# DATA CONCEPTS
# DATA -037-008

**STUDENT NAME:** PRAISELIN LYDIA GLADSTON

**STUDENT ID:** 000965570

# A CASE STUDY
# ON E-COMMERCE DATABASE

# Contents

# 1. Introduction

As a growing data scientist, it is important to learn to focus on a business to be able to analyze its data. The first step on analysis is to understand the structure and implementation of the database, the relationship between the tables (entities) and the flow of data.

This case study explores the structure and operations of an e-commerce clothing store, emphasizing data organization, inventory management, and customer satisfaction. We will delve into database design, queries, and business intelligence insights crucial for optimizing performance.

With the existence of e-commerce giants like Amazon, Shopify, Temu etc., it is made easy to understand what drives their business, how they make data-driven decisions and how they build a scalable and efficient database to handle their ever-growing customer data. With this understanding, we are going to create and explore an e-commerce database, queries and views that are normally executed on it to make decisions.

# 2. Mission Statement

To create a service that provides high-quality service, innovative, and affordable clothing that enhances everyday life for people worldwide.

# 3. Business Objectives

The journey of working towards a mission, will have several intermediate tasks that have to be completed to get the preferred outcome. Such tasks are called objectives and are made usually made measurable in a real-world scenario.

In our case study, since we are not aiming to solve a real business problem, we will stick to having generalized objectives to achieve our goal. Our objectives for our mission will be as follows:

1. Ensure a seamless shopping experience across the online platform.

   - This includes designing a well thought through website to engage and guide a user, providing uninterrupted services like payments, shipping, receiving, returns.
2. Utilize data-driven strategies to understand customer preferences and improve product offerings.
   - This includes identifying top selling products that produce more revenue, analyzing product reviews and customer feedback to make further improvements in service, making innovative features etc.
3. Maintain inventory efficiency and ensure timely product delivery.
   - This includes making sure high demand products are often restocked, flagging low stock items.

# 4. Database Structure

We will walk through the data flow of our business to identify the entities in it which will become the tables in our database.

1. There is **Customer** who will be the end user of our service. We will require their name, address and contact details.
2. The customer is going to search for **Products** for which there should be a description and price at which it is being sold for.
3. The product will have a **Supplier** whose name and contact details should be available if the product must be restocked.
4. There should be **Inventory** that handles the stock details of the products.
5. The products must be segregated into **Categories** which makes it easier to filter them out.
6. After looking for the desired products, the customer is going to **Order** them. The order should consist of details of what products are being ordered, their quantity, when the order is placed, when its shipped, status of the order, who is processing the order.
7. The customer should make **Payment** for the order to be confirmed. The details of the transaction should consist of payment method, status of the payment and the order for which the payment was made.
8. The order would be processed by an **Employee** who will assign shipping details and handle departure of the product.

   From this information, we can decide on the structure of the database and can get an idea of what relationship is to be established between them.

The entire database can be replicated in local database with this file. The SQL file was created on a MySQL database.

GitHub - https://www.github.com/praiselydia
Linkedin – https://www.linkedin.com/praiselydia

## List of entities:

| Name | Description |
|---|---|
| Customers | Stores customer information. |
| Orders | Contains details of customer purchases. |
| Payments | Captures payment transactions for orders. |
| Employees | Stores employee details related to order processing. |
| Products | Contains product details, pricing, and stock availability. |
| Categories | Classifies products into relevant groups. |
| Suppliers | Stores supplier information for inventory management. |
| Inventory | Tracks stock levels of products. |

## Entity Relationships:

There are 4 different types of relationships between entities:

1. One-to-one: Each entity in set A is related to at most one entity in set B, and vice versa.
2. One-to-many: A single entity in set A can be related to multiple entities in set B, but each entity in set B relates to only one entity in set A.
3. Many-to-one: This is the inverse of One-to-Many; multiple entities in set A relate to a single entity in set B.
4. Many-to-many: Entities in set A can be associated with multiple entities in set B, and vice versa.

