have a relationship as mentioned below. Return the total number of comments received for each user in the last 30 days. Assume that today is `2022-06-15`.

Fb_comments_count(
User_id int,
Created_at DateTime,
Number_of_comments int);

Difficulty: Easy

Company: Facebook

**Explanation:**
The approach to the solution should be:

Step 1: Filter the dataset from 2022-06-15 to 30 days before comments
Step 2: Calculate the sum of the number of comments
Step 3: Aggregate everything at the user level (group by user id)

Now to filter the dataset from 2022-06-15 to 30 days before, we can do it statically by manually mentioning the date that would be 30 days before. But it is not the recommended method. Therefore, we should move forward with the dynamic method and use the clause INTERVAL for the same.

Also, we need to cast the date string as the DateTime in the query, for that, we either need to use “::date” or cast( ) function directly.

**Static solution:**

```
Select User_id, SUM(Number_of_comments)
From Fb_comments_count
Where Creater_at BETWEEN “2022-05-15” : : date AND “2022-06-15” : : date
GROUP BY User_id;
```

**Dynamic Solution:**

```
Select User_id, SUM(Number_of_comments)
From Fb_comments_count
Where Creater_at BETWEEN ("2022-06-15" : : date - 30 * INTERVAL '1 day')  AND
"2022-06-15" : : date
GROUP BY User_id;
```

---

2) Which countries have risen in the rankings based on the number of comments between Dec 2021 vs Jan 2022?

Hint: Avoid gaps between ranks when ranking countries

```
Fb_comments_count(
User_id int,
Created_at datetime,
Number_of_comments int);

Fb_active_users(
User_id int,
Name varchar,
Status varchar,
Country varchar);
```

Difficulty: Hard

Company: Facebook

**Explanation:**
The approach to the solution should be:
**Step 1:** Join the two tables on user_id (left join because not all users may have made comments)
**Step 2:** Filter our table for Dec 2021 and Jan 2022
**Step 3:** Exclude rows where the country is empty
**Step 4:** Sum the number of comments per country
**Step 5:** Rank 2021 comments counts and 2022 comment counts
**Step 6:** Apply final filter to fetch only countries with ranking decline(Jan rank > dec rank)

```
With
dec_summary as
```

```
(
        Select Country,
        SUM(Number_of_comments) as number_of_comments_dec,
        dense_rank() over(order by sum(Number_of_comments) DESC) as country_rank
        FROM Fb_active_users as a LEFT JOIN
        Fb_comments_count as b
        On a.User_id = b.User_id;
        Where Created_at <= “2021-12-31” and Created_at > = “2021-12-01”
        AND Country IS NOT NULL
        GROUP BY Country
),
Jan_summary as(
        Select Country,
        SUM(Number_of_comments) as number_of_comments_jan,
        dense_rank() over(order by sum(Number_of_comments) DESC) as country_rank
        FROM Fb_active_users as a LEFT JOIN
        Fb_comments_count as b
        On a.User_id = b.User_id;
        Where Created_at <= “2022-01-31” and Created_at > = “2022-01-01”
        AND Country IS NOT NULL
        GROUP BY Country
);

Select j.country
From jan_summary j
LEFT JOIN dec_summary d
on j.country = d.country
WHERE(j.country_rank<d.country_rank) OR d.country is NULL;
```

---

## 3) Given the table R, compute the correlation coefficient of X1 and X2 columns.

## Definition of correlation:

$$Cor(X,Y) = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 (y_i - \bar{y})^2}}$$

| X1 | X2 |
|---|---|
| 1 | 34 |
| 2 | 34 |
| 3 | 4 |
| 10 | 5 |
| … | … |

Company: Google
Difficulty: Medium

**Explanation**
**Step1:** Find the numerator with help of the formula using windows function
**Step2:** Similarly find the denominator with windows function.Hint: You can reuse the numerator here.

```
with
num_tbl_1 as (select *
                        , avg(x1) over () as x1_mean
                        , avg(x2) over () as x2_mean
                from correlation_db.correlation)
, num_tbl_2 as (select
                        (x1 - x1_mean) as x1_diff,
                        (x2 - x2_mean) as x2_diff
                from num_tbl_1)
, numerator as (select
                        sum(x1_diff * x2_diff) as num
                from num_tbl_2)
, den_tbl_1 as (select
                        pow(x1_diff, 2) as x1_sqr,
                        pow(x2_diff, 2) as x2_sqr
                from num_tbl_2)
, denominator as (select
                        sqrt(sum(x1_sqr * x2_sqr)) as denom
                from den_tbl_1)


select
(select num from numerator) / (select denom from denominator) as correlation_co_eff
```

---

4) Find the number of emails received by each user under each built-in email label. The email labels are:

1. Promotion
2. Social
3. Shopping

Output the user along with the number of promotion, social, and shopping mails count.

