SQL Interview Questions for Experienced Candidates (3+ years)

Data Types & Miscellaneous Concepts

Normalization & Schema Design

Database Design Principles

Indexing Strategies & Keys

Mastering SQL Joins

1. What are the different types of SQL joins (`INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, `FULL JOIN`, `CROSS JOIN`, etc.) and when would you use each?

Answer:
SQL joins are used to combine rows from two or more tables based on related columns. The main types are:

- **INNER JOIN:** Returns only rows with matching values in both tables. Use when you need records present in both tables.
- **LEFT JOIN (LEFT OUTER JOIN):** Returns all rows from the left table and matched rows from the right table. Unmatched rows from the right table return NULLs. Use when you want all records from the left table, regardless of matches.
- **RIGHT JOIN (RIGHT OUTER JOIN):** Returns all rows from the right table and matched rows from the left table. Unmatched rows from the left table return NULLs. Use when you want all records from the right table.
- **FULL JOIN (FULL OUTER JOIN):** Returns all rows when there is a match in either table. Unmatched rows from either side return NULLs. Use when you want all records from both tables.
- **CROSS JOIN:** Returns the Cartesian product of both tables (every row of the first table joined with every row of the second). Use rarely, typically for generating combinations.

Example:

INNER JOIN:

```sql
SELECT a.*, b.*
FROM TableA a
INNER JOIN TableB b ON a.id = b.a_id;
```

LEFT JOIN:

```sql
SELECT a.*, b.*
FROM TableA a
LEFT JOIN TableB b ON a.id = b.a_id;
```

RIGHT JOIN:

```sql
SELECT a.*, b.*
FROM TableA a
RIGHT JOIN TableB b ON a.id = b.a_id;
```

FULL OUTER JOIN:

```sql
SELECT a.*, b.*
FROM TableA a
FULL OUTER JOIN TableB b ON a.id = b.a_id;
```

CROSS JOIN:

```sql
SELECT a.*, b.*
FROM TableA a
CROSS JOIN TableB b;
```

Explanation:
- Use **INNER JOIN** when you only want rows with matches in both tables.
- Use **LEFT JOIN** to get all rows from the left table, even if there are no matches in the right.
- Use **RIGHT JOIN** to get all rows from the right table, even if there are no matches in the left.
- Use **FULL OUTER JOIN** to get all rows from both tables, with NULLs where there are no matches.
- Use **CROSS JOIN** to get every combination of rows from both tables (rarely used in practice).

2. What is the difference between a `CROSS JOIN` and a `FULL OUTER JOIN`?

Answer:
`CROSS JOIN` and `FULL OUTER JOIN` are both used to combine rows from two tables, but they operate very differently:

Feature	CROSS JOIN	FULL OUTER JOIN
Purpose	Returns the Cartesian product of both tables (all possible combinations of rows).	Returns all rows from both tables, matching rows where possible, and filling with NULLs where there is no match.
Join Condition	No join condition is used.	Join condition is required (typically ON clause).

Feature	CROSS JOIN	FULL OUTER JOIN
Result Size	Number of rows = rows in TableA × rows in TableB.	Number of rows = all matched rows + unmatched rows from both tables.
Typical Use Case	Generating all possible combinations (e.g., scheduling, permutations).	Combining all data from both tables, showing matches and non-matches.

Example:

CROSS JOIN:

```
SELECT a.*, b.*
FROM TableA a
CROSS JOIN TableB b;
```

FULL OUTER JOIN:

```
SELECT a.*, b.*
FROM TableA a
FULL OUTER JOIN TableB b ON a.id = b.a_id;
```

Summary:
- **CROSS JOIN** creates every possible pair of rows from both tables.
- **FULL OUTER JOIN** returns all rows from both tables, matching where possible, and filling with NULLs where there is no match.

Answer:
To retrieve the first and last names of employees along with their managers' names, you typically use a **JOIN** between the `Employees` and `Managers` tables. The exact query depends on the schema, but here are two common scenarios:

Scenario 1: Separate `Employees` and `Managers` Tables

- `Employees` table: `EmployeeID`, `FirstName`, `LastName`, `ManagerID`
- `Managers` table: `ManagerID`, `FirstName`, `LastName`

Query:

```
SELECT
    e.FirstName AS EmployeeFirstName,
    e.LastName AS EmployeeLastName,
    m.FirstName AS ManagerFirstName,
    m.LastName AS ManagerLastName
FROM Employees e
LEFT JOIN Managers m ON e.ManagerID = m.ManagerID;
```

- This query uses a **LEFT JOIN** to include employees who may not have a manager (e.g., top-level managers).

Scenario 2: Self-Join on `Employees` Table (Manager is also an Employee)

- `Employees` table: `EmployeeID`, `FirstName`, `LastName`, `ManagerID`

Query:

```
SELECT
    e.FirstName AS EmployeeFirstName,
    e.LastName AS EmployeeLastName,
    m.FirstName AS ManagerFirstName,
    m.LastName AS ManagerLastName
FROM Employees e
LEFT JOIN Employees m ON e.ManagerID = m.EmployeeID;
```

- This **self-join** approach is common when managers are also listed as employees.

