Data Engineering Interview Questions

1. You want to find employees who earn more than their managers from the given employee table.

id	name	salary	managerID
1	Alice	5000	NULL
2	Bob	4000	1
3	Charlie	7000	1
4	David	3000	2
5	Eve	7000	2
6	Frank	2000	3

Find Employees Who Earn More Than Their Managers

Logic:

We join the employee table to itself:

- `e` = employee
- `m` = manager (based on `e.managerID = m.id`)
  Then compare: `e.salary > m.salary`

SQL Query:

```
SELECT e.id, e.name, e.salary, m.name AS manager_name, m.salary AS manage
FROM employee e
JOIN employee m ON e.managerID = m.id
WHERE e.salary > m.salary;
```

---

2. how to find the second highest salary using Window Function in SQL

```
SELECT id, name, salary
FROM (
    SELECT *,
           DENSE_RANK() OVER (ORDER BY salary DESC) AS rnk
    FROM employee
```

```
) ranked
WHERE rnk = 2;
```

Explanation:

- `DENSE_RANK()` assigns the same rank to duplicate salaries.
- We filter `where rank = 2` to get the second highest salary.

---

3. Delete Duplicate Emails (Without Window Functions)?

Id	Email
1	alice@mail.com
2	bob@mail.com
3	alice@mail.com
4	charlie@mail.com
5	bob@mail.com

```
DELETE FROM email_table
WHERE id NOT IN (
    SELECT MIN(id)
    FROM email_table
    GROUP BY email
);
```

- GROUP BY email groups duplicates.
- MIN(id) keeps the first occurrence.
- DELETE removes others.

---

4. add 1 to the last element of the list ( )

```
input list1 = [1, 2, 3, 4]
output list = [1, 2, 3, 5]

Input = [1, 2, 3, 9]
output= [1, 2, 4, o]
```

Answer:

```
def add_one_to_list(digits):
    for i in range(len(digits) - 1, -1, -1):  # start from the end
        if digits[i] < 9:
            digits[i] += 1
            return digits
        else:
            digits[i] = 0
    # If loop completes, all digits were 9
    return [1] + digits

# Test cases
print(add_one_to_list([1, 2, 3, 4]))  # Output: [1, 2, 3, 5]
print(add_one_to_list([1, 2, 3, 9]))  # Output: [1, 2, 4, 0]
print(add_one_to_list([9, 9, 9]))     # Output: [1, 0, 0, 0]
```

- This code adds 1 to a number that's represented as a list of digits — like how [1, 2, 3] means 123.
- I loop through the list from the end:
- If the digit is less than 9, I can just add 1 and return the result.
- But if the digit is 9, I set it to 0 and carry the 1 to the left.
- If all digits are 9, like [9, 9, 9], the loop finishes and I add a 1 at the beginning. So [9, 9, 9] becomes [1, 0, 0, 0].

---

5. Write a script to get manager id, manager name, count of the employess and the avg salary of employees reporting to the manager.

Employee Table

EmpID	Name	Dept	Salary	ManagerID
101	John	A	90000	001
102	Dan	A	100000	101
103	James	A	90000	101
104	Amy	A	80000	101
105	Anne	A	80000	101
106	Ron	B	70000	101

EmpID	Name	Dept	Salary	ManagerID
108	Joe	A	60000	106
109	Frank	A	10000	106

```sql
SELECT
    m.id AS manager_id,
    m.name AS manager_name,
    COUNT(e.id) AS employee_count,
    AVG(e.salary) AS avg_employee_salary
FROM employees e
JOIN employees m ON e.managerId = m.id
GROUP BY m.id, m.name;
```

---

6. Which customers ordered both keyboard and mouse?

```sql
SELECT customer_id
FROM orders
WHERE product IN ('keyboard', 'mouse')
GROUP BY customer_id
HAVING COUNT(DISTINCT product) = 2;
```

---

7. Which customers ordered both keyboard and mouse?

```sql
SELECT c.customer_id, c.first_name, c.last_name
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id
WHERE o.item IN ('Keyboard', 'Mouse')
GROUP BY c.customer_id, c.first_name, c.last_name
HAVING COUNT(DISTINCT o.item) = 2;
```

---

8. Find duplicates in a list and print?

Sample list

data = [1, 2, 3, 2, 4, 5, 1, 6, 3]

```
seen = set()
duplicates = set()

for item in data:
    if item in seen:
        duplicates.add(item)
    else:
        seen.add(item)
print("Duplicates:", list(duplicates))
```

---

9. Write a PySpark program to output the total salary paid per department per month, with the following columns:

`dep_name, salary, month_salary (where month_salary is in yyyy-MM format)`

You are given two JSON files:

