National Economics University

School of Advanced Education Program

E-commerce Order Manager Application

Be Thanh Tien-11247357

Pham Ba Viet-11247371

Hoang Thi Phuong Linh- 11247308

Supervisor: Tran Hung

A report submitted in partial fulfilment of the requirements of
National Economics University for the final exam
in *Faculty of Data Science and Artificial Intelligence*

December 10, 2025

Declaration

We, the undersigned members of the group, confirm that this report entitled **"E-commerce Order Manager Application"** is our own collective work. All figures, tables, code snippets, and other materials are original unless explicitly acknowledged. We understand that plagiarism is a form of academic misconduct and will be penalized accordingly.

We give consent to a copy of our report being shared with future students as an example.

We also give our consent for our work to be made available more widely for educational purposes.

Date: December 10, 2025

Contents

List of Figures

List of Abbreviations

SQL	Structured Query Language – standard language for managing relational databases.
GUI	Graphical User Interface – the visual part of the application users interact with.
ERD	Entity Relationship Diagram – a flowchart that illustrates how entities relate to each other.
UNF	Unnormalized Form – data that contains repeating groups and has not been normalized.
1NF	First Normal Form – a relation where the domain of each attribute contains only atomic values.
2NF	Second Normal Form – a form where all non-key attributes are fully functional dependent on the primary key.
3NF	Third Normal Form – a normal form that removes transitive dependencies.
CRUD	Create, Read, Update, Delete – the four basic operations of persistent storage.
PK	Primary Key – a minimal set of attributes that uniquely specify a tuple.
FK	Foreign Key – a field that refers to the PRIMARY KEY in another table.
KPI	Key Performance Indicator – a measurable value demonstrating how effectively objectives are met.
API	Application Programming Interface.
IDE	Integrated Development Environment.

1

Context and Overview

1.1 Project Context

In the modern retail environment, efficient data management is crucial for business sustainability. The **E-Commerce Manager Pro** is a comprehensive desktop application designed to streamline retail operations. It provides a centralized interface for tracking customers, managing visual product catalogs, and processing orders using a robust relational database backend.

1.2 System Objectives

The primary goals of this project are:

1. **Data Normalization:** To transition from unstructured data (UNF) to a normalized database schema (3NF) to ensure data integrity.
2. **User Experience (UX):** To provide a modern, professional GUI with visual elements (product images, dashboard cards) for better usability.
3. **Business Intelligence:** To offer real-time insights through a dynamic dashboard visualizing KPIs and revenue trends.

1.3 Tech Stack Scope

The application is built using a modular architecture:

- **Language:** Python 3.10+
- **GUI Framework:** Tkinter (Standard Lib) with custom styling.
- **Image Processing: Pillow (PIL)** for rendering product thumbnails and hero images.
- **Database:** MySQL (connected via `mysql-connector-python`).

- **Data Analysis:** Pandas (for report generation).
- **Visualization:** Matplotlib (embedded charts).

The system covers four main functional areas:

- **Dashboard:** Visual overview with KPI Cards and Charts.
- **Visual Catalog:** Grid-based product management with image support.
- **Order Processing:** Transaction management with status tracking.
- **Reporting:** Advanced sales analysis using multi-table joins.

2

Database Design & Normalization

2.1 Data Requirements

The system aims to manage sales activities. The core data entities required are Customers, Products, Orders, and Order Details.

2.2 Normalization Process

The database schema was derived through a rigorous normalization process to eliminate redundancy and ensure data integrity. We will illustrate this process using sample data.

2.2.1 Unnormalized Form (UNF)

Initially, the data can be viewed as a flat "Universal Relation" (like a spreadsheet), where repeating groups exist. A single order contains details of the customer and a list of multiple products bought.

Example Data (UNF):

Order	Date	Customer	Items (Product, Price, Qty)	Total
O001	2025-11-14	Tien (Hanoi)	{Mouse, 150, 2}; {Keybd, 300, 1}	600
O002	2025-10-30	Viet (HCM)	{Monitor, 2500, 1}	2500

Table 2.1: Unnormalized Data with Repeating Groups

Issues:

- **Atomicity Violation:** The "Items" column contains multiple values (lists).
- **Redundancy:** Customer information is repeated if they buy multiple times.

2.2.2 First Normal Form (1NF)

To achieve 1NF, we eliminate repeating groups. Each cell must contain a single (atomic) value. We expand the list so that each product-order combination has its own row.

