```sql
--Employees table
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Department VARCHAR(50),
    Salary DECIMAL(10, 2),
    HireDate DATE
);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Department, Salary, HireDate)
VALUES
    (1, 'John', 'Doe', 'HR', 50000.00, '2020-01-15'),
    (2, 'Jane', 'Smith', 'Marketing', 55000.00, '2019-05-20'),
    (3, 'Mike', 'Johnson', 'IT', 60000.00, '2018-09-10'),
    (4, 'Emily', 'Williams', 'Finance', 58000.00, '2021-03-12'),
    (5, 'David', 'Lee', 'Operations', 52000.00, '2017-11-25');

       --Customers table
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Email VARCHAR(100),
    Address VARCHAR(200),
    City VARCHAR(50),
    Country VARCHAR(50)
);

INSERT INTO Customers (CustomerID, FirstName, LastName, Email, Address, City, Country)
VALUES
    (1, 'Michael', 'Brown', 'michael@example.com', '123 Main St', 'New York', 'USA'),
    (2, 'Emma', 'Johnson', 'emma@example.com', '456 Elm St', 'Los Angeles', 'USA'),
    (3, 'Oliver', 'Smith', 'oliver@example.com', '789 Oak St', 'Chicago', 'USA'),
    (4, 'Sophia', 'Williams', 'sophia@example.com', '101 Maple Ave', 'Houston',
'USA'),
    (5, 'James', 'Lee', 'james@example.com', '222 Pine St', 'San Francisco', 'USA');

--Orders table
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    TotalAmount DECIMAL(10, 2),
    IsShipped BIT
);

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount, IsShipped)
VALUES
    (1, 3, '2023-07-01', 100.00, 1),
    (2, 1, '2023-07-05', 250.00, 1),
    (3, 4, '2023-07-10', 180.00, 0),
    (4, 2, '2023-07-15', 300.00, 1),
    (5, 5, '2023-07-20', 120.00, 1);
```

SQL Queries - Beginner Level

1. Retrieve the top 3 highest-paid employees.

```sql
SELECT TOP 3 * FROM Employees ORDER BY Salary DESC;
```

2. Find the customers who have placed orders.

```sql
SELECT DISTINCT Customers.CustomerID, FirstName, LastName FROM Customers

INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

3. Show employees and their department names in alphabetical order.

```sql
SELECT Employees.*, Departments.Department FROM Employees

LEFT JOIN Departments ON Employees.Department = Departments.Department

ORDER BY Departments.Department;
```

4. Find the customers who have placed orders for more than once.

```sql
SELECT Customers.* FROM Customers

INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID

GROUP BY Customers.CustomerID, FirstName, LastName

HAVING COUNT(Orders.OrderID) > 1;
```

5. Display orders with the customer's first name and last name.

```sql
SELECT Orders.*, Customers.FirstName, Customers.LastName FROM Orders

INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

6. Retrieve employees hired in the year 2022.

```sql
SELECT * FROM Employees WHERE YEAR(HireDate) = 2022;
```

7. Show customers who have placed orders on different dates.

```sql
SELECT DISTINCT Customers.* FROM Customers

INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID

GROUP BY Customers.CustomerID, FirstName, LastName

HAVING COUNT(DISTINCT Orders.OrderDate) > 1;
```

8. Retrieve the employees with the second and third highest salaries.

```
SELECT * FROM Employees

WHERE Salary IN (SELECT DISTINCT TOP 2 Salary FROM Employees ORDER BY Salary
DESC);
```

9. Find the total number of orders placed by each customer.

```
SELECT Customers.CustomerID, FirstName, LastName, COUNT(Orders.OrderID) AS
TotalOrders FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID

GROUP BY Customers.CustomerID, FirstName, LastName;
```

10. Retrieve employees who work in the IT department.

```
SELECT * FROM Employees WHERE Department = 'IT';
```

11. Find customers who have not placed any orders.

```
SELECT Customers.* FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID

WHERE Orders.CustomerID IS NULL;
```

12. Show the average salary of employees in each department.

```
SELECT Department, AVG(Salary) AS AverageSalary FROM Employees

GROUP BY Department;
```

13. Retrieve the employees with salaries above the average salary in their respective department.

```
SELECT E1.* FROM Employees E1

INNER JOIN (

    SELECT Department, AVG(Salary) AS AvgSalary FROM Employees GROUP BY Department

) E2 ON E1.Department = E2.Department

WHERE E1.Salary > E2.AvgSalary;
```