For the below given relations:

google_gmail_emails(
id int
from_user varchar
to_user varchar
day int
)

google_gmail_labels(
email_id int
label varchar
)

Difficulty: Medium
Company: Google

**Explanation:**
**Step 1**: Group the column of user and label before counting them individually.Here we have used count in place of sum in order to ignore the null values, if we use sum we may get a blank output for any grouped values that include a null
**Step 2**: We put together the individual counts for each of the different labels.
**Step 3**: Finally, we write in our initial joined table.

```
SELECT
        to_user,
        SUM(CASE WHEN label = 'Promotion' THEN cnt
                ELSE 0 END) AS promotion_count,
        SUM(CASE WHEN label = 'Social' THEN cnt
                ELSE 0 END) AS social_count,
        SUM(CASE WHEN label = 'Shopping' THEN cnt
                ELSE 0 END) AS shopping_count
FROM (SELECT mails.to_user,
                Labels.label,
```

```
        COUNT(*) AS cnt
    FROM google_gmail_emails as mails
        INNER JOIN google_gmail_labels as labels
            ON mails.id = labels.email_id
    GROUP BY mails.to_user, Labels.label
    )
GROUP BY to_user
ORDER BY to_user;
```

---

5) Find the total costs of each customer's orders. Output the customer's id, first name, and the total order cost. Order records by customer's first name are alphabetical.

The relation schema given is

```
customers(
    id int,
    first_name varchar,
    last_name varchar,
    city varchar,
    address varchar,
    phone_number varchar
)

orders(
    id int,
    cust_id int,
    order_date datetime,
    order_details varchar,
    total_order_cost int
)
```

Difficulty: Easy
Company: Amazon

**Explanation**:
Join given tables of customers and orders where we can group the customer id and name and then sum the total cost of the orders that they have.
**Step1**: Join the relations
**Step2**: Aggregate each customer's order's total costs and then group the relevant column.
**Step3**: Since questions ask us to order the customer's first name alphabetically, that required a

simple order at the end of the above query.

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

---

6) You have a table of in-app purchases by user. Users that make their first in-app purchase are placed in a marketing campaign where they see call-to-actions for more in-app purchases. Find the number of users that made additional in-app purchases due to the marketing campaign's success.

The marketing campaign doesn't start until one day after the initial in-app purchase so users that only made one or multiple purchases on the first day do not count, nor do we count users that over time purchase only the products they purchased on the first day.

The relation given below is:

```
marketing_campaign(
user_id int,
created_at datetime,
product_id int,
quantity int,
price int
);
```

Difficulty: Hard
Company: Amazon

**Explanation**:
To be considered in the marketing campaign, the user needs to buy a product that is not the same product as what was bought in their first purchase date. That is a product needs to be different + it needs to be purchased on a different date.

**Scenarios to be considered:**

- 1 item, 1 date of purchase (not eligible for a marketing campaign)
- Multiple products, 1 date of purchase (not eligible for a marketing campaign)
- 1 product, multiple days (not eligible for a marketing campaign)
- Multiple products, multiple days, but same products as the 1st day of purchase (not

eligible for a marketing campaign)
- Multiple dates, multiple products (should be in a marketing campaign)

**Step1**: Implement product needs to be different + it needs to be purchased in a different date.(this has handled the scenario 1, 2, and 3)
**Step2**: Identify the the user first purchase and date
**Step3**: Combine logic of step 1 and step 2

```
Select count(distinct user_id)
From  marketing_campaign
Where user_id in(
                Select user_id
                From marketing_campaign
                Group by user_id
                Having count(distinct product_id) > 1
                And count(distinct created_at) > 1
                )
                And concat((user_id), '_', (product_id)) not in
                (
                Select user_product
                        From(
                                Select *
                                  rank() over (PARTITION by user_id Order by created_at) as rn,
                                  concat((user_id), '_', (product_id)) as user_product
                                From marketing_campaign
                        ) X
                        Where rn = 1
                        );
```

---

## 7) Given a users table, write a query to get the cumulative number of new users added by day, with the total reset every month.

