# 1. Employee Salary Data:

You are given a table Employee with columns id, name, and salary.

**Scenario:**

You need to find the second-highest salary in the table.

**Question:**

Write a SQL query to find the second-highest salary from the Employee table.

**Answer:**

SELECT MAX(salary) AS SecondHighestSalary

FROM Employee

WHERE salary < (SELECT MAX(salary) FROM Employee);

**Explanation:**

* The subquery gets the highest salary.
* The main query finds the highest salary less than the maximum salary, which gives the second-highest.

# 2. Customer Orders:

There is a Customer table with columns id, name, and region. There's also an Orders table with columns order\_id, customer\_id, order\_date, and amount.

**Scenario:**

Find all customers who have placed more than 5 orders in the last year.

**Question:**

Write a SQL query to get the names of customers who placed more than 5 orders in the last year.

**Answer:**

SELECT C.name

FROM Customer C

JOIN Orders O ON C.id = O.customer\_id

WHERE O.order\_date >= DATEADD(YEAR, -1, GETDATE())

GROUP BY C.name

HAVING COUNT(O.order\_id) > 5;

**Explanation:**

* DATEADD(YEAR, -1, GETDATE()) gets the date exactly one year before today.
* Join Customer with Orders to get customer details for each order.
* Group by Customer.name and use HAVING to filter customers with more than 5 orders.

# 3. Top-N Products:

You have an Orders table with columns product\_id, quantity, and order\_date.

**Scenario:**

Find the top 3 products that were sold the most in terms of quantity in the current year.

**Question:**

Write a SQL query to list the top 3 products by total quantity sold this year.

**Answer:**

SELECT product\_id, SUM(quantity) AS TotalSold

FROM Orders

WHERE YEAR(order\_date) = YEAR(GETDATE())

GROUP BY product\_id

ORDER BY TotalSold DESC

LIMIT 3;

**Explanation:**

* Filter orders from the current year using YEAR(order\_date) = YEAR(GETDATE()).
* Group by product\_id and sum the quantities.
* Use ORDER BY to sort in descending order and LIMIT 3 to get the top 3.

# 4. Finding Duplicates:

You are given a User table with columns id, email, and name.

**Scenario:**

Find all users who have registered with duplicate email addresses.

**Question:**

Write a SQL query to find duplicate email addresses.

**Answer:**

SELECT email, COUNT(\*)

FROM User

GROUP BY email

HAVING COUNT(\*) > 1;

**Explanation:**

* Group by email and count the number of occurrences for each email.
* Use HAVING COUNT(\*) > 1 to filter out unique emails and keep only duplicates.

# 5. Aggregated Data:

## There is a Sales table with columns sale\_id, product\_id, sale\_date, and amount.

**Scenario:**

Calculate the total sales and the average sales amount for each product.

**Question:**

Write a SQL query to calculate the total and average sales per product.

**Answer:**

SELECT product\_id, SUM(amount) AS TotalSales, AVG(amount) AS AverageSales

FROM Sales

GROUP BY product\_id;

**Explanation:**

* Use SUM to get total sales per product and AVG to get the average sales.
* Group by product\_id to aggregate the data per product.

# 6. Subquery Usage:

You have a Students table with columns student\_id, name, and gpa, and a Courses table with columns course\_id, student\_id, and grade.

**Scenario:**

Find the names of students who have enrolled in more than 3 courses.

**Question:**

Write a SQL query to find the students who have enrolled in more than 3 courses.

**Answer:**

SELECT name

FROM Students

WHERE student\_id IN (

SELECT student\_id

FROM Courses

GROUP BY student\_id

HAVING COUNT(course\_id) > 3

);

**Explanation:**

* The subquery finds student\_id values for students enrolled in more than 3 courses.
* The main query selects student names based on those IDs.

# 7. Date Filtering:

There is a Logs table with columns log\_id, log\_date, and log\_message.

**Scenario:**

Find the logs that were created in the last 7 days.

**Question:**

Write a SQL query to get logs from the last 7 days.

**Answer:**

SELECT \*

FROM Logs

WHERE log\_date >= DATEADD(DAY, -7, GETDATE());

**Explanation:**

* DATEADD(DAY, -7, GETDATE()) calculates the date 7 days ago.
* The query filters logs where the log\_date is within the last 7 days.

