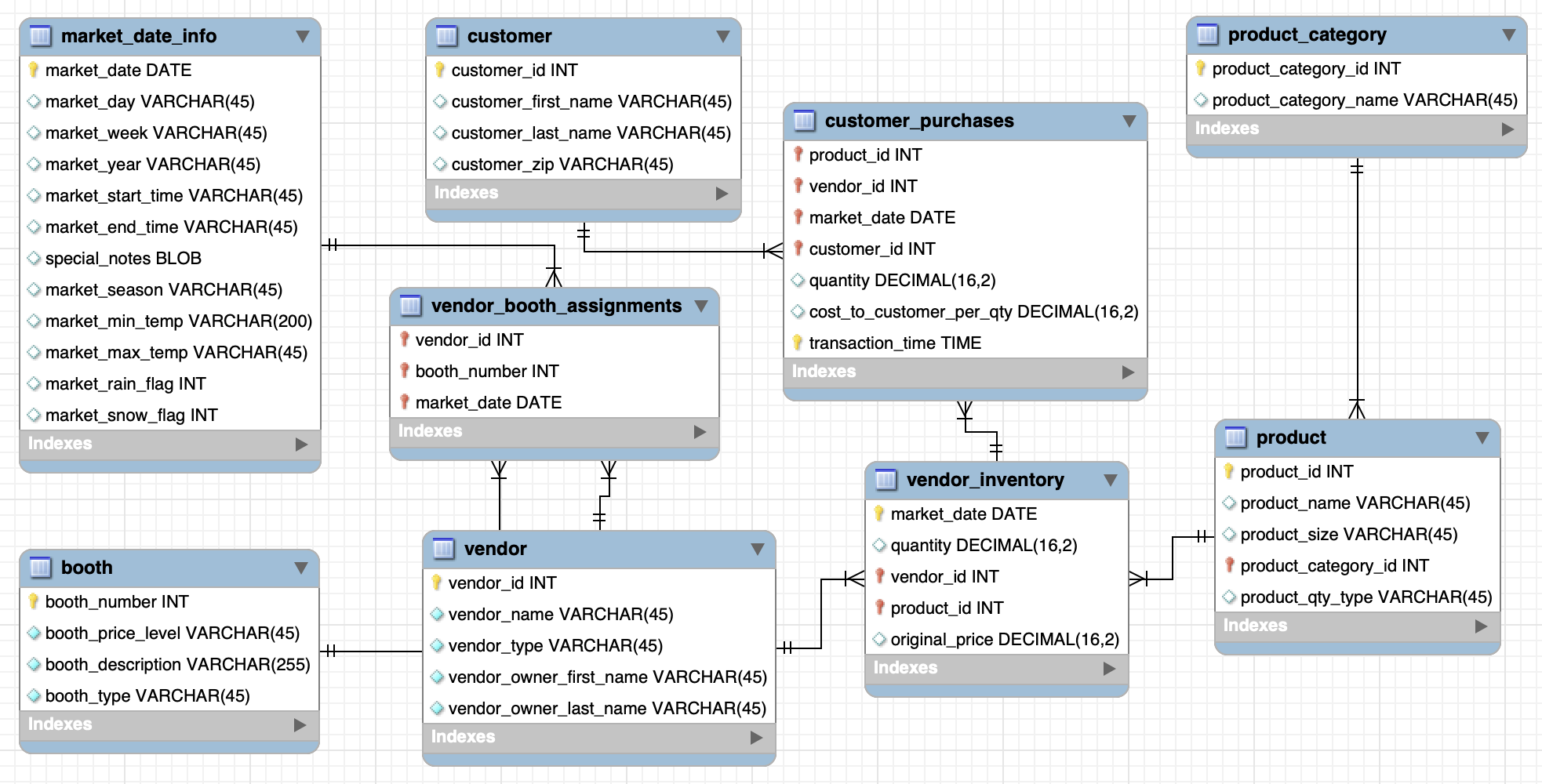
Advanced Constructs - CTEs and Views

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Problem Statement:

You are a Data Analyst at Amazon Fresh. You have been tasked to study the Farmer’s Market.

Dataset: Farmer’s Market database



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## So far…

* Now you know how to work with JOINs, Window functions, and deal with date time values.

