Approach for Solving SQL Tasks – BookstoreDB

This document outlines the approach used to complete the SQL tasks related to the **BookstoreDB** database, including database creation, data insertion, querying, and advanced SQL operations.

1. **Database Creation**

   Approach:

   - The **BookstoreDB** database was created using the CREATE DATABASE statement.
   - After creating the database, the USE command was used to switch to the **BookstoreDB** context.
   - The tables were designed to represent the necessary entities in the bookstore system: Books, Authors, Categories, Orders, and OrderDetails.
   - Each table was created using the CREATE TABLE statement, defining primary keys (Primary Key constraint) and foreign keys (Foreign Key constraint) to establish relationships between tables.

2. **Data Insertion**

   Approach:

   - The task involved inserting sample data into the **Authors**, **Categories**, **Books**, **Orders**, and **OrderDetails** tables using the INSERT INTO statement. After creating the database, the USE command was used to switch to the **BookstoreDB** context.
   - Each table was populated with at least 5 authors, 5 categories, 10 books, 3 orders, and their respective order details.

3. **Querying and Analysis**

   Basic Queries:

   - **Retrieve all books along with their authors and categories**:
     This query uses JOIN operations to link the Books, Authors, and Categories tables. The INNER JOIN is used to retrieve the relevant information.

- **Find books that are out of stock (StockQuantity = 0)**:
  A simple SELECT query with a WHERE clause filters books with StockQuantity=0

Aggregate Functions:

- **Total revenue generated from all orders**:
  A SUM function was used to calculate the total revenue by multiplying the Price of each book by the Quantity ordered in OrderDetails.

- **Number of books available in each category**:
  The COUNT function was used along with GROUP BY to get the number of books for each category.

Joins:

- **List all orders along with the customer name, order date, book titles, and quantity ordered**:
  The JOIN operation was used to link the Orders, OrderDetails, and Books tables to retrieve the required data.

Subqueries:

- **Most expensive book in the Poetry category**:
  A subquery was used to get the CategoryID for the "Poetry" category. The outer query then retrieves the most expensive book in that category.

- **Customers who have ordered more than 5 books in a single order**:
  The HAVING clause was used with GROUP BY to filter customers who have ordered more than 5 books in total for a single order.

Advanced Queries:

- **Identify authors whose books have generated revenue exceeding $500**:
  To solve this task, we need to calculate the revenue generated by each book. Revenue is calculated by multiplying the Price of the book by the Quantity ordered (from the OrderDetails table. The SUM function is used to calculate the total revenue per author. A JOIN operation is performed between Books, Authors, and OrderDetails to link the relevant data. The HAVING clause is used to filter authors whose total revenue exceeds $500.

- **Calculate the stock value (price × stock quantity) of all books and list the top 3 books by stock value**:
  The stock value of each book is calculated by multiplying its Price with the StockQuantity. To identify the top 3 books by stock value, the ORDER BY clause is used to sort the books in descending order based on the calculated stock value. The TOP 3 clause is used to retrieve only the top 3 books.

Stored Procedures:

A stored procedure, GetBooksByAuthor, was created to accept an AuthorID as input and return all books by that author. The procedure uses a SELECT query with a WHERE clause to filter books by AuthorID.

Views:

A view named TopSellingBooks was created to list the top 5 books based on total quantity sold. This required joining the Books and OrderDetails tables and using GROUP BY with SUM(Quantity) to calculate total sales per book.

Index Creation:

An index was created on the Title column of the Books table to optimize search performance. This is helpful for queries that frequently search for books by their title.