National Economics University

School of Advanced Education Program

Hospital Patient Manager

11247306 - Nguyen Xuan Kiet - Leader
11247290 - Nguyen Phung Hoa
11247322 - Nguyen Huy Minh
11247286 - Hoang Thi Ngoc Han

Supervisor: Tran Duc Minh

A report submitted in partial fulfilment of the requirements of
the University of Reading for the degree of
Master of Science in *Data Science and Advanced Computing*

December 6, 2025

Abstract

In healthcare data management, storing records in unnormalized forms often leads to data redundancy and update anomalies. This project addresses these issues by designing and developing the **“Hospital Patient Manager,”** a comprehensive system to streamline the administration of patients, doctors, and treatment sessions. The methodology involved **normalizing** raw hospital data from an Unnormalized Form (UNF) to the **Third Normal Form (3NF)** to ensure data integrity. A **relational database** was implemented using **MySQL**, serving as the foundation for a web application built with **Python Flask** and **Bootstrap 5** following the **Model-View-Controller (MVC)** architecture. Key features include full **CRUD** functionality, dynamic session scheduling, and real-time dashboards. The results demonstrate a robust system that successfully eliminates data duplication and visualizes critical **Key Performance Indicators (KPIs)** through interactive charts. Furthermore, complex **SQL queries** were executed to generate operational reports, such as high-cost treatment analysis and multi-table joins, allowing for efficient data export and review. In conclusion, the project validates the importance of database normalization in preventing anomalies and highlights the effectiveness of combining MySQL with Python Flask to create scalable, user-friendly solutions for hospital information management.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

In the modern healthcare sector, the **efficient management of information** is critical for operational success and patient care. Hospitals generate vast amounts of data daily, ranging from patient demographic details and medical history to doctor schedules and treatment records. Traditionally, or in legacy systems, this data might be stored in flat files, spreadsheets, or **unnormalized database tables**. Such methods, while simple to initiate, often lead to significant **data integrity issues** as the volume of information grows.

Relational Database Management Systems (RDBMS) combined with **normalization theories** provide a mathematical framework to organize data efficiently. By structuring data into related tables, organizations can minimize **redundancy** and ensure **consistency**. Furthermore, combining a robust backend database with a user-friendly **Graphical User Interface (GUI)** allows non-technical staff to interact with the data effectively, bridging the gap between raw data storage and practical usability. This project focuses on the transition from an unmanageable data structure to a streamlined, computerized hospital management system.

1.2 Problem statement

The specific problem addressed in this project stems from a hospital currently storing its records in a single **Unnormalized Form (UNF)** table. This table aggregates all information regarding patients, doctors, treatments, and session dates into one flat structure.

This architecture presents several critical issues:

- **Data Redundancy:** Information such as doctor names and patient details is repeated for every treatment session, wasting storage space.
- **Update Anomalies:** Modifying a doctor's details (e.g., a name change) requires updating every single row where that doctor appears, carrying the risk of inconsistent data.
- **Insertion Anomalies:** It is impossible to record a new doctor or patient in the system without them undergoing a treatment session, as the primary key structure of the UNF table depends on transaction data.
- **Deletion Anomalies:** Deleting a specific treatment record might unintentionally result in the permanent loss of a patient's or doctor's master data.

Furthermore, without a dedicated application interface, hospital staff must interact directly with the database or spreadsheet, which is **inefficient and prone to human error**.

1.3 Aims and objectives

Aims

The primary aim of this project is to **design and develop** "Hospital Patient Manager," a robust software solution that **normalizes the hospital's data architecture to the Third Normal Form (3NF)** and provides an **interactive web-based interface** for managing daily operations.

Objectives

To achieve the stated aim, the following specific objectives have been defined:

1. **Data Modeling:** Analyze the provided Functional Dependencies (FDs) to transform the UNF table into a normalized **3NF schema**, identifying appropriate Primary Keys (PK) and Foreign Keys (FK).

