**LEETCODE:**

**1.**

**SELECT name**

**FROM customer WHERE IFNULL(referee\_id, 0) <> 2**

**Above is Slower**

**OR**

**SELECT name FROM Customer WHERE referee\_id != 2 OR referee\_id IS NULL;**

**OR**

**select name**

**from Customer**

**where referee\_id <> 2 or coalesce(referee\_id, 0) = 0;**

Above is The Fastest

2.

SELECT name , population, area FROM World WHERE area >= 3000000 OR population >= 25000000;

Pandas Equivalent

df = world[(world['area'] >= 3000000) | (world['population'] >= 25000000)]

3

SELECT DISTINCT author\_id AS id FROM Views WHERE author\_id = viewer\_id ORDER BY author\_id ASC;

Pandas

import pandas as pd

def article\_views(views: pd.DataFrame) -> pd.DataFrame:

return views[(views.author\_id == views.viewer\_id)]\

.rename(columns={"author\_id": "id"})[['id']]\

.drop\_duplicates()\

.sort\_values(by=['id'])

4)

SELECT tweet\_id FROM Tweets WHERE LENGTH(content) >15;

Pandas

def invalid\_tweets(tweets: pd.DataFrame) -> pd.DataFrame:

return tweets[

tweets['content'].str.len() > 15

][['tweet\_id']]

5)

SELECT EmployeeUNI.unique\_id, Employees.name FROM Employees LEFT JOIN EmployeeUNI ON

Employees.id=EmployeeUNI.id;

***After joining if any columns of rows doesn’t have data then it automatically sets NULL***

**Pandas Solution**

import pandas as pd

def replace\_employee\_id(employees: pd.DataFrame, employee\_uni: pd.DataFrame) -> pd.DataFrame:

# Merge the DataFrames using the 'id' column and keep only necessary columns

merged\_df = pd.merge(employees, employee\_uni, how='left', on='id')

# Select the columns 'unique\_id' and 'name' from the merged DataFrame

result\_df = merged\_df[['unique\_id', 'name']]

return result\_df

6)

# Write your MySQL query statement below

SELECT Product.product\_name, Sales.year, Sales.price FROM Sales LEFT JOIN Product ON

Product.product\_id=Sales.product\_id;

Pandas Solution

import pandas as pd

def sales\_analysis(sales: pd.DataFrame, product: pd.DataFrame) -> pd.DataFrame:

merged=pd.merge(sales,product,how="left",on="product\_id")

merged=merged[["product\_name","year","price"]]

return merged

7)

SELECT Visits.customer\_id, COUNT(Visits.customer\_id) AS count\_no\_trans FROM Visits LEFT JOIN Transactions ON Visits.visit\_id=Transactions.visit\_id WHERE Transactions.transaction\_id IS NULL GROUP BY Visits.customer\_id;

SAME RESULT SQL QUERY

# Write your MySQL query statement below

SELECT customer\_id, COUNT(customer\_id) AS count\_no\_trans FROM ( SELECT Visits.customer\_id, Transactions.transaction\_id AS count\_no\_trans FROM Visits LEFT JOIN Transactions ON Visits.visit\_id=Transactions.visit\_id WHERE Transactions.transaction\_id IS NULL) AS subquery GROUP BY customer\_id;

Same Solution in Pandas:

import pandas as pd

def find\_customers(visits: pd.DataFrame, transactions: pd.DataFrame) -> pd.DataFrame:

visits\_no\_trans = visits.merge(transactions, on='visit\_id', how='left')

visits\_no\_trans = visits\_no\_trans[visits\_no\_trans.transaction\_id.isna()]

df = visits\_no\_trans.groupby('customer\_id', as\_index=False)['visit\_id'].count()

return df.rename(columns={'visit\_id': 'count\_no\_trans'})

8)

# Write your MySQL query statement below

SELECT weather.id AS Id

FROM weather

JOIN weather w ON DATEDIFF(weather.recordDate, w.recordDate) = 1 AND weather.Temperature > w.Temperature

Pandas Equivalent Solution

import pandas as pd

def rising\_temperature(weather: pd.DataFrame) -> pd.DataFrame:

# Ensure the 'recordDate' column is a datetime type

weather['recordDate'] = pd.to\_datetime(weather['recordDate'])

# Sorting the DataFrame by 'recordDate' to ensure the shift operation works correctly

weather.sort\_values('recordDate', inplace=True)

