**Bookstore Sales Analysis Report**

**Author: Akshay Yede  
Date: 5th February 2025**

**1. Problem Statement**

Managing a bookstore requires handling a vast collection of books, tracking customer purchases, and ensuring stock availability at all times. To achieve this, a well-structured database system is essential for storing, managing, and retrieving critical information efficiently. Without a robust database, bookstores may struggle with inventory mismanagement, delayed order processing, and inaccurate sales tracking, ultimately affecting profitability and customer satisfaction.

This project focuses on designing and implementing a **PostgreSQL database** to optimize bookstore operations by efficiently managing book details, customer records, and order transactions. By leveraging structured query language (SQL), the database allows seamless retrieval of key business insights, such as identifying best-selling books, analyzing customer purchasing patterns, and monitoring stock levels. The structured approach ensures that bookstore managers can make data-driven decisions to enhance business efficiency.

Beyond basic record-keeping, the database is designed to **support advanced analytical queries**, providing insights into sales trends, revenue generation, and inventory optimization. The ability to track real-time stock availability and customer preferences helps in strategic planning, such as targeted marketing and stock replenishment. Through this PostgreSQL-based system, bookstores can enhance operational efficiency and improve customer experience by ensuring a smooth and data-driven approach to business management.

**2. Project Overview**

The bookstore database project provides a structured and efficient approach to managing book sales, customer records, and order transactions. By implementing a **PostgreSQL-based system**, the project ensures seamless data handling, allowing bookstore owners to streamline operations and maintain accurate records. A well-designed database helps prevent data redundancy, enhances retrieval speed, and supports complex queries for better business insights.

This database is designed to offer multiple functionalities that enhance bookstore management. It allows for **efficient storage and retrieval** of book details, ensuring that information such as title, author, genre, price, and stock availability is easily accessible. Additionally, it enables **tracking customer purchases and spending behavior**, helping businesses understand customer preferences and improve their marketing strategies. Furthermore, the system dynamically **updates stock levels** after each order is fulfilled, reducing the risk of overstocking or running out of popular books.

Beyond operational efficiency, the database also plays a crucial role in **business analytics**. By generating detailed reports on **revenue, sales trends, and book availability**, bookstore owners can make informed decisions about inventory management, pricing strategies, and promotional campaigns. This project not only optimizes day-to-day operations but also provides valuable insights that contribute to long-term business growth.

**3. Database Schema**

**Tables:**

1. **Books**
   * Book\_ID (Primary Key)
   * Title
   * Author
   * Genre
   * Published\_Year
   * Price
   * Stock
2. **Customers**
   * Customer\_ID (Primary Key)
   * Name
   * Email
   * Phone
   * City
   * Country
3. **Orders**
   * Order\_ID (Primary Key)
   * Customer\_ID (Foreign Key -> Customers)
   * Book\_ID (Foreign Key -> Books)
   * Order\_Date
   * Quantity
   * Total\_Amount

**4. Queries & Results**

**1) Retrieve all books in the "Fiction" genre**

SELECT \* FROM books

WHERE Genre = 'Fiction';

Result -->

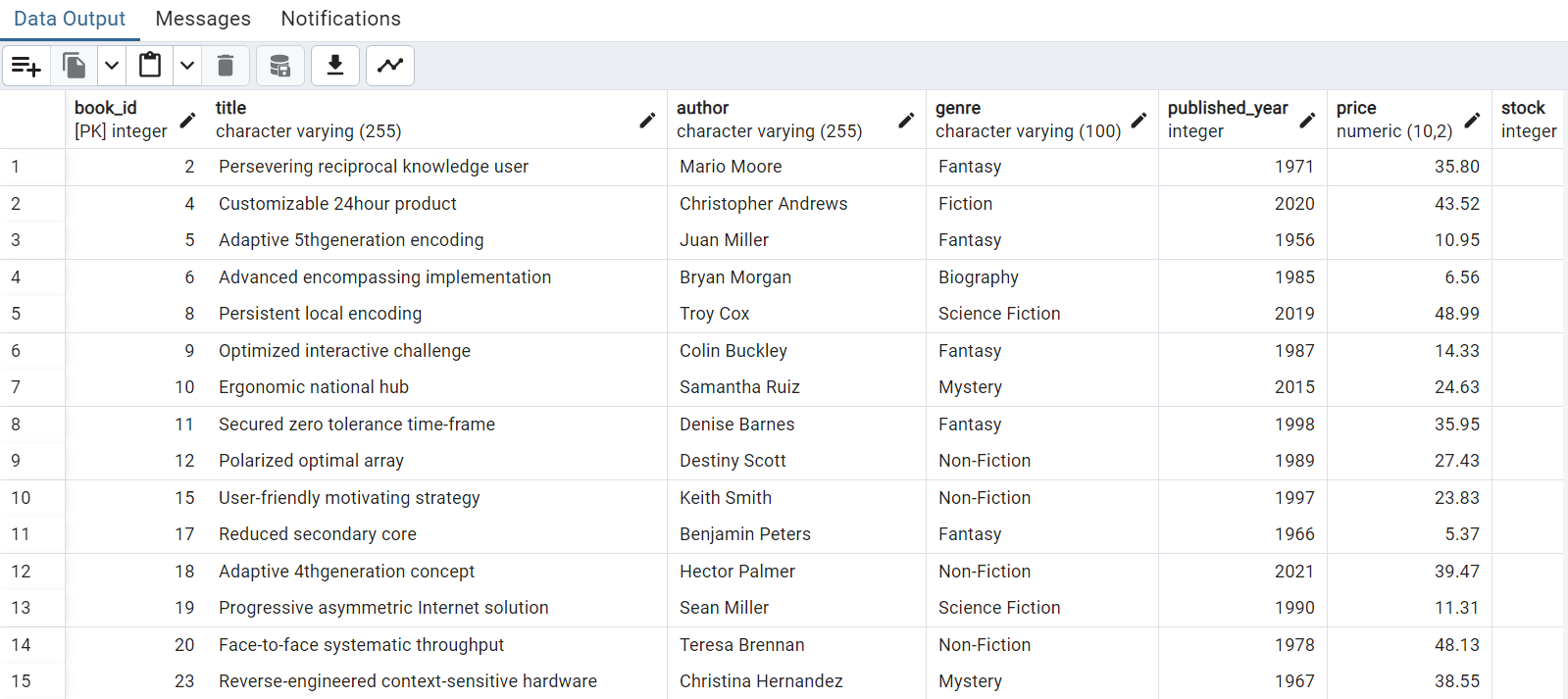


**2) Find books published after the year 1950**

SELECT \* FROM books

WHERE Published\_Year > 1950;

