Advanced JOIN Questions – Answers

---

1. Explain internal execution of JOINs.

- MySQL uses **nested loop join** by default, may use **hash join** or **merge join** depending on indexes and optimizer.
- Optimizer decides join order using **statistics**, **indexes**, and **table size**.

---

2. Emulate FULL OUTER JOIN in MySQL.

```
SELECT e.name, d.dept_name
FROM employees e
LEFT JOIN departments d ON e.dept_id = d.id
UNION
SELECT e.name, d.dept_name
FROM employees e
RIGHT JOIN departments d ON e.dept_id = d.id;
```

---

3. JOIN vs Subquery Performance

- JOIN: Usually **faster**, allows index usage, combines data in a single step.
- Subquery: Can be slower, executes separately for each row (depends on optimizer).
- Prefer JOIN when **combining tables**, subquery for **conditional existence checks**.

---

4. How indexes affect JOIN performance

- Indexes on join columns **improve performance**.
- Without indexes, MySQL may perform **full table scans** → slower queries.

---

5. Difference between ON, WHERE, HAVING

Clause	Purpose
ON	Join condition, determines which rows match
WHERE	Filters result after JOIN
HAVING	Filters **aggregated results** (after GROUP BY)

---

6. JOIN on non-key columns

- Possible, but may create **duplicates** or **slow performance** if not indexed.

---

7. Joining 3+ tables efficiently

- Join **smaller tables first**, or **tables with indexes on join columns**.
- Avoid joining **large tables without conditions** first.

---

8. Filtering on right table in LEFT JOIN

- Move filter from WHERE → ON to **retain unmatched left rows**.

SELECT *

FROM A

LEFT JOIN B ON A.id = B.id AND B.status = 'active';

---

9. Self Join for hierarchy

SELECT e1.name AS Employee, e2.name AS Manager

FROM employees e1

LEFT JOIN employees e2 ON e1.manager_id = e2.id;

---

10. Difference between queries

SELECT * FROM A LEFT JOIN B ON A.id = B.id WHERE B.id IS NOT NULL;

-- behaves exactly like INNER JOIN

SELECT * FROM A INNER JOIN B ON A.id = B.id;

✅ Output is identical.

---

11. Count students per class (include zero students)

SELECT c.class_name, COUNT(s.student_id) AS student_count

FROM classes c

LEFT JOIN students s ON c.class_id = s.class_id

GROUP BY c.class_name;

---

12. Employee → Manager

SELECT e.name AS Employee, m.name AS Manager

FROM employees e

LEFT JOIN employees m ON e.manager_id = m.id;

---

13. Customers with no orders

```sql
SELECT c.name

FROM customers c

LEFT JOIN orders o ON c.customer_id = o.customer_id

WHERE o.order_id IS NULL;
```

---

14. Departments with >5 employees

```sql
SELECT d.dept_name, COUNT(e.id) AS emp_count

FROM departments d

INNER JOIN employees e ON d.id = e.dept_id

GROUP BY d.dept_name

HAVING COUNT(e.id) > 5;
```

---

15. Sales vs Targets

```sql
SELECT t.region, SUM(s.amount) AS total_sales,

    CASE WHEN SUM(s.amount) >= t.target_amount THEN 'Met'

        ELSE 'Missed'

    END AS status

FROM targets t

LEFT JOIN sales s ON t.region = s.region

GROUP BY t.region, t.target_amount;
```

---

16. Projects with no employees

```sql
SELECT p.name

FROM projects p

LEFT JOIN employees e ON p.id = e.project_id

WHERE e.id IS NULL;
```

---

17. Show all products and categories (FULL OUTER JOIN simulation)

```sql
SELECT p.id AS product_id, c.name AS category_name

FROM products p

LEFT JOIN categories c ON p.category_id = c.id

UNION

SELECT p.id AS product_id, c.name AS category_name

FROM products p
```

```sql
RIGHT JOIN categories c ON p.category_id = c.id;
```

---

18. Top 3 customers by total order value

```sql
SELECT c.name, SUM(o.amount) AS total_order
FROM customers c
INNER JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.name
ORDER BY total_order DESC
LIMIT 3;
```

---

19. Students with mentor

```sql
SELECT s.name AS Student, m.name AS Mentor
FROM students s
LEFT JOIN students m ON s.mentor_id = m.id;
```

---

20. Price changes between old and new

```sql
SELECT o.product_id, o.price AS old_price, n.price AS new_price,
    CASE
      WHEN n.price > o.price THEN 'Increased'
      WHEN n.price < o.price THEN 'Decreased'
      ELSE 'Same'
    END AS change_type
FROM old_prices o
INNER JOIN new_prices n ON o.product_id = n.product_id;
```