14. Display the names of employees who were hired on the same day as 'John Smith'.

```
SELECT FirstName, LastName FROM Employees

WHERE HireDate = (SELECT HireDate FROM Employees WHERE FirstName = 'John' AND
LastName = 'Smith');
```

15. Find customers who have placed orders for consecutive days.

```
SELECT DISTINCT Customers.* FROM Customers

INNER JOIN Orders O1 ON Customers.CustomerID = O1.CustomerID

INNER JOIN Orders O2 ON Customers.CustomerID = O2.CustomerID

WHERE DATEDIFF(DAY, O1.OrderDate, O2.OrderDate) = 1;
```

16. Find the nth highest salary from the Employees table.

```
DECLARE @N INT = 5; -- Change N to desired nth value

SELECT DISTINCT Salary FROM Employees

ORDER BY Salary DESC

OFFSET @N - 1 ROWS FETCH NEXT 1 ROW ONLY;
```

17. Show customers who have placed orders every day for the past week.

```sql
SELECT DISTINCT C.* FROM Customers C

WHERE NOT EXISTS (

    SELECT DATEADD(DAY, -7, GETDATE()) AS DateLimit

    WHERE NOT EXISTS (

        SELECT * FROM Orders O

        WHERE C.CustomerID = O.CustomerID

        AND O.OrderDate > DATEADD(DAY, -7, GETDATE())

        AND O.OrderDate <= GETDATE()

    )

);
```

18. Retrieve orders that were shipped after their expected shipment date.

```sql
SELECT * FROM Orders

WHERE IsShipped = 1 AND ShipDate > ExpectedShipDate;
```

19. Display the employees with their age (in years) calculated from the HireDate.

```sql
SELECT EmployeeID, FirstName, LastName, DATEDIFF(YEAR, HireDate, GETDATE()) AS Age
FROM Employees;
```

20. Find the customers who have not placed any orders in the last 3 months.

```sql
SELECT DISTINCT Customers.* FROM Customers

WHERE NOT EXISTS (

    SELECT * FROM Orders

    WHERE Customers.CustomerID = Orders.CustomerID

    AND OrderDate >= DATEADD(MONTH, -3, GETDATE())

);
```

21. Retrieve the list of employees and their managers.

```sql
SELECT E.EmployeeID, E.FirstName, E.LastName, M.FirstName AS ManagerFirstName,
M.LastName AS ManagerLastName

FROM Employees E

LEFT JOIN Employees M ON E.ManagerID = M.EmployeeID;
```

22. Show the customers who have the same first name or last name as employees.

```sql
SELECT DISTINCT C.* FROM Customers C

WHERE EXISTS (

    SELECT * FROM Employees E

    WHERE C.FirstName = E.FirstName OR C.LastName = E.LastName

);
```

23. Find the orders placed by the customers from the same city as 'John Smith'.

```
SELECT O.* FROM Orders O

INNER JOIN Customers C ON O.CustomerID = C.CustomerID

WHERE C.City = (SELECT City FROM Customers WHERE FirstName = 'John' AND LastName = 'Smith');
```

24. Retrieve customers who have placed orders for more than the average order amount.

```
SELECT C.* FROM Customers C

INNER JOIN Orders O ON C.CustomerID = O.CustomerID

WHERE O.TotalAmount > (SELECT AVG(TotalAmount) FROM Orders);
```

SQL Queries - Intermediate Level

25. Show the departments along with the number of employees in each department.

```
SELECT Department, COUNT(*) AS EmployeeCount FROM Employees

GROUP BY Department;
```

26. Retrieve the latest order for each customer.

```
SELECT O1.* FROM Orders O1

LEFT JOIN Orders O2 ON O1.CustomerID = O2.CustomerID AND O1.OrderDate < O2.OrderDate

WHERE O2.OrderID IS NULL;
```

27. Find the customers who have placed at least one order in each city.

```
SELECT C.* FROM Customers C

WHERE NOT EXISTS (

    SELECT DISTINCT City FROM Customers

    WHERE NOT EXISTS (

        SELECT * FROM Orders

        WHERE Customers.CustomerID = Orders.CustomerID

        AND Customers.City = Orders.ShipCity

    )

);
```

28. Show the employees who have the same hire date as their manager.

```
SELECT E.* FROM Employees E

INNER JOIN Employees M ON E.ManagerID = M.EmployeeID

WHERE E.HireDate = M.HireDate;
```