Resulting 1NF Table:

OrderID	Date	CustID	CustName	ProdID	ProdName	Price
O001	11-14	C041	Tien	P001	Mouse	150
O001	11-14	C041	Tien	P002	Keybd	300
O002	10-30	C006	Viet	P003	Monitor	2500

Table 2.2: First Normal Form (Atomic Values)

Issues: There is significant redundancy. For `O001`, the Date and Customer info are repeated. The Primary Key is a composite of *(OrderID, ProductID)*.

2.2.3 Second Normal Form (2NF)

2NF requires the table to be in 1NF and strictly removes **Partial Dependencies**. Non-key attributes must depend on the *entire* primary key, not just a part of it.

Analysis:

- `Date`, `CustID`, `CustName` depend only on **OrderID** (not ProductID).
- `ProdName`, `StandardPrice` depend only on **ProductID** (not OrderID).
- `Quantity` depends on both (**OrderID** + **ProductID**).

Solution: We decompose the 1NF table into three tables:

1. **Orders Table:** (OrderID, Date, CustID, CustName)
2. **Products Table:** (ProductID, ProdName)
3. **OrderItems Table:** (OrderID, ProductID, Quantity, Price)

2.2.4 Third Normal Form (3NF)

3NF requires removing **Transitive Dependencies**. Non-key attributes should not depend on other non-key attributes.

Analysis in Orders Table:

- We have: `OrderID` $\rightarrow$ `CustID` $\rightarrow$ `CustName`.
- `CustName` depends on `CustID`, which is not the primary key of the Orders table. This is a transitive dependency.

Solution: We move Customer information to a dedicated table.

Final 3NF Schema:

- **Customers:** `CustomerID` (PK), `Name`.
- **Orders:** `OrderID` (PK), `OrderDate`, `Status`, `CustomerID` (FK).
- **Products:** `ProductID` (PK), `Name`, `Price`.
- **OrderItems:** `OrderItemID` (PK), `OrderID` (FK), `ProductID` (FK), `Quantity`, `Price`.

2.3 Entity Relationship Diagram (ERD)

The normalized schema is visualized below, showing the relationships between entities.

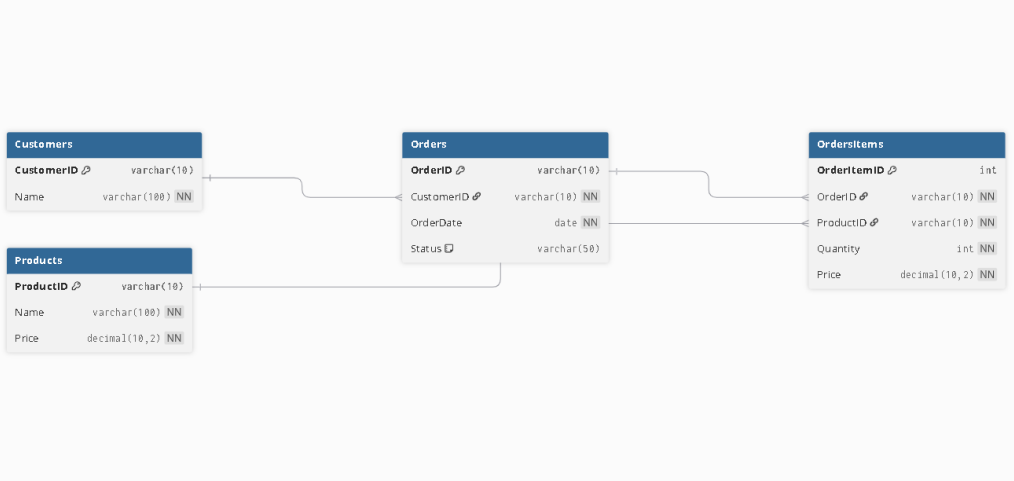


Figure 2.1: Entity Relationship Diagram (ERD)

2.4 Data Dictionary

Detailed specification of the final tables in MySQL:

Table	Column	Type	Description
Customers	CustomerID	VARCHAR(10)	Primary Key
	Name	VARCHAR(100)	Customer Full Name
Products	ProductID	VARCHAR(10)	Primary Key
	Name	VARCHAR(100)	Product Name
	Price	DECIMAL(10,2)	Unit Price
Orders	OrderID	VARCHAR(10)	Primary Key
	CustomerID	VARCHAR(10)	Foreign Key
	OrderDate	DATE	Transaction Date
OrderItems	OrderItemID	INT	Auto-increment PK
	OrderID	VARCHAR(10)	Foreign Key
	ProductID	VARCHAR(10)	Foreign Key
	Price	DECIMAL(10,2)	Snapshot Price

Table 2.3: Database Schema Specification

3

Implementation & Results

3.1 System Architecture

The application follows the MVC (Model-View-Controller) pattern, enhanced with a Service layer for complex data processing.