Let’s look at a few advanced concepts in SQL that will often come in handy. Until now, we have looked at the simple SELECT statements, let’s go beyond that.

## **Ad-hoc Reporting**

In the data analysis world, being asked questions, exploring a database, writing SQL statements to find and pull the data needed to determine the answers, and conducting the analysis of that data to calculate the answers to the questions, is called **ad-hoc reporting.**

* In advanced SQL, we’ll take those skills to the next level and demonstrate how to think through multiple analysis questions,
* simulate what it might be like to write queries to answer a question posed by a business stakeholder.
* We’ll design and develop analytical datasets that can be used repeatedly to facilitate ad-hoc reporting.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

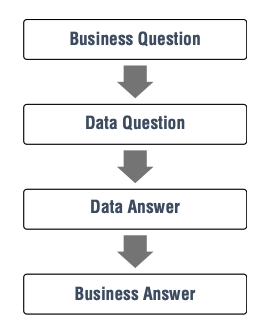
# **The job of a Data Analyst / Scientist**

* As an Analyst, you should be able to anticipate potential follow-up questions is a skill that analysts develop over time through experience.
* For example, if the manager of the farmer’s market asked me “What were the total sales at the market last week?”
* I would expect to be asked more questions after delivering the answer to that one, such as **“How many sales were at the Wednesday market versus the Saturday market last week?”** or **“Can you calculate the total sales over another period?”** or **“Let’s track these weekly market sales over time.”**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Your task is to build a single dataset :

* Given time, you could build a single dataset that could be imported into a reporting system like Tableau and used to answer all of these questions.



In the Farmer’s Market database, the sales are tracked in the **customer\_purchases** table, **which has one row per item purchased** (with multiples of the same item potentially included in a single row, since there is a quantity column).

Each row in **customer\_purchases** contains the -

* ID of the product purchased,
* the ID of the vendor the product was purchased from,
* the ID of the customer making the purchase,
* the transaction date and time, and
* the quantity and cost per quantity.

**Goal**: We want to design the dataset to have one row per date, we do not need to include detailed information about the customers or products.

And because our most granular time dimension is ***market\_date***, we do not need to consider the field that tracks the time of purchase.

## **Step 1: Filter out the data we need**

Write a query that pulls only the fields I need, leaving out unnecessary information and allowing us to summarize at the selected level of detail.

SELECT

market\_date,

quantity \* cost\_to\_customer\_per\_qty

FROM farmers\_market.customer\_purchases

* After reviewing the output without aggregation, we’ll **group** and **sort** by ***market\_date***,
* **SUM** the calculated cost column, **round** it to two decimal places, and give it an **alias of sales.**

SELECT

market\_date,

ROUND(SUM(quantity \* cost\_to\_customer\_per\_qty),2) AS sales

FROM farmers\_market.customer\_purchases

GROUP BY market\_date, vendor\_id

ORDER BY market\_date, vendor\_id

From the output, will you be able to answer the anticipated questions:

* **What were the total sales at the market last week?**
* **How many of last week’s sales were made on Wednesdays versus on Saturdays?**
* **Can we calculate the total sales over another time period?**
* **Can we track the weekly market sales over time?**

## **Step 2: Add some more information from other tables to answer the questions.**

* Some additional information that might be valuable to have on hand for reporting from other tables, including ***market\_day***, ***market\_week***, and ***market\_year*** from the **market\_date\_info** table.

Let’s **LEFT** **JOIN** those into our dataset, so we keep all the existing rows and add more columns where available.

SELECT

cp.market\_date,

md.market\_year, md.market\_week, md.market\_day,

ROUND(SUM(cp.quantity \* cp.cost\_to\_customer\_per\_qty),2) AS sales

FROM farmers\_market.customer\_purchases AS cp

LEFT JOIN farmers\_market.market\_date\_info AS md

ON cp.market\_date = md.market\_date

GROUP BY

cp.market\_date,

md.market\_year, md.market\_week, md.market\_day

ORDER BY

cp.market\_date,

md.market\_year, md.market\_week, md.market\_day

Now we can use this custom dataset to create reports and conduct further analysis.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# **Using custom analytics datasets in SQL using CTEs**

There are multiple ways to store queries (and the results of queries) for reuse in reports and other analyses.