# 8. Self-Join Scenario:

You have an Employee table with columns id, name, manager\_id (which refers to another employee).

**Scenario:**

Find the employees who have the same manager.

**Question:**

Write a SQL query to find pairs of employees who report to the same manager.

**Answer:**

SELECT e1.name AS Employee1, e2.name AS Employee2, e1.manager\_id

FROM Employee e1

JOIN Employee e2 ON e1.manager\_id = e2.manager\_id

WHERE e1.id <> e2.id;

**Explanation:**

* Perform a self-join on the Employee table to compare employees with the same manager\_id.
* Use WHERE e1.id <> e2.id to ensure the query does not return the same employee.

# 10. Joins and Subqueries

* **Problem:**
  + You have two tables:
    - orders: order\_id, customer\_id, order\_date, total\_amount
    - customers: customer\_id, customer\_name, city
  + Write a SQL query to retrieve the customer\_name and city for customers who placed an order in the last 30 days.
* **Expected Solution Outline:**

SQL

SELECT c.customer\_name, c.city

FROM customers c

JOIN orders o ON c.customer\_id = o.customer\_id

WHERE o.order\_date >= DATE\_SUB(CURDATE(), INTERVAL 30 DAY);

# 11. Question : Window Functions

* **Problem:**
  + You have a table sales with columns: product\_id, sale\_date, quantity\_sold.
  + Calculate the 7-day moving average of quantity\_sold for each product\_id.
* **Expected Solution Outline:**

SQL

SELECT

product\_id,

sale\_date,

AVG(quantity\_sold) OVER (

PARTITION BY product\_id

ORDER BY sale\_date

ROWS BETWEEN 6 PRECEDING AND CURRENT ROW

) AS moving\_average

FROM sales;

# 12. Customer Transactions:

You have a Transactions table with columns transaction\_id, customer\_id, transaction\_date, and amount.

**Scenario:**

Find customers who have made transactions on two consecutive days.

**Question:**

Write a SQL query to find all customers who made transactions on consecutive days.

**Answer:**

SELECT DISTINCT t1.customer\_id

FROM Transactions t1

JOIN Transactions t2

ON t1.customer\_id = t2.customer\_id

AND DATEDIFF(DAY, t1.transaction\_date, t2.transaction\_date) = 1;

# 10. Product Reviews:

You have a Reviews table with columns review\_id, product\_id, rating, and review\_date.

**Scenario:**

Find the top 3 products with the highest average rating.

**Question:**

Write a SQL query to get the top 3 products by average rating.

**Answer:**

SELECT product\_id, AVG(rating) AS avg\_rating

FROM Reviews

GROUP BY product\_id

ORDER BY avg\_rating DESC

LIMIT 3;

# 11. Employee Hierarchy:

You have an Employee table with columns id, name, and manager\_id. Each employee has a manager\_id which refers to another employee.

**Scenario:**

Find all employees who report directly or indirectly to a specific manager (e.g., manager with ID 2).

**Question:**

Write a SQL query to find all employees reporting to the manager with ID 2, either directly or indirectly.

**Answer:**

WITH RecursiveCTE AS (

SELECT id, name, manager\_id

FROM Employee

WHERE manager\_id = 2

UNION ALL

SELECT e.id, e.name, e.manager\_id

FROM Employee e

INNER JOIN RecursiveCTE r

ON e.manager\_id = r.id

)

SELECT id, name

FROM RecursiveCTE;

**Explanation:**

* This is a recursive common table expression (CTE) that starts with employees reporting directly to the manager.
* The recursive part continues to find employees reporting to the subordinates of the initial manager.

