**3. Get the total number of orders placed by each customer, but only include those who have placed more than 1 order.**

**Answer:**

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName, COUNT(o.OrderID) AS TotalOrders

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID

HAVING COUNT(o.OrderID) > 1;

**4. Find the product(s) with the highest price.**

**Answer:**

sql

CopyEdit

SELECT \* FROM products

WHERE Price = (SELECT MAX(Price) FROM products);

**6. Find the average salary for each job title, sorted from highest to lowest.**

**Answer:**

sql

CopyEdit

SELECT j.JobTitle, AVG(p.Salary) AS AvgSalary

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

GROUP BY j.JobTitle

ORDER BY AvgSalary DESC;

**7. Find customers from Bangladesh whose city name has exactly 5 letters.**

**Answer:**

sql

CopyEdit

SELECT \* FROM customers

WHERE Country = 'Bangladesh' AND City REGEXP '^.{5}$';

**8. List all orders along with the total number of products in each order.**

**Answer:**

sql

CopyEdit

SELECT o.OrderID, COUNT(od.ProductID) AS TotalProducts

FROM orders o

JOIN orderdetails od ON o.OrderID = od.OrderID

GROUP BY o.OrderID;

**9. Retrieve the name of the highest-paid employee in each job category.**

**Answer:**

SELECT p.PersonID, p.FirstName, p.LastName, j.JobTitle, p.Salary

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE p.Salary = (SELECT MAX(Salary) FROM persons WHERE JobID = p.JobID);

**10. Find the second-highest-priced product.**

**Answer:**

sql

CopyEdit

SELECT ProductID, ProductName, Price

FROM products

ORDER BY Price DESC

LIMIT 1 OFFSET 1;

**11. Find the first order placed in each month of 2024.**

**Answer:**

SELECT o.OrderID, o.OrderDate, o.CustomerID

FROM orders o

JOIN (

SELECT MONTH(OrderDate) AS order\_month,

MIN(OrderDate) AS first\_date

FROM orders

WHERE YEAR(OrderDate) = 2024

GROUP BY MONTH(OrderDate)

) first\_orders

ON MONTH(o.OrderDate) = first\_orders.order\_month

AND o.OrderDate = first\_orders.first\_date

WHERE YEAR(o.OrderDate) = 2024;

SELECT OrderID, OrderDate, CustomerID

FROM orders

WHERE YEAR(OrderDate) = 2024

AND OrderID IN (

SELECT TOP 1 OrderID

FROM orders o2

WHERE MONTH(o2.OrderDate) = MONTH(orders.OrderDate)

AND YEAR(o2.OrderDate) = 2024

ORDER BY o2.OrderDate

);

**12. Get all employees whose salary is within 10% of the maximum salary.**

**Answer:**

sql

CopyEdit

SELECT \* FROM persons

WHERE Salary >= (SELECT MAX(Salary) \* 0.9 FROM persons);

**13. Find the total sales (quantity \* price) for each product, sorted in descending order.**

**Answer:**

sql

CopyEdit

SELECT p.ProductName, SUM(od.Quantity \* p.Price) AS TotalSales

FROM orderdetails od

JOIN products p ON od.ProductID = p.ProductID

GROUP BY p.ProductName

ORDER BY TotalSales DESC;

**14. Retrieve customers who have never placed an order.**

**Answer:**

sql

CopyEdit

SELECT \* FROM customers

WHERE CustomerID NOT IN (SELECT DISTINCT CustomerID FROM orders);

**15. Update the price of all products in CategoryID 3 by increasing it by 10%.**

**Answer:**

sql

CopyEdit

UPDATE products

SET Price = Price \* 1.10

WHERE CategoryID = 3;

**16. Delete orders that were placed before 2023.**

**Answer:**

sql

CopyEdit

DELETE FROM orders

WHERE OrderDate < '2023-01-01';

**17. Count how many employees earn more than the average salary.**

**Answer:**

sql

CopyEdit

SELECT COUNT(\*) AS AboveAvgEmployees

FROM persons

WHERE Salary > (SELECT AVG(Salary) FROM persons);

**18. Retrieve the name of the youngest employee along with their job title.**

**Answer:**

sql

CopyEdit

SELECT p.FirstName, p.LastName, j.JobTitle, p.Age

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE p.Age = (SELECT MIN(Age) FROM persons);

**19. Find the most frequently ordered product.**

**Answer:**

sql

CopyEdit

SELECT p.ProductName, SUM(od.Quantity) AS TotalQuantity

FROM orderdetails od

JOIN products p ON od.ProductID = p.ProductID

GROUP BY p.ProductName

ORDER BY TotalQuantity DESC

LIMIT 1;

**20. Retrieve all employees whose last name starts with 'S' and have a salary above 50,000.**

**Answer:**

sql

CopyEdit

SELECT \* FROM persons

WHERE LastName LIKE 'S%' AND Salary > 50000;

**21. Show the rank of each employee's salary within their job category.**

**Answer:**

sql

CopyEdit

SELECT p.PersonID, p.FirstName, p.LastName, j.JobTitle, p.Salary,

RANK() OVER (PARTITION BY p.JobID ORDER BY p.Salary DESC) AS SalaryRank

FROM persons p

JOIN jobs j ON p.JobID = j.JobID;

**22. Retrieve customers who live in Dhaka and have a postal code starting with ‘12’.**

**Answer:**

SELECT \* FROM customers

WHERE City = 'Dhaka' AND PostalCode LIKE '12%';

**23. Find the highest-paid female employee.**

**Answer:**

SELECT \* FROM persons

WHERE Gender = 'Female' AND Salary = (SELECT MAX(Salary) FROM persons WHERE Gender = 'Female');

**24. Retrieve products that have never been ordered.**

**Answer:**

sql

CopyEdit

SELECT \* FROM products

WHERE ProductID NOT IN (SELECT DISTINCT ProductID FROM orderdetails);

**25. Count how many employees have been hired each year.**

**Answer:**

SELECT YEAR(HireDate) AS HireYear, COUNT(\*) AS EmployeeCount

FROM persons

GROUP BY HireYear

ORDER BY HireYear;

**1. Retrieve all customers whose names start with ‘S’ and live in Dhaka, but exclude those in PostalCode 1203.**

sql

CopyEdit

SELECT \* FROM customers

WHERE CustomerName LIKE 'S%'

AND City = 'Dhaka'

AND PostalCode <> 1203;

SELECT CustomerName, City, PostalCode

FROM customers

WHERE CustomerName LIKE 'S%'

AND (City = 'Dhaka' OR City = 'Mumbai’)

AND PostalCode <> 1203

AND Gender = 'Male'

GROUP BY CustomerName, City, PostalCode

**2. Fetch the top 3 customers who have placed the most orders, but only those who have placed more than 1 order.**

SELECT c.CustomerID, c.CustomerName, COUNT(o.OrderID) AS TotalOrders

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID

HAVING TotalOrders > 1

ORDER BY TotalOrders DESC

LIMIT 3;

**3. Find the total number of products ordered for each customer, but only show customers who have ordered more than 30 items.**

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName, SUM(od.Quantity) AS TotalQuantity

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

JOIN orderdetails od ON o.OrderID = od.OrderID

GROUP BY c.CustomerID

HAVING SUM(od.Quantity) > 30;

**4. Retrieve all orders placed in 2024 by customers who live outside Bangladesh and sort them by order date.**

sql

CopyEdit

SELECT o.\*

FROM orders o

JOIN customers c ON o.CustomerID = c.CustomerID

WHERE c.Country <> 'Bangladesh'

AND YEAR(o.OrderDate) = 2024

ORDER BY o.OrderDate;

**5. Get the names of all customers whose address contains a number followed by a slash (e.g., ‘102/F’) using REGEXP.**

sql

CopyEdit

SELECT CustomerName, Address

FROM customers

WHERE Address REGEXP '[0-9]+/[A-Za-z]';

**6. Find the top 2 highest-paid employees whose salaries are below the maximum salary for their respective jobs.**

sql

CopyEdit

SELECT p.PersonID, p.FirstName, p.LastName, p.Salary, j.MaxSalary

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE p.Salary < j.MaxSalary

ORDER BY p.Salary DESC

LIMIT 2;

**7. Get the employees who were hired before 2020 and have a salary lower than the minimum salary for their job role.**

sql

CopyEdit

SELECT p.\*

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE p.HireDate < '2020-01-01'

AND p.Salary < j.MinSalary;

**8. Retrieve the first order made by each customer.**

sql

CopyEdit

SELECT o.\*

FROM orders o

WHERE OrderDate = (

SELECT MIN(OrderDate) FROM orders o2 WHERE o.CustomerID = o2.CustomerID

);

**9. Fetch the average salary of all employees in each job role, but only for jobs with more than one employee.**

sql

CopyEdit

SELECT j.JobTitle, AVG(p.Salary) AS AvgSalary

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

GROUP BY j.JobTitle

HAVING COUNT(p.PersonID) > 1;

**10. List the names of customers who have not placed any orders.**

sql

CopyEdit

SELECT c.CustomerName

FROM customers c

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

WHERE o.OrderID IS NULL;

**11. Retrieve the most expensive product in each category.**

sql

CopyEdit

SELECT p.\*

FROM products p

WHERE Price = (

SELECT MAX(Price) FROM products p2 WHERE p.CategoryID = p2.CategoryID

);

**12. Fetch employees whose last name starts with ‘S’ and have worked for more than 5 years.**

sql

CopyEdit

SELECT \* FROM persons

WHERE LastName LIKE 'S%'

AND DATEDIFF(CURDATE(), HireDate) > 5 \* 365;

**13. Find the second-highest salary among employees.**

sql

CopyEdit

SELECT DISTINCT Salary

FROM persons

ORDER BY Salary DESC

LIMIT 1 OFFSET 1;

**14. Get the total revenue generated from each product and sort it in descending order.**

sql

CopyEdit

SELECT p.ProductName, SUM(od.Quantity \* p.Price) AS TotalRevenue

FROM products p

JOIN orderdetails od ON p.ProductID = od.ProductID

GROUP BY p.ProductName

ORDER BY TotalRevenue DESC;

**15. Find the most popular product by order count.**

sql

CopyEdit

SELECT p.ProductName, COUNT(od.OrderdetailID) AS OrderCount

FROM products p

JOIN orderdetails od ON p.ProductID = od.ProductID

GROUP BY p.ProductName

ORDER BY OrderCount DESC

LIMIT 1;

