NATIONAL ECONOMICS UNIVERSITY

FACULTY OF DATA SCIENCE AND ARTIFICIAL INTELLIGENCE

Developing a Medical Database Management Application

Student Members

Name	Student ID
Le Sy Huy	11247296
Khuat Dinh Trung	11247362
Be Thanh Dat	11247272

Supervisor

Dr. Tran Hung

December 11, 2025

Declaration

We, Le Sy Huy, Khuat Dinh Trung, Be Thanh Dat of the Artificial Intelligence,the University of National Economics University , confirm that this is my own work and 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. I understand that if failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.
I give consent for my work to be made available more widely to members of UoR and public with interest in teaching, learning and research.

Le Sy Huy
Khuat Dinh Trung
Be Thanh Dat
December 11, 2025

Abstract

The "Hospital Patient Manager" project aims to standardize hospital data from a single UNF table into 3NF, implement a MySQL database with sample data, and develop a Python GUI application to manage patients, doctors, treatments, and treatment sessions. The application supports CRUD functions for the key entities, provides reports for four required queries (INNER JOIN, LEFT JOIN, multi-table JOIN, high-cost treatments), CSV export, KPI dashboard with charts, and a search/filter toolkit. The sample dataset (seed) includes 50 patients, 8–12 doctors, 10–15 treatment types, and 40–80 treatment sessions spanning multiple days. The report fully documents the UNF $\rightarrow$ 1NF $\rightarrow$ 2NF $\rightarrow$ 3NF conversion process, ERD diagrams, explanations of PK/FK relationships, interface design, testing methods, as well as submission procedures on GitHub and a demo video.

Contents

Chapter 1

Introduction

1.1 Real-life context

Currently, hospitals often store information about patients, doctors, and treatment sessions in a single unnormalized table, also known as UNF (Unnormalized Form). This method of data storage leads to redundancy and makes data management and updates challenging. For example, when a doctor's or patient's name changes, multiple rows must be updated, increasing the risk of inconsistencies and errors. Additionally, inserting or deleting data can create anomalies, and storage requirements are unnecessarily high. These issues directly affect the ability to generate accurate and timely reports, making it difficult to track treatment costs, treatment counts, and appointment schedules. Against this backdrop, the Hospital Patient Manager project is developed to normalize the data, restructure the database into a relational model, and provide a user-friendly management tool that improves the daily workflow of healthcare and administrative staff.

1.2 System Objectives

The main objective of the Hospital Patient Manager system is to develop an effective hospital data management application that allows accurate, consistent, and user-friendly management of patients, doctors, treatment categories, and treatment sessions. The system is designed to normalize data from UNF to 3NF, minimizing redundancy and reducing data update errors. Additionally, the system provides a graphical user interface (GUI) that enables staff to perform core functions such as creating, editing, deleting, searching, and viewing reports. The system also integrates statistical queries and a dashboard to monitor treatment activities, average costs, and high-cost treatments, supporting hospital management in making more informed operational decisions.

1.3 Scope and Target Users

The scope of the system is limited to basic hospital data management, focusing on input, processing, storage, and retrieval of information related to patients, doctors, treatment categories, and treatment sessions. The system does not extend to advanced functionalities such as electronic medical record (EMR) management, insurance, billing, user authorization, or specialized hospital workflows. The primary users of the application are healthcare and administrative staff who interact directly with the data to enter, modify, search, and generate reports. Furthermore, the system can serve educational and research purposes or act as

a demonstration tool in courses related to database systems and application development, helping learners understand data normalization principles, relational database design, and the implementation of a fully functional data management application. Core functionalities, including CRUD operations, reporting, dashboards, search, and data filtering, are all within the scope, ensuring the system is practical, deployable, and extensible for future development.

1.4 Methods and Technologies Used

To build the system, the project adopts an approach based on the relational database model, combined with a data-normalization process from UNF to 3NF to ensure that the data structure is logical, consistent, and free from redundancy. The design procedure includes analyzing functional dependencies, identifying primary and foreign keys, constructing the ERD model, and transforming it into a complete relational schema.

In terms of technology, the system is implemented using the Python programming language, with the user interface developed through the Tkinter library — a built-in GUI framework in Python that supports window creation, interactive widgets, and basic drawing operations through the Canvas component. (Depending on the group's preference, alternatives such as PyQt or Streamlit may also be used.) The database is managed using MySQL, with connectivity provided by the mysql-connector or PyMySQL libraries to ensure stable, efficient, and scalable data operations.

All source code is organized and maintained on GitHub, alongside schema.sql and seed.sql files used to initialize and populate the database automatically. This methodology ensures high reusability, ease of deployment, straightforward testing, and overall suitability for the requirements of the course project.

Chapter 2

UNF to 3NF Normalization Process

2.1 Initial UNF Structure

The initial dataset of the hospital management system is stored in a single **UNF (Unnormalized Form)** table that combines **patient, doctor, treatment**, and **session** information into one record. A typical UNF record contains:

Patient information (ID, Name, Birthdate, Gender, Phone)
Doctor information (ID, Name, Specialty, Gender, Phone)
Treatment information (ID, Treatment Name, Cost)
Session information (ID, Patient Name, Doctor Name, Treatment, Cost, Date)

This structure leads to **data redundancy, update anomalies, insertion anomalies**, and **deletion anomalies**.

Data redundancy: doctor or patient information repeated many times.

Insertion anomaly – cannot add a new doctor unless there is an associated treatment session.

Update anomaly – updating a doctor's phone number requires modifying it in multiple rows.

Deletion anomaly – deleting a treatment session may cause the loss of patient/doctor data.

2.2 Conversion to 1NF

To convert the initial UNF dataset into **First Normal Form (1NF)**, several structural improvements must be applied to ensure the data becomes well-organized and free of repeating groups.