29. Retrieve the customers who have placed orders on all weekdays (Monday to Friday).

```
SELECT C.* FROM Customers C

WHERE NOT EXISTS (

    SELECT * FROM Orders

    WHERE C.CustomerID = Orders.CustomerID

    AND DATEPART(WEEKDAY, OrderDate) NOT BETWEEN 2 AND 6

);
```

30. Find the total sales amount for each year.

```
SELECT YEAR(OrderDate) AS Year, SUM(TotalAmount) AS TotalSales FROM Orders

GROUP BY YEAR(OrderDate);
```

31. Find the employees whose first name starts with 'J' and last name starts with 'S'.

```
SELECT * FROM Employees

WHERE FirstName LIKE 'J%' AND LastName LIKE 'S%';
```

32. Retrieve customers who have placed orders with a total amount greater than their average order amount.

```
SELECT C.* FROM Customers C

INNER JOIN (

    SELECT CustomerID, AVG(TotalAmount) AS AvgOrderAmount FROM Orders

    GROUP BY CustomerID

) O ON C.CustomerID = O.CustomerID

INNER JOIN Orders O2 ON C.CustomerID = O2.CustomerID

WHERE O2.TotalAmount > O.AvgOrderAmount;
```

33. Show the employees who have not been assigned any department.

```
SELECT * FROM Employees

WHERE Department IS NULL;
```

34. Retrieve the top 5 customers with the highest total order amount.

```
SELECT TOP 5 C.*, SUM(O.TotalAmount) AS TotalOrderAmount

FROM Customers C

INNER JOIN Orders O ON C.CustomerID = O.CustomerID

GROUP BY C.CustomerID, C.FirstName, C.LastName

ORDER BY TotalOrderAmount DESC;
```

35. Find the employees whose age is a prime number.

```
SELECT * FROM Employees

WHERE DATEDIFF(YEAR, HireDate, GETDATE()) IN (

    2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59

);
```

36. Retrieve the customers who have placed orders on all weekdays (Monday to Friday) in the last month.

```
SELECT C.* FROM Customers C

WHERE NOT EXISTS (

    SELECT * FROM Orders

    WHERE C.CustomerID = Orders.CustomerID

    AND DATEPART(WEEKDAY, OrderDate) NOT BETWEEN 2 AND 6

    AND OrderDate >= DATEADD(MONTH, -1, GETDATE())

);
```

37. Show the departments with the highest average employee salary.

```
SELECT TOP 1 Department, AVG(Salary) AS AverageSalary

FROM Employees

GROUP BY Department

ORDER BY AVG(Salary) DESC;
```

38. Retrieve employees who were hired in the same year and month as 'John Smith'.

```
SELECT * FROM Employees

WHERE YEAR(HireDate) = YEAR((SELECT HireDate FROM Employees WHERE FirstName =
'John' AND LastName = 'Smith'))

AND MONTH(HireDate) = MONTH((SELECT HireDate FROM Employees WHERE FirstName =
'John' AND LastName = 'Smith'));
```

39. Find the customers with the longest time between two consecutive orders.

```
SELECT DISTINCT C.* FROM Customers C

INNER JOIN Orders O1 ON C.CustomerID = O1.CustomerID

INNER JOIN Orders O2 ON C.CustomerID = O2.CustomerID AND O1.OrderDate <
O2.OrderDate

WHERE DATEDIFF(DAY, O1.OrderDate, O2.OrderDate) = (

    SELECT MAX(DATEDIFF(DAY, OrderDate, (SELECT MIN(OrderDate) FROM Orders O3
WHERE O3.CustomerID = O1.CustomerID AND O3.OrderDate > O1.OrderDate)))

    FROM Orders O1

    WHERE O1.CustomerID = C.CustomerID

);
```

40. Show the employees who have the same salary as their manager.

```
SELECT E.* FROM Employees E

INNER JOIN Employees M ON E.ManagerID = M.EmployeeID

WHERE E.Salary = M.Salary;
```

41. Find the top 5 highest-earning departments along with the total salary expense for each department.

```
SELECT TOP 5 Department, SUM(Salary) AS TotalSalaryExpense

FROM Employees

GROUP BY Department

ORDER BY TotalSalaryExpense DESC;
```

42. Retrieve the employees who have been assigned to multiple departments.

```
SELECT EmployeeID, FirstName, LastName

FROM Employees

WHERE EmployeeID IN (

    SELECT EmployeeID

    FROM Employees

    GROUP BY EmployeeID

    HAVING COUNT(DISTINCT Department) > 1

);
```

