TCS DATA ANALYST INTERVIEW QUESTIONS
0-3 YOE
0-8 lpa

1. Write a query to find the second highest salary from an employee table.

Assumingyourtable is namedEmployeesand ithasacolumncalledsalary:

```
SELECT MAX(salary) AS SecondHighestSalary
FROM Employees
WHERE salary < (
    SELECT MAX(salary) FROM Employees
);
```

Alternative using LIMIT / OFFSET (for MySQL/PostgreSQL):

```
SELECT DISTINCT salary
FROM Employees
ORDER BY salary DESC
LIMIT 1 OFFSET 1;
```

---

2. How would you use a JOIN to combine data from two tables: one with employee information and another with department information?

Assumethefollowingstructure:

- Employees(emp_id, emp_name, dept_id, salary)
- Departments(dept_id, dept_name)

You can use an **INNER JOIN** to combine them:

```
SELECT e.emp_id, e.emp_name, e.salary, d.dept_name
FROM Employees e
INNER JOIN Departments d ON e.dept_id = d.dept_id;
```

This query will return only those employees who are assigned to a department.

---

3. Write a query to retrieve employees who joined in the last 30 days.

Assuming theEmployeestablehasacolumnjoin_dateoftypeDATE:

```
SELECT *
FROM Employees
WHERE join_date >= CURRENT_DATE - INTERVAL 30 DAY;
```

This works in **MySQL**.

For **SQL Server**, use:

```
SELECT *
FROM Employees
```

swipe

WHERE join_date >= DATEADD(DAY, -30, GETDATE()); For **PostgreSQL**, use: SELECT * FROM Employees WHERE join_date >= CURRENT_DATE - INTERVAL '30 days';

---

4. How can you handle NULL values in SQL when performing calculations?

You can use the COALESCE() or ISNULL() functions to replace NULL values with a default value (typically 0 for numeric operations):

- **Using COALESCE() (standard SQL):**

```
SELECT emp_id, salary, bonus,
    COALESCE(salary, 0) + COALESCE(bonus, 0) AS total_pay
FROM Employees;
```

- **In SQL Server, you can also use ISNULL():**

```
SELECT emp_id, salary, bonus,
    ISNULL(salary, 0) + ISNULL(bonus, 0) AS total_pay
FROM Employees; This avoids issues where arithmetic with NULL results in NULL.
```

5.ExplainthedifferencebetweenWHEREandHAVING clauses in SQL

Clause Use Case

WHERE Filters **rows** before grouping; used with individual column conditions

HAVING Filters **groups** after aggregation; used with aggregate functions

Example:

```
-- Filters rows with salary > 30000 before grouping
SELECT dept_id, COUNT(*) AS emp_count
FROM Employees
WHERE salary > 30000
GROUP BY dept_id
HAVING COUNT(*) > 5; -- Filters groups where count > 5
```

6.Writeaquerytofindthecountofemployeesineach department

Assuming the table Employees(dept_id, emp_id, emp_name, ...):

```
SELECT dept_id, COUNT(*) AS employee_count
FROM Employees
GROUP BY dept_id;
```

You can also join with the Departments table to get department names:

```
SELECT d.dept_name, COUNT(e.emp_id) AS employee_count
FROM Employees e
JOIN Departments d ON e.dept_id = d.dept_id
GROUP BY d.dept_name;
```

---

7. How would you optimize a slow-running query?

To optimize a slow query, you can:

- **Use indexes** on columns used in WHERE, JOIN, ORDER BY, and GROUP BY. **Avoid SELECT ***; only retrieve needed columns. **Use EXISTS instead of IN** for large subqueries. **Analyze execution plans** to identify bottlenecks. **Use joins efficiently** and avoid unnecessary ones. **Filter early** using WHERE before aggregation. **Partition large tables** for better data access. **Avoid functions on indexed columns** in WHERE clauses.

8.Writeaquerytoupdatethesalaryofall employees by 10%

Assuming the salary column exists in the Employees table:

```
UPDATE Employees
SET salary = salary * 1.10;
```

This increases each employee's salary by 10%.

---

9. Explain the purpose of indexes and how they improve query performance

- **Indexes** are special data structures (like B-trees) used to **speed up data retrieval**.
- They **reduce the number of rows** scanned during queries by pointing directly to the location of the desired data.

Example:

```
CREATE INDEX idx_emp_name ON Employees(emp_name);
```

Now, queries filtering by emp_name will execute faster.

Important Notes:

- Indexes improve **read performance**, but may slightly slow down **write operations** (INSERT,
- UPDATE, DELETE) due to index maintenance.
  Avoid indexing columns with high number of distinct values unless frequently queried.

