30 Expert-Level SQL JOIN Interview Questions & Answers

Q1. List each product with its category name.

```
SELECT p.ProductName, c.CategoryName
FROM products p
JOIN categories c ON p.CategoryID = c.CategoryID;
```

Explanation: Simple INNER JOIN ensuring product-category mapping.

Q2. Find all customers and their city, including customers without city data.

```
SELECT cu.CustomerID, cu.FirstName, ci.CityName
FROM customers cu
LEFT JOIN cities ci ON cu.CityID = ci.CityID;
```

Explanation: LEFT JOIN ensures customers without valid CityID still appear.

Q3. Get all employees and their city & country.

```
SELECT e.EmployeeID, e.FirstName, ci.CityName, co.CountryName
FROM employees e
JOIN cities ci ON e.CityID = ci.CityID
JOIN countries co ON ci.CountryID = co.CountryID;
```

Explanation: Demonstrates multi-table joins across location hierarchy.

Q4. Show orders with customer and employee handling the order.

```
SELECT o.OrderID, cu.FirstName AS Customer, e.FirstName AS Employee
FROM orders o
JOIN customers cu ON o.CustomerID = cu.CustomerID
JOIN employees e ON o.EmployeeID = e.EmployeeID;
```

Explanation: Classic join combining business (orders) with actors (customers & employees).

Q5. List products sold with order ID.

```
SELECT DISTINCT oi.OrderID, p.ProductName
FROM order_items oi
JOIN products p ON oi.ProductID = p.ProductID;
```

Explanation: JOIN between transaction line and product master.

Q6. Find revenue per product.

```
SELECT p.ProductName, SUM(oi.Quantity * oi.UnitPrice) AS Revenue
FROM order_items oi
JOIN products p ON oi.ProductID = p.ProductID
GROUP BY p.ProductName
ORDER BY Revenue DESC;
```

Explanation: Aggregate JOIN with GROUP BY.

Q7. Find the top 5 customers by total spend.

```
SELECT cu.CustomerID, cu.FirstName, SUM(oi.Quantity * oi.UnitPrice) AS TotalSpe
FROM customers cu
JOIN orders o ON cu.CustomerID = o.CustomerID
JOIN order_items oi ON o.OrderID = oi.OrderID
GROUP BY cu.CustomerID
ORDER BY TotalSpent DESC
LIMIT 5;
```

Explanation: Multi-join + ranking by SUM.

Q8. Show all customers who never placed an order.

```
SELECT cu.CustomerID, cu.FirstName
FROM customers cu
LEFT JOIN orders o ON cu.CustomerID = o.CustomerID
WHERE o.OrderID IS NULL;
```

Explanation: LEFT JOIN + NULL check for 'no match' scenario.

Q9. Show employees who handled no orders.

```
SELECT e.EmployeeID, e.FirstName
FROM employees e
LEFT JOIN orders o ON e.EmployeeID = o.EmployeeID
WHERE o.OrderID IS NULL;
```

Explanation: Testing understanding of anti-join logic.

Q10. Find products never sold.

```
SELECT p.ProductID, p.ProductName
FROM products p
LEFT JOIN order_items oi ON p.ProductID = oi.ProductID
WHERE oi.OrderID IS NULL;
```

Explanation: Outer join filtering NULL for 'unsold' products.

Q11. Show average order value per customer.

```
SELECT cu.CustomerID, cu.FirstName,
       AVG(order_totals.TotalAmount) AS AvgOrderValue
FROM customers cu
JOIN (
   SELECT o.OrderID, o.CustomerID, SUM(oi.Quantity * oi.UnitPrice) AS TotalAmou
   FROM orders o
   JOIN order_items oi ON o.OrderID = oi.OrderID
   GROUP BY o.OrderID
) order_totals ON cu.CustomerID = order_totals.CustomerID
GROUP BY cu.CustomerID;
```

Explanation: Derived table join, tests nesting ability.

Q12. List countries with total revenue.

```
SELECT co.CountryName, SUM(oi.Quantity * oi.UnitPrice) AS Revenue
FROM countries co
JOIN cities ci ON co.CountryID = ci.CountryID
JOIN orders o ON ci.CityID = o.CityID
JOIN order_items oi ON o.OrderID = oi.OrderID
GROUP BY co.CountryName;
```

Explanation: Joining through hierarchy (country → city → order).

