SQL Guide for Software QA’s

By-
REEHAN NIZAMI SHAZADA

Introduction

This document provides detailed explanations and SQL queries for commonly asked questions in **QA interviews**. These queries help in **data validation, database testing, and automation scripts** for QA professionals.

1. Find Employees Whose Salary Exceeds the Company's Average Salary

Query:

```
SELECT name, salary

FROM employees

WHERE salary > (SELECT AVG(salary) FROM employees);
```

Explanation:

- AVG(salary) calculates the **average salary**.
- The **subquery** retrieves the average salary.
- The WHERE salary > condition filters employees earning **above average**.

2. Retrieve Employees Who Work in the Same Department as 'John Doe'

Query:

```
SELECT name

FROM employees

WHERE department_id = (

    SELECT department_id FROM employees WHERE name = 'John Doe'
```

```
);
```

Explanation:

- The **subquery** gets 'John Doe''s department_id.
- The main query finds **all employees** in that department.

3. Find the Second Highest Salary Without Using MAX() Twice

Query:

```
SELECT MAX(salary) AS second_highest_salary

FROM employees

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

Explanation:

- The first MAX(salary) gets the **highest salary**.
- The WHERE salary < condition filters out the **highest salary**, leaving the **second highest**.

4. Find Customers Who Placed More Than 5 Orders

Query:

```
SELECT customer_id, COUNT(*) AS total_orders

FROM orders

GROUP BY customer_id

HAVING COUNT(*) > 5;
```

Explanation:

- COUNT(*) counts the **total orders** for each customer.
- HAVING COUNT(*) > 5 filters customers who **ordered more than 5 times**.

5. Count the Total Number of Orders for Each Customer

Query:

```
SELECT customer_id, COUNT(*) AS total_orders

FROM orders

GROUP BY customer_id;
```

Explanation:

- Groups by customer_id and counts **total orders per customer**.

6. Find Employees Who Joined in the Last 6 Months

Query:

```
SELECT * FROM employees

WHERE joining_date >= DATEADD(MONTH, -6, GETDATE());
```

Explanation:

- DATEADD(MONTH, -6, GETDATE()) calculates **6 months ago**.
- The query filters employees who **joined after that date**.

7. Calculate the Total Sales Amount for Each Product

Query:

```
SELECT product_id, SUM(price * quantity) AS total_sales

FROM orders

GROUP BY product_id;
```

Explanation:

- SUM(price * quantity) calculates **total sales per product**.
- GROUP BY product_id groups the data **by product**.

8. List Products That Have Never Been Sold

Query:

```
SELECT p.product_id, p.product_name

FROM products p

LEFT JOIN orders o ON p.product_id = o.product_id

WHERE o.product_id IS NULL;
```

Explanation:

- LEFT JOIN ensures **all products** are included.
- WHERE o.product_id IS NULL filters products **without orders**.

9. Remove Duplicate Rows from a Table

Query:

```
DELETE FROM employees

WHERE id NOT IN (

    SELECT MIN(id) FROM employees GROUP BY name, department, salary

);
```

Explanation:

- Finds the **earliest entry** for each name, department, salary.
- Deletes **all other duplicates**.

10. Find the Top 10 Customers Who Haven't Ordered in the Past Year

Query:

```
SELECT c.customer_id, c.name

FROM customers c

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

WHERE o.order_date IS NULL OR o.order_date < DATEADD(YEAR, -1, GETDATE())

LIMIT 10;
```

Explanation:

- LEFT JOIN finds **all customers, even those without orders**.
- o.order_date < DATEADD(YEAR, -1, GETDATE()) filters orders **older than a year**.
- LIMIT 10 selects the **top 10 customers**.

11. Find Employees with the Highest Salary in Each Department

Query:

```
SELECT department, name, salary

FROM employees e1
```

```
WHERE salary = (SELECT MAX(salary) FROM employees e2 WHERE e1.department =
e2.department);
```

Explanation:

- **Subquery** finds the **highest salary per department**.
- The main query selects the employee **matching that salary**.

12. Find Employees Who Have the Same Salary

Query:

```
SELECT e1.name, e1.salary

FROM employees e1

JOIN employees e2 ON e1.salary = e2.salary AND e1.id != e2.id;
```

Explanation:

- JOIN pairs **employees with the same salary**.
- e1.id != e2.id ensures **they are different people**.

13. Find Customers Who Ordered a Specific Product

Query:

```
SELECT DISTINCT c.name

FROM customers c

JOIN orders o ON c.customer_id = o.customer_id

WHERE o.product_id = 101;
```

Explanation:

- JOIN links customers with orders.
- WHERE o.product_id = 101 filters for **a specific product**.

14. Find Orders Placed on Weekends

Query:

```
SELECT * FROM orders

WHERE DATEPART(WEEKDAY, order_date) IN (1, 7);
```

Explanation:

- DATEPART(WEEKDAY, order_date) extracts **day of the week**.
- 1 (Sunday) and 7 (Saturday) filter **weekend orders**.

Here's a **super detailed guide** explaining each SQL query **step by step**, as if we're explaining to someone completely new to SQL. 🎯

📌 1. Find Employees with Salaries Higher than the Company's Average

📝 Question:

Write a query to find all employees whose salaries exceed the company's average salary.

🧠 Explanation:

Imagine a classroom where we calculate the **average score** of all students. Now, we want to **list all students** whose scores are **above the class average**.

In a company, **salaries** work the same way! We need:

1. Find the **average salary** of all employees.
2. Get all employees **whose salary is greater than this average**.

👨‍💻 Query:

```
SELECT *

FROM employees

WHERE salary > (SELECT AVG(salary) FROM employees);
```

🔍 Breakdown:

- AVG(salary) → Finds the **average salary** of all employees.
- SELECT * FROM employees → Selects all employees.
- WHERE salary > (SELECT AVG(salary) FROM employees) → Filters only those with **higher than average salaries**.

📌 2. Find Employees in the Same Department as ‘John Doe’

📝 Question:

Write a query to retrieve the names of employees who work in the same department as 'John Doe'.

🧠 Explanation:

Imagine a **school** where students belong to **different classes**. If we want to find **who else is in the same class as ‘John Doe’**, we need:

1. Find **which class** John Doe is in.
2. List **everyone else** in that same class.

In a company, **departments** are like classes! 🎯

👨‍💻 Query:

```
SELECT name

FROM employees

WHERE department_id = (SELECT department_id FROM employees WHERE name = 'John Doe')

AND name <> 'John Doe';
```

🔍 Breakdown:

- SELECT department_id FROM employees WHERE name = 'John Doe' → Finds **John’s department**.
- WHERE department_id = (...) → Finds **everyone** in that department.
- AND name <> 'John Doe' → **Excludes John Doe** from the list.

📌 3. Find the Second Highest Salary (Without Using MAX Twice)

📝 Question:

Write a query to display the second-highest salary from the Employee table **without using MAX twice**.

🧠 Explanation:

If we have **a list of salaries**:
👉 5000, 7000, 10000, 12000, 7000

1. The **highest** salary is 12000.
2. The **second highest** is 10000.

To get this, we:

- **Sort salaries** in descending order.
- **Skip the highest one** and pick the next highest.

👨‍💻 Query:

```
SELECT MAX(salary)

FROM employees

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

🔍 Breakdown:

- MAX(salary) FROM employees → Finds the **highest salary**.
- WHERE salary < (...) → Picks the **second-highest salary**.

📌 4. Find Customers Who Placed More Than Five Orders

📝 Question:

Write a query to find all customers who have placed more than five orders.

🧠 Explanation:

Think of a **coffee shop loyalty card** ☕. If a customer buys coffee **more than 5 times**, they get a free one! We need:

1. **Count the number of orders** for each customer.
2. **Filter customers** where the count is **greater than 5**.

👨‍💻 Query:

```
SELECT customer_id
```

```
FROM orders

GROUP BY customer_id

HAVING COUNT(order_id) > 5;
```

🔍 Breakdown:

- GROUP BY customer_id → Groups orders **by customer**.
- COUNT(order_id) → Counts how many **orders** each customer placed.
- HAVING COUNT(order_id) > 5 → Selects **only customers with more than 5 orders**.

📌 5. Count Total Orders for Each Customer

📝 Question:

Write a query to count the total number of orders placed by each customer.

🧠 Explanation:

Imagine a **bakery** 🥖 where customers order **multiple times**. We need:

1. Count **how many times** each customer ordered.

👨‍💻 Query:

```
SELECT customer_id, COUNT(order_id) AS total_orders

FROM orders

GROUP BY customer_id;
```

🔍 Breakdown:

- COUNT(order_id) → Counts the **total orders per customer**.
- GROUP BY customer_id → Groups by **customer**.

📌 6. Find Employees Who Joined in the Last 6 Months

📝 Question:

Write a query to list employees who joined the company within the last 6 months.