**16. Retrieve customers who have ordered at least 3 different products.**

sql

CopyEdit

SELECT o.CustomerID, c.CustomerName, COUNT(DISTINCT od.ProductID) AS UniqueProducts

FROM orders o

JOIN customers c ON o.CustomerID = c.CustomerID

JOIN orderdetails od ON o.OrderID = od.OrderID

GROUP BY o.CustomerID

HAVING COUNT(DISTINCT od.ProductID) >= 3;

**17. Get all orders where the total price exceeds 5000 BDT.**

sql

CopyEdit

SELECT o.OrderID, SUM(od.Quantity \* p.Price) AS TotalPrice

FROM orders o

JOIN orderdetails od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

GROUP BY o.OrderID

HAVING TotalPrice > 5000;

**18. Retrieve employees along with the difference between their salary and their job’s average salary.**

sql

CopyEdit

SELECT p.PersonID, p.FirstName, p.Salary, j.JobTitle,

p.Salary - (SELECT AVG(Salary) FROM persons WHERE JobID = p.JobID) AS SalaryDifference

FROM persons p

JOIN jobs j ON p.JobID = j.JobID;

**19. Retrieve all customers who live in either Dhaka or Mumbai and have placed at least one order.**

sql

CopyEdit

SELECT DISTINCT c.\*

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

WHERE c.City IN ('Dhaka', 'Mumbai');

**20. Find employees who have a salary higher than the average salary of all employees.**

sql

CopyEdit

SELECT \* FROM persons

WHERE Salary > (SELECT AVG(Salary) FROM persons);

**21. Retrieve customers whose names have at least two occurrences of the letter 'a'.**

sql

CopyEdit

SELECT \* FROM customers

WHERE CustomerName REGEXP 'a.\*a';

**22. Get the highest-priced product in each category and include categories where no products exist.**

sql

CopyEdit

SELECT c.CategoryID, MAX(p.Price) AS MaxPrice

FROM products p

RIGHT JOIN (SELECT DISTINCT CategoryID FROM products) c ON p.CategoryID = c.CategoryID

GROUP BY c.CategoryID;

**23. Retrieve the total number of male and female customers.**

sql

CopyEdit

SELECT Gender, COUNT(\*) AS TotalCount

FROM customers

GROUP BY Gender;

**24. Get a running total of salaries for employees ordered by hire date.**

sql

CopyEdit

SELECT PersonID, FirstName, Salary, HireDate,

SUM(Salary) OVER (ORDER BY HireDate) AS RunningTotalSalary

FROM persons;

**25. Retrieve employees who are the only ones in their job role.**

SELECT p.\*

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE (SELECT COUNT(\*) FROM persons WHERE JobID = p.JobID) = 1;

### ****1. Retrieve Customers Who Have Placed More Than Two Orders in the Last Three Months****

Retrieve the CustomerID, CustomerName, and TotalOrders for customers who have placed **more than two orders** in the last **three months**. Include only those customers who are from **Bangladesh** and sort the result by the **number of orders** in descending order.

#### **Answer:**

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName, COUNT(o.OrderID) AS TotalOrders

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

WHERE o.OrderDate >= DATE\_SUB(CURDATE(), INTERVAL 3 MONTH)

AND c.Country = 'Bangladesh'

GROUP BY c.CustomerID

HAVING TotalOrders > 2

ORDER BY TotalOrders DESC;

* DATE\_SUB(CURDATE(), INTERVAL 3 MONTH): Filters orders from the last 3 months.
* HAVING TotalOrders > 2: Ensures customers have placed more than 2 orders.

### ****2. Retrieve Customers Who Ordered More Than 100 Units of a Single Product****

Find customers who have ordered **more than 100 units** of a single product. Display the CustomerID, CustomerName, ProductID, and total quantity ordered. Exclude customers who belong to India.

#### **Answer:**

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName, od.ProductID, SUM(od.Quantity) AS TotalQuantity

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

JOIN orderdetails od ON o.OrderID = od.OrderID

WHERE c.Country != 'India'

GROUP BY c.CustomerID, od.ProductID

HAVING TotalQuantity > 100;

* SUM(od.Quantity) > 100: Ensures customers ordered **over 100 units**.
* c.Country != 'India': Excludes Indian customers.

### ****3. Retrieve the Names of Employees Earning Above the Average Salary in Their Job Category****

Find all employees earning **above the average salary** in their respective job categories. Display the PersonID, FirstName, LastName, JobTitle, and Salary. Sort results by JobTitle.

#### **Answer:**

sql

CopyEdit

SELECT p.PersonID, p.FirstName, p.LastName, j.JobTitle, p.Salary

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE p.Salary > (

SELECT AVG(Salary)

FROM persons

WHERE JobID = p.JobID

)

ORDER BY j.JobTitle;

* Subquery calculates **average salary** for each job.
* **Only employees above their category’s average salary are retrieved**.

### ****4. Retrieve the Most Expensive Product Ordered by Each Customer****

Find the **most expensive product** each customer has ordered. Display CustomerID, CustomerName, ProductName, and Price.

#### **Answer:**

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName, p.ProductName, p.Price

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

JOIN orderdetails od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

WHERE p.Price = (

SELECT MAX(p2.Price)

FROM orderdetails od2

JOIN products p2 ON od2.ProductID = p2.ProductID

WHERE od2.OrderID = o.OrderID

);

* Subquery finds the **highest-priced product** per order.
* **Ensures only the most expensive product per customer is shown**.

### ****5. Retrieve Employees Hired in the Last Five Years Who Earn More Than 50,000****

Find employees **hired within the last five years** who **earn more than 50,000**. Display PersonID, FirstName, LastName, Salary, HireDate, and JobTitle.

#### **Answer:**

sql

CopyEdit

SELECT p.PersonID, p.FirstName, p.LastName, p.Salary, p.HireDate, j.JobTitle

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE p.HireDate >= DATE\_SUB(CURDATE(), INTERVAL 5 YEAR)

AND p.Salary > 50000;

* DATE\_SUB(CURDATE(), INTERVAL 5 YEAR): Filters **employees hired in the last 5 years**.

### ****6. Retrieve the Name of the Highest-Paid Employee in Each Job Category****

Find the **highest-paid employee** in each **job category**. Display PersonID, FirstName, LastName, JobTitle, and Salary.

#### **Answer:**

sql

CopyEdit

SELECT p.PersonID, p.FirstName, p.LastName, j.JobTitle, p.Salary

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE p.Salary = (SELECT MAX(Salary) FROM persons WHERE JobID = p.JobID);

* Uses **subquery** to find **maximum salary per job**.

### ****7. Retrieve the Customers Who Ordered at Least Two Different Products in a Single Order****

Find **customers who ordered at least two different products in one order**. Display CustomerID, CustomerName, OrderID, and TotalProductsOrdered.

#### **Answer:**

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName, o.OrderID, COUNT(DISTINCT od.ProductID) AS TotalProductsOrdered

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

JOIN orderdetails od ON o.OrderID = od.OrderID

GROUP BY o.OrderID, c.CustomerID, c.CustomerName

HAVING TotalProductsOrdered >= 2;

* COUNT(DISTINCT od.ProductID): Ensures **different products are counted**.
* HAVING TotalProductsOrdered >= 2: Filters **orders with at least 2 products**.

### ****8. Retrieve the Second Highest Salary in Each Job Category****

Find the **second highest salary** in each **job category**. Display JobTitle, PersonID, FirstName, LastName, and Salary.

#### **Answer:**

#

SELECT PersonID, FirstName, LastName, JobTitle, Salary FROM (

SELECT p.PersonID, p.FirstName, p.LastName, j.JobTitle, p.Salary,

RANK() OVER (PARTITION BY p.JobID ORDER BY p.Salary DESC) AS salary\_rank

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

) ranked

WHERE salary\_rank = 2;

* Uses **window function RANK()** to assign ranks.
* **Filters only the second-highest salary** in each job category.

Find customers who have **placed at least one order in each of the last three months**. Display CustomerID, CustomerName, and total orders placed in the last three months.

### ****Answer:****

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName, COUNT(o.OrderID) AS TotalOrders

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

WHERE o.OrderDate >= DATE\_SUB(CURDATE(), INTERVAL 3 MONTH)

GROUP BY c.CustomerID, c.CustomerName

HAVING COUNT(DISTINCT MONTH(o.OrderDate)) = 3;

### ****Explanation:****

* DATE\_SUB(CURDATE(), INTERVAL 3 MONTH): Filters orders **from the last three months**.
* COUNT(DISTINCT MONTH(o.OrderDate)) = 3: Ensures the customer has at least **one order in each month**.

## **2. Retrieve the Name of the Employee Who Has Served the Most Unique Customers**

Find the **employee who has handled the most unique customers**. Display EmployeeID, EmployeeName, and total unique customers.

### ****Answer:****

sql

CopyEdit

SELECT e.EmployeeID, e.EmployeeName, COUNT(DISTINCT o.CustomerID) AS UniqueCustomers

FROM employees e

JOIN orders o ON e.EmployeeID = o.EmployeeID

GROUP BY e.EmployeeID, e.EmployeeName

ORDER BY UniqueCustomers DESC

LIMIT 1;

### ****Explanation:****

* COUNT(DISTINCT o.CustomerID): Counts **unique customers served**.
* ORDER BY UniqueCustomers DESC LIMIT 1: Retrieves **only the top employee**.

## **3. Find Customers Who Have Never Ordered the Same Product Twice**

Find customers who have **never ordered the same product twice** in separate orders.

### ****Answer:****

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

JOIN orderdetails od ON o.OrderID = od.OrderID

GROUP BY c.CustomerID

HAVING MAX(od.ProductID) = MIN(od.ProductID);

### ****Explanation:****

* Ensures each customer has **only ever ordered one unique product**.
* Uses MAX(ProductID) = MIN(ProductID) to check if they only ordered **one product type**.

## **4. Retrieve the Most Popular Product in Terms of Distinct Customers**

Find the product ordered by the **highest number of distinct customers**.

### ****Answer:****

sql

CopyEdit

SELECT p.ProductID, p.ProductName, COUNT(DISTINCT o.CustomerID) AS CustomerCount

FROM orderdetails od

JOIN products p ON od.ProductID = p.ProductID

JOIN orders o ON od.OrderID = o.OrderID

GROUP BY p.ProductID, p.ProductName

ORDER BY CustomerCount DESC

LIMIT 1;

### ****Explanation:****

* COUNT(DISTINCT o.CustomerID): Counts **unique customers ordering a product**.
* **Most popular product** is sorted using ORDER BY CustomerCount DESC.