| Entities | Relationship | Details |
|---|---|---|
| Customers -> Orders | One to Many | Multiple orders per customers |
| Orders -> Payments | One to One | Single payment for single order |

| | | |
|---|---|---|
| Orders -> Employees | One to Many | Multiple order processing per employee |
| Orders -> Products | Many to Many | Order can consist of multiple products |
| Products -> Categories | One to Many | Multiple products belong to single category |
| Products -> Suppliers | One to Many | Multiple products per supplier |
| Products -> Inventory | One to One | Product has unique identifier for inventory |

Here, Orders -> Products has many-to-many relationship which is not desired since it violates normalization principles of preventing redundancy and inconsistencies in a relational database. In this case, we introduce a linking table Orders_Products which will hold M:1 relationship with the entities. We will have the following 2 relationships

| Entities | Relationship | Details |
|---|---|---|
| Orders -> Orders_Products | One to Many | Multiple products per order |
| Products -> Orders_Products | One to Many | Multiple orders for single product |

## Data Dictionary:

To create a database, we will have to decide on the attributes of the tables that will be helpful in maintaining the data and make queries on them to extract information which will be useful in making business decisions. Below are the tables' attributes and their specifications for our use case:

### Categories:

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| CategoryID | INT | PRIMARY KEY | Unique identifier for each category. |

| | | | |
|---|---|---|---|
| CategoryName | VARCHAR(255) | NOT NULL | Name of the product category. |
| Description | VARCHAR(255) | None | Detailed information about the category. |

## Suppliers:

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| SupplierID | INT | PRIMARY KEY | Unique identifier for each supplier. |
| SupplierName | VARCHAR(255) | NOT NULL | Name of the supplier. |
| ContactName | VARCHAR(255) | NOT NULL | Name of the primary contact person for the supplier. |
| ContactEmail | VARCHAR(255) | NOT NULL | Email address of the supplier’s contact person. |
| Phone | VARCHAR(15) | NOT NULL | Supplier’s contact phone number. |
| Address | VARCHAR(255) | None | Physical address of the supplier. |

## Products:

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| ProductID | INT | PRIMARY KEY | Unique identifier for each product. |

| | | | |
|---|---|---|---|
| ProductName | VARCHAR(255) | NOT NULL | Name of the product. |
| Description | VARCHAR(255) | None | Brief details about the product. |
| Price | FLOAT | NOT NULL | Cost of the product. |
| CategoryID | INT | FOREIGN KEY | The category to which the product belongs. |
| SupplierID | INT | FOREIGN KEY | The supplier who provides the product. |

## Inventory:

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| InventoryID | INT | PRIMARY KEY | Unique identifier for inventory records. |
| ProductID | INT | FOREIGN KEY, NOT NULL | Product associated with the inventory. |
| StockLevel | INT | None | Current stock quantity. |
| ReorderLevel | INT | None | Threshold at which stock should be reordered. |

## Customers:

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| CustomerID | INT | PRIMARY KEY | Unique identifier for each customer. |

| | | | |
|---|---|---|---|
| FirstName | VARCHAR(255) | NOT NULL | Customer's first name. |
| LastName | VARCHAR(255) | NOT NULL | Customer's last name. |
| Email | VARCHAR(255) | NOT NULL | Customer's email address. |
| Phone | VARCHAR(255) | NOT NULL | Customer's contact number. |
| Address | VARCHAR(255) | None | Customer's address. |
| City | VARCHAR(255) | None | City where the customer resides. |
| State | VARCHAR(255) | None | State where the customer resides. |
| PostalCode | VARCHAR(255) | None | ZIP or postal code of the customer’s location. |
| Country | VARCHAR(255) | None | Country where the customer lives. |

## Employees:

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| EmployeeID | INT | PRIMARY KEY | Unique identifier for employees. |
| FirstName | VARCHAR(255) | NOT NULL | Employee’s first name. |
| LastName | VARCHAR(255) | NOT NULL | Employee’s last name. |
| JobTitle | VARCHAR(255) | NOT NULL | Employee’s job title or role. |
| DepartmentID | INT | None | Department to which the employee belongs. |

| | | | |
|---|---|---|---|
| ManagerID | INT | None | Manger under whom the employee works for. |