👨‍💻 Query:

```
SELECT *

FROM employees

WHERE join_date >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH);
```

🔍 Breakdown:

- CURDATE() → Gets today's **date**.
- DATE_SUB(..., INTERVAL 6 MONTH) → **Subtracts 6 months** from today.
- WHERE join_date >= ... → Picks employees who **joined after that date**.

📌 7. Calculate Total Sales per Product

📝 Question:

Write a query to calculate the total sales amount for each product.

👨‍💻 Query:

```
SELECT product_id, SUM(quantity * price) AS total_sales

FROM orders

GROUP BY product_id;
```

🔍 Breakdown:

- SUM(quantity * price) → Adds up the total sales **for each product**.
- GROUP BY product_id → Groups by **product**.

📌 8. List Products That Have Never Been Sold

📝 Question:

Write a query to list all products that have **never** been sold.

👨‍💻 Query:

```
SELECT *

FROM products
```

```
WHERE product_id NOT IN (SELECT DISTINCT product_id FROM orders);
```

🔍 Breakdown:

- SELECT DISTINCT product_id FROM orders → Gets a list of **sold products**.
- NOT IN (...) → Selects products that are **not in that list**.

📌 9. Remove Duplicate Rows from a Table

📝 Question:

Write a query to **remove duplicate rows** from a table.

👨‍💻 Query:

```
DELETE FROM employees

WHERE id NOT IN (

  SELECT MIN(id) FROM employees GROUP BY name, department, salary

);
```

🔍 Breakdown:

- GROUP BY name, department, salary → Groups duplicates.
- MIN(id) → Keeps only the **first occurrence**.
- DELETE WHERE id NOT IN (...) → Removes duplicates.

📌 10. Find the Top 10 Customers Who Haven't Ordered in the Past Year

📝 Question:

Write a query to find the **top 10 customers** who have **not placed any orders** in the past year.

👨‍💻 Query:

```
SELECT customer_id

FROM customers

WHERE customer_id NOT IN (
```

```
 SELECT DISTINCT customer_id FROM orders WHERE order_date >=
DATE_SUB(CURDATE(), INTERVAL 1 YEAR)

)

LIMIT 10;
```

🔍 Breakdown:

- DATE_SUB(CURDATE(), INTERVAL 1 YEAR) → **One year ago**.
- WHERE order_date >= ... → Finds customers **who ordered in the past year**.
- NOT IN (...) → **Excludes** those who ordered.
- LIMIT 10 → Gets **only the top 10 customers**.

📌 11. Find the Third Highest Salary Without Using LIMIT

📝 Question:

Write a query to find the **third highest salary** in the Employee table **without using LIMIT**.

👨‍💻 Query:

```
SELECT DISTINCT salary

FROM employees e1

WHERE 2 = (SELECT COUNT(DISTINCT salary) FROM employees e2 WHERE e2.salary >
e1.salary);
```

🔍 Breakdown:

- The **highest salary** has **0** salaries greater than it.
- The **second highest** has **1** salary greater than it.
- The **third highest** has **2** salaries greater than it.
- COUNT(DISTINCT salary) = 2 → Finds the **third highest** salary.

📌 12. Find Employees Who Earn More Than Their Manager

📝 Question:

Write a query to find all employees who earn more than their managers.

👨‍💻 Query:

```
SELECT e.name AS employee, e.salary, m.name AS manager, m.salary AS manager_salary
FROM employees e
JOIN employees m ON e.manager_id = m.id
WHERE e.salary > m.salary;
```

🔍 Breakdown:

- JOIN employees m ON e.manager_id = m.id → Joins employees with their **managers**.
- WHERE e.salary > m.salary → Filters **employees earning more than their manager**.

📌 13. Find Employees Whose Names Start with ‘A’

📝 Question:

Write a query to find employees whose names start with ‘A’.

👨‍💻 Query:

```
SELECT * FROM employees WHERE name LIKE 'A%';
```

🔍 Breakdown:

- LIKE 'A%' → Finds names **starting** with ‘A’.

📌 14. Find Customers Who Have Placed Orders but Never Bought ‘Product X’

📝 Question:

Write a query to find customers who have **placed orders** but **never bought ‘Product X’**.

👨‍💻 Query:

```
SELECT customer_id
```

```
FROM orders
WHERE customer_id NOT IN (
  SELECT DISTINCT customer_id FROM orders WHERE product_id = 'X'
);
```

🔍 Breakdown:

- Finds all customers **who ordered**.
- Excludes those who **bought Product X**.

📌 15. Find the Most Frequently Purchased Product

📝 Question:

Write a query to find the **most frequently purchased product**.