3.2 Graphical User Interface (GUI)

3.2.1 Main Layout and Welcome Screen

Upon launching, the application presents a clean and professional **Welcome Screen**. The interface utilizes a **minimalist design** approach, featuring clear typography ("Welcome Back, Administrator!") centered on a neutral background. This design choice reduces visual clutter and immediately directs the user's attention to the sidebar navigation.

The application features a robust **Sidebar Navigation** system with "Active State" highlighting, ensuring users always know their current location within the app.

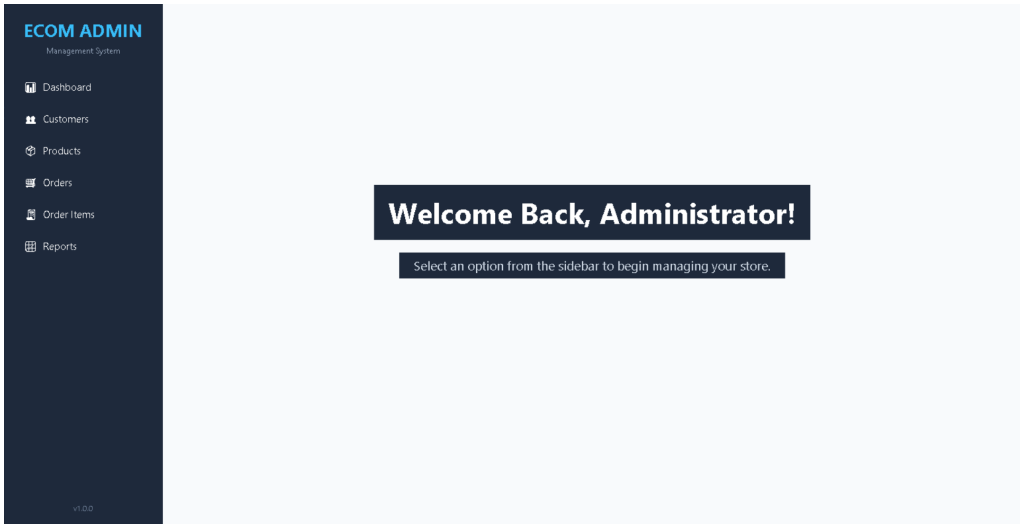


Figure 3.1: Minimalist Welcome Screen and Main Layout

3.2.2 Modern Dashboard

The Dashboard utilizes a "Card UI" design pattern to display key metrics (KPIs) such as Total Revenue and Order Counts. Below the cards, a Matplotlib Bar Chart visualizes the "Top 10 Products by Revenue", integrated seamlessly into the Tkinter window.

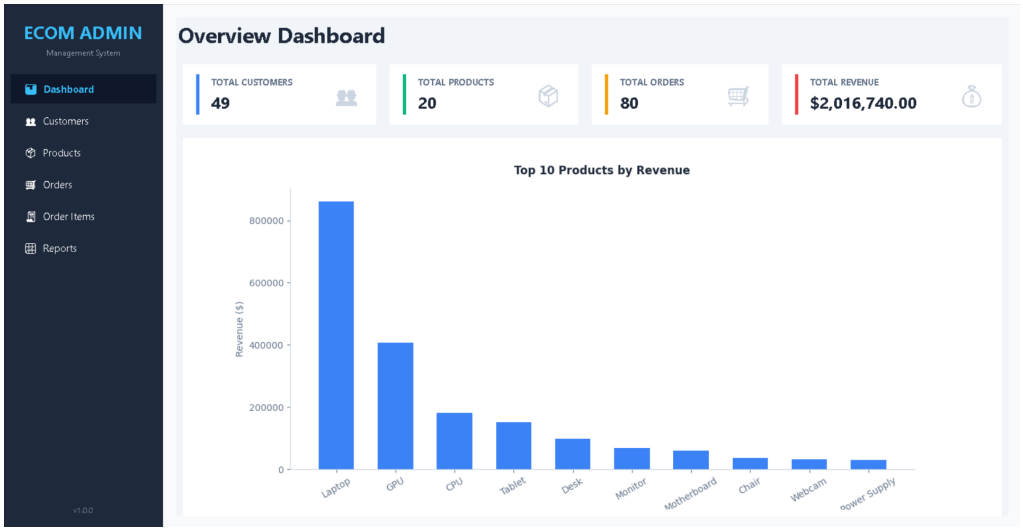


Figure 3.2: Dashboard with KPI Cards and Revenue Chart

3.2.3 Visual Product Catalog (Grid View)

Unlike traditional list views, the Product Manager is implemented as a **Scrollable Grid**.