## Orders:

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| OrderID | INT | PRIMARY KEY | Unique identifier for each order. |
| CustomerID | INT | NOT NULL, FOREIGN KEY | Customer who placed the order. |
| OrderDate | TIMESTAMP | None | Date when the order was placed. |
| ShippedDate | TIMESTAMP | None | Date when the order was shipped. |
| TotalAmount | FLOAT | None | Total price of the order. |
| Status | VARCHAR(255) | None | Current status of the order (e.g., pending, shipped, delivered). |
| EmployeeID | INT | FOREIGN KEY | Employee responsible for the order |

## **Orders_Products**:

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| OrderID | INT | COMPOSITE FOREIGN KEY | Order associated with the product |

| | | | |
|---|---|---|---|
| ProductID | INT | COMPOSITE FOREIGN KEY | Product included in the order. |
| Quantity | INT | DEFAULT 1 | Number of units of the product in the order. |

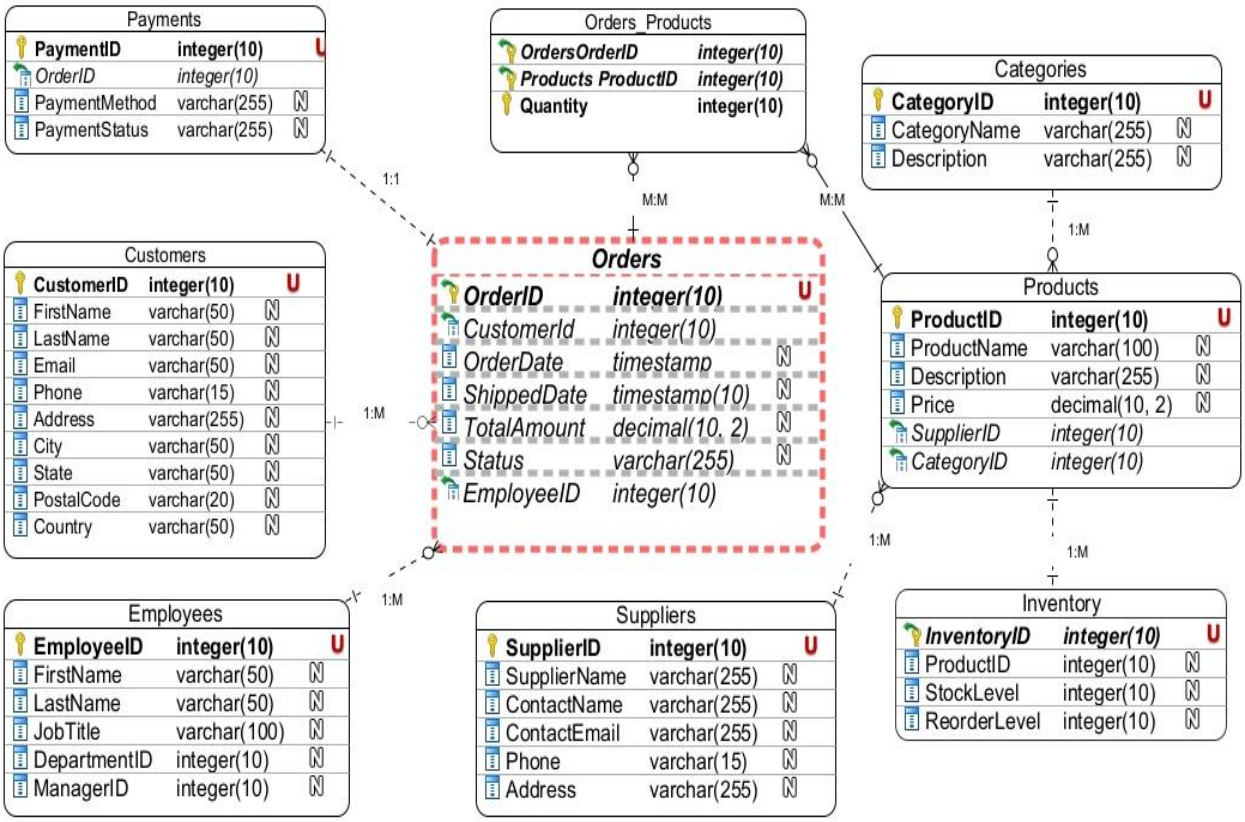
## **Payments**:

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| PaymentID | INT | PRIMARY KEY | Unique identifier for each payment. |
| OrderID | INT | NOT NULL, FOREIGN KEY | Order for which the payment was made |
| PaymentMethod | VARCHAR(255) | None | Mode of payment (e.g., credit card, PayPal). |
| PaymentStatus | VARCHAR(255) | None | Status of the payment (e.g., pending, completed, failed). |

## Entity Relationship Diagram:

An entity relationship diagram pictographically shows how the relationship between tables is established and what kind of relationship it is.

For our use case, the database follows a snowflake schema with **Orders** being the fact table and the other tables surround it.




# 5. SQL Queries

SQL (Structured Query Language) is the standardized language used to store and retrieve data from a database. The types of queries that can be done over a database can be segregated into the following:

**1. Data Query Language (DQL) – Retrieving Data**

- Used to retrieve data from the database.
- Includes the SELECT statement.

**2. Data Manipulation Language (DML) – Modifying Data**

- Used to modify data within tables.
- Includes INSERT, UPDATE, and DELETE statements.

**3. Data Definition Language (DDL) – Defining Structure**

- Used to define or modify the structure of database objects.

- Includes CREATE, ALTER, DROP, and TRUNCATE statements.

**4. Transaction Control Language (TCL) – Managing Transactions**

- Used to manage transactions within the database.
- Includes COMMIT, ROLLBACK, and SAVEPOINT statements.

**5. Data Control Language (DCL) – Managing Permissions**

- Used to manage access and security permissions.
- Includes GRANT and REVOKE statements.

**6. Joins – Combining Data from Multiple Tables**

- Used to retrieve related data from multiple tables.
- Includes INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN.

In the case study, we will explore the queries that are useful to an e-commerce business to perform analysis on its data to improve its performance.

## Revenue Trend Analysis (Using GROUP BY and ORDER BY)

```
mysql> -- Revenue trend from Jan 2024
mysql>     SELECT
    ->         cat.CategoryName,
    ->         SUM(o.TotalAmount) AS TotalRevenue,
    ->         COUNT(DISTINCT o.OrderID) AS TotalOrders
    ->     FROM Orders o
    ->     JOIN Orders_Products op ON o.OrderID = op.OrderID
    ->     JOIN Products p ON op.ProductID = p.ProductID
    ->     JOIN Categories cat ON p.CategoryID = cat.CategoryID
    ->     WHERE o.OrderDate >= '2024-01-01'
    ->     GROUP BY cat.CategoryName
    ->     ORDER BY TotalRevenue DESC;
+-----------------+-------------------+-------------+
| CategoryName    | TotalRevenue      | TotalOrders |
+-----------------+-------------------+-------------+
| Male_Shirts     |             135.5 |           2 |
| Male_Tshirts    | 123.4900016784668 |           2 |
| Female_Trousers |  95.9900016784668 |           2 |
| Male_Trousers   |                80 |           1 |
| Female_Shirts   |  76.9900016784668 |           2 |
| Female_Tshirts  |  55.9900016784668 |           1 |
+-----------------+-------------------+-------------+
6 rows in set (0.00 sec)
```

## Returning Customers vs. One-Time Buyers (Using GROUP BY)

```
mysql> -- Returning Customers vs. One-Time Buyers
mysql> SELECT
    ->     CASE
    ->         WHEN order_count > 1 THEN 'Returning Customer'
    ->         ELSE 'One-Time Buyer'
    ->     END AS CustomerType,
    ->     COUNT(*) AS TotalCustomers
    -> FROM (
    ->     SELECT CustomerID, COUNT(OrderID) AS order_count
    ->     FROM Orders
    ->     GROUP BY CustomerID
    -> ) AS OrderSummary
    -> GROUP BY CustomerType;
+--------------------+----------------+
| CustomerType       | TotalCustomers |
+--------------------+----------------+
| Returning Customer |              2 |
| One-Time Buyer     |              3 |
+--------------------+----------------+
2 rows in set (0.00 sec)
```