## Alternative Approach: Self-Join Method

This approach involves continuously joining the Employee table to itself to traverse the hierarchy manually. Here’s how you can do it:

**SQL Query:**

SELECT DISTINCT e1.id, e1.name

FROM Employee e1

JOIN Employee e2

ON e1.manager\_id = e2.id

JOIN Employee e3

ON e2.manager\_id = e3.id

-- Continue joining as many levels deep as necessary, depending on the hierarchy depth

WHERE e3.id = 2;

**Explanation:**

* In this approach, you repeatedly join the Employee table to itself to go deeper into the hierarchy, step by step.
* Start by finding employees whose manager\_id corresponds to the direct reports of the manager with ID 2.
* Continue the self-join to go deeper into the hierarchy, depending on the depth of your hierarchy structure.
* The number of JOIN operations depends on the depth of the hierarchy you need to traverse. Each join goes one level deeper.

# 12. Sales Targets:

You have a Sales table with columns sale\_id, employee\_id, sale\_amount, and sale\_date.

**Scenario:**

Identify employees who exceeded their sales target of $10,000 in a single month.

**Question:**

Write a SQL query to find employees who exceeded $10,000 in sales in any single month.

**Answer:**

SELECT employee\_id, SUM(sale\_amount) AS total\_sales, MONTH(sale\_date) AS sale\_month, YEAR(sale\_date) AS sale\_year

FROM Sales

GROUP BY employee\_id, YEAR(sale\_date), MONTH(sale\_date)

HAVING SUM(sale\_amount) > 10000;

# 13. Dormant Customers:

You have a Customers table with columns customer\_id, name, and last\_purchase\_date.

**Scenario:**

Identify customers who have not made any purchases in the last 6 months.

**Question:**

Write a SQL query to find all customers who have not made a purchase in the last 6 months.

**Answer:**

SELECT customer\_id, name

FROM Customers

WHERE last\_purchase\_date < DATEADD(MONTH, -6, GETDATE());

**14. Order Discounts:**

You have an Orders table with columns order\_id, customer\_id, order\_amount, and discount\_code.

**Scenario:**

Find customers who have placed more than 3 orders and never used a discount code.

**Question:**

Write a SQL query to get the customer IDs of customers who placed more than 3 orders and never used a discount code.

**Answer:**

SELECT customer\_id

FROM Orders

WHERE discount\_code IS NULL

GROUP BY customer\_id

HAVING COUNT(order\_id) > 3;

# 15. Event Attendance:

You have an Events table with columns event\_id, attendee\_id, attendance\_date, and location.

**Scenario:**

Find attendees who attended more than 2 different events at the same location.

**Question:**

Write a SQL query to get the attendees who attended more than 2 events at the same location.

**Answer:**

SELECT attendee\_id, location, COUNT(event\_id) AS event\_count

FROM Events

GROUP BY attendee\_id, location

HAVING COUNT(event\_id) > 2;

# 16. Product Inventory:

You have an Inventory table with columns product\_id, quantity, restock\_date, and warehouse\_id.

**Scenario:**

Find products that have not been restocked in the last 30 days.

**Question:**

Write a SQL query to find products that haven't been restocked in the last 30 days.

**Answer:**

SELECT product\_id

FROM Inventory

WHERE restock\_date < DATEADD(DAY, -30, GETDATE());

# 7. Churned Customers:

You have a Customers table with columns customer\_id, name, and last\_order\_date.

Scenario:

Identify customers who have not placed any orders in the last year but had placed at least one order before.

Question:

Write a SQL query to find customers who have not placed an order in the last year but have placed at least one order before.

Answer:

SELECT customer\_id, name

FROM Customers

WHERE last\_order\_date < DATEADD(YEAR, -1, GETDATE())

AND last\_order\_date IS NOT NULL;

# 18. Order Summary:

You have an Orders table with columns order\_id, customer\_id, order\_date, and total\_amount.

**Scenario:**

Summarize the total amount spent by each customer in the past year.

**Question:**

Write a SQL query to summarize the total amount spent by each customer over the past year.

**Answer:**

SELECT customer\_id, SUM(total\_amount) AS total\_spent

FROM Orders

WHERE order\_date >= DATEADD(YEAR, -1, GETDATE())

GROUP BY customer\_id;

# 1. Find Employees and Their Managers:

You have an Employee table with columns id, name, and manager\_id, where manager\_id refers to the employee’s manager.

**Scenario:**

Write a SQL query to list the employees along with their manager’s name.

