SQL PRACTICE QUESTIONS

Practice Questions:

1. You're building a reporting system for an e-commerce platform. The company has three tables: "orders", "order_details", and "products". Each order consists of multiple products listed in the "order_details" table. Write a query to retrieve the order details, including the product name, quantity ordered, and the total price for each product. Ensure that products with no orders are also included in the result set.
   Here are simplified versions of the "orders", "order_details", and "products" tables:

   Orders Table:

Order_ID	Customer_ID	Order_date	Total_amount
1	101	2024-03-25	50.00
2	102	2024-03-26	30.00
3	103	2024-03-27	20.00
4	104	2024-03-27	15.00
5	105	2024-03-28	45.00

   Order_Details Table:

Order_id	Product_Id	Quantity
1	1	2
1	2	1

2	3	1
3	2	3
4	4	1
5	1	1

Products Table:

Product_id	Product_name	Price
1	Product A	10.00
2	Product B	15.00
3	Product C	20.00
4	Product D	25.00

ANSWER:

```
-- Create Orders table
CREATE TABLE Orders (
    Order_ID INT PRIMARY KEY,
    Customer_ID INT,
    Order_date DATE,
    Total_amount DECIMAL(10, 2)
);

-- Insert data into Orders table
INSERT INTO Orders (Order_ID, Customer_ID, Order_date,
Total_amount) VALUES
(1, 101, '2024-03-25', 50.00),
```

SQL PRACTICE QUESTIONS

```
(2, 102, '2024-03-26', 30.00),
(3, 103, '2024-03-27', 20.00),
(4, 104, '2024-03-27', 15.00),
(5, 105, '2024-03-28', 45.00);

-- Create Order_Details table
CREATE TABLE Order_Details (
    Order_id INT,
    Product_Id INT,
    Quantity INT,
    FOREIGN KEY (Order_id) REFERENCES Orders(Order_ID),
    FOREIGN KEY (Product_Id) REFERENCES Products(Product_id)
);

-- Insert data into Order_Details table
INSERT INTO Order_Details (Order_id, Product_Id, Quantity)
VALUES
(1, 1, 2),
(1, 2, 1),
(1, 2, 3),
(1, 3, 2),
(4, 1, 5),
(5, 1, 1);

-- Create Products table
CREATE TABLE Products (
    Product_id INT PRIMARY KEY,
    Product_name VARCHAR(50),
    Price DECIMAL(10, 2)
);

-- Insert data into Products table
INSERT INTO Products (Product_id, Product_name, Price) VALUES
(1, 'Product A', 10.00),
(2, 'Product B', 15.00),
(3, 'Product C', 20.00),
(4, 'Product D', 25.00);

SELECT p.Product_name,
```

SQL PRACTICE QUESTIONS

```
       COALESCE(SUM(od.Quantity), 0) AS Quantity_ordered,
       COALESCE(SUM(od.Quantity * pr.Price), 0) AS Total_price
FROM Products p
LEFT JOIN Order_Details od ON p.Product_id = od.Product_Id
LEFT JOIN Orders o ON od.Order_id = o.Order_ID
LEFT JOIN Products pr ON od.Product_Id = pr.Product_id
GROUP BY p.Product_name;
```

2. You're working for a social media platform that tracks user engagement metrics. The "posts" table contains data about user posts, including the number of likes each post receives. However, due to a recent system upgrade, some posts have duplicate entries in the database. Write an SQL query to calculate the average number of likes per post, ensuring that each post is counted only once.

post_id	likes
1	50
2	70
3	50
4	80
5	60
1	50
3	50
5	60

6	90
7	40

Answer:

```
SELECT post_id, AVG(likes) AS average_likes
FROM (
    SELECT DISTINCT post_id, likes
    FROM posts
) AS unique_posts
GROUP BY post_id;
```

3. You're analyzing customer feedback data for a software company. The "feedback" table stores feedback ratings provided by customers for different features of the software. Each feedback entry includes the feature name, the rating (on a scale of 1 to 5), and the customer ID. However, some customers have provided feedback for multiple features. Write an SQL query to calculate the average rating for each feature, ensuring that each customer's rating is counted only once per feature.

