Comprehensive SQL Problem Set for E-Commerce Database Management and Analysis

Designed By

Nayeem Islam

Table of Contents

Designed By

Nayeem Islam

Primary Task: Creating an E-Commerce Database with Comprehensive Tables

Objective:
As a database administrator for an e-commerce company, your task is to create a new SQL database named `ECommerceDB`. This database will support various business operations including customer management, product cataloging, order processing, sales analysis, handling returns, capturing customer feedback, and managing shipping.

Task Details:

Create the 'ECommerceDB' Database:

- Focus on creating a structured and efficient database that can handle e-commerce operations.

Design the Required Tables:

- `Customers` Table:
  - Columns: `CustomerID` (INT, Primary Key), `FirstName` (VARCHAR), `LastName` (VARCHAR), `Email` (VARCHAR), `SignUpDate` (DATE).
- `Products` Table:
  - Columns: `ProductID` (INT, Primary Key), `ProductName` (VARCHAR), `Price` (DECIMAL), `StockQuantity` (INT), `CategoryID` (INT, Foreign Key to `Categories` table).
- `Categories` Table:
  - Columns: `CategoryID` (INT, Primary Key), `CategoryName` (VARCHAR).
- `Orders` Table:
  - Columns: `OrderID` (INT, Primary Key), `CustomerID` (INT, Foreign Key to `Customers` table), `OrderDate` (DATE), `TotalAmount` (DECIMAL), `RegionID` (INT, Foreign Key to `Regions` table).
- `OrderDetails` Table:
  - Columns: `OrderDetailID` (INT, Primary Key), `OrderID` (INT, Foreign Key to `Orders` table), `ProductID` (INT, Foreign Key to `Products` table), `Quantity` (INT), `UnitPrice` (DECIMAL).
- `Regions` Table:
  - Columns: `RegionID` (INT, Primary Key), `RegionName` (VARCHAR).
- `Returns` Table:
  - Columns: `ReturnID` (INT, Primary Key), `OrderID` (INT, Foreign Key to `Orders` table), `ProductID` (INT, Foreign Key to `Products` table), `ReturnDate` (DATE), `QuantityReturned` (INT), `RefundAmount` (DECIMAL).
- `Reviews` Table:

Designed By

Nayeem Islam

        - Columns: `ReviewID` (INT, Primary Key), `ProductID` (INT, Foreign Key to `Products` table), `CustomerID` (INT, Foreign Key to `Customers` table), `Rating` (INT), `ReviewText` (TEXT), `ReviewDate` (DATE).
    - `ShippingAddresses` Table:
        - Columns: `AddressID` (INT, Primary Key), `CustomerID` (INT, Foreign Key to `Customers` table), `Address` (TEXT).

Requirements:

- Assign appropriate data types to each column.
- Establish primary keys for unique identification in each table.
- Define foreign key relationships to maintain data integrity.
- Consider indexing columns that are frequently used in search queries for performance optimization.

Goal:

By completing this task, you will create a well-structured and comprehensive database, capable of handling and streamlining the complex data needs of an e-commerce company. This will facilitate efficient data management, analysis, and decision-making processes across various business functions.

Designed By

Nayeem Islam

Problem 1: Basic Customer Data Retrieval

Scenario:
You are working for an e-commerce company as a junior SQL developer. Your first task is to familiarize yourself with the `Customers` database. The marketing team needs a report on the customers to plan their new campaign.

Database Structure:

- `Customers` table with columns: `CustomerID`, `FirstName`, `LastName`, `Email`, `SignUpDate`, `LastPurchaseDate`.

Task:

Write a query to retrieve the following information:

- List all customers who have signed up in the last 30 days.
- Display their `CustomerID`, `FirstName`, `LastName`, and `Email`.
- Sort the results by `SignUpDate` in descending order.

Objective:

This problem will help you practice:

- Basic `SELECT` statements.
- Filtering data using `WHERE` clause.
- Sorting results using `ORDER BY`.

Designed By

Nayeem Islam

Problem 2: Aggregating Product Sales Data

Scenario:
You are now tasked with helping the sales team at your e-commerce company. They need insights into the product sales for the last quarter.

