Tech Mahindra SQL interview questions for a Data Engineer (3-5 years of experience.)

```
-- 1. Find the top 3 cities with the highest sales per month. (sales_table):
drop table ##sales
Create table ##sales
(sale_id int, city varchar (50),
sale_date date, amount int)
insert into ##sales values
(1,'Mumbai','2024-01-10','5000'),
(2,'Delhi','2024-01-15','7000'),
(3,'Bangalore','2024-01-20','10000'),
(4,'Mangalore','2024-01-20','12000'),
(5,'Chennai','2024-02-05','3000'),
(6,'Mumbai','2024-02-08','4000'),
(7,'Patna','2024-02-08','5000'),
(7,'Mumbai','2024-03-08','5000')

WITH cte
    AS (SELECT city,
            Format(sale_date, 'yyyy- MM')                    sale_month,
            Sum(amount)                amt,
            Row_number()
             OVER(
               partition BY Format(sale_date, 'yyyy-MM')
               ORDER BY Sum(amount) DESC) AS rn
       FROM   ##sales
       GROUP  BY city,
              Format(sale_date, 'yyyy-MM'))
SELECT *
FROM   cte
WHERE  rn <= 3
```

--2. Write an SQL query to calculate the running total of sales for each city. (sales_data):

```
drop table ##sales
Create table ##sales
(sale_id int, city varchar (50),
sale_date date, amount int)
insert into ##sales values
(1,'Mumbai','2024-01-10','5000'),
(2,'Delhi ','2024-01-15','7000'),
(3,'Mumbai','2024-01-20','3000'),
(4,'Delhi ','2024-02-05','6000'),
(5,'Mumbai','2024-02-08','8000')

SELECT *,
     Sum(amount)
      OVER (
        partition BY city
        ORDER BY sale_date) calculatetotal
FROM   ##sales
```

```
--3. Find the second highest salary of employees. (employees):
drop table ##employees
create table ##employees
(emp_id int, emp_name varchar (50),
salary int, department varchar (50))
insert into ##employees values
(1,'Ravi','70000','HR'),
(2,'Priya','90000','IT'),
(3,'Kunal','85000','Finance'),
(4,'Aisha','60000','IT'),
(5,'Rahul','95000','HR')
---method 1

WITH cte
   AS (SELECT *,
          Dense_rank()
           OVER(
                ORDER BY salary
DESC) rn
      FROM   ##employees)
SELECT *
FROM   cte
WHERE  rn = 2


---method 2
SELECT TOP 1 *
FROM   ##employees
WHERE  salary < (SELECT Max(salary)
           FROM   ##employees)
ORDER  BY salary DESC
```

```
---method 3
WITH cte
    AS (SELECT TOP 2 *
       FROM   ##employees
       ORDER  BY salary DESC)
SELECT TOP 1 *
FROM   cte
ORDER  BY salary ASC;
---method 4 sub Query
SELECT *
FROM   (SELECT *,
           Dense_rank()
            OVER(
              ORDER BY salary DESC) AS rn
      FROM   ##employees) aa
WHERE  rn = 2
```

```sql
-- 4. Find employees who have the same salary as someone in the same department. (employee_salary):
drop table ##employees
create table ##employees(emp_id int,
emp_name varchar (50),
salary int, department varchar (50))
insert into ##employees values
(1,'Neha','50000','HR'),
(2,'Ravi','70000','IT'),
(3,'Aman','50000','HR'),
(4,'Pooja','90000','IT'),
(5,'Karan','70000','IT')
--Method 1
WITH cte
    AS (SELECT *,
            Dense_rank()
             OVER (
               partition BY department
               ORDER BY salary)rn
       FROM   ##employees)
SELECT *
FROM   cte
WHERE  rn = 1
---Method 2
SELECT e1.*
FROM   ##employees e1
      JOIN ##employees e2
        ON e1.department = e2.department
          AND e1.salary = e2.salary
          AND e1.emp_id <> e2.emp_id
ORDER  BY e1.department,
          e1.salary,
          e1.emp_id;
```

```sql
--5. Write an SQL query to find duplicate records in a table. (users):
create table ##users(users_id int,
users_name varchar (10),
email varchar (50))
insert into ##users values
(1,'Sameer','sameer@gmail.com'),
(2,'Anjali','anjali@gmail.com'),
(3,'Sameer','sameer@gmail.com'),
(4,'Rohan','rohan@gmail.com'),
(5,'Rohan','rohan@gmail.com')
WITH cte
    AS (SELECT *,
            Row_number()
              OVER(
                partition BY users_name, email
                ORDER BY users_name, email) as rn
       FROM   ##users)
--select * from cte where rn>1
--delete from cte where rn>1
SELECT *
FROM   cte
WHERE  rn = 1
```

