1) **[Easy]** Given we have a relation
   Fb_comments_count(
   User_id int,
   Created_at datetime,
   Number_of_comments int);

   Return the total number of comments received for each user in the last 30 days. Assume that today is `2022-06-15`.

   Solution:
   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;

   Source:
   https://www.stratascratch.com/blog/data-science-sql-interview-questions-from-faang-companies/

---

2) **[Hard]**Given two relations
   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);

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

Solution:
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)

```
=> Select *
FROM Fb_active_users as a LEFT JOIN
Fb_comments_count as b
On a.User_id = b.User_id;
```

Step 2: Filter our table for Dec 2021 and Jan 2022

```
With dec_summary as(
Select *
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”
),
Jan_summary as(
Select *
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”
);
```

Step 3: Exclude rows where the country is empty

```
With dec_summary as(
Select *
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
),
Jan_summary as(
Select *
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
);
```

Step 4: Sum the number of comments per country

```
With dec_summary as(
Select
        Country,
        SUM(Number_of_comments) as number_of_comments_dec
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
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
);
```

To check the output at this stage:

```
Select *
From jan_summary j
LEFT JOIN dec_summary d on j.country = d.country;
```

Step 5: Rank 2021 comments counts and 2022 comment counts

There are a number of ranking functions but since it is given in the problem that we can avoid gaps in between the ranks, therefore, we can go with the Dense rank function.

```
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 *
From jan_summary j
LEFT JOIN dec_summary d on j.country = d.country;
```

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;
```

Source:
https://www.stratascratch.com/blog/data-science-sql-interview-questions-from-faang-companies/

---

3) **[Medium] Given the table R, compute the correlation coefficient of X1 and X2 columns.**

X1	X2
1	34
2	34
3	4
10	5
…	…

Solution:

Given two variables, finding at what extent X1 and X2 are related/correlated to each other is correlation in simple language.

The formula is: Cov(X1, X2)/[Std(X1)*Std(X2)]

Here,

1. **Cov is Covariance = Avg(X1- $X1_{mu}$) * (X2 - $X2_{mu}$)**
2. **Var is Variance = Avg((X1 - $X1_{mu}$)**2)**
3. **Std is Standard Deviation = Sqrt(Avg(X1 - $X1_{mu}$)**2))**

Step 1: Calculate the mean

```
With mean as(
Select X1, X2,
Avg(X1) OVER() as mean_X1,
Avg(X2) OVER() as mean_X2,
FROM R);
```

Step 2: Calculate the variance

```
With mean as(
Select X1, X2,
Avg(X1) OVER() as mean_X1,
Avg(X2) OVER() as mean_X2,
FROM R
),
Variance as(
Select
Avg(POWER(X1 - mean_X1, 2)) as var_X1,
Avg(POWER(X2 - mean_X2, 2)) as var_X2
From mean
);
```

Step 3: Calculate the standard deviation

```
With mean as(
Select X1, X2,
Avg(X1) OVER() as mean_X1,
Avg(X2) OVER() as mean_X2,
FROM R
),
Variance as(
Select
Avg(POWER(X1 - mean_X1, 2) as var_X1,
Avg(POWER(X2 - mean_X2, 2) as var_X2
From mean
),
StdDev as(
Select
POWER(var_X1, 0.5) as std_X1,
POWER(var_X2, 0.5) as std_X2
From Variance
);
```

Step 4: Calculate the covariance

```
With mean as(
Select X1, X2,
Avg(X1) OVER() as mean_X1,
Avg(X2) OVER() as mean_X2,
```

```
FROM R
),
Variance as(
Select
Avg(POWER(X1 - mean_X1, 2) as var_X1,
Avg(POWER(X2 - mean_X2, 2) as var_X2
From mean
),
StdDev as(
Select
POWER(var_X1, 0.5) as std_X1,
POWER(var_X2, 0.5) as std_X2
From Variance
),
Covariance as(
Select
AVG((X1 - mean_X1)*(X2 - mean_X2)) as cov_X1_X2
From mean
);
```

Step 4: Calculate the correlation coefficient

```
With mean as(
Select X1, X2,
Avg(X1) OVER() as mean_X1,
Avg(X2) OVER() as mean_X2,
FROM R
),
Variance as(
Select
Avg(POWER(X1 - mean_X1, 2) as var_X1,
Avg(POWER(X2 - mean_X2, 2) as var_X2
From mean
),
StdDev as(
Select
POWER(var_X1, 0.5) as std_X1,
POWER(var_X2, 0.5) as std_X2
From Variance
),
Covariance as(
Select
AVG((X1 - mean_X1)*(X2 - mean_X2)) as cov_X1_X2
From mean
)
```