Result -->

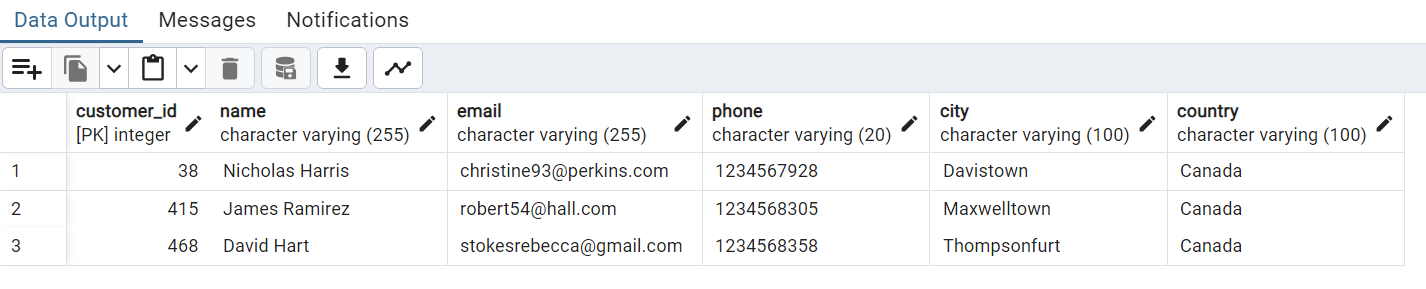


**3) List all customers from Canada**

SELECT \* FROM customers

WHERE Country = 'Canada';

Result -->

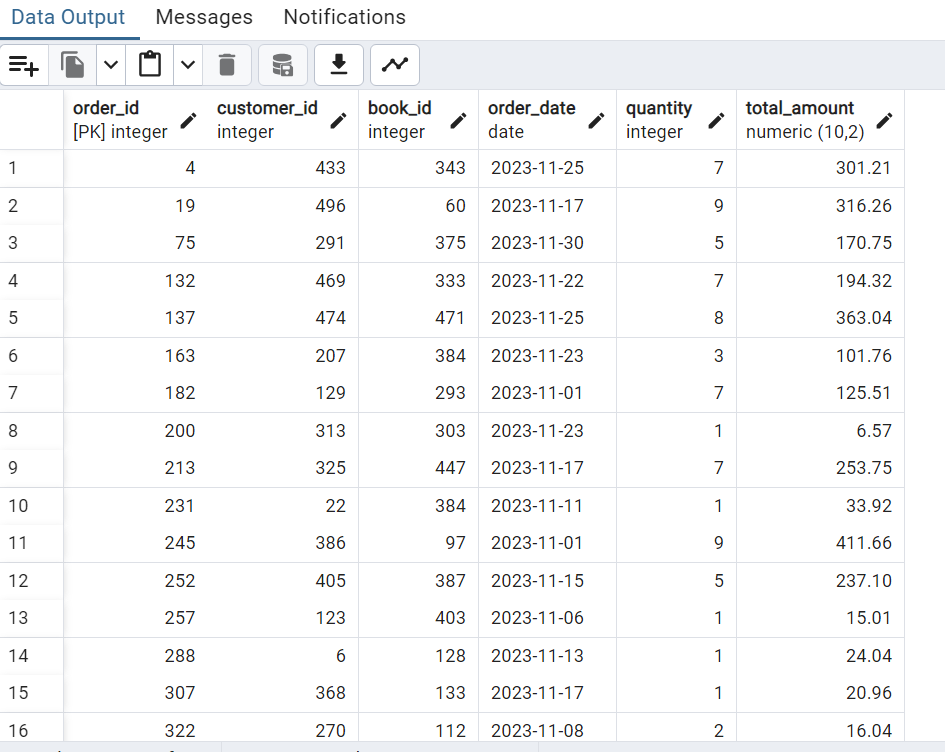


**4) Show orders placed in November 2023**

SELECT \* FROM orders

WHERE Order\_Date BETWEEN '2023-11-01' AND '2023-11-30';

Result -->

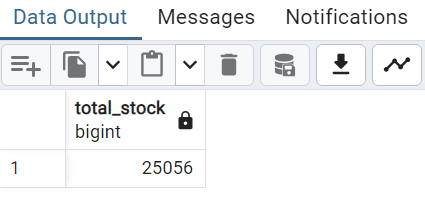


**5) Retrieve the total stock of books available**

SELECT SUM(Stock) AS Total\_Stock

FROM books;

Result -->



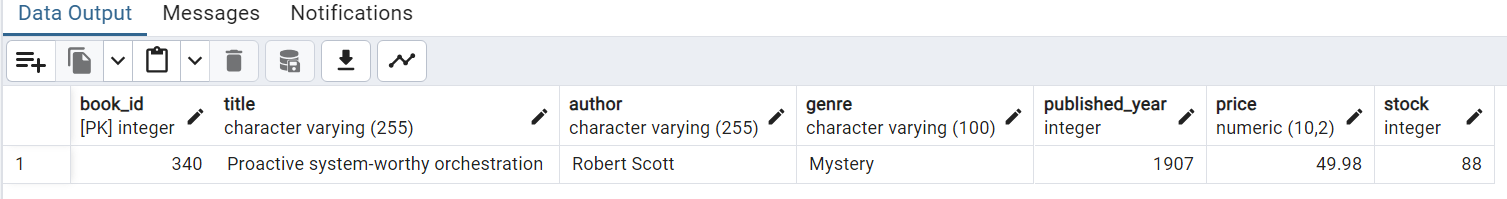
**6) Find the details of the most expensive book**

SELECT \* FROM books

ORDER BY Price DESC

LIMIT 1;

Result -->



**7) Show all customers who ordered more than 1 quantity of a book**

SELECT DISTINCT c.Customer\_ID, c.Name

FROM customers c

JOIN orders o ON c.Customer\_ID = o.Customer\_ID

WHERE o.Quantity > 1;

Result -->



**8) Retrieve all orders where the total amount exceeds $20**

SELECT o.Order\_ID, o.Customer\_ID, SUM(b.Price \* o.Quantity) AS Total\_Amount

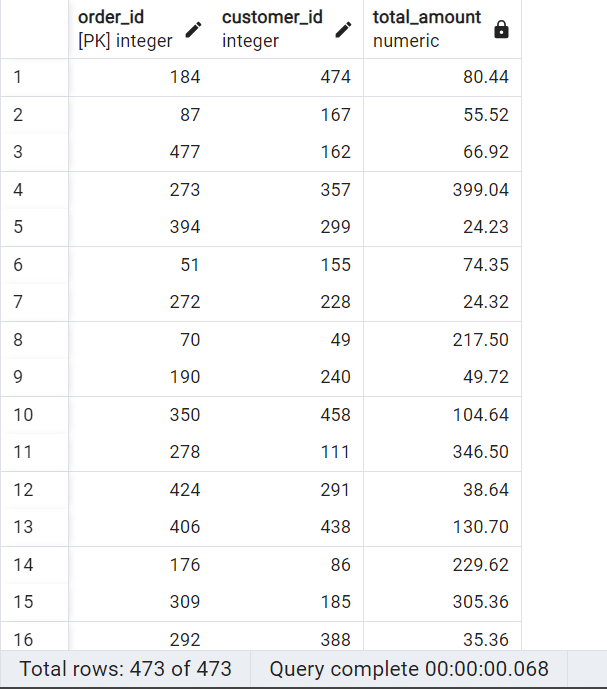
FROM orders o

JOIN books b ON o.Book\_ID = b.Book\_ID

GROUP BY o.Order\_ID, o.Customer\_ID

HAVING SUM(b.Price \* o.Quantity) > 20;

