Day 54 – SQL Interview Case Studies & Real-World Scenarios (Part 4)

🔹 Section 15: Query Scenarios

Q78. How do you find the second highest salary in a table?

```
SELECT MAX(salary)
FROM employees
WHERE salary < (SELECT MAX(salary) FROM employees);
```

💡 Tip: Mention both methods → using subquery and ROW_NUMBER().

Q79. How do you retrieve duplicate rows from a table?

```
SELECT name, COUNT(*)
FROM students
GROUP BY name
HAVING COUNT(*) > 1;
```

💡 Tip: GROUP BY + HAVING is the standard approach.

Q80. How do you delete duplicate rows but keep one copy?

```
DELETE FROM students
WHERE id NOT IN (
  SELECT MIN(id)
  FROM students
  GROUP BY name
);
```

💡 Tip: Always clarify which row to keep (MIN, MAX, etc.).

Q81. How do you find employees who do not belong to any department?

```
SELECT e.name
FROM employees e
LEFT JOIN departments d ON e.dept_id = d.id
WHERE d.id IS NULL;
```

💡 Tip: LEFT JOIN + IS NULL is the best pattern here.

Q82. How do you find the nth highest salary?

```
SELECT salary
FROM (
  SELECT salary, ROW_NUMBER() OVER (ORDER BY salary DESC) AS rn
  FROM employees
) t
WHERE rn = 3;
```

💡 Tip: Always mention ROW_NUMBER() vs DENSE_RANK().

🔹 Section 16: Problem-Solving with Joins & Aggregates

Q83. Find departments with more than 5 employees.

```
SELECT dept_id, COUNT(*)
FROM employees
GROUP BY dept_id
HAVING COUNT(*) > 5;
```

💡 Tip: GROUP BY + HAVING is critical for aggregates.

Q84. Retrieve students enrolled in more than 3 courses.

```
SELECT student_id, COUNT(course_id)
FROM enrollments
GROUP BY student_id
HAVING COUNT(course_id) > 3;
```

Q85. Find employees with salaries above their department average.

```
SELECT e.name, e.salary, e.dept_id
FROM employees e
WHERE e.salary > (
  SELECT AVG(salary)
  FROM employees
  WHERE dept_id = e.dept_id
);
```

💡 Tip: This is a classic **correlated subquery**.

Q86. List customers who placed orders but never made a payment.

```
SELECT c.customer_name
FROM customers c
JOIN orders o ON c.id = o.customer_id
LEFT JOIN payments p ON o.id = p.order_id
WHERE p.id IS NULL;
```

🔹 Section 17: Date & Time Based Questions

Q87. Find employees who joined in the last 30 days.

```
SELECT *
FROM employees
WHERE join_date >= CURRENT_DATE - INTERVAL '30 days';
```

💡 Tip: Syntax may differ in **MySQL, PostgreSQL, Oracle**.

Q88. Calculate monthly sales totals.

```
SELECT DATE_TRUNC('month', order_date) AS month, SUM(amount)
FROM sales
GROUP BY DATE_TRUNC('month', order_date);
```

Q89. Get top 3 products sold each month.

```
SELECT product_id, month, total_sales
FROM (
  SELECT product_id,
        DATE_TRUNC('month', order_date) AS month,
        SUM(amount) AS total_sales,
        RANK() OVER (
            PARTITION BY DATE_TRUNC('month', order_date)
            ORDER BY SUM(amount) DESC
        ) AS rnk
  FROM sales
  GROUP BY product_id, DATE_TRUNC('month', order_date)
) t
WHERE rnk <= 3;
```

💡 Tip: Sometimes they ask “top N per group” → Always use ROW_NUMBER() or RANK() with PARTITION BY.

🔹 Section 18: Complex Case Scenarios

Q90. Write a query to pivot rows into columns (Sales per Region).

```
SELECT region,
    SUM(CASE WHEN product = 'A' THEN sales ELSE 0 END) AS ProductA,
    SUM(CASE WHEN product = 'B' THEN sales ELSE 0 END) AS ProductB
FROM sales
GROUP BY region;
```

💡 Tip: Pivoting is often used in **reporting/analytics systems** to restructure data for dashboards.

Q91. Find customers who ordered consecutively for 3 days.

```
SELECT DISTINCT customer_id
FROM (
 SELECT customer_id, order_date,
     LAG(order_date,1) OVER (PARTITION BY customer_id ORDER BY order_date) AS prev_day,
     LAG(order_date,2) OVER (PARTITION BY customer_id ORDER BY order_date) AS prev2_day
 FROM orders
) t
WHERE order_date = prev_day + INTERVAL '1 day'
```

```
AND prev_day = prev2_day + INTERVAL '1 day';
```

- LAG(column, offset) returns the **previous row's value** without using a self-join.
- Example: LAG(order_date,1) → previous day's order date.
- LAG(order_date,2) → order date from 2 rows before.

💡 **Tip**:

- Use LAG() when you need to compare a row with **earlier rows** (like consecutive dates, trends, or patterns).
- Mention in interviews that **LEAD() is the opposite**, used to look forward.

Q92. Rank students by marks, breaking ties fairly.

```
SELECT name, marks,
    RANK() OVER (ORDER BY marks DESC) AS ranking
FROM students;
```

💡 Tip: RANK() leaves gaps, DENSE_RANK() doesn't.

Q93. Detect gaps in sequences (missing invoice numbers).

```
SELECT t1.invoice_no + 1 AS missing_no
FROM invoices t1
LEFT JOIN invoices t2 ON t1.invoice_no + 1 = t2.invoice_no
WHERE t2.invoice_no IS NULL;
```

💡 Tip: These are tricky real-world problems often asked.