# Creating new columns for the previous day's temperature and record date

weather['PreviousTemperature'] = weather['temperature'].shift(1)

weather['PreviousRecordDate'] = weather['recordDate'].shift(1)

# Filtering the DataFrame to find rows where the temperature is higher

# than the previous day and the date is exactly one day more than the previous record date

result = weather[

(weather['temperature'] > weather['PreviousTemperature']) &

(weather['recordDate'] == weather['PreviousRecordDate'] + pd.Timedelta(days=1))

][['id']].rename(columns={'id': 'Id'})

return result

9)

# Write your MySQL query statement below

SELECT

start.machine\_id,

ROUND(AVG(end.timestamp - start.timestamp), 3) AS processing\_time

FROM

Activity AS start

JOIN

Activity AS end ON start.machine\_id = end.machine\_id

AND start.process\_id = end.process\_id

AND start.activity\_type = 'start'

AND end.activity\_type = 'end'

GROUP BY

start.machine\_id;

10)

# Write your MySQL query statement below

SELECT

Students.student\_id,

Students.student\_name,

Subjects.subject\_name,

coalesce(S.attended\_exams,0) AS attended\_exams

FROM

Students

CROSS JOIN

Subjects

LEFT JOIN

(SELECT

student\_id,

subject\_name,

COUNT(subject\_name) AS attended\_exams

FROM

Examinations

GROUP BY

student\_id, subject\_name

) AS S

ON

Students.student\_id = S.student\_id

AND Subjects.subject\_name = S.subject\_name ORDER BY Students.student\_id,Subjects.subject\_name;

11)

SELECT name FROM Employee WHERE id IN (SELECT managerID FROM (SELECT managerID,COUNT(managerId) AS C FROM Employee GROUP BY managerID HAVING C>=5) AS S )

12)

Solution

SELECT Signups.user\_id, ROUND( COALESCE(COALESCE(S2.CON,0)/COALESCE(S3.TOT,0),0),2) AS confirmation\_rate

FROM Signups LEFT JOIN

(SELECT

user\_id,

COUNT(action) AS CON

FROM

Confirmations

WHERE

action = "confirmed"

GROUP BY

user\_id

) AS S2

ON S2.user\_id=Signups.user\_id

LEFT JOIN

(SELECT

user\_id,

COUNT(action) AS TOT

FROM

Confirmations

GROUP BY

user\_id

) AS S3

ON

S3.user\_id=Signups.user\_id

13)

SELECT Prices.product\_id, COALESCE(ROUND(SUM(units\*price)/SUM(units),2),0) AS average\_price FROM Prices LEFT JOIN UnitsSold ON UnitsSold.product\_id=Prices.product\_id AND UnitsSold.purchase\_date BETWEEN Prices.start\_date AND Prices.end\_date GROUP BY Prices.product\_id;

14)

SELECT project\_id, ROUND(AVG(experience\_years),2) AS average\_years FROM Project LEFT JOIN Employee ON Project.employee\_id=Employee.employee\_id GROUP BY project\_id;

15)

SELECT Register.contest\_id, ROUND(COUNT( Register.user\_id)/(SELECT COUNT(DISTINCT user\_id) FROM Users)\*100,2) AS percentage FROM Users LEFT JOIN Register ON Users.user\_id=Register.user\_id GROUP BY Register.contest\_id HAVING Register.contest\_id IS NOT NULL ORDER BY percentage DESC, Register.contest\_id ASC;

16)

SELECT P.query\_name, ROUND(AVG(P.R),2) AS quality, COALESCE(ROUND(L/COUNT(L)\*100,2),0) AS poor\_query\_percentage FROM (SELECT K.query\_name,rating/position AS R,S.C AS L FROM Queries AS K LEFT JOIN (SELECT query\_name, COUNT(rating) AS C FROM Queries WHERE rating<3 GROUP BY query\_name) AS S ON S.query\_name=K.query\_name) AS P WHERE P.query\_name IS NOT NULL

GROUP BY query\_name;

SAME RESULT QUERY

SELECT

Q.query\_name,

ROUND(AVG(Q.rating / Q.position), 2) AS quality,

COALESCE(ROUND(SUM(CASE WHEN Q.rating < 3 THEN 1 ELSE 0 END) / COUNT(Q.rating) \* 100, 2), 0) AS poor\_query\_percentage

FROM

Queries AS Q WHERE Q.query\_name IS NOT NULL

GROUP BY

Q.query\_name;

17)

