Case Study: Solving SQL Challenges

Introduction:
In this case study, we will explore a database scenario for a fictional company called "TechMart," which sells various electronic products. We'll present a series of SQL challenges, ranging from medium to advanced difficulty levels, that require you to use your SQL skills to extract, manipulate, and analyze data from the TechMart database. The case study will cover topics such as querying, aggregation, joins, subqueries, and advanced SQL functions.

Database Schema:
TechMart's database consists of the following tables:

1. **products:** Contains information about the electronic products sold by TechMart.
   - Columns: `product_id`, `product_name`, `category_id`, `unit_price`, `stock_quantity`.

2. **categories:** Contains information about the different product categories.
   - Columns: `category_id`, `category_name`.

3. **orders:** Contains information about the orders placed by customers.
   - Columns: `order_id`, `customer_id`, `order_date`.

4. **order_items:** Contains information about the products included in each order.
   - Columns: `order_item_id`, `order_id`, `product_id`, `quantity`, `total_price`.

Case Study Challenges:

1. Retrieve Product Information: (Medium)
Write a SQL query to retrieve the product_id, product_name, unit_price, and stock_quantity for all products in the "Laptops" category.

2. Top Selling Categories: (Medium)
Write a SQL query to determine the top 3 product categories based on the total quantity of products sold. The result should include the category_id, category_name, and the total quantity sold across all orders.

3. Customer Purchase History: (Medium)
Write a SQL query that shows the order_id, order_date, product_id, product_name, and quantity for each product purchased by a specific customer with customer_id = 1001.

4. Revenue by Category: (Advanced)
Write a SQL query to calculate the total revenue generated by each product category, considering the unit price and quantity sold for each product. The result should display the category_id, category_name, and total revenue.

5. Monthly Sales Growth: (Advanced)
Write an SQL query to calculate the monthly sales growth percentage for TechMart. The result should include the month and year of the orders and the corresponding sales growth percentage compared to the previous month.

6. Rank Customers by Total Spending: (Advanced)
Write a SQL query to rank TechMart's customers based on their total spending (sum of total_price) in descending order. The result should display the customer_id and their respective rank.

7. Product Recommendations: (Advanced)
Write a SQL query that suggests three product recommendations to customers who have purchased products in the "Smartphones" category. The recommendations should be based on the purchasing history of other customers who bought products from the same category.

THE SQL CODE TO IMPORT DUMMY TABLES TO SOLVE THE QUESTIONS ARE LISTED BELOW

```
-- Create the 'categories' table
CREATE TABLE categories (
  category_id INT PRIMARY KEY,
  category_name VARCHAR(50)
);

-- Insert data into the 'categories' table
INSERT INTO categories (category_id, category_name) VALUES
(1, 'Laptops'),
(2, 'Smartphones'),
(3, 'Tablets'),
(4, 'Accessories'),
(5, 'Cameras');

-- Create the 'products' table
CREATE TABLE products (
  product_id INT PRIMARY KEY,
  product_name VARCHAR(100),
  category_id INT,
  unit_price DECIMAL(10, 2),
  stock_quantity INT
);

-- Insert data into the 'products' table
```

```
INSERT INTO products (product_id, product_name, category_id, unit_price, stock_quantity)
VALUES
(101, 'Dell XPS 13', 1, 1200.00, 50),
(102, 'iPhone 12', 2, 999.99, 100),
(103, 'Samsung Galaxy S21', 2, 899.00, 80),
(104, 'iPad Pro', 3, 799.00, 60),
(105, 'Logitech Wireless Mouse', 4, 29.99, 200),
(106, 'JBL Bluetooth Speaker', 4, 89.99, 150),
(107, 'Nikon D850 DSLR Camera', 5, 2499.00, 30),
(108, 'Sony Alpha A7 III', 5, 1999.00, 40);

-- Create the 'orders' table
CREATE TABLE orders (
  order_id INT PRIMARY KEY,
  customer_id INT,
  order_date DATE
);

-- Insert data into the 'orders' table
INSERT INTO orders (order_id, customer_id, order_date) VALUES
(1001, 5001, '2023-07-01'),
(1002, 5002, '2023-07-15'),
(1003, 5001, '2023-07-20'),
(1004, 5003, '2023-07-22'),
(1005, 5004, '2023-07-25');

-- Create the 'order_items' table
CREATE TABLE order_items (
  order_item_id INT PRIMARY KEY,
  order_id INT,
  product_id INT,
  quantity INT,
  total_price DECIMAL(10, 2)
);

-- Insert data into the 'order_items' table
INSERT INTO order_items (order_item_id, order_id, product_id, quantity, total_price) VALUES
(2001, 1001, 101, 2, 2400.00),
(2002, 1001, 102, 1, 999.99),
(2003, 1002, 104, 3, 2397.00),
(2004, 1003, 103, 2, 1798.00),
(2005, 1003, 105, 5, 149.95),
(2006, 1003, 106, 2, 179.98),
(2007, 1004, 107, 1, 2499.00),
```

```
(2008, 1004, 106, 3, 269.97),
(2009, 1005, 102, 2, 1999.98),
(2010, 1005, 108, 1, 1999.00),
(2011, 1005, 105, 2, 59.98),
(2012, 1005, 103, 1, 899.00);
```