First, all values must be atomic, meaning each attribute contains only a single value rather than a list, combination, or nested structure. In the original UNF table, information such as doctor details, patient details, and treatment information may appear grouped together in one field, which violates 1NF principles.

Second, repeating groups must be removed. In the **UNF dataset**, a single patient could have multiple treatments or sessions listed within the same row. Converting to 1NF requires separating these repeated occurrences so that each row represents one unique event or transaction, typically one treatment session.

Third, a **primary key** must be clearly defined to uniquely identify each record. For this dataset, the session-level data serves as the most appropriate unique identifier because each session corresponds to one patient, one doctor, one treatment, and one date. After applying 1NF:

All columns contain atomic values

Repeated groups are eliminated
The table now consistently represents one session per row
The redesigned 1NF structure contains distinct fields for:
Doctor information (doctor id, doctor name, specialty, gender, phone)
Patient information (patient id, patient name, birthdate, gender, phone)
Treatment information (treatment id, treatment name, cost)
Session information (session id, doctor id, patient id, treatment id, date)

2.3 Conversion to 2NF

After the dataset has been transformed into **First Normal Form (1NF)**, the next step is to convert it into **Second Normal Form (2NF)**. The objective of **2NF** is to eliminate **partial dependencies**, meaning that every **non-key attribute** must depend on the **entire primary key**, rather than only a part of it. In the 1NF table, the **session** could be identified by a composite primary key formed from the **patient ID, doctor ID, treatment ID**, and the date. However, many attributes do not depend on the full combination of these fields; instead, they depend on only one specific identifier.
For example, all information related to a doctor—such as the doctor's name, specialty, gender, and phone number—depends solely on the doctor id. Likewise, patient-related information such as the patient's name, birthdate, gender, and phone number depends only on the patient id. Treatment information like the treatment name and its cost depends exclusively on the treatment id. These relationships indicate redundancy, because if a doctor participates in multiple sessions, all their personal details would be repeated many times.
To resolve this, the table must be **decomposed** so that each category of information is stored in a separate relation where all attributes depend entirely on a single **primary key**. As a result, the data originally stored in one large table is reorganized into a structured set of tables: **a doctor table, a patient table, a treatment table**, and **a session table** that connects them using **foreign keys**. This decomposition removes **partial dependencies** and ensures that each piece of information is stored only once in the appropriate entity table.
The resulting **2NF** design significantly reduces redundancy and prevents update anomalies. For instance, if a doctor's phone number changes, the system only needs to update a single row, instead of modifying it across multiple session records. With partial dependencies eliminated, the database is ready for the next stage of normalization: converting the schema into **Third Normal Form (3NF)** to eliminate transitive dependencies.

2.4 Conversion to 3NF

After achieving **Second Normal Form (2NF)**, the next step is converting the schema into **Third Normal Form (3NF)**. The purpose of **3NF** is to eliminate transitive dependencies, meaning that no **non-key attribute** should depend on another **non-key attribute**. Each attribute must depend directly and only on the **primary key** of its own table.
In the **2NF** structure, most dependencies have already been separated into the **doctor, patient, treatment**, and **session** tables. However, **3NF** requires a deeper check to ensure that no attribute indirectly depends on a primary key through another attribute. For instance, if the **doctor table** contained both the **specialty ID** and the **specialty name**, then the **specialty name** would be dependent on the **specialty ID**, not directly on the **doctor ID**. This would be considered a transitive dependency, and 3NF would require extracting specialty into a separate table. Similar issues could occur if patient age were stored instead of **birthdate**,

causing age to depend on the **current date** rather than the **patient ID**.
By ensuring that each table keeps only attributes that depend directly on its **primary key**, the final **3NF** structure becomes clean and logically organized. **The doctor table** contains only attributes that describe a doctor; **the patient table** contains details related solely to a patient; **the treatment table** stores treatment-specific information; and the **session table** records only the scheduling and linkage details between entities via **foreign keys**. At this point, the database structure is free of transitive dependencies, reducing the risk of anomalies and ensuring easier maintenance.

2.5 Final ERD

The final 3NF schema includes four core entities:

- **DOCTORS:**

DoctorID (PK), DoctorName, Gender, Phone, Specialty

- **PATIENTS:**

PatientID (PK), PatientName, Gender, Birthdate, Phone

- **TREATMENTS:**

TreatmentID (PK), TreatmentName, Cost

- **SESSIONS:**

SessionID (PK), DoctorID (FK), PatientID (FK), TreatmentID (FK), Date

The ERD visually represents all relationships and ensures **referential integrity** through the use of **primary keys** and **foreign keys**, with the Sessions table serving as the central linking entity between Doctors, Patients, and Treatments.

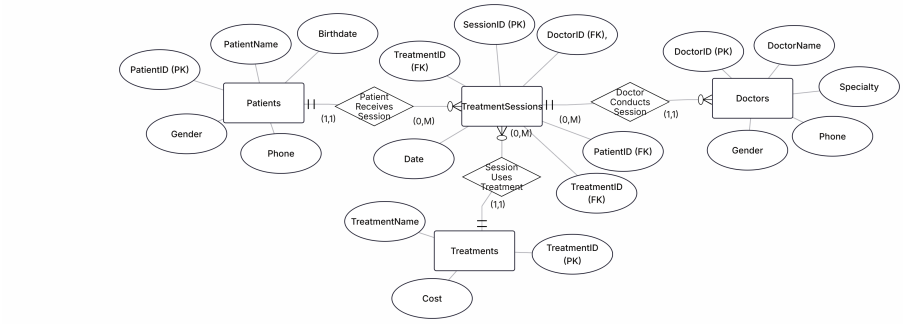

Figure 2.1: ERD diagram of the hospital management system

Chapter 3

ERD Relationships

3.1 Patients – TreatmentSessions (1:N)

The relationship between **Patients** and **TreatmentSessions** is a **one-to-many (1:N)** relationship.
A single patient can participate in multiple treatment sessions, while each treatment session is associated with exactly one patient.