## **5. Retrieve Customers Who Ordered More Than Five Products but Never Ordered Tea**

Find customers who have ordered **more than five different products** but **never ordered tea**.

### ****Answer:****

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

JOIN orderdetails od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

GROUP BY c.CustomerID

HAVING COUNT(DISTINCT od.ProductID) > 5

AND SUM(CASE WHEN p.ProductName LIKE '%Tea%' THEN 1 ELSE 0 END) = 0;

### ****Explanation:****

* HAVING COUNT(DISTINCT od.ProductID) > 5: Ensures **at least five different products**.
* SUM(CASE WHEN p.ProductName LIKE '%Tea%' THEN 1 ELSE 0 END) = 0: Ensures **never ordered tea**.

## **6. Retrieve the Order ID with the Longest Delivery Time**

Find the OrderID with the **longest time between order placement and delivery**.

### ****Answer:****

sql

CopyEdit

SELECT OrderID, DATEDIFF(DeliveryDate, OrderDate) AS DeliveryTime

FROM orders

ORDER BY DeliveryTime DESC

LIMIT 1;

### ****Explanation:****

* DATEDIFF(DeliveryDate, OrderDate): Calculates **days taken for delivery**.
* ORDER BY DeliveryTime DESC LIMIT 1: Finds **order with longest delivery time**.

## **SUPER HARD QUESTION 1: Retrieve the Top Three Products Contributing to the Highest Revenue Each Month**

Find the **top 3 products in revenue per month**.

### ****Answer:****

sql

CopyEdit

WITH MonthlyRevenue AS (

SELECT

p.ProductID, p.ProductName, MONTH(o.OrderDate) AS OrderMonth,

SUM(od.Quantity \* p.Price) AS TotalRevenue,

RANK() OVER (PARTITION BY MONTH(o.OrderDate) ORDER BY SUM(od.Quantity \* p.Price) DESC) AS RevenueRank

FROM orders o

JOIN orderdetails od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

GROUP BY p.ProductID, OrderMonth

)

SELECT ProductID, ProductName, OrderMonth, TotalRevenue

FROM MonthlyRevenue

WHERE RevenueRank <= 3;

### ****Explanation:****

* Uses **window function RANK()** to rank products by **total revenue per month**.
* Filters only the **top 3 revenue-generating products**.

## **SUPER HARD QUESTION 2: Find the Two Most Frequently Bought Together Products**

Find the **two products that appear together the most in the same orders**.

### ****Answer:****

sql

CopyEdit

SELECT od1.ProductID AS Product1, od2.ProductID AS Product2, COUNT(\*) AS Frequency

FROM orderdetails od1

JOIN orderdetails od2 ON od1.OrderID = od2.OrderID AND od1.ProductID < od2.ProductID

GROUP BY od1.ProductID, od2.ProductID

ORDER BY Frequency DESC

LIMIT 1;

### ****Explanation:****

* Self-joins orderdetails on OrderID to **pair products in the same order**.
* AND od1.ProductID < od2.ProductID: Avoids **duplicate pairs**.
* Retrieves **most frequently bought-together product pair**.

## **More Hard Questions (7-15)**

* **7:** Find customers who have ordered from at least **three different categories** but never ordered snacks.
* **8:** Retrieve employees who have handled orders worth **at least $10,000** but have not served any VIP customers.
* **9:** Find the **most expensive order** where **at least three different products were purchased**.
* **10:** Retrieve the customers whose **average order value is greater than $500**.
* **11:** Retrieve orders where the **total quantity ordered was greater than 200 units** but was **delivered late**.
* **12:** Retrieve customers who have placed **more than 10 orders but have at least one order canceled**.
* **13:** Find the employees whose **average handling time per order is the longest**.
* **14:** Retrieve orders where **more than one employee was involved**.
* **15:** Find the customers who have spent **more than $5,000 in total but have never ordered on a weekend**.

## **7. Find Customers Who Have Ordered from at Least Three Different Categories but Never Ordered Snacks**

Retrieve customers who have placed orders from **at least three different product categories**, but **never ordered any product from the "Snacks" category**.

### ****Answer:****

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

JOIN orderdetails od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

JOIN categories cat ON p.CategoryID = cat.CategoryID

GROUP BY c.CustomerID, c.CustomerName

HAVING COUNT(DISTINCT cat.CategoryID) >= 3

AND SUM(CASE WHEN cat.CategoryName = 'Snacks' THEN 1 ELSE 0 END) = 0;

### ****Explanation:****

* COUNT(DISTINCT cat.CategoryID) >= 3: Ensures the customer has ordered from **at least 3 different categories**.
* SUM(CASE WHEN cat.CategoryName = 'Snacks' THEN 1 ELSE 0 END) = 0: Ensures they **never ordered snacks**.

## **8. Retrieve Employees Who Have Handled Orders Worth at Least $10,000 but Have Not Served Any VIP Customers**

Find employees who have processed orders worth **at least $10,000 total** but **have not handled any VIP customers**.

### ****Answer:****

sql

CopyEdit

SELECT e.EmployeeID, e.EmployeeName, SUM(od.Quantity \* p.Price) AS TotalSales

FROM employees e

JOIN orders o ON e.EmployeeID = o.EmployeeID

JOIN orderdetails od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

JOIN customers c ON o.CustomerID = c.CustomerID

GROUP BY e.EmployeeID, e.EmployeeName

HAVING TotalSales >= 10000

AND SUM(CASE WHEN c.CustomerType = 'VIP' THEN 1 ELSE 0 END) = 0;

### ****Explanation:****

* SUM(od.Quantity \* p.Price) >= 10000: Ensures the employee has handled **at least $10,000 in sales**.
* SUM(CASE WHEN c.CustomerType = 'VIP' THEN 1 ELSE 0 END) = 0: Ensures they **never served VIP customers**.

## **9. Find the Most Expensive Order Where at Least Three Different Products Were Purchased**

Retrieve the OrderID with the **highest total order value**, considering only orders where **at least 3 different products were bought**.

### ****Answer:****

sql

CopyEdit

SELECT o.OrderID, SUM(od.Quantity \* p.Price) AS TotalOrderValue

FROM orders o

JOIN orderdetails od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

GROUP BY o.OrderID

HAVING COUNT(DISTINCT od.ProductID) >= 3

ORDER BY TotalOrderValue DESC

LIMIT 1;

### ****Explanation:****

* HAVING COUNT(DISTINCT od.ProductID) >= 3: Ensures **at least 3 unique products**.
* ORDER BY TotalOrderValue DESC LIMIT 1: Retrieves the **most expensive valid order**.

## **10. Retrieve Customers Whose Average Order Value Is Greater Than $500**

Find customers whose **average order value** is **greater than $500**.

### ****Answer:****

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName, AVG(o.TotalAmount) AS AvgOrderValue

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID

HAVING AvgOrderValue > 500;

### ****Explanation:****

* Uses AVG(o.TotalAmount) > 500 to **filter customers** based on **average order value**.

## **11. Retrieve Orders Where the Total Quantity Ordered Was Greater Than 200 Units but Was Delivered Late**

Find orders where the **total quantity of items ordered was more than 200** but **the delivery was late**.

### ****Answer:****

sql

CopyEdit

SELECT o.OrderID, SUM(od.Quantity) AS TotalQuantity, o.DeliveryDate, o.RequiredDate

FROM orders o

JOIN orderdetails od ON o.OrderID = od.OrderID

GROUP BY o.OrderID

HAVING TotalQuantity > 200 AND o.DeliveryDate > o.RequiredDate;

### ****Explanation:****

* HAVING TotalQuantity > 200: Filters orders **with more than 200 units**.
* o.DeliveryDate > o.RequiredDate: Ensures the order **was delivered late**.

## **12. Retrieve Customers Who Have Placed More Than 10 Orders but Have at Least One Order Canceled**

Find customers who have placed **more than 10 orders**, and at least **one of their orders was canceled**.

### ****Answer:****

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName, COUNT(o.OrderID) AS TotalOrders

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID

HAVING COUNT(o.OrderID) > 10

AND SUM(CASE WHEN o.Status = 'Canceled' THEN 1 ELSE 0 END) > 0;

### ****Explanation:****

* COUNT(o.OrderID) > 10: Ensures they placed **more than 10 orders**.
* SUM(CASE WHEN o.Status = 'Canceled' THEN 1 ELSE 0 END) > 0: Ensures **at least one was canceled**.

## **13. Find the Employees Whose Average Handling Time Per Order Is the Longest**

Retrieve employees whose **average handling time per order** (from order placement to delivery) is **the highest**.

### ****Answer:****

sql

CopyEdit

SELECT e.EmployeeID, e.EmployeeName, AVG(DATEDIFF(o.DeliveryDate, o.OrderDate)) AS AvgHandlingTime

FROM employees e

JOIN orders o ON e.EmployeeID = o.EmployeeID

GROUP BY e.EmployeeID

ORDER BY AvgHandlingTime DESC

LIMIT 1;

### ****Explanation:****

* AVG(DATEDIFF(o.DeliveryDate, o.OrderDate)): Calculates **average handling time**.
* Retrieves **the employee with the longest average handling time**.

## **14. Retrieve Orders Where More Than One Employee Was Involved**

Find OrderIDs where **more than one employee** was involved.

### ****Answer:****

sql

CopyEdit

SELECT o.OrderID, COUNT(DISTINCT e.EmployeeID) AS EmployeeCount

FROM orders o

JOIN employees\_orders eo ON o.OrderID = eo.OrderID

JOIN employees e ON eo.EmployeeID = e.EmployeeID

GROUP BY o.OrderID

HAVING EmployeeCount > 1;

### ****Explanation:****

* Uses a **many-to-many relationship (employees\_orders table)**.
* Filters orders where COUNT(DISTINCT e.EmployeeID) > 1, meaning **more than one employee handled the order**.

## **15. Find the Customers Who Have Spent More Than $5,000 in Total but Have Never Ordered on a Weekend**

Find customers who **spent more than $5,000 in total** but **never placed an order on a Saturday or Sunday**.

