SQL Take Home Tasks

File Structure

1. **Database Design and Creation**
   - **File Name: BookstoreDB_Creation.sql**
   - **Content: Scripts to create the database and define all tables, along with necessary relationships and constraints.**
2. **Data Insertion**
   - **File Name: BookstoreDB_Insertion.sql**
   - **Content: Scripts for inserting sample data into the tables.**
3. **Querying and Analysis**
   - **File Name: BookstoreDB_Queries.sql**
   - **Content: All task-specific queries, such as basic retrieval, aggregate functions, joins, subqueries, and advanced queries.**
4. **Extras**
   - **File Name: BookstoreDB_Extras.sql**
   - **Content: Indexes, views, and stored procedures.**

Approach for Each Task

1. Basic Queries

- **Objective:** Perform fundamental operations like retrieving and filtering data.
- **Approach:**
  - Used joins (JOIN) to fetch related data from multiple tables.
  - Implemented WHERE conditions to filter records, such as finding books out of stock.
  - Ensured simplicity and clarity for new users to the database schema.

Example Tasks:

- Querying all books along with author and category details.
- Finding books with StockQuantity = 0.

2. Aggregate Functions

- **Objective:** Summarize data using mathematical functions.
- **Approach:**
  - Used aggregate functions like SUM, COUNT, and ROUND.
  - Grouped data by relevant columns to perform per-category or per-author analysis.

Example Tasks:

- Total revenue calculation by multiplying Quantity and Price.
- Count of books available per category.

3. Joins

- **Objective:** Combine records from two or more tables based on relationships.
- **Approach:**
    - Used INNER JOIN for standard associations (e.g., Orders and Books).
    - Utilized LEFT JOIN to ensure no missing data for non-existent references (e.g., Categories with no books).

Example Task:

- Listing all orders with customer name, order date, book titles, and quantities.

4. Subqueries

- **Objective:** Retrieve results with dependent or embedded queries.
- **Approach:**
    - Implemented nested SELECT statements for targeted data like specific categories or expensive books.
    - Designed subqueries for queries with dynamic conditions (e.g., fetching CategoryID).

Example Task:

- Finding the most expensive book in a given category.

5. Advanced Queries

- **Objective:** Solve complex tasks like revenue breakdown and multi-level operations.
- **Approach:**
    - Employed HAVING to filter groups post-aggregation.
    - Combined calculations for operations like revenue generation per author.

Example Tasks:

- Identifying authors with revenue exceeding a threshold.
- Listing books by their stock value in descending order.

6. Stored Procedures

- **Objective:** Define reusable and dynamic database operations.
- **Approach:**
    - Created a stored procedure GetBooksByAuthor with optional parameters.
    - Added functionality for handling NULL values to retrieve all books if no specific author is provided.

Example Usage:

- Retrieve books dynamically by supplying an AuthorID parameter.

7. Views

- **Objective:** Provide pre-defined virtual tables for common tasks.
- **Approach:**
    - Defined a view TopSellingBooks to list the top 5 books by total sales.
    - Utilized aggregate functions and proper ordering for dynamic updates to the view content.

Example Task:

- View-based reporting for frequently sold books.

8. Indexes

- **Objective:** Enhance performance for search-intensive tasks.
- **Approach:**
    - Created a non-clustered index on the Title column in the Books table to optimize title-based searches.
    - Ensured unique constraint where applicable to avoid duplicate records.

Example Task:

- Efficient lookup operations for book titles.

Error Handling

- Checked all scripts with test data to ensure no runtime errors.
- Handled NULL scenarios in queries and procedures.

Data Integrity

- Verified foreign key dependencies before inserting related records.
- Ensured no orphan records exist due to improper inserts/deletes.

Scalability

- Used modularized SQL scripts, allowing easy updates or additions to the database structure.
- Applied indexing strategically to ensure that growing data does not degrade performance.

Tools Used

- **Database Management System:** MySQL Server
- **IDE:** SQL Server Management Studio (SSMS)
- **Testing:** Data verification with sample data and runtime query execution