2. **Database Implementation:** Construct the relational schema in **MySQL** and populate it with realistic seed data to simulate a working environment.

3. **Application Development:** Develop a Python-based application (using the **Flask framework**) that connects to the MySQL database to perform **Create, Read, Update, and Delete (CRUD)** operations.

4. **Reporting and Visualization:** Implement complex **SQL queries** (Inner Join, Left Join, Subqueries) to generate operational reports and a **visual dashboard** for monitoring Key Performance Indicators (KPIs).

1.4 Solution approach

The methodology adopted for this project follows the **Database Development Life Cycle (DDLC)** integrated with **agile software development principles**.

Theoretical Analysis

The project begins with a rigorous **normalization process**. By applying the rules of **1NF**, **2NF**, and **3NF**, the monolithic UNF table is decomposed into four distinct entities: **Patients, Doctors, Treatments**, and **TreatmentSessions**.

System Design

An **Entity-Relationship Diagram (ERD)** is modeled to visualize the relationships between these entities.

Implementation Stack

- **Database: MySQL** is selected for its reliability and support for relational constraints.
- **Backend: Python** is used for logic processing, leveraging the **Flask micro-framework** for handling HTTP requests and routing.
- **Frontend: HTML5** and **Bootstrap 5** are utilized to create a responsive and accessible user interface.
- **Connectivity:** The `mysql-connector-python` library serves as the bridge between the application layer and the data layer.

1.5 Summary of contributions and achievements

This project delivers a fully functional hospital management system that addresses the initial data anomalies. The major contributions include:

- A verified **3NF Database Schema** that eliminates data redundancy and enforces referential integrity.
- A **Web Application** featuring a **real-time Dashboard** that visualizes total patients, doctors, and revenue metrics.
- A suite of **SQL-driven Reports**, including functionality to identify high-cost treatments and export data to `CSV` format for external analysis.
- A secure and user-friendly interface that abstracts the complexity of `SQL` from the end-user, allowing for efficient data entry and retrieval.

Chapter 2

Database Design

This chapter presents the detailed design of the database architecture for the Hospital Patient Manager system. It begins with an analysis of the initial Unnormalized Form (UNF) data structure and the identification of key Functional Dependencies (FDs) based on business rules. A rigorous normalization process is then documented, demonstrating the transformation of data from UNF through to the Third Normal Form (3NF) to eliminate redundancy and prevent data anomalies. Finally, the chapter defines the logical data model via an Entity-Relationship Diagram (ERD) and specifies the physical schema implemented in the MySQL database.

2.1 Unnormalized Form (UNF)

The project begins with a dataset representing hospital operations stored in a single, flat structure (**Unnormalized Form** - UNF). This structure captures the basic transaction details of patient visits but lacks relational organization.

Attributes of UNF

The initial dataset contains the following attributes:

- `PatientID`, `PatientName`
- `DoctorID`, `DoctorName`
- `TreatmentID`, `TreatmentName`, `Cost`
- `TreatmentDate`

Anomalies in UNF

Storing data in this manner leads to several anomalies:

1. **Redundancy:** Patient and Doctor details are repeated for every single treatment session.
2. **Update Anomaly:** Updating a doctor's name requires changes in multiple rows, potentially leading to data inconsistency.
3. **Insertion Anomaly:** A new doctor cannot be added to the system until they perform a treatment, as the primary key is composite and transaction-dependent.

2.2 Functional Dependencies (FDs)

Based on the analysis of the business rules and data semantics, the following **Functional Dependencies (FDs)** have been identified:

- **FD1:** $PatientID \to PatientName$ (A patient's name is uniquely determined by their ID).
- **FD2:** $DoctorID \to DoctorName$ (A doctor's name is uniquely determined by their ID).
- **FD3:** $TreatmentID \to \{TreatmentName, Cost\}$ (The name and standard cost of a treatment depend solely on the treatment ID).
- **FD4:** $\{PatientID, DoctorID, TreatmentID\} \to TreatmentDate$ (The date of a specific session is determined by the combination of the patient, doctor, and the treatment performed).

