SQL Project Report: Online Bookstore Analysis

Project Overview

The goal of this project was to analyze sales, inventory, and customer behavior for an online bookstore using SQL. Key business questions included identifying top-selling books, customer purchase trends, inactive stock, and optimizing stock levels.

Problem Statement

The business lacked visibility into which books, authors, and genres were contributing most to overall revenue.

There was no structured method to track customer order behavior and purchasing frequency over time.

Inactive and low-selling inventory could not be easily identified, resulting in inefficient stock management.

Sales trends across days and months were not analyzed, limiting opportunities for optimized promotions and timing.

Existing SQL logic was not reusable or optimized for performance, making reporting tasks manual and repetitive.

Tools Used

- MySQL 8.0
- MySQL Workbench
- Excel (for final exports and reporting)
- Python (optional, for query integration and automation)

Project Process Brief

1. Data Exploration

   - Examined table structures (`books`, `customers`, `orders`)

   - Ran basic `SELECT` queries to understand data

2. Basic Analysis

- Filtered books by genre, publication year, and stock levels
- Calculated total revenue, average book prices, and sales trends

3. Advanced Analytic

- Used CTEs to track month-over-month sales growth
- Applied window functions to rank books by revenue within genres
- Identified customers with consistent purchasing behavior

4. Automation & Optimization

- Created stored procedures for customer summaries and genre-based sales reports
- Developed a view (`book_inventory_status`) to track stock levels

5. Business Insights

- Found peak order days (e.g., weekends vs. weekdays)
- Detected books sold below listed price (possible discounts)

Review & Optimization

- Used indexing suggestions for performance improvement
- Avoided redundant joins and unnecessary nested queries
- Simplified logic using CTEs and CASE statements
- Ensured queries complied with `ONLY_FULL_GROUP_BY` mode

Data Extraction Queries

1. Retrieve the total number of books sold for each genre:

```
SELECT
    Genre, SUM(quantity)
FROM
    books
        JOIN
    orders ON books.Book_ID = orders.Order_ID
GROUP BY Genre;
```

2. Retrieve the total quantity of books sold by each author:

```
SELECT
    Author, SUM(Quantity)
FROM
    books
        JOIN
    orders ON books.Book_ID = orders.Book_ID
GROUP BY Author;
```

3. Which books had the largest month-over-month sales increase:

```sql
WITH monthly_sales AS (
    SELECT
        Book_ID,
        DATE_FORMAT(Order_Date, '%Y-%m') AS year_months,
        SUM(Total_Amount) AS total_sales
    FROM orders
    GROUP BY Book_ID, DATE_FORMAT(Order_Date, '%Y-%m')
),
sales_with_diff AS (
    SELECT
        Book_ID,
        year_months,
        total_sales,
        LAG(total_sales) OVER (PARTITION BY Book_ID ORDER BY year_months) AS prev_month_sales,
        (total_sales - LAG(total_sales) OVER (PARTITION BY Book_ID ORDER BY year_months)) AS sales_increase
    FROM monthly_sales
)
SELECT *
FROM sales_with_diff
ORDER BY sales_increase DESC
LIMIT 5;
```

4. Which customers consistently placed orders every month for the last 3 months:

```sql
WITH month_orders AS (
    SELECT
        Customer_ID,
        DATE_FORMAT(Order_Date, '%Y-%m') AS order_month
    FROM orders
    WHERE DATE_FORMAT(Order_Date, '%Y-%m') IN ('2024-12', '2024-11', '2024-10')
),
customer_month_count AS (
    SELECT
        Customer_ID,
        COUNT(DISTINCT order_month) AS months_active
    FROM month_orders
    GROUP BY Customer_ID
)
SELECT Customer_ID
FROM customer_month_count
WHERE months_active = 3;
```

5. Rank books by total revenue within each genre:

```sql
WITH Total_sales AS (
    SELECT
        Book_ID,
        SUM(Total_Amount) AS sales
    FROM orders
    GROUP BY Book_ID
)

SELECT
    b.Book_ID,
    b.Title,
    b.Genre,
    ts.sales,
    DENSE_RANK() OVER (
        PARTITION BY b.Genre
        ORDER BY ts.sales DESC
    ) AS revenue_rank
FROM Total_sales ts
JOIN books b ON ts.Book_ID = b.Book_ID;
```

6. Which books have been sold at prices lower than their listed price (Price column):

```sql
WITH sell_price AS (
    SELECT
        o.Order_ID,
        o.Book_ID,
        o.Quantity,
        o.Total_Amount,
        (o.Total_Amount / o.Quantity) AS Selling_Price
    FROM orders o
)

SELECT
    b.Book_ID,
    b.Title,
    b.Author,
    b.Genre,
    b.Price AS Listed_Price,
    sp.Selling_Price,
    sp.Order_ID
FROM sell_price sp
JOIN books b ON sp.Book_ID = b.Book_ID
WHERE sp.Selling_Price < b.Price;
```