Company: Amazon, LinkedIn
Data: Easy

users table

| Columns | Type |
|---|---|
| Id | Integer |
| name | varchar |
| Created_at | datetime |

**Output:**

| Date | **Monthly Cumulative** |
|---|---|
| 2020-01-01 | 5 |
| 2020-01-02 | 12 |
| ... | … |
| 2020-02-01 | 8 |
| 2020-02-02 | 17 |
| 2020-02-03 | 23 |

**Solution:**
**Step1**: Convert the date to a proper data format which might be present as string
**Step2**: Extract the month from date, since the count has to be reset every month.
**Step3**: to get the cumulative count use count() with partition on the months.

Note: By default the window frame will be **range** so users who joined on the same day will get the same count. Modify it with **rows between unbounded preceding and current row**

```
with
tbl1 as
(select id
        , name
        , str_to_date(created_at, "%m/%d/%Y") as created_at
        , month(str_to_date(created_at, "%m/%d/%Y")) as created_month from google.g_users
From user_tbl
)

, final_tbl as
(select id
        , name
        , created_at
        , created_month
        , count(id) over (partition by created_month order by created_at
                rows between unbounded preceding and current row) as cumulative_count
        from tbl1
)

select * from final_tbl
```

---

## 8) Get a histogram of users to count based on the number of tweets they have sent.

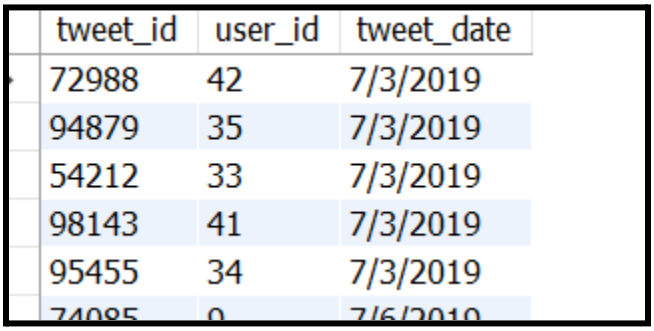
The histogram bins should be

- less_than_11
- Between_11_and_15

- Between_16_and_20
- Between_21_and_25
- Between_25_and_30
- Greater_than_30

Company: Twitter

Difficulty: Hard



| tweet_id | user_id | tweet_date |
|---|---|---|
| 72988 | 42 | 7/3/2019 |
| 94879 | 35 | 7/3/2019 |
| 54212 | 33 | 7/3/2019 |
| 98143 | 41 | 7/3/2019 |
| 95455 | 34 | 7/3/2019 |
| 74085 | 0 | 7/6/2019 |

**Explanation**
**Step1**: Get the count of tweets sent by each user.
**Step2**: Bin the users into different buckets based on the tweet count using case statement
**Step3**: Find the number of users in each bucket by taking a group by on the buckets and count the users in it.

```
with
tweet_cnt as (
            select user_id,
           count(tweet_id) tweet_count
          from user_tweet
          group by user_id
)

, tweet_cnt_s as (
          select *
          from tweet_cnt
          order by tweet_count desc
)

, tweet_buck as (
          select *,
              case when tweet_count <= 10 then 'less_than_11'
                        when tweet_count between 11 and 15 then 'between_11_and_15'
                        when tweet_count between 16 and 20 then 'between_16_and_20'
                        when tweet_count between 21 and 25 then 'between_21_and_25'
                        when tweet_count between 26 and 30 then 'between_25_and_30'
               else 'greater_than_30' end as tweet_bucket
                    from tweet_cnt_s
)
```