Result -->

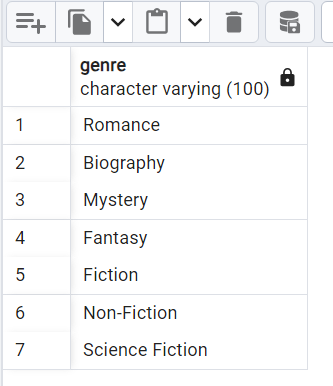


**9) List all genres available in the Books table**

SELECT DISTINCT Genre

FROM books;

Result -->



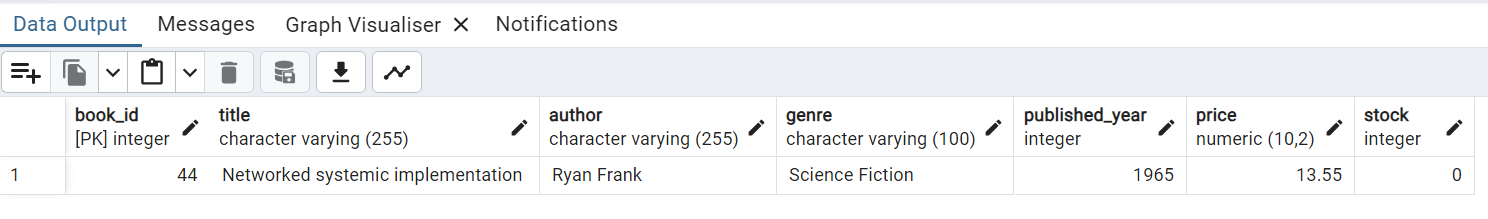
**10) Find the book with the lowest stock**

SELECT \* FROM books

ORDER BY Stock ASC

LIMIT 1;

Result -->



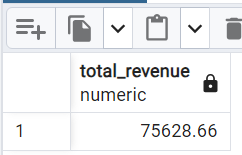
**11) Calculate the total revenue generated from all orders**

SELECT SUM(b.Price \* o.Quantity) AS Total\_Revenue

FROM orders o

JOIN books b ON o.Book\_ID = b.Book\_ID;

Result -->



**12) Total Number of Books Sold for Each Genre**

SELECT b.Genre, SUM(o.Quantity) AS Total\_Books\_Sold

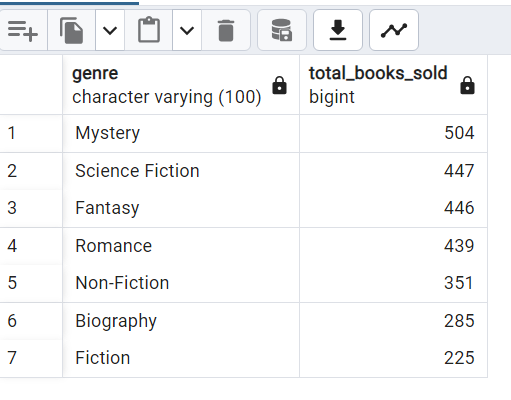
FROM orders o

JOIN books b ON o.Book\_ID = b.Book\_ID

GROUP BY b.Genre

ORDER BY Total\_Books\_Sold DESC;

Result -->



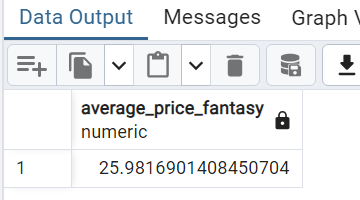
**13) Calculate the Average Price of Books in the Fantasy Genre**

SELECT AVG(Price) AS Average\_Price\_Fantasy

FROM books

WHERE Genre = 'Fantasy';

Result -->



**14) Retrieve Customers Who Have Placed at Least Two Orders**

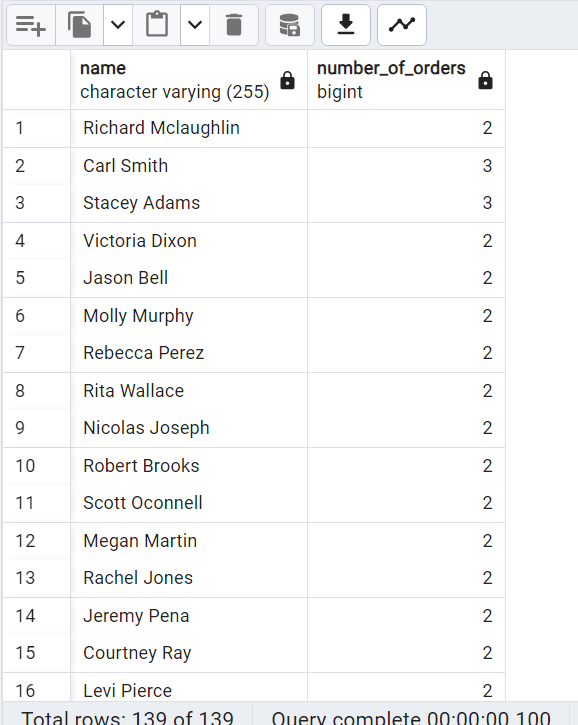
SELECT c.Name, COUNT(o.Order\_ID) AS Number\_of\_Orders

FROM customers c

JOIN orders o ON c.Customer\_ID = o.Customer\_ID

GROUP BY c.Customer\_ID, c.Name

HAVING COUNT(o.Order\_ID) >= 2;



**15) Identify the Most Frequently Ordered Book**

SELECT b.Title, COUNT(o.Order\_ID) AS Order\_Count

FROM orders o

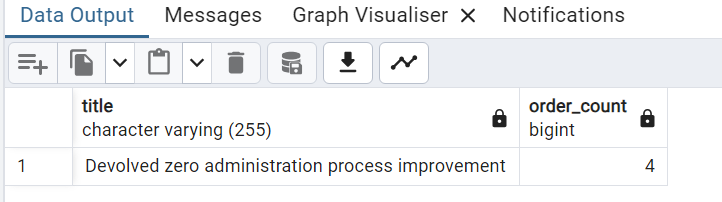
JOIN books b ON o.Book\_ID = b.Book\_ID

GROUP BY b.Title

ORDER BY Order\_Count DESC

LIMIT 1;

Result -->



**16) Fetch the Top 3 Most Expensive Books in the Fantasy Genre**

SELECT Title, Price

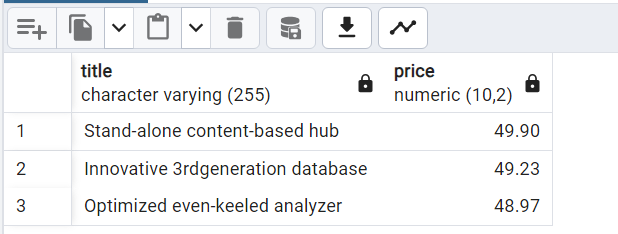
FROM books

WHERE Genre = 'Fantasy'

ORDER BY Price DESC

LIMIT 3;

