1. **Write a query to find all customers who placed more than one order in a single day.**

SQL

SELECT customer\_id, order\_date

FROM Orders

GROUP BY customer\_id, order\_date

HAVING COUNT(\*) > 1;

1. **Get the details of employees who do not have a department assigned (LEFT JOIN with NULL check).**

SQL

SELECT e.\*

FROM Employees e

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

WHERE d.department\_id IS NULL;

1. **Find the second-highest salary in each department using a self-join.**

SQL

SELECT DISTINCT e1.\*

FROM Employees e1

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

WHERE e1.salary < e2.salary

AND NOT EXISTS (

SELECT DISTINCT e3.salary

FROM Employees e3

WHERE e3.department\_id = e1.department\_id

AND e3.salary > e1.salary

AND NOT EXISTS (

SELECT e4.salary

FROM Employees e4

WHERE e4.department\_id = e1.department\_id

AND e4.salary > e3.salary

)

);

1. **Join employee and department tables to find departments with no employees.**

SQL

SELECT d.\*

FROM Departments d

LEFT JOIN Employees e ON d.department\_id = e.department\_id

WHERE e.employee\_id IS NULL;

1. **Find all customers who ordered only once using LEFT JOIN and aggregation.**

SQL

SELECT c.\*

FROM Customers c

LEFT JOIN Orders o ON c.customer\_id = o.customer\_id

GROUP BY c.customer\_id

HAVING COUNT(o.order\_id) = 1;

1. **Use a FULL OUTER JOIN to find mismatched records between two tables.**

SQL

SELECT \*

FROM Table1

FULL OUTER JOIN Table2 ON Table1.common\_field = Table2.common\_field

WHERE Table1.common\_field IS NULL OR Table2.common\_field IS NULL;

1. **Find products that have never been ordered using an ANTI JOIN (LEFT JOIN + IS NULL).**

SQL

SELECT p.\*

FROM Products p

LEFT JOIN OrderDetails od ON p.product\_id = od.product\_id

WHERE od.order\_id IS NULL;

1. **Use CROSS JOIN to generate all possible pairs of employees and projects.**

SQL

SELECT e.employee\_name, p.project\_name

FROM Employees e

CROSS JOIN Projects p;

1. **Find employees who share the same manager using a self-join.**

SQL

SELECT DISTINCT e1.\*

FROM Employees e1

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

WHERE e1.employee\_id <> e2.employee\_id;

1. **Join customer and orders to find the top spender per city.**

SQL

SELECT c.city, c.customer\_id, SUM(o.total\_amount) as TotalSpent

FROM Customers c

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

GROUP BY c.city, c.customer\_id

HAVING SUM(o.total\_amount) = (

SELECT MAX(TotalSpent)

FROM (

SELECT c2.city, c2.customer\_id, SUM(o2.total\_amount) as TotalSpent

FROM Customers c2

JOIN Orders o2 ON c2.customer\_id = o2.customer\_id

WHERE c2.city = c.city

GROUP BY c2.city, c2.customer\_id

) as CityTotals

);

1. **Use a JOIN to calculate the average order value per customer.**

SQL

SELECT c.customer\_id, AVG(o.total\_amount) as AverageOrderValue

FROM Customers c

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

GROUP BY c.customer\_id;

1. **Join products and order details to find the most sold product in each category.**

SQL

SELECT p.category, p.product\_id, SUM(od.quantity) as TotalQuantitySold

FROM Products p

JOIN OrderDetails od ON p.product\_id = od.product\_id

GROUP BY p.category, p.product\_id

HAVING SUM(od.quantity) = (

SELECT MAX(TotalQuantitySold)

FROM (

SELECT p2.category, p2.product\_id, SUM(od2.quantity) as TotalQuantitySold

FROM Products p2

JOIN OrderDetails od2 ON p2.product\_id = od2.product\_id

WHERE p2.category = p.category

GROUP BY p2.category, p2.product\_id

) as CategoryTotals

);

1. **Combine two sales tables (same structure) from different years using UNION ALL, then join with products.**

SQL

SELECT p.\*, s.\*

FROM Products p

JOIN (

SELECT \* FROM Sales2022

UNION ALL

SELECT \* FROM Sales2023

) as s ON p.product\_id = s.product\_id;

1. **Write a query to show customers who have never logged in using LEFT JOIN and login table.**

SQL

SELECT c.\*

FROM Customers c

LEFT JOIN LoginDetails l ON c.customer\_id = l.customer\_id

WHERE l.login\_date IS NULL;

1. **Show all employees with and without projects using LEFT JOIN and a CASE expression.**

SQL

SELECT e.\*,

CASE

WHEN p.project\_id IS NULL THEN 'No Project'

ELSE 'Assigned to Project'

END as ProjectStatus

FROM Employees e

LEFT JOIN EmployeeProjects ep ON e.employee\_id = ep.employee\_id

LEFT JOIN Projects p ON ep.project\_id = p.project\_id;