   Sample Data:

feedback_id	feature_name	rating	customer_id
1	Feature A	4	101
2	Feature B	5	101
3	Feature A	3	102

SQL PRACTICE QUESTIONS

4	Feature C	4	103
5	Feature B	5	104
6	Feature A	5	105
7	Feature C	4	101
8	Feature B	4	102
9	Feature A	3	103
10	Feature B	5	104
11	Feature C	5	105

Answer;

SQL PRACTICE QUESTIONS

```
-- Create Posts table
CREATE TABLE Posts (
    post_id INT,
    likes INT
);

-- Insert data into Posts table
INSERT INTO Posts (post_id, likes) VALUES
(1, 50),
(2, 70),
(3, 50),
(4, 80),
(5, 60),
(1, 50),
(3, 50),
(5, 60),
(6, 90),
(7, 40);

SELECT post_id, AVG(likes) AS average_likes
FROM (
    SELECT DISTINCT post_id, likes
    FROM posts
) AS unique_posts
GROUP BY post_id;
```

4. You're working for a retail company that sells products online. The "orders" table contains data about individual orders, including the order ID, customer ID, order date, and total amount. Write an SQL query to calculate the cumulative total amount of orders for each customer, ordered by the order date.

   Sample Data - Orders Table:

Order_id	Customer_id	Order_date	Total_amount
1	101	2024-03-01	50.00
2	102	2024-03-02	30.00
3	101	2024-03-03	20.00
4	103	2024-03-04	15.00
5	101	2024-03-05	45.00

Answer:

```
-- Create Orders table
CREATE TABLE Orders (
    Order_id INT PRIMARY KEY,
    Customer_id INT,
    Order_date DATE,
    Total_amount DECIMAL(10, 2)
);

-- Insert data into Orders table
INSERT INTO Orders (Order_id, Customer_id, Order_date,
```

SQL PRACTICE QUESTIONS

```
Total_amount) VALUES
(1, 101, '2024-03-01', 50.00),
(2, 102, '2024-03-02', 30.00),
(3, 101, '2024-03-03', 20.00),
(4, 103, '2024-03-04', 15.00),
(5, 101, '2024-03-05', 45.00);

-- SQL query to calculate cumulative total amount of orders
for each customer, ordered by order date
SELECT
    Order_id,
    Customer_id,
    Order_date,
    Total_amount,
    SUM(Total_amount) OVER (PARTITION BY Customer_id ORDER
BY Order_date) AS cumulative_total
FROM
    Orders
ORDER BY
    Customer_id, Order_date;
```

5. You're analyzing sales data for a company that operates in multiple regions. The "sales" table contains data about individual sales transactions, including the sale date, salesperson ID, and sale amount. Write an SQL query to calculate the average sale amount for each salesperson, considering the three most recent sales for each salesperson.

Sample Data - Sales Table:

Sale_id	Salesperson_id	Sale_id	Sale_amount
1	201	2024-03-01	100.00
2	202	2024-03-02	150.00

3	201	2024-03-03	200.00
4	203	2024-03-04	80.00
5	201	2024-03-05	120.00
6	202	2024-03-06	90.00
7	203	2024-03-07	110.00

Answer:

```
-- Create Sales table
CREATE TABLE Sales (
    Sale_id INT PRIMARY KEY,
    Salesperson_id INT,
    Sale_date DATE,
    Sale_amount DECIMAL(10, 2)
);

-- Insert data into Sales table
INSERT INTO Sales (Sale_id, Salesperson_id, Sale_date, Sale_amount)
VALUES
(1, 201, '2024-03-01', 100.00),
(2, 202, '2024-03-02', 150.00),
(3, 201, '2024-03-03', 200.00),
(4, 203, '2024-03-04', 80.00),
(5, 201, '2024-03-05', 120.00),
(6, 202, '2024-03-06', 90.00),
(7, 203, '2024-03-07', 110.00);

-- SQL query to calculate the average sale amount for each
```

```
salesperson, considering the three most recent sales for each
salesperson
SELECT
   Salesperson_id,
   AVG(Sale_amount) AS average_sale_amount
FROM (
   SELECT
      Salesperson_id,
      Sale_amount,
      ROW_NUMBER() OVER (PARTITION BY Salesperson_id ORDER BY
Sale_date DESC) AS row_num
   FROM
      Sales
) AS ranked_sales
WHERE
   row_num <= 3
GROUP BY
   Salesperson_id;