```
Select
cov_X1_X2 / (std_X1 * std_X2) as corr_X1_X2
From Covariance, StdDev;
```

Source: https://www.youtube.com/watch?v=yliRaLGzBfI

---

4) **[Hard]**We want to generate an inventory age report which would show the distribution of remaining inventory across the length of time the inventory has been sitting at the warehouse. We are trying to classify the inventory on hand across the below 4 buckets to denote the time the inventory has been lying in the warehouse.

0-90 days old
91-180 days old
181-270 days old
271 – 365 days old

For example, the warehouse received 100 units yesterday and shipped 30 units today, then there are 70 units that are a day old. Note that the warehouses use FIFO (first in first out) approach to manage inventory, i.e., the inventory that comes first will be sent out first.

ID	OnHandQuantity	OnHandQuantityDelta	event_type	event_datetime
TR0013	278	99	OutBound	25/05/2020 00:25
TR0012	377	31	InBound	24/05/2020 22:00
TR0011	346	1	OutBound	24/05/2020 15:01
TR0010	346	1	OutBound	23/05/2020 05:00
TR009	348	102	InBound	25/04/2020 18:00
TR008	246	43	InBound	25/04/2020 02:00
TR007	203	2	OutBound	25/02/2020 09:00
TR006	205	129	OutBound	18/02/2020 07:00
TR005	334	1	OutBound	18/02/2020 08:00
TR004	335	27	OutBound	29/01/2020 05:00
TR003	362	120	InBound	31/12/2019 02:00
TR002	242	8	OutBound	22/05/2019 00:50
TR001	250	250	InBound	20/05/2019 00:45

The relation ”Warehouse” looks as follows:
ID: ID of the log entry (Varchar(10))
OnHandQuantity: Quantity in the warehouse after an event (INT)
OnHandQuantityDelta: Change in on-hand quantity due to an event (INT)
event_type: Inbound – inventory being brought into the warehouse; Outbound – inventory being sent out of the warehouse (Varchar(10))
event_datetime: date-time of the event (timestamp)
The data is sorted with the latest entry at the top.

Example: On 20th May 2019, 250 units were inbounded into the FC. On 22nd May 2019, 8 units were shipped out (outbound) from the FC, reducing inventory on hand to 242 units. On 31st December, 120 units were further inbounded into the FC increasing the inventory on hand from 242 to 362.On 29th January 2020, 27 units were shipped out reducing the inventory on hand to 335 units.

On 29th January, of the 335 units on hands, 120 units were 0-90 days old (29 days old) and 215 units were 181-270 days old (254 days old).

Sample output:

0-90 days old	91-180 days old	181-270 days old	271-365 days old
176	102	0	0

Solution:
Step 1: Sort the data based on date time (as data needs to be sorted with latest event at the top)

```
Select *
From Warehouse
Order by event_datetime desc;
```

Step 2: Identify the day 1 and date time for the further calculation. And since we need several buckets based on number of days, therefore, we will use "With".

```
With Wh as(
 Select *
From Warehouse
Order by event_datetime desc
),
Days as(
Select onhandquantity, event_datetime
From Wh limit 1)
Select * from days;
```

Step 3: Segregate the data for different buckets

```
With Wh as(
 Select *
From Warehouse
Order by event_datetime desc
),
Days as(
Select onhandquantity, event_datetime,
(event_datetime - INTERVAL '90 day') as day 90,
(event_datetime - INTERVAL '180 day') as day 180,
(event_datetime - INTERVAL '270 day') as day 270,
(event_datetime - INTERVAL '365 day') as day 365,
From Wh limit 1)
Select * from days;
```

Step 4: Count total number of shipments in last 90 days

```
With Wh as(
 Select *
From Warehouse
Order by event_datetime desc
),
Days as(
Select onhandquantity, event_datetime,
(event_datetime - INTERVAL '90 day') as day90,
(event_datetime - INTERVAL '180 day') as day180,
(event_datetime - INTERVAL '270 day') as day270,
(event_datetime - INTERVAL '365 day') as day365,
From Wh limit 1
),
Inv_90_days as(
Select sum(onhandquantitydelta) as daysold_90
From Wh cross join days as d
Where event_type = 'Inbound'
And Wh.event_datetime >= d.day90
),
Inv_90_days_final as(
Select case when daysold_90 > d.onhandquantity then d.onshandquantity
        Else daysold_90
        End daysold_90
From Inv_90_days cross join days as d)

Select daysold_90 as '0-90 days old'  from Inv_90_days_final;
```

Step 5: Now we need to do the same for 91-180 days buckets as well

```
With Wh as(
 Select *
From Warehouse
Order by event_datetime desc
),
Days as(
Select onhandquantity, event_datetime,
(event_datetime - INTERVAL '90 day') as day90,
(event_datetime - INTERVAL '180 day') as day180,
(event_datetime - INTERVAL '270 day') as day270,
(event_datetime - INTERVAL '365 day') as day365,
From Wh limit 1
),
Inv_90_days as(
Select sum(onhandquantitydelta) as daysold_90
From Wh cross join days as d
Where event_type = 'Inbound'
And Wh.event_datetime >= d.day90
),
Inv_90_days_final as(
Select case when daysold_90 > d.onhandquantity then d.onshandquantity
```