### ****Answer:****

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName, SUM(o.TotalAmount) AS TotalSpent

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID

HAVING TotalSpent > 5000

AND SUM(CASE WHEN DAYOFWEEK(o.OrderDate) IN (1,7) THEN 1 ELSE 0 END) = 0;

### ****Explanation:****

* SUM(o.TotalAmount) > 5000: Ensures **total spending exceeds $5,000**.
* SUM(CASE WHEN DAYOFWEEK(o.OrderDate) IN (1,7) THEN 1 ELSE 0 END) = 0: Ensures **no weekend orders**.

### ****1. Retrieve Employees Who Live in 'Bashundhara' and Earn More Than the Highest Paid Employee in 'Bailey Road'****

Find employees who **live in 'Bashundhara'** and earn **more than the highest-paid employee living in 'Bailey Road'**.

#### **Answer:**

sql

CopyEdit

SELECT e.EmployeeID, e.EmployeeName, e.Salary, a.Area

FROM employees e

JOIN addresses a ON e.AddressID = a.AddressID

WHERE a.Area = 'Bashundhara'

AND e.Salary > (

SELECT MAX(e2.Salary)

FROM employees e2

JOIN addresses a2 ON e2.AddressID = a2.AddressID

WHERE a2.Area = 'Bailey Road'

);

#### **Explanation:**

* First, finds the **highest salary of employees living in 'Bailey Road'**.
* Then, filters employees **who live in 'Bashundhara' and earn more**.

### ****2. Retrieve Customers Who Live in 'Dhanmondi' and Have Spent More Than Any Customer Living in 'Gulshan'****

Find customers from **Dhanmondi** who have spent **more than the highest total spending customer from Gulshan**.

#### **Answer:**

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName, SUM(o.TotalAmount) AS TotalSpent, a.Area

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

JOIN addresses a ON c.AddressID = a.AddressID

WHERE a.Area = 'Dhanmondi'

GROUP BY c.CustomerID

HAVING TotalSpent > (

SELECT MAX(TotalSpent)

FROM (

SELECT c2.CustomerID, SUM(o2.TotalAmount) AS TotalSpent

FROM customers c2

JOIN orders o2 ON c2.CustomerID = o2.CustomerID

JOIN addresses a2 ON c2.AddressID = a2.AddressID

WHERE a2.Area = 'Gulshan'

GROUP BY c2.CustomerID

) AS GulshanCustomers

);

#### **Explanation:**

* First, calculates **total spending for each customer from Gulshan**.
* Then, selects customers **from Dhanmondi** who spent **more than the highest spender in Gulshan**.

### ****3. Find Employees Who Have Handled Orders Worth More Than Their Own Salary****

Retrieve employees who have **handled orders with a total value greater than their own salary**.

#### **Answer:**

sql

CopyEdit

SELECT e.EmployeeID, e.EmployeeName, e.Salary, SUM(od.Quantity \* p.Price) AS TotalHandledOrders

FROM employees e

JOIN orders o ON e.EmployeeID = o.EmployeeID

JOIN orderdetails od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

GROUP BY e.EmployeeID

HAVING TotalHandledOrders > e.Salary;

#### **Explanation:**

* Calculates **total order value handled by each employee**.
* Compares it to their **own salary**.

### ****4. Find Customers Who Ordered the Most Expensive Item but Live in an Area Where the Cheapest Item Was Also Ordered****

Find customers who ordered the **most expensive product** but live in an area where **the cheapest product was also ordered**.

#### **Answer:**

sql

CopyEdit

SELECT DISTINCT c.CustomerID, c.CustomerName, a.Area

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

JOIN orderdetails od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

JOIN addresses a ON c.AddressID = a.AddressID

WHERE p.Price = (SELECT MAX(Price) FROM products)

AND a.Area IN (

SELECT DISTINCT a2.Area

FROM orders o2

JOIN orderdetails od2 ON o2.OrderID = od2.OrderID

JOIN products p2 ON od2.ProductID = p2.ProductID

JOIN customers c2 ON o2.CustomerID = c2.CustomerID

JOIN addresses a2 ON c2.AddressID = a2.AddressID

WHERE p2.Price = (SELECT MIN(Price) FROM products)

);

#### **Explanation:**

* Finds **customers who ordered the most expensive product**.
* Ensures they live in an **area where someone else ordered the cheapest product**.

### ****5. Find Employees Who Work in 'Gulshan' but Have Handled More Orders Than Any Employee in 'Banani'****

Retrieve employees who work in **Gulshan** but have **handled more orders than the busiest employee in Banani**.

#### **Answer:**

sql

CopyEdit

SELECT e.EmployeeID, e.EmployeeName, COUNT(o.OrderID) AS OrdersHandled

FROM employees e

JOIN orders o ON e.EmployeeID = o.EmployeeID

JOIN addresses a ON e.AddressID = a.AddressID

WHERE a.Area = 'Gulshan'

GROUP BY e.EmployeeID

HAVING OrdersHandled > (

SELECT MAX(OrdersHandled)

FROM (

SELECT COUNT(o2.OrderID) AS OrdersHandled

FROM employees e2

JOIN orders o2 ON e2.EmployeeID = o2.EmployeeID

JOIN addresses a2 ON e2.AddressID = a2.AddressID

WHERE a2.Area = 'Banani'

GROUP BY e2.EmployeeID

) AS BananiEmployees

);

#### **Explanation:**

* Finds the **maximum orders handled by an employee in Banani**.
* Filters **Gulshan employees who handled more**.

### ****6. Find Customers Who Have Placed Orders in Multiple Areas and Never Ordered From Their Own Area****

Retrieve customers who **placed orders in at least 3 different areas** but **never in their own area**.

#### **Answer:**

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName, COUNT(DISTINCT a2.Area) AS DifferentAreasOrdered

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

JOIN addresses a2 ON o.DeliveryAddressID = a2.AddressID

JOIN addresses a ON c.AddressID = a.AddressID

WHERE a2.Area <> a.Area

GROUP BY c.CustomerID

HAVING DifferentAreasOrdered >= 3;

#### **Explanation:**

* Filters customers **who ordered from at least 3 different areas**.
* Ensures they **never ordered from their own area**.

### ****7. Retrieve Customers Who Have Spent More Than the Wealthiest Customer in 'Uttara' but Have Never Ordered Online****

Find customers who spent **more than the highest spending customer from Uttara** but **only ordered in-store**.

#### **Answer:**

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName, SUM(o.TotalAmount) AS TotalSpent

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

WHERE o.OrderType = 'In-Store'

GROUP BY c.CustomerID

HAVING TotalSpent > (

SELECT MAX(TotalSpent)

FROM (

SELECT c2.CustomerID, SUM(o2.TotalAmount) AS TotalSpent

FROM customers c2

JOIN orders o2 ON c2.CustomerID = o2.CustomerID

JOIN addresses a ON c2.AddressID = a.AddressID

WHERE a.Area = 'Uttara'

GROUP BY c2.CustomerID

) AS UttaraCustomers

);

#### **Explanation:**

* First, finds the **highest spending customer from Uttara**.
* Then, filters **customers who spent more but never ordered online**.

### ****1. Retrieve the First Letter of Each Customer's Name in Uppercase****

Write a query that extracts the **first letter** of each customer’s **first name**, converts it to **uppercase**, and displays it along with their **full name**.

#### **Answer:**

sql

CopyEdit

SELECT CustomerID, FirstName, LastName, UPPER(LEFT(FirstName, 1)) AS FirstLetter

FROM customers;

✔ Uses UPPER() to **convert to uppercase**.  
✔ Uses LEFT() to **extract the first letter**.

### ****2. Calculate the Length of Each Product Name****

Find the **length of each product name** and display it alongside the **product name**.

#### **Answer:**

sql

CopyEdit

SELECT ProductID, ProductName, LENGTH(ProductName) AS NameLength

FROM products;

✔ Uses LENGTH() to **find string length**.

### ****3. Round Off All Employee Salaries to the Nearest Hundred****

Write a query to **round off** employee salaries to the **nearest hundred**.

#### **Answer:**

sql

CopyEdit

SELECT EmployeeID, EmployeeName, Salary, ROUND(Salary, -2) AS RoundedSalary

FROM employees;

✔ ROUND(Salary, -2) **rounds to the nearest hundred**.

### ****4. Extract the Year from Each Customer's Registration Date****

Retrieve each customer’s **registration year**.

#### **Answer:**

sql

CopyEdit

SELECT CustomerID, CustomerName, YEAR(RegistrationDate) AS RegistrationYear

FROM customers;

✔ Uses YEAR() to **extract the year** from a date.

## **🟡 MEDIUM LEVEL QUESTIONS**

### ****5. Display Each Employee's Full Name in "LASTNAME, FIRSTNAME" Format****

Concatenate each employee's first and last name in the format **"LastName, FirstName"**.

#### **Answer:**

sql

CopyEdit

SELECT EmployeeID, CONCAT(LastName, ', ', FirstName) AS FullName

FROM employees;

✔ Uses CONCAT() for **string merging**.

### ****6. Find the Average Order Value, Rounded to 2 Decimal Places****

Calculate the **average order total** and round it to **2 decimal places**.

#### **Answer:**

sql

CopyEdit

SELECT ROUND(AVG(TotalAmount), 2) AS AvgOrderValue

FROM orders;

✔ Uses AVG() for **averaging**.  
✔ Uses ROUND() for **decimal precision**.

### ****7. Retrieve Orders Where Customers Have the Same First and Last Name****

Find orders placed by customers who **have identical first and last names**.

#### **Answer:**

sql

CopyEdit

SELECT CustomerID, FirstName, LastName, OrderID, TotalAmount

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

WHERE FirstName = LastName;

✔ Uses **self-comparison (FirstName = LastName)**.

### ****8. Find Employees Who Were Hired on a Leap Year****

List employees whose **hire date falls in a leap year**.

#### **Answer:**

sql

CopyEdit

SELECT EmployeeID, EmployeeName, HireDate

FROM employees

WHERE MOD(YEAR(HireDate), 4) = 0

AND (MOD(YEAR(HireDate), 100) <> 0 OR MOD(YEAR(HireDate), 400) = 0);

✔ Uses MOD() for **leap year logic**.

## **🔴 HARD LEVEL QUESTIONS**

### ****9. Retrieve Customers Whose Names Contain Exactly 3 Vowels****

Find customers whose **first names contain exactly three vowels (A, E, I, O, U)**.