-- 6. Write an SQL query to delete duplicate rows while keeping onlyone unique record. (Same sample data as Question 5)

```
WITH cte
    AS (SELECT *,
            Row_number()
             OVER(
               partition BY users_name, email
               ORDER BY users_name, email) as rn
       FROM   ##users)
--select * from cte where rn>1
--delete from cte where rn>1
DELETE FROM cte
WHERE  rn > 1
```

```
-- 7. Write an SQL query to pivot a table by months. Sample Data (sales_data):
Create table ##sales_data
(sale_id int, city varchar (50), sale_date date, amount int)

insert into ##sales_data values
(1,'Mumbai','2024-01-10','5000'),    (2,'Delhi    ','2024-02-15','7000'),
(3,'Mumbai','2024-01-20','3000'),    (4,'Delhi    ','2024-03-05','6000'),
(5,'Mumbai','2024-02-08','8000')


WITH cte
   AS (SELECT *,
           CONVERT (VARCHAR (3), Datename(month, sale_date)) AS mon
      FROM   ##sales_data)
SELECT city,
     Sum([jan]) AS [jan],
     Sum([feb]) AS [Feb],
     Sum([mar]) AS [Mar]
FROM   cte
     PIVOT (Sum(amount)
         FOR mon IN([jan],
                [Feb],
                [Mar])) AS pvt
GROUP  BY city
```

```
--8. Find customers who placed at least 3 orders in the last 6 months.
Sample Data (orders):
Create table ##orders(order_id int, customer_id int, order_date date,
amount int)
insert into ##orders values
(1,101,'2024-01-10',1000),
(2,102,'2024-02-15',2000),
(3,101,'2024-03-20',1500),
(4,103,'2024-04-05',2500),
(5,101,'2024-05-08',3000)


SELECT *
FROM   (SELECT *,
           Count(1)
            OVER(
              ORDER BY customer_id)RNK
       FROM   ##orders
       WHERE  order_date >= Dateadd(month, -6, Getdate()))AA
WHERE  rnk = 3
```

----9. NORMALIZATION VS. DENORMALIZATION – WHAT ARE THEY, AND WHEN SHOULD EACH BE USED IN A DATA PIPELINE?

Feature	Normalization (OLTP)	Denormalization (OLAP)
Goal	Reduce redundancy and ensure data integrity	Improve read and query performance
Joins	More joins (complex queries)	Fewer joins (faster queries)
Storage	Less storage required	More storage due to redundancy
Use Case	Transactional systems (Banking, E-commerce)	Analytical systems (Data Warehouses, Reporting)
Update Speed	Faster updates (less redundant data)	Slower updates (multiple copies of data)
Query Performance	Slower (due to joins)	Faster (pre-aggregated or redundant data)

10. Indexing in SQL – Clustered vs. Non-Clustered Indexes & Their Impact on Performance

1. Clustered Index

👉 Definition

A clustered index determines the physical order of data in a table.

- Table data is stored in the order of the clustered index key
- Only one clustered index per table (because data can be ordered only once)

```
CREATE CLUSTERED INDEX idx_orders_orderdate
ON Orders(OrderDate);
```

👉 **How it works**

- Leaf nodes of the index contain the actual table data
- Query directly navigates to the data pages

👉 **Performance Impact**

✅ **Very fast for:**

- Range queries (BETWEEN, <, >)
- Sorting (ORDER BY)
- Queries that return large result sets

❌ **Slower for:**

- Frequent INSERT, UPDATE, DELETE on indexed columns (page splits)
- Random key values (e.g., GUIDs)

2. Non-Clustered Index:

Does not change the physical order of data in the table. It is stored separately from the table and contains index keys along with pointers (row locators) to the actual data. A table can have multiple non-clustered indexes.

```
CREATE NONCLUSTERED INDEX idx_orders_customer
ON Orders(CustomerID);
```

👉 **How it works**

- Leaf nodes store row locators:
  - Clustered index key (if table has one)
  - Row ID (RID) if heap table

👉 **Performance Impact**

✅ **Very fast for:**

- Point lookups (WHERE CustomerID = 101)
- Highly selective queries
- Covering queries (with INCLUDE columns)

❌ **Slower when:**

- Many lookups required (Key Lookup)
- Large result sets are returned

3. Key Differences (Quick Comparison)

Feature	Clustered Index	Non-Clustered Index
Physical order of data	Yes	No
Number per table	One	Multiple
Leaf nodes contain	Actual data	Pointers to data
Best for	Range scans, sorting	Point lookups, filters
Insert/Update cost	Higher	Lower

4. Impact on Query Performance

SELECT Queries

- Clustered index → Faster scans & range queries
- Non-clustered index → Faster selective lookups

JOIN Operations

- Indexed join columns significantly reduce I/O
- Clustered index on PK, non-clustered on FK is common

DML Operations

- Each index adds overhead on INSERT/UPDATE/DELETE
- Over-indexing hurts write performance

5. Real-World Best Practices (Interview-Friendly)

✓ Use clustered index on:

- Primary key
- Frequently sorted or range-filtered columns

✓ Use non-clustered index on:

- WHERE, JOIN, GROUP BY columns
- High-selectivity fields

✓ Use covering indexes:

```
CREATE NONCLUSTERED INDEX idx_orders_customer
ON Orders(CustomerID)
INCLUDE (OrderDate, Amount);
```

✓ Avoid:

- Indexing low-cardinality columns (e.g., gender)
- Too many indexes on write-heavy tables