Database Structure:

- `Products` table with columns: `ProductID`, `ProductName`, `Price`.
- `Orders` table with columns: `OrderID`, `CustomerID`, `OrderDate`.
- `OrderDetails` table with columns: `OrderDetailID`, `OrderID`, `ProductID`, `Quantity`.

Task:

Write a query to:

- Calculate the total sales (quantity * price) for each product in the last quarter.
- Display `ProductName`, and the calculated total sales as `TotalSales`.
- Include only those products that have been sold in the last quarter.
- Order the results by `TotalSales` in descending order.

Objective:

This problem will help you practice:

- Joining multiple tables.
- Using aggregate functions like `SUM`.
- Filtering data based on a date range.
- Using `GROUP BY` to aggregate data.

Designed By

Nayeem Islam

Problem 3: Customer Segmentation Based on Purchase Behavior

Scenario:
Your e-commerce company is planning a personalized marketing campaign. The marketing team needs to segment customers based on their purchasing behavior.

Database Structure:

- `Customers` table with columns: `CustomerID`, `FirstName`, `LastName`, `Email`.
- `Orders` table with columns: `OrderID`, `CustomerID`, `OrderDate`.
- `OrderDetails` table with columns: `OrderDetailID`, `OrderID`, `ProductID`, `Quantity`, `UnitPrice`.

Task:

Write a query to:

- Segment customers into three categories based on their total spending: 'Low' (less than $100), 'Medium' ($100 to $500), and 'High' (more than $500).
- Calculate the total spending for each customer.
- Display `CustomerID`, `FirstName`, `LastName`, `Email`, and a new column `SpendingCategory` indicating their segment.
- Order the results by the total spending in descending order.

Objective:

This problem will help you practice:

- Complex joins across multiple tables.
- Using `CASE` statements for conditional logic.
- Aggregate functions and grouping data.
- Creating calculated fields.

Designed By

Nayeem Islam

Problem 4: Inventory Management and Reorder Level Alert

Scenario:
You are now working with the inventory management team of your e-commerce company. They need an automated way to identify products that are low in stock and require reordering.

Database Structure:

- `Products` table with columns: `ProductID`, `ProductName`, `Price`, `StockQuantity`.
- `OrderDetails` table with columns: `OrderDetailID`, `OrderID`, `ProductID`, `Quantity`.

Task:

Write a query to:

- Identify products whose `StockQuantity` is less than the reorder level. Assume the reorder level is set at a quantity of 20.
- Calculate the total quantity sold for each of these products in the past month.
- Display `ProductID`, `ProductName`, `StockQuantity`, and `TotalQuantitySold`.
- Include a column `ReorderNeeded` that shows 'Yes' if `StockQuantity` is less than 10 and 'No' otherwise.
- Order the results by `StockQuantity` in ascending order.

Objective:

This problem will help you practice:

- Using subqueries or joins to combine sales and stock data.
- Implementing conditional logic with `CASE`.
- Filtering data with specific criteria.
- Aggregating sales data based on a date range.

Problem 5: Tracking Customer Order Frequency and Last Purchase

Scenario:
As part of a customer retention strategy, your e-commerce company wants to identify customers who have not made a purchase recently and those who order frequently.

Database Structure:

- `Customers` table with columns: `CustomerID`, `FirstName`, `LastName`, `Email`, `SignUpDate`.
- `Orders` table with columns: `OrderID`, `CustomerID`, `OrderDate`.

Task:

Write a query to:

- Calculate the total number of orders placed by each customer.
- Identify the date of the last order placed by each customer.
- Segment customers based on their order frequency: 'Frequent' (5 or more orders), 'Occasional' (2-4 orders), 'Rare' (1 order).
- Display `CustomerID`, `FirstName`, `LastName`, `Email`, `TotalOrders`, `LastOrderDate`, and `OrderFrequency`.
- Order the results by `LastOrderDate` in ascending order.