Here, we will cover two approaches for more easily querying from the results of custom dataset queries you build:

1. **Common Table Expressions**
2. **Views.**

Most **database systems**, including MySQL since version 8.0, support Common Table Expressions (CTEs), also known as **“WITH clauses.”**

CTEs allow you to create an alias for an entire query and reference it in other queries like any database table.

**The syntax for CTEs is**:

WITH [query\_alias] AS (

[query]

),

[query\_2\_alias] AS (

[query\_2]

)

SELECT [column list]

FROM [query\_alias]

... [remainder of query that references aliases created above]

where “[**query\_alias**]” is a placeholder for the name you want to use to refer to a query later, and “[**query**]” is a placeholder for the query you want to reuse.

Let’s use it for our use case.

## Question: If we wanted to reuse the previous query we wrote to generate the dataset of sales summarized by date for a report that summarizes sales by market week, we could put that query inside a WITH clause.

WITH sales\_at\_date\_level AS

(

SELECT

cp.market\_date,

md.market\_year, md.market\_week, md.market\_day,

ROUND(SUM(cp.quantity \* cp.cost\_to\_customer\_per\_qty),2) AS sales

FROM farmers\_market.customer\_purchases AS cp

LEFT JOIN farmers\_market.market\_date\_info AS md

ON cp.market\_date = md.market\_date

GROUP BY

cp.market\_date,

md.market\_year, md.market\_week, md.market\_day

ORDER BY

cp.market\_date,

md.market\_year, md.market\_week, md.market\_day

)

SELECT

s.market\_year,

s.market\_week,

SUM(s.sales) AS weekly\_sales

FROM sales\_at\_date\_level AS s

GROUP BY s.market\_year, s.market\_week

**Break down:**

* Notice how the SELECT statement at the bottom references the **sales\_at\_date\_level Common Table Expression using its alias**, treating it just like a table, and even giving it an even shorter alias, s.
* You can filter it, perform calculations, and do anything with its fields that you would do with a normal database table.

**Note:**

* If you want to alias multiple queries in the WITH clause, you put each query inside its own set of parentheses, separated by commas.
* You only use the WITH keyword once at the top, and enter “[alias\_name] AS” before each new query you want to reference later. (The AS is not optional in this case.)
* Each query in the WITH clause can reference any query that preceded it, by using its alias.

**Advantages of CTE:**

* Making recursive query.
* Hold a query output virtually in a temporary area named as given while definition.
* No need to save Metadata.
* Useful when there is a need to do more operations on some query output.
* Query output retain while till then query is running
* Best use of holding temporary data for further processing.
* Allow more grouping options than a single query.
* Allow to get scalar data from a complicated query

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# **Views**

* Another approach to CTEs is Views.
* This **involves storing the query as a database view.**
* A view is treated just like a table in SQL, **the only difference being that it has run when it’s referenced to dynamically generate a result set** (**where a table stores the data instead of storing the query**),
* So **queries that reference views can take longer to run than queries** that reference tables.
* However, the view is retrieving the latest data from the underlying tables each time it is run, so you are working with the freshest data available when you query from a view.

### 

### **How it works!**

* If you want to store your dataset as a view, you simply precede your SELECT statement with
* “CREATE VIEW [schema\_name]. [view\_name] AS ”,  
  replacing the bracketed statements with the actual schema name, and the name you are giving the view.

CREATE VIEW farmers\_market.vw\_sales\_at\_date\_level AS

SELECT

cp.market\_date,

md.market\_year, md.market\_week, md.market\_day,

ROUND(SUM(cp.quantity \* cp.cost\_to\_customer\_per\_qty),2) AS sales

FROM farmers\_market.customer\_purchases AS cp

LEFT JOIN farmers\_market.market\_date\_info AS md

ON cp.market\_date = md.market\_date

GROUP BY

cp.market\_date,

md.market\_year, md.market\_week, md.market\_day

ORDER BY

cp.market\_date,

md.market\_year, md.market\_week, md.market\_day

No results will be displayed when you run this query, other than a confirmation message indicating that the view was created -

*“This statement created a new view named vw\_sales\_at\_date\_level.”*

**Getting Data From the View :**

* Since this dataset we created has one row per market date per vendor, we can filter this view by a range of ***market\_date*** values, just like we could if it were a table.
* Our stored view query is then run to retrieve and summarize data from the underlying tables.

