National Economics University
Faculty of Data Science and Artificial Intelligence

PROJECT REPORT

Topic: E-commerce Order Manager

Course name: Database Management System

Group Members:

Nguyen Dinh Thang	11247349
Pham Quang Vu	11247372
Bui Tuan Anh	11247255

Supervisor: Dr. Hung Tran

A report submitted in partial fulfillment of the requirements for the Database Management System course final assessment at the Faculty of Data Science and Artificial Intelligence

Hanoi, December 12, 2025

Declaration

I, **Pham Quang Vu**, on behalf of my group members, confirm that this is our own work. All of the figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced.

We understand that failing to do so will be considered as a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalized accordingly.

I give my consent to having a copy of my report shared with future students as an example.

December 12, 2025

Pham Quang Vu,
Nguyen Dinh Thang,
Bui Tuan Anh

Abstract

This project develops a fully functional E-commerce Order Management System using MySQL and Python. The initial unnormalized dataset is decomposed step by step into 1NF, 2NF, and 3NF, resulting in a clean relational schema consisting of Customers, Products, Orders, and OrderItems. A graphical application (Tkinter) enables CRUD operations, search functionality, and automated report generation through SQL views and stored procedures. The project demonstrates normalization, data integrity, and practical DB–app integration.

Contents

List of Figures

List of Tables

List of Abbreviations

UNF	Unormalized Form
1NF	First Normalized Form
2NF	Second Normalized Form
3NF	Third Normalized Form
CRUD	Create, Read, Update, Delete
PK	Primary Key
FK	Foreign Key
ERD	Entity-Relationship Diagram
RS	Relational Schema
DDL	Data Definition Language
RegExp	Regular Expression
GUI	Graphic User Interface

Chapter 1

Introduction

1.1 Project Context

In the rapidly expanding domain of electronic commerce, efficient order management systems have become indispensable for business success. E-commerce platforms handle thousands of transactions daily, involving complex interactions between customers, products, and orders. Manual management of these transactions is not only time-consuming but also prone to errors that can lead to customer dissatisfaction, inventory discrepancies, and financial losses.

The foundation of this project began with an unnormalized table containing interrelated data about e-commerce transactions. This initial table combined multiple entities—customer information, product details, order data, and order items—into a single structure, leading to significant data redundancy and update anomalies. For instance, a single customer's information would be repeated across multiple orders, and product details would be duplicated in every order containing that product.
This unnormalized design presented several critical issues:

- **Data Redundancy**: Customer and product information repeated across multiple rows
- **Update Anomalies**: Changing a customer's name required updating multiple rows
- **Insertion Anomalies**: New products couldn't be added without an associated order
- **Deletion Anomalies**: Deleting an order could unintentionally remove product information

The need for normalization became apparent to eliminate these anomalies and create a robust, maintainable database structure. This project addresses these challenges by developing a comprehensive E-commerce Order Management System that supports staff in efficiently managing customers, products, orders, and order items through an intuitive graphical interface.

1.2 Objectives

The primary objectives of this project are:

1.2.1 Database Design and Normalization

- Analyze the initial unnormalized table and identify functional dependencies
- Normalize the database structure to Third Normal Form (3NF) to eliminate data redundancy and anomalies

- Design a relational schema with appropriate tables, relationships, and constraints

1.2.2 Database Implementation

- Implement the normalized database schema in MySQL with proper table definitions
- Define and enforce data integrity constraints including primary keys, foreign keys, and check constraints
- Create stored procedures to encapsulate business logic and ensure transactional integrity
- Develop comprehensive SQL views for reporting and business intelligence

1.2.3 Application Development

- Build a graphical user interface (GUI) application using Python and Tkinter
- Implement complete CRUD (Create, Read, Update, Delete) operations for all entities:
  - Customers: Add, update, delete, and search customer information
  - Products: Manage product catalog with pricing and inventory considerations
  - Orders: Create and track orders with status management
  - Order Items: Add products to orders with quantity tracking
- Develop user-friendly interfaces with proper validation and error handling

1.2.4 Reporting and Analytics

- Implement SQL views for comprehensive reporting:
  - Order details with customer and product information
  - Customer order history
  - Revenue analysis by date and product
  - Order status distribution
- Create data export functionality for external analysis
- Develop a dashboard interface for key performance indicators (KPIs)

1.2.5 System Integration and Testing

- Ensure seamless integration between the GUI application and MySQL database
- Implement robust error handling and data validation at multiple levels
- Conduct comprehensive testing of all functionalities and edge cases
- Document the system architecture, implementation, and usage guidelines

1.3 Scope

1.3.1 In-Scope Items

The following functionalities and features are included in this project:

- **Database Design**: Complete normalization to 3NF with four main tables (customers, products, orders, orderitems) and appropriate relationships
- **Data Management**: Full CRUD operations for all entities through the GUI interface
- **Data Validation**: Comprehensive input validation at both application and database levels
- **Business Logic**: Implementation of stored procedures for critical operations including:
  - Adding customers, products, orders, and order items
  - Deleting records with proper cascading operations
  - Searching customers and products with flexible criteria
  - Updating product prices and order statuses
- **Reporting**: Five predefined SQL views for business intelligence:
  - Order details view
  - Customer orders view
  - Revenue by date view
  - Products revenue summary view
  - Orders by status view
- **Data Export**: CSV export functionality for all views and table data
- **User Interface**: Intuitive GUI with categorized operations and responsive design
- **Error Handling**: Comprehensive error messages and validation feedback

1.3.2 Out-of-Scope Items

The following items are explicitly excluded from this project:

- **User Authentication**: No login system or user account management
- **Role-Based Access Control**: All users have equal access to all functions
- **Web Interface**: The application is desktop-based, not web-accessible
- **Real-time Inventory Management**: Basic product tracking without automated stock level updates
- **Payment Processing Integration**: No integration with payment gateways or financial systems
- **Shipping Integration**: No integration with shipping carriers or tracking systems
- **Customer-Facing Interface**: The system is designed for staff use only

- **Mobile Application**: No mobile version or responsive design for mobile devices
- **Advanced Analytics**: Basic reporting without predictive analytics or machine learning
- **Multi-language Support**: Interface available in English only
- **Data Backup and Recovery**: Basic database functionality without automated backup systems

1.3.3 Project Boundaries

This project focuses specifically on the core order management functionality required for a small to medium e-commerce operation. The system is designed as a proof-of-concept and educational demonstration of database application development principles, rather than a production-ready enterprise solution. While comprehensive in its core functionality, the system acknowledges practical constraints and focuses on demonstrating fundamental database concepts, application development techniques, and system integration principles.

The successful implementation of this project provides a solid foundation that could be extended with additional features for production deployment, but within the current scope, emphasis remains on demonstrating proper database design, application architecture, and user interface development practices.

Chapter 2

Normalization Process

2.1 Unormalized Form Table

Below is the original Unnormalized Form (UNF) table structure along with its attributes.

UNF Columns
OrderID
CustomerID
CustomerName
ProductID
ProductName
Quantity
Price
OrderDate
Status

Table 2.1: UNF Database Schema

In UNF, a single order may contain multiple products, causing repeating groups and non-atomic values within the same logical row.

2.2 Functional Dependencies

Based on the business rules and the e-commerce workflow, the following functional dependencies (FDs) are identified:

- CustomerID $\rightarrow$ CustomerName
- ProductID $\rightarrow$ {ProductName, Price}
- OrderID $\rightarrow$ CustomerID, OrderDate, Status
- {OrderID, ProductID} $\rightarrow$ Quantity

2.3 Unormalized Form to First Normal Form

First Normal Form (1NF) requires that all attributes contain atomic values and that no repeating groups appear within a relation.

To achieve 1NF, we convert each product within an order into a separate row. The resulting intermediate 1NF structure is shown below as a single table: `OrderItems`.

OrderItems
OrderID
CustomerID
CustomerName
ProductID
ProductName
Quantity
Price
OrderDate
Status

Table 2.2: 1NF Database Schema

At this stage, the logical primary key becomes the composite key **(OrderID, ProductID)**, since the same order can include multiple products.

2.4 First Normal Form to Second Normal Form

Second Normal Form (2NF) eliminates partial dependencies in tables whose primary key is composite. A relation is in 2NF if it is already in 1NF and every non-key attribute depends on the whole primary key.

In the 1NF table, several attributes depend on only part of the composite key:

- Attributes that depend on **OrderID** only: CustomerID, CustomerName, OrderDate, Status
- Attributes that depend on **ProductID** only: ProductName, Price

These are partial dependencies and must be removed. To achieve 2NF, we decompose the 1NF table into four relations based on the identified functional dependencies.

Additionally, we introduce **UnitPrice** in the `OrderItems` table to preserve the historical selling price at the time of the transaction. This addition does not violate 2NF because UnitPrice still depends on the full composite key {OrderID, ProductID}.

2.5 Second Normal Form to Third Normal Form

Third Normal Form (3NF) requires that all non-key attributes depend directly on the primary key, with no transitive dependencies.

From the 2NF schema above, we check for any attribute that depends on another non-key attribute:

Entity	Attributes			
Customers	CustomerID (PK)	CustomerName		
`Products`	ProductID (PK)	ProductName	Price	
`Orders`	OrderID (PK)	OrderDate	Status	CustomerID (FK)
`OrderItems`	OrderID (PK, FK)	ProductID (PK, FK)	Quantity	UnitPrice

Table 2.3: 2NF Database Schema

- In `Customers`, CustomerName depends only on CustomerID.
- In `Products`, ProductName and Price depend only on ProductID.
- In `Orders`, OrderDate, Status, and CustomerID depend only on OrderID.
- In `OrderItems`, Quantity and UnitPrice depend on the composite key (OrderID, ProductID), and no attribute depends on another non-key attribute.

Since no non-key attribute determines another non-key attribute in any of the tables, the schema satisfies 3NF.

2.6 Final Schema

Entity	Attributes			
Customers	CustomerID (PK)	CustomerName		
`Products`	ProductID (PK)	ProductName	Price	
`Orders`	OrderID (PK)	OrderDate	Status	CustomerID (FK)
`OrderItems`	OrderID (PK, FK)	ProductID (PK, FK)	Quantity	UnitPrice

Table 2.4: Final Database Schema

The final schema fully satisfies 3NF: every non-key attribute depends solely on its relation's primary key, and no transitive dependencies exist. Both `schema.sql` and `seed.sql` implement this model, with the former defining the relational structure and constraints, and the latter providing sample data for testing the system.