- **Image Handling:** The system automatically loads product images from the `assets/products/` directory based on the Product ID.
- **Fallback Mechanism:** If an image is missing, a default placeholder is rendered to maintain layout consistency.

- **Card Component:** Each product is displayed in a styled card showing the Image, Name, Price, and ID.

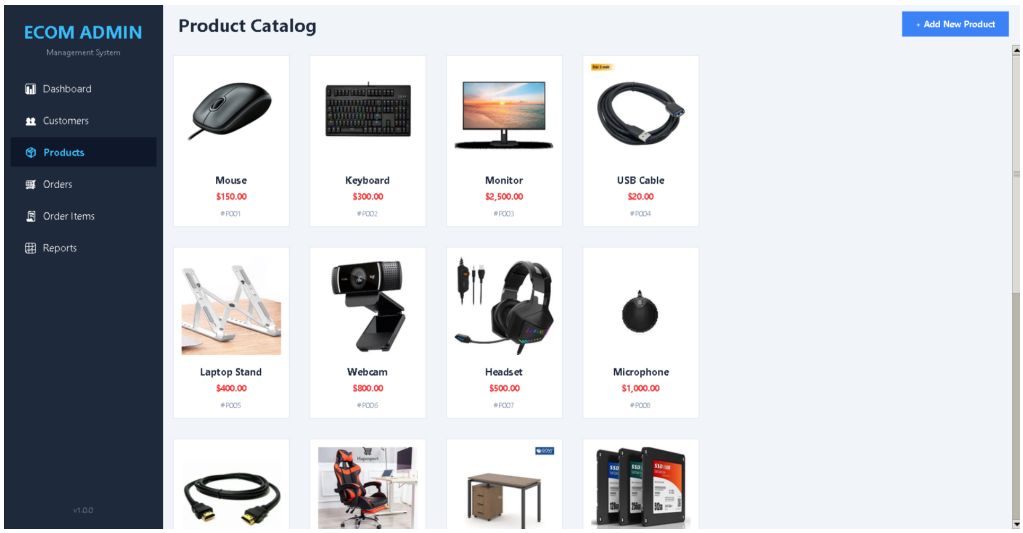


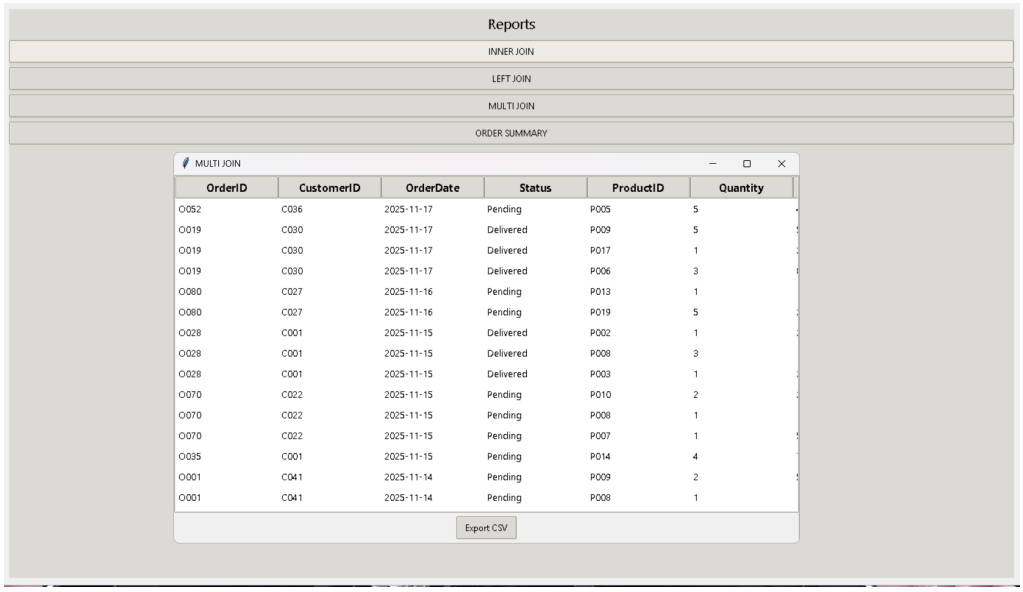
Figure 3.3: Product Catalog with Grid Layout and Images

3.3 Query Logic and Reporting

The application leverages **Pandas** for high-performance data manipulation.