- `department.json` containing `dep_id, dep_name`
- `employee.json` containing `emp_id, dep_id, salary, salary_date`

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import date_format, sum as _sum

# Start Spark session
spark = SparkSession.builder.appName("DepartmentMonthlySalary").getOrCrea

# Load JSON files
dept_df = spark.read.json("department.json")
emp_df = spark.read.json("employee.json")

# Extract month in yyyy-MM format
emp_df = emp_df.withColumn("month_salary", date_format("salary_date", "yy

# Join employee with department on dep_id
joined_df = emp_df.join(dept_df, on="dep_id", how="inner")

# Group by dep_name and month_salary to get total salary
result_df = joined_df.groupBy("dep_name", "month_salary").agg(
    _sum("salary").alias("salary")
)
```

```
# Show result
result_df.show()
```

10. From a student table based on student ID best of 3 marks using sql and avg of that for best of three?

```
SELECT
  student_id,
  ROUND(AVG(marks), 2) AS best_of_3_avg
FROM (
  SELECT
    student_id,
    marks,
    ROW_NUMBER() OVER (PARTITION BY student_id ORDER BY marks DESC) AS ra
  FROM student_marks
) ranked
WHERE rank <= 3
GROUP BY student_id;
```

Explanation:

- `ROW_NUMBER()` assigns ranks based on descending marks per student.
- Only the top 3 marks are selected per student.
- `AVG()` then calculates the average of these top 3 marks.
- `ROUND(..., 2)` is used to round the average to 2 decimal places.

---

11. select names that start with A

```
SELECT name
FROM your_table
WHERE name LIKE 'A%';
```

12. second Highest Salary; Rank, row number and dense rank difference

```
--  Second Highest Salary using DENSE_RANK
SELECT *
```

```
  SELECT name, salary, DENSE_RANK() OVER (ORDER BY salary DESC) AS rnk
  FROM employee
) AS ranked
WHERE rnk = 2;
--  Second Highest Salary using RANK
SELECT *
FROM (
  SELECT name, salary, RANK() OVER (ORDER BY salary DESC) AS rnk
  FROM employee
) AS ranked
WHERE rnk = 2;
--  Second Highest Salary using ROW_NUMBER (not ideal if duplicates exist
SELECT *
FROM (
  SELECT name, salary, ROW_NUMBER() OVER (ORDER BY salary DESC) AS rn
  FROM employee
) AS ranked
WHERE rn = 2;
```

Difference Between RANK, DENSE_RANK, and ROW_NUMBER:

- `RANK()`
- ➢ Skips next number on tie.
- ➢ E.g., Ranks: 1, 2, 2, 4
- `DENSE_RANK()`
- ➢ No gaps on tie.
- ➢ E.g., Ranks: 1, 2, 2, 3
- `ROW_NUMBER()`
- ➢ Always unique, no ties allowed.
- ➢ E.g., 1, 2, 3, 4 regardless of duplicate salaries.

Use:

- `DENSE_RANK()` or `RANK()` when handling ties.
- `ROW_NUMBER()` when a unique position is needed

---

13. Write an SQL query to return the top 3 products with the highest total quantity sold in the last 6 months

Sales Data

sale_id	product_id	quantity_sold	sale_date
1	101	10	2025-05-01
2	102	5	2025-03-15
3	103	20	2025-04-20
4	101	15	2025-02-25
5	102	8	2024-12-30
6	104	25	2025-05-10
7	103	5	2025-01-10
8	101	7	2025-03-05
9	105	30	2025-04-01
10	104	10	2024-11-01

Answer:

```
WITH filtered_sales AS (
  SELECT *
  FROM sales
  WHERE sale_date >= DATE '2025-06-17' - INTERVAL '6 months'
),
product_totals AS (
  SELECT
    product_id,
    SUM(quantity_sold) AS total_quantity_sold
  FROM filtered_sales
  GROUP BY product_id
),
ranked_products AS (
  SELECT *,
         RANK() OVER (ORDER BY total_quantity_sold DESC) AS rank
  FROM product_totals
)
```

```
FROM ranked_products
WHERE rank <= 3;
```

---

14. List all employees who earn more than their manager.