Result -->



**17) Calculate the Total Quantity of Books Sold by Each Author**

SELECT b.Author, SUM(o.Quantity) AS Total\_Quantity\_Sold

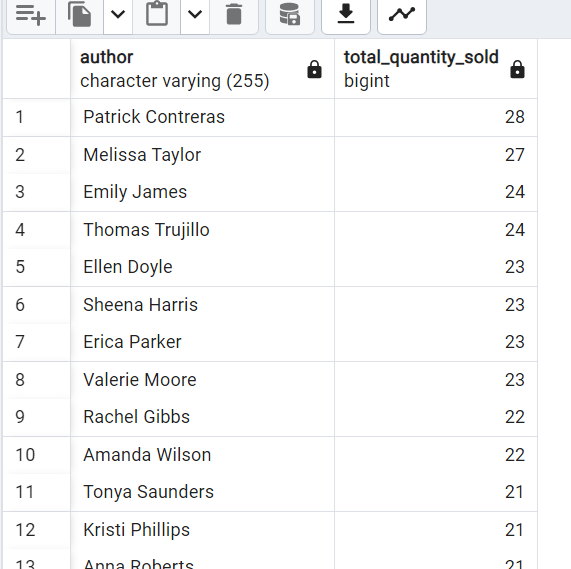
FROM orders o

JOIN books b ON o.Book\_ID = b.Book\_ID

GROUP BY b.Author

ORDER BY Total\_Quantity\_Sold DESC;

Result -->



**18) Find the Customer Who Spent the Most on Orders**

SELECT c.Name, SUM(b.Price \* o.Quantity) AS Total\_Spent

FROM customers c

JOIN orders o ON c.Customer\_ID = o.Customer\_ID

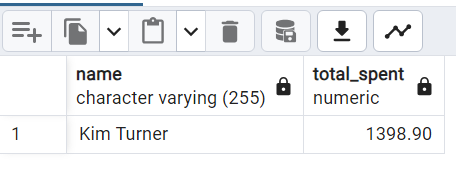
JOIN books b ON o.Book\_ID = b.Book\_ID

GROUP BY c.Name

ORDER BY Total\_Spent DESC

LIMIT 1;

Result -->



**19) Compute the Stock Remaining After Fulfilling Orders**

SELECT b.Book\_ID, b.Title, (b.Stock - COALESCE(SUM(o.Quantity), 0)) AS Remaining\_Stock

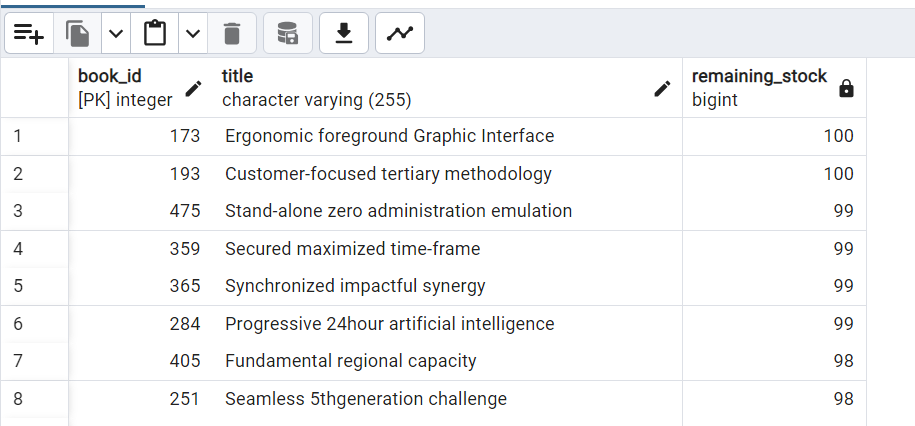
FROM books b

LEFT JOIN orders o ON b.Book\_ID = o.Book\_ID

GROUP BY b.Book\_ID, b.Title

ORDER BY Remaining\_Stock DESC;

Result -->



**20) List Cities Where Customers Who Spent Over $30 Are Local**

SELECT c.City

FROM customers c

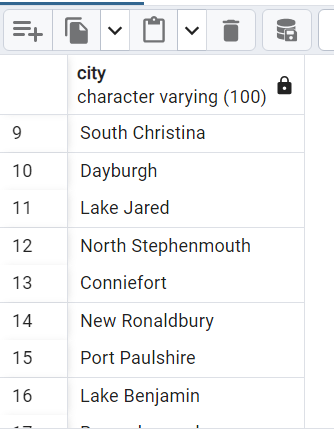
JOIN orders o ON c.Customer\_ID = o.Customer\_ID

JOIN books b ON o.Book\_ID = b.Book\_ID

GROUP BY c.City

HAVING SUM(b.Price \* o.Quantity) > 30;

Result -->



**21) Retrieve the top 3 best-selling books along with their total sales amount**

SELECT b.Title, SUM(o.Quantity) AS Total\_Sold, SUM(o.Quantity \* b.Price) AS Total\_Sales

FROM orders o

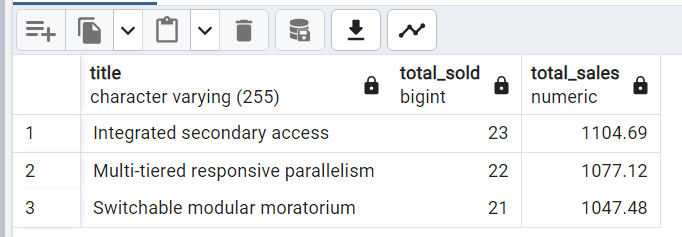
JOIN books b ON o.Book\_ID = b.Book\_ID

GROUP BY b.Title

ORDER BY Total\_Sales DESC

LIMIT 3;

Result -->



**22) Find customers who have spent more than the average order value**

SELECT c.Customer\_ID, c.Name, SUM(o.Quantity \* b.Price) AS Total\_Spent

FROM customers c

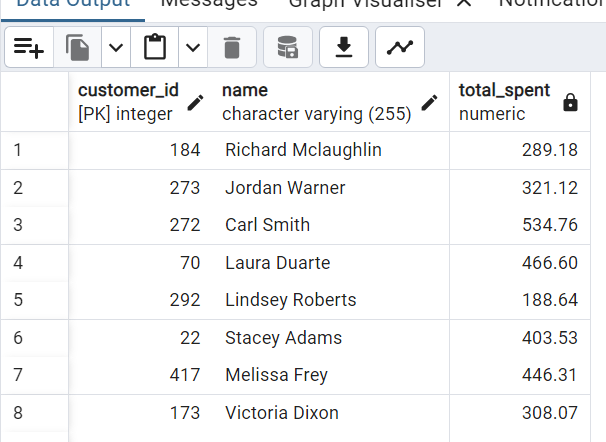
JOIN orders o ON c.Customer\_ID = o.Customer\_ID

JOIN books b ON o.Book\_ID = b.Book\_ID

GROUP BY c.Customer\_ID, c.Name

HAVING SUM(o.Quantity \* b.Price) > (SELECT AVG(o.Quantity \* b.Price) FROM orders o JOIN books b ON o.Book\_ID = b.Book\_ID);

