**Question: 1**

**Write an SQL query to swap the students between every pair of consecutive seats (id = 1 with id = 2, id = 3 with id = 4, etc.). If the total number of seats is odd, the student in the last seat should remain unchanged.**

Example 1:

Input:

Seat table:

|  |  |
| --- | --- |
| **id** | **student** |
| 1 | Abbot |
| 2 | Doris |
| 3 | Emerson |
| 4 | Green |
| 5 | Jeames |

Output:

|  |  |
| --- | --- |
| **id** | **student** |
| 1 | Doris |
| 2 | Abbot |
| 3 | Green |
| 4 | Emerson |
| 5 | Jeames |

**Answer:**

SELECT

  CASE

    WHEN id % 2 = 1 AND id <> (SELECT MAX(id) FROM exe\_temp1) THEN id + 1

    WHEN id % 2 = 0 THEN id - 1

    ELSE id

  END AS id,

  student

FROM exe\_temp1

ORDER BY id;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Which records will be returned by applying INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN, and CROSS JOIN between the given Table1 and Table2?

|  |
| --- |
| **id** |
| 1 |
| 1 |
| 2 |
| 2 |
| 4 |
| NULL |

|  |
| --- |
| **id** |
| 1 |
| 1 |
| 2 |
| 3 |
| NULL |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write an SQL query to retrieve all employees and order them by hiring\_date, placing employees with NULL or empty hiring\_date values at the bottom of the result set?

Input Table :-

|  |  |  |
| --- | --- | --- |
| **id** | **name** | **hiring\_date** |
| 1 | John | 2020-05-12 |
| 2 | Alice | NULL |
| 3 | Bob | 2018-11-03 |
| 4 | David | (blank) |
| 5 | Emma | 2021-01-20 |

Expected Table:-

**Employee Hiring Dates**

|  |  |  |
| --- | --- | --- |
| **id** | **name** | **hiring\_date** |
| 3 | Bob | 2018-11-03 |
| 1 | John | 2020-05-12 |
| 5 | Emma | 2021-01-20 |
| 2 | Alice | NULL |
| 4 | David |  |

Answer:

SELECT \*

FROM employees

ORDER BY hiring\_date NULLS LAST;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write an SQL query to transform the data into a ledger-style format that shows credits, debits, and a running balance.

Input Table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Description** | **Type** | **Amount** |
| 12-Dec-24 | EMI Amount | credit | 10,000 |
| 12-Dec-24 | Paid to friend | debit | 1,000 |
| 13-Dec-24 | Loan amount | debit | 5,000 |

Expected Output:

WITH entries AS (

  SELECT

    "Date",

    "Description",

    "Type",

    "Amount"::numeric(12,2) AS amount,

    CASE WHEN "Type" = 'credit' THEN  "Amount"::numeric

         WHEN "Type" = 'debit'  THEN - "Amount"::numeric

    END AS signed\_amount,

    ROW\_NUMBER() OVER (

      PARTITION BY "Date"

      ORDER BY CASE WHEN "Type"='credit' THEN 0 ELSE 1 END, "Description"

    ) AS seq

  FROM ledger

)

SELECT

  to\_char("Date",'YYYY-MM-DD')                  AS "Date",

  "Description",

  CASE WHEN "Type"='credit' THEN amount END     AS credit,

  CASE WHEN "Type"='debit'  THEN amount END     AS debit,

  SUM(signed\_amount) OVER (

    ORDER BY "Date", seq

    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

  )                                             AS balance

FROM entries

ORDER BY "Date", seq;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write an SQL query to fetch the 3rd highest salary using window functions. Explain why you chose the specific window function.

|  |  |
| --- | --- |
| Id | salary |
| 1 | 7000 |
| 2 | 10000 |
| 3 | 6000 |
| 4 | 9000 |
| 5 | 8000 |
| 6 | 9000 |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write an SQL query to fetch the 3rd highest salary in each department.

# Department Salary Table

|  |  |  |
| --- | --- | --- |
| id | department | salary |
| 1 | HR | 5000 |
| 2 | HR | 7000 |
| 3 | HR | 6000 |
| 4 | HR | 7000 |
| 5 | IT | 9000 |
| 6 | IT | 8000 |
| 7 | IT | 7500 |
| 8 | IT | 9500 |
| 9 | Sales | 4000 |
| 10 | Sales | 4500 |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write an SQL query to find the list of unique customers and their orders where the orders belong to the category of FMCG products.