Objective:

This problem will help you practice:

- Calculating counts and using aggregate functions like `MAX`.
- Implementing `GROUP BY` to consolidate data per customer.
- Using `CASE` statements for customer segmentation.
- Managing date operations.

Designed By

Nayeem Islam

Problem 6: Analyzing Sales Performance by Region

Scenario:
Your e-commerce company is expanding its market analysis. The sales team needs a report on sales performance by region to strategize their efforts.

Database Structure:

- `Orders` table with columns: `OrderID`, `CustomerID`, `OrderDate`, `TotalAmount`, `RegionID`.
- `Regions` table with columns: `RegionID`, `RegionName`.
- `OrderDetails` table with columns: `OrderDetailID`, `OrderID`, `ProductID`, `Quantity`, `UnitPrice`.

Task:

Write a query to:

- Calculate the total sales (`TotalAmount`) for each region.
- Count the number of orders placed in each region.
- Calculate the average order value for each region.
- Display `RegionName`, `TotalSales`, `TotalOrders`, and `AverageOrderValue`.
- Order the results by `TotalSales` in descending order.

Objective:

This problem will help you practice:

- Joining multiple tables.
- Using aggregate functions like `SUM`, `COUNT`, and `AVG`.
- Grouping data by a specific column (`RegionID`).
- Handling complex calculations and presenting them in a meaningful way.

Designed By

Nayeem Islam

Problem 7: Product Returns and Refund Analysis

Scenario:
You are tasked with analyzing product returns and refunds for your e-commerce company, which is critical for understanding customer satisfaction and product quality.

Database Structure:

- `Orders` table with columns: `OrderID`, `CustomerID`, `OrderDate`, `TotalAmount`.
- `OrderDetails` table with columns: `OrderDetailID`, `OrderID`, `ProductID`, `Quantity`, `UnitPrice`.
- `Returns` table with columns: `ReturnID`, `OrderID`, `ProductID`, `ReturnDate`, `QuantityReturned`, `RefundAmount`.

Task:

Write a query to:

- Identify the total number of returns and the total refund amount for each product.
- Calculate the return rate for each product (returns/total quantity sold).
- Display `ProductID`, `TotalReturns`, `TotalRefundAmount`, and `ReturnRate`.
- Include only products that have been returned at least once.
- Order the results by `TotalRefundAmount` in descending order.

Objective:

This problem will help you practice:

- Joining multiple tables to combine sales and return data.
- Using aggregate functions like `SUM` and `COUNT`.
- Calculating ratios and percentages.
- Filtering and sorting data based on specific conditions.

Designed By

Nayeem Islam

Problem 8: Monitoring Stock Levels and Identifying Slow-Moving Products

Scenario:
The inventory management team at your e-commerce company needs assistance in identifying products that are slow-moving and may need promotional efforts to increase sales.

Database Structure:

- `Products` table with columns: `ProductID`, `ProductName`, `Price`, `StockQuantity`.
- `OrderDetails` table with columns: `OrderDetailID`, `OrderID`, `ProductID`, `Quantity`.

Task:

Write a query to:

- Calculate the total quantity sold for each product over the past year.
- Identify products with a high stock quantity but low sales volume (slow-moving products).
- Define slow-moving as products with `StockQuantity` greater than 50 and total quantity sold less than 10 in the past year.
- Display `ProductID`, `ProductName`, `StockQuantity`, and `TotalQuantitySold`.
- Include a column `Status` indicating 'Slow-Moving' for the relevant products.
- Order the results by `TotalQuantitySold` in ascending order.

Objective:

This problem will help you practice:

- Combining sales and inventory data to analyze product movement.
- Using date functions to filter sales within a specific time frame.
- Implementing conditional logic using `CASE`.
- Handling complex filtering criteria.

Designed By

Nayeem Islam

Problem 9: Analyzing Customer Reviews and Ratings for Products

Scenario:
Your e-commerce company is focusing on improving customer experience. The product team needs an analysis of customer reviews and ratings to identify top-rated and low-rated products.