Result -->

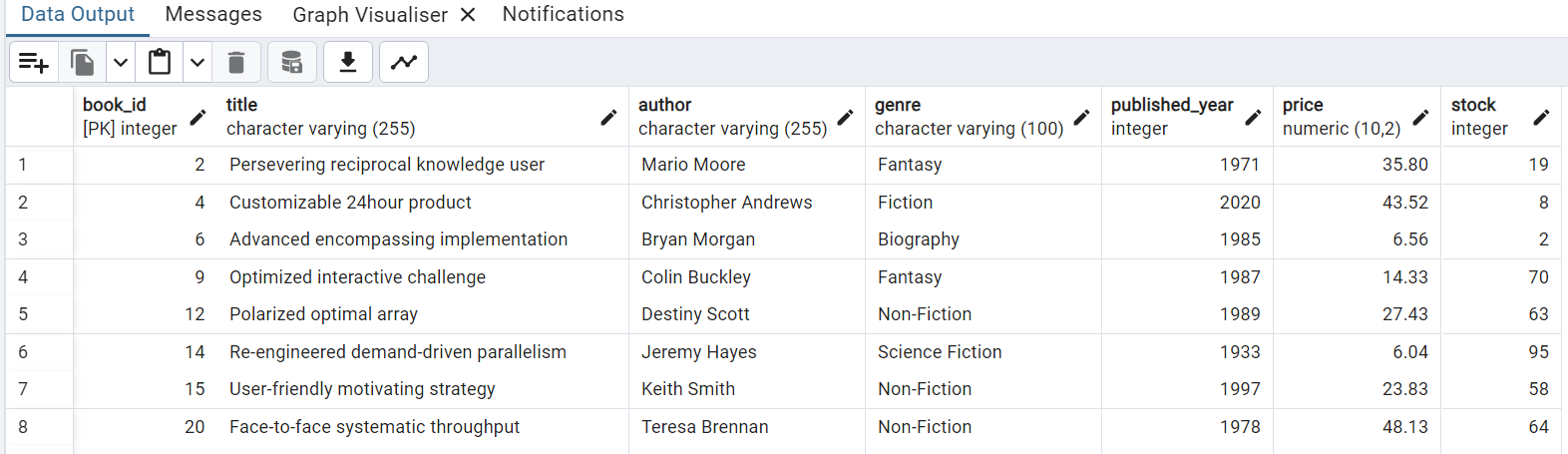


**23) Retrieve the books that have never been ordered**

SELECT \* FROM books

WHERE Book\_ID NOT IN (SELECT DISTINCT Book\_ID FROM orders);

Result -->



**24) Find the genre that generated the highest revenue**

SELECT b.Genre, SUM(o.Quantity \* b.Price) AS Total\_Revenue

FROM books b

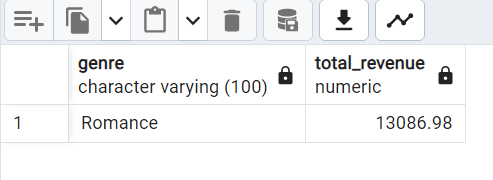
JOIN orders o ON b.Book\_ID = o.Book\_ID

GROUP BY b.Genre

ORDER BY Total\_Revenue DESC

LIMIT 1;

Result -->



**25) Retrieve the customer who placed the earliest order**

SELECT c.Customer\_ID, c.Name, MIN(o.Order\_Date) AS First\_Order\_Date

FROM customers c

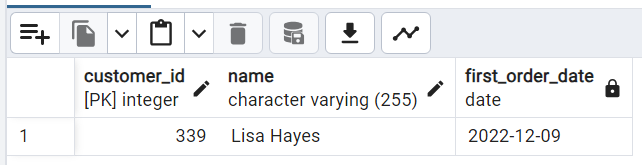
JOIN orders o ON c.Customer\_ID = o.Customer\_ID

GROUP BY c.Customer\_ID, c.Name

ORDER BY First\_Order\_Date ASC

LIMIT 1;

Result -->



**26) Show the cumulative revenue for each month in 2023 using a window function**

SELECT DATE\_TRUNC('month', o.Order\_Date) AS Order\_Month,

SUM(b.Price \* o.Quantity) AS Monthly\_Revenue,

SUM(SUM(b.Price \* o.Quantity)) OVER (ORDER BY DATE\_TRUNC('month', o.Order\_Date)) AS Cumulative\_Revenue

FROM orders o

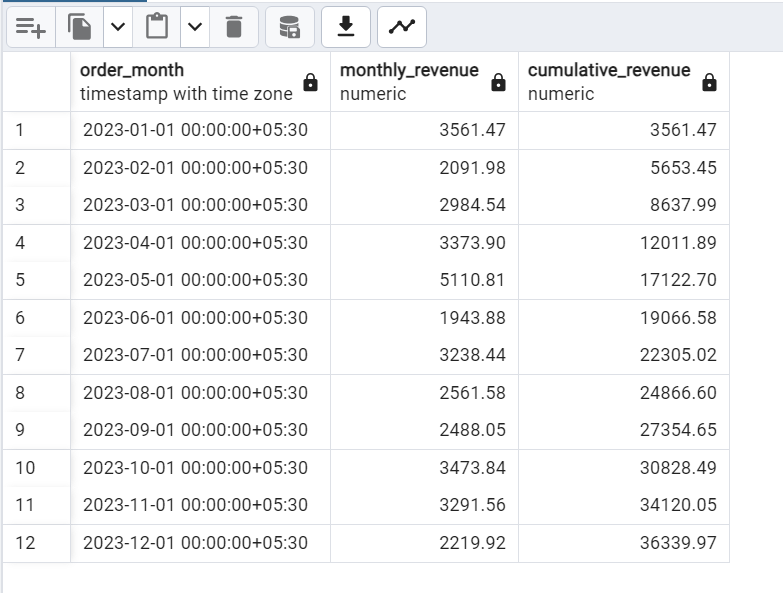
JOIN books b ON o.Book\_ID = b.Book\_ID

WHERE o.Order\_Date BETWEEN '2023-01-01' AND '2023-12-31'

GROUP BY Order\_Month

ORDER BY Order\_Month;

Result -->



**27) Retrieve the latest order for each customer**

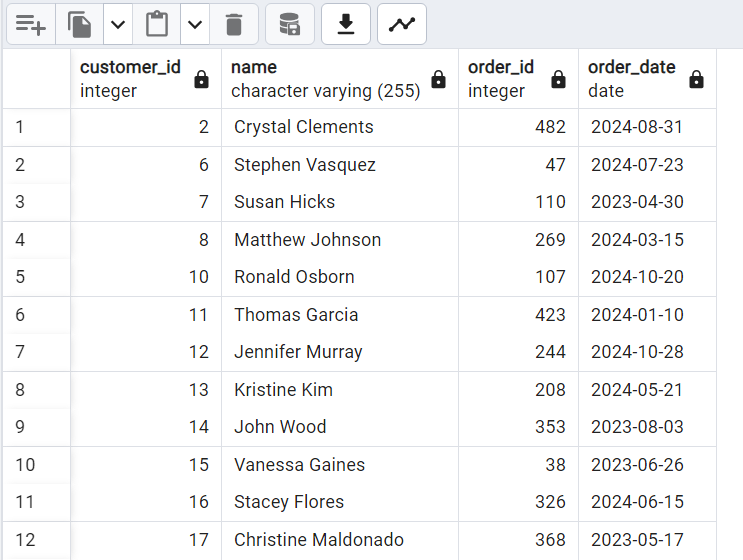
SELECT DISTINCT ON (o.Customer\_ID) o.Customer\_ID, c.Name, o.Order\_ID, o.Order\_Date

FROM orders o

JOIN customers c ON o.Customer\_ID = c.Customer\_ID

ORDER BY o.Customer\_ID, o.Order\_Date DESC;