Chapter 3

Database Implementation

3.1 Entity-Relationship Diagram

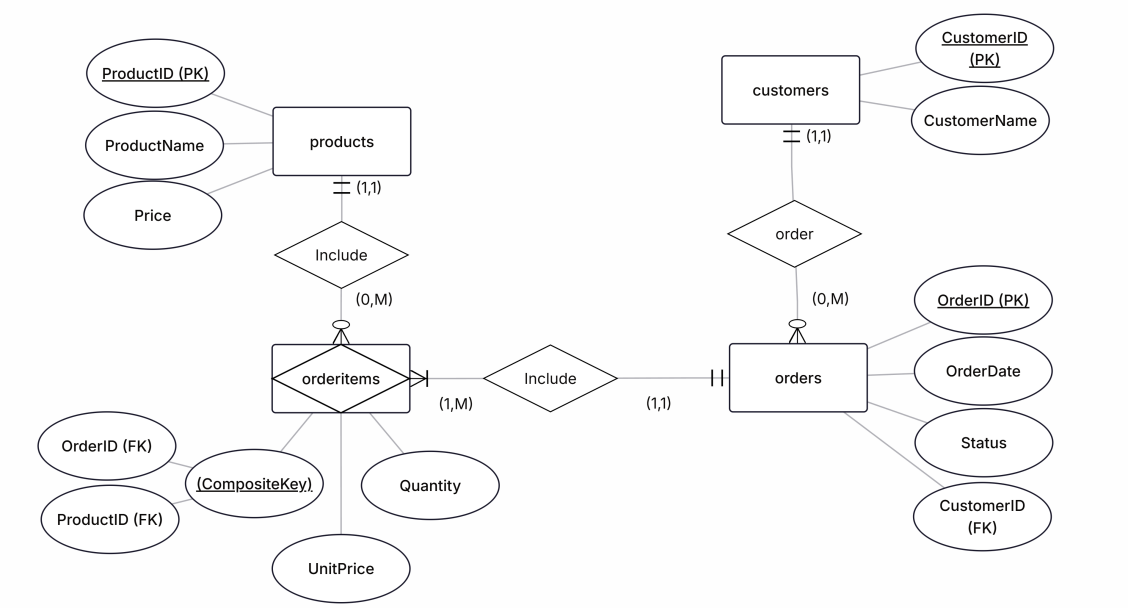

Figure 3.1: Entity–Relationship Diagram (ERD)

It's clear that from this ERD, each customer can place zero or many orders (1–to–M), while each order must belong to exactly one customer (M–to–1). Similarly, each product can appear in multiple order records, and each order can contain multiple products, forming an M–to–M relationship. This many-to-many relationship is resolved through the `OrderItems` table, which acts as an **associative entity** (also known as a bridge table) that links **`Orders`** and **`Products`**. Its composite primary key (OrderID, ProductID) ensures that each product appears only once per order. Cardinalities in the ERD—such as (1,1), (0,M), and (1,M)—express the minimum and maximum participation constraints between entities, accurately modeling real-world business rules for ordering, purchasing, and inventory tracking.

3.2 Relational Schema

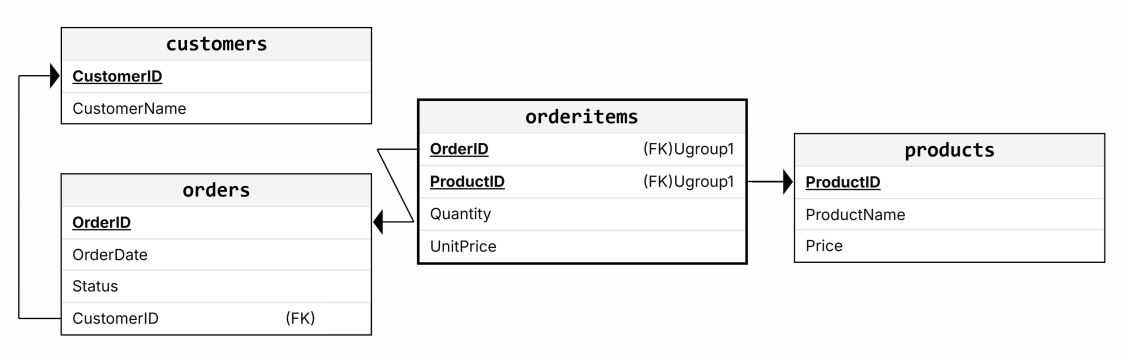

Figure 3.2: Relational Schema

The Relational Schema (RS) shown above is the tabular representation of the entities derived from the ERD above. It highlights the primary keys, foreign keys, and structural relationships between tables. Since the conceptual relationships and cardinalities have already been explained in the ERD section, the RS mainly serves to confirm the finalized logical design, showing how each entity is implemented as a relational table in the database.

3.3 DDL Explanation

Using **(schema.sql)**, this section summarizes the Data Definition Language (DDL) elements used to implement the database.

3.3.1 Primary Keys and Foreign Keys

Each table defines a Primary Key (PK), which uniquely identifies its records:

- In `Customers`, CustomerID is the PK.
- In `Products`, ProductID is the PK.
- In `Orders`:
  - OrderID is the PK.
  - CustomerID is the FK referencing `Customers`
- In `OrderItems`: Composite PK: (OrderID, ProductID). Both attributes reference `Orders` and `Products`, respectively.

3.3.2 CHECK Constraints

The schema uses several CHECK constraints to enforce business rules at the database level:

- **Order status constraint** ensures Status is one of the allowed values: 'pending', 'processing', 'shipped', or 'delivered' (from the `Orders` table).
- **Quantity constraint** ensures that ordered quantities are always positive in `OrderItems`.
- **Price constraint** enforces positive pricing for product prices and unit prices.

These constraints ensure the validity and meaningfulness of data in the transactional tables.

3.3.3 Format Constraints (RegExp)

The DDL enforces strict formatting using regular expressions:

- **CustomerID format** is an 8-character string, beginning with a letter **C**, followed by exactly 7 digits.
- **ProductID format** is the same as above, but using the letter **P** instead.
- **OrderID format** is the same as above, but using the letter **O** instead.

These RegExp patterns help enforce consistent identifier formats across all tables.

3.3.4 Views for Reporting

The schema defines several views to simplify data retrieval and support analytics:

- **order_details:** Joins `Orders`, `Customers`, `OrderItems`, and `Products` to show line-level order information, including calculated total price per line.
- **customer_orders:** Lists all customers and their orders, using a LEFT JOIN to include customers with no orders.
- **orders_by_status:** Summarizes order counts by status category.
- **products_revenue_summary:** Aggregates total quantity sold and revenue per product, then sorts records in descending order, useful for KPI and sales analysis.
- **revenue_by_date:** Counts orders and aggregates revenue on a single day, then sorts records by date in descending order.

3.3.5 Stored Procedures for CRUD

The schema includes a full set of stored procedures to support Create, Read, Update, and Delete (CRUD) operations, encapsulating business logic inside the database layer.

Create (INSERT/ADD) procedures

- add_customer
- add_product
- add_order (by status, default status is **pending**)
- add_orderitem (automatically fetches price and stores it as UnitPrice)

Read (SEARCH) procedures

- search_customer (by name)
- search_product (by name)
- search_orderdetail (full order breakdown)
- search_customer_orders (all orders for a given customer)

Update procedures

- update_status (order status)
- update_price (product price)

Remove (DELETE) procedures

- delete_customer (cascade-like deletion: `OrderItems` → `Orders` → `Customers`)
- delete_product (cascade-like deletion: `OrderItems` → `Products`)
- delete_order (by ID)
- delete_orderitem (by ID)

These procedures standardize database interactions, reduce repetitive SQL code, and enforce consistent rules when modifying data.

3.4 Seed Data Summary

The **seed.sql** file provides sample data for populating the database. It consists of:

- **52 customers**, each with a unique CustomerID following the required **C0000000** format.
- **33 products**, each with a unique ProductID following the required **P0000000** format, along with its price.
- **52 orders**, each with a unique OrderID following the required **O0000000** format, and links to a valid CustomerID.
- **114 order items**, each with a composite key made of OrderID and ProductID.

This ensures diversity results when we run the Python GUI (which will be introduced later).

Chapter 4

Application Design

This chapter details the implementation of the E-commerce Order Manager, a comprehensive desktop application built using Python's Tkinter library for the graphical user interface (GUI) and MySQL as the backend database. The application follows a modular architecture that separates database operations, business logic, and user interface components for maintainability and scalability.

4.1 System Architecture

The system employs a three-tier architecture consisting of:

1. **Presentation Layer (GUI)**: Built with Tkinter, providing an intuitive interface for users to interact with the system.
2. **Business Logic Layer (Python Modules)**: Handles data validation, business rules, and coordinates between the GUI and database.
3. **Data Access Layer (MySQL)**: Stores and manages all application data through structured tables, views, and stored procedures.

4.1.1 Python-MySQL Communication

The application establishes database connectivity through the `Connection` class in `connection.py`, which utilizes the `pymysql` library. Key aspects of this communication include:

- Environment-based configuration using `dotenv` for database credentials
- Connection pooling through reusable `Connection` instances
- Parameterized queries to prevent SQL injection attacks
- Comprehensive error handling for database operations

The data flow follows this pattern:

`GUI → Business Logic (Python Classes) → Stored Procedures → MySQL Database`

This architecture ensures that database logic is encapsulated within stored procedures, providing better security and performance.

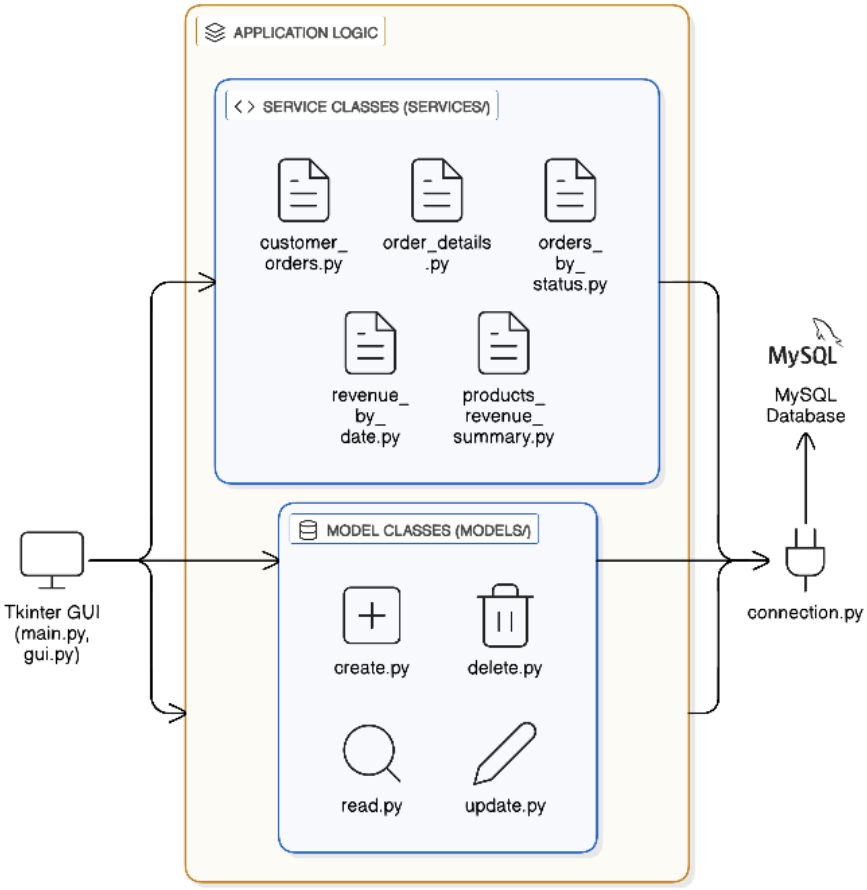

Figure 4.1: System Architecture Diagram

4.2 GUI Design and Implementation

The GUI is designed with user-friendliness and efficiency in mind, featuring a clean layout with a persistent main menu and dynamic content area. The main window (Figure 4.2) is divided into two primary sections: a left-side menu panel containing seven core operational categories, and a right-side details panel that dynamically updates based on user selection.