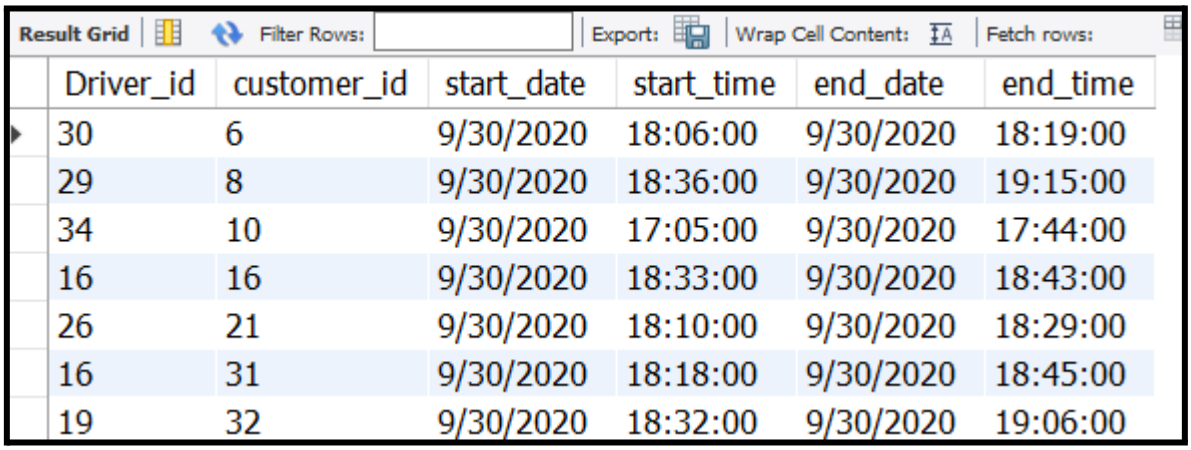
```
select tweet_bucket,
        count(user_id) as user_count
from tweet_buck
group by tweet_bucket
order by user_count
```

---

## 9) Consider you are given a table of ride info from Uber. How frequently do customers book a ride on average?

Difficulty: Medium

Company:Uber



Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

| Driver_id | customer_id | start_date | start_time | end_date | end_time |
|---|---|---|---|---|---|
| 30 | 6 | 9/30/2020 | 18:06:00 | 9/30/2020 | 18:19:00 |
| 29 | 8 | 9/30/2020 | 18:36:00 | 9/30/2020 | 19:15:00 |
| 34 | 10 | 9/30/2020 | 17:05:00 | 9/30/2020 | 17:44:00 |
| 16 | 16 | 9/30/2020 | 18:33:00 | 9/30/2020 | 18:43:00 |
| 26 | 21 | 9/30/2020 | 18:10:00 | 9/30/2020 | 18:29:00 |
| 16 | 31 | 9/30/2020 | 18:18:00 | 9/30/2020 | 18:45:00 |
| 19 | 32 | 9/30/2020 | 18:32:00 | 9/30/2020 | 19:06:00 |

**Solution**

**Step1**: Convert the date which may be in the string format to the correct date format
**Step2:** For each customer, create a new column as ‘prev_booked_ride’ to find the last time they took a ride using the lag function.
**Step3**: Find the days difference between the current and previous ride and call it as ‘days_since_last_ride’
**Step4**: Each rows in column ‘days_since_last_ride’ will contain how long it took for each user to book a ride from their previous ride.
**Step5**: Taking an average on ‘‘days_since_last_ride’’ grouped by each user will be the final solution.

```
With
table_1 as (
        Select
        Customer_id,
        str_to_date(ride_start_date, "%m/%d/%Y") as current_booked_ride
```

```
 from uber.uber_rides
)

, data_with_prev_ride as (
        Select Customer_id
            , current_booked_ride
            , lag(current_booked_ride) over (partition by customer_id order by
        current_booked_ride) as prev_booked_ride
        from table_1
)

, days_between_ride as (
        select
            Customer_id
            , current_booked_ride
            , prev_booked_ride
            , datediff(current_booked_ride, prev_booked_ride) as
        days_since_last_ride
        from data_with_prev_ride
)

SELECT
    customer_id, AVG(days_since_last_ride) avg_days
FROM
    days_between_ride
GROUP BY customer_id
order by avg_days
```

---

10) Find the users who have watched the first ad of a Youtube video at least 80% of it overall duration
You are provided with 4 tables:

- Event table
- Video table
- Ads table
- User table

Difficulty: Hard

Company: Youtube

| user_id | session_id | event_type_id | date | start_time | end_time |
|---|---|---|---|---|---|
| U1 | S1 | A1 | 11/23/2021 | 7:32:10 | 7:32:20 |
| U1 | S1 | V1 | 11/23/2021 | 7:32:20 | 7:42:58 |
| U3 | S1 | A2 | 11/23/2021 | 9:36:00 | 9:36:55 |
| U3 | S1 | V3 | 11/23/2021 | 9:36:55 | 9:53:55 |
| U3 | S1 | A3 | 11/23/2021 | 9:53:55 | 9:54:03 |
| U3 | S1 | V3 | 11/23/2021 | 9:54:03 | 9:57:03 |
| U3 | S2 | V2 | 11/23/2021 | 10:03:00 | 10:13:38 |

| video_id | duration |
|---|---|
| V1 | 0:10:38 |
| V2 | 0:15:19 |
| V3 | 0:42:00 |

| Ad_id | Duration |
|---|---|
| A1 | 0:00:30 |
| A2 | 0:01:00 |
| A3 | 0:00:45 |

| user_id | name | country |
|---|---|---|
| U1 | John | USA |
| U2 | Alan | Aus |
| U3 | Anna | UK |
| U4 | Jean | Germany |

**Added information:**

- Visiting a website(here Youtube) until closing the tab/window is considered as a session
- Within a session there are many actions that the user will take like clicking on a video, posting comments, like, dislike etc which are considered events within an active session.
- While watching a video, Ads appearing is considered an event, then resuming the video is another event, if there is a second ad and the video resuming are all considered subsequents events.

**Solution:**

**Step1:** Since we are not worried about video information we don't have to use the video table

**Step2**: Inner Join the Ads table with the event table. This will remove all event corresponding to videos
**Step3:** Create a rank column within each user and each session sorted based on the ads starting time. This will help us which ad was displayed first, second, third etc..
**Step4**: As we are interested only on the first ad, filter the rows corresponding to rank=1
**Step5**: Create a column to find the difference between start-time and the end-time in seconds of the event which is the duration the user watched the ad i.e. they could have skipped it without watching them completely
**Step6**: Similarly get the duration of the actual ad in seconds from the ads table.
**Step7**: Now that we have ads watched time and the actual ads duration in seconds, dividing them will give the % watch duration.
**Step8**: Final step we can filter all users who have watched more than 80%

```
with  session_with_ad as
(
select s.user_id
 , s.session_id
          , s.event_type_id
          , s.start_time
          , s.end_time
          , a.Duration as ad_duration
 from session s inner join advertisement a
 on s.event_type_id = a.ad_id
)

, ad_rank as
(
select *,
row_number() over(partition by user_id, session_id
order by start_time) as rank_
from session_with_ad
)

, first_ad as
(
Select *
from ad_rank
where rank_ =  1
)

, ad_watch_duration as
(
select *,
timediff(end_time, start_time) as ad_watch_duration
from first_ad
)

, ad_watch_duration_in_sec as
(
select *
, minute(ad_watch_duration)*60 + second(ad_watch_duration) as ad_watch_duration_in_sec
, minute(ad_duration)*60 + second(ad_duration) as ad_duration_in_sec
from ad_watch_duration
)
```

```
, ad_watch_percent as (
select user_id
        , ad_watch_duration_in_sec
                , ad_duration_in_sec
        , (ad_watch_duration_in_sec/ad_duration_in_sec)*100 as watch_percentage
        from ad_watch_duration_in_sec
)

select * from ad_watch_percent adwp
inner join user_info u
on adwp.user_id = u.user_id
where adwp.watch_percentage >= 80
```

---

## 11) Find the total number of concurrent users who were online when a particular user is online

Company: Pinterest

Difficulty: Medium

| user_id | Session_id | date | start_time | end_time |
|---|---|---|---|---|
| 42 | S_60 | 2022-10-15 | 3:06:46 | 4:00:00 |
| 35 | S_77 | 2022-10-15 | 3:53:00 | 4:15:00 |
| 33 | S_31 | 2022-10-15 | 3:20:00 | 5:00:00 |
| 41 | S_49 | 2022-10-15 | 6:00:01 | 7:00:00 |
| 34 | S_30 | 2022-10-15 | 2:55:00 | 3:10:00 |

**Added information**: Assume this scenario

- User 1 is online from 8:00 AM to 11:00 AM.
- User 2 is online from 7:00 AM to 9:00 AM.
- User 3 is online from 10:00 AM to 11:30 AM.
- User 4 is online from 11.15 AM to 11:45 AM.