---

10. Write a query to create a new table with columns for employee ID, name,and salary

You can use the CREATE TABLE statement to define a new table.

Basic Query:

```
CREATE TABLE Employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(100),
    salary DECIMAL(10, 2)
);
```

Explanation:

- emp_id INT PRIMARY KEY: Defines the employee ID as an integer and sets it as the unique identifier (primary key).
- emp_name VARCHAR(100): Stores the employee's name, allowing up to 100 characters.
  salary DECIMAL(10, 2): Stores salary values with up to 10 digits, including 2 after the decimal point (e.g., 12345678.90).

You can also create a new table using a SELECT statement:

```
CREATE TABLE HighEarners AS SELECT emp_id, emp_name, salary FROM ExistingEmployees WHERE salary > 50000;
```

---

11. How would you retrieve the top 5 highest-paid employees from an employee table?

There are different approaches based on the database system you use.

In MySQL / PostgreSQL (using LIMIT)

```
SELECT emp_id, emp_name, salary
FROM Employees
ORDER BY salary DESC
LIMIT 5;
```

- ORDER BY salary DESC: Sorts salaries from highest to lowest.
- LIMIT 5: Returns only the top 5 records.

In SQL Server (using TOP)

```
SELECT TOP 5 emp_id, emp_name, salary
FROM Employees
ORDER BY salary DESC;
```

- TOP 5: Limits output to 5 rows.
- ORDER BY salary DESC: Ensures the highest-paid employees come first.

In Oracle (using ROWNUM)

```
SELECT emp_id, emp_name, salary
FROM (
    SELECT emp_id, emp_name, salary
    FROM Employees
    ORDER BY salary DESC
)
WHERE ROWNUM <= 5;
```

Additional Insight:

- If salaries have duplicates and you want **ties included**, consider using DENSE_RANK():

```
SELECT emp_id, emp_name, salary
FROM (
    SELECT emp_id, emp_name, salary,
        DENSE_RANK() OVER (ORDER BY salary DESC) AS rank
    FROM Employees
) ranked
WHERE rank <= 5;
```

This ensures that if two employees tie for 2nd place, both are shown, and the next one is ranked 3rd.

---

12. Write a query to delete duplicate rows from a table based on a specific column

Let's assume you have a table called Employees with duplicate emp_email entries.

Option 1: Using ROW_NUMBER() (SQL Server, PostgreSQL, Oracle, etc.)

```
WITH RankedEmployees AS (
    SELECT *,
        ROW_NUMBER() OVER (PARTITION BY emp_email ORDER BY emp_id) AS rn
    FROM Employees
)
DELETE FROM Employees
WHERE emp_id IN (
    SELECT emp_id
    FROM RankedEmployees
    WHERE rn > 1
);
```

Explanation:

- PARTITION BY emp_email: Groups rows with the same email.
- ROW_NUMBER() assigns a unique number to each row within that group.
- Only the **first** row is kept (rn = 1); duplicates (rn > 1) are deleted.

Option 2: Using DELETE with NOT IN (MySQL-style workaround)

```
DELETE FROM Employees
WHERE emp_id NOT IN (
    SELECT MIN(emp_id)
    FROM Employees
    GROUP BY emp_email
);
```

Explanation:

- Keeps the row with the **lowest emp_id** for each emp_email.
- Deletes all others (duplicates).

---

13. Explain the difference between INNER JOIN, LEFT JOIN, RIGHT JOIN,and FULL OUTERJOIN

INNER JOIN

- Returns **only matching rows** from both tables.

```
SELECT *
FROM Employees e
INNER JOIN Departments d ON e.dept_id = d.dept_id;
```

- If there's no match, the row is **excluded**.

LEFT JOIN (or LEFT OUTER JOIN)

- Returns **all rows from the left table** and matching rows from the right table.
- If there's no match on the right side, you'll get NULLs.

```
SELECT *
FROM Employees e
LEFT JOIN Departments d ON e.dept_id = d.dept_id;
```

RIGHT JOIN (or RIGHT OUTER JOIN)

- Opposite of LEFT JOIN.
- Returns **all rows from the right table** and matching rows from the left.

```
SELECT *
FROM Employees e
```

RIGHT JOIN Departments d ON e.dept_id = d.dept_id;

FULL OUTER JOIN

- Returns **all rows** when there is a match in one of the tables.
- If no match exists on either side, it returns NULLs in place.

```
SELECT *
FROM Employees e
FULL OUTER JOIN Departments d ON e.dept_id = d.dept_id;
```