Result -->



**28) Find customers who have only placed one order**

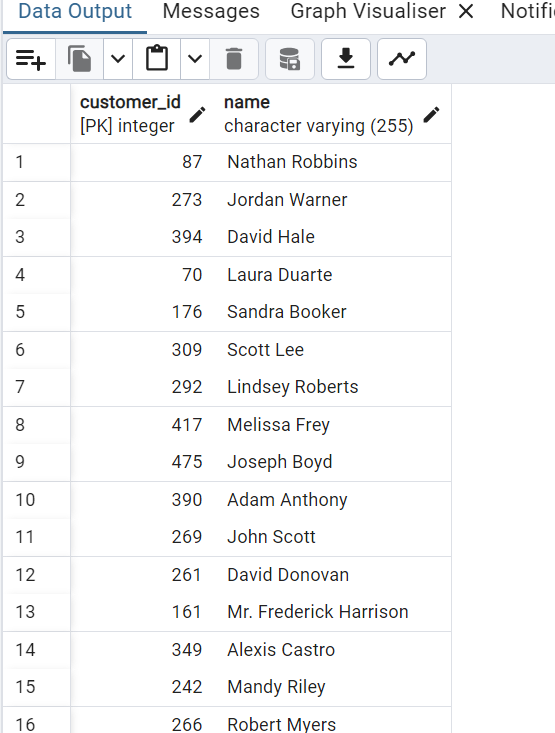
SELECT c.Customer\_ID, c.Name

FROM customers c

JOIN orders o ON c.Customer\_ID = o.Customer\_ID

GROUP BY c.Customer\_ID, c.Name

HAVING COUNT(o.Order\_ID) = 1;



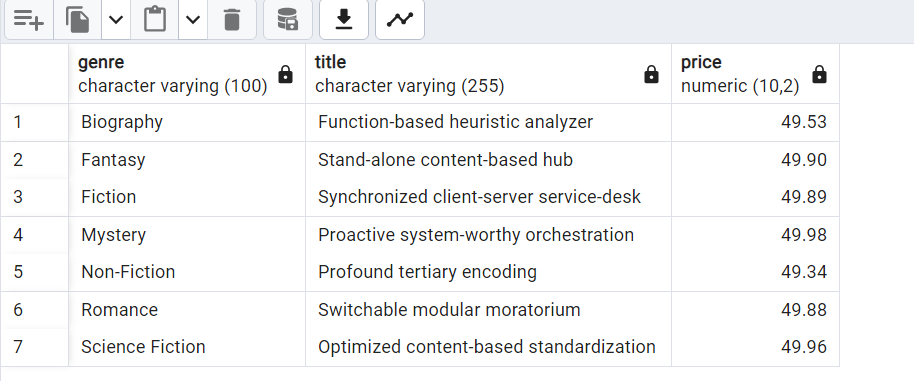
**29) Retrieve the most expensive book in each genre**

SELECT DISTINCT ON (b.Genre) b.Genre, b.Title, b.Price

FROM books b

ORDER BY b.Genre, b.Price DESC;

Result -->



**30) Find the percentage of total sales each genre contributes**

SELECT b.Genre,

SUM(o.Quantity \* b.Price) AS Genre\_Revenue,

(SUM(o.Quantity \* b.Price) \* 100) / (SELECT SUM(o.Quantity \* b.Price) FROM orders o JOIN books b ON o.Book\_ID = b.Book\_ID) AS Revenue\_Percentage

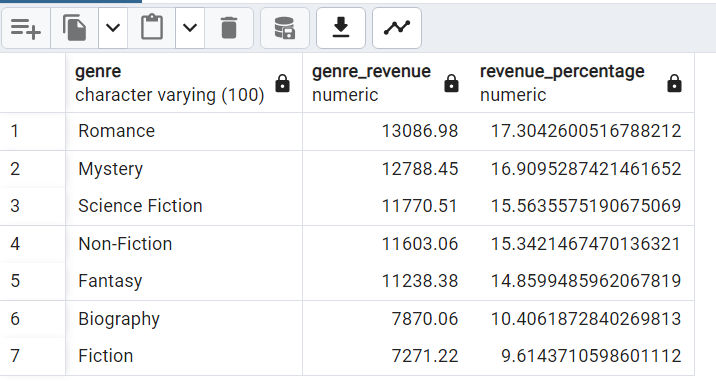
FROM books b

JOIN orders o ON b.Book\_ID = o.Book\_ID

GROUP BY b.Genre

ORDER BY Revenue\_Percentage DESC;

Result -->



**31) Identify books that have been ordered in every month of 2023**

SELECT b.Title

FROM books b

JOIN orders o ON b.Book\_ID = o.Book\_ID

WHERE EXTRACT(YEAR FROM o.Order\_Date) = 2023

GROUP BY b.Title

HAVING COUNT(DISTINCT EXTRACT(MONTH FROM o.Order\_Date)) = 12;

Result -->

There are no books that have been ordered in every month of 2023.

**32) Find the customer who placed the most expensive single order**

SELECT o.Customer\_ID, c.Name, o.Order\_ID, SUM(o.Quantity \* b.Price) AS Order\_Total

FROM orders o

JOIN customers c ON o.Customer\_ID = c.Customer\_ID

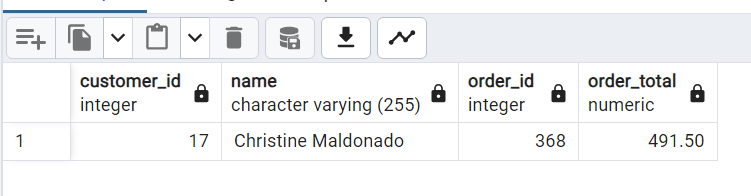
JOIN books b ON o.Book\_ID = b.Book\_ID

GROUP BY o.Customer\_ID, c.Name, o.Order\_ID

ORDER BY Order\_Total DESC

LIMIT 1;

Result -->



**33) Retrieve books that have been ordered more than the average order quantity**

SELECT b.Title, SUM(o.Quantity) AS Total\_Ordered

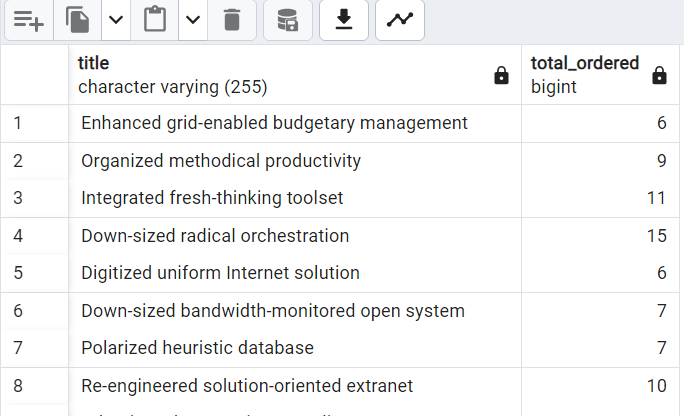
FROM books b

JOIN orders o ON b.Book\_ID = o.Book\_ID

GROUP BY b.Title

HAVING SUM(o.Quantity) > (SELECT AVG(Quantity) FROM orders);