  ```
```

6. You're working for a company that sells products online. The "orders" table contains data about individual orders, including the order ID and total amount. The company wants to categorize orders based on their total amount into three categories: "High", "Medium", and "Low". Orders with a total amount greater than 100 should be classified as "High", between 50 and 100 as "Medium", and below 50 as "Low". Write an SQL query to retrieve the order ID, total amount, and the category of each order based on the total amount using a CASE statement.

   Sample data:
```

SQL PRACTICE QUESTIONS

order_id	total_amount
1	30.00
2	90.00
3	110.00
4	50.00
5	20.00

Answer:

```
      -- Create Orders table
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    total_amount DECIMAL(10, 2)
);

-- Insert data into Orders table
INSERT INTO orders (order_id, total_amount) VALUES
(1, 30.00),
(2, 90.00),
(3, 110.00),
(4, 50.00),
(5, 20.00);

-- SQL query to categorize orders based on total amount
SELECT
    order_id,
    total_amount,
    CASE
        WHEN total_amount > 100 THEN 'High'
```

SQL PRACTICE QUESTIONS

```
            WHEN total_amount BETWEEN 50 AND 100 THEN 'Medium'
            ELSE 'Low'
        END AS category
    FROM
        orders;
```

7. You're analyzing customer data for a marketing campaign. The "customers" table contains data about individual customers, including their age. The marketing team wants to segment customers into different age groups: "Young" (age <= 30), "Middle-aged" (age > 30 and age <= 50), and "Senior" (age > 50). Write an SQL query to retrieve the customer ID, age, and the age group of each customer using a CASE statement.

   Sample Data - Customers Table:

customer_id	age
1	25
2	40
3	55
4	30
5	20

Answer:

```
-- Create Customers table
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    age INT
);

-- Insert data into Customers table
INSERT INTO customers (customer_id, age) VALUES
(1, 25),
(2, 40),
(3, 55),
(4, 30),
(5, 20);

-- SQL query to segment customers into different age groups
SELECT
    customer_id,
    age,
    CASE
        WHEN age <= 30 THEN 'Young'
        WHEN age > 30 AND age <= 50 THEN 'Middle-aged'
        ELSE 'Senior'
    END AS age_group
FROM
    customers;
```

8. You're working for a company that tracks employee performance. The "employee_performance" table contains data about individual employees, including their employee ID, department, and performance score. The company wants to identify the top-performing employees in each department. Write an SQL query using a CTE to retrieve the employee ID, department, and performance score of the top-performing employee in each department.

SQL PRACTICE QUESTIONS

Sample Data - Employee_Performance Table:

employee_id	department	performance_score
1	Sales	95
2	HR	85
3	Sales	90
4	Finance	88
5	HR	92

Answer:

```
-- Create Employee_Performance table
CREATE TABLE employee_performance (
    employee_id INT PRIMARY KEY,
    department VARCHAR(50),
    performance_score INT
);

-- Insert data into Employee_Performance table
INSERT INTO employee_performance (employee_id, department,
performance_score) VALUES
(1, 'Sales', 95),
(2, 'HR', 85),
(3, 'Sales', 90),
(4, 'Finance', 88),
(5, 'HR', 92);

-- SQL query to retrieve the top-performing employee in each
department
WITH TopEmployees AS (
    SELECT
        employee_id,
        department,
        performance_score,
        ROW_NUMBER() OVER (PARTITION BY department ORDER BY
performance_score DESC) AS rank
    FROM
        employee_performance
)
SELECT
    employee_id,
    department,
    performance_score
FROM
    TopEmployees
WHERE
    rank = 1;
```