43. Show the cumulative total amount of orders for each customer in ascending order of the order date.

```
SELECT OrderID, CustomerID, OrderDate, TotalAmount,

       SUM(TotalAmount) OVER (PARTITION BY CustomerID ORDER BY OrderDate) AS
CumulativeTotalAmount

FROM Orders;
```

44. Retrieve the employees who have the same salary as the highest-earning employee in each department.

```
SELECT E.*

FROM Employees E

INNER JOIN (

    SELECT Department, MAX(Salary) AS MaxSalary

    FROM Employees

    GROUP BY Department

) MaxSalaries ON E.Department = MaxSalaries.Department AND E.Salary =
MaxSalaries.MaxSalary;
```

45. Find the customers who have placed orders with consecutive order IDs.

```sql
SELECT C.*

FROM Customers C

WHERE EXISTS (

    SELECT *

    FROM Orders O1

    WHERE C.CustomerID = O1.CustomerID

    AND EXISTS (

        SELECT *

        FROM Orders O2

        WHERE O1.CustomerID = O2.CustomerID AND O1.OrderID = O2.OrderID - 1

    )

);
```

SQL Queries - Intermediate Level

51. Find the customers who have placed the highest number of orders in their respective countries.

```sql
SELECT C.*

FROM Customers C

WHERE C.CustomerID IN (

    SELECT TOP 1 WITH TIES CustomerID

    FROM Orders O

    GROUP BY CustomerID, Country

    ORDER BY COUNT(*) DESC);
```

52. Retrieve the orders that have the highest total amount for each year.

```sql
SELECT O.*

FROM Orders O

WHERE O.OrderID IN (

    SELECT TOP 1 WITH TIES OrderID

    FROM Orders

    WHERE YEAR(OrderDate) = YEAR(O.OrderDate)

    ORDER BY TotalAmount DESC

);
```

53. Show the employees who have not been assigned to any department but have a higher salary than the average salary of all employees.

```sql
SELECT * FROM Employees

WHERE Department IS NULL

AND Salary > (

    SELECT AVG(Salary) FROM Employees

);
```

54. Retrieve the customers who have placed orders on all weekdays (Monday to Friday) and have spent the highest total amount in their respective cities.

```sql
SELECT C.*

FROM Customers C

WHERE EXISTS (

    SELECT * FROM Orders O1

    WHERE C.CustomerID = O1.CustomerID

    AND NOT EXISTS (

        SELECT * FROM Orders O2

        WHERE O1.CustomerID = O2.CustomerID

        AND DATEPART(WEEKDAY, O2.OrderDate) NOT BETWEEN 2 AND 6

    )

)

AND C.CustomerID IN (

    SELECT TOP 1 WITH TIES CustomerID

    FROM Orders O

    GROUP BY CustomerID, City

    ORDER BY SUM(TotalAmount) DESC

);
```

55. Find the employees who have more than one subordinate and have been hired before their manager.

```
SELECT E.*

FROM Employees E

INNER JOIN (

    SELECT ManagerID, COUNT(*) AS SubordinateCount

    FROM Employees

    GROUP BY ManagerID

    HAVING COUNT(*) > 1

) Subordinates ON E.EmployeeID = Subordinates.ManagerID

WHERE E.HireDate < (

    SELECT HireDate FROM Employees WHERE EmployeeID = E.ManagerID

);
```

61. Retrieve the top 5 customers with the highest average order amount.

```
SELECT TOP 5 C.*, AVG(O.TotalAmount) AS AverageOrderAmount

FROM Customers C

INNER JOIN Orders O ON C.CustomerID = O.CustomerID

GROUP BY C.CustomerID, C.FirstName, C.LastName

ORDER BY AverageOrderAmount DESC;
```

62. Show the employees who have been assigned to multiple departments and have the highest salary in their respective departments.

```
SELECT E.*

FROM Employees E

INNER JOIN (

    SELECT Department, MAX(Salary) AS MaxSalary

    FROM Employees

    GROUP BY Department

) MaxSalaries ON E.Department = MaxSalaries.Department AND E.Salary =
MaxSalaries.MaxSalary

WHERE E.EmployeeID IN (

    SELECT EmployeeID

    FROM Employees

    GROUP BY EmployeeID

    HAVING COUNT(DISTINCT Department) > 1

);
```

