# ONLINE BOOK STORE ANALYSIS

By: Rajeev Kumar

Rajeev Kumar

# INTRODUCTION

This presentation showcases an analysis of an Online Book Store's database using MySQL. The aim is to extract meaningful insights regarding the store's inventory, customer behavior, and sales performance.

**The key steps in the analysis involve:**

- Exploring the database structure
- Writing complex SQL queries to extract relevant data
- Presenting findings and insights that could inform business decisions.

# PROJECT SETUP

```sql
CREATE DATABASE Online_Book_Store;
USE Online_Book_Store;
```

```sql
CREATE TABLE Books (
    Book_ID INT AUTO_INCREMENT PRIMARY KEY,
    Title VARCHAR(100),
    Author VARCHAR(100),
    Genre VARCHAR(100),
    Published_Year INT,
    Price DECIMAL(10,2),
    Stock INT
);
```

```sql
CREATE TABLE Orders (
    Order_ID INT AUTO_INCREMENT PRIMARY KEY,
    Customer_ID INT,
    Book_ID INT,
    Order_Date DATE,
    Quantity INT,
    Total_Amount DECIMAL(10,2),
    FOREIGN KEY (Customer_ID) REFERENCES Customers(Customer_ID) ON DELETE CASCADE,
    FOREIGN KEY (Book_ID) REFERENCES Books(Book_ID) ON DELETE CASCADE
);
```

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

# 1. RETRIEVE ALL BOOKS IN THE "FICTION" GENRE

**SOLUTION**

```
select * from books
where genre = 'Fiction';
```

**Explanation**

- The books table stores information about all available books.
- The WHERE clause filters books where the genre column is equal to 'Fiction'.
- This helps in identifying all Fiction books available in the store.

**RESULT**

| Book_ID | Title | Author | Genre | Published_Year | Price | Stock |
|---|---|---|---|---|---|---|
| 4 | Customizable 24hour product | Christopher Andrews | Fiction | 2020 | 43.52 | 8 |
| 22 | Multi-layered optimizing migration | Wesley Escobar | Fiction | 1908 | 39.23 | 78 |
| 28 | Expanded analyzing portal | Lisa Coffey | Fiction | 1941 | 37.51 | 79 |
| 29 | Quality-focused multi-tasking challenge | Katrina Underwood | Fiction | 1905 | 31.12 | 100 |
| 31 | Implemented encompassing conglomeration | Melissa Taylor | Fiction | 2010 | 21.23 | 44 |

# 2. FIND BOOKS PUBLISHED AFTER THE YEAR 1950

## SOLUTION

```sql
SELECT
    *
FROM
    books
WHERE
    Published_Year > 1950
ORDER BY Published_Year;
```

## Explanation

- The WHERE clause filters books where the Published_Year is greater than 1950.
- The ORDER BY Published_Year ensures the results are sorted in ascending order by publication year.
- This helps in analyzing modern publications in the store.

## RESULT

| Book_ID | Title | Author | Genre | Published_Year | Price | Stock |
|---|---|---|---|---|---|---|
| 166 | Customizable discrete Graphical User Interface | Rebecca Alexander | Romance | 1951 | 11.02 | 56 |
| 174 | Pre-emptive executive knowledge user | Rebecca Mann | Mystery | 1951 | 37.83 | 18 |
| 432 | Horizontal disintermediate alliance | Rodney Ward | Non-Fiction | 1951 | 8.84 | 55 |
| 43 | Function-based zero-defect initiative | Daniel Nunez | Romance | 1952 | 47.39 | 61 |

# 3.LIST ALL CUSTOMERS FROM THE CANADA

**SOLUTION**

```
select * from customers
where Country = 'Canada';
```

**Explanation**

- The customers table contains details of all registered customers.
- The WHERE clause filters customers whose Country column is set to 'Canada'.
- This helps in identifying and analyzing customers from Canada for targeted marketing or regional insights.