## Employee Order Management Performance (Using JOIN and ORDER BY)

```
mysql> -- Employee Order Management from Jan 2024
mysql> SELECT
    ->     e.EmployeeID,
    ->     CONCAT(e.FirstName, ' ', e.LastName) AS EmployeeName,
    ->     COUNT(o.OrderID) AS TotalOrdersHandled
    -> FROM Orders o
    -> JOIN Employees e ON o.EmployeeID = e.EmployeeID
    -> WHERE o.OrderDate >= '2024-01-01'
    -> GROUP BY e.EmployeeID, EmployeeName
    -> ORDER BY TotalOrdersHandled DESC;
+------------+--------------+--------------------+
| EmployeeID | EmployeeName | TotalOrdersHandled |
+------------+--------------+--------------------+
|        601 | Liam Brown   |                  3 |
|        604 | Ava Miller   |                  2 |
|        603 | Noah Wilson  |                  2 |
+------------+--------------+--------------------+
3 rows in set (0.00 sec)
```

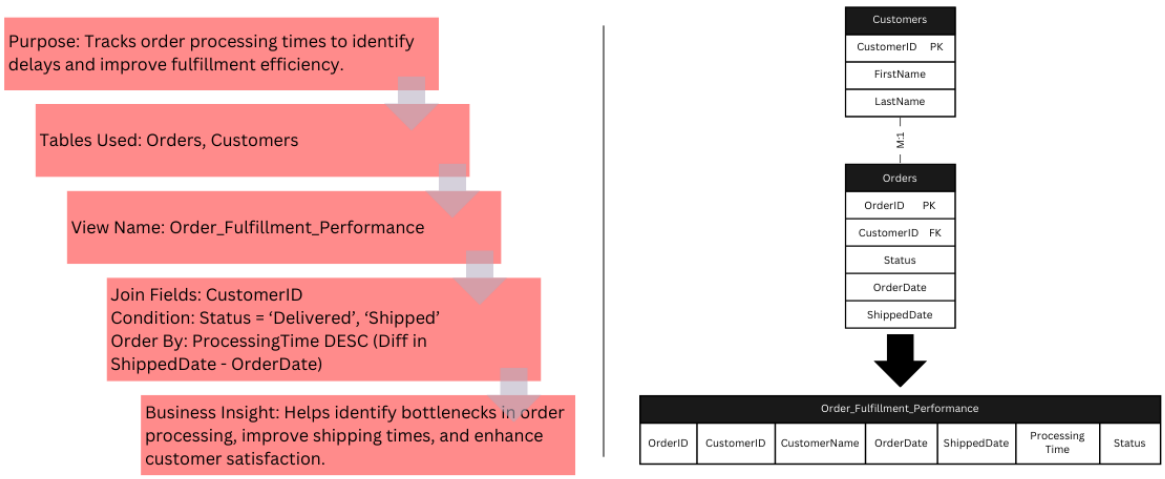
In an e-commerce business, data-driven insights help improve decision-making and drive growth. Revenue Trend Analysis using GROUP BY and ORDER BY allows businesses to track sales performance over time, identifying peak seasons and declining trends to adjust pricing, inventory, and marketing strategies accordingly. Returning Customers vs. One-Time Buyers helps segment customer behavior, enabling businesses to focus on customer retention strategies, personalized offers, and loyalty programs. Employee Order Management Performance using JOIN and ORDER BY helps assess employee efficiency in handling orders, ensuring better workflow management and identifying top performers. Together, these queries provide critical insights for optimizing revenue, customer experience, and operational efficiency.