Primary Key (Patients): patient id

Foreign Key (TreatmentSessions): patient id

In the original UNF dataset, patient information (name, birthdate, gender, phone) was duplicated across numerous rows whenever the patient appeared in different sessions.
After normalization into 3NF, all patient-specific attributes are stored once in the Patients table, and TreatmentSessions only references patients through a **foreign key**.
This relationship ensures:

Elimination of repeated patient information across sessions.

Data consistency when updating patient details (updated in one location only).

Support for queries such as retrieving a patient's entire treatment history.

3.2 Doctors – TreatmentSessions (1:N)

The relationship between **Doctors** and **TreatmentSessions** is also **one-to-many (1:N)**. A doctor may supervise **many treatment sessions**, but **each session is handled by only one doctor**.

Primary Key (Doctors): doctor id

Foreign Key (TreatmentSessions): doctor id

Before normalization, doctor information (name, specialty, gender, phone) was repeated for every session they handled. Separating Doctors into its own table removes redundancy and increases data accuracy.
This 1:N relationship provides the following benefits:

Easy reporting on how many sessions each doctor has conducted.

Avoids update anomalies because doctor information is stored in a single row.

Enforces referential integrity in sessions through foreign key constraints.

3.3 Treatments – TreatmentSessions (1:N)

The relationship between **Treatments** and **TreatmentSessions** follows the **one-to-many (1:N)** model. A single treatment type may be used in **multiple sessions**, but **each session applies exactly one treatment**.

Primary Key (Treatments): treatment id

Foreign Key (TreatmentSessions): treatment id

Initially, treatment names and costs were duplicated multiple times across sessions.

After normalization, treatment details are stored centrally in the Treatments table, and sessions reference them by ID.

Benefits of this relationship include:

Reduced redundancy and prevention of cost inconsistencies.

Centralized management of treatment information.

Ability to analyze usage frequency of specific treatments.

Chapter 4

Database Implementation

4.1 Table Structures

The database includes four core entity tables: **PATIENTS, DOCTORS, TREATMENTS**, and **TREATMENTSESSIONS**, each representing a primary business entity after the normalization process.

PATIENTS

This table stores all patient information, including patient ID, full name, birthdate, gender, and phone number. All attributes are fully dependent on the PatientID, ensuring compliance with 3NF and eliminating redundancy.

DOCTORS

This table contains attributes related to doctors, such as doctor ID, name, specialty, gender, and phone number. Each attribute depends only on the DoctorID, preventing duplication even when a doctor appears in many treatment sessions.

TREATMENTS

This table defines all medical treatments, with fields including treatment ID, treatment name, and treatment cost. Storing treatments separately avoids repeating treatment names and costs in multiple records.

TREATMENTSESSIONS

This is the core relationship table, linking patients, doctors, and treatments. It stores session ID, references to each of the other entities, and the session date. It acts as the central point connecting all entities using foreign keys.

Together, these structures form a clean and normalized 3NF-compliant schema.

4.2 Integrity Constraints

To ensure data accuracy and prevent inconsistencies, multiple integrity constraints are applied across the schema.

4.2.1 Primary Keys (PK)

each table includes a **primary key**:

PatientID for PATIENTS
DoctorID for DOCTORS
TreatmentID for TREATMENTS
SessionID for TREATMENTSESSIONS

These keys guarantee that each record is unique and identifiable.

4.2.2 Foreign Keys (FK)

The TREATMENTSESSIONS table uses **foreign keys** to maintain **referential integrity**:
PatientID → PATIENTS
DoctorID → DOCTORS
TreatmentID → TREATMENTS
These ensure that a session cannot reference a non-existent entity.

4.2.3 UNIQUE

Constraints prevent duplicated names where required

4.2.4 Check Constraints

Logical validations are enforced using **CHECK()**, including:
Cost $>$ **0** for treatments
Gender must satisfy allowed values
Phone numbers match a valid format
TreatmentDate must be a valid date
These ensure that invalid data cannot be inserted.

4.3 Sample Data Generation

Sample data is inserted using **seed.sql** to support application testing, CRUD operations, JOIN queries, and dashboard analytics.
The dataset includes the following:

At least 50 patients A diverse set of names, genders, birthdates, and phone numbers is generated to simulate real hospital data. This variety supports realistic testing.

At least 10 doctors Doctors with different specialties are included (e.g., cardiology, dermatology, neurology). This ensures diversity in session assignments.

At least 10 treatments Medical treatments with unique names and valid cost values ($>$ 0) are inserted. Treatment diversity allows meaningful querying and reporting.

40–80 treatment sessions These session entries create realistic relationships between patients, doctors, and treatments across a range of dates. They provide sufficient depth for:analytic queries, patient treatment history, doctor workload analysis, treatment cost summary, and dashboard visualizations.

Chapter 5

Application Design Architecture

5.1 Technology Stack Tools

This project is implemented using a combination of Python, MySQL, and GUI libraries to ensure efficient data management and user interaction. Key technologies and tools include:

Programming Language: Python 3.14
Database Management System: MySQL 8.4
GUI Framework: Tkinter (for desktop interface)
Database Connectivity: mysql-connector-python
IDE/Development Tools: VSCode
Version Control: GitHub

The chosen stack allows for a lightweight, responsive desktop application with secure and reliable database operations.

