Project Title: Online Retail Sales Database System
Intern Name: Cherukuri Harshith Sai
Domain: SQL Developer Intern
Date: 26/11/25

1. Introduction

Abstract The objective of this project was to design and implement a robust relational database system for an online retail platform. This system manages critical e-commerce data, including customers, product inventories, orders, and payment transactions. The project simulates a real-world backend environment, demonstrating data normalization, complex retrieval logic, and database automation.

Tools Used

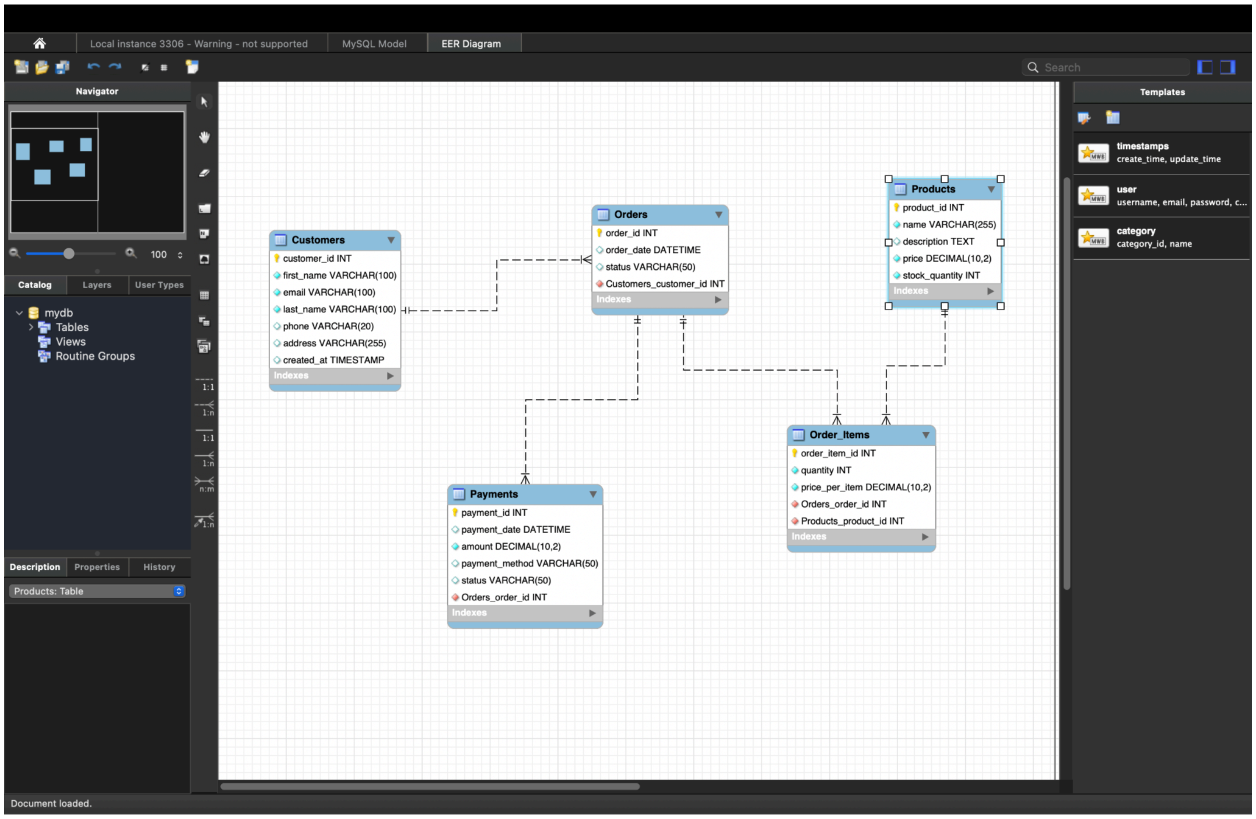
- **Database Engine:** MySQL
- **IDE/GUI:** MySQL Workbench
- **Modeling:** EER Diagramming Tool

2. Database Design

The database schema was designed following 3rd Normal Form (3NF) principles to reduce redundancy and ensure data integrity.

- **Entities Created:**
    - **Customers:** Stores user demographic and contact information.
    - **Products:** Manages inventory, pricing, and stock levels.
    - **Orders:** Acts as the transaction header, linking customers to purchase dates.
    - **Order_Items:** A junction table resolving the Many-to-Many relationship between Orders and Products. This table freezes the `price_per_item` at the time of sale to ensure historical accuracy.
    - **Payments:** Tracks transaction amounts and methods linked to specific orders.

Entity-Relationship Diagram (ERD): The schema utilizes One-to-Many relationships to link entities (e.g., One Customer can have Many Orders).



3. Key Features & Implementation

This project went beyond basic data storage by implementing advanced SQL features for analysis and automation.

- **Advanced Reporting:** utilized `INNER JOIN`s to combine data from four different tables (`Orders`, `Customers`, `Order_Items`, `Products`) to generate comprehensive sales reports.
- **Data Aggregation:** Implemented `GROUP BY` and `SUM()` functions to identify top-spending customers and calculate total revenue.
- **Views:** Created a view named `v_SalesSummary` to simplify complex joins into a single virtual table for management dashboards.
- **Stored Procedures:** Developed `sp_GetCustomerOrders` to accept an email parameter and return a dynamic order history for that specific client.
- **Automation (Trigger):** Implemented an `AFTER INSERT` trigger (`tr_AfterOrderInsert`) on the `Order_Items` table. This trigger automatically deducts the purchased quantity from the `Products` inventory, ensuring stock levels are always accurate in real-time.

4. Project Outcomes

The system was successfully tested with sample data. Below are the results of the core reporting and automation features.

A. Full Sales Report This query retrieves a human-readable list of all items sold, the customer who bought them, and the transaction status.

B. Automation Test (Trigger Implementation) To verify the inventory automation, a test sale was conducted.

- **Before Sale:** Product Stock was checked.
- **Action:** An order for 5 units was placed.
- **After Sale:** The system automatically reduced stock by 5 units without manual intervention.

5. Conclusion

This project successfully demonstrates a full-stack approach to SQL development. From designing a normalized schema to implementing business logic via Stored Procedures and Triggers, the system is efficient, scalable, and data-driven. It highlights the ability to not only store data but to transform it into actionable business insights.