4.2.1 Main Application Interface

The main interface (Figure 4.2) presents a professional blue-themed design with the following key components:

- **Application Title Bar**: Displays "E-commerce Order Manager v1.0" in prominent typography
- **Primary Menu Panel**: Located on the left, contains seven functional buttons in a vertical layout
- **Details Display Panel**: Occupies the right side, showing data tables, input forms, or operation results
- **Responsive Design**: Adapts to window resizing with scrollable content areas for data tables

There are 8 main menu options:

1. **Add**: For creating new records (customers, products, orders, order items)
2. **Update**: For modifying existing records (product prices, order statuses)
3. **Remove**: For deleting records (customers, products, orders)
4. **Search**: For finding specific records via various criteria
5. **Show All**: For displaying complete tables (customers, products, orders)
6. **Views**: For accessing analytical reports and summaries
7. **Export CSV**: For exporting displayed data to comma-separated value files
8. **Exit**: Immediately close the GUI

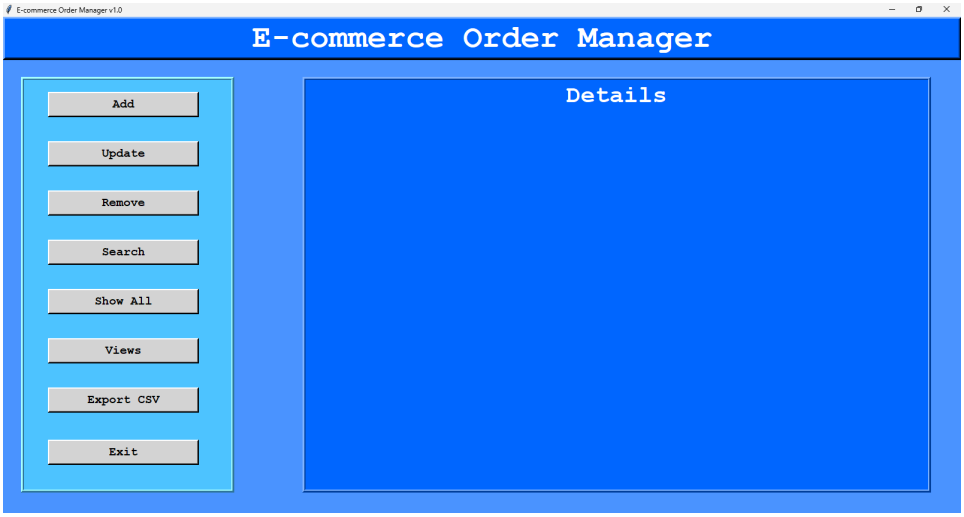

Figure 4.2: Main application interface with seven primary menu options

4.2.2 Functional Modules

Add Module

When the "Add" button is selected, a secondary menu appears (Figure 4.3) offering four creation options:

- **Add Customer**: For registering new customers with ID validation (C0000000 format)
- **Add Product**: For adding products to inventory with price validation
- **Add Order**: For creating new orders with automatic date assignment
- **Add Ordered Products**: For adding items to existing orders

Each option presents a dedicated input form with field validation and immediate feedback.

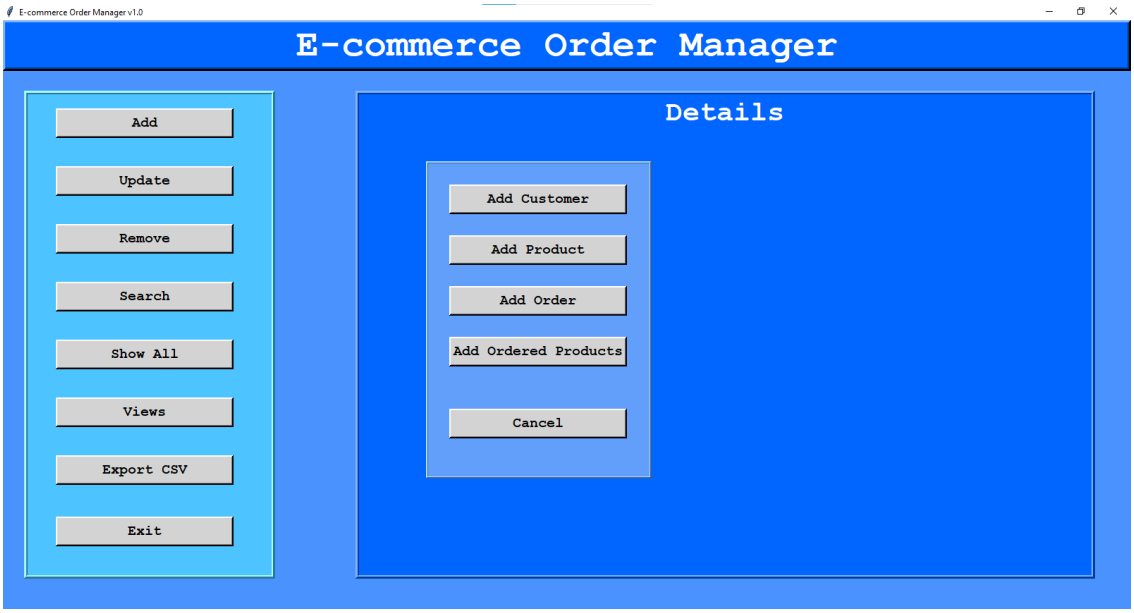

Figure 4.3: Add module interface showing the four creation options

Update Module

The update module (Figure 4.4) provides two modification functions:

- **Product's Price**: Allows price updates for existing products with decimal validation
- **Order's Status**: Enables status changes through predefined states (pending, processing, shipped, delivered)

Both functions require valid ID input and provide success/error confirmation messages.

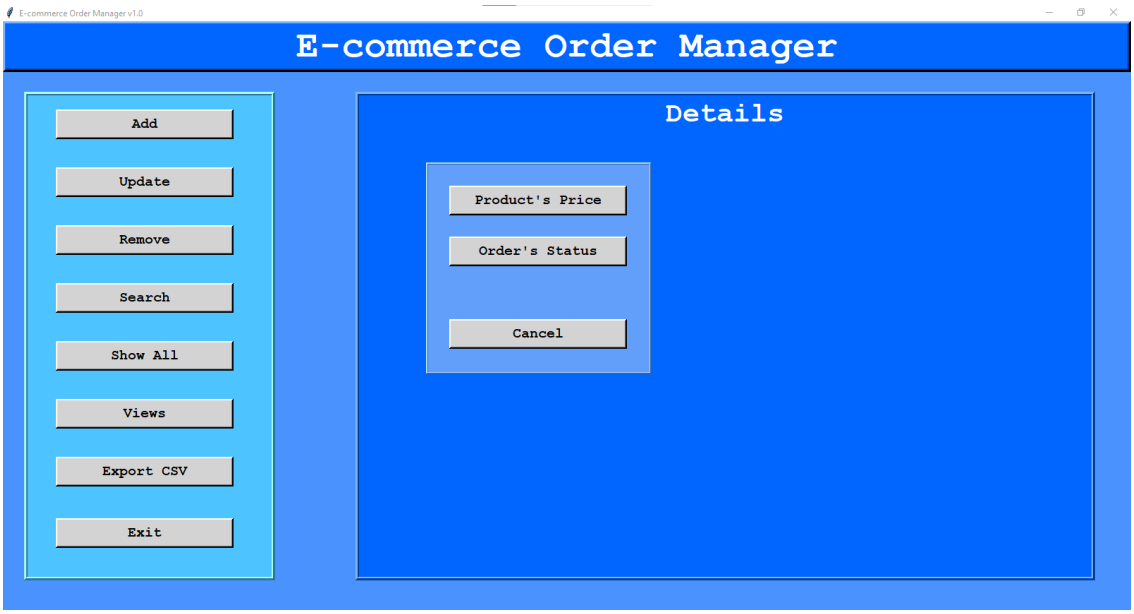

Figure 4.4: Update module with product price and order status options

Remove Module

The removal interface (Figure 4.5) includes three deletion options with cascading effects:

- **Remove Customer**: Deletes customer and all associated orders (with confirmation)
- **Remove Product**: Removes product from inventory and all order items
- **Remove Order**: Deletes order and all associated order items

Safety measures include ID validation and confirmation prompts to prevent accidental deletions.

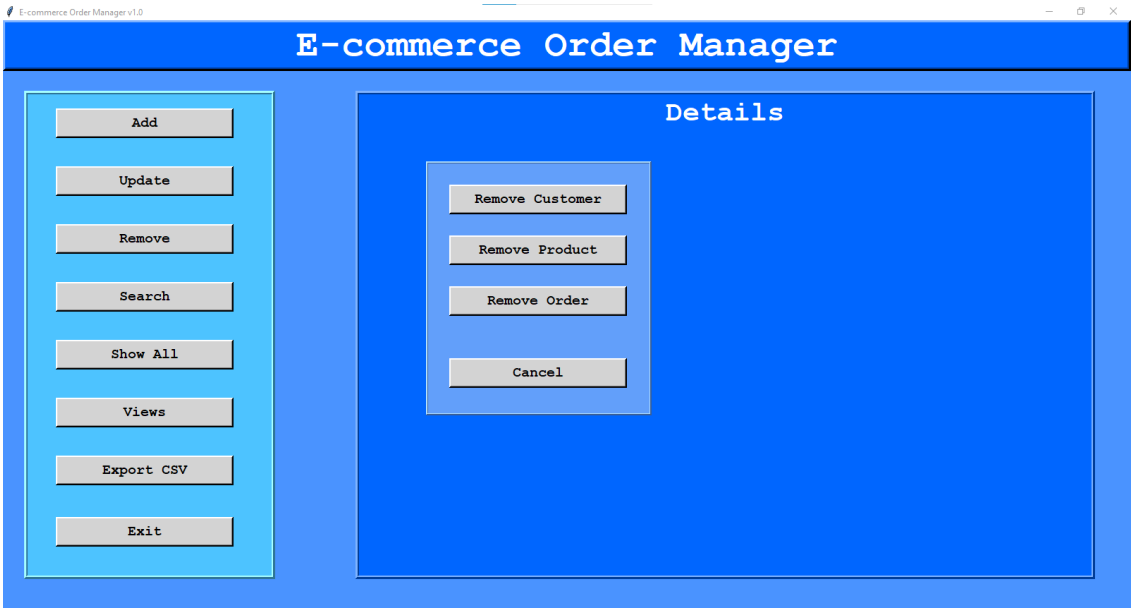

Figure 4.5: Remove module with customer, product, and order deletion options

Search Module

The search functionality (Figure 4.6) offers four targeted search types:

- **Search Customer**: Finds customers by name using partial matching
- **Search Product**: Locates products by name with wildcard support
- **Search Order**: Retrieves order details by order ID
- **Search Customer Orders**: Finds all orders for a specific customer

Results are displayed in tabular format within the details panel.