2.3 Normalization Process

2.3.1 First Normal Form (1NF)

To achieve **1NF**, we ensure that all attributes contain atomic values and there are no repeating groups. The provided UNF dataset is already in a tabular format where each cell contains a single value (e.g., a single patient name, a single date). Therefore, the relation is considered to be in **1NF**.

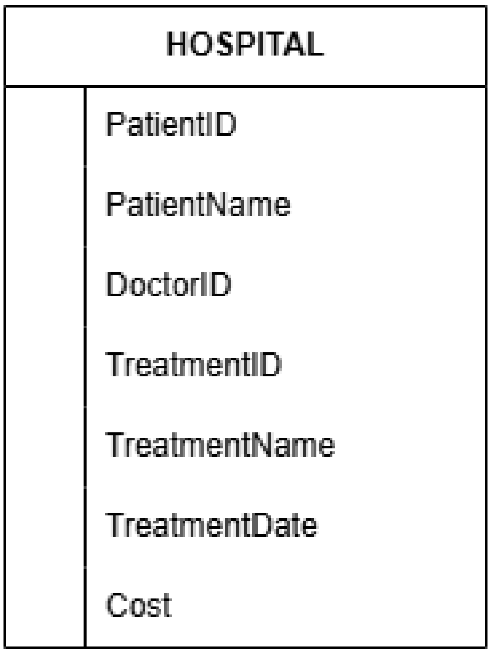


Figure 2.1: 1NF Table

2.3.2 Second Normal Form (2NF)

A relation is in **2NF** if it is in **1NF** and contains no **Partial Dependencies**. A partial dependency occurs when a non-prime attribute depends on only a part of a composite primary key.

Analyzing our FDs against the composite key of the transaction table:

- `PatientName` depends only on `PatientID`, not the full transaction key. → **Decompose to table Patients**.
- `DoctorName` depends only on `DoctorID`. → **Decompose to table Doctors**.
- `TreatmentName` and `Cost` depend only on `TreatmentID`. → **Decompose to table Treatments**.

Resulting Relations in 2NF

- **Patients:** (PatientID, PatientName)
- **Doctors:** (DoctorID, DoctorName)
- **Treatments:** (TreatmentID, TreatmentName, Cost)
- **Sessions:** (PatientID, DoctorID, TreatmentID, TreatmentDate)