5.2 Project Structure Organization

The project directory is organized into clearly separated modules:

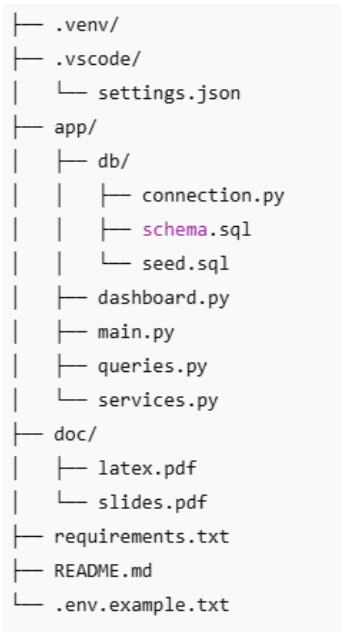

Figure 5.1: Project Structure

This structure separates database, business logic, user interface modules, and documentation, allowing efficient development and easy maintenance.

5.3 Database Access Connection Layer

The database connection layer handles all interactions with MySQL, separating data access logic from other parts of the application. This layer provides:

Centralized connection interface: All modules use a single connection point, making management easier.

Reusable query execution functions: Reduces code duplication when performing database operations.

Automatic error handling: Manages exceptions when queries fail, preventing application interruptions.

Transaction-safe operations: Ensures data consistency and reliable execution of CRUD operations.

5.4 User Interface Design Validation

The application features a **Graphical User Interface (GUI)**, which is a visual interface consisting of windows, forms, buttons, and other components that allow users to interact with the system intuitively.

The GUI is designed for simplicity and usability, enabling efficient management of patients, doctors, treatments, and treatment sessions.

The graphical interface is designed for clarity and ease of use, providing intuitive and efficient interaction with the system:

CRUD Forms with Validation: Simple forms for creating, updating, and managing patients, doctors, and treatment sessions, with built-in input checks.

Scrollable Data Tables: Allow users to view, sort, and edit records conveniently.

Dashboard Visualizations: Summarize key statistics for treatments, patients, and doctors.

Search and Filter Tools: Applied to major entities to quickly locate records.

Chapter 6

Functional Features

6.1 CRUD Interfaces

The system provides complete **CRUD (Create, Read, Update, Delete)** functionality for four core data entities. Each interface includes a data-entry form, a data table, validation logic, and database synchronization.

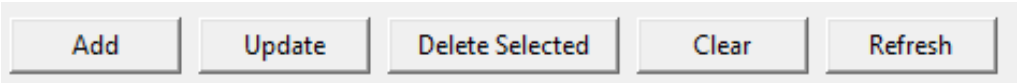

Figure 6.1: CRUD (Create, Read, Update, Delete)

6.1.1 Patients

The Patients interface provides a complete set of CRUD operations for managing patient information. The screen is divided into two main areas: a **data-entry form** on the left and a **patient table** on the right.

Main Features:

Create (Add): Users can create a new patient by filling in Name, Birthdate (YYYY-MM-DD), Phone, and selecting Gender from a dropdown list. The Patient ID field is left blank for new records. Pressing Add inserts the data into the MySQL database.

Read (View): All patient records are displayed in the table on the right, with fields: Id, Name, Birthdate, Phone, Gender. The table supports vertical scrolling.

Update: Selecting a row loads its values into the form. Users can modify the information and click Update to save the changes.

Delete: The Delete Selected button removes the selected patient from the database.

Clear Refresh:Clear resets the form fields.

Refresh reloads the patient list to reflect updated database content.

Data Validation

Birthdate requires the YYYY-MM-DD format.

Gender is selected from a dropdown to prevent typing errors.

Phone must be non-empty and follow numeric formatting.

Manual Patient ID entry is allowed, but duplication is prevented by PK constraints.

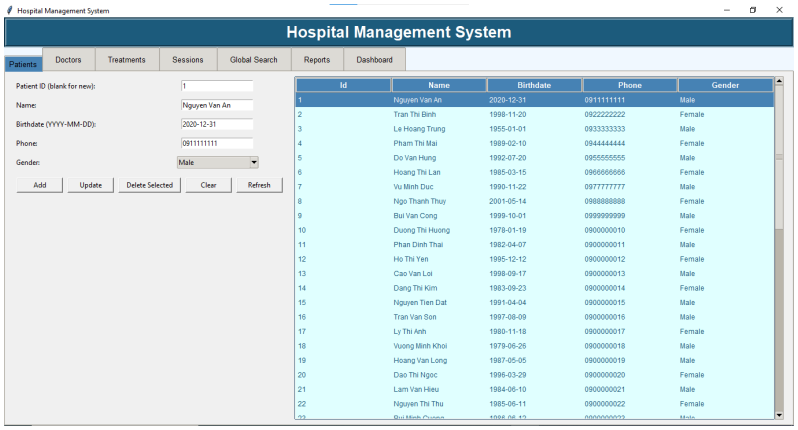

Figure 6.2: Patients

6.1.2 Doctors

The **Doctors** interface enables full CRUD operations for managing doctor records in the hospital system. The layout consists of a **left-side** input form and a **right-side** table listing all existing doctors. The interface uses dropdown menus for fields that must follow predefined values to ensure data consistency.

Main Features

Create (Add): Users can register a new doctor by entering the Name and Phone, and selecting Specialty and Gender from dropdown lists. The Doctor ID field is left blank for new entries and is automatically assigned by the system or database.

Read (View): The table on the right displays all doctors with the following columns:Id, Name, Specialty, Phone, Gender.The table supports vertical scrolling to browse longer lists.

Update: Clicking a doctor in the table automatically fills the form with existing information. Users can modify the details and select updated values from the Specialty and Gender dropdowns, then click Update to commit the changes to the database.

