Celebal week-6 Assignment

Leetcode SQL-50

1.Recyclable and Low Fat Products

```
SELECT product_id

FROM Products

WHERE low_fats = "Y" AND recyclable = "Y";
```

2.Find Customer Referee

```
SELECT name FROM Customer WHERE referee_id != 2 OR referee_id IS NULL;
```

3.Big Countries

```
SELECT name, population, area

FROM World

WHERE area >= 3000000 OR population >= 25000000;
```

4.Article views I

```
SELECT DISTINCT author_id AS id

FROM Views

WHERE author_id = viewer_id

ORDER BY author_id ASC;
```

5.Invalid Tweets

```
SELECT tweet_id

FROM Tweets

WHERE LENGTH(content) > 15;
```

6. Replace Employee ID With The Unique Identifier

```
SELECT name, unique_id
FROM Employees
LEFT JOIN EmployeeUNI USING(id);
```

7. Product Sales Analysis I

```
SELECT product_name, year, price
FROM Product
JOIN Sales USING (product_id)
```

8. Customer Who Visited but Did Not Make Any Transactions

```
SELECT customer_id,  COUNT(customer_id) AS count_no_trans
FROM Visits
LEFT JOIN Transactions USING (visit_id)
WHERE transaction_id IS NULL
GROUP BY customer_id;
```

9. Rising Temperature

```
SELECT w1.id
FROM Weather w1
JOIN Weather w2
WHERE DATEDIFF(w1.recordDate, w2.recordDate) = 1
   AND w1.temperature > w2.temperature;
```

10. Average Time of Process per Machine

```
SELECT a.machine_id,
   ROUND(AVG(b.timestamp - a.timestamp), 3) AS processing_time
```

```
FROM Activity a
JOIN Activity b
ON a.machine_id = b.machine_id
   AND a.process_id = b.process_id
   AND a.activity_type = "start"
   AND b.activity_type = "end"
GROUP BY a.machine_id;
```

11. Employee Bonus

```
SELECT name, bonus
FROM Employee
LEFT JOIN Bonus USING(empID)
WHERE bonus < 1000 OR bonus IS NULL;
```

12. Students and Examinations

```
SELECT
   s.student_id,
   s.student_name,
   sub.subject_name,
   COUNT(e.subject_name) AS attended_exams
FROM Students s
CROSS JOIN Subjects sub
LEFT JOIN Examinations e
   ON s.student_id = e.student_id AND sub.subject_name = e.subject_name
GROUP BY s.student_id, s.student_name, sub.subject_name
ORDER BY s.student_id, sub.subject_name;
```

13. Managers with at Least 5 Direct Reports

```
SELECT e1.name
FROM Employee e1
JOIN Employee e2 ON e1.id = e2.managerId
GROUP BY e1.id HAVING COUNT(*) >= 5;
```

14. Confirmation Rate

```
SELECT
   user_id,
   ROUND(IFNULL(SUM(action = "confirmed")/COUNT(*), 0), 2) AS confirmation_rate
FROM Signups
LEFT JOIN Confirmations USING (user_id)
GROUP BY user_id;
```

15. Not Boring Movies

```
SELECT *
FROM Cinema
WHERE id % 2 = 1 AND description != "boring"
ORDER BY rating DESC;
```

16. Average Selling Price

```
SELECT
   p.product_id,
   IFNULL(ROUND(SUM(p.price * u.units) / SUM(u.units), 2), 0) AS average_price
FROM Prices p
LEFT JOIN UnitsSold u ON p.product_id = u.product_id
AND u.purchase_date BETWEEN p.start_date AND p.end_date
GROUP BY product_id;
```

17. Project Employees I

```
SELECT
    project_id,
    ROUND(AVG(experience_years), 2) AS average_years
FROM Project
JOIN Employee USING(employee_id)
GROUP BY project_id;
```

18. Percentage of Users Attended a Contest

```
SELECT
    contest_id,
    -- unique users registered for this contest
    -- total number of users in the system
    ROUND(COUNT(DISTINCT user_id) * 100 / (SELECT COUNT(*) FROM Users), 2) AS
percentage
FROM Users
JOIN Register USING(user_id)
GROUP BY contest_id
ORDER BY percentage DESC, contest_id ASC;
```

19. Queries Quality and Percentage

```
SELECT
    query_name,
    ROUND(AVG(rating / position), 2) AS quality,
    ROUND(AVG(rating < 3) * 100, 2) AS poor_query_percentage
FROM Queries
GROUP BY query_name;
```

20. Monthly Transactions I

```
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 month, country;
```

21. Immediate Food Delivery II

```
SELECT
   ROUND(100 * AVG(d.customer_pref_delivery_date = f.first_order_date), 2) AS
immediate_percentage
FROM Delivery d
JOIN (
   SELECT
      customer_id,
      MIN(order_date) AS first_order_date
   FROM Delivery
   GROUP BY customer_id
) f
ON d.customer_id = f.customer_id AND d.order_date = f.first_order_date;
```

