SQL Question

Basic SQL Question

1.Retrieve all records from the "Customers" table

```
SELECT * FROM Customers;
```

2.Display the names of all employees in the "Employees" table

```
SELECT FirstName, LastName FROM Employees;
```

3.List the distinct product categories from the "Products" table

```
SELECT DISTINCT Category FROM Products;
```

4.Find the total number of orders in the "Orders" table

```
SELECT COUNT(*) AS TotalOrders FROM Orders;
```

5. Retrieve the first 5 records from the "Products" table

```
SELECT * FROM Products LIMIT 5;
```

6.Show the highest price from the "Products" table

```
SELECT MAX(Price) AS HighestPrice FROM Products;
```

7.Display the order ID, order date, and customer name from the "Orders" and "Customers" tables

```
SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

8.List the products with a price greater than $50

```
SELECT * FROM Products WHERE Price > 50;
```

9.Retrieve the count of products in each category

```sql
SELECT Category, COUNT(*) AS ProductCount FROM Products GROUP BY Category;
```

10.Calculate the average salary of employees in the "Employees" table

```sql
SELECT AVG(Salary) AS AverageSalary FROM Employees;
```

Intermediate Level SQL Question

1. Retrieve the names of customers who have placed more than 3 orders

```sql
SELECT Customers.CustomerName
FROM Customers
INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID
GROUP BY Customers.CustomerName
HAVING COUNT(Orders.OrderID) > 3;
```

2. Display the top 3 most expensive products

```sql
SELECT * FROM Products ORDER BY Price DESC LIMIT 3;
```

3. Find the customer who placed the most recent order

```sql
SELECT Customers.CustomerName
FROM Customers
INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Orders.OrderDate DESC
LIMIT 1;
```

4. Calculate the total revenue generated by each product

```sql
SELECT Products.ProductName, SUM(OrderDetails.Quantity * Products.Price)
FROM Products
INNER JOIN OrderDetails ON Products.ProductID = OrderDetails.ProductID
GROUP BY Products.ProductName;
```

5. List the employees who manage other employees (use the "ReportsTo" column)

```sql
SELECT Manager.FirstName AS ManagerName, Employee.FirstName AS EmployeeName
FROM Employees AS Manager
INNER JOIN Employees AS Employee ON Manager.EmployeeID = Employee.ReportsTo;
```

6. Retrieve the orders placed in the year 2022

```sql
SELECT * FROM Orders WHERE YEAR(OrderDate) = 2022;
```

7. Display the names of customers who have not placed any orders

```sql
SELECT Customers.CustomerName
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
WHERE Orders.CustomerID IS NULL;
```

8. Find the total number of orders for each year

```sql
SELECT YEAR(OrderDate) AS OrderYear, COUNT(*) AS TotalOrders
FROM Orders
GROUP BY YEAR(OrderDate);
```

9. Calculate the average order amount for each customer

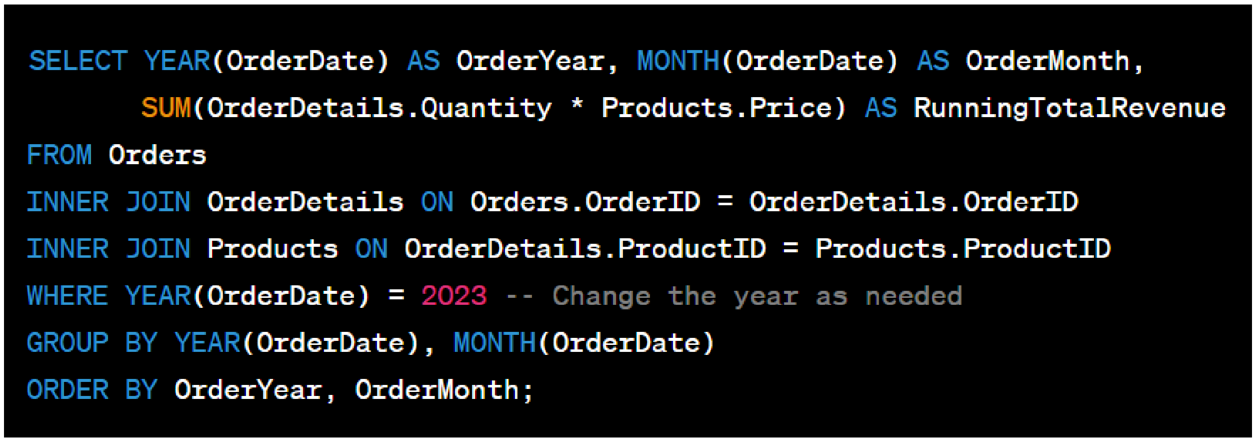
```sql
SELECT Customers.CustomerName, AVG(OrderDetails.Quantity * Products.Price)
FROM Customers
INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID
INNER JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
INNER JOIN Products ON OrderDetails.ProductID = Products.ProductID
GROUP BY Customers.CustomerName;
```

10. List the employees who have the same manager as the employee with ID 5