Delete: The Delete Selected button removes the chosen doctor record. Foreign key constraints prevent deleting a doctor who is referenced in existing treatment sessions.

Clear Refresh:Clear resets all form fields.

Refresh reloads the latest doctor list from the database.

Data Validation

Specialty must be selected from a predefined dropdown list (e.g., Cardiology, Pediatrics).

Gender is selected from a dropdown to avoid invalid or inconsistent entries.

Phone numbers must be valid and non-empty.

Manual Doctor ID entry is allowed but checked against PK duplication rules.

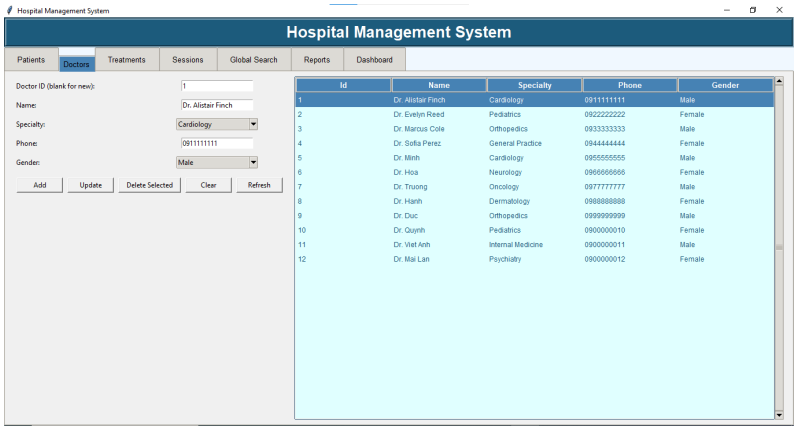

Figure 6.3: Doctors

6.1.3 Treatments

The **Treatments** interface manages the list of available medical procedures and their corresponding standard costs. This screen provides a clean and straightforward CRUD workflow, with a **data-entry form** on the left and a **treatment list table** on the right.

Main Features

Create (Add):Users can register a new treatment by entering the Name and Cost.The Treatment ID field remains blank for new entries and is assigned automatically by the database.

Read (View): The table on the right displays all treatment records with the following fields: Id, Name, Cost. Costs are represented as numeric values (float).

Update: Selecting a treatment in the table automatically fills the form with its details. Users can edit the Name and Cost, then click Update to save modifications.

Delete:The Delete Selected button removes the selected treatment. If the treatment is referenced in Treatment Sessions, database constraints prevent deletion to maintain referential integrity.

Clear Refresh: Clear resets all input fields. Refresh reloads the treatment list from the database.

Data Validation

Name cannot be empty.

Cost must be a positive numeric value (supports decimals).

Manual Treatment ID entry is allowed but checked against primary key constraints to avoid duplication.

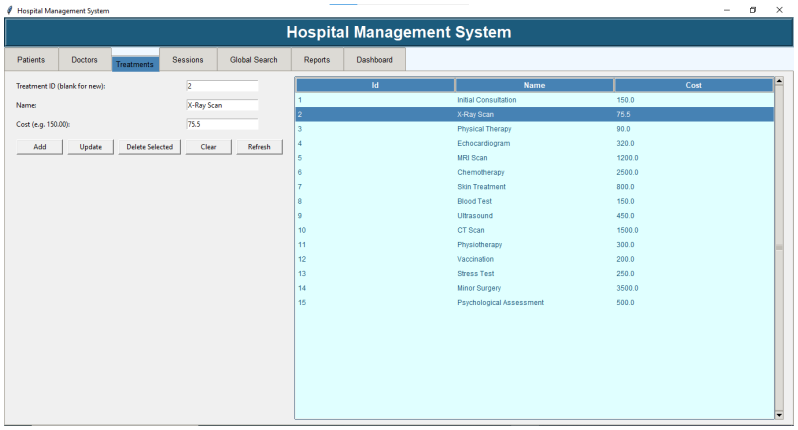

Figure 6.4: Treatment

6.1.4 Treatment Sessions

The **Treatment Sessions** interface manages the record of medical sessions, linking patients, doctors, and treatments together with a specific treatment date. This module ensures that every treatment event is stored as a separate session. The interface includes a **data-entry form** on the left and a **scrollable session table** on the right.

Main Features

Create (Add):Users can create a new treatment session by selecting:Patient (dropdown list with ID + name), Doctor (dropdown list), Treatment (dropdown list) and entering the Date in YYYY-MM-DD format. The Session ID field stays blank for new entries.

Read (View):The table on the right displays all existing sessions with the fields: Id, Patient, Doctor, Treatment, Date. The list supports vertical scrolling to handle large datasets.

Update:Selecting a row loads the session information into the form. Users may change the patient, doctor, treatment, or date, then click Update to save the new values.

Delete: The Delete Selected button removes the selected session record. Foreign key constraints ensure that referenced patients/doctors/treatments remain valid.

Clear Refresh: **Clear** resets the form fields. **Refresh** reloads all session records from the database.

Data Validation

Patient, doctor, and treatment must be selected from their respective dropdown menus.

Date must follow the YYYY-MM-DD format.

The system prevents inserting logically invalid sessions (e.g., missing selections).

Figure 6.5: Session

6.2 Search Filter Mechanisms

The system provides integrated search and filtering capabilities to help users quickly access essential information during hospital operations. The **Global Search** function allows querying across multiple entities without navigating through individual CRUD screens.

Global Search

Global Search is presented as a dedicated tab within the Sessions section. Users can enter a keyword, and the system displays all treatment sessions related to that keyword. The search covers:

- Patient names
- Doctor names
- Treatment names

Results are shown in a table that includes details such as session ID, patient name, doctor name, treatment name, cost, and treatment date.