Assume:

customers table has: id (PK), name, …

orders table has: order\_id (PK), category, customer\_id (FK → customers.id),

|  |  |
| --- | --- |
| **id** | **name** |
| 1 | Alice |
| 2 | Bob |
| 3 | Charlie |

|  |  |  |
| --- | --- | --- |
| **order\_id** | **category** | **customer\_id** |
| 101 | FMCG | 1 |
| 102 | Electronics | 1 |
| 103 | FMCG | 2 |
| 104 | FMCG | 2 |
| 105 | Fashion | 3 |

Expected Output:

|  |  |  |  |
| --- | --- | --- | --- |
| **customer\_id** | **customer\_name** | **order\_id** | **category** |
| 1 | Alice | 101 | FMCG |
| 2 | Bob | 103 | FMCG |
| 2 | Bob | 104 | FMCG |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write an SQL query to calculate the 7-day moving average of sales for each product

Input Table

+------------+------------+--------------+

| product\_id | sale\_date  | sales\_amount |

+------------+------------+--------------+

| P1         | 2024-01-01 | 100          |

| P1         | 2024-01-02 | 200          |

| P1         | 2024-01-03 | 150          |

| P1         | 2024-01-04 | 300          |

| P1         | 2024-01-05 | 250          |

| P1         | 2024-01-06 | 400          |

| P1         | 2024-01-07 | 350          |

| P1         | 2024-01-08 | 500          |

+------------+------------+--------------+

Expected Output

+------------+------------+------------------+

| product\_id | sale\_date  | moving\_avg\_7\_days|

+------------+------------+------------------+

| P1         | 2024-01-01 | 100.0            |

| P1         | 2024-01-02 | 150.0            |

| P1         | 2024-01-03 | 150.0            |

| P1         | 2024-01-04 | 187.5            |

| P1         | 2024-01-05 | 200.0            |

| P1         | 2024-01-06 | 233.3            |

| P1         | 2024-01-07 | 250.0            |

| P1         | 2024-01-08 | 307.1            |

+------------+------------+------------------+

**SELECT PRODUCT\_ID, SALE\_DATE, SALES\_AMOUNT,**

**ROUND(AVG(SALES\_AMOUNT) over (PARTITION PRODUCT\_ID ORDER BY SALEES\_DATE)**

**UNBOUNDED 6 PRECEDING ROW AND CURRENT ROW)),2) AS ROLLING\_AVG**

**FROM PRODUCT\_SALES;**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write a query to list managers whose salary is lower than their subordinate’s salary.

Input Table:

+----+----------+-----------+---------+

| id | name     | manager\_id| salary  |

+----+----------+-----------+---------+

|  1 | Alice    | 10        | 9000    |

|  2 | Bob      | 5         | 12000   |

|  3 | Charlie  | 1         | 11000   |

|  4 | David    | 3         | 8000    |

|  5 | Eva      | 1         | 9500    |

|  6 | Frank    | 2         | 7000    |

+----+----------+-----------+---------+

+----------+

| manager  |

+----------+

| Alice    |

+----------+

**select e.empname,e.empid**

**from emp e inner join emp m**

**where e.empid<> m.empid and e.salary > m.salary**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

You have a table called sales with the columns: sale\_id (unique identifier), customer\_id, sale\_date, amount

Write an SQL query to retrieve the top 2 highest sales per customer within the last 30 days.