63. Retrieve the customers who have placed orders on all weekdays (Monday to Friday) and have spent the highest total amount in their respective countries.

```
SELECT C.*

FROM Customers C

WHERE EXISTS (

    SELECT * FROM Orders O1

    WHERE C.CustomerID = O1.CustomerID

    AND NOT EXISTS (

        SELECT * FROM Orders O2

        WHERE O1.CustomerID = O2.CustomerID

        AND DATEPART(WEEKDAY, O2.OrderDate) NOT BETWEEN 2 AND 6

    )

)

AND C.CustomerID IN (

    SELECT TOP 1 WITH TIES CustomerID

    FROM Orders O

    GROUP BY CustomerID, Country

    ORDER BY SUM(TotalAmount) DESC

);
```

64. Find the employees who have more than one subordinate and have been hired before their manager, and their age is a prime number.

```
SELECT E.*

FROM Employees E

INNER JOIN (

    SELECT ManagerID, COUNT(*) AS SubordinateCount

    FROM Employees

    GROUP BY ManagerID

    HAVING COUNT(*) > 1

) Subordinates ON E.EmployeeID = Subordinates.ManagerID

WHERE E.HireDate < (

    SELECT HireDate FROM Employees WHERE EmployeeID = E.ManagerID

)

AND DATEDIFF(YEAR, E.HireDate, GETDATE()) IN (

    2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59

);
```

65. Retrieve the customers who have placed orders with consecutive order IDs and have spent the highest total amount in their respective cities.

```
SELECT C.*

FROM Customers C

WHERE EXISTS (

    SELECT *

    FROM Orders O1

    WHERE C.CustomerID = O1.CustomerID

    AND EXISTS (

        SELECT *

        FROM Orders O2

        WHERE O1.CustomerID = O2.CustomerID AND O1.OrderID = O2.OrderID - 1

    ))

AND C.CustomerID IN (

    SELECT TOP 1 WITH TIES CustomerID

    FROM Orders O

    GROUP BY CustomerID, City

    ORDER BY SUM(TotalAmount) DESC);
```

66. Show the customers who have not placed any orders in the last 6 months and have the highest total order amount.

```
SELECT TOP 1 WITH TIES C.*

FROM Customers C

LEFT JOIN Orders O ON C.CustomerID = O.CustomerID

WHERE O.OrderDate IS NULL OR O.OrderDate <= DATEADD(MONTH, -6, GETDATE())

ORDER BY O.TotalAmount DESC;
```

67. Retrieve the orders with the top 5 highest total amount that have not been shipped yet.

```
SELECT TOP 5 O.*

FROM Orders O

WHERE O.IsShipped = 0

ORDER BY O.TotalAmount DESC;
```

68. Find the employees who have been assigned to all departments available in the company.

```
SELECT E.*

FROM Employees E

WHERE NOT EXISTS (

    SELECT Department FROM Departments

    WHERE NOT EXISTS (

        SELECT * FROM Employees

        WHERE EmployeeID = E.EmployeeID AND Department = Departments.Department

    )

);
```

69. Show the customers who have placed the highest number of orders in the last 3 months and have spent the highest total amount in their respective countries.

```
SELECT C.* FROM Customers C

WHERE C.CustomerID IN (SELECT TOP 1 WITH TIES CustomerID

    FROM Orders O WHERE O.OrderDate >= DATEADD(MONTH, -3, GETDATE())

    GROUP BY CustomerID

    ORDER BY COUNT(*) DESC)
```

```
AND C.CustomerID IN (

    SELECT TOP 1 WITH TIES CustomerID

    FROM Orders O

    GROUP BY CustomerID, Country

    ORDER BY SUM(TotalAmount) DESC

);
```

SQL Queries - Advanced Level

70. Retrieve the top 3 employees with the highest average salary in their respective departments, and their age is a prime number.

```
SELECT TOP 3 E.*

FROM Employees E

INNER JOIN (

    SELECT Department, AVG(Salary) AS AvgSalary

    FROM Employees

    GROUP BY Department

) AvgSalaries ON E.Department = AvgSalaries.Department AND E.Salary =
AvgSalaries.AvgSalary

WHERE DATEDIFF(YEAR, E.HireDate, GETDATE()) IN (

    2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59

);
```

71. Retrieve the top 5 departments with the highest average salary of employees whose age is a prime number.

```
SELECT TOP 5 Department, AVG(Salary) AS AvgSalary

FROM Employees

WHERE DATEDIFF(YEAR, HireDate, GETDATE()) IN (

    2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59

)

GROUP BY Department

ORDER BY AvgSalary DESC;
```

72. Find the customers who have placed orders with a total amount that is a perfect square.

```
SELECT DISTINCT C.*

FROM Customers C

INNER JOIN Orders O ON C.CustomerID = O.CustomerID

WHERE SQRT(O.TotalAmount) = CAST(SQRT(O.TotalAmount) AS INT);
```