7. Which genre has the highest average revenue per book sold:

```sql
SELECT
    b.Genre,
    AVG(o.Total_Amount / o.Quantity) AS avg_revenue_per_book
FROM orders o
JOIN books b ON o.Book_ID = b.Book_ID
GROUP BY b.Genre
ORDER BY avg_revenue_per_book DESC
LIMIT 1;
```

8. Use a CTE to find books that have sold more than the average quantity sold across all books:

```sql
WITH book_sales AS (
    SELECT
        Book_ID,
        SUM(Quantity) AS total_quantity
    FROM orders
    GROUP BY Book_ID
),
overall_avg AS (
    SELECT AVG(total_quantity) AS avg_quantity_sold
    FROM book_sales
)
SELECT
    bs.Book_ID,
    bs.total_quantity
FROM book_sales bs
JOIN overall_avg oa ON 1=1
WHERE bs.total_quantity > oa.avg_quantity_sold;
```

9. Use a CTE to get total sales per customer, then list only those above the overall customer average:

```sql
WITH Total_sales AS (
    SELECT
        Customer_ID,
        SUM(Total_Amount) AS total_sales
    FROM orders
    GROUP BY Customer_ID
),
overall_avg AS (
    SELECT
        AVG(total_sales) AS avg_sales_per_customer
    FROM Total_sales
)
SELECT
    ts.Customer_ID,
    ts.total_sales,
    avg_sales_per_customer
FROM Total_sales ts
JOIN overall_avg oa ON 1=1
WHERE ts.total_sales > oa.avg_sales_per_customer;
```

10. Create a view showing book title, total quantity sold, remaining stock, and stock status (e.g., 'In Stock', 'Low Stock', 'Out of Stock').

```sql
CREATE VIEW book_inventory_status AS
WITH total_sold AS (
    SELECT
        Book_ID,
        SUM(Quantity) AS quantity_sold
    FROM orders
    GROUP BY Book_ID
)
SELECT
    b.Book_ID,
    b.Title,
    COALESCE(ts.quantity_sold, 0) AS quantity_sold,
    b.Stock AS remaining_stock,

    CASE
        WHEN b.Stock = 0 THEN 'Out of Stock'
        WHEN b.
        Stock <= 10 THEN 'Low Stock'
        ELSE 'In Stock'
    END AS stock_status

FROM books b
LEFT JOIN total_sold ts ON b.Book_ID = ts.Book_ID;
```

11. Write a stored procedure that takes a customer ID and returns:

- Number of orders
- Total quantity ordered
- Total amount spent

```
DELIMITER $$

CREATE PROCEDURE get_customer_summary(IN cust_id INT)
BEGIN
    SELECT
        COUNT(Order_ID) AS total_orders,
        SUM(Quantity) AS total_quantity,
        SUM(Total_Amount) AS total_spent
    FROM orders
    WHERE Customer_ID = cust_id;
END$$
```

12. Write a stored procedure that accepts a genre and returns the top 3 selling books in that genre:

```
DELIMITER $$

CREATE PROCEDURE top_3_books_by_genre(IN gen VARCHAR(50))
BEGIN
    -- Get top 3 books by total sales in the specified genre
    SELECT
        b.Book_ID,
        b.Title,
        b.Genre,
        SUM(o.Total_Amount) AS total_sales
    FROM books b
    JOIN orders o ON b.Book_ID = o.Book_ID
    WHERE b.Genre = gen
    GROUP BY b.Book_ID, b.Title, b.Genre
    ORDER BY total_sales DESC
    LIMIT 3;
END$$

DELIMITER ;
```

13. Identify peak order days of the week:

```sql
SELECT
    DAYNAME(Order_Date) AS order_day,
    COUNT(*) AS total_orders
FROM orders
GROUP BY order_day
ORDER BY total_orders DESC;
```

Project Outcome

Sales Insights

- Identified top-selling genres (Fantasy had the highest avg. revenue per book)
- Found monthly sales trends (best/worst-performing months)

Customer Behavior

- Discovered most loyal customers (those ordering every month)
- Calculated average spending per customer**

Inventory Management

- Tracked low-stock books (needing restocking)
- Detected unsold books (potential clearance candidates)

Automation

- Stored procedures simplified repetitive queries (e.g., customer summaries)
- Views provided quick inventory status checks

Problems Faced

- Encountered errors due to strict SQL modes (e.g., `ONLY_FULL_GROUP_BY`)
- Needed to debug logic when using LAG/LEAD with NULL values
- Initial confusion around multi-step transformations using CTEs
- Schema inconsistencies (e.g., date formats) that required standardization

Key Learnings

- Stronger understanding of CTEs and window functions in MySQL
- Practice in writing scalable and optimized queries
- Experience with stored procedures for business automation
- Real-world analytics thinking using structured SQL logic