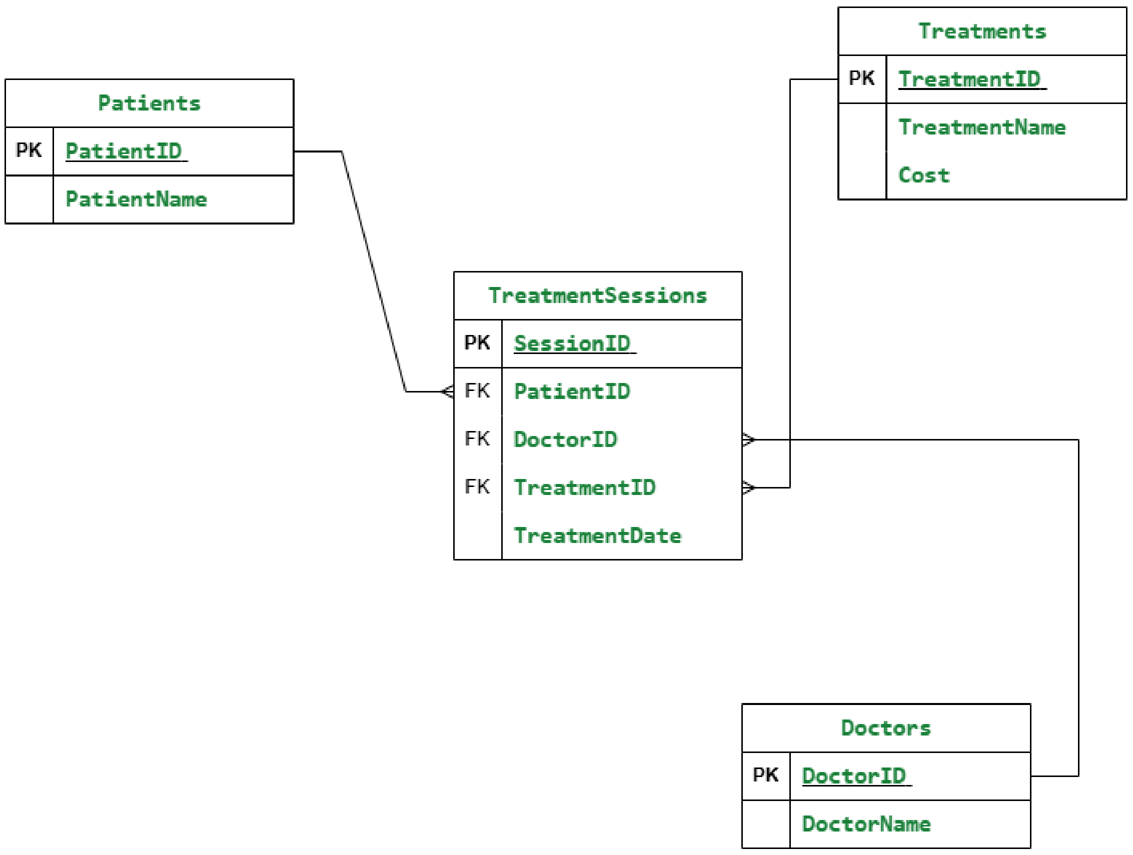


Figure 2.2: 2NF Table

2.3.3 Third Normal Form (3NF)

A relation is in **3NF** if it is in **2NF** and contains no **Transitive Dependencies** (attributes depending on other non-prime attributes).

Upon reviewing the 2NF relations:

- In the **Patients**, **Doctors**, and **Treatments** tables, all non-key attributes depend directly on the Primary Key.
- In the **Sessions** table, the non-key attribute `TreatmentDate` depends directly on the composite foreign keys.

No transitive dependencies exist. Therefore, the schema is in **3NF**. To enhance data management and referencing, a surrogate key `SessionID` is added to the **Sessions** table (renamed to **TreatmentSessions**).