Result -->



**34) Identify customers who have purchased books from at least 3 different genres**

SELECT c.Customer\_ID, c.Name, COUNT(DISTINCT b.Genre) AS Unique\_Genres

FROM customers c

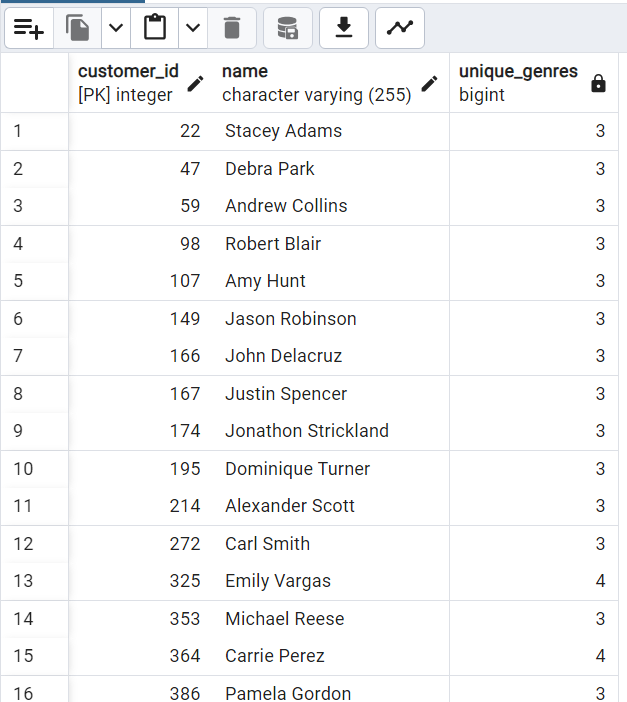
JOIN orders o ON c.Customer\_ID = o.Customer\_ID

JOIN books b ON o.Book\_ID = b.Book\_ID

GROUP BY c.Customer\_ID, c.Name

HAVING COUNT(DISTINCT b.Genre) >= 3;

Result -->



**35) Find books that have at least 5 orders but still have stock remaining**

SELECT b.Book\_ID, b.Title, SUM(o.Quantity) AS Total\_Sold, b.Stock

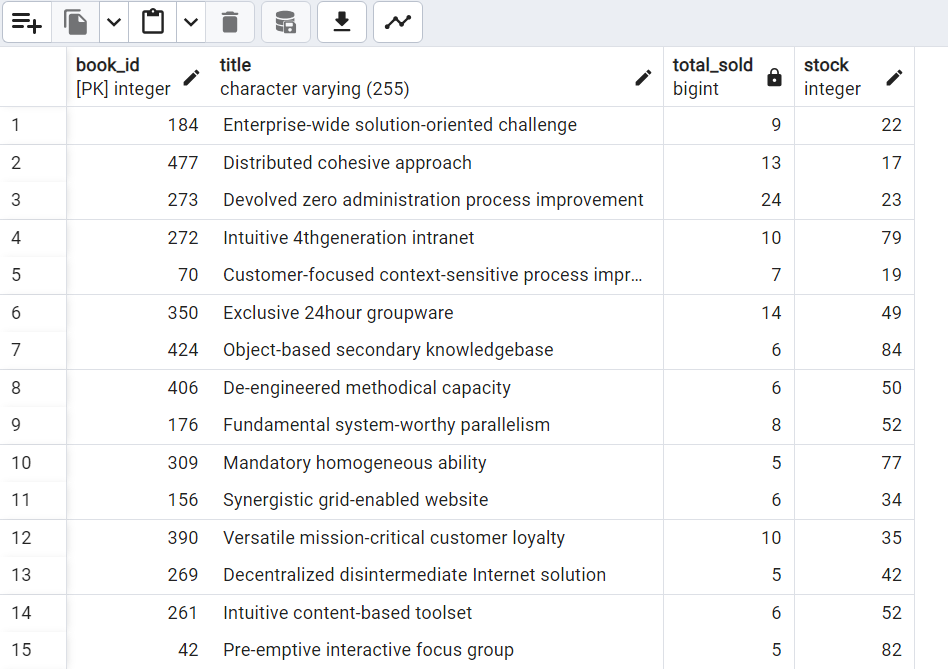
FROM books b

JOIN orders o ON b.Book\_ID = o.Book\_ID

GROUP BY b.Book\_ID, b.Title, b.Stock

HAVING SUM(o.Quantity) >= 5 AND b.Stock > 0;

Result -->



**36) Retrieve books where the total sales revenue is more than the average revenue of all books**

SELECT b.Title, SUM(o.Quantity \* b.Price) AS Total\_Revenue

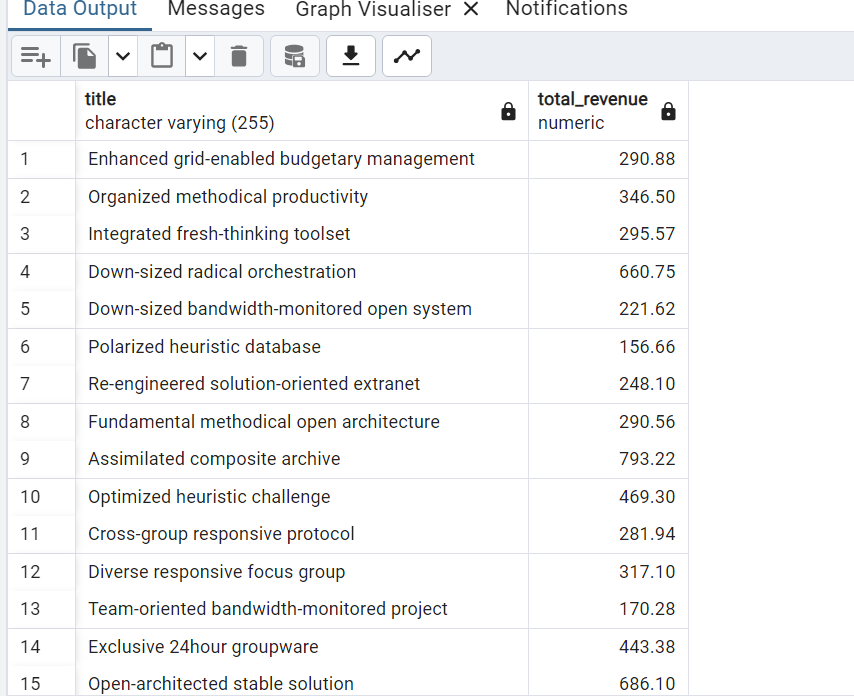
FROM books b

JOIN orders o ON b.Book\_ID = o.Book\_ID

GROUP BY b.Title

HAVING SUM(o.Quantity \* b.Price) > (SELECT AVG(o.Quantity \* b.Price) FROM orders o JOIN books b ON o.Book\_ID = b.Book\_ID);

Result -->



**37) Show the most recent 5 orders with customer details**

SELECT o.Order\_ID, c.Name, o.Order\_Date, SUM(b.Price \* o.Quantity) AS Total\_Cost