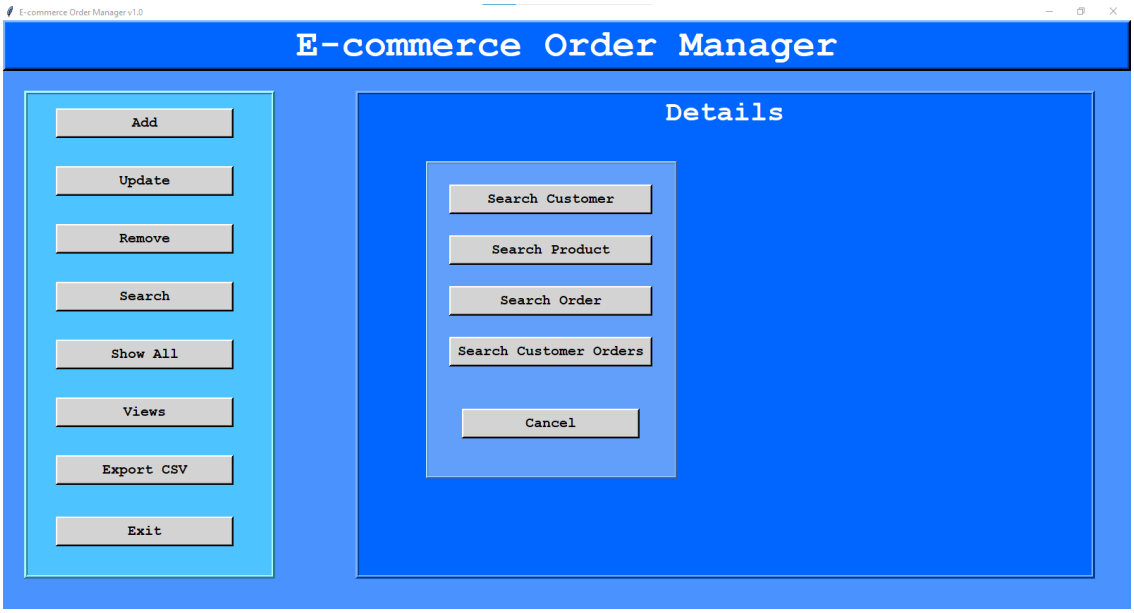

Figure 4.6: Search module with four different search criteria options

Show All Module

This module (Figure 4.7) provides complete table views:

- **All Customers**: Displays the entire customer table with ID and name columns
- **All Products**: Shows complete product catalog with prices
- **All Orders**: Presents all orders with dates, statuses, and customer references

Tables include horizontal and vertical scrolling for large datasets.

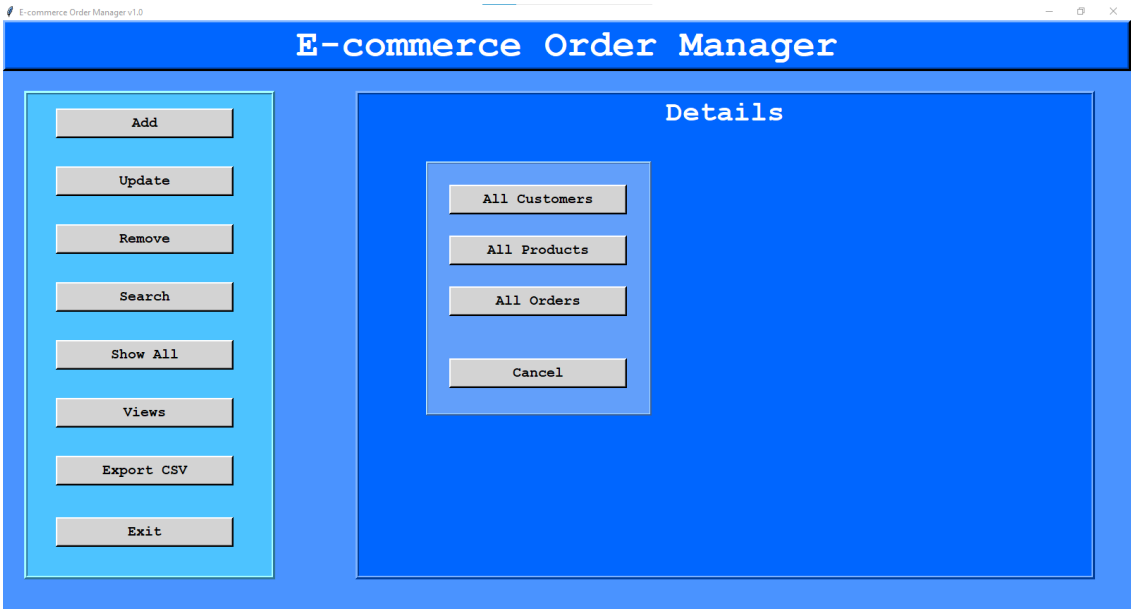

Figure 4.7: Show All module options for complete table displays

Views Module

The analytical views section (Figure 4.8) offers five pre-defined reports:

- **Order Details**: Comprehensive view linking orders, customers, and products
- **Customer Orders**: Summary of all orders grouped by customer
- **Revenue by Dates**: Daily revenue analysis with order counts
- **Revenue by Products**: Product performance metrics with quantities sold
- **Order Status Count**: Distribution of orders across different statuses

These views utilize MySQL database views for optimized query performance.

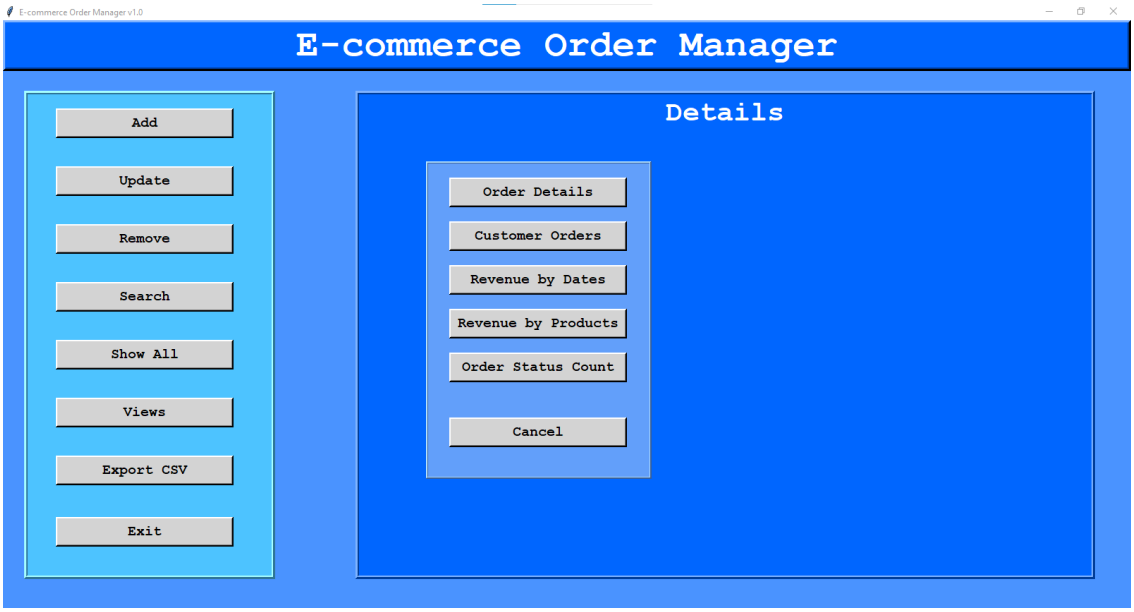

Figure 4.8: Views module showing five analytical report options

Export CSV Module

The export feature allows any displayed table (by using functions in the Show All module or Views module) to be saved as a CSV file. When activated, it:

- Automatically names files based on the current view (e.g., "all_customers.csv")
- Saves files to a dedicated directory: "csv/"
- Provides success/error notifications for the export operation

This functionality enables easy data sharing and offline analysis.

4.2.3 Data Display and Interaction

All data presentation follows consistent design principles:

- **Tabular Display**: Data is shown in scrollable treeview tables with clear column headers

- **Form Validation**: All input fields validate data format before submission
- **Real-time Feedback**: Immediate success/error messages for user actions
- **Context Preservation**: The main menu remains accessible during all operations
- **Cancel Options**: All secondary operations include cancellation buttons

The interface ensures that users can easily navigate between different functions without losing context, with the active operation always clearly indicated in the details panel.

4.3 Database Integration

The application leverages MySQL's advanced features for robust data management:

4.3.1 Stored Procedures

All CRUD operations are implemented as stored procedures, providing:

- Transaction management for data integrity
- Reduced network traffic through batch operations
- Enhanced security through parameterized execution
- Business logic encapsulation at the database level

4.3.2 Database Views

Several analytical views are created for reporting:

- `products_revenue_summary`: Product performance metrics
- `revenue_by_date`: Daily revenue tracking
- `customer_orders`: Customer order history
- `orders_by_status`: Order status distribution

4.4 Code Organization

The application follows a modular structure:

```
ecommerce-order-manager
    app
        csv/                    # to store output .csv files.

        db/
            .env                # Store connection data.
            connection.py       # Python's database connector (PyMySQL).
            schema.sql          # Database's structure.
            procedures.sql      # Database's stored procedures.
            seed.sql            # Sample data.
```

```
models/                              # CRUD; data access
    create.py
    read.py
    update.py
    delete.py

queries/                             # Resuable SQL views for reports
    customer_orders.sql
    order_details.sql
    orders_by_status.sql
    products_revenue_summary.sql
    revenue_by_date.sql

services/                            # Business logic
    customer_orders.py
    order_details.py
    orders_by_status.py
    products_revenue_summary.py
    revenue_by_date.py

ui/
    gui.py                           # GUI source

main.py                              # Entry point to run the program.
```

4.5 Error Handling and Validation

The application implements comprehensive error handling:

- Input validation for all user entries
- Database constraint enforcement
- Graceful error messages for user feedback
- Transaction rollback on failures
- Connection timeout management

4.6 Export Functionality

The CSV export feature allows data extraction in multiple formats:

- Complete table exports (Customers, Products, Orders)
- Custom report exports (Revenue, Order Status)
- Automated file naming with timestamps
- Error handling for file system operations

This chapter has detailed the design and implementation of the E-commerce Order Manager application, highlighting its modular architecture, comprehensive feature set, and robust database integration. The following chapter will discuss testing methodologies and performance evaluation.

