Online Bookstore SQL Project Documentation

1. Project Overview

This document provides a comprehensive overview of my SQL project for an online bookstore, detailing the creation and management of a relational database named OnlineBookstore. The project involves establishing the database, defining its table structures, importing sample data, and executing a series of analytical queries to derive actionable insights. These queries support inventory management, customer behavior analysis, sales performance evaluation, and data quality assurance, enabling informed decision-making for the bookstore's operations.

2. Database Creation

Query:

```
CREATE DATABASE OnlineBookstore;
```

Explanation: This step initializes a database named OnlineBookstore to serve as the central repository for all data related to the bookstore's operations, including books, customers, and their orders.

3. Table Creation

The database comprises three primary tables to organize data efficiently: Books, Customers, and Orders. Each table is designed with appropriate attributes and constraints to ensure data integrity and relational consistency.

3.1 Books Table

Query:

```
CREATE TABLE Books (
   Book_ID SERIAL PRIMARY KEY,
   Title VARCHAR(100),
   Author VARCHAR(100),
   Genre VARCHAR(50),
   Published_Year INT,
   Price NUMERIC(10, 2),
   Stock INT
);
```

Explanation: The Books table stores comprehensive information about the books available in the bookstore, including a unique, automatically generated book identifier, title, author, genre, publication year, price (with two decimal places), and stock quantity, ensuring all necessary details are captured for inventory management.

3.2 Customers Table

Query:

```
CREATE TABLE Customers (
   Customer_ID SERIAL PRIMARY KEY,
   Name VARCHAR(100),
   Email VARCHAR(100),
   Phone VARCHAR(15),
   City VARCHAR(50),
   Country VARCHAR(150)
);
```

Explanation: The Customers table holds details about the bookstore's customers, including a unique, automatically generated customer identifier, name, email address, phone number, city, and country, facilitating customer management and communication.

3.3 Orders Table

Query:

```
CREATE TABLE Orders (
    Order_ID SERIAL PRIMARY KEY,
    Customer_ID INT REFERENCES Customers(Customer_ID),
    Book_ID INT REFERENCES Books(Book_ID),
    Order_Date DATE,
    Quantity INT,
    Total_Amount NUMERIC(10, 2)
);
```

Explanation: The Orders table tracks customer purchases, linking books and customers through foreign keys, and includes a unique, automatically generated order identifier, customer and book identifiers, order date, quantity ordered, and total amount (with two decimal places), ensuring accurate order tracking.

4. Data Import

Sample data is imported into the Books, Customers, and Orders tables from CSV files located at D:\SQL\CSV. The import process ensures accurate mapping of data to the respective table structures, with CSV headers guiding the column assignments.

4.1 Books Data Import

Query:

```
COPY Books(Book_ID, Title, Author, Genre, Published_Year, Price, Stock)
FROM 'D:\SQL\CSV\Books.csv'
CSV HEADER;
```

Explanation: This step populates the Books table with sample data from a CSV file, importing fields such as book identifier, title, author, genre, publication year, price, and stock quantity for inventory analysis.

4.2 Customers Data Import

Query:

```
COPY Customers(Customer_ID, Name, Email, Phone, City, Country)
FROM 'D:\SQL\CSV\Customers.csv'
CSV HEADER;
```

Explanation: This step imports customer data from a CSV file, populating fields such as customer identifier, name, email address, phone number, city, and country, enabling customer-related queries and analysis.

4.3 Orders Data Import

Query

```
COPY Orders(Order_ID, Customer_ID, Book_ID, Order_Date, Quantity, Total_Amount)
FROM 'D:\SQL\CSV\Orders.csv'
CSV HEADER;
```

Explanation: This step imports order data from a CSV file, populating fields like order identifier, customer and book identifiers, order date, quantity, and total amount, facilitating sales and order analysis.

5. Analytical Queries and Their Explanations

The following sections present 30 analytical queries, each written in plain text form exactly as they appear in PostgreSQL syntax, followed by a professional explanation of their purpose and functionality. These queries support various aspects of bookstore operations, including inventory management, customer analysis, and sales performance.

5.1 High-Priced Books

Query:

```
SELECT
   *
FROM
   books
WHERE
   price > 1000;
```

Explanation: This query retrieves all details of books with a price exceeding 1000, identifying premium offerings in the inventory to support targeted marketing strategies for high-value products.

5.2 Customers from Karachi

Query:

```
SELECT
   *
FROM
   customers
WHERE
   city = 'Karachi';
```

Explanation: This query lists all details of customers located in Karachi, aiding regional analysis and informing localized marketing or customer engagement initiatives specific to this city.

5.3 Orders After January 1, 2024

Query:

```
SELECT
   *
FROM
   orders
WHERE
   order_date > '2024-01-01'
ORDER BY
   order_date;
```

Explanation: This query fetches all order details for purchases made after January 1, 2024, sorted by order date, providing insights into recent sales activity and demand trends.