User-Level Operation

When a keyword is entered, the system:

- Searches for matching sessions across all supported information fields
- Displays all relevant records in a unified results table
- Provides both vertical and horizontal scrolling for easier data navigation

Benefits

- Speeds up information retrieval
- Reduces the need to switch between multiple management screens
- Useful for patient reception, record verification, and quick history lookup
- Ensures users always have a comprehensive view of related treatment data

Figure 6.6: global search

6.3 Required Analytical Queries

The system provides four analytical report types, organized under the **Reports** tab, enabling staff to extract consolidated information for operational review and decision-making. Each report represents a common data-analysis scenario in hospital workflows, such as treatment tracking, identifying untreated patients, and highlighting high-cost procedures.

Inner Join Report: Sessions by Patient/Treatment

This report displays treatment sessions only for patients who have actually undergone procedures. The results include patient name, treatment type, date performed, and cost. It helps track completed treatment histories.

	A	B	C	D
1	Patient	Treatment	Date	Cost
2	Hoang Van	Skin Treatn	12/30/2050	800
3	Robert Bro	X-Ray Sca	11/10/2024	75.5
4	John Doe	Echocardio	10/15/2024	320
5	Patient 28	X-Ray Sca	1/29/2024	75.5
6	Patient 27	Vaccinatior	1/28/2024	200
7	Patient 26	CT Scan	1/27/2024	1500
8	Patient 25	Ultrasound	1/26/2024	450
9	Patient 24	MRI Scan	1/25/2024	1200
10	Patient 23	Skin Treatn	1/24/2024	800
11	Patient 22	Chemother	1/23/2024	2500

Figure 6.7: Inner Join query result.

Left Join Report: All Patients (with/without Sessions)

This report lists every patient in the system, including those who have not had any treatment sessions. Patients without sessions appear with empty treatment-related fields. The report assists the hospital in identifying individuals who have not yet been treated or whose appointments are pending.

	A	B	C	D
1	Patient	Treatment	Date	Cost
2	John Doe	None	10/1/2024	None
3	John Doe	Echocardio	10/15/2024	320
4	Jane Smith	None	11/5/2024	None
5	Robert Bro	X-Ray Sca	11/10/2024	75.5
6	Nguyen Va	MRI Scan	1/5/2024	1200
7	Tran Thi B	X-Ray Sca	1/6/2024	75.5
8	Le Van C	Physical Th	1/7/2024	90
9	Pham Thi D	None	1/8/2024	None
10	Hoang Van	Skin Treatn	12/30/2050	800
11	Ngo Thi F	Vaccinatior	1/10/2024	200

Figure 6.8: Left Join query result.

Multi-Table Join Report: Sessions (Patients, Doctors, Treatments)
This report provides a full view of each treatment session by combining information from patients, doctors, and treatments. It offers a complete picture of who received treatment, which doctor was involved, and when the session took place.

	A	B	C	D	E
1	Patient	Doctor	Treatment	Date	Cost
2	Hoang Van	Dr. Minh	Skin Treatn	12/30/2050	800
3	Robert Bro	Dr. Marcus	X-Ray Sca	11/10/2024	75.5
4	John Doe	Dr. Alistair	Echocardio	10/15/2024	320
5	Patient 28	Dr. Minh	X-Ray Sca	1/29/2024	75.5
6	Patient 27	Dr. Sofia P	Vaccinatior	1/28/2024	200
7	Patient 26	Dr. Marcus	CT Scan	1/27/2024	1500
8	Patient 25	Dr. Evelyn	Ultrasound	1/26/2024	450
9	Patient 24	Dr. Alistair	MRI Scan	1/25/2024	1200
10	Patient 23	Dr. Quynh	Skin Treatn	1/24/2024	800
11	Patient 22	Dr. Duc	Chemother	1/23/2024	2500

Figure 6.9: Multi-table Join result

Subquery/Aggregation Report: High-cost Treatments Patients
This report highlights treatments whose cost exceeds the overall average. It also lists the patients who received these high-cost procedures. The report is useful for financial monitoring, cost-trend evaluation, and budgeting considerations.

	A	B	C	D
1	Treatment	Cost	Patient	
2	Chemother	2500	Bui Thi H	
3	Chemother	2500	Patient 22	
4	CT Scan	1500	Vo Van I	
5	CT Scan	1500	Patient 26	
6	MRI Scan	1200	Nguyen Van A	
7	MRI Scan	1200	Tran Thi L	
8	MRI Scan	1200	Patient 24	
9	Skin Treatn	800	Hoang Van E	
10	Skin Treatn	800	Patient 23	

Figure 6.10: High-cost treatments query result.

Export Report CSV
All reports can be exported to CSV files, enabling offline storage, sharing, and extended analysis outside the application.

Figure 6.11: Report

6.4 Dashboard Visualizations

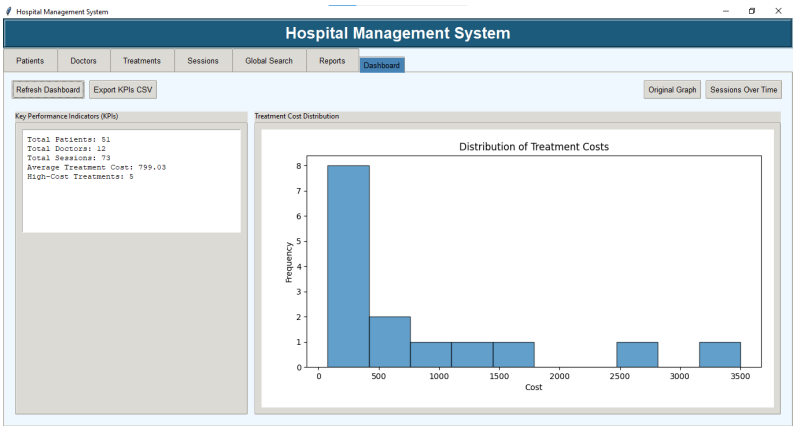

Figure 6.12: KPIs and Treatment Cost Distribution

Figure shows the Dashboard in its default mode, displaying the distribution of treatment costs. On the left, the **Key Performance Indicators (KPIs)** provide essential metrics such as total patients, total doctors, total sessions, average treatment cost, and the number of high-cost treatments. On the right, the histogram visualizes how treatment costs are distributed across the dataset, helping users identify common cost ranges and outliers. This mode gives an immediate overview of treatment cost patterns within the hospital.