#### **Answer:**

sql

CopyEdit

SELECT CustomerID, FirstName

FROM customers

WHERE LENGTH(FirstName) - LENGTH(REPLACE(LOWER(FirstName), 'a', '')) +

LENGTH(FirstName) - LENGTH(REPLACE(LOWER(FirstName), 'e', '')) +

LENGTH(FirstName) - LENGTH(REPLACE(LOWER(FirstName), 'i', '')) +

LENGTH(FirstName) - LENGTH(REPLACE(LOWER(FirstName), 'o', '')) +

LENGTH(FirstName) - LENGTH(REPLACE(LOWER(FirstName), 'u', '')) = 3;

✔ Uses **LENGTH() and REPLACE()** to count vowels.

### ****10. Find the Customer Who Placed the Most Orders in Each Month of the Year****

For each month, return the **customer who placed the most orders**.

#### **Answer:**

sql

CopyEdit

SELECT DISTINCT MONTH(OrderDate) AS OrderMonth, CustomerID, COUNT(OrderID) AS OrderCount

FROM orders

GROUP BY OrderMonth, CustomerID

HAVING OrderCount = (

SELECT MAX(OrderCount)

FROM (

SELECT CustomerID, COUNT(OrderID) AS OrderCount

FROM orders

WHERE MONTH(OrderDate) = OrderMonth

GROUP BY CustomerID

) AS MonthlyCounts

);

✔ Uses **GROUP BY**, **HAVING**, and **subqueries**.

## **🟣 SUPER HARD LEVEL QUESTIONS**

### ****11. Find the Running Total of Each Employee's Sales, Ordered by Date****

For each **employee**, calculate the **running total** of sales over time.

#### **Answer:**

sql

CopyEdit

SELECT EmployeeID, OrderDate, SUM(TotalAmount)

OVER (PARTITION BY EmployeeID ORDER BY OrderDate) AS RunningTotal

FROM orders;

✔ Uses SUM() OVER() for **windowed running totals**.

### ****12. Find Customers Who Spent More in the Last 3 Months Than the 3 Months Before That****

Find customers who **spent more in the last three months than they did in the three months before that**.

#### **Answer:**

sql

CopyEdit

SELECT CustomerID

FROM (

SELECT CustomerID,

SUM(CASE WHEN OrderDate >= DATE\_SUB(CURDATE(), INTERVAL 3 MONTH) THEN TotalAmount ELSE 0 END) AS Last3Months,

SUM(CASE WHEN OrderDate < DATE\_SUB(CURDATE(), INTERVAL 3 MONTH) AND OrderDate >= DATE\_SUB(CURDATE(), INTERVAL 6 MONTH) THEN TotalAmount ELSE 0 END) AS Prev3Months

FROM orders

GROUP BY CustomerID

) AS SpendingComparison

WHERE Last3Months > Prev3Months;

✔ Uses CASE, DATE\_SUB(), and **grouped filtering**.

### ****13. Find Employees Who Earn More Than the Average Salary of Their Job Title****

Find employees whose **salary is greater than the average salary of their job title**.

#### **Answer:**

sql

CopyEdit

SELECT EmployeeID, EmployeeName, Salary, JobTitle

FROM employees e1

WHERE Salary > (

SELECT AVG(Salary)

FROM employees e2

WHERE e1.JobTitle = e2.JobTitle

);

✔ Uses **correlated subquery** for **grouped comparison**.

### ****14. Find Products That Have Been Ordered More in the Last Month Than Any Previous Month****

Find products whose **last month’s order quantity was higher than any previous month**.

#### **Answer:**

sql

CopyEdit

SELECT ProductID, SUM(Quantity) AS LastMonthQuantity

FROM orderdetails

WHERE OrderDate >= DATE\_SUB(CURDATE(), INTERVAL 1 MONTH)

GROUP BY ProductID

HAVING LastMonthQuantity > ALL (

SELECT SUM(Quantity)

FROM orderdetails

WHERE OrderDate < DATE\_SUB(CURDATE(), INTERVAL 1 MONTH)

GROUP BY ProductID

);

✔ Uses HAVING and **ALL()** for **comparisons across time periods**.

### ****Function: Calculate Yearly Income Tax Based on Monthly Salary****

This function will:

* Accept **monthly salary** as input.
* Convert it to **yearly salary**.
* Apply **progressive tax slabs** and return the total tax.

sql

CopyEdit

DELIMITER //

CREATE FUNCTION calculateIncomeTax(monthlySalary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

DETERMINISTIC

BEGIN

DECLARE yearlySalary DECIMAL(10,2);

DECLARE tax DECIMAL(10,2) DEFAULT 0;

-- Convert Monthly Salary to Yearly Salary

SET yearlySalary = monthlySalary \* 12;

-- Apply Tax Slabs (Hypothetical Rates)

IF yearlySalary > 3000000 THEN

SET tax = tax + (yearlySalary - 3000000) \* 0.25;

SET yearlySalary = 3000000;

END IF;

IF yearlySalary > 1500000 THEN

SET tax = tax + (yearlySalary - 1500000) \* 0.20;

SET yearlySalary = 1500000;

END IF;

IF yearlySalary > 700000 THEN

SET tax = tax + (yearlySalary - 700000) \* 0.15;

SET yearlySalary = 700000;

END IF;

IF yearlySalary > 300000 THEN

SET tax = tax + (yearlySalary - 300000) \* 0.10;

END IF;

RETURN tax;

END //

DELIMITER ;

#### **✅ How to Use**

sql

CopyEdit

SELECT calculateIncomeTax(30000); -- If monthly salary is 30,000 BDT

### ****2️⃣ Function: Calculate Net Salary After Tax****

This function calculates the net salary after deducting income tax.

sql

CopyEdit

DELIMITER //

CREATE FUNCTION calculateNetSalary(monthlySalary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

DETERMINISTIC

BEGIN

DECLARE yearlySalary DECIMAL(10,2);

DECLARE tax DECIMAL(10,2);

DECLARE netYearlySalary DECIMAL(10,2);

DECLARE netMonthlySalary DECIMAL(10,2);

-- Calculate yearly salary

SET yearlySalary = monthlySalary \* 12;

-- Get the tax amount

SET tax = calculateIncomeTax(monthlySalary);

-- Deduct tax from salary

SET netYearlySalary = yearlySalary - tax;

SET netMonthlySalary = netYearlySalary / 12;

RETURN netMonthlySalary;

END //

DELIMITER ;

#### **✅ How to Use**

sql

CopyEdit

SELECT calculateNetSalary(50000); -- If monthly salary is 50,000 BDT

### ****3️⃣ Function: Find Employees Who Pay the Highest Tax in Each City****

This function finds the employee who pays the highest tax in each city.

sql

CopyEdit

DELIMITER //

CREATE FUNCTION highestTaxPayer(cityName VARCHAR(255))

RETURNS VARCHAR(255)

DETERMINISTIC

BEGIN

DECLARE empName VARCHAR(255);

SELECT CONCAT(FirstName, ' ', LastName)

INTO empName

FROM employees

WHERE City = cityName

ORDER BY calculateIncomeTax(MonthlySalary) DESC

LIMIT 1;

RETURN empName;

END //

DELIMITER ;

#### **✅ How to Use**

sql

CopyEdit

SELECT highestTaxPayer('Bashundhara');

### ****4️⃣ Function: Compare Salaries of Employees in Different Areas****

This function checks if employees in **Bashundhara** earn more than those in **Bailey Road** on average.

sql

CopyEdit

DELIMITER //

CREATE FUNCTION compareSalary()

RETURNS VARCHAR(255)

DETERMINISTIC

BEGIN

DECLARE avgBashundhara DECIMAL(10,2);

DECLARE avgBaileyRoad DECIMAL(10,2);

DECLARE result VARCHAR(255);

-- Get average salaries

SELECT AVG(MonthlySalary) INTO avgBashundhara FROM employees WHERE City = 'Bashundhara';

SELECT AVG(MonthlySalary) INTO avgBaileyRoad FROM employees WHERE City = 'Bailey Road';

-- Compare salaries

IF avgBashundhara > avgBaileyRoad THEN

SET result = 'Employees in Bashundhara earn more on average.';

ELSEIF avgBashundhara < avgBaileyRoad THEN

SET result = 'Employees in Bailey Road earn more on average.';

ELSE

SET result = 'Employees in both areas have the same average salary.';

END IF;

RETURN result;

END //

DELIMITER ;

#### **✅ How to Use**

sql

CopyEdit

SELECT compareSalary();

### ****5️⃣ Super Hard Function: Find Top 3 Cities Where Employees Pay the Most Tax on Average****

This function returns a **comma-separated list** of the **top 3 cities** where employees pay the highest tax on average.

sql

CopyEdit

DELIMITER //

CREATE FUNCTION topTaxPayingCities()

RETURNS VARCHAR(255)

DETERMINISTIC

BEGIN

DECLARE cityList VARCHAR(255);

SELECT GROUP\_CONCAT(City ORDER BY AVG(calculateIncomeTax(MonthlySalary)) DESC SEPARATOR ', ')

INTO cityList

FROM employees

GROUP BY City

ORDER BY AVG(calculateIncomeTax(MonthlySalary)) DESC

LIMIT 3;

RETURN cityList;

END //

DELIMITER ;

#### **✅ How to Use**

sql

CopyEdit

SELECT topTaxPayingCities();

### ****6️⃣ Super Hard Function: Rank Employees Based on Tax and Show Their Salary Category****

This function assigns **rank** to employees based on their tax amount and also categorizes them as **Low, Medium, or High Earners**.

sql

CopyEdit

DELIMITER //

CREATE FUNCTION employeeTaxRank(empID INT)

RETURNS VARCHAR(255)

DETERMINISTIC

BEGIN

DECLARE rankNum INT;

DECLARE category VARCHAR(50);

-- Assign rank based on tax amount

SELECT RANK() OVER (ORDER BY calculateIncomeTax(MonthlySalary) DESC)

INTO rankNum

FROM employees

WHERE EmployeeID = empID;

-- Categorize salary

IF MonthlySalary < 30000 THEN

SET category = 'Low Earner';

ELSEIF MonthlySalary BETWEEN 30000 AND 70000 THEN

SET category = 'Medium Earner';

ELSE

SET category = 'High Earner';

END IF;

RETURN CONCAT('Rank: ', rankNum, ', Category: ', category);