# 6. Views for Business Insights

Views in a database help simplify complex queries, enhance security, and improve performance for an e-commerce business. They allow users to access specific data without exposing the underlying table structures, ensuring better data privacy—especially for customer and financial records. Views also help in generating **frequently used reports**, such as **sales summaries, top-selling products, or customer order history**, without requiring repeated complex queries. Additionally, they improve maintainability by providing a consistent data structure, even when the underlying tables change. This makes views a valuable tool for streamlining analytics and improving data accessibility.

In our use case we will use views that align with our business objectives.

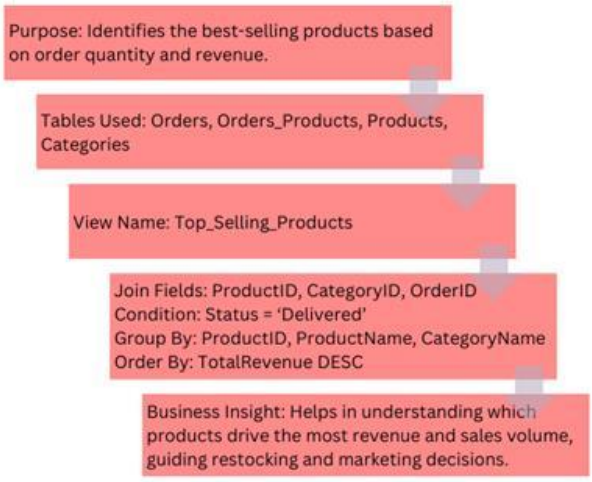
## Order Fulfillment Performance View

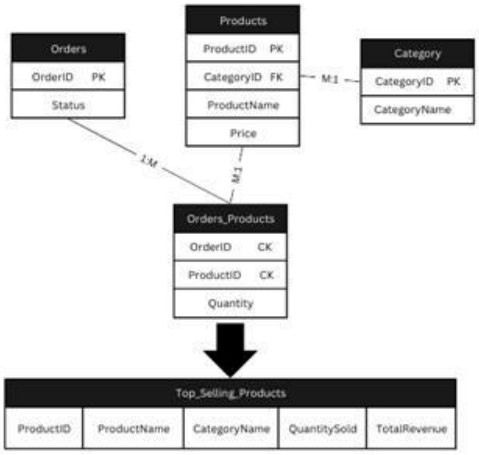


```
mysql> CREATE VIEW Order_Fulfillment_Performance_View AS
    -> SELECT
    ->     o.OrderID,
    ->     c.CustomerID,
    ->     CONCAT(c.FirstName, ' ', c.LastName) AS customer_name,
    ->     o.OrderDate,
    ->     o.ShippedDate,
    ->     DATEDIFF(o.ShippedDate, o.OrderDate) AS processing_time_days,
    ->     o.Status
    -> FROM Orders o
    -> JOIN Customers c ON o.CustomerID = c.CustomerID
    -> WHERE o.Status IN ('Delivered', 'Shipped')
    -> ORDER BY processing_time_days DESC;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from Order_Fulfillment_Performance_View;
+---------+------------+---------------+---------------------+---------------------+----------------------+-----------+
| OrderID | CustomerID | customer_name | OrderDate           | ShippedDate         | processing_time_days | Status    |
+---------+------------+---------------+---------------------+---------------------+----------------------+-----------+
|     700 |     500100 | Alex Roberts  | 2024-01-10 14:30:00 | 2024-01-11 10:00:00 |                    1 | Shipped   |
|     701 |     500101 | Jennifer Kirk | 2024-01-11 10:15:00 | 2024-01-12 12:00:00 |                    1 | Delivered |
|     703 |     500103 | Sophia Grace  | 2024-01-13 13:10:00 | 2024-01-14 09:30:00 |                    1 | Shipped   |
|     704 |     500104 | Henry Powell  | 2024-01-14 11:55:00 | 2024-01-15 14:20:00 |                    1 | Delivered |
+---------+------------+---------------+---------------------+---------------------+----------------------+-----------+
4 rows in set (0.00 sec)
```