**RESULT**

| Customer_ID | Name | Email | Phone | City | Country |
|---|---|---|---|---|---|
| 38 | Nicholas Harris | christine93@perkins.com | 1234567928 | Davistown | Canada |
| 415 | James Ramirez | robert54@hall.com | 1234568305 | Maxwelltown | Canada |
| 468 | David Hart | stokesrebecca@gmail.com | 1234568358 | Thompsonfurt | Canada |

# 4. SHOW ORDERS PLACED IN NOVEMBER 2023

## SOLUTION

```sql
SELECT
    *
FROM
    orders
WHERE
    MONTHNAME(order_date) = 'November'
        AND YEAR(order_date) = 2023
ORDER BY order_date;
```

## Explanation

- The MONTHNAME(order_date) = 'November' filters orders placed in November.
- The YEAR(order_date) = 2023 ensures only orders from the year 2023 are selected.
- The ORDER BY order_date sorts the results in chronological order.
- This helps in analyzing sales trends for a specific month.

## RESULT

| Order_ID | Customer_ID | Book_ID | Order_Date | Quantity | Total_Amount |
|---|---|---|---|---|---|
| 182 | 129 | 293 | 2023-11-01 | 7 | 125.51 |
| 245 | 386 | 97 | 2023-11-01 | 9 | 411.66 |
| 429 | 449 | 146 | 2023-11-01 | 7 | 101.50 |
| 432 | 420 | 168 | 2023-11-04 | 3 | 42.39 |
| 257 | 123 | 403 | 2023-11-06 | 1 | 15.01 |

# 5. RETRIEVE THE TOTAL STOCK OF BOOKS AVAILABLE

**SOLUTION**

```
SELECT
    SUM(stock) AS Total_Stocks
FROM
    books;
```

**RESULT**

| Total_Stocks |
| --- |
| 25056 |

**Explanation**

- The SUM(stock) function calculates the total number of books available.
- The AS Total_Stocks renames the result column for better readability.
- This helps in understanding the overall inventory available for sale.

# 6. FIND THE DETAILS OF THE MOST EXPENSIVE BOOK

## SOLUTION

```
SELECT
    *
FROM
    books
ORDER BY price DESC
LIMIT 1;
```

## Explanation

- ORDER BY price DESC sorts the books in descending order based on price.
- LIMIT 1 ensures only the most expensive book is retrieved.
- This helps in identifying high-value books in the inventory.

## RESULT

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell C

| Book_ID | Title | Author | Genre | Published_Year | Price | Stock |
|---|---|---|---|---|---|---|
| 340 | Proactive system-worthy orchestration | Robert Scott | Mystery | 1907 | 49.98 | 88 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

# 7. SHOW ALL CUSTOMERS WHO ORDERED MORE THAN 1 QUANTITY OF A BOOK

**SOLUTION**

```sql
SELECT
    c.Customer_ID,
    c.Name,
    c.Email,
    c.Phone,
    c.city,
    c.country,
    o.quantity
FROM
    customers c
        JOIN
    orders o ON o.customer_id = c.Customer_ID
WHERE
    o.quantity > 1
ORDER BY o.quantity DESC;
```

**Explanation**

- The JOIN operation links the customers and orders tables using Customer_ID.
- The WHERE o.Quantity > 1 filters customers who ordered more than one quantity of a book.
- ORDER BY o.Quantity DESC sorts the results, showing customers with the highest orders first.
- This helps in identifying frequent or bulk buyers.

**RESULT**

| Customer_ID | Name | Email | Phone | city | country | quantity |
|---|---|---|---|---|---|---|
| 137 | Steven Miller | tsummers@yahoo.com | 1234568027 | North Keith | Papua New Guinea | 10 |
| 265 | Cassandra Cole | mckenziealfred@gmail.com | 1234568155 | Port Erinberg | Iraq | 10 |
| 348 | Matthew Gardner | wayne40@wilkerson.com | 1234568238 | Gentryfort | Mozambique | 10 |
| 246 | Jeffery Lewis | marytorres@wolfe.biz | 1234568136 | Angelastad | Dominican Republic | 10 |
| 461 | Crystal Pierce | webbteresa@gutierrez-garcia.com | 1234568351 | Port Davidhaven | Denmark | 10 |

# 8. RETRIEVE ALL ORDERS WHERE THE TOTAL AMOUNT EXCEEDS $20

**SOLUTION**

```sql
SELECT
    *
FROM
    orders
WHERE
    total_amount > 20
ORDER BY total_amount;
```

**Explanation**

- The WHERE total_amount > 20 filters orders where the total purchase value is greater than $20.
- ORDER BY total_amount sorts the orders in ascending order based on total amount.
- This helps in identifying higher-value transactions and customer spending patterns.