END //

DELIMITER ;

#### **✅ How to Use**

sql

CopyEdit

SELECT employeeTaxRank(102);

## **🔍 Summary**

| **SQL Function** | **Purpose** |
| --- | --- |
| calculateIncomeTax(monthlySalary) | Calculates **yearly tax** based on salary. |
| calculateNetSalary(monthlySalary) | Finds **monthly salary after tax** deduction. |
| highestTaxPayer(cityName) | Finds **highest taxpayer in a given city**. |
| compareSalary() | Compares **salaries in Bashundhara & Bailey Road**. |
| topTaxPayingCities() | Returns **top 3 cities where employees pay the most tax**. |
| employeeTaxRank(empID) | Ranks employees by tax and categorizes their salary level. |

### ****1️⃣ Function: Calculate Yearly Income Tax Based on Monthly Salary****

This function calculates the **income tax** of an employee based on their **monthly salary**. We multiply the salary by 12 to convert it to a **yearly salary** and then apply **progressive tax slabs** to calculate the total tax.

sql

CopyEdit

DELIMITER //

CREATE FUNCTION calculateIncomeTax(monthlySalary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

DETERMINISTIC

BEGIN

DECLARE yearlySalary DECIMAL(10,2);

DECLARE tax DECIMAL(10,2) DEFAULT 0;

-- Convert Monthly Salary to Yearly Salary

SET yearlySalary = monthlySalary \* 12;

-- Apply Tax Slabs (Hypothetical Rates)

IF yearlySalary > 3000000 THEN

SET tax = tax + (yearlySalary - 3000000) \* 0.25;

SET yearlySalary = 3000000;

END IF;

IF yearlySalary > 1500000 THEN

SET tax = tax + (yearlySalary - 1500000) \* 0.20;

SET yearlySalary = 1500000;

END IF;

IF yearlySalary > 700000 THEN

SET tax = tax + (yearlySalary - 700000) \* 0.15;

SET yearlySalary = 700000;

END IF;

IF yearlySalary > 300000 THEN

SET tax = tax + (yearlySalary - 300000) \* 0.10;

END IF;

RETURN tax;

END //

DELIMITER ;

### ****Explanation of Each Part****

#### **1. DELIMITER //**

sql

CopyEdit

DELIMITER //

* **Purpose:**
  + By default, SQL uses the **semicolon (;)** to end commands. However, when defining a **stored procedure** or function, we may want to use semicolons inside the function. This is where **DELIMITER** comes in.
  + **Why:** It changes the statement delimiter temporarily to //, so the SQL engine knows when the function definition ends (instead of using a semicolon).
  + **Alternative:** You could choose any delimiter, such as $$ or //, but // is commonly used.

#### **2. CREATE FUNCTION**

sql

CopyEdit

CREATE FUNCTION calculateIncomeTax(monthlySalary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

DETERMINISTIC

* **Purpose:**
  + This defines the function **calculateIncomeTax**. It takes **monthlySalary** (a decimal number) as input and returns a decimal value (the calculated tax).
  + **RETURNS DECIMAL(10,2):** The function will output a **decimal value** (the total tax) with 10 digits, of which 2 are after the decimal point. The total value represents the calculated tax.
  + **DETERMINISTIC:** This means that for the same input (monthly salary), the function will always return the same result. It's a type of optimization hint to the database system.
  + **Alternative:** You could use VOLATILE if the result might change based on the environment (though this is uncommon in most tax scenarios).

#### **3. DECLARE Statements**

sql

CopyEdit

DECLARE yearlySalary DECIMAL(10,2);

DECLARE tax DECIMAL(10,2) DEFAULT 0;

* **Purpose:**
  + We declare two **variables**:
    - **yearlySalary** stores the annual salary after multiplying the monthly salary by 12.
    - **tax** stores the tax value, starting with 0 (default). This will hold the cumulative tax as we apply the tax slabs.
  + **Why:** Declaring variables is necessary to perform calculations in SQL functions.

#### **4. SET Statements**

sql

CopyEdit

SET yearlySalary = monthlySalary \* 12;

* **Purpose:**
  + This line multiplies the **monthlySalary** by 12 to calculate the **yearly salary**.
  + **Why:** We need yearly salary to apply tax slabs, as taxes are often calculated on annual income.
  + **Alternative:** You could perform this calculation directly in the **IF statements** without storing it in a variable, but this would make the code less readable.

#### **5. IF Statements (Tax Slabs)**

sql

CopyEdit

IF yearlySalary > 3000000 THEN

SET tax = tax + (yearlySalary - 3000000) \* 0.25;

SET yearlySalary = 3000000;

END IF;

* **Purpose:**
  + This applies **progressive tax rates** based on yearly salary.
  + For example:
    - If the salary is above **3,000,000**, we charge **25%** on the amount above that threshold and reduce the **yearlySalary** to 3,000,000 to apply the next slab.
  + The process repeats for each tax bracket, reducing the taxable salary after each condition.
  + **Why:** It mimics the real-world tax system, where different portions of the salary are taxed at different rates.
  + **Alternative:** You could use a **CASE** statement, which is a little more concise for handling multiple conditions.

#### **6. RETURN Statement**

sql

CopyEdit

RETURN tax;

* **Purpose:**
  + The **RETURN** statement returns the **calculated tax** value. Once the function completes all the calculations, it provides the result back to the caller.
  + **Why:** It's necessary to output the result from the function.
  + **Alternative:** You could store the result in a temporary table and select from it, but returning directly is more efficient.

#### **7. DELIMITER ;**

sql

CopyEdit

DELIMITER ;

* **Purpose:**
  + Resets the **delimiter** back to the default semicolon (;) after defining the function.
  + **Why:** To ensure that the rest of the queries in SQL use the default delimiter.
  + **Alternative:** This is just a housekeeping step to return the delimiter to normal. It's essential to do this after defining a stored procedure or function to avoid issues with future queries.

### ****How to Use the Function:****

Once the function is created, you can call it like this:

sql

CopyEdit

SELECT calculateIncomeTax(30000); -- If monthly salary is 30,000 BDT

This will return the **yearly income tax** based on the monthly salary provided (in this case, **30,000 BDT**).

### ****Why is Each Command Necessary?****

1. **DELIMITER**: This is needed to temporarily change the SQL statement terminator. We do it to avoid conflicts with semicolons used inside the function.
2. **CREATE FUNCTION**: Defines the custom function that you can reuse in queries. Without it, there is no way to store reusable SQL logic.
3. **DECLARE**: You need to declare variables to store intermediate results (like yearly salary and tax).
4. **SET**: These statements assign values to the declared variables to calculate tax.
5. **IF Statements**: They apply **tax brackets** to calculate the tax progressively, depending on the yearly salary.
6. **RETURN**: It sends the result (calculated tax) back to the caller, completing the function.
7. **DELIMITER ;**: Resets the delimiter back to the default for normal queries.

### ****Alternative Ways to Implement the Tax Calculation:****

1. **Using a CASE Statement Instead of Multiple IFs:** Instead of multiple IF statements for tax brackets, you could use a CASE statement for a more compact approach:

sql

CopyEdit

SET tax = CASE

WHEN yearlySalary > 3000000 THEN (yearlySalary - 3000000) \* 0.25 + (yearlySalary - 1500000) \* 0.20

WHEN yearlySalary > 1500000 THEN (yearlySalary - 1500000) \* 0.20 + (yearlySalary - 700000) \* 0.15

ELSE (yearlySalary - 300000) \* 0.10

END;

1. **Using a Temporary Table**: Instead of returning the result directly, you could store the result in a **temporary table** and then select from it, though returning directly is simpler and more efficient in most cases.

**Question:**  
You have been asked to find the **total number of orders** placed by each **customer**, considering the order amount per product (Quantity \* Price). Group the results by CustomerID and order the customers by **total order value** in descending order. Also, partition the result set by Country to show the order value for each customer within their respective country.

**Answer:**

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName, SUM(od.Quantity \* p.Price) AS TotalOrderValue, c.Country

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

JOIN orderdetails od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

GROUP BY c.CustomerID, c.Country

ORDER BY TotalOrderValue DESC;

**Explanation:**

* **JOIN**: Joins tables customers, orders, orderdetails, and products to calculate the total order value by multiplying Quantity and Price.
* **GROUP BY**: Groups the data by CustomerID and Country to get totals for each customer.
* **ORDER BY**: Orders the result by TotalOrderValue in descending order to show customers with the highest order value first.

**2️⃣**

**Question:**  
Write a query to find the **average salary** of employees grouped by their JobTitle. Additionally, include the **highest salary** for each JobTitle in your result. Sort the results by **average salary** in ascending order. Partition the results by Location to compare the average salaries of employees in different locations for the same job title.

**Answer:**

sql

CopyEdit

SELECT j.JobTitle, p.Location, AVG(p.Salary) AS AvgSalary, MAX(p.Salary) AS MaxSalary

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

GROUP BY j.JobTitle, p.Location

ORDER BY AvgSalary ASC;

**Explanation:**

* **JOIN**: Joins persons and jobs to get the job titles for employees.
* **GROUP BY**: Groups by JobTitle and Location to calculate the average and maximum salary.
* **ORDER BY**: Orders the result by AvgSalary in ascending order to show lower salaries first.

**3️⃣**

**Question:**  
Using the persons table, write a query to find the **top 3 highest-paid employees** for each job title. For each job title, partition the result set by Location and order the employees by Salary in **descending** order.

**Answer:**

sql

CopyEdit

SELECT PersonID, FirstName, LastName, JobTitle, Salary, Location,

ROW\_NUMBER() OVER (PARTITION BY JobTitle, Location ORDER BY Salary DESC) AS Rank

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE Rank <= 3

ORDER BY JobTitle, Location, Rank;

**Explanation:**

* **ROW\_NUMBER()**: Partitions the results by JobTitle and Location and ranks employees based on their Salary within each partition.
* **WHERE Rank <= 3**: Filters the results to get only the top 3 highest-paid employees for each job title.
* **ORDER BY**: Orders the final result set by JobTitle, Location, and Rank.

**4️⃣**

**Question:**  
Write a query to calculate the **total sales value** (Quantity \* Price) for each **product** sold in the last month, and group the results by ProductName. Additionally, **order** the products by TotalSales in descending order. Use the HAVING clause to exclude products where total sales are less than 5000.