SELECT DATE\_FORMAT(trans\_date, '%Y-%m') AS month , country, COUNT(id) AS trans\_count, SUM(CASE WHEN state="approved" THEN 1 ELSE 0 END) AS approved\_count, SUM(amount) AS trans\_total\_amount, SUM(CASE WHEN state="approved" THEN amount ELSE 0 END) AS approved\_total\_amount

FROM Transactions GROUP BY DATE\_FORMAT(trans\_date, '%Y-%m'),country;

18)

SELECT ROUND(SUM( CASE WHEN order\_type='immediate' THEN 1 ELSE 0 END)/COUNT(customer\_id )\*100,2) AS immediate\_percentage

FROM (

SELECT delivery\_id,

customer\_id,

order\_date,

customer\_pref\_delivery\_date AS pref,

CASE

WHEN order\_date = customer\_pref\_delivery\_date THEN 'immediate'

ELSE 'scheduled'

END AS order\_type,

ROW\_NUMBER() OVER (PARTITION BY customer\_id ORDER BY order\_date, customer\_pref\_delivery\_date) AS rn

FROM Delivery

) AS S

WHERE rn = 1;

19)

WITH FirstLoginDates AS (

SELECT

player\_id,

MIN(event\_date) AS first\_login\_date

FROM

Activity

GROUP BY

player\_id

)

SELECT

ROUND(

COUNT(DISTINCT f.player\_id) /

(SELECT COUNT(DISTINCT player\_id) FROM Activity),

2

) AS fraction

FROM

FirstLoginDates f

JOIN

Activity a ON f.player\_id = a.player\_id AND a.event\_date = DATE\_ADD(f.first\_login\_date, INTERVAL 1 DAY);

20)

SELECT activity\_date AS day, COUNT(DISTINCT user\_id) AS active\_users FROM Activity WHERE activity\_date >"2019-06-27" AND activity\_date<= "2019-07-27" GROUP BY activity\_date;

Same type sOLUTION

SELECT activity\_date AS day, COUNT(DISTINCT user\_id) AS active\_users FROM Activity WHERE activity\_date BETWEEN date\_sub('2019-07-27', INTERVAL 29 DAY)

AND "2019-07-27" GROUP BY activity\_date;

21)

WITH F AS

(

SELECT product\_id, MIN(year) as year, quantity, price FROM Sales GROUP BY product\_id

)

SELECT S.product\_id, F.year AS first\_year, S.quantity, S.price

FROM Sales S

LEFT JOIN F ON F.product\_id = S.product\_id AND F.year = S.year

WHERE F.year IS NOT NULL;

22)

SELECT class FROM Courses GROUP BY class HAVING COUNT(student)>=5;

23)

SELECT user\_id, COUNT(follower\_id) AS followers\_count FROM Followers GROUP BY user\_id ORDER BY user\_id;

24)

The COALESCE function is typically used when you want to provide a fallback value in case a column is NULL. However, in this context, it seems you want to handle the case where there are no rows meeting the condition and return NULL in that specific situation.

The COALESCE in your query doesn't change the result because you're providing NULL as the fallback value, which is essentially saying "if num is NULL, then return NULL, otherwise, return num."

To explicitly return NULL when there are no rows meeting the condition, you can use a subquery and adjust the logic:

SELECT COALESCE((SELECT MAX(num)

FROM MyNumbers

GROUP BY num

HAVING COUNT(num) = 1

ORDER BY num DESC

LIMIT 1), NULL) AS num;

24)

WITH F AS

(SELECT customer\_id ,product\_key FROM Customer WHERE product\_key IN(SELECT product\_key FROM Product)

)

SELECT F.customer\_id FROM F GROUP BY F.customer\_id HAVING COUNT( DISTINCT F.product\_key) = (SELECT COUNT(product\_key) FROM Product)

Easier Solution

SELECT customer\_id

FROM Customer

GROUP BY customer\_id

HAVING COUNT(DISTINCT product\_key) = (SELECT COUNT(product\_key) FROM Product);

25)

WITH F AS

(

SELECT employee\_id, name FROM Employees WHERE employee\_id IN (SELECT DISTINCT reports\_to FROM Employees WHERE reports\_to IS NOT NULL)

),

S AS

( SELECT reports\_to, COUNT(\*) AS reports\_count , AVG(age) AS average\_age FROM Employees WHERE reports\_to IS NOT NULL GROUP BY reports\_to)