```sql
SELECT Employee.FirstName
FROM Employees AS Manager
INNER JOIN Employees AS Employee ON Manager.EmployeeID = Employee.ReportsTo
WHERE Manager.EmployeeID = 5;
```

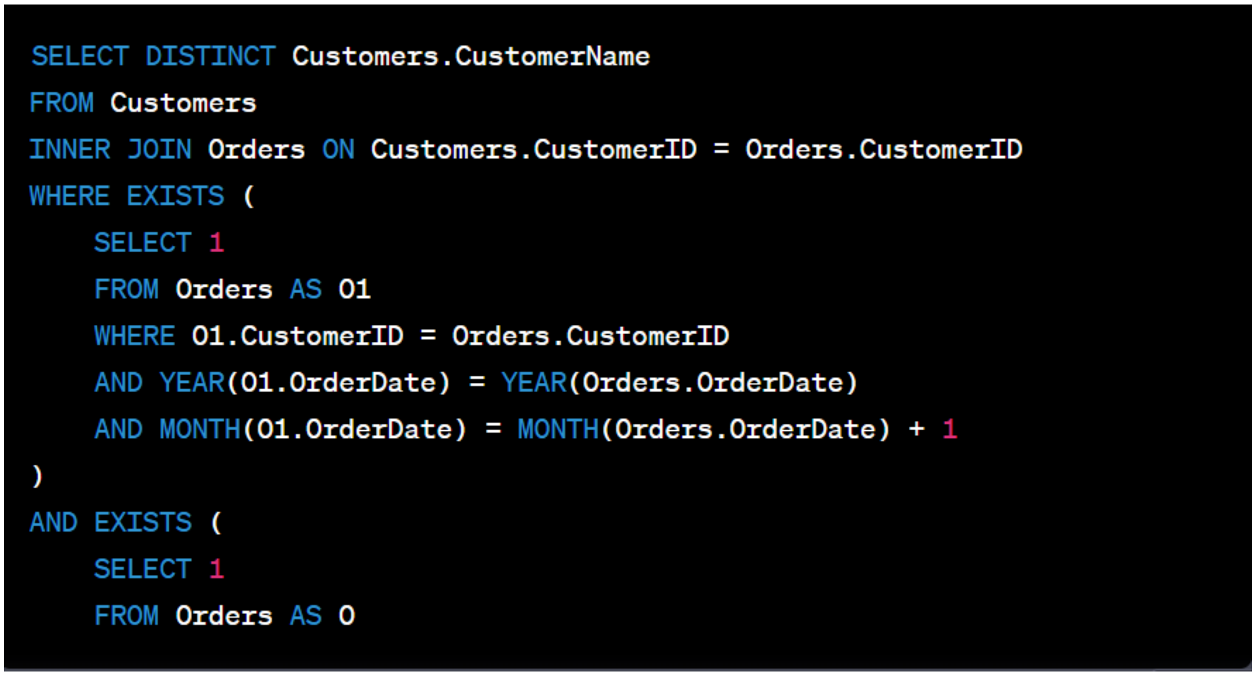
Advanced Level SQL Question

1. Calculate the running total of revenue for each month within a year