1. **Join orders with payments to check for unpaid orders.**

SQL

SELECT o.\*

FROM Orders o

LEFT JOIN Payments p ON o.order\_id = p.order\_id

WHERE p.payment\_status <> 'Paid' OR p.payment\_status IS NULL;

1. **Use a self-join to find employees who have the same job title as someone else.**

SQL

SELECT DISTINCT e1.\*

FROM Employees e1

JOIN Employees e2 ON e1.job\_title = e2.job\_title

WHERE e1.employee\_id <> e2.employee\_id;

1. **Create a report of customers who have ordered all products in a specific category.**

SQL

SELECT c.\*

FROM Customers c

WHERE NOT EXISTS (

SELECT p.product\_id

FROM Products p

WHERE p.category = 'SpecificCategory' -- Replace 'SpecificCategory'

AND NOT EXISTS (

SELECT o.order\_id

FROM Orders o

JOIN OrderDetails od ON o.order\_id = od.order\_id

WHERE o.customer\_id = c.customer\_id

AND od.product\_id = p.product\_id

)

);

1. **Use a RIGHT JOIN to identify regions with no assigned sales reps.**

SQL

SELECT r.\*

FROM SalesReps sr

RIGHT JOIN Regions r ON sr.region\_id = r.region\_id

WHERE sr.sales\_rep\_id IS NULL;

1. **Join invoices and customers, and flag invoices that exceed the customer's credit limit.**

SQL

SELECT i.\*,

CASE

WHEN i.amount > c.credit\_limit THEN 'Exceeds Credit Limit'

ELSE 'Within Credit Limit'

END as CreditLimitStatus

FROM Invoices i

JOIN Customers c ON i.customer\_id = c.customer\_id;

1. **Find orders where the shipping address and billing address are from different cities.**

SQL

SELECT \*

FROM Orders

WHERE shipping\_city <> billing\_city;

1. **Use a join to find duplicate email addresses in the customer table.**

SQL

SELECT c1.\*

FROM Customers c1

JOIN Customers c2 ON c1.email = c2.email

WHERE c1.customer\_id <> c2.customer\_id;

1. **Join to find users who placed an order in the last 30 days but not before that.**

SQL

SELECT DISTINCT c.\*

FROM Customers c

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

WHERE o.order\_date >= date('now', '-30 days')

AND c.customer\_id NOT IN (

SELECT customer\_id

FROM Orders

WHERE order\_date < date('now', '-30 days')

);

1. **Use multiple joins to get order details including customer, product, and delivery info.**

SQL

SELECT o.\*, c.\*, p.\*, d.\*

FROM Orders o

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

JOIN OrderDetails od ON o.order\_id = od.order\_id

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

JOIN DeliveryInfo d ON o.delivery\_id = d.delivery\_id;

1. **Use EXISTS in place of joins to identify customers with multiple addresses.**

SQL

SELECT c.\*

FROM Customers c

WHERE EXISTS (

SELECT \*

FROM Addresses a

WHERE a.customer\_id = c.customer\_id

AND EXISTS (

SELECT \*

FROM Addresses a2

WHERE a2.customer\_id = c.customer\_id

AND a.address\_id <> a2.address\_id

)

);

**Subquery Questions**

1. **Find the highest-paid employee in each department using a correlated subquery.**

SQL

SELECT e.\*

FROM Employees e

WHERE e.salary = (

SELECT MAX(e2.salary)

FROM Employees e2

WHERE e2.department\_id = e.department\_id

);

1. **Write a query to list customers who have placed orders with all available sales agents.**

SQL

SELECT c.\*

FROM Customers c

WHERE NOT EXISTS (

SELECT sa.sales\_agent\_id

FROM SalesAgents sa

WHERE NOT EXISTS (

SELECT o.order\_id

FROM Orders o

WHERE o.customer\_id = c.customer\_id

AND o.sales\_agent\_id = sa.sales\_agent\_id

)

);

1. **Find all products that have been ordered more than the average quantity.**

SQL

SELECT p.\*

FROM Products p

JOIN OrderDetails od ON p.product\_id = od.product\_id

GROUP BY p.product\_id

HAVING SUM(od.quantity) > (SELECT AVG(quantity) FROM OrderDetails);

1. **Use a subquery to get the product with the highest revenue.**

SQL

SELECT p.\*

FROM Products p

JOIN OrderDetails od ON p.product\_id = od.product\_id

GROUP BY p.product\_id

ORDER BY SUM(od.quantity \* p.price) DESC

LIMIT 1;

1. **List departments where the average salary is higher than the company-wide average.**

SQL

SELECT d.\*

FROM Departments d

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

GROUP BY d.department\_id

HAVING AVG(e.salary) > (SELECT AVG(salary) FROM Employees);

1. **Find employees who earn more than their department's average salary.**

SQL

SELECT e1.\*

FROM Employees e1

WHERE e1.salary > (

SELECT AVG(e2.salary)