Chapter 5

Functional Features

5.1 CRUD Features

The E-commerce Order Manager application provides comprehensive CRUD (Create, Read, Update, Delete) functionality for all major entities in the system. These operations are implemented through stored procedures in the database and accessed via the Python GUI.

5.1.1 Customer Management

Adding Customers

Customers can be added through the GUI by selecting the "Add" option and then "Add Customer". The system requires:

- **Customer ID**: Must follow format "C" + 7 digits (e.g., C0000053)
- **Customer Name**: Full name of the customer

The application calls the `add_customer()` stored procedure, which validates the ID format and inserts the record into the customers table.

Updating Customers

Customer information can be updated directly through the database, though the current GUI focuses on product and order updates. The system maintains referential integrity through foreign key constraints.

Deleting Customers

Customers are removed via the "Remove Customer" option. The `delete_customer()` stored procedure:

- Deletes all order items associated with the customer's orders
- Deletes all orders placed by the customer
- Finally deletes the customer record

This cascade deletion ensures database integrity.

Searching Customers

Customers can be searched by name using partial matching. The `search_customer()` procedure returns all customers whose names contain the search string, supporting case-insensitive searches.

5.1.2 Product Management

Adding Products

Products are added with the following requirements:

- **Product ID**: Format "P" + 7 digits (e.g., P0000034)
- **Product Name**: Name of the product
- **Price**: Positive decimal value with two decimal places

The `add_product()` procedure validates all inputs before insertion.

Updating Products

Product prices can be updated via the "Update Product's Price" feature. The `update_price()` procedure modifies the price while maintaining referential integrity.

Deleting Products

Product deletion uses the `delete_product()` procedure which:

- Removes the product from all order items
- Deletes the product record from the products table

Searching Products

Products can be searched by name with partial matching through the `search_product()` procedure.

5.1.3 Order Management

Adding Orders

New orders are created with:

- **Order ID**: Format "O" + 7 digits (e.g., O0000053)
- **Customer ID**: Valid existing customer ID

The `add_order()` procedure automatically sets the order date to current date and status to "pending".

Updating Orders

Order status can be updated through the "Update Order's Status" feature. Valid statuses are: pending, processing, shipped, and delivered. The `update_status()` procedure ensures only valid status transitions.

Deleting Orders

Order deletion via `delete_order()` removes:

- All order items associated with the order
- The order record itself

Searching Orders

Orders can be searched by:

- Order ID (returns detailed order information)
- Customer ID (returns all orders for a specific customer)

The `search_orderdetail()` and `search_customer_orders()` procedures provide these functionalities.

5.1.4 Order Items Management

Adding Order Items

Products are added to orders using the "Add Ordered Products" feature, requiring:

- **Order ID**: Existing order ID
- **Product ID**: Existing product ID
- **Quantity**: Positive integer

The `add_orderitem()` procedure calculates the unit price from the products table at the time of addition.

Deleting Order Items

Specific products can be removed from orders using the `delete_orderitem()` procedure, though this feature is not exposed in the current GUI interface.

5.2 Reports & Views

The application provides several pre-defined views for business intelligence and reporting purposes.

5.2.1 Order Details View

The `order_details` view combines information from orders, order items, customers, and products to provide comprehensive order information.

Figure 5.1: Order Details View showing comprehensive order information

This view displays:

- Order identification and date
- Customer information
- Product details and quantities
- Calculated total prices
- Current order status

5.2.2 Customer Orders View

The `customer_orders` view shows all customers and their associated orders using a LEFT JOIN to include customers without orders. Its key features include:

- Lists all customers regardless of order history
- Shows order details for customers with orders
- Helps identify inactive customers

Figure 5.2: Customer Orders View showing customer-order relationships

5.2.3 Products Revenue Summary View

This KPI view calculates revenue generated by each product.

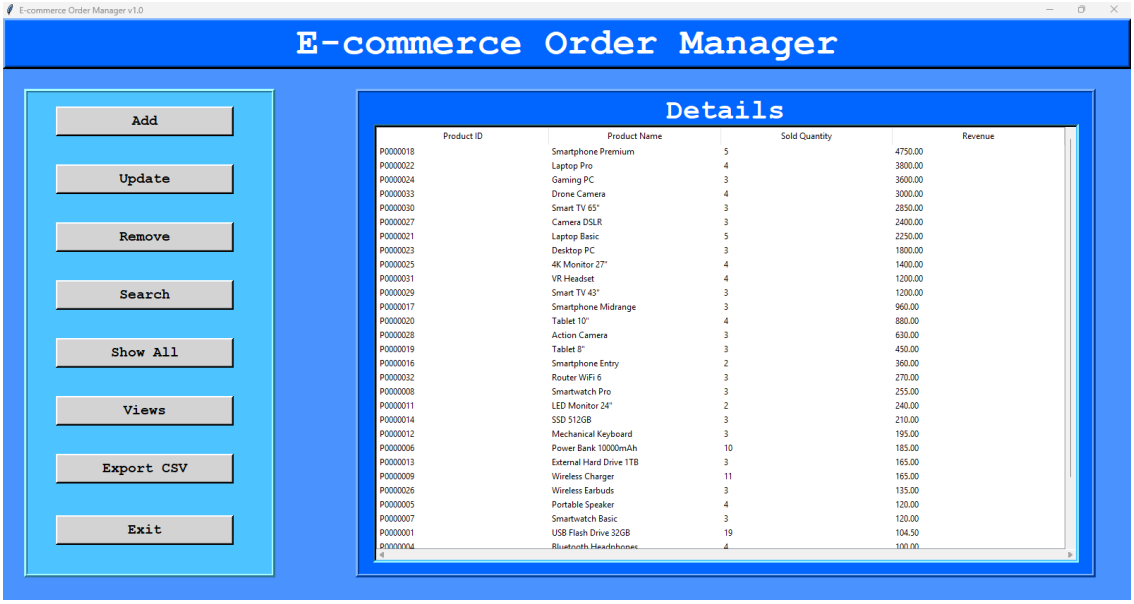

Figure 5.3: Product Revenue Summary

The view provides:

- Total quantity sold per product
- Total revenue generated per product
- Automatic sorting by revenue (descending)
- Insights into product performance

5.2.4 Orders by Status View

This view provides a summary of order status distribution.

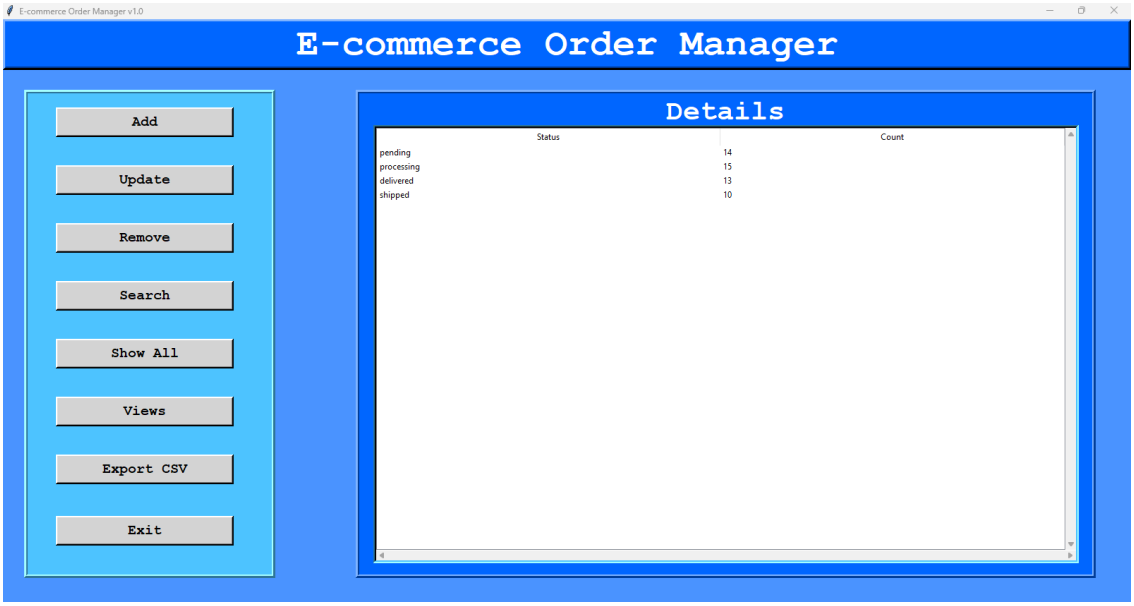

Figure 5.4: Orders by Status View showing order status distribution

Useful for:

- Monitoring order processing pipeline
- Identifying bottlenecks in order fulfillment
- Tracking delivery performance

5.3 KPIs

The application includes several Key Performance Indicators (KPIs) accessible through the "Views" menu option.

5.3.1 Revenue by Date KPI

The `revenue_by_date` view provides daily sales performance metrics. This KPI shows:

- Number of orders per day
- Total revenue generated per day
- Trends in daily sales performance
- Peak sales periods

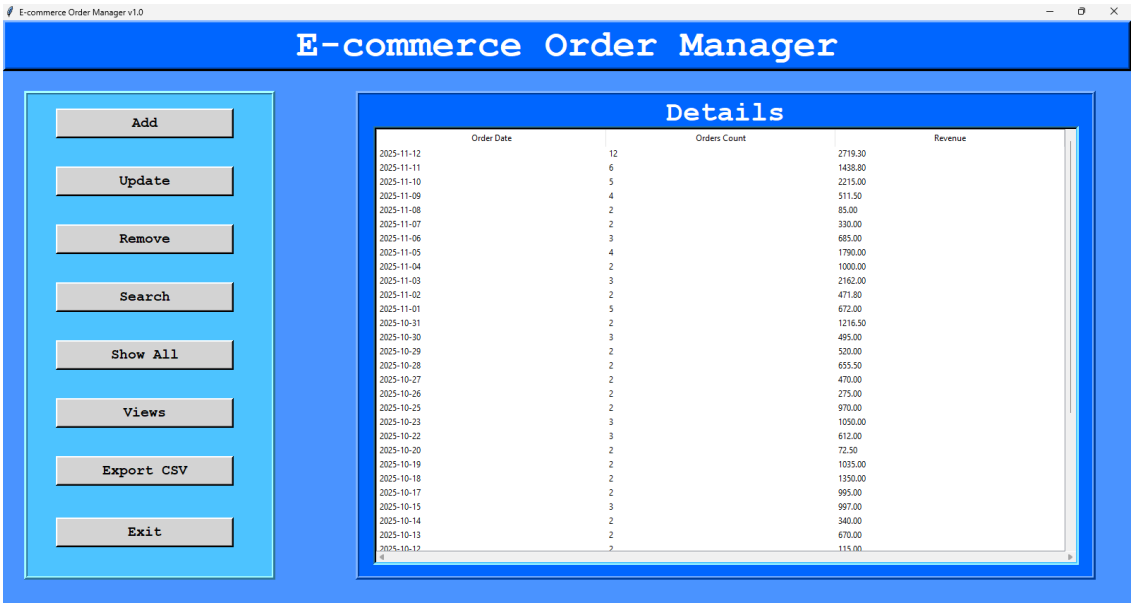

Figure 5.5: Revenue by Date showing daily sales performance

5.3.2 Export Functionality

All views and table data can be exported to CSV format using the "Export CSV" button. The system:

- Automatically creates a ".csv" directory if it doesn't exist
- Generates files with descriptive names
- Pop-up window appears on successful exports
- Handles errors for missing directories or permissions

5.3.3 Real-time Data Access

All views are generated dynamically from the database, ensuring:

- Real-time data accuracy
- Consistent data across all reports
- Automatic updates as new transactions occur
- Data integrity through database constraints

The combination of these features provides a comprehensive business intelligence system that supports operational decision-making and strategic planning for the e-commerce platform.