```
SELECT emp_id, name, salary, manager_id
FROM (
    SELECT e.*,
           MAX(salary) OVER (PARTITION BY emp_id) AS manager_salary
    FROM employees e
    LEFT JOIN employees m
        ON e.manager_id = m.emp_id
) sub
WHERE salary > manager_salary;
```

- We use a window function `MAX(salary) OVER (PARTITION BY emp_id)` to get the manager's salary.
- Finally, we filter to keep only employees whose `salary > manager_salary`.

15.Identify duplicate email addresses in the users table?

SQL Query to Find Duplicate Emails

```
SELECT email, COUNT(*) AS count
FROM users
GROUP BY email
HAVING COUNT(*) > 1;
```

Explanation:

- `GROUP BY` email groups all rows by the email address.
- `COUNT(*)` counts how many times each email appears.
- `HAVING COUNT(*) > 1` filters only those emails that appear more than once (i.e., duplicates).

---

16. List all employees who are not assigned to any project?

```sql
SELECT emp_id, name
FROM (
    SELECT e.emp_id, e.name,
           COUNT(pa.project_id) OVER (PARTITION BY e.emp_id) AS project_c
    FROM employees e
    LEFT JOIN project_assignments pa
        ON e.emp_id = pa.emp_id
) sub
WHERE project_count = 0;
```

Explanation:

- We `LEFT JOIN` employees with project_assignments.
- We use `COUNT(project_id) OVER (PARTITION BY e.emp_id)` to count how many projects each employee is assigned to (even if it's 0).
- We filter for `project_count = 0` to get employees not assigned to any project.

---

17. SQL Query to Find People with Same Name Across Different Countries

```sql
SELECT Name
FROM your_table
GROUP BY Name
HAVING COUNT(DISTINCT Country) > 1;
```

---

18. How to retrieve unique handshakes from a table using a window function?

```sql
CREATE TABLE people (name VARCHAR(30));

INSERT INTO people (name) VALUES
('Bheem'),
('Shyam'),
('Ram');
```

Answer:

```sql
WITH ranked_people AS (
  SELECT name, ROW_NUMBER() OVER (ORDER BY name) AS rn
  FROM people
)
SELECT
  CONCAT(p1.name, '-', p2.name) AS handshake
FROM ranked_people p1
JOIN ranked_people p2
  ON p1.rn < p2.rn
ORDER BY handshake;
-- # Output
-- Bheem-Ram
-- Bheem-Shyam
-- Ram-Shyam
```

Explanation (Short)

- `ROW_NUMBER()` assigns a unique order to each person.
- The self-join with p1.rn < p2.rn ensures:
- No self-pairs (e.g., Ram-Ram)
- No duplicate pairs (e.g., both Ram-Bheem and Bheem-Ram)
- This gives unique, unordered handshake pairs.

---

19. Write a SQL query to retrieve the employee details from the Employee table

- such that the results are returned in alternating values of male and female.
- Also make sure that the employees are arranged as per their age (i.e., oldest person at the top and youngest person at the bottom for male, and youngest person at the top and oldest person at the bottom for female).

```sql
WITH male_cte AS (
  SELECT *,
         ROW_NUMBER() OVER (ORDER BY dob ASC) AS rn_m
  FROM Employee
  WHERE gender = 'M'
),
female_cte AS (
  SELECT *,
```

```
  FROM Employee
  WHERE gender = 'F'
),
merged AS (
  SELECT rn_f AS rn, emp_id, emp_name, gender, dob FROM female_cte
  UNION ALL
  SELECT rn_m AS rn, emp_id, emp_name, gender, dob FROM male_cte
)
SELECT
  emp_id, emp_name, gender, dob,
  DATEDIFF(YEAR, dob, GETDATE()) AS age
FROM merged
ORDER BY rn, gender DESC;  -- Ensures female comes before male at same ra
```

Explanation: Alternating Male-Female with Window Functions

Objective:

- Alternate male and female employees.
- Females: youngest → oldest (`dob DESC`)
- Males: oldest → youngest (`dob ASC`)
- Start with female.

Step 1: Assign Row Numbers

```
-- For Males (oldest first)
ROW_NUMBER() OVER (ORDER BY dob ASC) AS rn_m

-- For Females (youngest first)
ROW_NUMBER() OVER (ORDER BY dob DESC) AS rn_f
```

Step 2: Combine Using UNION ALL

```
SELECT rn_f AS rn, ... FROM female_cte
UNION ALL
SELECT rn_m AS rn, ... FROM male_cte
```

- Align males and females by row number (rn).

Step 3: Final Output

```
ORDER BY rn, gender DESC
Orders by row number (rn)
```

- For each row number, FEMALE comes first due to gender DESC

---

20. Find customers who have not placed any orders but have a shipping status marked as "Pending". Show their names and shipping status.

```
SELECT c.first_name, c.last_name, s.status
FROM Customers c
JOIN Shipping s ON c.customer_id = s.customer_id
LEFT JOIN Orders o ON c.customer_id = o.customer_id
WHERE o.customer_id IS NULL
  AND s.status = 'Pending';
```

Thanks for Downloading