SQL OPTIMIZATION

6

INTERVIEW QUESTION

Deloitte. Infosys

🎯 Interview Question:

Q: When should you use JOINs and CTEs instead of subqueries in SQL?
Asked by: Deloitte, Infosys

☑ Answer:

• Use JOINs for better performance and maintainability, especially on large datasets.

• Correlated subqueries are executed once per row — making them inefficient as data size grows.

• JOINs and CTEs support parallel processing, are more readable, and scale well with indexed tables.

💡 Why it matters:

- In production databases (like Azure SQL, Snowflake, or PostgreSQL), correlated subqueries often cause significant performance bottlenecks.

- They trigger nested loop executions per row, increasing latency, memory usage, and query complexity — especially in analytical workloads.

- Using JOINs or CTEs helps databases optimize execution plans and utilize indexes or partitioning strategies.

🛠️ Optimization Tips:

- 🔹 Replace correlated subqueries with LEFT JOINs and GROUP BY

- 🔹 Use CTEs for clarity and modular query building

- 🔹 Leverage execution plans to validate performance improvements

- 🔹 Always check query cost before and after transformation

📌 Examples:

🚫 Slow Correlated Subquery:

```
SELECT name,
       (SELECT COUNT(*)
        FROM orders o
        WHERE o.customer_id = c.id)
FROM customers c;
```

⛔ Executes the subquery for **each row** in customers

☑ JOIN-based Rewrite:

```
SELECT c.name, COUNT(o.id)
FROM customers c
LEFT JOIN orders o ON c.id = o.customer_id
GROUP BY c.name;
```

✔ Processes data in a **single pass**, supports index scans and is highly scalable.

🔍 Best Practice:

• ☑ Default to JOINs when retrieving related data across tables

• ❌ Avoid correlated subqueries unless absolutely necessary

• ☑ Use CTEs for layering complex logic and improving readability

• ⚠ Watch out for duplicated rows — always validate aggregate logic when switching from subqueries to joins

🌍 Real-World Example:

You're querying customers with their total order count from a table of 10 million rows:

❌ Inefficient Query (Correlated Subquery):

```
SELECT name,
       (SELECT COUNT(*)
        FROM orders o
        WHERE o.customer_id = c.id)
FROM customers c;
```

⛔ Takes 3–4 minutes due to nested execution

☑ Optimized JOIN:

```
SELECT c.name, COUNT(o.id)
FROM customers c
LEFT JOIN orders o ON c.id = o.customer_id
GROUP BY c.name;
```

✔ Runs in seconds — joins leverage indexes and process rows in parallel

💥 Massive improvement on systems like Azure Synapse, BigQuery, and Snowflake

---