Q13. Find category with highest revenue.

```
SELECT c.CategoryName, SUM(oi.Quantity * oi.UnitPrice) AS Revenue
FROM categories c
JOIN products p ON c.CategoryID = p.CategoryID
JOIN order_items oi ON p.ProductID = oi.ProductID
GROUP BY c.CategoryName
ORDER BY Revenue DESC
LIMIT 1;
```

Explanation: Testing GROUP BY with LIMIT.

Q14. Show top employee by sales value.

```
SELECT e.EmployeeID, e.FirstName, SUM(oi.Quantity * oi.UnitPrice) AS Sales
FROM employees e
JOIN orders o ON e.EmployeeID = o.EmployeeID
JOIN order_items oi ON o.OrderID = oi.OrderID
GROUP BY e.EmployeeID
ORDER BY Sales DESC
```

```
LIMIT 1;
```

Explanation: Combines employee performance analysis with joins.

Q15. List each order with #products.

```
SELECT o.OrderID, COUNT(oi.ProductID) AS NumProducts
FROM orders o
JOIN order_items oi ON o.OrderID = oi.OrderID
GROUP BY o.OrderID;
```

Explanation: JOIN + COUNT aggregate.

Q16. Find employees who handled orders in more than one country.

```
SELECT e.EmployeeID, e.FirstName, COUNT(DISTINCT co.CountryName) AS Countries
FROM employees e
JOIN orders o ON e.EmployeeID = o.EmployeeID
JOIN cities ci ON o.CityID = ci.CityID
JOIN countries co ON ci.CountryID = co.CountryID
GROUP BY e.EmployeeID
HAVING COUNT(DISTINCT co.CountryName) > 1;
```

Explanation: Multi-level JOIN + DISTINCT aggregation.

Q17. Show orders handled by both employee and customer in same city.

```
SELECT o.OrderID, cu.FirstName AS Customer, e.FirstName AS Employee, ci.CityName
FROM orders o
JOIN customers cu ON o.CustomerID = cu.CustomerID
JOIN employees e ON o.EmployeeID = e.EmployeeID
JOIN cities ci ON cu.CityID = ci.CityID AND e.CityID = ci.CityID;
```

Explanation: Demonstrates join condition across multiple roles.

Q18. List customers with same last name as an employee.

```
SELECT DISTINCT cu.CustomerID, cu.LastName
FROM customers cu
JOIN employees e ON cu.LastName = e.LastName;
```

Explanation: Non-key join condition (string join).

Q19. Find orders with total revenue above 500.

```
SELECT o.OrderID, SUM(oi.Quantity * oi.UnitPrice) AS Total
FROM orders o
```

```
JOIN order_items oi ON o.OrderID = oi.OrderID
GROUP BY o.OrderID
HAVING Total > 500;
```

Explanation: JOIN + HAVING filter.

Q20. Show revenue by city.

```
SELECT ci.CityName, SUM(oi.Quantity * oi.UnitPrice) AS Revenue
FROM cities ci
JOIN orders o ON ci.CityID = o.CityID
JOIN order_items oi ON o.OrderID = oi.OrderID
GROUP BY ci.CityName;
```

Explanation: Classic city-level aggregation.

Q21. Find duplicate customer names across different cities.

```
SELECT cu.FirstName, cu.LastName, COUNT(DISTINCT cu.CityID) AS Cities
FROM customers cu
GROUP BY cu.FirstName, cu.LastName
HAVING COUNT(DISTINCT cu.CityID) > 1;
```

Explanation: Tests understanding of GROUP BY uniqueness.

Q22. Show employees who handled >10 orders.

```
SELECT e.EmployeeID, e.FirstName, COUNT(o.OrderID) AS OrdersHandled
FROM employees e
JOIN orders o ON e.EmployeeID = o.EmployeeID
GROUP BY e.EmployeeID
HAVING OrdersHandled > 10;
```

Explanation: Join + HAVING filter.