```sql
SELECT YEAR(OrderDate) AS OrderYear, MONTH(OrderDate) AS OrderMonth,
       SUM(OrderDetails.Quantity * Products.Price) AS RunningTotalRevenue
FROM Orders
INNER JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
INNER JOIN Products ON OrderDetails.ProductID = Products.ProductID
WHERE YEAR(OrderDate) = 2023 -- Change the year as needed
GROUP BY YEAR(OrderDate), MONTH(OrderDate)
ORDER BY OrderYear, OrderMonth;
```

2. List the customers who have placed orders in three consecutive months



```sql
SELECT DISTINCT Customers.CustomerName
FROM Customers
INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID
WHERE EXISTS (
    SELECT 1
    FROM Orders AS O1
    WHERE O1.CustomerID = Orders.CustomerID
    AND YEAR(O1.OrderDate) = YEAR(Orders.OrderDate)
    AND MONTH(O1.OrderDate) = MONTH(Orders.OrderDate) + 1
)
AND EXISTS (
    SELECT 1
    FROM Orders AS O
```

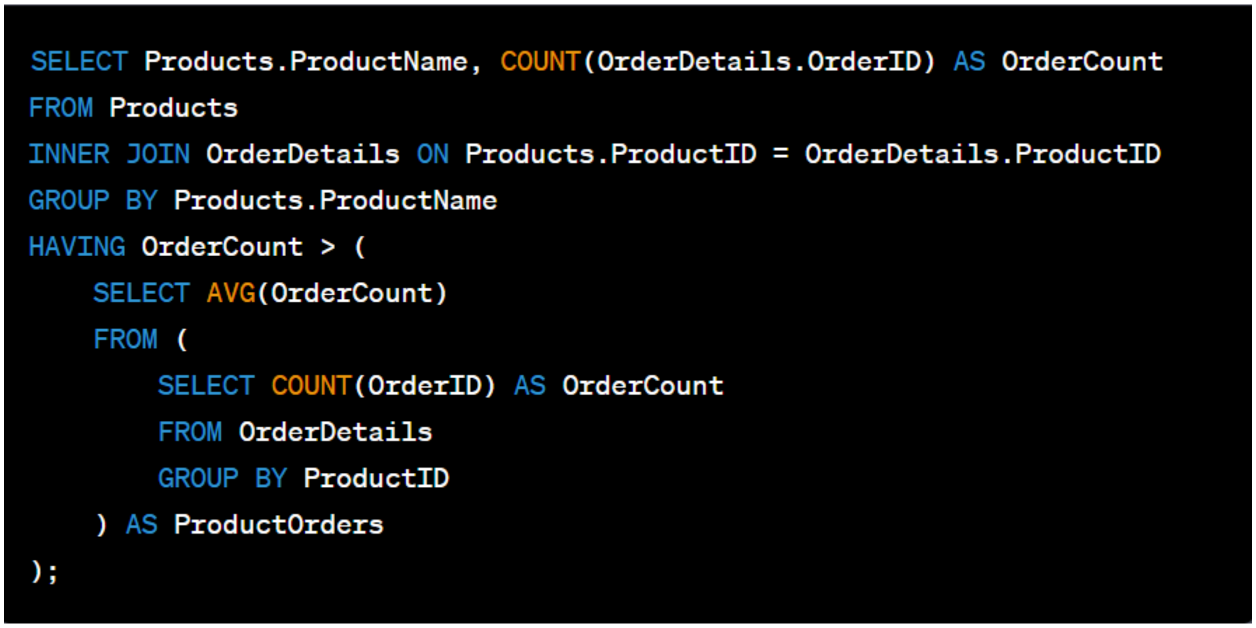
3. List the employees who have never placed an order

```sql
SELECT Employees.EmployeeID, Employees.FirstName, Employees.LastName
FROM Employees
LEFT JOIN Orders ON Employees.EmployeeID = Orders.EmployeeID
WHERE Orders.EmployeeID IS NULL;
```

4. Retrieve the names of customers who have placed orders on the same day they registered.

```sql
SELECT Customers.CustomerName
FROM Customers
INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID
WHERE DATE(Customers.RegistrationDate) = DATE(Orders.OrderDate);
```

5. Find the products that have been ordered more than the average number of times.