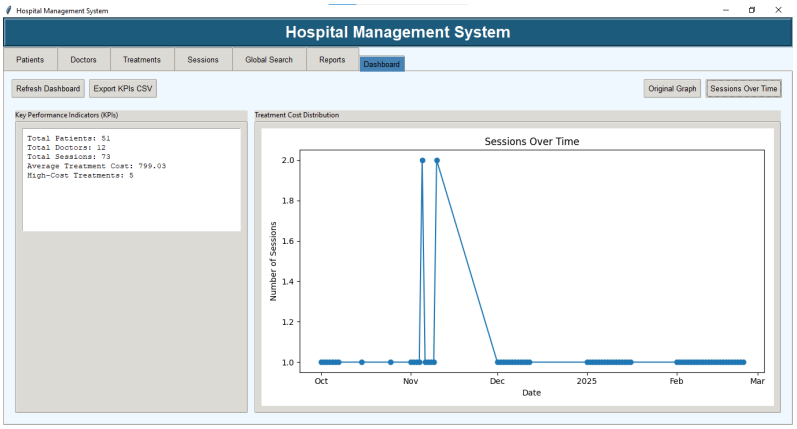

Figure 6.13: Sessions Over Time

Figure illustrates the Dashboard in **Sessions Over Time** mode. The line chart displays the number of treatment sessions according to date, allowing users to track activity trends, detect seasonal fluctuations, and understand workload changes over time. The KPI panel remains visible on the left to ensure that users retain access to essential high-level metrics while analyzing the time-based chart.

Dashboard Functional Buttons

Refresh Dashboard: Reloads the KPIs and charts with the most recent data.

Export KPIs CSV: Exports the KPI values to a CSV file.

Original Graph: Returns to the default treatment cost distribution view.

Sessions Over Time: Switches the Dashboard to the time-based session chart.

Chapter 7

Testing and Reliability

7.1 Input Validation

Ensuring data integrity is a core requirement of the Hospital Patient Manager system. Validation mechanisms were implemented across all input forms to prevent incorrect or incomplete data.

Field-Level Validation

Required fields: Key fields such as Patient Name, Doctor Name, Treatment Name, Cost, and Treatment Date cannot be left empty.

Date validation: Treatment dates must follow a valid calendar format and cannot be unrealistically far in the future. Invalid dates trigger descriptive warning messages.

Numeric constraints: Treatment cost must be a positive numerical value. Non-numeric or negative values are blocked.

Dropdown selections: Doctor and Treatment fields must be selected from predefined lists retrieved from the database.

Relational Validation

Duplicate prevention: Duplicate PatientID, DoctorID, or TreatmentID. Duplicate logical session (Patient + Treatment + Date)

Foreign key enforcement: Treatment sessions must reference existing patients, doctors, and treatments. Records cannot be deleted if they are still referenced elsewhere (e.g., a doctor linked to sessions)

These validation rules guarantee consistency, prevent anomalies, and maintain the reliability of the underlying relational model.

7.2 Error Handling

The application incorporates structured and user-friendly error handling to improve overall reliability.

Input Errors

If the user enters invalid or incomplete information, the system stops the action immediately.

No results or data are shown when the input does not meet the required format.

Database Errors

Connection failures, constraint violations, and foreign-key errors are intercepted and displayed through readable dialog messages.

Instead of terminating abruptly, the application safely returns control to the user and keeps the interface responsive.

Unhandled Exceptions

Unexpected system errors are automatically logged into a local log file.

These logs provide developers with diagnostic information without exposing technical details to the end user.

Fail-Safe Behavior

If the database becomes unreachable, the system notifies the user and pauses critical operations while keeping the program stable.

The user can retry or exit gracefully without the application freezing.

This approach ensures reliability and offers a consistent, predictable user experience.

7.3 Sample Test Scenarios

Entering an incorrect date when creating a patient

Expected: Display a warning; do not display data, do not save the record.

Entering alphabetic characters into the treatment cost field

Expected: Warning; do not display results, do not allow saving.

Running a query with an empty or invalid keyword

Expected: Empty result table, message “No valid results.”

Chapter 8

GitHub Workflow Collaboration

8.1 Repository Structure

The project is hosted on GitHub under the repository Hospital-Patient-Manager 1.
The repository is organized to separate application logic, database scripts, environment configuration, and documentation. The main directory structure is as follows:

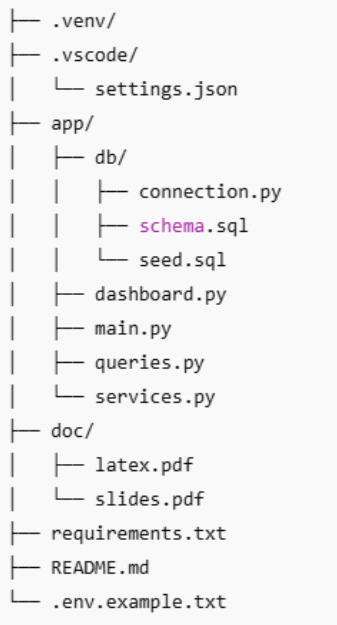

Figure 8.1: Repository Structure

The **app/** folder contains all Python source code, including the Tkinter GUI, CRUD operations, analytical queries, and dashboard visualizations.
The **app/db/** subfolder stores database-related files such as the schema, seed data, and the database connection module.
The **doc/** directory contains the final LaTeX report and slide deck in PDF format.
Additional files such as **requirements.txt** and .**env.example.txt** support deployment and configuration.
The **README.md** file provides usage instructions, project overview, and links to the demo video.
This structure ensures clarity, maintainability, and ease of navigation throughout development.