**RESULT**

| Order_ID | Customer_ID | Book_ID | Order_Date | Quantity | Total_Amount |
|---|---|---|---|---|---|
| 307 | 368 | 133 | 2023-11-17 | 1 | 20.96 |
| 100 | 207 | 63 | 2023-07-14 | 1 | 22.38 |
| 260 | 112 | 170 | 2024-05-07 | 2 | 22.56 |
| 319 | 172 | 296 | 2024-06-27 | 2 | 22.98 |
| 345 | 325 | 253 | 2023-07-04 | 2 | 23.32 |

# 9. LIST ALL GENRES AVAILABLE IN THE BOOKS TABLE

**SOLUTION**

```
SELECT DISTINCT
    genre
FROM
    books;
```

**RESULT**

| genre |
| --- |
| Biography |
| Fantasy |
| Non-Fiction |
| Fiction |
| Romance |
| Science Fiction |

**Explanation**

- SELECT DISTINCT genre ensures only unique genres are retrieved.
- This helps in understanding the variety of books available in the store.
- Useful for categorizing books and making genre-based recommendations.

# 10. CALCULATE THE TOTAL REVENUE GENERATED FROM ALL ORDERS

**SOLUTION**

```
SELECT
    SUM(Total_Amount) AS Total_revenue
FROM
    orders;
```

**RESULT**

| | Total_revenue |
|---|---|
| ▶ | 75443.27 |

**Explanation**

- SUM(Total_Amount) calculates the total revenue from all orders.
- AS Total_Revenue renames the result column for clarity.
- This helps in understanding the overall sales performance of the bookstore.

# 11. RETRIEVE THE TOTAL NUMBER OF BOOKS SOLD FOR EACH GENRE

**SOLUTION**

```sql
SELECT
    SUM(o.quantity) AS Book_sold, b.genre AS genre
FROM
    orders o
        JOIN
    books b ON o.Book_ID = b.Book_ID
GROUP BY genre
ORDER BY Book_sold DESC;
```

**RESULT**

| Book_sold | genre |
|---|---|
| 504 | Mystery |
| 447 | Science Fiction |
| 446 | Fantasy |
| 433 | Romance |
| 351 | Non-Fiction |
| 284 | Biography |

**Explanation**

- SUM(o.Quantity) calculates the total number of books sold.
- JOIN books b ON o.Book_ID = b.Book_ID links the orders table with the books table.
- GROUP BY genre groups the results by book genre, allowing for a genre-wise sales summary.
- ORDER BY Book_Sold DESC sorts the results by the highest number of books sold.

# 12. FIND THE AVERAGE PRICE OF BOOKS IN THE "FANTASY" GENRE

**SOLUTION**

```
SELECT
    AVG(price) AS verage_price
FROM
    books
WHERE
    genre = 'Fantasy';
```

**RESULT**

| | verage_price |
|---|---|
| ▶ | 25.981690 |

**Explanation**

- AVG(price) calculates the average price of the books in the Fantasy genre.
- The WHERE genre = 'Fantasy' filters the books to include only those in the Fantasy genre.

# 13. LIST CUSTOMERS WHO HAVE PLACED AT LEAST 2 ORDERS

**SOLUTION**

```sql
SELECT
    c.Customer_ID,
    c.name,
    c.email,
    c.phone,
    c.city,
    c.country,
    COUNT(o.Order_ID) AS total_orders
FROM
    customers c
        JOIN
    Orders o ON o.Customer_ID = c.Customer_ID
GROUP BY c.Customer_ID
HAVING COUNT(o.Order_ID) >= 2
ORDER BY total_orders DESC;
```

**Explanation**

- The JOIN operation links customers and orders using Customer_ID.
- COUNT(o.Order_ID) counts the number of orders placed by each customer.
- HAVING COUNT(o.Order_ID) >= 2 filters customers who have placed two or more orders.
- ORDER BY Total_Orders DESC sorts the customers by the number of orders placed in descending order.