Summary Table:

JOIN Type	Rows Returned
INNER JOIN	Only matching rows from both tables
LEFT JOIN	All rows from the left table + matched from right
RIGHT JOIN	All rows from the right table + matched from left
FULL OUTER JOIN	All matched and unmatched rows from both tables

---

14. Write a query to calculate the average salary for each department

Assuming the table is Employees(emp_id, emp_name, dept_id, salary):

```
SELECT dept_id, AVG(salary) AS avg_salary
FROM Employees
GROUP BY dept_id;
```

Explanation:

- AVG(salary): Calculates the average salary.
- GROUP BY dept_id: Groups employees by department to compute the average per department.

To include department names (assuming a Departments table), use a join:

```
SELECT d.dept_name, AVG(e.salary) AS avg_salary
FROM Employees e
JOIN Departments d ON e.dept_id = d.dept_id
GROUP BY d.dept_name;
```

15.HowdoyouusetheCASEstatementinSQL?Providean example

The CASE statement is used for **conditional logic** in SQL, similar to if-else.

Syntax:

```
SELECT emp_name,
       salary,
       CASE
          WHEN salary >= 100000 THEN 'High'
          WHEN salary >= 50000 THEN 'Medium'
          ELSE 'Low'
       END AS salary_band
FROM Employees;
```

Explanation:

- The CASE evaluates conditions in order and returns the first match.
- Here, we categorize each employee's salary into **High**, **Medium**, or **Low**.

---

16. Write a query to find employees who have not been assigned to any department

Assume:

- Employees(dept_id) is a **foreign key** referencing Departments(dept_id).
- Employees without a department will have NULL in dept_id, or there might be no match in the Departments table.

Option 1: Using LEFT JOIN with IS NULL

```
SELECT e.*
FROM Employees e
LEFT JOIN Departments d ON e.dept_id = d.dept_id
WHERE d.dept_id IS NULL;
```

Explanation:

- LEFT JOIN returns all employees.
- WHERE d.dept_id IS NULL: filters employees **without a matching department**.

Option 2: Simple WHERE dept_id IS NULL (if NULLs are used for unassigned depts)

```
SELECT *
FROM Employees
WHERE dept_id IS NULL;
```

Use this only if unassigned departments are truly NULL in the Employees table.

---

17. Explain the concept of a Primary Key and a Foreign Key in SQL

PrimaryKey:

- A **Primary Key** uniquely identifies each record in a table.
- It **cannot be NULL** and must be **unique** across all rows.
- A table can have **only one** primary key (which can consist of one or more columns).

Example:

```
CREATE TABLE Employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(100),
    salary DECIMAL(10,2)
);
```

- emp_id is the primary key — no two employees can have the same ID, and it can't be NULL.

Foreign Key:

: A **Foreign Key** is a column (or set of columns) in one table that refers to the **primary key** in another table.
It creates a **relationship between two tables**, ensuring **referential integrity**.

Example:

```
CREATE TABLE Departments (
    dept_id INT PRIMARY KEY,
    dept_name VARCHAR(100)
);

CREATE TABLE Employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(100),
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
```

);

- dept_id in Employees is a **foreign key** referencing Departments(dept_id).
- This ensures that you cannot insert a dept_id into Employees unless it exists in Departments.

---

18. Write a query to add a new column to an existing table

Use the ALTER TABLE statement.

Example:

Add a new column email to the Employees table:

```
ALTER TABLE Employees
ADD email VARCHAR(100);
```

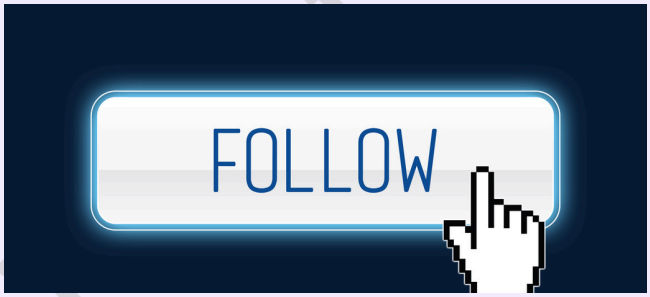
Explanation:

- ALTER TABLE Employees: Target the table you want to change.
- ADD email VARCHAR(100): Adds a new column email that can store up to 100 characters.

You can also set a default value:

```
ALTER TABLE Employees
ADD status VARCHAR(10) DEFAULT 'Active';
```

This adds a status column with a default value of 'Active' for new rows.



Comment "Yes" to get in depth pdf