Input Table:-

+---------+-------------+------------+--------+

| sale\_id | customer\_id | sale\_date  | amount |

+---------+-------------+------------+--------+

| 101     | 1           | 2025-08-05 |    500 |

| 102     | 1           | 2025-08-15 |    700 |

| 103     | 1           | 2025-08-20 |    300 |

| 104     | 2           | 2025-08-12 |    900 |

| 105     | 2           | 2025-08-22 |    850 |

| 106     | 2           | 2025-08-28 |    400 |

+---------+-------------+------------+--------+

Expected Output:-

+-------------+---------+------------+--------+

| customer\_id | sale\_id | sale\_date  | amount |

+-------------+---------+------------+--------+

| 1           | 102     | 2025-08-15 |    700 |

| 1           | 101     | 2025-08-05 |    500 |

| 2           | 104     | 2025-08-12 |    900 |

| 2           | 105     | 2025-08-22 |    850 |

+-------------+---------+------------+--------+

Answer:

**select \***

**from (select customer\_id, sale\_id,sale\_date, amount,**

**dense\_rank() over (partition by customer\_id order by amount DESC) as rk**

**from public.sales)**

**where rk in (1,2) and sale\_date >= current\_date - interval '30 day';**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

You are given a table ExamResults with the following columns: id ,Name, Subject, Result

Write an SQL query to find students who have passed in all the exams they appeared for.

Input Table:-

+----+------+---------+--------+

| id | Name | Subject | Result |

+----+------+---------+--------+

| 1  | A    | AA      | Pass   |

| 2  | B    | AA      | Pass   |

| 3  | C    | CC      | Fail   |

| 3  | C    | AA      | Fail   |

| 2  | B    | BB      | Fail   |

+----+------+---------+--------+

Expected Output:-

+----+------+

| id | Name |

+----+------+

| 1  | A    |

|  |  |
| --- | --- |
|  |  |

**select id, name**

**from student**

**group by id**

**having count(\*) = count(case when result ='pass' then 1 end);**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

You have two tables: Employee : emp\_id , emp\_name

Salary: emp\_id, salary\_date, salary

Q. Write an SQL query to calculate the percentage hike in salary for each employee compared to their previous salary record.

Input Table

+--------+-------------+--------+   +--------+----------+

| emp\_id | salary\_date | salary |   | emp\_id | emp\_name |

+--------+-------------+--------+   +--------+----------+

| 1      | 2024-01-01  | 50000  |   | 1      | Alice    |

| 1      | 2025-01-01  | 60000  |   | 2      | Bob      |

| 2      | 2024-06-01  | 45000  |   | 3      | Carol    |

| 2      | 2025-01-01  | 49500  |   +--------+----------+

| 3      | 2024-03-01  | 70000  |

| 3      | 2025-01-01  | 73500  |

+--------+-------------+--------+

Expected Output:-

+--------+----------+-------------+--------+-------------+--------+--------------+

| emp\_id | emp\_name | prev\_salary | p\_date | curr\_salary | c\_date | hike\_percent |

+--------+----------+-------------+--------+-------------+--------+--------------+

| 1      | Alice    | 50000       | 2024-01-01 | 60000  | 2025-01-01 | 20.00     |

| 2      | Bob      | 45000       | 2024-06-01 | 49500  | 2025-01-01 | 10.00     |

| 3      | Carol    | 70000       | 2024-03-01 | 73500  | 2025-01-01 | 5.00      |

+--------+----------+-------------+--------+-------------+--------+--------------+

**Ans: SELECT**

**emp\_id,**

**effective\_date,**

**salary,**

**LAG(salary) OVER (**

**PARTITION BY emp\_id**

**ORDER BY effective\_date**

**) AS prev\_salary,**

**ROUND(**

**(salary - LAG(salary) OVER (PARTITION BY emp\_id ORDER BY effective\_date))**

**\* 100.0 / NULLIF(LAG(salary) OVER (PARTITION BY emp\_id ORDER BY effective\_date), 0)**

**, 2) AS pct\_change -- positive = hike, negative = drop**

**FROM employee\_salaries**

**ORDER BY emp\_id, effective\_date;**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write an SQL query to calculate the 3-month rolling sales for each product using window functions.