FROM Employees e2

WHERE e2.department\_id = e1.department\_id

);

1. **Get all customers who have placed more orders than customer X.**

SQL

SELECT c.\*

FROM Customers c

WHERE (SELECT COUNT(\*) FROM Orders WHERE customer\_id = c.customer\_id) >

(SELECT COUNT(\*) FROM Orders WHERE customer\_id = 'X'); -- Replace 'X' with the actual customer ID

1. **Use a subquery to find the second-highest order value.**

SQL

SELECT MAX(total\_amount)

FROM Orders

WHERE total\_amount < (SELECT MAX(total\_amount) FROM Orders);

1. **Return the names of employees who have the same salary as someone in another department.**

SQL

SELECT DISTINCT e1.employee\_name

FROM Employees e1

WHERE e1.salary IN (

SELECT e2.salary

FROM Employees e2

WHERE e2.department\_id <> e1.department\_id

);

1. **Get products whose price is greater than the average price in their category.**

SQL

SELECT p1.\*

FROM Products p1

WHERE p1.price > (

SELECT AVG(p2.price)

FROM Products p2

WHERE p2.category = p1.category

);

1. **List employees whose salary is in the top 10% of their department.**

SQL

SELECT e1.\*

FROM Employees e1

WHERE e1.salary >= (

SELECT PERCENTILE\_CONT(0.90) WITHIN GROUP (ORDER BY e2.salary)

FROM Employees e2

WHERE e2.department\_id = e1.department\_id

);

1. **Write a query to find orders placed on the same day as the customer's birthday.**

SQL

SELECT o.\*

FROM Orders o

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

WHERE o.order\_date = c.birthday;

1. **Find users who have reviewed every product they've purchased.**

SQL

SELECT DISTINCT c.\*

FROM Customers c

WHERE NOT EXISTS (

SELECT od.product\_id

FROM Orders o

JOIN OrderDetails od ON o.order\_id = od.order\_id

WHERE o.customer\_id = c.customer\_id

AND NOT EXISTS (

SELECT r.product\_id

FROM Reviews r

WHERE r.customer\_id = c.customer\_id

AND r.product\_id = od.product\_id

)

);

1. **Use NOT EXISTS to list customers who have never ordered.**

SQL

SELECT c.\*

FROM Customers c

WHERE NOT EXISTS (

SELECT o.order\_id

FROM Orders o

WHERE o.customer\_id = c.customer\_id

);

1. **Get a list of suppliers who supply products that were never ordered.**

SQL

SELECT s.\*

FROM Suppliers s

WHERE NOT EXISTS (

SELECT p.product\_id

FROM Products p

JOIN OrderDetails od ON p.product\_id = od.product\_id

WHERE p.supplier\_id = s.supplier\_id

);

1. **Find products that are cheaper than all other products in the same category.**

SQL

SELECT p1.\*

FROM Products p1

WHERE NOT EXISTS (

SELECT p2.\*

FROM Products p2

WHERE p2.category = p1.category

AND p2.price < p1.price

);

1. **Write a subquery to find the most recent order per customer.**

SQL

SELECT o1.\*

FROM Orders o1

WHERE o1.order\_date = (

SELECT MAX(o2.order\_date)

FROM Orders o2

WHERE o2.customer\_id = o1.customer\_id

);

1. **Use a scalar subquery to show the average salary next to each employee.**

SQL

SELECT e.\*, (SELECT AVG(salary) FROM Employees) as AverageSalary

FROM Employees e;

1. **Find customers who spent more than the average customer but made fewer purchases.**

SQL

SELECT c.\*

FROM Customers c

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

GROUP BY c.customer\_id

HAVING SUM(o.total\_amount) > (SELECT AVG(TotalSpent) FROM (SELECT SUM(total\_amount) as TotalSpent FROM Orders GROUP BY customer\_id))

AND COUNT(o.order\_id) < (SELECT AVG(OrderCount) FROM (SELECT COUNT(\*) as OrderCount FROM Orders GROUP BY customer\_id));

1. **Use a subquery to check if a product was ever sold at a discount.**

SQL

SELECT p.\*

FROM Products p

WHERE EXISTS (

SELECT \*

FROM OrderDetails od

WHERE od.product\_id = p.product\_id

AND od.discount > 0

);

1. **Find employees whose salary is below the department's median salary.**

SQL

SELECT e1.\*

FROM Employees e1

WHERE e1.salary < (

SELECT PERCENTILE\_CONT(0.5) WITHIN GROUP (ORDER BY e2.salary)

FROM Employees e2

WHERE e2.department\_id = e1.department\_id

);

1. **Use a correlated subquery to show the running total of sales per day.**

SQL

SELECT o1.order\_date, SUM(o1.total\_amount) as DailySales,

(SELECT SUM(o2.total\_amount)

FROM Orders o2

WHERE o2.order\_date <= o1.order\_date) as RunningTotal

FROM Orders o1

GROUP BY o1.order\_date