Explanation:

- The **LEFT JOIN** ensures all employees are listed, even if they do not have a manager.
- The manager's name is retrieved by joining the employee's `ManagerID` to the manager's `EmployeeID`.
- This pattern is useful for hierarchical data, such as organizational charts.

Answer:

4. Write a SQL query to find the average salary for each department, given tables `Employees` (with `DepartmentID`) and `Departments` (with `DepartmentName`).

To find the average salary for each department, you need to join the `Employees` and `Departments` tables on `DepartmentID`, then group by department.

Query:

```
SELECT
    d.DepartmentName,
    AVG(e.Salary) AS AverageSalary
FROM Employees e
INNER JOIN Departments d ON e.DepartmentID = d.DepartmentID
GROUP BY d.DepartmentName;
```

- **INNER JOIN** ensures only departments with employees are included.
- **AVG()** calculates the average salary per department.
- **GROUP BY** groups results by department name.

Answer:

5. Write a SQL query to list all products that have never been ordered (products in a `Product` table with no matching rows in the `Orders` table).

You can use a **LEFT JOIN** and filter for NULLs in the `Orders` table, or use `NOT EXISTS`.

Query using LEFT JOIN:

```
SELECT p.*
FROM Product p
LEFT JOIN Orders o ON p.ProductID = o.ProductID
WHERE o.ProductID IS NULL;
```

Query using NOT EXISTS:

```
SELECT p.*
FROM Product p
WHERE NOT EXISTS (
    SELECT 1 FROM Orders o WHERE o.ProductID = p.ProductID
);
```

- Both queries return products that do not appear in any order.

Answer:

6. Write a SQL query to list all employees who are also managers (for example, employees who appear as managers in the same table).

Assuming the `Employees` table has `EmployeeID` and `ManagerID` columns:

Query:

```
SELECT DISTINCT e.*
FROM Employees e
WHERE e.EmployeeID IN (
    SELECT DISTINCT ManagerID FROM Employees WHERE ManagerID IS NOT NULL
);
```

- This finds employees whose `EmployeeID` appears as a `ManagerID` for someone else.

Answer:

7. What is a `self-join`, and when might you use it? Provide an example scenario.

A **self-join** is a join where a table is joined to itself. This is useful for hierarchical or recursive relationships, such as employees and their managers.

Example Scenario:
Suppose you have an `Employees` table with `EmployeeID`, `Name`, and `ManagerID` (where `ManagerID` references another `EmployeeID`).

Query:

```sql
SELECT
    e.Name AS Employee,
    m.Name AS Manager
FROM Employees e
LEFT JOIN Employees m ON e.ManagerID = m.EmployeeID;
```

- This lists each employee with their manager's name.
- Useful for organizational charts, bill of materials, etc.

Answer:

8. How would you join more than two tables in a single SQL query? What factors affect the performance when joining multiple tables?

You can join multiple tables by chaining `JOIN` clauses:

Example:

```sql
SELECT e.Name, d.DepartmentName, l.LocationName
FROM Employees e
INNER JOIN Departments d ON e.DepartmentID = d.DepartmentID
INNER JOIN Locations l ON d.LocationID = l.LocationID;
```

Performance Factors:

Factor	Impact
Indexes	Proper indexes on join columns speed up joins.
Join Order	The order of joins can affect execution plans and performance.
Table Size	Large tables increase join cost; filtering early helps.
Join Type	OUTER JOINs are generally slower than INNER JOINs.
Query Complexity	More joins and conditions increase parsing and execution time.
Database Statistics	Up-to-date statistics help the optimizer choose efficient plans.

- Use **EXPLAIN** or query plans to analyze performance.
- Minimize the number of rows joined by filtering early (using `WHERE`).

Answer:

9. Explain how an `OUTER JOIN` works when one side has no matching rows. How does this differ from an `INNER JOIN` in practice?

Join Type	Rows Returned When No Match	NULLs in Result?	Use Case Example
INNER JOIN	Row is excluded	No	Only matching records from both tables
LEFT OUTER JOIN	Row from left table shown, right columns NULL	Yes	All left table rows, even if no match
RIGHT OUTER JOIN	Row from right table shown, left columns NULL	Yes	All right table rows, even if no match
FULL OUTER JOIN	Rows from both tables shown, unmatched side NULL	Yes	All rows from both tables

Example:

```sql
-- INNER JOIN: Only rows with matches
SELECT a.*, b.*
FROM A a
INNER JOIN B b ON a.id = b.a_id;

-- LEFT OUTER JOIN: All rows from A, NULLs for B if no match
SELECT a.*, b.*
FROM A a
LEFT JOIN B b ON a.id = b.a_id;
```

- **OUTER JOIN** includes unmatched rows from one or both tables, filling in NULLs for missing data.
- **INNER JOIN** only includes rows where a match exists in both tables.
- OUTER JOINs are useful for finding missing or unmatched data (e.g., customers with no orders).

Working with Views

Stored Procedures & Functions

Triggers & Automation

Transactions & Concurrency Control

Performance Tuning & Query Optimization