**RESULT**

| Customer_ID | name | email | phone | city | country | total_orders |
|---|---|---|---|---|---|---|
| 364 | Carrie Perez | chelsea23@gillespie-walker.com | 1234568254 | Kennethland | Hungary | 6 |
| 474 | Anthony Young | rogersbill@gmail.com | 1234568364 | East Chelsea | Cook Islands | 5 |
| 107 | Amy Hunt | emilybecker@perkins.com | 1234567997 | Ericborough | Aruba | 4 |
| 174 | Jonathon Strickland | ryan10@yahoo.com | 1234568064 | Bakerton | Dominica | 4 |

# 14. FIND THE MOST FREQUENTLY ORDERED BOOK

**SOLUTION**

```sql
SELECT b.Title, o.Book_ID, COUNT(o.Order_ID) AS total_orders
FROM Orders o
JOIN Books b ON o.Book_ID = b.Book_ID
GROUP BY o.Book_ID, b.Title
HAVING total_orders = (
    SELECT MAX(order_count)
    FROM (
        SELECT COUNT(Order_ID) AS order_count
        FROM Orders
        GROUP BY Book_ID
    ) AS subquery
);
```

**Explanation**

- The JOIN operation links the Orders and Books tables using Book_ID.
- COUNT(o.Order_ID) counts the number of times each book has been ordered.
- The HAVING clause ensures that only the book(s) with the maximum number of orders are selected.
- A subquery is used to find the maximum number of orders across all books.

**RESULT**

| Title | Book_ID | total_orders |
|---|---|---|
| Implemented encompassing conglomeration | 31 | 4 |
| Realigned multi-tasking installation | 73 | 4 |
| Robust tangible hardware | 88 | 4 |
| Integrated secondary access | 120 | 4 |
| Devolved zero administration process improvem... | 273 | 4 |

# 15. SHOW THE TOP 3 MOST EXPENSIVE BOOKS OF 'FANTASY' GENRE

**SOLUTION**

```sql
SELECT
    *
FROM
    books
WHERE
    genre = 'Fantasy'
ORDER BY price DESC
LIMIT 3;
```

**Explanation**

- The WHERE genre = 'Fantasy' filters books in the Fantasy genre.
- ORDER BY price DESC sorts the books in descending order by price, showing the most expensive ones first.
- LIMIT 3 ensures only the top 3 most expensive books are selected.

**RESULT**

| Book_ID | Title | Author | Genre | Published_Year | Price | Stock |
|---|---|---|---|---|---|---|
| 240 | Stand-alone content-based hub | Lisa Ellis | Fantasy | 1957 | 49.90 | 41 |
| 462 | Innovative 3rdgeneration database | Allison Contreras | Fantasy | 1988 | 49.23 | 62 |
| 238 | Optimized even-keeled analyzer | Sherri Griffith | Fantasy | 1975 | 48.97 | 72 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

# 16. RETRIEVE THE TOTAL QUANTITY OF BOOKS SOLD BY EACH AUTHOR

**SOLUTION**

```
SELECT
    SUM(o.quantity) AS Book_sold, b.author
FROM
    orders o
        JOIN
    books b ON o.Book_ID = b.Book_ID
GROUP BY b.Author
ORDER BY Book_sold DESC;
```

**RESULT**

| | Book_sold | author |
|---|---|---|
| ▶ | 28 | Patrick Contreras |
| | 27 | Melissa Taylor |
| | 24 | Emily James |
| | 24 | Thomas Trujillo |
| | 23 | Erica Parker |

**Explanation**

- SUM(o.Quantity) calculates the total number of books sold by each author.
- The JOIN operation links the orders and books tables using Book_ID.
- GROUP BY b.Author groups the results by author to calculate the total sales per author.
- ORDER BY Book_Sold DESC sorts the authors by the number of books sold, showing the most successful authors first.