73. Show the orders that have the same total amount as the average total amount of orders for each year.

```
SELECT O.*

FROM Orders O

WHERE O.TotalAmount IN (
    SELECT AVG(TotalAmount) AS AvgAmount
    FROM Orders
    WHERE YEAR(OrderDate) = YEAR(O.OrderDate)
    GROUP BY YEAR(OrderDate)
);
```

74. Retrieve the employees who have been assigned to the maximum number of departments.

```
SELECT E.*
FROM Employees E
WHERE EmployeeID IN (
    SELECT TOP 1 WITH TIES EmployeeID
    FROM Employees
    GROUP BY EmployeeID
    ORDER BY COUNT(DISTINCT Department) DESC
);
```

75. Find the customers who have placed orders with the same total amount as another customer from a different country.

```
SELECT C.*
FROM Customers C
WHERE EXISTS (
    SELECT * FROM Orders O1

    WHERE C.CustomerID = O1.CustomerID

    AND EXISTS (

        SELECT * FROM Orders O2

        WHERE O1.TotalAmount = O2.TotalAmount

        AND O1.CustomerID <> O2.CustomerID

        AND C.Country <> (

            SELECT Country FROM Customers WHERE CustomerID = O2.CustomerID

        )

    )

);
```

76. Retrieve the employees who have not been assigned to any department and have a higher salary than any employee assigned to a department.

```
SELECT * FROM Employees

WHERE Department IS NULL

AND Salary > (

    SELECT MAX(Salary) FROM Employees WHERE Department IS NOT NULL

);
```

77. Show the customers who have placed orders with consecutive order IDs and have spent the highest total amount in their respective countries.

```
SELECT C.*
FROM Customers C
WHERE EXISTS (
    SELECT *
    FROM Orders O1
    WHERE C.CustomerID = O1.CustomerID
    AND EXISTS (
        SELECT *
        FROM Orders O2
        WHERE O1.CustomerID = O2.CustomerID AND O1.OrderID = O2.OrderID - 1
    )
)
AND C.CustomerID IN (
    SELECT TOP 1 WITH TIES CustomerID
    FROM Orders O
    GROUP BY CustomerID, Country
    ORDER BY SUM(TotalAmount) DESC
);
```

78. Retrieve the orders with the top 5 highest total amount that have not been shipped yet, and the customers who placed these orders.

```
SELECT TOP 5 O.*, C.FirstName, C.LastName
FROM Orders O
```

```sql
INNER JOIN Customers C ON O.CustomerID = C.CustomerID

WHERE O.IsShipped = 0

ORDER BY O.TotalAmount DESC;
```

79. Find the employees who have been assigned to all departments available in the company, and their age is an odd number.

```sql
SELECT E.*

FROM Employees E

WHERE NOT EXISTS (

    SELECT Department FROM Departments

    WHERE NOT EXISTS (

        SELECT * FROM Employees

        WHERE EmployeeID = E.EmployeeID AND Department = Departments.Department

    )

)

AND DATEDIFF(YEAR, E.HireDate, GETDATE()) % 2 <> 0;
```

SQL Queries - Expert Level

80. Show the top 3 departments with the highest average salary of employees, and the employees who have the highest salary in each of these departments.

```
SELECT D.Department, E.*

FROM (

    SELECT TOP 3 Department, AVG(Salary) AS AvgSalary

    FROM Employees

    GROUP BY Department

    ORDER BY AvgSalary DESC

) D

INNER JOIN Employees E ON D.Department = E.Department

WHERE E.Salary = (

    SELECT MAX(Salary) FROM Employees WHERE Department = D.Department

);
```

91. Retrieve the customers who have placed the highest number of orders on weekends (Saturday and Sunday).

```
SELECT TOP 5 C.*, COUNT(*) AS WeekendOrders

FROM Customers C

INNER JOIN Orders O ON C.CustomerID = O.CustomerID

WHERE DATEPART(WEEKDAY, O.OrderDate) IN (1, 7) -- 1=Sunday, 7=Saturday

GROUP BY C.CustomerID, C.FirstName, C.LastName

ORDER BY WeekendOrders DESC;
```

92. Show the employees who have not been assigned to any department and have the highest salary in the company.

```
SELECT TOP 1 WITH TIES *

FROM Employees

WHERE Department IS NULL

ORDER BY Salary DESC;
```

93. Retrieve the customers who have placed orders with the total amount closest to the average total amount of all orders.

```
SELECT TOP 5 C.*

FROM Customers C

INNER JOIN Orders O ON C.CustomerID = O.CustomerID

ORDER BY ABS(O.TotalAmount - (SELECT AVG(TotalAmount) FROM Orders)) ASC;
```