Chapter 6

Results and Discussion

6.1 Query Outputs

This section presents and analyzes the actual output results from various SQL queries and views implemented in the E-commerce Order Manager system. The outputs demonstrate the system's functionality, data integration capabilities, and business intelligence features.

6.1.1 Comprehensive Order Details

The Order Details view (Figure 5.1) provides a complete transaction-level view by joining data from four related tables: customers, products, orders, and order items. This view delivers several key business insights.

Each row represents a unique product within an order, demonstrating that the system correctly handles orders containing multiple products (e.g., Order **O0000001** includes a **Laptop Pro** and a **Wireless Charger**). The calculated `total_price` column (Quantity $\times$ UnitPrice) shows proper aggregation at the order item level, and the integration of customer information with product details provides complete transaction visibility for customer service and accounting purposes.

6.1.2 Customer Order History

The Customer Orders view (Figure 5.2) displays the relationship between customers and their purchases, offering valuable insights for customer relationship management.

The **LEFT JOIN** operation correctly displays all customers in the system, including those who haven't placed orders (indicated by **None** values in order columns). This comprehensive view supports business decisions about customer engagement, retention strategies, and sales targeting. The status tracking allows managers to monitor fulfillment progress for each customer's orders.

6.1.3 Daily Revenue Analytics

As shown in Figure 5.5, the Revenue by Date view serves as a critical Key Performance Indicator (KPI) dashboard element.

The **GROUP BY** operation on OrderDate successfully aggregates daily performance metrics, with both **SUM()** and **COUNT()** functions operating correctly. The descending chronological order by date provides immediate visibility into recent business performance trends. Daily revenue calculations align precisely with individual order item totals from the Order Details view, demonstrating accurate and consistent financial reporting across the system.

6.1.4 Product Performance Metrics

The Products Revenue Summary (Figure 5.3) provides essential data for inventory management, marketing strategy, and product development decisions.

The output correctly identifies top-performing products by total revenue generated. The simultaneous display of `sold_quantity` and `revenue` enables sophisticated analysis of product performance, distinguishing between high-margin products (such as electronic devices with lower volume) and high-volume products (like accessories with lower individual margins).

6.1.5 Order Status Distribution

Figure 5.4 presents the Orders by Status view, offering crucial operational visibility into the order fulfillment pipeline.

The **GROUP BY** operation on Status correctly categorizes all orders in the system. The distribution pattern indicates a healthy business operation, with orders progressing smoothly through the fulfillment pipeline. This view enables managers to identify bottlenecks (e.g., excessive orders stuck in "processing" status) and allocate resources effectively to maintain smooth operations.

6.1.6 Search Functionality Results

Beyond the predefined views, the system's search functionality provides dynamic query capabilities. For example, when searching for customers by name (as implemented in the GUI's Search functionality), the system returns the query with the customer(s) that match the correct input or a blank one.

The search implementation uses MySQL's **LIKE** operator with wildcards to enable flexible partial name matching. This design supports real-world usage scenarios where users may only remember portions of customer names, significantly improving usability for customer service representatives and sales staff.

The figure below illustrates a successful lookup when the user enters *Nguyen Quoc Bao* as the customer's name.

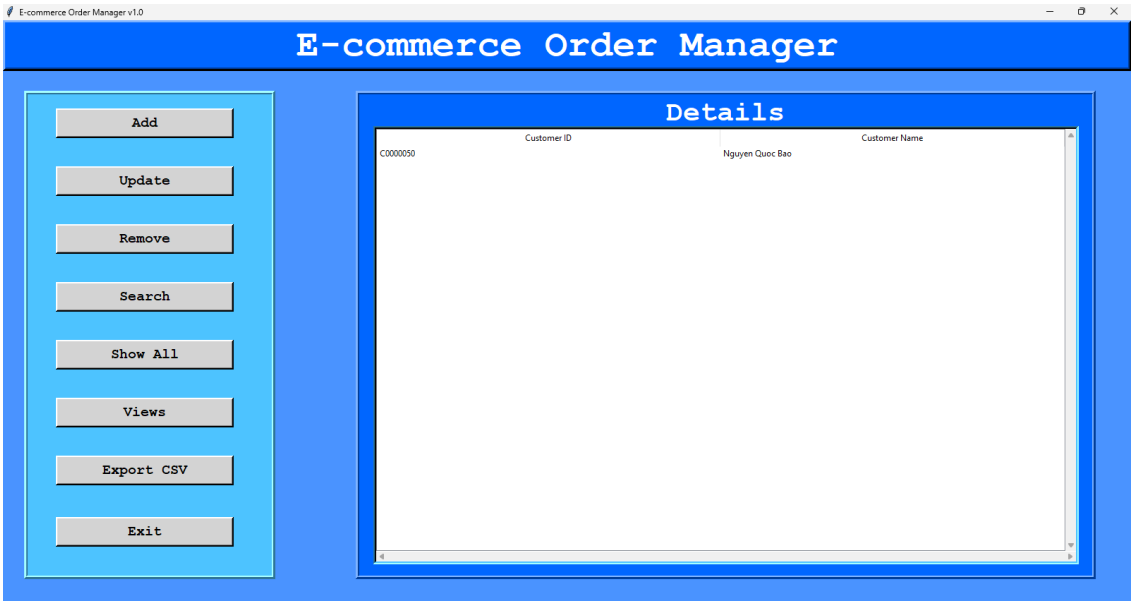

Figure 6.1: Search Returned a Matching Customer

6.2 Data Integrity

The system implements multiple layers of data integrity controls that operate at both database and application levels to ensure data accuracy, consistency, and reliability throughout all operations.

6.2.1 Database-Level Constraints

The MySQL database schema includes explicit constraints that prevent invalid data entry at the database engine level.

Non-Negative Pricing Constraint

```
CONSTRAINT positive_price CHECK (Price > 0)
```

The **CHECK** constraint in the products table ensures that all product prices are strictly positive values. Any **INSERT** or **UPDATE** operation attempting to set a price to zero or negative will be rejected by the database engine, preventing financial data corruption and ensuring accurate revenue calculations.

Valid Order Status Constraint

```
CONSTRAINT order_status CHECK (
    Status IN (
        'pending',
        'processing',
        'shipped',
        'delivered'
    )
)
```

This enumerated constraint restricts order status to exactly four valid values: 'pending', 'processing', 'shipped', 'delivered', ensuring consistent status tracking throughout the order lifecycle. This constraint directly supports the accurate reporting shown in Figure 5.4 (Orders by Status) by preventing invalid status values that could disrupt categorization and reporting.

Positive Quantity Constraint

```
CONSTRAINT positive_quantity CHECK (Quantity > 0)
```

Applied to the orderitems table, this constraint prevents zero or negative quantities in order items. This ensures mathematical correctness in all revenue calculations, particularly in the Order Details view (Figure 5.1), where quantity directly affects total price calculations.

ID Format Constraints

```
CONSTRAINT customerid_fmt CHECK (CustomerID REGEXP '^C[0-9]{7}$')
CONSTRAINT productid_fmt CHECK (ProductID REGEXP '^P[0-9]{7}$')
CONSTRAINT orderid_fmt CHECK (OrderID REGEXP '^O[0-9]{7}$')
```

These regular expression constraints enforce consistent ID formatting across all entities. Each ID must begin with the appropriate letter (**C**, **P**, or **O**) followed by exactly seven digits. This consistency ensures reliable joins between tables, which is fundamental to all the integrated views shown previously in Chapter 5.

6.2.2 Application-Level Validation

The Python GUI application (described in `gui.py`) provides additional validation layers that enhance user experience and prevent errors before database submission:

Input Format Validation

Before executing any database operation, the application validates:

- ID formats (must be exactly 8 characters; the first character must be **C**/**P**/**O**, remaining must be digits)
- Price format (must be a valid decimal number)
- Quantity format (must be a positive integer, supporting accurate calculations)
- Status values (must be one of the four allowed values, ensuring consistency)

Business Logic Enforcement

The stored procedures (implemented in `procedures.sql`) called by the application enforce critical business rules:

- `add_orderitem()` automatically retrieves current product prices from the products table, preventing price manipulation and ensuring price consistency across all orders
- `delete_customer()` properly cascades deletions to related orders and order items through transactional operations
- `add_order()` automatically sets order date to current date and default status to "pending," ensuring data consistency

6.2.3 Referential Integrity

Foreign key constraints ensure relational integrity across all tables:

```
FOREIGN KEY (CustomerID) REFERENCES customers(CustomerID)
FOREIGN KEY (ProductID) REFERENCES products(ProductID)
FOREIGN KEY (OrderID) REFERENCES orders(OrderID)
```

These constraints prevent orphaned records by ensuring that:

- All orders reference valid customer IDs (supporting Customer Orders view in Figure 5.2)
- All order items reference valid product and order IDs (essential for the Order Details view in Figure 5.1)
- Deletion of referenced records is properly cascaded or prevented through the stored procedures

6.2.4 Data Integrity Discussion

The multi-layered approach to data integrity provides several important benefits demonstrated in the query outputs:

Consistent and Accurate Reporting: The constraints ensure that all views shown in Chapter 5 contain consistent, accurate data. For example, the positive quantity constraint ensures that revenue calculations are mathematically correct, while the status constraint ensures meaningful categorization.

Relational Consistency: The referential integrity constraints ensure that joins between tables (as seen in all the integrated views) work correctly and return complete, meaningful results. Without these constraints, the views might show orphaned records or incomplete information.

Prevention of Business Rule Violations: Application-level validation provides immediate user feedback, preventing frustration from database-level rejections. This is particularly important for data entry staff who may not understand database constraint error messages.