When the start time or end time of one user is within the online duration of the other user then they are said to be online at the same time

Here when User1 is online, there are 2 other users who are online as well (User 2 and User 3) whereas User4 is not offline. So the count for user 1 is 2.

Similarly, when User 4 is online only User 3 is online so the count of user 4 is 1.

**Solution:**

**Step1**: Convert the date to the proper format

**Step2**: Do a self join with inequality on user id, this will compare one user with every other user but themselves.i.e. It will create rows with one user Vs the rest…

**Step3**: Filter all rows where start-time or end-time of user1 is within the start-time,end-time of user_2
**Step4**: Find the count of total users online by grouping on each user.

```
with tbl_1 as
(
select user_id
            , session_id
            , date
            , str_to_date(start_time, "%H:%i:%s" ) start_time
              , str_to_date(end_time, "%H:%i:%s" ) end_time
        from fb_online_users
)

, tbl_2 as
(select t1.user_id as user_1
        , t2.user_id user_2
              , t1.start_time as u1_start_time
              , t1.end_time as u1_end_time
              , t2.start_time as u2_start_time
              , t2.end_time as u2_end_time
        from tbl_1 t1 join tbl_1 t2
        on t1.user_id <> t2.user_id
)

, final_tbl as (
        select * from tbl_2
                        where (u1_start_time between u2_start_time and u2_end_time)
                          or (u1_end_time between u2_start_time and u2_end_time)
        order by user_1
)

select user_1
            , count(distinct user_2) as online_users_count
                    , group_concat(user_2)
from final_tbl
group by user_1
order by online_users_count desc
```

---

12) Assume you are given the table below containing measurement values obtained from a sensor over several days. Measurements are taken several times within a given day. Write a query to obtain the sum of the odd-numbered and even-numbered measurements on a particular day, in two different columns.

Note that the 1st, 3rd, and 5th measurements within a day are considered odd-numbered measurements and the 2nd, 4th, 6th measurements are even-numbered measurements.

Company: FAANG

Difficulty: Medium

`measurements` **Table:**

| Column Name | Type |
|---|---|
| measurement_id | integer |
| measurement_value | decimal |
| measurement_time | datetime |

**Example Output:**

| measurement_day | odd_sum | even_sum |
|---|---|---|
| 07/10/2022 00:00:00 | 2355.75 | 1662.74 |
| 07/11/2022 00:00:00 | 1124.50 | 1234.14 |

**Explanation:**
On 07/11/2022, there are only two measurements. In chronological order, the first measurement (odd-numbered) is 1124.50, and the second measurement(even-numbered) is 1234.14.

**Solution:**

```
WITH ranked_measurements AS (
  SELECT
    CAST(measurement_time AS DATE) AS measurement_day,
    measurement_value,
    ROW_NUMBER() OVER (
      PARTITION BY CAST(measurement_time AS DATE)
      ORDER BY measurement_time) AS measurement_num
  FROM measurements
)

SELECT
  measurement_day,
  SUM(
    CASE WHEN measurement_num % 2 != 0 THEN measurement_value
      ELSE 0 END) AS odd_sum,
  SUM(
  CASE WHEN measurement_num % 2 = 0 THEN measurement_value
    ELSE 0 END) AS even_sum
FROM ranked_measurements
GROUP BY measurement_day;
```

---

13) Assume you are given the tables below containing information on Snapchat users, their ages, and their time spent sending and opening snaps.

Write a query to obtain a breakdown of the time spent sending vs. opening snaps (as a percentage of total time spent on these activities) for each age group.

Output the age bucket and percentage of sending and opening snaps. Round the percentage to 2 decimal places.
Notes:

- You should calculate these percentages:
  - time sending / (time sending + time opening)ˆ
  - time opening / (time sending + time opening)
- To avoid integer division in percentages, multiply by 100.0 and not 100.