Q23. List top 3 products per category by revenue.

```
SELECT CategoryName, ProductName, Revenue
FROM (
   SELECT c.CategoryName, p.ProductName,
          SUM(oi.Quantity * oi.UnitPrice) AS Revenue,
          ROW_NUMBER() OVER (PARTITION BY c.CategoryName ORDER BY SUM(oi.Quanti
   FROM categories c
   JOIN products p ON c.CategoryID = p.CategoryID
   JOIN order_items oi ON p.ProductID = oi.ProductID
   GROUP BY c.CategoryName, p.ProductName
) t
WHERE rn <= 3;
```

Explanation: Uses window function + JOIN.

Q24. Find customers who ordered same product more than once.

```
SELECT cu.CustomerID, cu.FirstName, p.ProductName, COUNT(*) AS TimesOrdered
FROM customers cu
JOIN orders o ON cu.CustomerID = o.CustomerID
JOIN order_items oi ON o.OrderID = oi.OrderID
JOIN products p ON oi.ProductID = p.ProductID
GROUP BY cu.CustomerID, p.ProductName
HAVING COUNT(*) > 1;
```

Explanation: JOIN + GROUP HAVING duplicates.

Q25. Show average order value per employee.

```
SELECT e.EmployeeID, e.FirstName, AVG(order_totals.TotalAmount) AS AvgOrder
FROM employees e
JOIN (
   SELECT o.OrderID, o.EmployeeID, SUM(oi.Quantity * oi.UnitPrice) AS TotalAmou
   FROM orders o
   JOIN order_items oi ON o.OrderID = oi.OrderID
   GROUP BY o.OrderID
) order_totals ON e.EmployeeID = order_totals.EmployeeID
GROUP BY e.EmployeeID;
```

Explanation: Derived table usage again.

Q26. List categories never sold in some countries.

```
SELECT DISTINCT c.CategoryName, co.CountryName
FROM categories c
CROSS JOIN countries co
WHERE NOT EXISTS (
   SELECT 1
   FROM products p
   JOIN order_items oi ON p.ProductID = oi.ProductID
   JOIN orders o ON oi.OrderID = o.OrderID
   JOIN cities ci ON o.CityID = ci.CityID
   WHERE p.CategoryID = c.CategoryID AND ci.CountryID = co.CountryID
);
```

Explanation: Cross join + NOT EXISTS pattern.

Q27. Find revenue split by employee and category.

```
SELECT e.EmployeeID, e.FirstName, c.CategoryName,
       SUM(oi.Quantity * oi.UnitPrice) AS Revenue
```

```
FROM employees e
JOIN orders o ON e.EmployeeID = o.EmployeeID
JOIN order_items oi ON o.OrderID = oi.OrderID
JOIN products p ON oi.ProductID = p.ProductID
JOIN categories c ON p.CategoryID = c.CategoryID
GROUP BY e.EmployeeID, c.CategoryName;
```

Explanation: Multi-key aggregation.

Q28. Show month-wise sales trend.

```
SELECT DATE_FORMAT(o.OrderDate, '%Y-%m') AS Month,
       SUM(oi.Quantity * oi.UnitPrice) AS Revenue
FROM orders o
JOIN order_items oi ON o.OrderID = oi.OrderID
GROUP BY DATE_FORMAT(o.OrderDate, '%Y-%m')
ORDER BY Month;
```

Explanation: Join + date function grouping.

Q29. Find customers who bought from all categories.

```
SELECT cu.CustomerID, cu.FirstName
FROM customers cu
JOIN orders o ON cu.CustomerID = o.CustomerID
JOIN order_items oi ON o.OrderID = oi.OrderID
JOIN products p ON oi.ProductID = p.ProductID
GROUP BY cu.CustomerID
HAVING COUNT(DISTINCT p.CategoryID) = (SELECT COUNT(*) FROM categories);
```

Explanation: Relational division with JOIN.

Q30. Compare sales handled by employees in same city vs outside.

```
SELECT e.EmployeeID, e.FirstName,
       SUM(CASE WHEN e.CityID = o.CityID THEN oi.Quantity * oi.UnitPrice ELSE 0
       SUM(CASE WHEN e.CityID <> o.CityID THEN oi.Quantity * oi.UnitPrice ELSE 
FROM employees e
JOIN orders o ON e.EmployeeID = o.EmployeeID
JOIN order_items oi ON o.OrderID = oi.OrderID
GROUP BY e.EmployeeID;
```

Explanation: Conditional aggregation with JOIN.