3.3.1 Complex Reporting (Multi-Join)

The system performs an Inner Join across three tables (Orders, Items, Products) to generate a detailed sales report, which can be exported to CSV.



OrderID	CustomerID	OrderDate	Status	ProductID	Quantity
O052	C036	2025-11-17	Pending	P005	5
O019	C030	2025-11-17	Delivered	P009	5
O019	C030	2025-11-17	Delivered	P017	1
O019	C030	2025-11-17	Delivered	P006	3
O080	C027	2025-11-16	Pending	P013	1
O080	C027	2025-11-16	Pending	P019	5
O028	C001	2025-11-15	Delivered	P002	1
O028	C001	2025-11-15	Delivered	P008	3
O028	C001	2025-11-15	Delivered	P003	1
O070	C022	2025-11-15	Pending	P010	2
O070	C022	2025-11-15	Pending	P008	1
O070	C022	2025-11-15	Pending	P007	1
O035	C001	2025-11-15	Pending	P014	4
O001	C041	2025-11-14	Pending	P009	2
O001	C041	2025-11-14	Pending	P008	1

Figure 3.4: Generated Sales Report

4

Testing and Conclusion

4.1 Testing Strategy

To ensure the reliability of the application, we implemented a dual-testing strategy consisting of Automated Unit Testing (using `pytest`) and Manual Functional Testing (focusing on UI and Resources).

4.2 Automated Unit Testing

We utilized the `pytest` framework to test business logic in isolation by mocking database connections using `monkeypatch`. This ensures that tests run fast and do not alter the actual production database.

4.2.1 Revenue Calculation Logic

The file `test_revenue.py` validates the financial core of the application.

- **Mocking Data:** We simulated database responses (e.g., `mock_fetch_order_items`) to return controlled list of items.
- **Test Case 1 (Basic Calculation):** Verified that $Revenue = \sum(Quantity \times Price)$.
- **Test Case 2 (Edge Cases):** Verified system behavior when `Quantity` or `Price` is zero. The system correctly returns 0 revenue without crashing.
- **Test Case 3 (Empty Data):** Verified that an order with no items returns an empty DataFrame structure, preventing UI errors.

4.2.2 Data Integrity & Validation Rules

The file `test_date_status.py` enforces business rules regarding data quality.

- **Status Validation:** We defined a set of valid statuses: `{"Pending", "Completed", "Cancelled"}`. The test `test_status_invalid` confirms that the system flags any unknown status codes.
- **Temporal Consistency:** The test `test_date_not_future` ensures that no orders can be recorded with a future date (e.g., Year 2026), preventing logical anomalies in reporting.

4.3 Manual Functional & UI Testing

In addition to code tests, we performed manual verification on the GUI and resource handling:

1. **Dashboard Rendering:** Confirmed that the Matplotlib chart correctly redraws when new orders are added.
2. **Input Constraints:** Verified that the "Add Order" form rejects invalid date formats (e.g., "31-02-2025") before calling the backend.
3. **Image Fallback (Resource Handling):** We intentionally removed a product image (e.g., `P001.jpg`) from the assets folder to verify that the application correctly loads the `default.png` placeholder instead of crashing or breaking the layout.
4. **Resolution Adaptation:** Verified that the Welcome Screen image resizes correctly (using the `LANCZOS` filter) when the window is resized, ensuring the layout remains responsive on different screen sizes.

4.4 Limitations

Despite the successful implementation, the current version has the following limitations:

- **Single-User Environment:** The application lacks user authentication and role-based access control (RBAC).
- **Local Resource Dependency:** Product images are stored locally in the `assets` folder. In a distributed system, cloud storage (like AWS S3) would be preferred to synchronize images across multiple clients.
- **Concurrency:** The system does not implement record locking. If used in a networked environment, race conditions could occur during simultaneous updates.
- **Static Reporting:** The current reports are static snapshots. There is no feature to filter reports by dynamic date ranges (e.g., "Last 7 days") in the GUI.

4.5 Conclusion

The **E-commerce Order Manager** project successfully demonstrates the integration of a relational database with a Python-based desktop application.

- **Design:** The database achieved 3NF, ensuring data integrity and reducing redundancy.
- **Implementation:** The application provides a modern, user-friendly interface with visual product management and interactive dashboards.
- **Quality Assurance:** The inclusion of automated unit tests ensures the robustness of critical business logic.

This project serves as a solid foundation for a scalable retail management system.

References