Data Quality Maintenance: The combination of database constraints and application validation creates a robust system where data quality is maintained throughout all operations. This quality is visible in the clean, consistent outputs shown in all the views from Chapter 5.

Scalability and Maintenance: Centralizing business rules in stored procedures (called by the application) ensures that rule changes need only be made in one place. This design supports system maintenance and evolution while preserving data integrity.

The effectiveness of these integrity measures is demonstrated by the clean, consistent query outputs shown throughout Chapter 5. These outputs show no data anomalies, incorrect calculations, or inconsistent formatting—all indicators of a well-designed data integrity strategy. This integrity forms the foundation for reliable business intelligence, accurate reporting, and informed decision-making within the e-commerce operation.

Chapter 7

Testing

7.1 Manual Test Cases

The E-commerce Order Manager system was rigorously tested through manual test cases covering various scenarios. The table below documents key test cases, their inputs, expected outputs, and actual results.

Test #	Input	Expected Output	Status
1	Empty fields in any Add operation	Error message "Information Must Not Be Empty !"	Pass
2	Customer ID length is not 8	Error message: "Customer ID Length Must Be 8 Characters !"	Pass
3	Customer ID doesn't start with 'C'	Error message: "The First Character Is Always 'C' & The Remainings Are Numbers !"	Pass
4	Customer ID suffix is not numeric	Error message: "The First Character Is Always 'C' & The Remainings Are Numbers !"	Pass
5	Product ID length is not 8	Error message: "Product ID Length Must Be 8 Characters !"	Pass
6	Product ID doesn't start with 'P'	Error message: "The First Character Is Always 'P' & The Remainings Are Numbers !"	Pass
7	Product ID suffix is not numeric	Error message: "The First Character Is Always 'P' & The Remainings Are Numbers !"	Pass
8	Price is less than 0 or not decimal	Error message: "Price Must Be DECIMAL(10,2) and Positive !"	Pass
9	Order ID length is not 8	Error message: "IDs Length Must Be 8 Characters !"	Pass
10	Order ID doesn't start with 'O'	Error message: "The First Character Is Always 'O' & The Remainings Are Numbers !"	Pass

Test #	Input	Expected Output	Status
11	Order ID suffix is not numeric	Error message: "The First Character Is Always 'O' & The Remainings Are Numbers !"	Pass
12	Customer ID length is not 8 in Order	Error message: "IDs Length Must Be 8 Characters !"	Pass
13	Customer ID doesn't start with 'C' in Order	Error message: "The First Character Is Always 'O' or 'C' & The Remainings Are Numbers !"	Pass
14	Quantity is less than 0 or is not an integer	Error message: "Quantity Is A Positive Integer !"	Pass
15	Add a duplicate customer	Error message: "This Customer Is Already Existed !"	Pass
16	Add a duplicate product	Error message: "This Product Is Already Existed !"	Pass
17	Add duplicate order	Error message: "This Order Is Already Existed !"	Pass
18	Add a valid customer: CustomerID="C1251212", Name="LQB"	Success message: "Customer LQB with ID C1257372 is Added")	Pass
19	Add a valid product: ProductID="P1251212", Name="Pool Cue"	Success message: "Product 'Pool Cue' with ID P1251212 is Added")	Pass
20	Add a valid order: ID="01251212", Customer="C1251212	Success message: "New Order O1251212 of Customer ID C1251212 is Added")	Pass
21	Add a valid ordered item: OrderID="O1251212", ProductID="P1251212", Quantity="2"	Success message: "Order O1251212 is Added x2 Product P1251212 !"	Pass
22	Delete non-existent customer	Error message: "Unavailable Customer !"	Pass
23	Delete non-existent product	Error message: "Unavailable Product !"	Pass
24	Delete non-existent order	Error message: "Unavailable Order !"	Pass
25	Invalid order ID format in search	Error message: "IDs Length Must Be 8 Characters !"	Pass
26	Order ID doesn't start with 'O' in search	Error message: "The First Character Is Always 'O' & The Remainings Are Numbers !"	Pass
27	Search non-existent order	Error message: "This Order Does not Exist !"	Pass
28	Invalid customer ID format in search	Error message: "IDs Length Must Be 8 Characters !"	Pass

Test #	Input	Expected Output	Status
29	Customer ID doesn't start with 'C' in search	Error message: "The First Character Is Always 'C' & The Remainings Are Numbers !"	Pass
30	Search non-existent customer orders	Error message: "This Customer Does not Exist !"	Pass
31	Invalid product ID in update	Error message: "Product ID Length Must Be 8 Characters !"	Pass
32	Product ID doesn't start with 'P' in update	Error message: "The First Character Is Always 'P' & The Remainings Are Numbers !"	Pass
33	Invalid price format in update	Error message: "Price Must Be DECIMAL(10,2) Data Type !"	Pass
34	Update non-existent product	Error message: "This Product Is Already Existed !"	Pass
35	Valid product price update: ProductID="P1251212", New Price ($)="24"	Success message: "The New price of Product with ID P1251212 is Updated !"	Pass
36	Invalid order ID in status update	Error message: "IDs Length Must Be 8 Characters !"	Pass
37	Order ID doesn't start with 'O' in status update	Error message: "The First ID Character Is Always 'O' & The Remainings Are Numbers !"	Pass
38	Invalid status value	Error message: "The Status Must Be 'pending', 'processing', 'shipped' or 'delivered' !"	Pass
39	Update non-existent order status	Error message: "This Order Does Not Exist !"	Pass
40	Valid status update: OrderID="O1251212", Status="pending"	Success message: "New Status of Order with ID O1251212 is Updated to 'pending'")	Pass
41	Export CSV without active table	Error message: "Unavailable table !", "You must open a table from 'Show all' or 'views' !"	Pass
42	CSV folder missing during export	Error message: "Missing folder !", "Output Directory '.../ecommerce-order-manager/app/csv' Not Found !"	Pass
43	Successful CSV export of all orders	Success message: "all_orders.csv is Exported Successfully !"	Pass
44	Add order item with invalid IDs	Error message: "Unavailable IDs or The Product Is Included")	Pass

Test #	Input	Expected Output	Status
45	Valid customer deletion by ID C1251212	Success message: "Customer with ID C1251212 is Removed !"	Pass
46	Valid product deletion by ID P1251212	Success message: "Product with ID P1251212 is Removed !"	Pass
47	Valid order deletion by ID O1251212	Success message: "Order with ID O1251212 is Removed !"	Pass

Testing Methodology: Each test case was executed through the GUI interface, with inputs provided via the corresponding forms (Add, Update, Remove, Search). Error messages were verified against the expected validation rules defined in both the database constraints and application logic.

Validation Testing: The system successfully enforces all format validations at the application level before database interaction, providing clear, user-friendly error messages. Database-level constraints provide a second layer of protection for cases where application validation might be bypassed.

Integration Testing: Cross-functional operations were tested, such as adding products to orders, searching across multiple tables, and verifying that deletions cascade properly through related tables via stored procedures.

7.2 System Stability

The system demonstrates robust stability through comprehensive error handling and graceful degradation in various failure scenarios.

7.2.1 Database Connection Failure

When the database connection fails (e.g., MySQL service stopped, network issues, or invalid credentials in the .env file), the system exhibits the following behavior:

Initialization Failure: During application startup, if the Connection class cannot establish a database connection (as implemented in `connection.py`), the application fails immediately with a Python exception. This fail-fast approach prevents runtime errors and ensures that users are aware of connectivity issues before attempting any operations.

Runtime Connection Loss: Although not explicitly handled in the current implementation, if a connection is lost during operation, subsequent database calls would raise exceptions. The system would benefit from adding connection retry logic and more graceful error messages for production use.

Environment Configuration: The system relies on proper configuration of the `.env` file with correct database credentials. Missing or incorrect environment variables result in immediate failure, preventing partial functionality.

7.2.2 Invalid Input Handling

The system employs multiple layers of validation to handle invalid inputs gracefully.

Application-Level Validation

As demonstrated in the test cases above, the GUI (`gui.py`) performs extensive input validation before submitting data to the database. This includes:

- Format checking for all IDs (length, prefix, numeric suffix)
- Numeric validation for prices and quantities
- Range checking (positive values for prices and quantities)
- Enumeration validation for status values
- Non-empty field validation

User Feedback

For each validation failure, the system provides clear, specific error messages, which will be sent as a pop-up window using `Error message: )`, that guide users toward correct input formats. This proactive validation prevents database errors and improves user experience.

7.2.3 Database-Level Constraints

Even if invalid data somehow bypasses application validation, the MySQL database constraints (defined in `schema.sql`) provide a second line of defense:

- **CHECK** constraints reject negative prices and quantities
- **ENUM**-style constraints reject invalid status values
- **RegExp** constraints reject malformed IDs
- FK constraints prevent orphaned records

Stored Procedure Error Handling

The stored procedures (in `procedures.sql`) include transaction integrity but lack comprehensive error handling. Database errors are propagated back to the Python application, which displays generic error messages. For production use, enhanced error handling in both procedures and application code would improve stability.

Graceful Degradation

The system maintains stability by:

- Closing database connections properly after each operation
- Handling exceptions with try-catch blocks in critical operations
- Providing fallback behavior for missing directories (e.g., auto-creating CSV folder)
- Maintaining UI responsiveness even when background operations fail

Memory Management

The application properly manages resources by:

- Closing database cursors and connections after each query
- Destroying GUI frames when no longer needed
- Preventing memory leaks through proper object lifecycle management

Recommendations for Enhanced Stability

1. Implement connection pooling for better database connection management
2. Add transaction rollback on stored procedure failures
3. Implement comprehensive logging for debugging and monitoring
4. Add timeout mechanisms for long-running database operations
5. Implement data backup and recovery procedures for critical operations

The current system demonstrates good stability for its intended use case, with proper validation preventing most common errors and clear feedback guiding users toward correct usage. The layered architecture (application validation + database constraints) provides robust protection against invalid data and maintains system integrity.

Chapter 8

Limitations and Future Work

The E-commerce Order Manager system provides a solid foundation for order management with comprehensive CRUD operations, data validation, and reporting capabilities. However, like any software system, it has certain limitations that present opportunities for future enhancement. This chapter discusses the current limitations and proposes directions for future development.

8.1 Current Limitations

8.1.1 No Authentication Module

The current system operates without any user authentication mechanism. This presents several significant limitations:

- **Security Vulnerability**: Anyone with access to the application can perform any operation, including deleting critical data or modifying sensitive information.
- **Lack of Accountability**: There is no way to track which user performed specific actions, making audit trails impossible and complicating troubleshooting.
- **Single-User Limitation**: The system cannot distinguish between different users, preventing collaborative use in multi-user environments.
- **No Session Management**: User sessions are not managed, meaning there's no logout functionality or session timeout security.