```sql
SELECT Products.ProductName, COUNT(OrderDetails.OrderID) AS OrderCount
FROM Products
INNER JOIN OrderDetails ON Products.ProductID = OrderDetails.ProductID
GROUP BY Products.ProductName
HAVING OrderCount > (
    SELECT AVG(OrderCount)
    FROM (
        SELECT COUNT(OrderID) AS OrderCount
        FROM OrderDetails
        GROUP BY ProductID
    ) AS ProductOrders
);
```

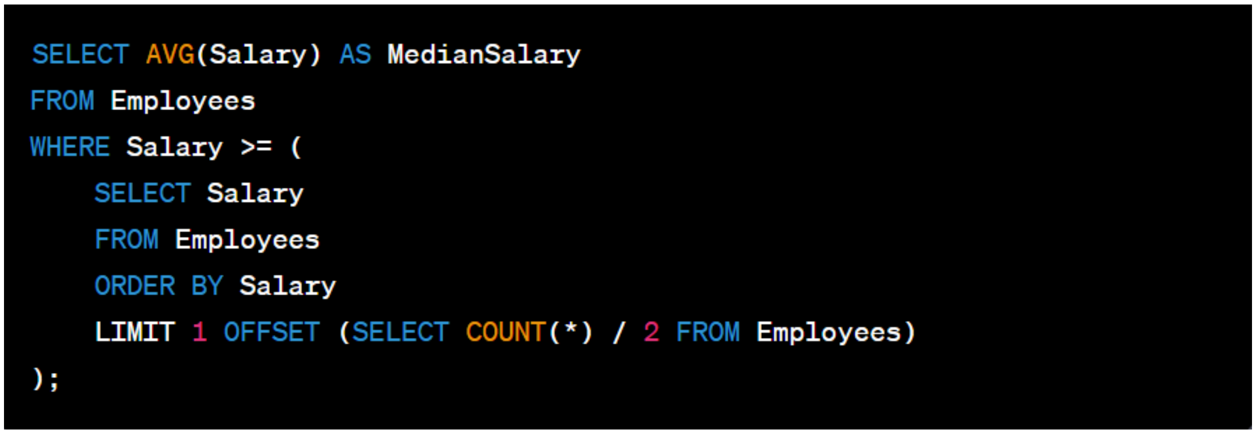
6. Find customers who have made orders of products from at least three different product categories in each of the last three years. Return their names along with the total count of such orders for each year.

```sql
SELECT C.CustomerName,
       YEAR(O.OrderDate) AS OrderYear,
       COUNT(DISTINCT P.Category) AS DistinctCategories,
       COUNT(*) AS OrderCount
FROM Customers C
INNER JOIN Orders O ON C.CustomerID = O.CustomerID
INNER JOIN OrderDetails OD ON O.OrderID = OD.OrderID
INNER JOIN Products P ON OD.ProductID = P.ProductID
WHERE YEAR(O.OrderDate) >= YEAR(CURRENT_DATE) - 3
GROUP BY C.CustomerID, YEAR(O.OrderDate)
HAVING COUNT(DISTINCT P.Category) >= 3
ORDER BY C.CustomerName, OrderYear;
```

7. Retrieve the names of employees who have placed orders before their managers.

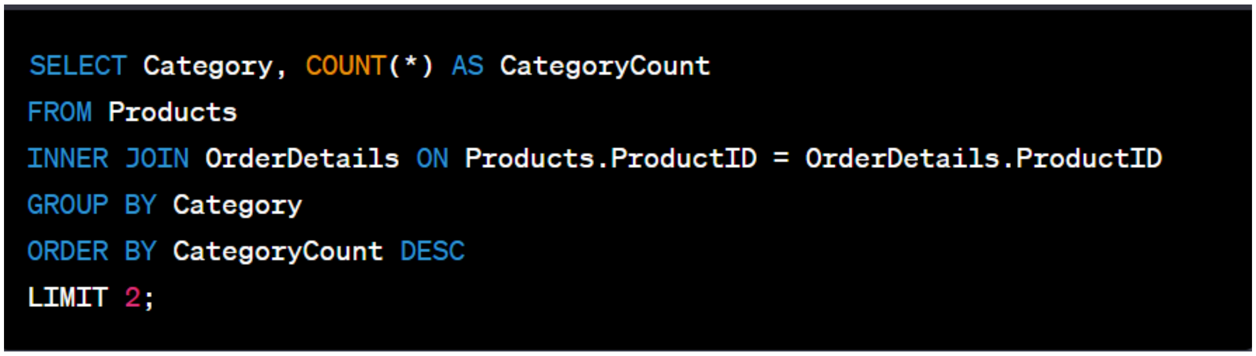
```sql
SELECT Employee.FirstName, Manager.FirstName AS ManagerName
FROM Employees AS Employee
INNER JOIN Employees AS Manager ON Employee.ReportsTo = Manager.EmployeeID
INNER JOIN Orders ON Employee.EmployeeID = Orders.EmployeeID
WHERE Orders.OrderDate < (
    SELECT MIN(OrderDate)
    FROM Orders
    WHERE Employee.ReportsTo = Orders.EmployeeID
);
```

8. Calculate the median salary of employees from the "Employees" tabl