**Answer:**

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

FROM Employee e1

LEFT JOIN Employee e2

ON e1.manager\_id = e2.id;

**Explanation:**

* This query uses a **self-join** where e1 represents employees, and e2 represents their corresponding managers.
* The LEFT JOIN ensures that employees without managers (e.g., the CEO) are still included in the result.

# 2. Find Employees Without a Manager:

You have an Employee table, and some employees do not report to a manager (i.e., their manager\_id is NULL).

**Scenario:**

Write a SQL query to find all employees who do not have a manager.

**Answer:**

SELECT e.name

FROM Employee e

WHERE e.manager\_id IS NULL;

# 3. Find Employees with the Same Manager:

You have an Employee table with columns id, name, and manager\_id. You want to find all pairs of employees who report to the same manager.

**Scenario:**

Write a SQL query to find all pairs of employees who report to the same manager.

**Answer:**

SELECT e1.name AS Employee1, e2.name AS Employee2, e1.manager\_id

FROM Employee e1

JOIN Employee e2

ON e1.manager\_id = e2.manager\_id

AND e1.id < e2.id;

# 4. Find Employee Pairs Where One Employee is the Manager of the Other:

You have an Employee table with columns id, name, and manager\_id. Find all employee pairs where one employee is the direct manager of the other.

**Scenario:**

Write a SQL query to list all pairs where one employee is the manager of the other.

**Answer:**

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

FROM Employee e1

JOIN Employee e2

ON e1.manager\_id = e2.id;

**Explanation:**

* This query performs a **self-join** where e1 represents employees, and e2 represents their managers.
* The condition e1.manager\_id = e2.id links employees to their direct supervisors.

# 5. Find All Managers Who Manage More Than 5 Employees:

You have an Employee table, and you want to find all managers who directly manage more than 5 employees.

**Scenario:**

Write a SQL query to find managers who manage more than 5 employees.

**Answer:**

SELECT e2.name AS Manager, COUNT(e1.id) AS NumEmployees

FROM Employee e1

JOIN Employee e2

ON e1.manager\_id = e2.id

GROUP BY e2.name

HAVING COUNT(e1.id) > 5;

**Explanation:**

* This query uses a **self-join** where e1 represents employees, and e2 represents managers.
* The GROUP BY clause groups the employees by their manager, and the HAVING clause filters out managers who manage 5 or fewer employees.

# 6. Find the Most Senior Employee in Each Department:

You have an Employee table with columns id, name, manager\_id, and hire\_date. Write a query to find the employee who has been in each department the longest.

**Scenario:**

Write a SQL query to find the most senior employee (the one with the earliest hire date) in each department.

**Answer:**

SELECT e1.name, e1.department\_id, e1.hire\_date

FROM Employee e1

LEFT JOIN Employee e2

ON e1.department\_id = e2.department\_id

AND e1.hire\_date > e2.hire\_date

WHERE e2.id IS NULL;

# 7. Find Employees at the Same Level in the Hierarchy:

You have an Employee table with columns id, name, and manager\_id. You want to find employees who are at the same level in the hierarchy (i.e., they have the same manager’s manager).

**Scenario:**

Write a SQL query to list employees who are at the same hierarchical level (they share the same grand-manager).

**Answer:**

SELECT e1.name AS Employee1, e2.name AS Employee2, e1.manager\_id AS Manager1, e2.manager\_id AS Manager2

FROM Employee e1

JOIN Employee e2

ON e1.manager\_id = e2.manager\_id

JOIN Employee e3

ON e1.manager\_id = e3.id

AND e2.manager\_id = e3.id;

**Explanation:**

* This query performs two levels of **self-join** to find employees (e1 and e2) who have the same manager (e3).

# Scenario: Imagine you're working with a database for an online bookstore. The database has the following tables:

1. Books (book\_id, title, author\_id, price, stock\_quantity)
2. Authors (author\_id, author\_name, country)
3. Orders (order\_id, customer\_id, order\_date)
4. OrderItems (order\_id, book\_id, quantity)
5. Customers (customer\_id, customer\_name, email)

Question: The marketing team wants to run a promotion for the top 5 bestselling books of the last month. They need a report that includes the book title, author name, and total number of copies sold. Can you write a SQL query to generate this report?