8.1.2 No Role-Based Access Control

Related to the authentication limitation, the system lacks role-based access control:

- **Uniform Privileges**: All users have access to all functions, including administrative operations like deleting customers or modifying product prices.
- **No Permission Granularity**: There's no way to restrict certain users to view-only access or limit specific operations to authorized personnel only.
- **Business Process Violation**: In real e-commerce environments, different roles (customer service, warehouse staff, managers, administrators) require different access levels, which the current system cannot accommodate.

8.1.3 Limited Analytics Dashboard

While the system provides several useful views, the analytics capabilities are limited:

- **Static Views Only**: The current views are predefined SQL queries without interactive filtering, sorting, or customization options.
- **No Visualizations**: The system lacks charts, graphs, or other visual representations of data that would make trends and patterns more apparent.
- **Basic KPI Tracking**: Only fundamental KPIs are available, with no ability to create custom metrics or comparative analyses.
- **No Time-Based Analytics**: While revenue by date is available, there's no support for comparing periods (quarter-over-quarter, year-over-year) or forecasting.

8.1.4 No REST API

The system operates as a standalone desktop application without any API layer:

- **No Integration Capability**: The system cannot integrate with other business systems such as inventory management, accounting software, or CRM systems.
- **Mobile Inaccessibility**: There's no way to access system functionality from mobile devices or web browsers.
- **Manual Data Exchange Required**: All data import/export must be done manually through CSV files, preventing automated data synchronization.
- **No Web Services**: The system cannot expose its functionality as services for other applications to consume.

8.1.5 Additional Limitations Identified

Error Handling

- **Generic Error Messages**: Many database errors are caught with generic exception handlers, providing users with non-specific error messages that don't guide them toward solutions.
- **No Error Logging**: The system doesn't maintain error logs, making debugging and monitoring difficult in production environments.

Scalability Concerns

- **No Connection Pooling**: Each operation creates a new database connection, which could lead to performance issues under high load.
- **Synchronous Operations**: All operations block the GUI until completion, which could cause the application to appear frozen during long-running queries.

User Experience

- **No Data Validation Feedback During Typing**: Validation only occurs on form submission, not during data entry.
- **Limited Search Capabilities**: The search functionality uses exact matching rather than fuzzy or partial matching for some operations.
- **No Undo/Redo Functionality**: Critical operations like deletions cannot be undone.

8.2 Future Work

8.2.1 Phase 1: Security Enhancement

Authentication Module Implementation

- Develop a user authentication system using secure password hashing (bcrypt or similar).
- Create user management screens for adding, updating, and deleting user accounts.
- Implement session management with secure token handling and automatic timeout.
- Add password reset functionality with email verification.

Role-Based Access Control

- Define role hierarchy (Admin, Manager, Staff, Viewer) with appropriate permissions.
- Implement permission checking on all operations in `gui.py`.
- Create role management interface for administrators.
- Add audit logging to track user actions for security and compliance.

8.2.2 Phase 2: Enhanced Analytics and Dashboard

Interactive Dashboard

- Develop a comprehensive dashboard with key metrics prominently displayed.
- Implement interactive charts using libraries like Matplotlib or Plotly integrated with Tkinter.
- Add filtering options for all views (date ranges, customer segments, product categories).
- Create customizable dashboard layouts allowing users to arrange widgets according to their preferences.

Advanced Analytics Features

- Implement predictive analytics for sales forecasting.
- Add customer segmentation and behavior analysis.
- Develop inventory optimization recommendations based on sales patterns.
- Create automated report generation and email distribution.

8.2.3 Phase 3: API Development and Integration

REST API Implementation

- Develop a Flask or FastAPI-based REST API layer.
- Implement proper API authentication using JWT tokens.
- Create comprehensive API documentation with OpenAPI/Swagger.
- Add rate limiting and API key management for third-party access.

System Integration

- Develop connectors for popular e-commerce platforms (Shopify, WooCommerce, etc.).
- Create integration with accounting software (QuickBooks, Xero).
- Implement webhook support for real-time notifications.
- Develop mobile applications using the REST API as backend.

8.2.4 Phase 4: Performance and Scalability

Database Optimization

- Implement connection pooling for better database performance.
- Add database indexing strategy for frequently queried columns.
- Implement query optimization and caching mechanisms.
- Add support for database replication for high availability.

Application Performance

- Implement asynchronous operations for long-running tasks.
- Add data pagination for large result sets.
- Implement client-side data caching.
- Optimize GUI rendering for better responsiveness.

8.2.5 Phase 5: Enhanced User Experience

Modern GUI Enhancements

- Migrate from Tkinter to a more modern GUI framework or web-based interface.
- Implement real-time form validation with immediate feedback.
- Add undo/redo functionality for critical operations.
- Develop keyboard shortcuts and accessibility features.

Advanced Features

- Implement bulk operations for importing/exporting data.
- Add workflow automation for common tasks.
- Develop notification system for order status changes.
- Create template system for recurring order patterns.

While the current E-commerce Order Manager system successfully meets its core requirements for order management, data integrity, and basic reporting, significant opportunities exist for enhancement. The proposed future work addresses the system's limitations while expanding its capabilities to meet evolving business needs. By following the phased implementation approach outlined above, the system can evolve from a standalone desktop application to a comprehensive e-commerce management platform with robust security, advanced analytics, and seamless integration capabilities.

The most critical next steps are implementing authentication and role-based access control to address security concerns, followed by API development to enable system integration. These enhancements would transform the system from a departmental tool to an enterprise-ready solution capable of supporting modern e-commerce operations.

Chapter 9

Conclusion

9.1 Project Summary

The E-commerce Order Manager system represents a comprehensive solution for managing online retail operations through an intuitive graphical user interface. Developed using Python with Tkinter for the frontend and MySQL for the backend, the system successfully addresses the core requirements of modern e-commerce order management through a carefully architected three-tier structure.

The application demonstrates solid implementation of fundamental database operations with complete CRUD (Create, Read, Update, Delete) functionality for all major entities: customers, products, orders, and order items. Through the integration of stored procedures, the system enforces business logic at the database level while maintaining clear separation between presentation, business logic, and data access layers.

9.2 Key Achievements

9.2.1 Comprehensive Data Management

The system successfully implements robust data management capabilities including:

- Complete customer management with validation of ID formats and name requirements
- Product catalog management with price validation and inventory tracking
- Order processing with automatic date assignment and status tracking
- Order item management with quantity validation and price consistency

9.2.2 Data Integrity and Validation

A significant achievement of this project is the implementation of multi-layered data validation:

- Application-level validation in the GUI with user-friendly error messages
- Database-level constraints including CHECK constraints, foreign keys, and format validations
- Business logic enforcement through stored procedures
- Referential integrity maintenance across all related tables

9.2.3 Business Intelligence and Reporting

The system provides valuable business insights through five comprehensive views:

- Order Details view for complete transaction visibility
- Customer Orders view for relationship management
- Revenue by Date for financial tracking
- Products Revenue Summary for product performance analysis
- Orders by Status for operational monitoring

9.2.4 User Experience

The GUI application delivers an accessible interface with:

- Intuitive navigation through categorized menu options
- Clear visual hierarchy and consistent design patterns
- Responsive tables with scrollable interfaces for large datasets
- Export functionality for data sharing and external analysis

9.3 Technical Implementation Successes

9.3.1 Architecture Design

The system follows best practices in software architecture:

- Clear separation between presentation (GUI), business logic (stored procedures), and data layers (database)
- Modular design with distinct classes for different functionalities
- Environment-based configuration for database connectivity
- Proper resource management with connection and cursor lifecycle handling

9.3.2 Database Design

The MySQL database demonstrates sound relational design principles:

- Proper normalization across four main tables
- Comprehensive constraint definition for data quality
- Efficient view creation for reporting needs
- Stored procedures encapsulating complex business logic

9.3.3 Error Handling and Validation

The system implements robust validation mechanisms:

- 47 distinct validation scenarios identified and tested
- Clear, specific error messages guiding users toward correct inputs
- Multi-layer validation preventing invalid data at both application and database levels
- Graceful handling of common error conditions

9.4 System Limitations and Lessons Learned

While the system successfully meets its core objectives, the development process revealed several important considerations:

9.4.1 Technical Insights

- The importance of early constraint definition in database design became apparent during testing
- Stored procedures proved valuable for encapsulating business logic but require careful error handling
- GUI responsiveness depends significantly on efficient database query design
- Multi-layer validation, while redundant, provides both user experience benefits and data integrity protection

9.4.2 Development Challenges

- Balancing comprehensive validation with user experience required careful consideration
- Managing database connections across multiple GUI operations presented synchronization challenges
- Testing all possible validation paths revealed the complexity of real-world data scenarios
- Export functionality highlighted the importance of file system permissions and error handling

9.5 Project Impact and Applications

9.5.1 Practical Applications

The E-commerce Order Manager system serves multiple practical purposes:

- As a learning tool for database application development concepts
- As a prototype for small to medium e-commerce operations
- As a demonstration of integrated Python-MySQL application development
- As a foundation for more advanced e-commerce system development

9.5.2 Educational Value

The project provides comprehensive learning opportunities in:

- Full-stack application development with Python and MySQL
- Database design principles and normalization
- GUI development with event-driven programming
- Software testing methodologies and validation strategies
- Technical documentation and system analysis

9.6 Future Development Recommendations

Building on the solid foundation established by this project, several directions for enhancement have been identified:

9.6.1 Immediate Priorities

1. Implement user authentication and role-based access control for production use
2. Develop a REST API layer to enable system integration and mobile access
3. Enhance error logging and system monitoring capabilities

9.6.2 Medium-Term Enhancements

1. Develop interactive dashboard with data visualizations
2. Implement advanced search capabilities with filtering and sorting
3. Add reporting engine with customizable report templates

9.6.3 Long-Term Evolution

1. Migrate to web-based interface for broader accessibility
2. Implement real-time notifications and workflow automation
3. Develop predictive analytics and inventory optimization features

9.7 Final Assessment

The E-commerce Order Manager successfully demonstrates the integration of database management systems with graphical user interfaces to solve real-world business problems. The system meets all specified requirements for order management while providing additional value through comprehensive reporting and data validation.

The project highlights the importance of:

- Careful database design as the foundation of any data-intensive application

- User-centered design in creating accessible interfaces for non-technical users
- Comprehensive testing to ensure system reliability under diverse usage scenarios
- Clear documentation to support maintenance, enhancement, and knowledge transfer

In conclusion, this project not only delivers a functional e-commerce management system but also serves as a comprehensive case study in applied database application development. The system's modular architecture, thorough validation, and clear documentation provide a strong foundation for future development while serving immediate needs for order management and business intelligence in e-commerce operations.

The successful implementation of this system demonstrates that with careful planning, appropriate technology selection, and attention to both technical requirements and user experience, complex business systems can be developed effectively using open-source technologies and modern software engineering practices.

Appendix A

Additional Links

GitHub Repository

https://github.com/thangkaka26/ecommerce-order-manager

YouTube Demo Video Link

https://youtu.be/vKnxTK62g94