```
SELECT AVG(Salary) AS MedianSalary
FROM Employees
WHERE Salary >= (
    SELECT Salary
    FROM Employees
    ORDER BY Salary
    LIMIT 1 OFFSET (SELECT COUNT(*) / 2 FROM Employees)
);
```

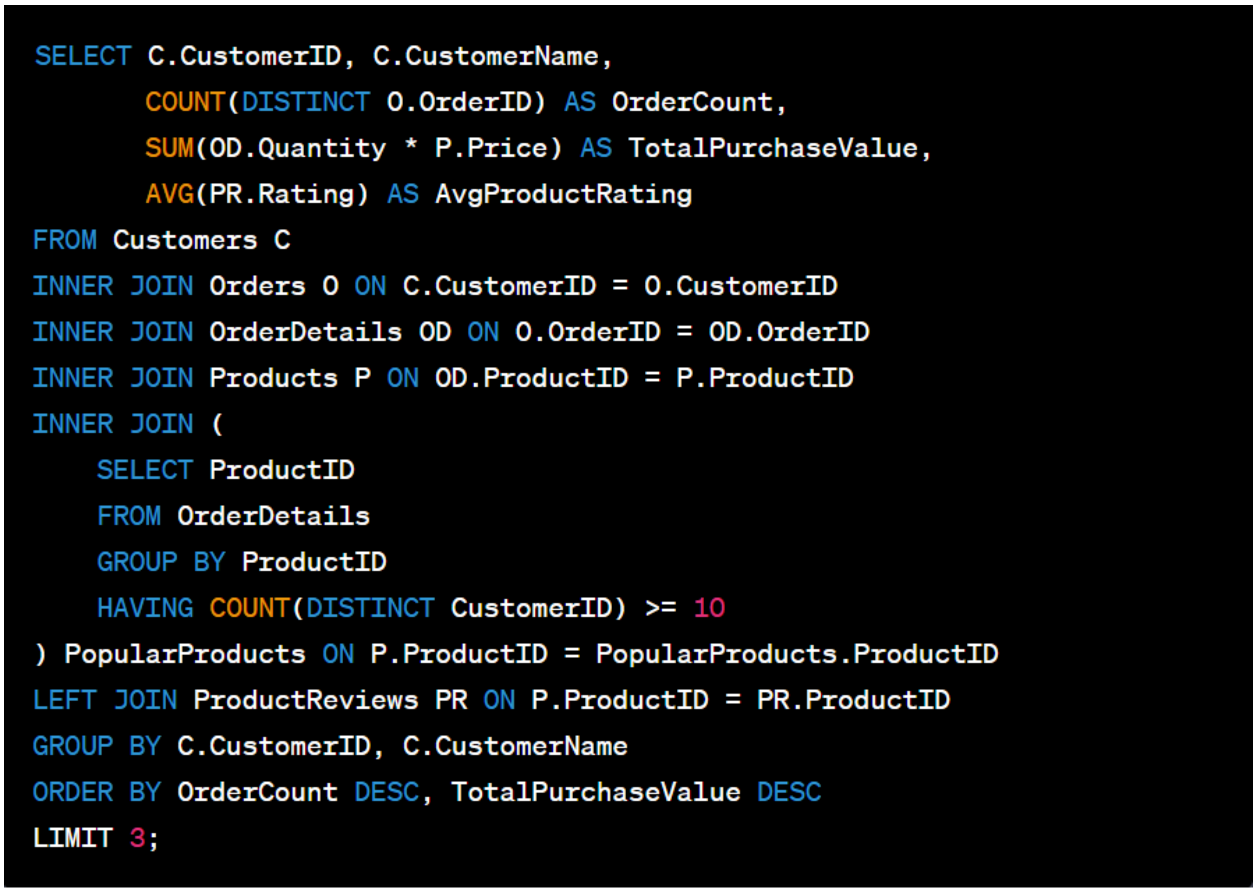
9. Retrieve the top 2 most common product categories from the "OrderDetails" table.