22. Game Play Analysis IV

```
SELECT
```

```
    ROUND(COUNT(DISTINCT d1.player_id) / (SELECT COUNT(DISTINCT player_id) FROM
Activity), 2)
    AS fraction
FROM Activity d1
JOIN Activity d2
    ON d1.player_id = d2.player_id
    AND DATEDIFF(d2.event_date, d1.event_date) = 1
WHERE d1.event_date = (
    SELECT MIN(event_date)
    FROM Activity
    WHERE player_id = d1.player_id
);
```

23. Number of Unique Subjects Taught by Each Teacher

```
SELECT teacher_id, COUNT(DISTINCT subject_id) AS cnt
FROM Teacher
GROUP BY teacher_id;
```

24. User Activity for the Past 30 Days I

```
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;
```

25. User Activity for the Past 30 Days I

```
SELECT
```

```
    s.product_id,
    f.first_year,
    s.quantity,
    s.price
FROM Sales s
JOIN (
    SELECT
        product_id,
        MIN(year) AS first_year
    FROM Sales
    GROUP BY product_id
) f
ON s.year = f.first_year AND s.product_id = f.product_id;
```

26. Classes With at Least 5 Students

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

27. Find Followers Count

```
SELECT user_id, COUNT(follower_id) AS followers_count
FROM Followers
GROUP BY user_id
ORDER BY user_id;
```

28. Biggest Single Number

```
SELECT MAX(num) AS num
FROM (
   SELECT num
   FROM MyNumbers
   GROUP BY num
   HAVING COUNT(*) = 1
) AS singles;
```

29. Customers Who Bought All Products

```
SELECT customer_id
FROM Customer
JOIN Product USING (product_key)
GROUP BY customer_id
HAVING COUNT(DISTINCT product_key) = (SELECT COUNT(*) From Product);
-- distinct user brought products compared to total products
```

30. The Number of Employees Which Report to Each Employee

```
SELECT
   e1.employee_id,
   e1.name,
   COUNT(e1.employee_id) AS reports_count,
   ROUND(AVG(e2.age), 0) AS average_age
FROM Employees e1, Employees e2
WHERE e2.reports_to = e1.employee_id
GROUP BY employee_id
ORDER BY employee_id;
```

31. Primary Department for Each Employee

```
SELECT employee_id, department_id
FROM Employee
WHERE primary_flag = "Y" OR employee_id IN (
    SELECT employee_id
    FROM Employee
    GROUP BY employee_id
    HAVING COUNT(employee_id) = 1
  );
```

32.Triangle Judgement

```
SELECT x, y, z,
(CASE WHEN (x + y > z) AND (x + z > y) AND (y + z > x) THEN "Yes" ELSE "No" END)
   AS triangle
FROM Triangle;
```

33. Consecutive Numbers

```
SELECT DISTINCT t.num ConsecutiveNums
FROM Logs f
JOIN Logs s ON f.id = s.id + 1
JOIN Logs t ON s.id = t.id + 1
WHERE f.num = s.num AND s.num = t.num
```

34. Product Price at a Given Date

```
SELECT
   p.product_id,
```

```
    -- if price found is on or before 2019-08-16, use it.
    -- or else, use the default price of 10
    IFNULL(pr.new_price, 10) AS price

-- step 1: create a list of unique product_ids
FROM (
    SELECT DISTINCT product_id
    FROM Products
) p

-- step 2: for each product, find the most recent price change
LEFT JOIN(
    SELECT product_id, new_price
    FROM Products
    WHERE change_date <= "2019-08-16"

        -- step 3: only keep the most recent change for each product before target date as there could be multiple change_date updates for same product
        AND (product_id, change_date) IN(
            SELECT product_id, MAX(change_date)
            FROM Products
            WHERE change_date <= "2019-08-16"
            GROUP BY product_id -- only one date per product
        )
) pr ON p.product_id = pr.product_id;
```

35. Last Person to Fit in the Bus

```
WITH total_weight AS(
```

```
    SELECT
        person_name,
        turn,
        weight,
        SUM(weight) OVER (ORDER BY turn) AS total_weight
    FROM Queue
)

-- main query:
SELECT tw.person_name
FROM total_weight tw
WHERE total_weight <= 1000
ORDER BY turn DESC
LIMIT 1;
```

36.Count Salary Categories

```
SELECT "Low Salary" AS category,
    SUM(CASE WHEN income < 20000 THEN 1 ELSE 0 END) AS accounts_count
FROM Accounts

UNION ALL

SELECT 'Average Salary' AS category,
    SUM(CASE WHEN income >= 20000 AND income <= 50000 THEN 1 ELSE 0 END) AS
accounts_count
FROM Accounts

UNION ALL
```

```
SELECT 'High Salary' AS category,
   SUM(CASE WHEN income > 50000 THEN 1 ELSE 0 END) AS accounts_count
FROM Accounts;
```

37. Employees Whose Manager Left the Company

```
SELECT employee_id
FROM Employees
WHERE salary < 30000 AND manager_id NOT IN(
   SELECT employee_id
   FROM Employees
)
ORDER BY employee_id;
```