Database Structure:

- `Products` table with columns: `ProductID`, `ProductName`.
- `Reviews` table with columns: `ReviewID`, `ProductID`, `CustomerID`, `Rating`, `ReviewText`, `ReviewDate`.

Task:

Write a query to:

- Calculate the average rating for each product.
- Count the number of reviews for each product.
- Identify products with an average rating below 3.0 as 'Low-Rated' and above 4.5 as 'Top-Rated'.
- Display `ProductID`, `ProductName`, `AverageRating`, `NumberOfReviews`, and `RatingCategory`.
- Order the results by `AverageRating` in ascending order.

Objective:

This problem will help you practice:

- Joining tables to correlate products with reviews.
- Using aggregate functions like `AVG` and `COUNT`.
- Implementing conditional logic with `CASE` for categorization.
- Presenting the results in an ordered and insightful manner.

Designed By

Nayeem Islam

Problem 10: Time Series Analysis of Monthly Sales Trends

Scenario:
The management team at your e-commerce company is interested in understanding the monthly sales trends to make informed decisions for marketing and inventory planning.

Database Structure:

- `Orders` table with columns: `OrderID`, `CustomerID`, `OrderDate`, `TotalAmount`.
- `OrderDetails` table with columns: `OrderDetailID`, `OrderID`, `ProductID`, `Quantity`, `UnitPrice`.

Task:

Write a query to:

- Calculate the total sales for each month over the past year.
- Break down the sales by product category (assume a `CategoryID` column exists in the `Products` table).
- Display the month, `CategoryID`, and total sales for that category in the month.
- Order the results by month and then by sales in descending order within each month.

Objective:

This problem will help you practice:

- Handling date and time functions to extract month and year.
- Joining and aggregating data across multiple tables.
- Grouping data by multiple columns (month and `CategoryID`).
- Sorting data in a multi-level order.

Designed By

Nayeem Islam

Problem 11: Identifying Cross-Selling Opportunities through Customer Purchase Patterns

Scenario:
Your e-commerce company wants to boost sales by identifying cross-selling opportunities. The marketing team needs insights into customer purchase patterns to recommend products that are frequently bought together.

Database Structure:

- `Orders` table with columns: `OrderID`, `CustomerID`, `OrderDate`.
- `OrderDetails` table with columns: `OrderDetailID`, `OrderID`, `ProductID`, `Quantity`.
- `Products` table with columns: `ProductID`, `ProductName`.

Task:

Write a query to:

- Identify pairs of products that are frequently bought together in the same order.
- Count the number of times each pair appears in orders.
- Display the `ProductID` and `ProductName` for both products in the pair and the `PairCount`.
- Order the results by `PairCount` in descending order.

Objective:

This problem will help you practice:

- Writing complex queries to analyze relationships between products in orders.
- Using self-joins to correlate products within the same order.
- Implementing aggregation with `COUNT`.
- Presenting data in a meaningful way to inform business strategies.

Designed By

Nayeem Islam

Problem 12: Customer Lifetime Value Calculation

Scenario:
The financial team at your e-commerce company is interested in calculating the Customer Lifetime Value (CLV) to make strategic decisions in marketing and customer relationship management.

Database Structure:

- `Customers` table with columns: `CustomerID`, `FirstName`, `LastName`, `Email`, `SignUpDate`.
- `Orders` table with columns: `OrderID`, `CustomerID`, `OrderDate`, `TotalAmount`.

Task:

Write a query to:

- Calculate the total revenue generated from each customer since their sign-up.
- Determine the duration (in years) since each customer's sign-up.
- Calculate the Customer Lifetime Value as total revenue divided by the number of years since sign-up.
- Display `CustomerID`, `FirstName`, `LastName`, `TotalRevenue`, `YearsSinceSignUp`, and `CLV`.
- Order the results by `CLV` in descending order.

Objective:

This problem will help you practice:

- Performing calculations across multiple tables.
- Handling date calculations to determine durations.
- Using aggregate functions like `SUM`.
- Creating complex calculated fields (like `CLV`).