Input Table:

+---------+------------+------------+--------+

| sale\_id | product\_id | sale\_date  | amount |

+---------+------------+------------+--------+

| 1       | P1         | 2025-01-01 | 100    |

| 2       | P1         | 2025-02-15 | 200    |

| 3       | P1         | 2025-03-05 | 300    |

| 4       | P1         | 2025-04-10 | 400    |

| 5       | P2         | 2025-01-20 | 500    |

| 6       | P2         | 2025-02-25 | 600    |

| 7       | P2         | 2025-04-02 | 700    |

+---------+------------+------------+--------+

Expected Output:-

+------------+------------+------------+------------------+

| product\_id | sale\_date  | amount     | rolling\_3m\_sales |

+------------+------------+------------+------------------+

| P1         | 2025-01-01 | 100        | 100              |

| P1         | 2025-02-15 | 200        | 300              |

| P1         | 2025-03-05 | 300        | 600              |

| P1         | 2025-04-10 | 400        | 900              |

| P2         | 2025-01-20 | 500        | 500              |

| P2         | 2025-02-25 | 600        | 1100             |

| P2         | 2025-04-02 | 700        | 1300             |

+------------+------------+------------+------------------+

Select product\_id, sale\_Date, amount,

Sum(amount) over (partition by product\_id order by sale\_date RANGE BETWEEN INTERVAL '3 months' PRECEDING AND CURRENT ROW) as rolling\_3m sales

From product

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write an SQL query to display the names (ids) of teachers who teach Math only (i.e., they don’t teach any other subject).

Input Table:-

+----+---------+

| id | subject |

+----+---------+

| 1  | Math    |

| 2  | Math    |

| 4  | Chem    |

| 5  | Math    |

| 2  | Eng     |

| 3  | Phy     |

+----+---------+

Expected Table:-

+----+

| id |

+----+

| 1  |

| 5  |

Select Id from course group by id having count(distinct id=1) and sub=’math’;

Or

SELECT DISTINCT t1.id

FROM teachers t1

WHERE t1.subject = 'Math'

AND NOT EXISTS (

SELECT 1

FROM teachers t2

WHERE t2.id = t1.id

AND t2.subject <> 'Math'

);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write an SQL query to delete the duplicate records while keeping only one unique record for each duplicate set.

+--------+----+-------+--------+

| emp\_id | id | name  | salary |

+--------+----+-------+--------+

| 1      | 1  | A     | 5000   |

| 2      | 2  | B     | 6000   |

| 3      | 2  | B     | 6000   |  <-- duplicate

| 4      | 3  | C     | 7000   |

| 5      | 4  | D     | 8000   |

| 6      | 4  | D     | 8000   |  <-- duplicate

+--------+----+-------+--------+

Expected Output:-

+--------+----+-------+--------+

| emp\_id | id | name  | salary |

+--------+----+-------+--------+

| 1      | 1  | A     | 5000   |

| 2      | 2  | B     | 6000   |

| 4      | 3  | C     | 7000   |

| 5      | 4  | D     | 8000   |

+--------+----+-------+--------+

With CTE emp\_rank as (

Select emp\_id, id, name salary , row\_number() over (partition by id,name,salary) as rk

),

Delete from emp where rk>1;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write an SQL query to find the maximum number of consecutive login days for each user.

Input Table:-

+----+----------+------------+

| id |  value   | login\_date |

+----+----------+------------+

| 1  | val1     | 01-08-2025 |

| 1  | val2     | 02-08-2025 |

| 1  | val2     | 03-08-2025 |

| 1  | val3     | 05-08-2025 |

| 2  | val4     | 01-08-2025 |

| 2  | val5     | 02-08-2025 |

| 2  | val5     | 05-08-2025 |

| 3  | val7     | 01-08-2025 |

+----+----------+------------+

Expected Output:-

+----+------------------+

| id | max\_consecutive  |

+----+------------------+

| 1  | 3                |

| 2  | 2                |

| 3  | 1                |

|  |  |
| --- | --- |
|  |  |

**with cte as (select \*,**

**login\_date - cast(dense\_rank() over (partition by user\_id order by user\_id,login\_date) as int) as date\_group**

**from user\_login)**

**select user\_id, date\_group, min(login\_date) as start\_date, max(login\_date) as end\_date,**

**(max(login\_date)-min(login\_date))+1 as consecutive\_day from cte**

**group by user\_id, date\_group**

**having (max(login\_date)-min(login\_date))+1 >=4**

**order by user\_id, consecutive\_day**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write an SQL query to merge two tables Basket1 and Basket2 on the Product field so that the output includes all values from both tables