38. Exchange Seats

```
SELECT
   CASE
      -- odd
      WHEN id % 2 = 1 AND id != (
         SELECT MAX(id)
         FROM Seat
      )
      THEN id + 1
      WHEN id % 2 = 0 THEN id - 1
      ELSE id
   END AS id,
   student
```

```
FROM Seat
ORDER BY id;
```

39. Movie Rating

```
WITH most_rated_user AS (
    SELECT name
    FROM Movies
    JOIN MovieRating USING(movie_id)
    JOIN Users USING(user_id)
    GROUP BY user_id
    ORDER BY COUNT(*) DESC, name ASC
    LIMIT 1
),

highest_rated_movie AS (
    SELECT title
    FROM MovieRating
    JOIN Movies USING (movie_id)
    WHERE created_at BETWEEN "2020-02-01" AND "2020-02-29"
    GROUP BY movie_id
    ORDER BY AVG(rating) DESC, title ASC
    LIMIT 1
)


SELECT name AS results
FROM most_rated_user
```

```
UNION ALL

SELECT title AS results
FROM highest_rated_movie
```

40. Restaurant Growth

```
WITH daily_amount AS (
   SELECT
      visited_on,
      SUM(amount) AS amount
   FROM Customer
   GROUP BY visited_on
)

-- main query to calculate the 7-day moving average
SELECT
   d1.visited_on,
   SUM(d2.amount) AS amount, -- sum of the 7-day window
   ROUND(AVG(d2.amount), 2) AS average_amount
FROM daily_amount d1 -- current day
JOIN daily_amount d2 -- previous 6 days + current day
   ON d2.visited_on BETWEEN DATE_SUB(d1.visited_on, INTERVAL 6 DAY) AND d1.visited_on
GROUP BY d1.visited_on
HAVING COUNT(*) = 7 -- only include rows where we have a full 7-day window
ORDER BY d1.visited_on;
```

41.Friend Requests II: Who Has the Most Friends

```
SELECT id, COUNT(*) AS num
```

```
FROM (
  SELECT requester_id AS id
  FROM RequestAccepted
  UNION ALL
  SELECT accepter_id AS id
  FROM RequestAccepted
) AS all_friends
GROUP BY id
ORDER BY num DESC
LIMIT 1;
```

42. Investments in 2016

```
WITH dup_tiv AS (
   SELECT tiv_2015
   FROM Insurance
   GROUP BY tiv_2015
   HAVING COUNT(*) > 1
),

unique_loc AS (
   SELECT lat, lon
   FROM Insurance
   GROUP BY lat, lon
   HAVING COUNT(*) = 1
)

SELECT ROUND(SUM(tiv_2016), 2) AS tiv_2016
FROM Insurance
```

```
WHERE tiv_2015 IN (SELECT * FROM dup_tiv)
   AND (lat, lon) IN (SELECT lat, lon FROM unique_loc);
```

43. Department Top Three Salaries

```
WITH ranked_salaries AS (
   SELECT
      d.name AS Department,
      e.name AS Employee,
      e.salary AS Salary,
      DENSE_RANK() OVER (
         PARTITION BY e.departmentId
         ORDER BY e.salary DESC
         ) AS salary_ranking
   FROM Employee e
   JOIN Department d ON e.departmentId = d.id
)
SELECT Department, Employee, Salary
FROM ranked_salaries
WHERE salary_ranking <= 3;
```

44. Fix Names in a Table

```
SELECT
   user_id,
   CONCAT(UPPER(LEFT(name, 1)), LOWER(SUBSTRING(name, 2))) AS name
FROM Users
ORDER BY user_id;
```

45. Patients With a Condition

```
SELECT
    patient_id,
    patient_name,
    conditions
FROM Patients
WHERE conditions LIKE "% DIAB1%"
    OR conditions LIKE "DIAB1%";
```

46. Delete Duplicate Emails

```
DELETE a
FROM Person a
JOIN Person b
WHERE a.email = b.email
    AND a.id > b.id;
```

47. Second Highest Salary

```
SELECT (
    SELECT DISTINCT salary
    FROM Employee
    ORDER BY salary DESC
    LIMIT 1 OFFSET 1
) AS SecondHighestSalary;
```

48. Group Sold Products By The Date

```
SELECT
    sell_date,
    COUNT(DISTINCT product) AS num_sold,
```

```
    GROUP_CONCAT(DISTINCT product) AS products
FROM Activities
GROUP BY sell_date
ORDER BY sell_date
```

49. List the Products Ordered in a Period

```
SELECT
    product_name,
    SUM(unit) AS unit
FROM Products JOIN Orders USING (product_id)
WHERE Order_date BETWEEN "2020-02-01" AND "2020-02-29"
GROUP BY product_name
HAVING unit >= 100
```

50.Find Users With Valid E-Mails

```
SELECT *
FROM Users
WHERE REGEXP_LIKE(mail,'^[A-Za-z][A-Za-z0-9._-]*@leetcode\\.com$', 'c');
```