```
        Else daysold_90
        End daysold_90
From Inv_90_days cross join days as d
),
Inv_180_days as(
Select sum(onhandquantitydelta) as daysold_180
From Wh cross join days as d
Where event_type = 'Inbound'
And Wh.event_datetime between d.day180 and d.day90
),
Inv_180_days_final as(
Select case when daysold_180 > (d.onhandquantity - daysold_90) then d.onshandquantity
        Else daysold_180
        End daysold_180
From Inv_180_days cross join days as d cross join Inv_90_days_final
),


Select daysold_90 as '0-90 days old' , daysold_180 as'91-180 days old'
From Inv_90_days_final cross join inv_180_days_final;
```

Step 6: Now we need to do the same for 181-270 days bucket, and 271-365 bucket

```
With Wh as(
 Select *
From Warehouse
Order by event_datetime desc
),
Days as(
Select onhandquantity, event_datetime,
(event_datetime - INTERVAL '90 day') as day90,
(event_datetime - INTERVAL '180 day') as day180,
(event_datetime - INTERVAL '270 day') as day270,
(event_datetime - INTERVAL '365 day') as day365,
From Wh limit 1
),
Inv_90_days as(
Select coalesce( sum(onhandquantitydelta), 0) as daysold_90
From Wh cross join days as d
Where event_type = 'Inbound'
And Wh.event_datetime >= d.day90
),
Inv_90_days_final as(
Select case when daysold_90 > d.onhandquantity then d.onshandquantity
        Else daysold_90
        End daysold_90
From Inv_90_days cross join days as d
),
Inv_180_days as(
Select coalesce( sum(onhandquantitydelta),0) as daysold_180
From Wh cross join days as d
```

```
Where event_type = ‘Inbound’
And Wh.event_datetime between d.day180 and d.day90
),
Inv_180_days_final as(
Select
case when daysold_180 > (d.onhandquantity - daysold_90) then (d.onshandquantity - daysold_90)
        Else daysold_180
        End daysold_180
From Inv_180_days cross join days as d cross join Inv_90_days_final
),
Inv_270_days as(
Select coalesce(sum(onhandquantitydelta), 0) as daysold_270
From Wh cross join days as d
Where event_type = ‘Inbound’
And Wh.event_datetime between d.day270 and d.day180
),
Inv_270_days_final as(
Select
case when daysold_270 > (d.onhandquantity - (daysold_90 + daysold_180)) then (d.onhandquantity - (daysold_90 + daysold_180))
        Else daysold_270
        End daysold_270
From Inv_270_days
cross join days as d
cross join Inv_90_days_final
Cross join Inv_180_days_final
),
Inv_365_days as(
Select coalesce(sum(onhandquantitydelta), 0) as daysold_365
From Wh cross join days as d
Where event_type = ‘Inbound’
And Wh.event_datetime between d.day365 and d.day270
),
Inv_365_days_final as(
Select
case when daysold_365 > (d.onhandquantity - (daysold_90 + daysold_180 + daysold_270)) then (d.onhandquantity - (daysold_90 + daysold_180 + daysold_270))
        Else daysold_365
        End daysold_365
From Inv_365_days
cross join days as d
cross join Inv_90_days_final
Cross join Inv_180_days_final
Cross join Inv_270_days_final
),

Select daysold_90 as ‘0-90 days old’,
daysold_180 as’91-180 days old’ ,
daysold_270 as ‘181-270 days old’,
daysold_365 as ‘271-365 days old’
```

```
From Inv_90_days_final
Cross join Inv_180_days_final
Cross join Inv_270_days_final
Cross join Inv_365_days_final;
```

Source: https://www.youtube.com/watch?v=xN2PRAd8IZQ&ab_channel=techTFQ

---

5) **[Medium]**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
)
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.

Solution:
Step 1: Group the column of user and label before counting them individually.

```
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;
```

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.

```
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 (CODE BLOCK)
GROUP BY to_user
ORDER BY to_user
```

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;
```

Source:
https://www.stratascratch.com/blog/data-science-sql-interview-questions-from-faang-companies/

---

6) **[Easy]** 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
)

Solution:
Join of our 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.
Step 1: Join the relations

```
SELECT customers.id,
        customers.first_name,
FROM orders
JOIN customers ON customers.id = orders.cust_id;
```

Step 2: Aggregate each customer's order's total costs and then group the relevant column.

```
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;
```

Step 3: 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;
```

Source:
https://www.stratascratch.com/blog/data-science-sql-interview-questions-from-faang-companies/

---

7) **[Hard]**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
);

Solution:
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)

Step 1: Implement product needs to be different + it needs to be purchased in a different date.

```
Select user_id, count(product_id), count(created_at)
From marketing_campaign
Group by user_id
Having count(distinct product_id) > 1
And count(distinct created_at) > 1;
```

(this has handled the scenario 1, 2, and 3)

Step 2: Identify the the user first purchase and date

```
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;
```

Step 3: 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);
```

Source: https://www.stratascratch.com/blog/data-science-sql-interview-questions-from-faang-companies/