Sample SQL Queries From Leetcode

1. **Restaurant Growth:** Write an SQL query to compute moving average of how much customer paid in a 7 days window (current day + 6 days before). (Link to problem statement)

# MySQL Solution:

```
SELECT a.visited_on,SUM(b.amount) as amount, ROUND(avg( b.amount),2) as average_amount

FROM (SELECT visited_on, sum(amount) as amount

    FROM Customer

    group by visited_on) a

JOIN

(SELECT visited_on,SUM(amount) as amount

    FROM Customer

    GROUP BY visited_on) b

on datediff(a.visited_on,b.visited_on) between 0 and 6

group by a.visited_on

having count(distinct b.visited_on) =7

order by a.visited_on
```

2. **All People Report to the Given Manager:** Write an SQL query to find employee_id of all employees that directly or indirectly report their work to the head of the company. (Link to problem statement)

# MySQL Solution:

```
SELECT d.employee_id
FROM
(SELECT *
FROM Employees
WHERE manager_id = 1) a
JOIN
Employees b
on a.employee_id = b.manager_id
JOIN
Employees c
on b.employee_id = c.manager_id
JOIN
Employees d
on c.employee_id = d.manager_id
WHERE d.employee_id != 1
```

3. **Product Price at a Given Date:** Write an SQL query to find the prices of all products on **2019-08-16**. Assume the price of all products before any change is **10**. (Link to problem statement)

# MySQL Solution:

```
SELECT DISTINCT a.product_id,

IFNULL (b.new_price,10) AS price

    FROM products a
    LEFT JOIN (select product_id,new_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
                                            )) b
    ON a.product_id = b.product_id
```

4. **Movie Rating: Write the following SQL query:** (Link to problem statement)
Find the name of the user who has rated the greatest number of the movies. In case of a tie, return lexicographically smaller user name.
Find the movie name with the *highest average* rating in February 2020. In case of a tie, return lexicographically smaller movie name.

# MySQL Solution:

```
(SELECT b.name as results

FROM Movie_Rating  a

JOIN  Users b

ON b.user_id = a.user_id

GROUP BY a.user_id

ORDER by COUNT(rating) DESC,name

LIMIT 1)

UNION
```

```
(SELECT d.title as results

FROM Movies d

JOIN  (SELECT * FROM Movie_Rating  WHERE SUBSTRING(created_at,1,7)

      like '2020-02') c

ON c.movie_id = d.movie_id

GROUP BY c.movie_id

ORDER by AVG(rating) DESC,title

LIMIT 1)
```

5. **Monthly Transactions II:** Write an SQL query to find for each month and country, the number of approved transactions and their total amount, the number of chargebacks and their total amount. (Link to problem statement)

```
# MySQL Solution

SELECT  month,country,
        SUM(CASE WHEN state='approved'THEN 1 ELSE 0 END) AS approved_count,
        SUM(CASE WHEN state='approved'THEN amount ELSE 0 END) AS approved_amount,
        SUM(CASE WHEN state='chargeback'THEN 1 ELSE 0 END) AS chargeback_count,
        SUM(CASE WHEN state='chargeback'THEN amount ELSE 0 END) AS chargeback_amount


FROM

(SELECT id,country,state,amount, SUBSTRING(trans_date,1,7) as month
FROM
Transactions
 WHERE state ='approved'

UNION ALL

SELECT trans_id as id, b.country,'chargeback' as state,b.amount, SUBSTRING(a.trans_date,1,7) as
month
FROM Chargebacks a

JOIN  Transactions b
on b.id=a.trans_id )  c

GROUP BY month,country
```

6. **Tree Node:** Write a query to print the node id and the type of the node. Sort your output by the node id [(Link to problem statement)]

# MySQL Solution

```
SELECT id, CASE WHEN p_id is NULL THEN 'Root'
                WHEN p_id is NOT NULL AND id in (SELECT p_id FROM tree) THEN 'Inner'
                ELSE 'Leaf' END
                AS Type
FROM tree
```

7. **Game Play Analysis III:** Write an SQL query that reports for each player and date, how many games played **so far** by the player. That is, the total number of games played by the player until that date. (Link to problem statement)

# MySQL Solution

```
SELECT player_id, event_date, sum(games_played) over (PARTITION BY player_id)

AS games_played_so_far

FROM activity

ORDER BY player_id, games_played_so_far;
```

8. **Page Recommendations:** Write an SQL query to recommend pages to the user with `user_id` = 1 using the pages that your friends liked. It should not recommend pages you already liked (Link to problem statement)

# MySQL Solution

```
SELECT DISTINCT page_id as recommended_page

FROM

(SELECT

 (CASE

       WHEN user1_id=1 THEN user2_id

      WHEN user2_id=1 THEN user1_id

      ELSE 0 END) as Friends

  FROM Friendship

 ) a

JOIN
```

```
(SELECT *
FROM Likes
WHERE  page_id NOT IN (SELECT page_id
                FROM Likes
                WHERE user_id=1)) b
 ON a.Friends=b.user_id
```

9. **Investments in 2016:** Write a query to print the sum of all total investment values in 2016 (TIV_2016), to a scale of 2 decimal places, for all policy holders who meet the following criteria:
Have the same TIV_2015 value as one or more other policyholders.
Are not located in the same city as any other policyholder (i.e.: the (latitude, longitude) attribute pairs must be unique). (Link to problem statement)

# MySQL Solution

```
SELECT
SUM(insurance.TIV_2016) AS TIV_2016
FROM
insurance
WHERE
    insurance.TIV_2015 in
    (SELECT TIV_2015
    FROM insurance
    GROUP BY TIV_2015
    HAVING COUNT(*)>1)

AND
    CONCAT(LAT,LON)
    IN
    (SELECT CONCAT(LAT,LON)
        FROM insurance
    GROUP BY LAT,LON
    HAVING COUNT(*)=1
    )
```