5.4 Out-of-Stock Books

Query:

```
SELECT
   book_id,
   title,
   stock
FROM
   books
WHERE
   stock = 0;
```

Explanation: This query identifies books with zero stock, retrieving their book identifier, title, and stock level, assisting in inventory management by highlighting items needing restocking.

5.5 Number of Books by Genre

Query:

```
SELECT
   genre,
   COUNT(book_id)
FROM
   books
GROUP BY
   genre;
```

Explanation: This query counts the number of books in each genre, grouping the results by genre, offering a clear view of inventory distribution to guide decisions on genre expansion or reduction.

5.6 Total Orders per Customer

Query:

```
SELECT
   customer_id,
   COUNT(order_id)
FROM
```

```
  orders
GROUP BY
  customer_id
ORDER BY
  COUNT(order_id) DESC;
```

Explanation: This query calculates the number of orders placed by each customer, grouped by customer identifier and sorted by order count in descending order, identifying frequent buyers for loyalty programs or personalized promotions.

5.7 Top 5 Most Expensive Books

Query:

```
SELECT
  *
FROM
  books
ORDER BY
  price DESC
LIMIT 5;
```

Explanation: This query retrieves the five most expensive books, including all details, sorted by price in descending order, highlighting premium products for pricing strategy and marketing efforts.

5.8 Customers Who Never Ordered

Query:

```
SELECT
  customer_id,
  name,
  country
FROM
  customers
WHERE
  customer_id NOT IN (
    SELECT customer_id FROM orders
  );
```

Explanation: This query lists the customer identifier, name, and country for customers who have not placed any orders, aiding targeted outreach to encourage purchases.

5.9 Total Sales by Book

Query:

```
SELECT
  book_id,
  SUM(total_amount)
FROM
```

```
    orders
GROUP BY
    book_id;
```

Explanation: This query sums the total revenue generated by each book, grouped by book identifier, revealing top-selling titles to inform inventory and marketing decisions.

5.10 Orders with Customer Name and Book Title

Query

```
SELECT
    o.*,
    c.name,
    b.title
FROM
    orders o
JOIN customers c ON c.customer_id = o.customer_id
JOIN books b ON b.book_id = o.book_id;
```

Explanation: This query combines data from the orders, customers, and books tables to provide detailed order information, including the customer's name and book title, for comprehensive transaction tracking.

5.11 Customers Who Ordered More Than 3 Copies

Query:

```
SELECT
    c.customer_id,
    c.name,
    c.phone,
    c.country,
    o.quantity
FROM
    customers c
JOIN orders o ON c.customer_id = o.customer_id
WHERE
    o.quantity > 3;
```

Explanation: This query identifies customers who ordered more than three copies in a single order, retrieving their customer identifier, name, phone number, country, and order quantity, highlighting bulk purchases for understanding buying patterns.

5.12 Average Price by Genre

Query:

```
SELECT
    genre,
    AVG(price) AS average_price
FROM
```

```
    books
GROUP BY
    genre;
```

Explanation: This query calculates the average price of books in each genre, grouping by genre, to support pricing strategy analysis by revealing price trends across genres.

5.13 Most Recent Order per Customer

Query:

```
SELECT
    customer_id,
    MAX(order_date)
FROM
    orders
GROUP BY
    customer_id;
```

Explanation: This query determines the most recent order date for each customer, grouped by customer identifier, helping track customer activity and recency of purchases.

5.14 Book Order Frequency

Query:

```
SELECT
    b.book_id,
    b.title,
    COUNT(o.order_id) AS times_ordered
FROM
    books b
JOIN orders o ON b.book_id = o.book_id
GROUP BY
    b.book_id, b.title;
```

Explanation: This query counts the number of times each book has been ordered, retrieving the book identifier, title, and order count, indicating popularity for inventory management decisions.

5.15 Customers Who Ordered Science Fiction Books

Query:

```
SELECT
    DISTINCT c.customer_id,
    c.name
FROM
    customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN books b ON b.book_id = o.book_id
WHERE
```

```
    b.genre = 'Science Fiction';
```

Explanation: This query identifies unique customers who purchased Science Fiction books, retrieving their customer identifier and name, supporting genre-specific marketing efforts.

5.16 Books with 'Python' in Title

Query:

```
SELECT
    title
FROM
    books
WHERE
    title ILIKE '%python%';
```

Explanation: This query retrieves titles of books containing the word "Python" (case-insensitive), facilitating specific searches for targeted promotions.

5.17 Total Revenue from Orders

Query:

```
SELECT
    SUM(total_amount) AS total_revenue
FROM
    orders;
```

Explanation: This query calculates the total revenue from all orders, providing an overview of the bookstore's sales performance.

5.18 Books Published Before 2000

Query:

```
SELECT
    *
FROM
    books
WHERE
    published_year < 2000;
```

Explanation: This query retrieves all details of books published before 2000, helping identify older inventory for stock management or targeting collectors.