94. Find the customers who have placed orders on their birthdays.

```
SELECT C.*

FROM Customers C

INNER JOIN Orders O ON C.CustomerID = O.CustomerID

WHERE DAY(O.OrderDate) = DAY(CAST(DATEADD(YEAR, DATEDIFF(YEAR, BirthDate,
GETDATE()), BirthDate) AS DATE))

AND MONTH(O.OrderDate) = MONTH(CAST(DATEADD(YEAR, DATEDIFF(YEAR, BirthDate,
GETDATE()), BirthDate) AS DATE));
```

95. Show the employees who have been assigned to departments with consecutive IDs.

```
SELECT E.*
FROM Employees E
WHERE NOT EXISTS (
    SELECT * FROM Departments D
    WHERE NOT EXISTS (
        SELECT * FROM Employees
        WHERE EmployeeID = E.EmployeeID AND Department = D.DepartmentID
    )
);
```

96. Retrieve the employees who have the same first name as another employee and their salaries are within $1000 of each other.

```
SELECT E.*
FROM Employees E
WHERE EXISTS (
    SELECT * FROM Employees E2
    WHERE E.EmployeeID <> E2.EmployeeID
    AND E.FirstName = E2.FirstName
    AND ABS(E.Salary - E2.Salary) <= 1000
);
```

97. Find the customers who have placed orders with the same total amount as other customers and have the same city as those customers.

```
SELECT C.*

FROM Customers C

WHERE EXISTS (

    SELECT * FROM Orders O1

    WHERE C.CustomerID = O1.CustomerID

    AND EXISTS (

        SELECT * FROM Orders O2

        WHERE O1.TotalAmount = O2.TotalAmount

        AND O1.CustomerID <> O2.CustomerID

        AND C.City = (

            SELECT City FROM Customers WHERE CustomerID = O2.CustomerID

        )

    )

);
```

98. Show the employees who have the same hire year as the highest-earning employee in each department.

```
SELECT E.*

FROM Employees E

WHERE E.HireDate IN (

    SELECT MAX(HireDate) FROM Employees WHERE Department = E.Department);
```

99. Retrieve the customers who have placed orders on the same day as their birthdays.

```
SELECT C.*

FROM Customers C

INNER JOIN Orders O ON C.CustomerID = O.CustomerID

WHERE CAST(O.OrderDate AS DATE) = CAST(DATEADD(YEAR, DATEDIFF(YEAR, BirthDate,
GETDATE()), BirthDate) AS DATE);
```

100. Find the employees who have been assigned to a department but have not received a salary raise in the last 2 years.

```
SELECT E.* FROM Employees E

WHERE E.Department IS NOT NULL

AND E.EmployeeID NOT IN (

    SELECT EmployeeID FROM SalaryHistory WHERE SalaryDate >= DATEADD(YEAR, -2,
GETDATE())

);
```

101. Retrieve the customers who have placed the highest number of orders on weekdays (Monday to Friday) during the last 3 months.

```
SELECT TOP 5 C.*, COUNT(*) AS WeekdayOrders

FROM Customers C

INNER JOIN Orders O ON C.CustomerID = O.CustomerID

WHERE DATEPART(WEEKDAY, O.OrderDate) BETWEEN 2 AND 6 -- Monday to Friday

AND O.OrderDate >= DATEADD(MONTH, -3, GETDATE())

GROUP BY C.CustomerID, C.FirstName, C.LastName

ORDER BY WeekdayOrders DESC;
```

102. Show the employees who have not been assigned to any department and have a higher salary than any employee in the company's history.

```
SELECT TOP 1 WITH TIES *

FROM Employees

WHERE Department IS NULL

AND Salary > (

    SELECT MAX(Salary) FROM Employees

);
```

103. Retrieve the customers who have placed orders with the total amount closest to the average total amount of all orders in their respective countries.

```
SELECT C.*

FROM Customers C

WHERE EXISTS (

    SELECT * FROM Orders O1

    WHERE C.CustomerID = O1.CustomerID

    AND EXISTS (

        SELECT * FROM Orders O2

        WHERE O1.Country = O2.Country

        AND ABS(O1.TotalAmount - (SELECT AVG(TotalAmount) FROM Orders WHERE
Country = O2.Country)) <=

            ABS(O2.TotalAmount - (SELECT AVG(TotalAmount) FROM Orders WHERE
Country = O2.Country))

    )

);
```