Company:

Difficulty: Medium

`activities` **Table:**

| Column Name | Type |
|---|---|
| activity_id | integer |
| user_id | integer |
| activity_type | string ('send', 'open', 'chat') |
| time_spent | float |
| activity_date | datetime |

`activities` **Example Input:**

| activity_id | user_id | activity_type | time_spent | activity_date |
|---|---|---|---|---|
| 7274 | 123 | open | 4.50 | 06/22/2022 12:00:00 |
| 2425 | 123 | send | 3.50 | 06/22/2022 12:00:00 |
| 1413 | 456 | send | 5.67 | 06/23/2022 12:00:00 |
| 1414 | 789 | chat | 11.00 | 06/25/2022 12:00:00 |
| 2536 | 456 | open | 3.00 | 06/25/2022 12:00:00 |

`age_breakdown` **Table:**

| Column Name | Type |
|---|---|
| user_id | integer |
| age_bucket | string ('21-25', '26-30', '31-25') |

`age_breakdown` **Example Input:**

| user_id | age_bucket |
|---|---|
| 123 | 31-35 |
| 456 | 26-30 |
| 789 | 21-25 |

**Example Output:**

| age_bucket | send_perc | open_perc |
|---|---|---|
| 26-30 | 65.40 | 34.60 |
| 31-35 | 43.75 | 56.25 |

**Explanation**

For the age bucket 26-30, the time spent sending snaps was 5.67 and opening 3. The percent of time sending snaps was 5.67/(5.67+3)=65.4%, and the percent of time opening snaps was 3/(5.67+3)=34.6%.

**Solution:**

```
WITH snaps_statistics AS (
        SELECT
                age.age_bucket,
                SUM(CASE WHEN activities.activity_type = 'send'
                  THEN activities.time_spent ELSE 0 END) AS send_timespent,
                SUM(CASE WHEN activities.activity_type = 'open'
                  THEN activities.time_spent ELSE 0 END) AS open_timespent,
                SUM(activities.time_spent) AS total_timespent
        FROM activities
        INNER JOIN age_breakdown AS age
                ON activities.user_id = age.user_id
        WHERE activities.activity_type IN ('send', 'open')
        GROUP BY age.age_bucket
)

SELECT
  age_bucket,
  ROUND(100.0 * send_timespent / total_timespent, 2) AS send_perc,
  ROUND(100.0 * open_timespent / total_timespent, 2) AS open_perc
FROM snaps_statistics;
```

---

14) Given a Transaction table and Users table, we want to find out the current balance of all users and check whether they have breached their credit limit (If their current credit is less than 0).

Write an SQL query to report.

- user_id,
- user_name,
- credit - current balance after performing transactions, and
- credit_limit_breached - check credit_limit ("Yes" or "No")

Difficulty: Medium

Company:

**Input:**
Users table:

| user_id | user_name | credit |
|---|---|---|
| 1 | Moustafa | 100 |
| 2 | Jonathan | 200 |
| 3 | Winston | 10000 |
| 4 | Luis | 800 |

Transactions table:

| trans_id | paid_by | paid_to | amount | transacted_on |
|---|---|---|---|---|
| 1 | 1 | 3 | 400 | 2020-08-01 |
| 2 | 3 | 2 | 500 | 2020-08-02 |
| 3 | 2 | 1 | 200 | 2020-08-03 |

**Output:**

| user_id | user_name | credit | credit_limit_breached |
|---|---|---|---|
| 1 | Moustafa | -100 | Yes |
| 2 | Jonathan | 500 | No |
| 3 | Winston | 9900 | No |
| 4 | Luis | 800 | No |

**Explanation:**
Moustafa paid $400 on "2020-08-01" and received $200 on "2020-08-03", credit (100 -400 +200) = -$100
Jonathan received $500 on "2020-08-02" and paid $200 on "2020-08-08", credit (200 +500 -200) = $500
Winston received $400 on "2020-08-01" and paid $500 on "2020-08-03", credit (10000 +400 -500) = $9990
Luis did not received any transfer, credit = $800