Answer: Here's a SQL query that would solve this problem:

SELECT

b.title AS book\_title,

a.author\_name,

SUM(oi.quantity) AS total\_copies\_sold

FROM

Books b

JOIN Authors a ON b.author\_id = a.author\_id

JOIN OrderItems oi ON b.book\_id = oi.book\_id

JOIN Orders o ON oi.order\_id = o.order\_id

WHERE

o.order\_date >= DATE\_SUB(CURDATE(), INTERVAL 1 MONTH)

GROUP BY

b.book\_id, b.title, a.author\_name

ORDER BY

total\_copies\_sold DESC

LIMIT 5;

Explanation:

1. We start by joining the necessary tables: Books, Authors, OrderItems, and Orders.
2. The WHERE clause filters for orders from the last month using DATE\_SUB(CURDATE(), INTERVAL 1 MONTH).
3. We GROUP BY book\_id, title, and author\_name to get the total sales for each book.
4. The results are ordered by total\_copies\_sold in descending order.
5. LIMIT 5 ensures we only get the top 5 bestselling books.

This query will return a result set with three columns: book\_title, author\_name, and total\_copies\_sold, showing the top 5 bestselling books of the last month.

Follow-up questions an interviewer might ask:

1. How would you modify this query to get the top selling book for each author?
2. What if we wanted to include books that haven't sold any copies in the last month?
3. How would you calculate the total revenue generated by these top 5 books?

# Scenario 1: Employee Management System

Tables:

1. Employees (employee\_id, first\_name, last\_name, hire\_date, department\_id, salary)
2. Departments (department\_id, department\_name, manager\_id)

## Question: Find the department with the highest average salary. Return the department name and the average salary.

Answer:

SELECT

d.department\_name,

AVG(e.salary) AS avg\_salary

FROM

Employees e

JOIN Departments d ON e.department\_id = d.department\_id

GROUP BY

d.department\_id, d.department\_name

ORDER BY

avg\_salary DESC

LIMIT 1;

# Scenario 2: E-commerce Platform

Tables:

1. Products (product\_id, product\_name, category\_id, price)
2. Categories (category\_id, category\_name)
3. Sales (sale\_id, product\_id, sale\_date, quantity)

## Find the top 3 categories by total revenue in the last quarter. Include the category name and total revenue.

Answer:

SELECT

c.category\_name,

SUM(p.price \* s.quantity) AS total\_revenue

FROM

Sales s

JOIN Products p ON s.product\_id = p.product\_id

JOIN Categories c ON p.category\_id = c.category\_id

WHERE

s.sale\_date >= DATE\_SUB(CURDATE(), INTERVAL 3 MONTH)

GROUP BY

c.category\_id, c.category\_name

ORDER BY

total\_revenue DESC

LIMIT 3;

# Scenario 3: Social Media Platform

Tables:

1. Users (user\_id, username, join\_date)
2. Posts (post\_id, user\_id, post\_date, content)
3. Likes (like\_id, post\_id, user\_id, like\_date)

## Question: Find the users who have posted more than 5 times but have never received a like.

Answer:

SELECT

u.username,

COUNT(p.post\_id) AS post\_count

FROM

Users u

JOIN Posts p ON u.user\_id = p.user\_id

LEFT JOIN Likes l ON p.post\_id = l.post\_id

GROUP BY

u.user\_id, u.username

HAVING

COUNT(p.post\_id) > 5 AND COUNT(l.like\_id) = 0;

# Scenario 5: Library Management System

Tables:

1. Books (book\_id, title, author, publication\_year, copies\_available)
2. Members (member\_id, member\_name, join\_date)
3. Loans (loan\_id, book\_id, member\_id, loan\_date, return\_date)

Question: Find the top 3 most frequently borrowed books that are currently overdue. Include the book title, author, and the number of times it has been borrowed.

Answer:

SELECT

b.title,

b.author,

COUNT(l.loan\_id) AS borrow\_count

FROM

Books b

JOIN Loans l ON b.book\_id = l.book\_id

WHERE

l.return\_date IS NULL AND l.loan\_date < CURDATE()

GROUP BY

b.book\_id, b.title, b.author

ORDER BY

borrow\_count DESC

LIMIT 3;