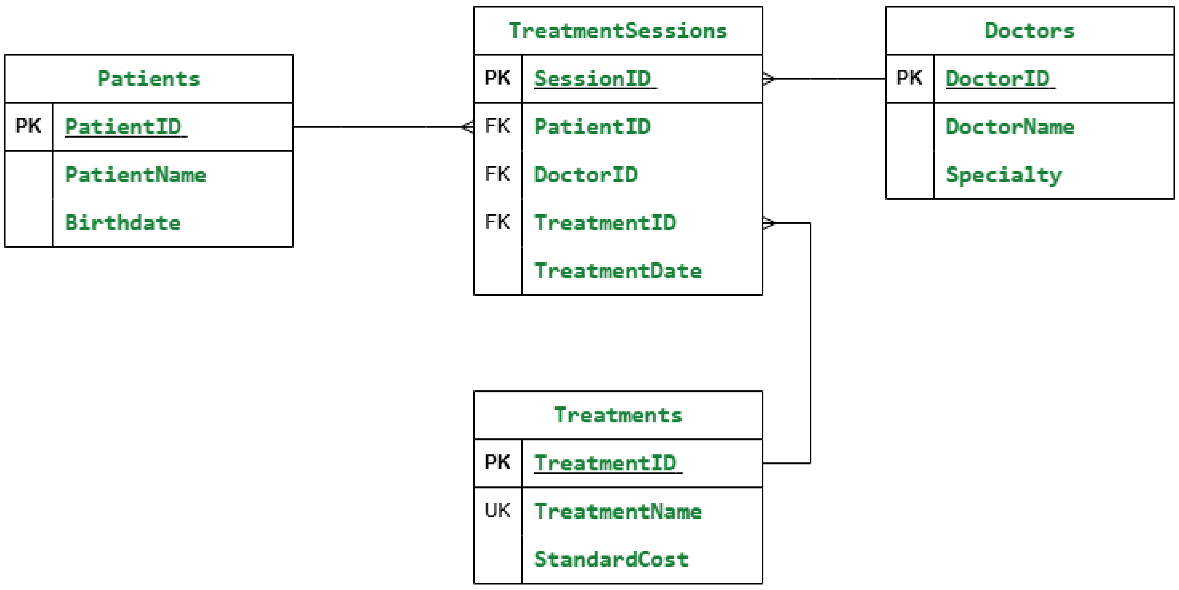


Figure 2.3: 3NF Table

2.4 Entity-Relationship Diagram (ERD)

The final logical model consists of three master entities (*Patients, Doctors, Treatments*) and one transactional entity (*TreatmentSessions*) that links them together.

Relationships

- **Doctor** - **Session:** One-to-Many (A doctor can perform multiple sessions).
- **Patient** - **Session:** One-to-Many (A patient can have multiple treatment sessions).
- **Treatment** - **Session:** One-to-Many (A specific treatment type can be performed many times).

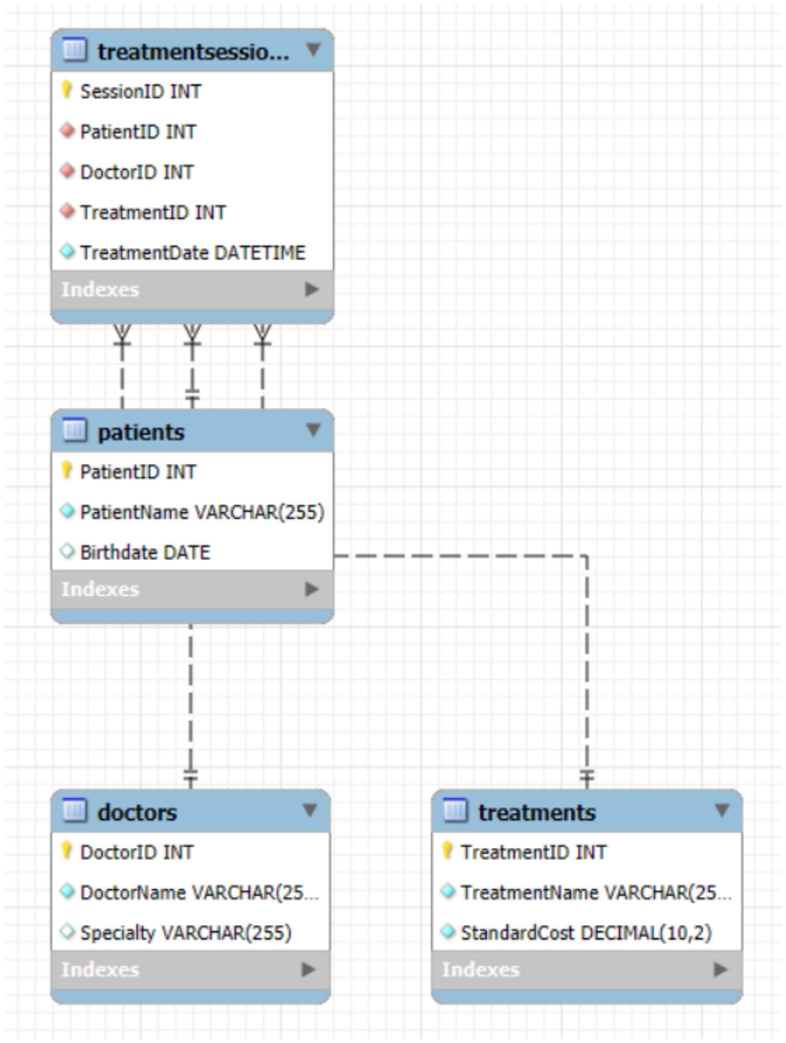


Figure 2.4: Entity-Relationship Diagram (ERD)

2.5 Database Schema

Table: Patients

Stores demographic information about patients.