**Explanation**:
A user can be either a creditor or a debitor in a transaction.
**Step1:** Find the overall transaction of each user as a debitor(i.e.paid_by). Since the money is debited change the overall transaction sum to negative
**Step2:** Similarly find the overall transaction of being a creditor(i.e. paid_to)
**Step3:** Combine both the results into a single table so the credit and debit amount are in a single column for each user(i.e. Union all)
**Step4:** Group by on each user and find the overall transaction left out amount(could be positive or negative).
**Step5**: We have to combine the leftout amount with the initial credit amount of each user i.e. join each user's credit amount from the users table.
**Step6**: Since we want information about all the users, do a right join with the users table being the right table or you can also do a left join with the users table being the left table.
**Step7**: Once the table is joined, compare the user's initial credit amount with the transaction leftover amount to find the final amount if the credit limit is breached or not. Note: There could be some users who may not have done any transaction, for them the final credit should be their initial credit amount(i.e. Use coalesce)

```
with debit as
```

```
        (
          select paid_by as user_id,
                 -sum(amount) as amount
          from transactions group by paid_by
        ),

credit as
        (
          select paid_to as user_id,
                 sum(amount) as amount
          from transactions group by paid_to
        ),

all_transaction as
        (
          select * from debit
             union all
          select * from credit
        ),

amount_with_users_info as
        (
          select u.user_id,
                 u.user_name,
                 coalesce(u.credit + sum(amount), u.credit) as final_amt
          from all_transaction t
          right join users u on t.user_id = u.user_id
          group by user_id, user_name, credit
        ),


final as
        (
          select user_id,
                  user_name,
                  final_amt as credit,
                  if (final_amt>0, 'No', 'Yes') as credit_limit_breached
          from amount_with_users_info
        )

select * from final
```

---

15) A telecommunications company wants to invest in new countries. The company intends to invest in the countries where the average call duration of the calls in this country is strictly greater than the global average call duration.

Write an SQL query to find the countries where this company can invest.
Return the result table in any order.
The query result format is in the following example.

Difficulty: Medium

Company:

Table `Person`:

```
+----------------+---------+
| Column Name    | Type    |
+----------------+---------+
| id             | int     |
| name           | varchar |
| phone_number   | varchar |
+----------------+---------+
id is the primary key for this table.
Each row of this table contains the name of a person and their phone number.
Phone number will be in the form 'xxx-yyyyyyy' where xxx is the country code (3 characters)
and yyyyyyy is the phone number (7 characters) where x and y are digits. Both can contain
leading zeros.
```

Table `Country`:

```
+----------------+---------+
| Column Name    | Type    |
+----------------+---------+
| name           | varchar |
| country_code   | varchar |
+----------------+---------+
country_code is the primary key for this table.
Each row of this table contains the country name and its code. country_code will be in the
form 'xxx' where x is digits.
```

Table `Calls`:

```
+-------------+------+
| Column Name | Type |
+-------------+------+
| caller_id   | int  |
| callee_id   | int  |
| duration    | int  |
+-------------+------+
There is no primary key for this table, it may contain duplicates.
Each row of this table contains the caller id, callee id and the duration of the call in
minutes. caller_id != callee_id
```

**Explanation**

If a call duration between user1 from country A to user2 in country B is 10 mins. Then the same duration of 10 mins is assigned to both the countries. If User1 and User2 are from the same

country A, then even in this case the duration is assigned to both countries i.e. duration is twice for country A

**Step1**: As a first step, find the global average from the Calls table. As explained above one call duration has to be attributed to both the countries (i.e. one call duration * 2). Since we are interested in only the avg. (i.e. (total call duration * 2)/(total calls * 2)) the * 2 gets canceled out so we can just take the normal avg using group by
**Step2**: Next, the country table has to be joined with the person table. As mentioned in the table info the first 3 characters of the phone number is country code. Use the `left` string operation to pull the country code from the phone number in the Person table and use this substring as the key to join with country code in the country table.
**Step3**: Now that we have mapped the country to each of the users, we have to get the call duration information.
**Step4:** As mentioned before a user can be a caller or a callee. So when joining with the Calls table it has to be joined on both `caller_id` and `callee_id` columns.
**Step5**: To join one column with 2 or more columns simultaneously we can use the `IN` clause.
**Step6**: Because of joining using the `IN` clause against 2 columns(caller_id, callee_id), each call will be repeated twice, one for each user from each country.
**Step7**: Now we can just groupby on the Country to get the average call duration for each country.
**Step8**: Filter all the countries that are greater than the global avg calculated from Step1.

```
SELECT Country.name AS country
        FROM Person
JOIN Country ON LEFT(phone_number, 3) = country_code
JOIN Calls ON Person.id IN (Calls.caller_id, Calls.callee_id)
GROUP BY Country.name
HAVING AVG(duration) > (SELECT AVG(duration) FROM calls)
```

---