```
SELECT Category, COUNT(*) AS CategoryCount
FROM Products
INNER JOIN OrderDetails ON Products.ProductID = OrderDetails.ProductID
GROUP BY Category
ORDER BY CategoryCount DESC
LIMIT 2;
```

10. Retrieve the top 3 customers who have made the most frequent and highest total value of purchases, considering products that have been ordered by at least 10 different customers. Include the average rating of the products they've purchased.

Assuming you have tables "Customers", "Orders", "OrderDetails", "Products", and "ProductReviews" with appropriate columns, here's a challenging SQL query



```sql
SELECT C.CustomerID, C.CustomerName,
       COUNT(DISTINCT O.OrderID) AS OrderCount,
       SUM(OD.Quantity * P.Price) AS TotalPurchaseValue,
       AVG(PR.Rating) AS AvgProductRating
FROM Customers C
INNER JOIN Orders O ON C.CustomerID = O.CustomerID
INNER JOIN OrderDetails OD ON O.OrderID = OD.OrderID
INNER JOIN Products P ON OD.ProductID = P.ProductID
INNER JOIN (
    SELECT ProductID
    FROM OrderDetails
    GROUP BY ProductID
    HAVING COUNT(DISTINCT CustomerID) >= 10
) PopularProducts ON P.ProductID = PopularProducts.ProductID
LEFT JOIN ProductReviews PR ON P.ProductID = PR.ProductID
GROUP BY C.CustomerID, C.CustomerName
ORDER BY OrderCount DESC, TotalPurchaseValue DESC
LIMIT 3;
```

Interview Related Question on SQL

In interviews, SQL questions can range from basic to advanced, covering various aspects of database querying, optimization, and data manipulation. Here's a list of common SQL interview questions categorized by difficulty

Basic Level

1.Retrieve all records from a table.

2. Display specific columns from a table.

3.Use the WHERE clause to filter records based on a condition.

4.Order records using the ORDER BY clause.

5.Group records and calculate aggregate functions with the GROUP BY clause.

6.Join tables using INNER JOIN, LEFT JOIN, RIGHT JOIN.

7.Perform calculations using aggregate functions like SUM, COUNT, AVG.

8.Find distinct values in a column using DISTINCT.
Use the LIMIT clause to retrieve a specific number of records.

Intermidate Level

1.Retrieve the second highest or nth highest value from a column.

2.Write a query to find duplicates in a table.

3.Retrieve records based on multiple conditions using the AND and OR operators.

4.Handle NULL values in queries using IS NULL and IS NOT NULL.

5.Use subqueries (correlated and non-correlated) to solve complex problems.

6.Work with string functions like CONCAT, SUBSTRING, and LENGTH.

7.Utilize date functions like DATEADD, DATEDIFF, and DATEPART.

8.Write a query to find the most common value in a column (mode).

9.Update records in a table using the UPDATE statement.

Advanced Level

1.Retrieve the top N records for each group using window functions.

2.Write a query to find the Nth highest value without using OFFSET.

3.Optimize a slow-performing query using indexing.

4.Implement a self-join to find relationships within a table.

5.Write a query to find gaps in a sequence of numbers.

6.Use recursive queries to traverse hierarchical data.

7.Solve complex problems involving data partitioning with window functions.

8.Design a database schema for a specific use case, considering normalization and relationships.