# 17. LIST THE CITIES WHERE CUSTOMERS WHO SPENT OVER $30 ARE LOCATED

**SOLUTION**

```sql
SELECT
    c.city, SUM(o.total_amount) AS amount_spent
FROM
    orders o
        JOIN
    customers c ON o.Customer_ID = c.Customer_ID
GROUP BY c.city
HAVING SUM(o.Total_Amount) > 30
ORDER BY amount_spent;
```

**RESULT**

| city | amount_spent |
|---|---|
| East Tracystad | 30.02 |
| Nelsonmouth | 31.48 |
| North Emily | 31.68 |
| Parkerside | 32.82 |
| Conniefort | 33.45 |

**Explanation**

- The JOIN operation links the orders and customers tables using Customer_ID.
- SUM(o.Total_Amount) calculates the total spending per customer in each city.
- GROUP BY c.City groups the data by city to calculate total spending for each city.
- HAVING SUM(o.Total_Amount) > 30 filters cities where the total spending exceeds $30.
- ORDER BY Amount_Spent sorts cities based on total amount spent, from lowest to highest.

# 18. FIND THE CUSTOMER WHO SPENT THE MOST ON ORDERS

**SOLUTION**

```sql
SELECT
    c.name, SUM(o.total_amount) AS Total_spent
FROM
    orders o
        JOIN
    customers c ON o.Customer_ID = c.Customer_ID
GROUP BY c.name
ORDER BY Total_spent DESC
LIMIT 1;
```

**RESULT**

| | name | Total_spent |
|---|---|---|
| ▶ | Kim Turner | 1398.90 |

**Explanation**

- The JOIN operation links the orders and customers tables using Customer_ID.
- SUM(o.Total_Amount) calculates the total amount spent by each customer.
- GROUP BY c.Name groups the data by customer name to calculate total spending per customer.
- ORDER BY Total_Spent DESC sorts the customers by total spending in descending order, showing the highest spender first.
- LIMIT 1 ensures that only the customer with the highest total is selected.

# 19. FIND THE CUSTOMER WHO SPENT THE MOST ON ORDERS

**SOLUTION**

```sql
SELECT
    b.Title,
    b.Book_ID,
    b.Stock,
    COALESCE(SUM(o.Quantity), 0) AS Sold,
    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 , b.Stock
ORDER BY Remaining_Stock DESC;
```

**Explanation**

- LEFT JOIN ensures all books are included, even those with no orders.
- COALESCE(SUM(o.Quantity), 0) calculates the total number of books sold, defaulting to 0 if there are no orders.
- b.Stock - COALESCE(SUM(o.Quantity), 0) calculates the remaining stock after fulfilling all orders.
- GROUP BY groups the results by Book_ID, Title, and Stock to calculate the stock and sales for each book.
- ORDER BY Remaining_Stock DESC sorts the books by the remaining stock, showing the highest remaining stock first.

**RESULT**

| Title | Book_ID | Stock | Sold | Remaining_Stock |
|---|---|---|---|---|
| Ergonomic foreground Graphic Interface | 173 | 100 | 0 | 100 |
| Customer-focused tertiary methodology | 193 | 100 | 0 | 100 |
| Progressive 24hour artificial intelligence | 284 | 99 | 0 | 99 |
| Secured maximized time-frame | 359 | 99 | 0 | 99 |
| Synchronized impactful synergy | 365 | 99 | 0 | 99 |

# CONCLUSION

**This project provided valuable insights into the operations of an online bookstore using MySQL. Key findings include:**

- Popular genres and authors based on sales
- High-value customers and repeat buyers
- Inventory analysis, including remaining stock and top-selling books


## Key Takeaways:

- SQL queries helped extract actionable data for business decisions.
- Understanding customer spending behavior and popular genres enables better stock management and targeted marketing.
- The analysis offers a foundation for further improvements in sales strategies and inventory management.