8.2 Workflow and Collaboration Practices

The team adopted a practical workflow combining GitHub version control with direct file sharing among members.
Each member worked on separate files or features and shared progress with the group before pushing changes to the repository.

Branch Usage
The team actively used Git branches throughout development. Each feature, bug fix, or UI update was implemented in a dedicated branch before being merged into the main branch. This workflow minimized merge conflicts and allowed members to work independently without interfering with each other's progress.

Issues
GitHub Issues were used to track tasks, feature requests, and identified bugs. Each issue documented the task description, assigned member, and progress notes. Using Issues helped maintain transparency and ensured that all required project components were completed systematically.

Pull Requests
Pull Requests were consistently used when merging branches into the main branch. Each PR included a summary of changes, screenshots when relevant, and references to related Issues. Team members reviewed PRs before approval, ensuring code quality and preventing accidental errors from being pushed to the main branch.

Version Tags
GitHub release tags were created to mark significant milestones in the project's development cycle, such as prototype versions, testing builds, and the final submission release. These tags allow the team to track stable checkpoints and easily revert or compare changes between different versions if necessary.

Commit Messages
Commit messages followed a consistent, structured convention such as *feat*, *fix*, *update*, and *refactor*. This standardized format improved readability, helped track the purpose of each change, and supported clean version-control practices throughout the development process.

8.3 Team Roles and Contributions

Workload distribution was based on individual strengths and ensured that all project requirements were adequately addressed:

Le Sy Huy
Managed the GitHub repository
Developed the main application (CRUD logic, database connection, and overall functionality)
Compiled and formatted the LaTeX report

Be Thanh Đat
Developed major components of the application
Contributed significantly to GUI construction and application logic
Assisted in refining and testing the overall system

Khuat Đinh Trung
Participated in GUI development (layout structuring, interface refinement)

Assisted with editing, polishing the report, and testing the application

Created the project slides, recorded the demo video, and uploaded presentation materials to YouTube

The division of responsibilities reflects a balanced contribution among team members, enabling efficient collaboration and timely project completion.

Chapter 9

Results Discussion

The Hospital Patient Manager system successfully integrates database design, SQL implementation, and a Python-based GUI into a consistent application workflow.

The transformation from raw unnormalized data into a fully normalized 3NF relational schema eliminates redundancy and ensures data integrity across all entities.

CRUD operations for Patients, Doctors, Treatments, and Sessions function reliably with strong validation and foreign-key enforcement. Analytical SQL queries provide meaningful insights, while the dashboard visualizations display key metrics such as treatment counts, doctor workload, and cost summaries.

Overall, the system demonstrates a complete and functional pipeline—from data modeling to interface design and analytical reporting—meeting all project requirements.

Chapter 10

Limitations Future Work

10.1 Current Limitations

No user authentication: The system operates openly without login roles, making it unsuitable for real medical environments.
Basic dashboard visualizations: Charts are simple and not interactive.
Limited treatment cost model: No support for history tracking, price changes, or multi-tier pricing.
GUI limitations: Tkinter is functional but lacks modern UI capabilities.

10.2 Future Enhancements

User roles and authentication for doctors, staff, and administrators.
Audit logs and temporal data to track changes over time.
More advanced analytics, such as treatment frequency forecasting or doctor workload prediction.
Web-based deployment using Flask/FastAPI for better scalability.
Automated unit and integration testing using GitHub Actions.

Chapter 11

Conclusion

The Hospital Patient Manager project demonstrates the full development cycle of a data-driven application, integrating database theory, SQL implementation, and GUI programming into a cohesive system. Starting from an unnormalized dataset, the project successfully transforms the data into a fully normalized 3NF schema, ensuring integrity, reducing redundancy, and enabling reliable analytical operations.

The MySQL backend combined with the Python/Tkinter interface provides complete CRUD functionality, robust validation, multi-table analytical queries, and dynamic dashboard visualizations. These components collectively support intuitive data management and real-time insight generation for patients, doctors, treatments, and treatment sessions.

Modern software engineering practices—such as GitHub version control, feature-branch workflow, issue tracking, and tagged releases—were applied consistently throughout development. This reinforces maintainability, clarity, and collaboration.

Overall, the system fulfills all functional and non-functional requirements while establishing a solid foundation for future enhancements including authentication, web deployment, interactive dashboards, audit logging, and advanced analytics. The project highlights strong competency in relational design, application development, and practical implementation of database-driven systems. As explained in [1], normalization reduces redundancy.

Bibliography

[1] Silberschatz, A., Korth, H. F., Sudarshan, S. *Database System Concepts*. McGraw-Hill.

[2] Connolly, T., Begg, C. *Database Systems: A Practical Approach to Design, Implementation, and Management*. Pearson.

[3] MySQL Documentation — `https://dev.mysql.com/doc/`

[4] Python Official Documentation — `https://docs.python.org/`

[5] Tkinter Programming Guide — `https://tkdocs.com/`

[6] Matplotlib Documentation — `https://matplotlib.org/`

[7] GitHub Documentation — `https://docs.github.com/`