9. You're analyzing sales data for a retail company. The "sales" table contains data about individual sales transactions, including the sale date, product ID, and sale amount. The company wants to calculate the total sales amount for each product, as well as the percentage of total sales amount contributed by each product. Write an SQL query using a CTE to retrieve the product ID, total sales amount, and percentage of total sales amount contributed by each product.
   Sample Data - Sales Table:

sale_id	product_id	sale_amount

1	101	100.00
2	102	150.00
3	101	200.00
4	103	80.00
5	101	120.00

Answer:

```
-- Create Sales table
CREATE TABLE sales (
    sale_id INT PRIMARY KEY,
    product_id INT,
    sale_amount DECIMAL(10, 2)
);

-- Insert data into Sales table
INSERT INTO sales (sale_id, product_id, sale_amount) VALUES
(1, 101, 100.00),
(2, 102, 150.00),
(3, 101, 200.00),
(4, 103, 80.00),
(5, 101, 120.00);

-- SQL query to calculate total sales amount for each product
and percentage of total sales amount contributed by each
product
WITH ProductSales AS (
    SELECT
        product_id,
        SUM(sale_amount) AS total_sales_amount,
```

SQL PRACTICE QUESTIONS

```
        SUM(sale_amount) * 100.0 / SUM(SUM(sale_amount)) OVER
() AS sales_percentage
    FROM
        sales
    GROUP BY
        product_id
)
SELECT
    product_id,
    total_sales_amount,
    ROUND(sales_percentage, 2) AS sales_percentage
FROM
    ProductSales;
```

10. You're working for a company that tracks employee performance. The "employees" table contains data about individual employees, including their employee ID, department ID, and salary. The company wants to find all employees whose salary is greater than the average salary of their department. Write an SQL query to retrieve the employee ID, department ID, and salary of such employees using a correlated subquery.

    Sample Data - Employees Table:

employee_id	department_id	salary
1	101	50000
2	102	60000
3	101	55000
4	103	48000

SQL PRACTICE QUESTIONS

5	102	62000

Answer:

```
 -- Create Employees table
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    department_id INT,
    salary DECIMAL(10, 2)
);

-- Insert data into Employees table
INSERT INTO employees (employee_id, department_id, salary)
VALUES
(1, 101, 50000.00),
(2, 102, 60000.00),
(3, 101, 55000.00),
(4, 103, 48000.00),
(5, 102, 62000.00);

-- SQL query to retrieve employee ID, department ID, and
salary of employees whose salary is greater than the average
salary of their department
SELECT
    employee_id,
    department_id,
    salary
FROM
    employees e1
WHERE
    salary > (
        SELECT
            AVG(salary)
        FROM
            employees e2
        WHERE
            e1.department_id = e2.department_id
    );
```

SQL PRACTICE QUESTIONS

11. You're analyzing sales data for a retail company. The "products" table contains data about individual products, including their product ID and unit price. The company wants to find all products whose unit price is lower than the average unit price of products with the same category. Write an SQL query to retrieve the product ID, unit price, and category of such products using a correlated subquery.

    Sample Data - Products Table:

product_id	category	unit_price
1	Electronics	500
2	Clothing	40
3	Electronics	800
4	Clothing	35
5	Electronics	600

Answer:

```
-- Create Products table
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    category VARCHAR(50),
    unit_price DECIMAL(10, 2)
);

-- Insert data into Products table
INSERT INTO products (product_id, category, unit_price) VALUES
```

SQL PRACTICE QUESTIONS

```
(1, 'Electronics', 500.00),
(2, 'Clothing', 40.00),
(3, 'Electronics', 800.00),
(4, 'Clothing', 35.00),
(5, 'Electronics', 600.00);

-- SQL query to retrieve product ID, unit price, and category
of products whose unit price is lower than the average unit
price of products with the same category
SELECT
    product_id,
    unit_price,
    category
FROM
    products p1
WHERE
    unit_price < (
        SELECT
            AVG(unit_price)
        FROM
            products p2
        WHERE
            p1.category = p2.category
    );
```