**Answer:**

sql

CopyEdit

SELECT p.ProductName, SUM(od.Quantity \* pr.Price) AS TotalSales

FROM products p

JOIN orderdetails od ON p.ProductID = od.ProductID

JOIN orders o ON od.OrderID = o.OrderID

JOIN products pr ON od.ProductID = pr.ProductID

WHERE o.OrderDate >= CURDATE() - INTERVAL 1 MONTH

GROUP BY p.ProductName

HAVING TotalSales >= 5000

ORDER BY TotalSales DESC;

**Explanation:**

* **JOIN**: Joins products, orderdetails, and orders to get sales information.
* **WHERE**: Filters sales made in the last month.
* **GROUP BY**: Groups the results by ProductName.
* **HAVING**: Filters out products with total sales less than 5000.
* **ORDER BY**: Orders the results by TotalSales in descending order to show the most popular products.

**5️⃣**

**Question:**  
Write an SQL query to find the **average salary** for each **job title** from the persons table. Additionally, partition the result set by Location and **order the results** by the **average salary** in descending order. Only include job titles where the average salary is greater than 50,000.

**Answer:**

sql

CopyEdit

SELECT j.JobTitle, p.Location, AVG(p.Salary) AS AvgSalary

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

GROUP BY j.JobTitle, p.Location

HAVING AVG(p.Salary) > 50000

ORDER BY AvgSalary DESC;

**Explanation:**

* **JOIN**: Joins persons with jobs to retrieve job titles for employees.
* **GROUP BY**: Groups the results by JobTitle and Location to calculate average salaries.
* **HAVING**: Filters job titles where the average salary is greater than 50,000.
* **ORDER BY**: Orders the results by average salary in descending order.

**6️⃣**

**Question:**  
Write a query to calculate the **total number of orders** made by each **customer**. Additionally, use the HAVING clause to only include customers who have placed more than 2 orders in total. Sort the results by the **total number of orders** in descending order.

**Answer:**

sql

CopyEdit

SELECT c.CustomerID, c.CustomerName, COUNT(o.OrderID) AS TotalOrders

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID

HAVING COUNT(o.OrderID) > 2

ORDER BY TotalOrders DESC;

**Explanation:**

* **JOIN**: Joins customers and orders to get the orders placed by each customer.
* **GROUP BY**: Groups the results by CustomerID to calculate the total number of orders.
* **HAVING**: Filters out customers who have placed 2 or fewer orders.
* **ORDER BY**: Orders the results by the total number of orders in descending order.

**7️⃣**

**Question:**  
Using the orderdetails and products tables, find the **total quantity** sold for each product where the ProductName starts with "Rice". Group the results by ProductName and **order** them by **total quantity** sold in ascending order.

**Answer:**

sql

CopyEdit

SELECT p.ProductName, SUM(od.Quantity) AS TotalQuantitySold

FROM orderdetails od

JOIN products p ON od.ProductID = p.ProductID

WHERE p.ProductName LIKE 'Rice%'

GROUP BY p.ProductName

ORDER BY TotalQuantitySold ASC;

**Explanation:**

* **JOIN**: Joins orderdetails and products to calculate the quantity sold for each product.
* **WHERE**: Filters products whose name starts with "Rice".
* **GROUP BY**: Groups by ProductName to calculate the total quantity sold for each product.
* **ORDER BY**: Orders the results by total quantity sold in ascending order.

**8️⃣**

**Question:**  
Write a query to find the **average order quantity** for each product sold by CustomerID 101 in the last 6 months. Also, partition the result set by ProductID and **order** the results by ProductName in **alphabetical** order.

**Answer:**

sql

CopyEdit

SELECT p.ProductName, AVG(od.Quantity) AS AvgOrderQuantity

FROM orderdetails od

JOIN products p ON od.ProductID = p.ProductID

JOIN orders o ON od.OrderID = o.OrderID

WHERE o.CustomerID = 101 AND o.OrderDate >= CURDATE() - INTERVAL 6 MONTH

GROUP BY p.ProductName

ORDER BY p.ProductName ASC;

**Explanation:**

* **JOIN**: Joins orderdetails, products, and orders to get the necessary details for calculating the average order quantity.
* **WHERE**: Filters orders placed by CustomerID 101 in the last 6 months.
* **GROUP BY**: Groups the results by ProductName to calculate the average order quantity per product.
* **ORDER BY**: Orders the results alphabetically by ProductName.

### ****1️⃣**** Retrieve data from tables

**Question:**  
Write a query to retrieve the **first name**, **last name**, **job title**, and **salary** of all employees working in the **Marketing** department. Include only those employees whose salary is above 50000. Sort the results by **salary** in ascending order.

**Answer:**

sql

CopyEdit

SELECT p.FirstName, p.LastName, j.JobTitle, p.Salary

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE j.Department = 'Marketing' AND p.Salary > 50000

ORDER BY p.Salary ASC;

### ****2️⃣**** Boolean and Relational Operators

**Question:**  
Write a query to find the **full names** of employees who are either in the **Sales** department and have a salary greater than 60000, or are in the **HR** department and have been working for more than 5 years.

**Answer:**

sql

CopyEdit

SELECT p.FirstName, p.LastName

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE (j.Department = 'Sales' AND p.Salary > 60000)

OR (j.Department = 'HR' AND p.YearsWorked > 5);

### ****3️⃣**** Wildcard and Special Operators

**Question:**  
Write a query to retrieve the **email addresses** of employees whose job titles start with **"Manager"** and whose email ends with **"example.com"**.

**Answer:**

sql

CopyEdit

SELECT p.Email

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE j.JobTitle LIKE 'Manager%' AND p.Email LIKE '%example.com';

### ****4️⃣**** Aggregate Functions

**Question:**  
Write a query to find the **average salary** for employees in the **Finance** department. Include only employees with a salary above 40000. Group the results by **department**.

**Answer:**

sql

CopyEdit

SELECT j.Department, AVG(p.Salary) AS AvgSalary

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE j.Department = 'Finance' AND p.Salary > 40000

GROUP BY j.Department;

### ****5️⃣**** Formatting query output

**Question:**  
Write a query to format the **salary** of all employees in the **IT** department to two decimal places and display it with a **USD** symbol.

**Answer:**

sql

CopyEdit

SELECT p.FirstName, p.LastName, CONCAT('$', FORMAT(p.Salary, 2)) AS Salary

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE j.Department = 'IT';

### ****6️⃣**** Query on Multiple Tables

**Question:**  
Write a query to retrieve the **name**, **category**, and **price** of all products along with the **order ID** and **quantity** of each product in the last month. Only include orders where the total value is greater than 1000.

**Answer:**

sql

CopyEdit

SELECT p.ProductName, p.Category, p.Price, o.OrderID, od.Quantity

FROM products p

JOIN orderdetails od ON p.ProductID = od.ProductID

JOIN orders o ON od.OrderID = o.OrderID

WHERE o.OrderDate >= DATEADD(MONTH, -1, GETDATE())

AND (od.Quantity \* p.Price) > 1000;

### ****7️⃣**** Advanced SQL Features

**Question:**  
Write a query to retrieve the **names** of customers who have placed orders that include products from **both** the **"Beverages"** and **"Snacks"** categories. Use **HAVING** to filter the results.

**Answer:**

sql

CopyEdit

SELECT c.CustomerName

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

JOIN orderdetails od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

WHERE p.Category IN ('Beverages', 'Snacks')

GROUP BY c.CustomerName

HAVING COUNT(DISTINCT p.Category) = 2;

### ****8️⃣**** SQL JOINS

**Question:**  
Write a query to retrieve the **name** of each **customer**, along with the **total value** of their orders (Quantity \* Price), sorted by **total value** in descending order. Include only customers who have placed more than 5 orders.

**Answer:**

sql

CopyEdit

SELECT c.CustomerName, SUM(od.Quantity \* p.Price) AS TotalValue

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

JOIN orderdetails od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

GROUP BY c.CustomerName

HAVING COUNT(o.OrderID) > 5

ORDER BY TotalValue DESC;

### ****9️⃣**** Triggers

**Question:**  
Write a trigger that automatically updates the **stock quantity** of a product when an order is placed. If the **stock quantity** goes below 5, an alert should be sent to the manager.

**Answer:**

sql

CopyEdit

CREATE TRIGGER UpdateStockOnOrder

AFTER INSERT ON orderdetails

FOR EACH ROW

BEGIN

UPDATE products

SET StockQuantity = StockQuantity - NEW.Quantity

WHERE ProductID = NEW.ProductID;

IF (SELECT StockQuantity FROM products WHERE ProductID = NEW.ProductID) < 5 THEN

-- Send an alert to the manager (pseudocode for the alert)

CALL SendAlertToManager(NEW.ProductID);

END IF;

END;

### ****🔟**** Window Functions

**Question:**  
Write a query to find the **rank** of each employee’s salary within their **job title**. Rank employees in each job title by **salary** in descending order.

**Answer:**

sql

CopyEdit

SELECT p.FirstName, p.LastName, j.JobTitle, p.Salary,

RANK() OVER (PARTITION BY j.JobTitle ORDER BY p.Salary DESC) AS SalaryRank

FROM persons p

JOIN jobs j ON p.JobID = j.JobID;

### ****1️⃣1️⃣**** Retrieve data from tables

**Question:**  
Write a query to find the **total number of orders** placed by customers in **New York**. Group the results by **customer** and sort by the **total number of orders**.

**Answer:**

sql

CopyEdit

SELECT c.CustomerName, COUNT(o.OrderID) AS TotalOrders

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

WHERE c.City = 'New York'

GROUP BY c.CustomerName

ORDER BY TotalOrders DESC;

### ****1️⃣2️⃣**** Boolean and Relational Operators

**Question:**  
Write a query to retrieve the **names** of employees who have either been with the company for more than 10 years and earn more than 70000, or are in the **Marketing** department and earn above 50000.

**Answer:**

sql

CopyEdit

SELECT p.FirstName, p.LastName

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE (p.YearsWorked > 10 AND p.Salary > 70000)

OR (j.Department = 'Marketing' AND p.Salary > 50000);

### ****1️⃣3️⃣**** Wildcard and Special Operators

**Question:**  
Write a query to find all **customer names** whose names contain the word **"John"** and whose email addresses contain **"gmail.com"**.