SELECT F.employee\_id AS employee\_id , F.name AS name , S.reports\_count AS reports\_count , ROUND(S.average\_age) AS average\_age FROM F LEFT JOIN S ON

S.reports\_to=F.employee\_id ORDER BY F.employee\_id

Easier Solution:

# Write your MySQL query statement below

SELECT

emp1.employee\_id,

emp1.name,

COUNT(emp2.employee\_id) AS reports\_count,

ROUND(AVG(emp2.age)) AS average\_age

FROM Employees emp1

INNER JOIN Employees emp2 ON emp1.employee\_id = emp2.reports\_to

GROUP BY emp1.employee\_id

ORDER BY emp1.employee\_id

26)

SELECT

employee\_id,

department\_id

FROM

Employee

WHERE

primary\_flag = 'Y'

UNION

SELECT

employee\_id,

department\_id

FROM

Employee

GROUP BY

employee\_id

HAVING

COUNT(employee\_id) = 1;

27)

SELECT x,y,z ,(CASE

WHEN x + y <= z OR x + z <= y OR y + z <= x THEN "No" ELSE "Yes" END)AS triangle FROM Triangle;

28)

SELECT DISTINCT

l1.Num AS ConsecutiveNums

FROM

Logs l1,

Logs l2,

Logs l3

WHERE

l1.Id = l2.Id - 1

AND l2.Id = l3.Id - 1

AND l1.Num = l2.Num

AND l2.Num = l3.Num

;

29)

SELECT

product\_id,

10 AS price

FROM

Products

GROUP BY

product\_id

HAVING

MIN(change\_date) > '2019-08-16'

UNION ALL

SELECT

product\_id,

new\_price AS price

FROM

Products

WHERE

(product\_id, change\_date) IN (

SELECT

product\_id,

MAX(change\_date)

FROM

Products

WHERE

change\_date <= '2019-08-16'

GROUP BY

product\_id

)

30)

# Write your MySQL query statement below

WITH D AS

(

SELECT

turn, person\_name,

SUM(weight) OVER (ORDER BY turn) AS cumulative\_sum

FROM

Queue

)

SELECT person\_name FROM D WHERE D.cumulative\_sum<=1000 ORDER BY D.cumulative\_sum DESC LIMIT 1;

31)

WITH F AS

(

SELECT account\_id,income, (CASE

WHEN income <20000 THEN "Low Salary"

WHEN income BETWEEN 20000 AND 50000 THEN "Average Salary"

WHEN income >50000 THEN "High Salary"

END ) AS cat FROM Accounts

)

SELECT "Low Salary" AS category , COUNT(cat) AS accounts\_count FROM F WHERE cat="Low Salary"

UNION

SELECT "Average Salary" AS category, COUNT(cat) AS accounts\_count FROM F WHERE cat="Average Salary"

UNION

SELECT "High Salary" AS category, COUNT(cat) AS accounts\_count FROM F WHERE cat="High Salary"

32)

# Write your MySQL query statement below

SELECT employee\_id FROM Employees WHERE salary<30000 AND manager\_id NOT IN(SELECT employee\_id FROM Employees) ORDER BY employee\_id ASC

33)

SELECT

CASE

WHEN id = (SELECT MAX(id) FROM seat) AND id % 2 = 1

THEN id

WHEN id % 2 = 1

THEN id + 1

ELSE id - 1

END AS id,

student

FROM seat

ORDER BY id

34)

# Write your MySQL query statement below

(SELECT Users.name AS results FROM MovieRating LEFT JOIN Users ON MovieRating.user\_id=Users.user\_id

GROUP BY MovieRating.user\_id ORDER BY COUNT(\*) DESC,Users.name LIMIT 1 )

UNION ALL

(SELECT Movies.title AS results FROM MovieRating LEFT JOIN Movies ON MovieRating.movie\_id =Movies.movie\_id WHERE created\_at BETWEEN "2020-02-01" AND "2020-02-29"

GROUP BY Movies.title ORDER BY AVG(rating) DESC, Movies.title ASC LIMIT 1)

Correct Use of Parentheses: When using ORDER BY and LIMIT within subqueries that are part of a UNION or UNION ALL, you need to enclose each subquery in parentheses. However, MySQL allows the use of ORDER BY and LIMIT without parentheses in the final part of a UNION/UNION ALL query. Your query does not show syntax errors in this aspect because MySQL's flexibility with UNION operations, but it's something to keep in mind for more strict SQL environments or complex queries.