👨‍💻 Query:

```
SELECT product_id, COUNT(*) AS total_purchases
FROM orders
GROUP BY product_id
ORDER BY total_purchases DESC
LIMIT 1;
```

🔍 Breakdown:

- COUNT(*) → Counts how many times each product was purchased.
- ORDER BY total_purchases DESC → Sorts by **highest** count.
- LIMIT 1 → Picks **the most popular product**.

📌 16. Find Employees Who Have the Same Salary as Someone Else

📝 Question:

Write a query to find employees **who have the same salary as another employee**.

👩‍💻 Query:

```
SELECT name, salary

FROM employees

WHERE salary IN (SELECT salary FROM employees GROUP BY salary HAVING COUNT(*) > 1);
```

🔍 Breakdown:

- Groups salaries and finds **duplicates** (HAVING COUNT(*) > 1).
- Filters employees with those **duplicate salaries**.

📌 17. Find Departments with More Than 10 Employees

📝 Question:

Write a query to list **departments with more than 10 employees**.

👩‍💻 Query:

```
SELECT department_id, COUNT(*) AS employee_count

FROM employees

GROUP BY department_id

HAVING COUNT(*) > 10;
```

🔍 Breakdown:

- GROUP BY department_id → Groups employees **by department**.
- HAVING COUNT(*) > 10 → Filters departments with **more than 10 employees**.

📌 18. Find Employees Who Work in Multiple Departments

📝 Question:

Write a query to find employees **who work in multiple departments**.

👨‍💻 Query:

```
SELECT employee_id

FROM employee_department

GROUP BY employee_id

HAVING COUNT(department_id) > 1;
```

🔍 Breakdown:

- Groups **by employee**.
- Filters employees assigned to **more than 1 department**.

📌 19. Find Customers Who Have Never Placed an Order

📝 Question:

Write a query to find customers who **never placed an order**.

👨‍💻 Query:

```
SELECT *

FROM customers

WHERE customer_id NOT IN (SELECT DISTINCT customer_id FROM orders);
```

🔍 Breakdown:

- Gets a list of **all customers**.
- Excludes those **who placed an order**.

📌 20. Find the Oldest Employee in Each Department

📝 Question:

Write a query to find the **oldest employee** in each department.

👨‍💻 Query:

```
SELECT department_id, name, age

FROM employees e1

WHERE age = (SELECT MAX(age) FROM employees e2 WHERE e1.department_id = e2.department_id);
```

🔍 Breakdown:

- Finds the **max age** for each department.
- Matches it with **employee records**.

📌 21. Find Employees Who Have a Higher Salary Than the Company's Median Salary

📝 Question:

Write a query to find all employees who have a **higher salary than the company's median salary**.

👷‍♀️ Query:

```
SELECT *

FROM employees

WHERE salary > (SELECT salary FROM employees ORDER BY salary LIMIT 1 OFFSET (SELECT COUNT(*)/2 FROM employees));
```

🔍 Breakdown:

- ORDER BY salary LIMIT 1 OFFSET (COUNT(*)/2) → Finds the **median salary**.
- Filters employees **above the median salary**.

🏆 Basic SQL Interview Questions with Explanations

📌 1. Select All Data from a Table

📝 Question:

Write an SQL query to select **all columns** from the employees table.

👨‍💻 Query:

```
SELECT * FROM employees;
```

🔍 Breakdown:

- The * means **"select everything"** from the table.
- **Simple!** This is the most basic SQL query.

📌 2. Select Specific Columns

📝 Question:

Write an SQL query to select only name and salary columns from the employees table.

👨‍💻 Query:

```
SELECT name, salary FROM employees;
```

🔍 Breakdown:

- Instead of *, we **list specific column names**.

📌 3. Filter Results Using WHERE

📝 Question:

Write an SQL query to find all employees with a salary greater than 50000.

👨‍💻 Query:

```
SELECT * FROM employees WHERE salary > 50000;
```

🔍 Breakdown:

- WHERE filters results **based on a condition**.
- salary > 50000 means **only show employees earning more than 50,000**.

📌 4. Use AND & OR in Conditions

📝 Question:

Find employees who **work in "HR" department AND** earn more than 40000.

👨‍💻 Query:

```
SELECT * FROM employees WHERE department = 'HR' AND salary > 40000;
```

🔍 Breakdown:

- AND means **both conditions must be true**.
- Use OR if **either condition can be true**.

📌 5. Sort Data Using ORDER BY

📝 Question:

Write a query to list all employees **sorted by salary in descending order**.