5.19 Customer Spending Summary

Query:

```
SELECT
    c.customer_id,
    c.name,
    c.email,
    SUM(o.total_amount) AS total_spent
```

```
FROM
   customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY
   c.customer_id, c.name, c.email
ORDER BY
   total_spent DESC;
```

Explanation: This query sums the total amount spent by each customer, retrieving their customer identifier, name, email, and total spent, sorted by total spent in descending order, identifying high-value customers for loyalty programs.

5.20 Books Never Ordered

Query:

```
SELECT
   book_id,
   title
FROM
   books
WHERE
   book_id NOT IN (
      SELECT book_id FROM orders
   );
```

Explanation: This query lists the book identifier and title of books that have not been purchased, helping assess unsold inventory for promotions or stock clearance.

5.21 Create View for Expensive and In-Stock Books

Query:

```
CREATE VIEW expensive_in_stock AS
SELECT
   *
FROM
   books
WHERE
   price > 500
   AND stock > 5;
```

Explanation: This query creates a view that includes all details of books with a price greater than 500 and a stock quantity greater than 5, simplifying frequent queries for high-value, readily available books.

5.22 Simulate a 10% Price Increase

Query:

```
SELECT

   title,
   price,
   price * 1.10 AS updated_price
FROM
   books;
```

Explanation: This query retrieves the title and price of each book, calculating a new price by increasing the current price by 10%, supporting pricing strategy planning by simulating a price increase.

5.23 Delete Orders with Zero Quantity

Query:

```
DELETE FROM orders
WHERE quantity = 0;
```

Explanation: This query removes orders with a quantity of zero, ensuring data cleanliness by eliminating invalid or erroneous order records.

5.24 Total Books Ordered per Customer (Using a Common Table Expression)

Query:

```
WITH customer_order_count AS (
    SELECT
        c.customer_id,
        c.name,
        SUM(o.quantity) AS total_books_ordered
    FROM
        customers c
    JOIN orders o ON c.customer_id = o.customer_id
    GROUP BY
        c.customer_id, c.name
)
SELECT * FROM customer_order_count;
```

Explanation: This query uses a temporary result set to calculate the total quantity of books ordered by each customer, retrieving their customer identifier, name, and total books ordered, providing insights into customer purchasing volume.

5.25 Rank Customers by Spending (Using a Window Function)

Query:

```
SELECT
    customer_id,
    total_spent,
    RANK() OVER (ORDER BY total_spent DESC) AS spending_rank
FROM (
    SELECT
        customer_id,
        SUM(total_amount) AS total_spent
    FROM
        orders
    GROUP BY
        customer_id
) AS spending_data;
```

Explanation: This query ranks customers based on their total spending, retrieving their customer identifier, total spent, and rank, with the highest spender ranked first, highlighting top spenders for targeted marketing.

5.26 Running Total of Revenue

Query:

```
SELECT
    order_id,
    order_date,
    total_amount,
    SUM(total_amount) OVER (ORDER BY order_date) AS running_total
FROM
    orders;
```

Explanation: This query calculates a cumulative total of revenue based on order dates, retrieving the order identifier, order date, total amount, and running total, tracking sales trends over time.

5.27 Most Expensive Book in Each Genre

Query:

```
SELECT
    title,
    genre,
    price
FROM
    books
WHERE (genre, price) IN (
    SELECT
        genre,
        MAX(price)
    FROM
        books
    GROUP BY
        genre
);
```

Explanation: This query identifies the highest-priced book in each genre, retrieving the title, genre, and price, supporting competitive analysis and pricing strategies by genre.

5.28 Duplicate Customer Emails

Query:

```
SELECT
    email,
    COUNT(*) AS occurrences
FROM
    customers
GROUP BY
    email
HAVING COUNT(*) > 1;
```

Explanation: This query identifies email addresses used by more than one customer, retrieving the email and the number of occurrences, helping detect potential data entry errors for correction.

5.29 Temporary Table for High-Value Orders

Query:

```
CREATE TEMP TABLE high_value_orders AS
SELECT
    *
FROM
    orders
WHERE
    total_amount > 5000;
SELECT * FROM high_value_orders;
```

Explanation: This query creates a temporary table containing all details of orders with a total amount exceeding 5000 and retrieves its contents, isolating high-value orders for focused analysis.

5.30 Categorize Books by Price

Query:

```
SELECT
    title,
    price,
    CASE
        WHEN price > 1000 THEN 'Expensive'
        ELSE 'Affordable'
    END AS price_category
FROM
    books;
```

Explanation: This query categorizes books based on their price, labeling those with a price greater than 1000 as Expensive and others as Affordable, retrieving the title, price, and category, aiding in pricing analysis and marketing segmentation.

6. Conclusion

This SQL project demonstrates a robust approach to managing an online bookstore's database. It includes creating a database, defining tables, importing data, and performing a wide range of analytical queries. These queries provide insights into inventory distribution, customer behavior, sales performance, and data quality, enabling the bookstore to optimize operations, enhance customer engagement, and make data-driven decisions.