SELECT \*

FROM farmers\_market.vw\_sales\_at\_date\_level AS s

WHERE s.market\_date BETWEEN '2020-04-01' AND '2020-04-30'

ORDER BY market\_date

**Important Notes**

* Because the results of CTEs and views are not stored, they pull the data dynamically each time they are referenced.
* So, if you use the preceding SQL to report on weekly sales using the **vw\_sales\_at\_date\_level** view, each time you run the query, it will include the latest week for which data exists in the **customer\_purchases** table, which the view code references.

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# **SQL CTE vs. View: When to use each one?**

# Although some differences exist, common table expressions and views seem to perform in a very similar way.

So, when should you use each one?

* **Ad-hoc queries.** For queries referenced occasionally (or just once), it’s usually better to use a CTE. If you need the query again, you can just copy the CTE and modify it if necessary.
* **Frequently used queries.** Creating a corresponding view is a good idea if you often reference the same query. However, you’ll need to **create view permission** in your database to create a view.
* **Access management.** A view might restrict particular users’ database access while allowing them to get the necessary information. You can give users access to specific views that query the data they’re allowed to see without exposing the whole database. In such a case, a view provides an additional access layer.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# FAANG Question - Facebook’s Data Scientist Interview Rounds

**Q1.** Given we have a table:

**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;

**Q2.** Given two relations/tables:

**fb\_comments\_count**(

user\_id VARCHAR,

date DATE,

post\_id VARCHAR,

num\_comments INT);

**fb\_active\_users**(

row\_no INT,

user\_id 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.

[**fb\_comments\_count**](https://drive.google.com/file/d/1q3vv-gdOJBIRhP3osVB5nRn58dKVdv7h/view?usp=share_link) table -

| user\_id | date | post\_id | num\_comments |
| --- | --- | --- | --- |
| usr1 | 11/20/2021 | p01 | 10 |
| usr4 | 11/25/2021 | p02 | 83 |
| usr4 | 12/1/2021 | p03 | 96 |
| usr5 | 12/3/2021 | p04 | 40 |
| usr8 | 12/10/2021 | p05 | 23 |
| usr7 | 12/19/2021 | p06 | 50 |
| usr6 | 12/23/2021 | p07 | 35 |
| usr2 | 12/29/2021 | p08 | 12 |
| usr3 | 1/5/2022 | p09 | 23 |
| usr8 | 1/8/2022 | p10 | 18 |
| usr1 | 1/12/2022 | p11 | 50 |
| usr2 | 1/16/2022 | p12 | 46 |
| usr3 | 1/21/2022 | p13 | 6 |
| usr6 | 1/22/2022 | p14 | 80 |
| usr5 | 1/25/2022 | p15 | 20 |
| usr6 | 1/26/2022 | p16 | 23 |
| usr8 | 1/8/2022 | p17 | 55 |
| usr1 | 2/5/2022 | p18 | 10 |
| usr5 | 2/8/2022 | p19 | 12 |

[**fb\_active\_users**](https://drive.google.com/file/d/1Gg_NZG5Qf_EEqHLP6V1k2-69nB-d4-FG/view?usp=share_link) table -

| row\_no | user\_id | Country |
| --- | --- | --- |
| 1 | usr1 | India |
| 2 | usr2 | India |
| 3 | usr3 | Australia |
| 4 | usr4 | Australia |
| 5 | usr5 | USA |
| 6 | usr6 | USA |
| 7 | usr7 | UK |
| 8 | usr8 | UK |

**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.fb\_active\_users as a

LEFT JOIN FB.fb\_comments\_count as b

On a.user\_id = b.user\_id

*Note that `****FB****` is the dataset name here.*

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

With dec\_summary as(

Select \*

FROM FB.fb\_active\_users as a

LEFT JOIN FB.fb\_comments\_count as b

On a.user\_id = b.user\_id

Where date <= ‘2021-12-31’ and date >= ‘2021-12-01’

),

Jan\_summary as(

Select \*

FROM FB.fb\_active\_users as a

LEFT JOIN FB.fb\_comments\_count as b

On a.user\_id = b.user\_id

Where date <= ‘2022-01-31’ and date >= ‘2022-01-01’

)

**Step 3: Exclude rows where the country is empty**

With dec\_summary as(

Select \*

FROM FB.fb\_active\_users as a

LEFT JOIN FB.fb\_comments\_count as b

On a.user\_id = b.user\_id

Where date <= ‘2021-12-31’ and date >= ‘2021-12-01’