👨‍💻 Query:

```
SELECT * FROM employees ORDER BY salary DESC;
```

🔍 Breakdown:

- ORDER BY salary → Sorts results **by salary**.
- DESC → **Highest salary first** (Descending).
- Use ASC for **lowest salary first**.

📌 6. Limit Results

📝 Question:

Write a query to find the **top 5 highest-paid employees**.

👨‍💻 Query:

```
SELECT * FROM employees ORDER BY salary DESC LIMIT 5;
```

🔍 Breakdown:

- LIMIT 5 → Shows **only the first 5 results**.

📌 7. Count the Number of Employees

📝 Question:

Find the **total number of employees**.

👨‍💻 Query:

```
SELECT COUNT(*) FROM employees;
```

🔍 Breakdown:

- COUNT(*) counts **all rows** in the table.

📌 8. Find the Average Salary

📝 Question:

Find the **average salary** of all employees.

👨‍💻 Query:

```
SELECT AVG(salary) FROM employees;
```

🔍 Breakdown:

- AVG(salary) → Calculates **average salary**.

📌 9. Find the Highest and Lowest Salary

📝 Question:

Find the **highest and lowest salaries**.

👨‍💻 Query:

```
SELECT MAX(salary) AS highest_salary, MIN(salary) AS lowest_salary FROM employees;
```

🔍 Breakdown:

- MAX(salary) → Finds the **highest** salary.
- MIN(salary) → Finds the **lowest** salary.

📌 10. Group Data Using GROUP BY

📝 Question:

Find the **total number of employees in each department**.

👨‍💻 Query:

```
SELECT department, COUNT(*) AS total_employees

FROM employees

GROUP BY department;
```

🔍 Breakdown:

- GROUP BY department → Groups employees **by department**.

- COUNT(*) → Counts employees in each department.

📌 11. Filter Groups Using HAVING

📝 Question:

Find **departments with more than 5 employees**.

👨‍💻 Query:

```
SELECT department, COUNT(*) AS total_employees

FROM employees

GROUP BY department

HAVING COUNT(*) > 5;
```

🔍 Breakdown:

- HAVING works **like WHERE** but is used with **GROUP BY**.
- It **filters groups** based on a condition.

📌 12. Find Duplicates Using GROUP BY & HAVING

📝 Question:

Find employees who **appear more than once**.

👨‍💻 Query:

```
SELECT name, COUNT(*)

FROM employees

GROUP BY name

HAVING COUNT(*) > 1;
```

🔍 Breakdown:

- Groups by name and **counts occurrences**.
- HAVING COUNT(*) > 1 → Shows **only duplicates**.

📌 13. Delete Data from a Table

📝 Question:

Delete employees who **earn less than 30000**.

👨‍💻 Query:

```
DELETE FROM employees WHERE salary < 30000;
```

🔍 Breakdown:

- DELETE removes **rows that match the condition**.

🚨 **Be careful!** This **permanently deletes data**.

📌 14. Update Data in a Table

📝 Question:

Increase salary by **10%** for employees **in the "IT" department**.

👨‍💻 Query:

```
UPDATE employees

SET salary = salary * 1.10

WHERE department = 'IT';
```

🔍 Breakdown:

- UPDATE modifies data.
- SET changes the salary.
- WHERE ensures **only IT employees** are updated.

📌 15. Find Employees Who Don't Have a Manager

📝 Question:

Find employees **without a manager**.

👨‍💻 Query:

```
SELECT * FROM employees WHERE manager_id IS NULL;
```

🔍 Breakdown:

- IS NULL finds **missing values**.

📌 16. Find Common Data Between Two Tables

📝 Question:

Find customers **who have placed orders**.

👨‍💻 Query:

```
SELECT DISTINCT customers.name

FROM customers

JOIN orders ON customers.customer_id = orders.customer_id;
```

🔍 Breakdown:

- JOIN combines **both tables**.
- DISTINCT removes **duplicate names**.

📌 17. Find Employees and Their Manager's Name

📝 Question:

Get **employees with their manager's name**.

👨‍💻 Query:

```
SELECT e.name AS employee, m.name AS manager

FROM employees e

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

🔍 Breakdown:

- LEFT JOIN → Matches employees with **their manager**.
- e.name AS employee, m.name AS manager → Renames columns.

Conclusion

These **SQL queries** are essential for **Software QA roles** to validate databases, test automation results, and verify data consistency. Mastering these will help in **database testing** and **automation frameworks**.

Let me know if you need more advanced queries! 🚀

By-

REEHAN NIZAMI SHAZADA

TEST SPECIALIST | TEST ARCHITECT

Add in Linkedin for similar contents