104. Find the customers who have placed orders on a leap day (February 29th).

```
SELECT C.*

FROM Customers C

INNER JOIN Orders O ON C.CustomerID = O.CustomerID

WHERE MONTH(O.OrderDate) = 2 AND DAY(O.OrderDate) = 29;
```

105. Show the employees who have been assigned to departments with consecutive IDs and have the highest salary among their peers.

```
SELECT E.*

FROM Employees E

WHERE NOT EXISTS (

    SELECT * FROM Departments D

    WHERE NOT EXISTS (

        SELECT * FROM Employees

        WHERE EmployeeID = E.EmployeeID AND Department = D.DepartmentID

    )

)

AND E.Salary = (

    SELECT MAX(Salary) FROM Employees WHERE Department = E.Department

);
```

106. Retrieve the employees who have received at least one salary raise every year since their hire date.

```
SELECT E.*

FROM Employees E
```

```
WHERE E.EmployeeID NOT IN (

    SELECT EmployeeID FROM SalaryHistory

    WHERE YEAR(SalaryDate) < YEAR(E.HireDate))

AND E.EmployeeID NOT IN (

    SELECT EmployeeID FROM SalaryHistory

    GROUP BY EmployeeID

    HAVING COUNT(DISTINCT YEAR(SalaryDate)) < DATEDIFF(YEAR, E.HireDate,
GETDATE()) + 1

);
```

107. Find the customers who have placed orders on their birthdays and the total amount spent on those orders is greater than $100.

```
SELECT C.*

FROM Customers C

INNER JOIN Orders O ON C.CustomerID = O.CustomerID

WHERE CAST(O.OrderDate AS DATE) = CAST(DATEADD(YEAR, DATEDIFF(YEAR, BirthDate,
GETDATE()), BirthDate) AS DATE)

AND O.TotalAmount > 100;
```

108. Show the employees who have a salary that is both the minimum and maximum salary in the company.

```
SELECT * FROM Employees

WHERE Salary = (

    SELECT MIN(Salary) FROM Employees

) AND EmployeeID IN (

    SELECT EmployeeID FROM Employees

    WHERE Salary = (
        SELECT MAX(Salary) FROM Employees
    )
);
```

109. Retrieve the customers who have placed orders on the same day of the week (e.g., all orders on Mondays).

```
SELECT C.*
FROM Customers C
INNER JOIN Orders O ON C.CustomerID = O.CustomerID
WHERE DATEPART(WEEKDAY, O.OrderDate) = DATEPART(WEEKDAY, (SELECT MIN(OrderDate)
FROM Orders));
```

110. Find the employees who have received a salary raise on their birthday.

```
SELECT E.*
FROM Employees E
INNER JOIN SalaryHistory S ON E.EmployeeID = S.EmployeeID
WHERE CAST(S.SalaryDate AS DATE) = CAST(DATEADD(YEAR, DATEDIFF(YEAR, E.BirthDate,
GETDATE()), E.BirthDate) AS DATE);
```

Conclusion

In conclusion, SQL queries are essential tools for working with relational databases. They allow us to extract, manipulate, and transform data to gain valuable insights and answer specific questions. In this practice session, we covered a wide range of SQL queries, starting from basic queries and gradually increasing the complexity for both beginners and experienced users.

For Beginners

For beginners, we focused on fundamental SQL concepts such as SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY, and JOIN clauses. We practiced writing queries to retrieve, filter, and aggregate data from different tables.

Advancing Complexity

As the practice advanced, we delved into more complex queries involving subqueries, common table expressions (CTEs), window functions, and set operations. We also explored scenarios like handling NULL values, using string functions, and working with date and time data.

For Experienced Users

For experienced users, we presented challenging queries involving advanced techniques like recursive CTEs, PIVOT, and complex subqueries. These queries demanded a deeper understanding of SQL and showcased how to tackle real-world scenarios effectively.

Continued Learning

The practice session also provided HTML-formatted SQL queries, making it convenient for users to copy and paste them into their blogs or practice environments. It's important to note that while these examples cover a wide array of SQL query types, the field of SQL is vast, and there is always more to learn. Continuously practicing and exploring new scenarios will help developers and data analysts become more proficient in SQL and gain a deeper understanding of database management and data manipulation.

Best Practices

Remember, understanding the database schema and the underlying data is crucial for writing effective and optimized SQL queries. Always test your queries thoroughly and ensure they produce the desired results before using them in production environments.

Happy querying and may this practice session contribute to your growth as a skilled SQL practitioner!