Input Table

+---------+

| Product |

+---------+

| Sugar   |

| Bread   |

| Juice   |

| Soda    |

| Flour   |

+---------+

+---------+

| Product |

+---------+

| Sugar   |

| Bread   |

| Butter  |

| Cheese  |

| Fruit   |

+---------+

Expected Output:-

+----------+----------+

| Bucket\_1 | Bucket\_2 |

+----------+----------+

| Sugar    | Sugar    |

| Bread    | Bread    |

| Juice    | NULL     |

| Soda     | NULL     |

| Flour    | NULL     |

| NULL     | Butter   |

| NULL     | Cheese   |

| NULL     | Fruit    |

|  |  |
| --- | --- |
| Ans: use full outer join |  |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write an SQL query to perform a pivot operation that shows one row per ID with separate columns for each phone type.

Input Table:

+------+-----------+--------------+

| ID   | CELL\_TYPE | CELL\_NO      |

+------+-----------+--------------+

| 1001 | Cellular  | 555-897-5421 |

| 1001 | Work      | 555-897-6542 |

| 1001 | Home      | 555-698-9874 |

| 2002 | Cellular  | 555-963-6544 |

| 2002 | Work      | 555-812-9856 |

| 3003 | Cellular  | 555-987-6541 |

+------+-----------+--------------+

Expected Output:-

+------+--------------+--------------+--------------+

| id   | Cellular     | Work         | Home         |

+------+--------------+--------------+--------------+

| 1001 | 555-897-5421 | 555-897-6542 | 555-698-9874 |

| 2002 | 555-963-6544 | 555-812-9856 | NULL         |

| 3003 | 555-987-6541 | NULL         | NULL         |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

SELECT

ID,

MAX(CASE WHEN CELL\_TYPE = 'Cellular' THEN CELL\_NO END) AS Cellular,

MAX(CASE WHEN CELL\_TYPE = 'Work' THEN CELL\_NO END) AS Work,

MAX(CASE WHEN CELL\_TYPE = 'Home' THEN CELL\_NO END) AS Home

FROM Phones

GROUP BY ID

ORDER BY ID;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write an SQL query to find the candidate\_id values of candidates who have all the required descriptions.

Input Table:-

+--------------+-------------+

| candidate\_id | Description |

+--------------+-------------+

| 1001         | Geologist   |

| 1001         | Astrogator  |

| 1001         | BioChemist  |

| 1001         | Technician  |

| 1001         | Mechanic    |

| 2002         | Geologist   |

| 3003         | Geologist   |

| 3003         | Astrogator  |

| 4004         | BioChemist  |

+--------------+-------------+

Expected Output:-

+--------------+

| candidate\_id |

+--------------+

| 1001         |

+--------------+  
SELECT candidate\_id

FROM CandidateSkills

WHERE description IN ('Geologist','Astrogator','BioChemist','Technician','Mechanic')

GROUP BY candidate\_id

HAVING COUNT(DISTINCT description) = 5;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write a query to return records that are unique to TableB when comparing both tables on col\_id.

Input Table:-

TableA

+--------+

| col\_id |

+--------+

| 1      |

| 1      |

| 2      |

| NULL   |

+--------+

TableB

+--------+

| col\_id |

+--------+

| 1      |

| 1      |

| NULL   |

| 3      |

| 5      |

+--------+

Expected Table:-

+--------+--------+

| col\_id | col\_id |

+--------+--------+

| NULL   | 3      |

| NULL   | 5      |

|  |  |
| --- | --- |
|  |  |

SELECT NULL AS col\_id\_A, col\_id AS col\_id\_B

FROM (

SELECT col\_id FROM TableB

EXCEPT

SELECT col\_id FROM TableA

) diff;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write a query find all employees whose salary is greater than the average salary of their department.

Input Table:-

+----+---------+-------+---------+

| id | name    | sal   | dept    |

+----+---------+-------+---------+

|  1 | Alice   | 60000 | HR      |

|  2 | Bob     | 75000 | Finance |

|  3 | Charlie | 50000 | IT      |

|  4 | Diana   | 82000 | IT      |

|  5 | Ethan   | 55000 | Sales   |

|  6 | Fiona   | 67000 | Finance |

+----+---------+-------+---------+

Expected Output:-

+----+-------+-------+---------+

| id | name  | sal   | dept    |

+----+-------+-------+---------+

|  2 | Bob   | 75000 | Finance |

|  4 | Diana | 82000 | IT      |

+----+-------+-------+---------+

SELECT id, name, dept\_id, salary

FROM (

SELECT e.\*,

AVG(salary) OVER (PARTITION BY dept\_id) AS avg\_salary

FROM Employees e

) t

WHERE t.salary > t.avg\_salary;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write a query to find the starting city and the ending city for each passenger’s complete journey.

