```sql
-- HandsOn

-- Customers table
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    City VARCHAR(50),
    Email VARCHAR(100),
    PhoneNumber VARCHAR(15)
);

INSERT INTO Customers (CustomerID, FirstName, LastName, City, Email, PhoneNumber)
VALUES
(1, 'amit', 'sharma', 'Mumbai', 'amit.sharma@example.com', '9876543210'),
(2, 'priya', 'mehta', 'Delhi', 'priya.mehta@example.com', '8765432109'),
(3, 'rohit', 'kumar', 'Bangalore', 'rohit.kumar@example.com', '7654321098'),
(4, 'neha', 'verma', 'Mumbai', 'neha.verma@example.com', '6543210987'),
(5, 'siddharth', 'singh', 'Delhi', 'siddharth.singh@example.com', '5432109876'),
(6, 'asha', 'rao', 'Bangalore', 'asha.rao@example.com', '4321098765');

-- Orders table
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderAmount DECIMAL(10, 2),
    OrderDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);



INSERT INTO Orders (OrderID, CustomerID, OrderAmount, OrderDate) VALUES
(1, 1, 500.00, '2024-01-15'),
(2, 1, 800.00, '2024-02-20'),
(3, 2, 1200.00, '2024-03-05'),
(4, 3, 700.00, '2024-01-25'),
(5, 4, 300.00, '2024-02-10'),
(6, 5, 1500.00, '2024-03-15'),
(7, 6, 400.00, '2024-01-30'),
(8, 3, 600.00, '2024-03-05'),
(9, 2, 500.00, '2024-01-18');
```

```sql
-- 1. Filter and Aggregate on Join Results using SQL
-- Task: Join the `Orders` and `Customers` tables to find the total order amount
per customer and filter out customers who have spent less than $1,000.

SELECT c.CustomerID, c.FirstName, c.LastName, SUM(o.OrderAmount) AS TotalSpent
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerID, c.FirstName, c.LastName
HAVING SUM(o.OrderAmount) >= 1000;


-- 2. Cumulative Aggregations and Ranking in SQL Queries
-- Task: Create a cumulative sum of the `OrderAmount` for each customer to track
the running total of how much each customer has spent.

SELECT c.CustomerID, c.FirstName, c.LastName, o.OrderAmount,SUM(o.OrderAmount)
OVER (PARTITION BY c.CustomerID ORDER BY o.OrderDate) AS RunningTotal
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
ORDER BY c.CustomerID, o.OrderDate;


-- 3. OVER and PARTITION BY Clause in SQL Queries
-- Task: Rank the customers based on the total amount they have spent,
partitioned by city.

SELECT c.CustomerID, c.FirstName, c.LastName, c.City, SUM(o.OrderAmount) AS
TotalSpent, RANK() OVER (PARTITION BY c.City ORDER BY SUM(o.OrderAmount) DESC) AS
RankInCity
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerID, c.FirstName, c.LastName, c.City;

-- 4. Total Aggregation using OVER and PARTITION BY in SQL Queries
-- Task: Calculate the total amount of all orders (overall total) and the
percentage each customers total spending contributes to the overall total.

WITH CustomerTotals AS (
    SELECT c.CustomerID, c.FirstName, c.LastName, SUM(o.OrderAmount) AS
TotalSpent
    FROM Customers c
```

```
    JOIN Orders o ON c.CustomerID = o.CustomerID
    GROUP BY c.CustomerID, c.FirstName, c.LastName
)
SELECT CustomerID, FirstName, LastName, TotalSpent,
       SUM(TotalSpent) OVER () AS OverallTotal,
       ROUND((TotalSpent / SUM(TotalSpent) OVER ()) * 100, 2) AS
PercentageOfTotal
FROM CustomerTotals;


-- 5. Ranking in SQL
-- Task: Rank all customers based on the total amount they have spent, without
partitioning.

SELECT c.CustomerID, c.FirstName, c.LastName, SUM(o.OrderAmount) AS TotalSpent,
       RANK() OVER (ORDER BY SUM(o.OrderAmount) DESC) AS OverallRank
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerID, c.FirstName, c.LastName;


Here are additional tasks that build on the concepts of filtering, aggregating,
ranking, and window functions in SQL:

-- 6. Task: Calculate the Average Order Amount per City
-- Task: Write a query that joins the `Orders` and `Customers` tables, calculates
the average order amount for each city, and orders the results by the average
amount in descending order.

SELECT c.City, AVG(o.OrderAmount) AS AverageOrderAmount
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.City
ORDER BY AverageOrderAmount DESC;

-- 7. Task: Find Top N Customers by Total Spending
-- Task: Write a query to find the top 3 customers who have spent the most, using
`ORDER BY` and `LIMIT`.

SELECT TOP 3 c.CustomerID, c.FirstName, c.LastName, SUM(o.OrderAmount) AS
TotalSpent
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
```

```sql
GROUP BY c.CustomerID, c.FirstName, c.LastName
ORDER BY TotalSpent DESC;

-- 8. Task: Calculate Yearly Order Totals
--- Task: Write a query that groups orders by year (using `OrderDate`),
calculates the total amount of orders for each year, and orders the results by
year.

SELECT YEAR(o.OrderDate) AS OrderYear, SUM(o.OrderAmount) AS YearlyTotal
FROM Orders o
GROUP BY YEAR(o.OrderDate)
ORDER BY OrderYear;

--9. Task: Calculate the Rank of Customers by Total Order Amount
-- Task: Write a query that ranks customers by their total spending, but only for
customers located in "Mumbai". The rank should reset for each customer in
"Mumbai".

SELECT c.CustomerID, c.FirstName, c.LastName, SUM(o.OrderAmount) AS TotalSpent,
RANK() OVER (PARTITION BY c.City ORDER BY SUM(o.OrderAmount) DESC) AS
RankInMumbai
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
WHERE c.City = 'Mumbai'
GROUP BY c.CustomerID, c.FirstName, c.LastName, c.City;


--10. Task: Compare Each Customers Total Order to the Average Order Amount
-- Task: Write a query that calculates each customers total order amount and
compares it to the average order amount for all customers.

WITH CustomerTotals AS (
    SELECT c.CustomerID, c.FirstName, c.LastName, SUM(o.OrderAmount) AS
TotalSpent
    FROM Customers c
    JOIN Orders o ON c.CustomerID = o.CustomerID
    GROUP BY c.CustomerID, c.FirstName, c.LastName
),
AverageTotal AS (
    SELECT AVG(TotalSpent) AS AverageOrderAmount
    FROM CustomerTotals
)
```

```sql
SELECT ct.CustomerID, ct.FirstName, ct.LastName, ct.TotalSpent,
at.AverageOrderAmount,
       CASE
           WHEN ct.TotalSpent > at.AverageOrderAmount THEN 'Above Average'
           WHEN ct.TotalSpent < at.AverageOrderAmount THEN 'Below Average'
           ELSE 'Average'
       END AS Comparison
FROM CustomerTotals ct, AverageTotal at;
```