Designed By

Nayeem Islam

Problem 13: Optimizing Shipping Costs through Order Consolidation

Scenario:
Your e-commerce company is looking to optimize shipping costs. The logistics team wants to explore the possibility of consolidating multiple orders from the same customer into a single shipment, if they are placed within a short time frame of each other.

Database Structure:

- `Orders` table with columns: `OrderID`, `CustomerID`, `OrderDate`, `ShippingAddressID`, `TotalAmount`.
- `OrderDetails` table with columns: `OrderDetailID`, `OrderID`, `ProductID`, `Quantity`.
- `ShippingAddresses` table with columns: `AddressID`, `CustomerID`, `Address`.

Task:

Write a query to:

- Identify orders that can be potentially consolidated. Consider orders eligible for consolidation if they are made by the same customer within 3 days of each other and have the same `ShippingAddressID`.
- Display the `CustomerID`, first `OrderID`, second `OrderID`, and the dates of these orders.
- Also, calculate the potential saving in shipping costs (assume a flat rate of $5 per order).
- Order the results by `CustomerID` and then by the dates of the orders.

Objective:

This problem will help you practice:

- Using self-joins to compare rows within the same table.
- Implementing date calculations to find orders within a specific time frame.
- Calculating potential savings based on business rules.
- Sorting and presenting data to support logistical decisions.

Problem 14: Forecasting Future Inventory Needs Based on Seasonal Sales Trends

Scenario:

As part of strategic planning, your e-commerce company wants to forecast future inventory needs. The inventory management team needs an analysis of seasonal sales trends to predict the stock requirements for the upcoming seasons.

Database Structure:

- `Products` table with columns: `ProductID`, `ProductName`, `CategoryID`, `StockQuantity`.
- `Orders` table with columns: `OrderID`, `OrderDate`.
- `OrderDetails` table with columns: `OrderDetailID`, `OrderID`, `ProductID`, `Quantity`.
- `Categories` table with columns: `CategoryID`, `CategoryName`.

Task:

Write a query to:

- Analyze the sales quantity of products in each category per season (define seasons as Winter, Spring, Summer, Autumn).
- Calculate the average sales quantity per season for the last three years.
- Predict the required stock for each category for the upcoming season based on the average sales quantity.
- Display `CategoryName`, `Season`, `AverageSalesQuantity`, and `PredictedStockRequirement`.
- Order the results by `CategoryName` and then by `Season`.

Objective:

This problem will help you practice:

- Handling complex date operations to categorize data into seasons.
- Joining multiple tables to consolidate necessary data.
- Performing historical data analysis to forecast future needs.
- Using aggregate functions for calculating averages and predictions.

Problem 15: Comprehensive Customer Behavior Analysis for Targeted Marketing

Scenario:
For a targeted marketing campaign, your e-commerce company wants a comprehensive analysis of customer behavior. The marketing team is interested in understanding the buying patterns, product preferences, and activity levels of different customer segments.

Database Structure:

- `Customers` table with columns: `CustomerID`, `FirstName`, `LastName`, `Email`, `SignUpDate`.
- `Orders` table with columns: `OrderID`, `CustomerID`, `OrderDate`, `TotalAmount`.
- `OrderDetails` table with columns: `OrderDetailID`, `OrderID`, `ProductID`, `Quantity`.
- `Products` table with columns: `ProductID`, `ProductName`, `CategoryID`.
- `Categories` table with columns: `CategoryID`, `CategoryName`.

Task:

Write a query to:

- Segment customers based on their total spending: 'High Spenders', 'Medium Spenders', 'Low Spenders'.
- For each segment, calculate the most popular product category.
- Identify the frequency of orders (e.g., weekly, monthly).
- Display `CustomerID`, `Segment`, `MostPopularCategory`, and `OrderFrequency`.
- Additionally, identify customers who have not made any purchase in the last six months.
- Order the results by `Segment` and then by `MostPopularCategory`.

Objective:

This problem will help you practice:

- Segmenting data based on calculated criteria.
- Joining multiple tables to extract comprehensive insights.
- Using aggregate functions to find popular categories and spending patterns.
- Implementing advanced conditional logic and date calculations.

Designed By

Nayeem Islam