**Answer:**

sql

CopyEdit

SELECT c.CustomerName

FROM customers c

WHERE c.CustomerName LIKE '%John%' AND c.Email LIKE '%gmail.com';

### ****1️⃣4️⃣**** Aggregate Functions

**Question:**  
Write a query to find the **total number of orders** placed in the last quarter by customers in the **"Electronics"** category.

**Answer:**

sql

CopyEdit

SELECT COUNT(o.OrderID) AS TotalOrders

FROM orders o

JOIN orderdetails od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

WHERE p.Category = 'Electronics' AND o.OrderDate >= DATEADD(QUARTER, -1, GETDATE());

### ****1️⃣5️⃣**** Formatting query output

**Question:**  
Write a query to display the **first name**, **last name**, and **salary** of all employees, where salary is displayed with a **comma separator** and **rounded to the nearest thousand**.

**Answer:**

sql

CopyEdit

SELECT p.FirstName, p.LastName, FORMAT(ROUND(p.Salary, -3), 'N0') AS Salary

FROM persons p;

### ****1️⃣6️⃣**** Query on Multiple Tables

**Question:**  
Write a query to retrieve the **product name**, **customer name**, and **quantity** of products ordered where the total price exceeds 5000, including only orders placed in the last month.

**Answer:**

sql

CopyEdit

SELECT p.ProductName, c.CustomerName, od.Quantity

FROM products p

JOIN orderdetails od ON p.ProductID = od.ProductID

JOIN orders o ON od.OrderID = o.OrderID

JOIN customers c ON o.CustomerID = c.CustomerID

WHERE (od.Quantity \* p.Price) > 5000 AND o.OrderDate >= DATEADD(MONTH, -1, GETDATE());

### ****1️⃣7️⃣**** Advanced SQL Features

**Question:**  
Write a query to find the **products** with the highest **sales** within the **"Snacks"** category. Only include products with **sales** greater than 10000.

**Answer:**

sql

CopyEdit

SELECT p.ProductName, SUM(od.Quantity \* p.Price) AS TotalSales

FROM products p

JOIN orderdetails od ON p.ProductID = od.ProductID

WHERE p.Category = 'Snacks'

GROUP BY p.ProductName

HAVING SUM(od.Quantity \* p.Price) > 10000

ORDER BY TotalSales DESC

LIMIT 1;

### ****1️⃣8️⃣**** SQL JOINS

**Question:**  
Write a query to retrieve the **employee name**, **department**, and **number of products** they manage. Only include employees who manage more than 3 products in the last month.

**Answer:**

sql

CopyEdit

SELECT p.FirstName, p.LastName, j.Department, COUNT(p.ProductID) AS NumberOfProducts

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

JOIN products pr ON pr.ManagerID = p.PersonID

WHERE pr.AddedDate >= DATEADD(MONTH, -1, GETDATE())

GROUP BY p.FirstName, p.LastName, j.Department

HAVING COUNT(pr.ProductID) > 3;

### ****1️⃣9️⃣**** Triggers

**Question:**  
Create a trigger that updates the **product stock** whenever a new order is placed. If the stock goes below 5, send an alert to the manager.

**Answer:**

sql

CopyEdit

CREATE TRIGGER UpdateProductStock

AFTER INSERT ON orderdetails

FOR EACH ROW

BEGIN

UPDATE products

SET StockQuantity = StockQuantity - NEW.Quantity

WHERE ProductID = NEW.ProductID;

IF (SELECT StockQuantity FROM products WHERE ProductID = NEW.ProductID) < 5 THEN

CALL SendAlertToManager(NEW.ProductID);

END IF;

END;

### ****2️⃣0️⃣**** Window Functions

**Question:**  
Write a query to find the **running total of sales** for each product, ordered by **sales value**. Partition the results by ProductCategory.

**Answer:**

sql

CopyEdit

SELECT p.ProductName, p.Category, od.Quantity \* p.Price AS SalesValue,

SUM(od.Quantity \* p.Price) OVER (PARTITION BY p.Category ORDER BY od.OrderID) AS RunningTotalSales

FROM products p

JOIN orderdetails od ON p.ProductID = od.ProductID

ORDER BY p.Category, od.OrderID;

### ****1️⃣**** Complex Query on Multiple Tables

**Question:**  
Write a query to retrieve the **name** of each **customer**, along with the **total number of orders** and the **average order value** (Quantity \* Price) in the last 6 months. Include only those customers who have made more than 5 orders during this period. Order the results by **total orders** in descending order.

**Answer:**

sql

CopyEdit

SELECT c.CustomerName, COUNT(o.OrderID) AS TotalOrders, AVG(od.Quantity \* p.Price) AS AvgOrderValue

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

JOIN orderdetails od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

WHERE o.OrderDate >= DATEADD(MONTH, -6, GETDATE())

GROUP BY c.CustomerName

HAVING COUNT(o.OrderID) > 5

ORDER BY TotalOrders DESC;

### ****2️⃣**** Aggregate Functions with Complex Grouping

**Question:**  
Write a query to find the **department** with the highest average **salary** among employees, but only include departments that have more than 3 employees. Order the results by **average salary** in descending order.

**Answer:**

sql

CopyEdit

SELECT j.Department, AVG(p.Salary) AS AvgSalary

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

GROUP BY j.Department

HAVING COUNT(p.PersonID) > 3

ORDER BY AvgSalary DESC;

### ****3️⃣**** Window Functions for Ranking

**Question:**  
Write a query to retrieve the **first name**, **last name**, and **salary** of all employees, along with their rank in their respective **job title** based on salary. Rank employees in each job title by **salary** in descending order. Display employees in the same order.

**Answer:**

sql

CopyEdit

SELECT p.FirstName, p.LastName, p.Salary,

RANK() OVER (PARTITION BY j.JobTitle ORDER BY p.Salary DESC) AS SalaryRank

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

ORDER BY j.JobTitle, SalaryRank;

### ****4️⃣**** Triggers and Stock Update

**Question:**  
Create a trigger to automatically update the **stock quantity** in the **products** table whenever a new **order** is placed. If the stock of any product drops below 10 after an order is placed, an alert should be generated to notify the manager.

**Answer:**

sql

CopyEdit

CREATE TRIGGER UpdateStockAfterOrder

AFTER INSERT ON orderdetails

FOR EACH ROW

BEGIN

UPDATE products

SET StockQuantity = StockQuantity - NEW.Quantity

WHERE ProductID = NEW.ProductID;

IF (SELECT StockQuantity FROM products WHERE ProductID = NEW.ProductID) < 10 THEN

-- Trigger an alert to the manager (simplified as pseudocode)

CALL NotifyManager(NEW.ProductID);

END IF;

END;

### ****5️⃣**** SQL Join with Multiple Conditions

**Question:**  
Write a query to find the **total number of orders** placed by customers from **New York** who have ordered products from both the **"Snacks"** and **"Beverages"** categories. Return the **customer name** and **total number of orders**.

**Answer:**

sql

CopyEdit

SELECT c.CustomerName, COUNT(o.OrderID) AS TotalOrders

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

JOIN orderdetails od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

WHERE c.City = 'New York'

AND p.Category IN ('Snacks', 'Beverages')

GROUP BY c.CustomerName

HAVING COUNT(DISTINCT p.Category) = 2;

### ****6️⃣**** Subqueries with Aggregates

**Question:**  
Write a query to retrieve the **name** and **salary** of employees whose salary is **greater than the average salary** in their respective **department**. Include only those departments where the average salary is greater than 50000.

**Answer:**

sql

CopyEdit

SELECT p.FirstName, p.LastName, p.Salary

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE p.Salary > (

SELECT AVG(Salary)

FROM persons

WHERE JobID = p.JobID

) AND j.Department IN (

SELECT Department

FROM jobs

GROUP BY Department

HAVING AVG(Salary) > 50000

);

### ****7️⃣**** Complex Query Using GROUP BY

**Question:**  
Write a query to find the **top 3 products** with the **highest total sales** in the last quarter, and show their **product name**, **total sales**, and **category**. Sort the results by **total sales** in descending order.

**Answer:**

sql

CopyEdit

SELECT p.ProductName, SUM(od.Quantity \* p.Price) AS TotalSales, p.Category

FROM products p

JOIN orderdetails od ON p.ProductID = od.ProductID

JOIN orders o ON od.OrderID = o.OrderID

WHERE o.OrderDate >= DATEADD(QUARTER, -1, GETDATE())

GROUP BY p.ProductName, p.Category

ORDER BY TotalSales DESC

LIMIT 3;

### ****8️⃣**** Using PARTITION BY with Window Functions

**Question:**  
Write a query to calculate the **running total** of orders for each **product** in the **last 3 months**. Display the **product name**, **order date**, and **running total** of sales for each product, partitioned by **product** and ordered by **order date**.

**Answer:**

sql

CopyEdit

SELECT p.ProductName, o.OrderDate,

SUM(od.Quantity \* p.Price) OVER (PARTITION BY p.ProductName ORDER BY o.OrderDate) AS RunningTotalSales

FROM products p

JOIN orderdetails od ON p.ProductID = od.ProductID

JOIN orders o ON od.OrderID = o.OrderID

WHERE o.OrderDate >= DATEADD(MONTH, -3, GETDATE())

ORDER BY p.ProductName, o.OrderDate;

### ****9️⃣**** Join with Self-Referencing

**Question:**  
Write a query to retrieve the **manager name** and **employee name** for employees who have been working for more than 5 years. Display the **manager's name** and **employee's first name** and **last name**.

**Answer:**

sql

CopyEdit

SELECT m.FirstName AS ManagerFirstName, m.LastName AS ManagerLastName, e.FirstName AS EmployeeFirstName, e.LastName AS EmployeeLastName

FROM persons e

JOIN persons m ON e.ManagerID = m.PersonID

WHERE e.YearsWorked > 5;

### ****🔟**** Advanced Subquery and Aggregates

**Question:**  
Write a query to find the **name** and **salary** of employees who earn more than the **average salary** of employees in the same **job title** and in the **same department**. Display results ordered by **salary** in descending order.

**Answer:**

sql

CopyEdit

SELECT p.FirstName, p.LastName, p.Salary

FROM persons p

JOIN jobs j ON p.JobID = j.JobID

WHERE p.Salary > (

SELECT AVG(p2.Salary)

FROM persons p2

JOIN jobs j2 ON p2.JobID = j2.JobID

WHERE j2.Department = j.Department AND p2.JobID = p.JobID

)

ORDER BY p.Salary DESC;