## Top Selling Product View



```
mysql> CREATE VIEW Top_Selling_Products_View AS
    -> SELECT
    ->     p.ProductID,
    ->     p.ProductName,
    ->     c.CategoryName,
    ->     SUM(op.Quantity) AS total_quantity_sold,
    ->     SUM(op.Quantity * p.Price) AS total_revenue
    -> FROM Orders_Products op
    -> JOIN Products p ON op.ProductID = p.ProductID
    -> JOIN Categories c ON p.CategoryID = c.CategoryID
    -> JOIN Orders o ON op.OrderID = o.OrderID
    -> WHERE o.Status = 'Delivered'
    -> GROUP BY p.ProductID, p.ProductName, c.CategoryName
    -> ORDER BY total_revenue DESC;
Query OK, 0 rows affected (0.03 sec)

mysql> select * from Top_Selling_Products_View;
+-----------+---------------+---------------+---------------------+---------------+
| ProductID | ProductName   | CategoryName  | total_quantity_sold | total_revenue |
+-----------+---------------+---------------+---------------------+---------------+
|       302 | Men Blue Jeans | Male_Trousers |                   2 |            80 |
|       305 | Women Blouse  | Female_Shirts |                   1 |          28.5 |
+-----------+---------------+---------------+---------------------+---------------+
2 rows in set (0.04 sec)
```

## Low Stock Products View

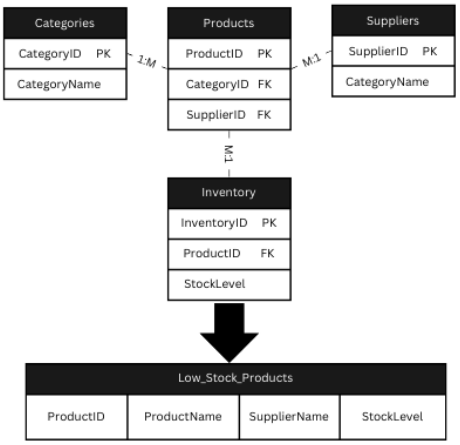
Purpose: Lists products that are low on stock, helping in inventory management.

Tables Used: Inventory, Products, Categories, Suppliers

View Name: Low_Stock_Products

Join Fields: ProductID, CategoryID, SupplierID
Order By: StockLevel DESC

Business Insight: Enables proactive restocking by identifying products with critically low stock levels.



```
mysql> CREATE VIEW Low_Stock_Products_View AS
    -> SELECT
    ->     p.ProductID,
    ->     p.ProductName,
    ->     c.CategoryName,
    ->     i.StockLevel,
    ->     s.SupplierName
    -> FROM Inventory i
    -> JOIN Products p ON i.ProductID = p.ProductID
    -> JOIN Categories c ON p.CategoryID = c.CategoryID
    -> JOIN Suppliers s ON p.SupplierID = s.SupplierID
    -> WHERE i.StockLevel < 50
    -> ORDER BY i.StockLevel ASC;
Query OK, 0 rows affected (0.05 sec)

mysql> select * from Low_Stock_Products_View;
+-----------+------------------+-----------------+------------+------------------+
| ProductID | ProductName      | CategoryName    | StockLevel | SupplierName     |
+-----------+------------------+-----------------+------------+------------------+
|       303 | Women Black Jeans | Female_Trousers |         30 | Trendy Wear Inc. |
|       305 | Women Blouse      | Female_Shirts   |         35 | Trendy Wear Inc. |
|       302 | Men Blue Jeans    | Male_Trousers   |         40 | ABC Clothing Co. |
|       304 | Men Formal Shirt  | Male_Shirts     |         45 | ABC Clothing Co. |
+-----------+------------------+-----------------+------------+------------------+
4 rows in set (0.00 sec)
```

# 7. Conclusion

This case study demonstrates how a well-structured e-commerce database enables seamless operations, efficient inventory management, and data-driven decision-making. By leveraging SQL queries and views, businesses can:

- Optimize order fulfillment by monitoring processing times.
- Improve product management by tracking top-selling and low-stock items.
- Enhance customer relationships by identifying returning customers and tailoring marketing strategies accordingly.

A well-designed e-commerce database is the backbone of a successful online business. With the right insights, businesses can drive growth, enhance customer experience, and stay ahead in the competitive market.