AND Country IS NOT NULL

),

Jan\_summary as(

Select \*

FROM FB.fb\_active\_users as a

LEFT JOIN FB.fb\_comments\_count as b

On a.user\_id = b.user\_id

Where date <= ‘2022-01-31 and date >= ‘2022-01-01’

AND Country IS NOT NULL

)

**Step 4: Sum the number of comments per country**

With dec\_summary as(

Select Country,

SUM(num\_comments) as number\_of\_comments\_dec

FROM FB.fb\_active\_users as a

LEFT JOIN FB.fb\_comments\_count as b

On a.user\_id = b.user\_id

Where date <= ‘2021-12-31’ and date >= ‘2021-12-01’

AND Country IS NOT NULL

GROUP BY Country

),

Jan\_summary as(

Select Country,

SUM(num\_comments) as number\_of\_comments\_jan

FROM FB.fb\_active\_users as a

LEFT JOIN FB.fb\_comments\_count as b

On a.user\_id = b.user\_id

Where date <= ‘2022-01-31’ and date >= ‘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(num\_comments) as number\_of\_comments\_dec,

dense\_rank() over(order by sum(num\_comments) DESC) as country\_rank

FROM FB.fb\_active\_users as a

LEFT JOIN FB.fb\_comments\_count as b

On a.user\_id = b.user\_id

Where date <= ‘2021-12-31’ and date >= ‘2021-12-01’

AND Country IS NOT NULL

GROUP BY Country

),

Jan\_summary as(

Select Country,

SUM(num\_comments) as number\_of\_comments\_jan,

dense\_rank() over(order by sum(num\_comments) DESC) as country\_rank

FROM FB.fb\_active\_users as a

LEFT JOIN FB.fb\_comments\_count as b

On a.user\_id = b.user\_id

Where date <= ‘2022-01-31’ and date >= ‘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(num\_comments) as number\_of\_comments\_dec,

dense\_rank() over(order by sum(num\_comments) DESC) as country\_rank

FROM FB.fb\_active\_users as a

LEFT JOIN FB.fb\_comments\_count as b

On a.user\_id = b.user\_id

Where date <= ‘2022-01-31’ and date >= ‘2022-01-01’

AND Country IS NOT NULL

GROUP BY Country

),

Jan\_summary as(

Select Country,

SUM(num\_comments) as number\_of\_comments\_jan,

dense\_rank() over(order by sum(num\_comments) DESC) as country\_rank

FROM FB.fb\_active\_users as a

LEFT JOIN FB.fb\_comments\_count as b

On a.user\_id = b.user\_id;

Where date <= ‘2022-01-31’ and date >= ‘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;

**Complete Solution**

**Query:**

WITH

dec\_summary AS(

SELECT

Country,

SUM(num\_comments) AS number\_of\_comments\_dec,

DENSE\_RANK() OVER(ORDER BY SUM(num\_comments) DESC) AS country\_rank

FROM

FB.fb\_active\_users AS a

LEFT JOIN

FB.fb\_comments\_count AS b

ON

a.user\_id = b.user\_id

WHERE

date <= '2021-12-31'

AND date>= '2021-12-01'

AND Country IS NOT NULL

GROUP BY

Country ),

Jan\_summary AS(

SELECT

Country,

SUM(num\_comments) AS number\_of\_comments\_jan,

DENSE\_RANK() OVER(ORDER BY SUM(num\_comments) DESC) AS country\_rank

FROM

FB.fb\_active\_users AS a

LEFT JOIN

FB.fb\_comments\_count AS b

ON

a.user\_id = b.user\_id

WHERE

date <= '2022-01-31'

AND date >= '2022-01-01'

AND Country IS NOT NULL

GROUP BY

Country )

SELECT

j.country

FROM

jan\_summary j

LEFT JOIN

dec\_summary d

ON

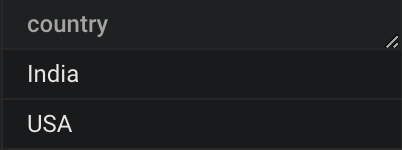
j.country = d.country

WHERE

(j.country\_rank<d.country\_rank)

OR d.country IS NULL;

**Output:**



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# 