FROM orders o

JOIN customers c ON o.Customer\_ID = c.Customer\_ID

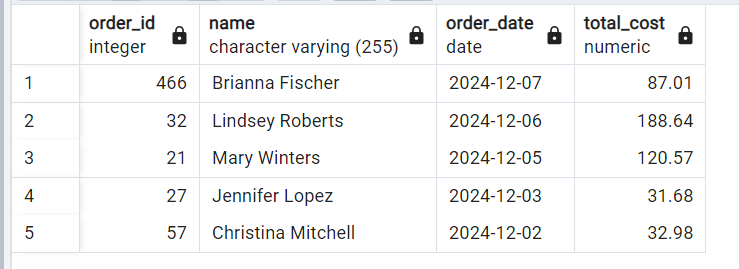
JOIN books b ON o.Book\_ID = b.Book\_ID

GROUP BY o.Order\_ID, c.Name, o.Order\_Date

ORDER BY o.Order\_Date DESC

LIMIT 5;

Result -->



**38) Retrieve customers who ordered the same book more than once on different dates**

SELECT o.Customer\_ID, c.Name, o.Book\_ID, b.Title, COUNT(DISTINCT o.Order\_Date) AS Order\_Count

FROM orders o

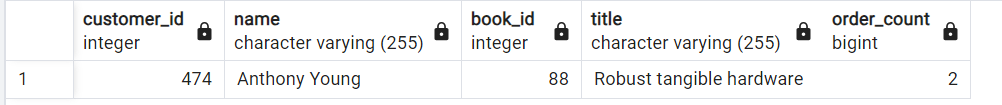
JOIN customers c ON o.Customer\_ID = c.Customer\_ID

JOIN books b ON o.Book\_ID = b.Book\_ID

GROUP BY o.Customer\_ID, c.Name, o.Book\_ID, b.Title

HAVING COUNT(DISTINCT o.Order\_Date) > 1;

Result -->



**39) Show the bestselling book in each genre**

WITH RankedBooks AS (

SELECT b.Genre, b.Title, SUM(o.Quantity) AS Total\_Sold,

RANK() OVER (PARTITION BY b.Genre ORDER BY SUM(o.Quantity) DESC) AS rnk

FROM books b

JOIN orders o ON b.Book\_ID = o.Book\_ID

GROUP BY b.Genre, b.Title

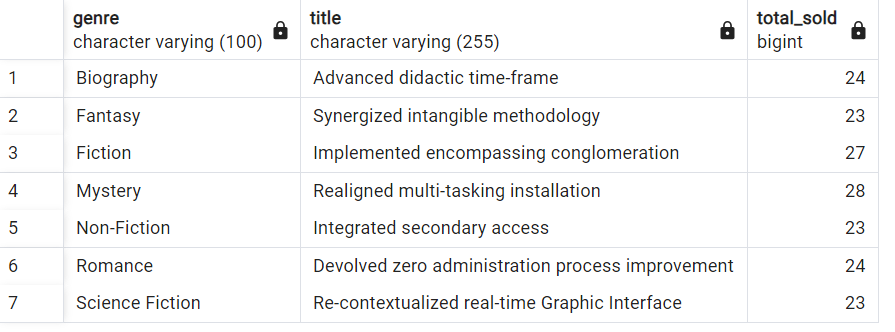
)

SELECT Genre, Title, Total\_Sold

FROM RankedBooks

WHERE rnk = 1;

Result -->



**40) Retrieve books that have at least one order but have not been ordered in the last 6 months**

SELECT b.Book\_ID, b.Title, MAX(o.Order\_Date) AS Last\_Ordered\_Date

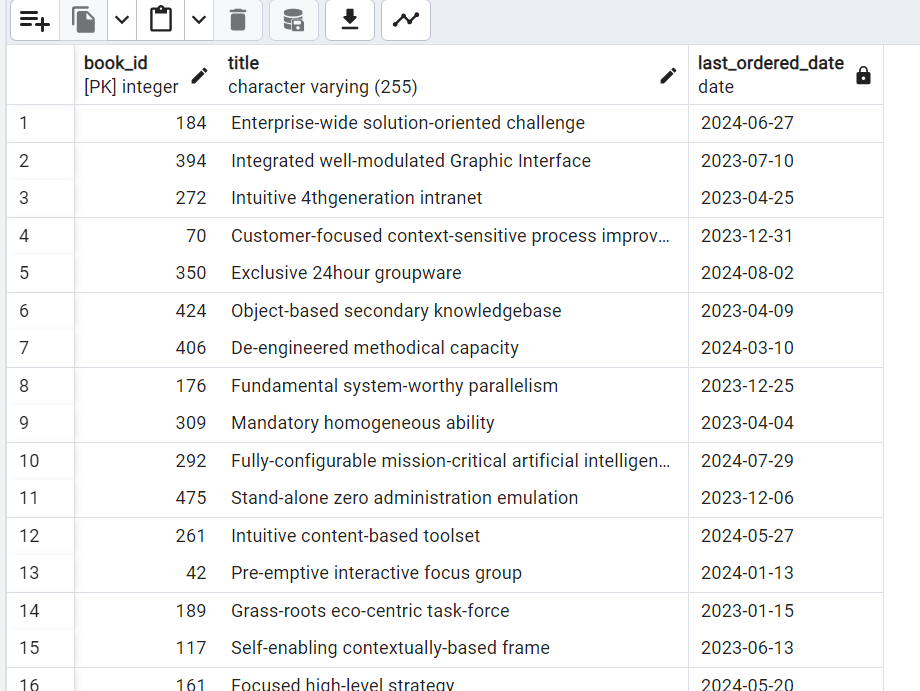
FROM books b

JOIN orders o ON b.Book\_ID = o.Book\_ID

GROUP BY b.Book\_ID, b.Title

HAVING MAX(o.Order\_Date) < CURRENT\_DATE - INTERVAL '6 months';

Result -->



**5. Project Implementation Steps**

1. **Database Setup:** Install PostgreSQL and create the required tables using the provided schema. Ensure proper data types, constraints, and relationships (primary and foreign keys).
2. **Data Insertion via CSV Files:**
   * Prepare **clean and structured CSV files** for tables like books, customers, and orders.
   * Use the COPY command or pgAdmin to efficiently load data into PostgreSQL.
3. **Query Execution & Analysis:**
   * Run SQL queries to extract business insights.
   * Optimize queries using indexing where needed for faster execution.
4. **Result Validation & Debugging:**
   * Cross-check results with expected outputs.
   * Use aggregate functions, joins, and subqueries to ensure data accuracy.
5. **Documentation & Submission:**
   * Prepare a detailed **project report** with problem statements, database design, queries, and their outputs.
   * Upload SQL schema, queries (queries.sql), CSV files, and documentation (report.docx) to **GitHub** for version control and sharing.

**6. Conclusion**

This project successfully implements a structured PostgreSQL database that enhances the management of a bookstore. The database is designed with a clear and logical schema that tracks essential data points, including books, authors, genres, sales, and inventory levels. By organizing the data into well-defined tables, the database ensures seamless retrieval of relevant information, enabling efficient data management across different departments of the bookstore.

Using advanced SQL queries, the system provides actionable business insights that can optimize day-to-day operations. Reports can be generated to track sales trends, identify popular genres or authors, and forecast future demand. This data-driven approach helps in making informed decisions to improve inventory management and tailor marketing efforts, ultimately driving growth in sales and customer satisfaction.

Moreover, the database is flexible and scalable, allowing the bookstore to expand its operations with ease. As new book titles, authors, or genres are added, the system can accommodate them without disrupting existing workflows. The implementation also ensures that all data is stored securely, with features like backup and restore functionalities, further enhancing operational efficiency and business continuity.