Input Table:-

+----+-----------+-------------+

| id | source    | destination |

+----+-----------+-------------+

|  1 | Mumbai    | Hyderabad   |

|  1 | Hyderabad | Bangalore   |

|  2 | Delhi     | Jaipur      |

|  2 | Jaipur    | Agra        |

|  3 | Chennai   | Pune        |

|  3 | Pune      | Goa         |

|  3 | Goa       | Mysore      |

+----+-----------+-------------+

Expected Output:-

+----+-----------+-----------+

| id | start     | end       |

+----+-----------+-----------+

|  1 | Mumbai    | Bangalore |

|  2 | Delhi     | Agra      |

|  3 | Chennai   | Mysore    |

|  |  |  |
| --- | --- | --- |
|  |  |  |

WITH start\_city AS (

SELECT DISTINCT t.id, t.source AS start

FROM Trips t

LEFT JOIN Trips t2

ON t.id = t2.id AND t.source = t2.destination

WHERE t2.id IS NULL -- sources that never appear as a destination → journey start

),

end\_city AS (

SELECT DISTINCT t.id, t.destination AS "end"

FROM Trips t

LEFT JOIN Trips t2

ON t.id = t2.id AND t.destination = t2.source

WHERE t2.id IS NULL -- destinations that never appear as a source → journey end

)

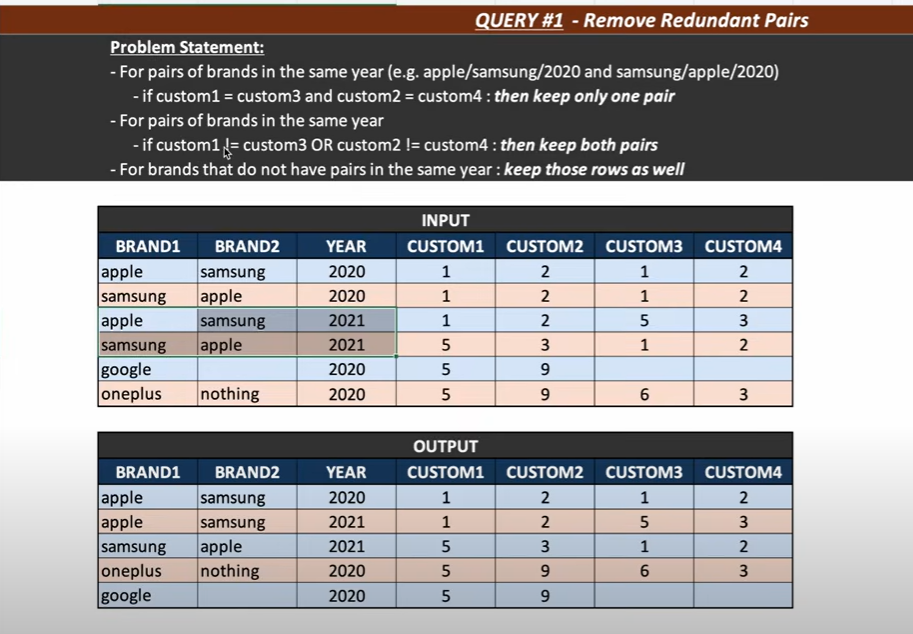
SELECT s.id, s.start, e."end"

FROM start\_city s

JOIN end\_city e USING (id)

ORDER BY s.id;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



**Ans: with cte as (**

**select \*,**

**case when brand1 < brand2 then concat(brand1, brand2, year)**

**else concat(brand2, brand1, year) end as pair\_id2**

**from brands),**

**rank\_pair as (**

**select \*, row\_number() over(partition by pair\_id2 order by pair\_id2) as rnk from cte )**

**select \* from rank\_pair where rnk=1 or custom1<>custom3 and custom2<>custom4**