Column	Type	Key	Description
PatientID	INT	PK	Auto-increment unique identifier
PatientName	VARCHAR(255)		Full name of the patient
Birthdate	DATE		Date of birth

Table 2.1: Schema for Patients Table

Table: Doctors

Stores details of medical staff.

Column	**Type**	**Key**	**Description**
DoctorID	INT	PK	Auto-increment unique identifier
DoctorName	VARCHAR(255)		Full name of the doctor
Specialty	VARCHAR(255)		Medical specialization

Table 2.2: Schema for Doctors Table

Table: Treatments

Stores the catalog of available medical services and their standard costs.

Column	**Type**	**Key**	**Description**
TreatmentID	INT	PK	Auto-increment unique identifier
TreatmentName	VARCHAR(255)		Name of the procedure (Unique)
StandardCost	DECIMAL(10,2)		Cost per unit

Table 2.3: Schema for Treatments Table

Table: TreatmentSessions

The associative table recording the event of a doctor treating a patient.

Column	**Type**	**Key**	**Description**
SessionID	INT	PK	Surrogate Primary Key
PatientID	INT	FK	Refers to Patients.PatientID
DoctorID	INT	FK	Refers to Doctors.DoctorID
TreatmentID	INT	FK	Refers to Treatments.TreatmentID
TreatmentDate	DATETIME		Timestamp of the session

Table 2.4: Schema for TreatmentSessions Table

2.6 Comparison of Normalization Forms

Table 2.5: Comparison of Normalization Forms

Criteria	**1NF**	**2NF**	**3NF**
Definition	Ensures atomic values and eliminates repeating groups.	Achieves 1NF and removes partial dependencies.	Achieves 2NF and removes transitive dependencies.
Requirement	A primary key is defined; entries in a column are of the same type.	Non-key attributes must depend on the *entire* primary key.	Non-key attributes must depend *only* on the primary key.
Problem Solved	Lack of structure; composite values.	Redundancy due to dependency on part of a composite key.	Redundancy due to dependencies between non-key attributes.
Context Example	Patient & Doctor names repeat for every session row.	Patient & Doctor tables are separated from the Session table.	Treatment Cost is moved to the Treatments table (depends on TreatmentID).

Chapter 3

Implementation

This chapter details the technical realization of the "Hospital Patient Manager" system. It outlines the software architecture, the technology stack employed, the organization of the project files, and the implementation details of both the backend logic and the frontend graphical user interface (GUI). The system is developed to be robust, secure, and user-friendly, strictly adhering to the 3NF database schema designed in the previous chapter.

3.1 System Architecture

The application follows the **Model-View-Controller (MVC)** architectural pattern, which separates the application logic into three interconnected elements:

- **Model (Data Layer):** Represents the data structure and database interactions. In this project, **MySQL** serves as the database, and Python modules (e.g., `patient_model.py`) act as the data access layer executing **CRUD** operations.
- **View (Presentation Layer):** The user interface where data is displayed. This is implemented using **HTML5 templates** rendered by the **Jinja2 engine** and styled with the **Bootstrap 5** framework.
- **Controller (Application Logic):** Handles user input and system logic. The **Flask framework** (`main.py`) manages routing, processes requests, and acts as the bridge between the Model and the View.

3.2 Technology Stack

Table 3.1: Project Technology Stack

Category	**Technology**	**Purpose**
Language	Python 3.x	Core programming language for backend logic.
Web Framework	Flask	Lightweight web framework for routing and HTTP handling.
Database	MySQL	Relational database management system (RDBMS).
Connector	`mysql-connector-python`	API for connecting Python with MySQL.
Frontend	Bootstrap 5	CSS framework for responsive and modern GUI design.
Templating	Jinja2	Engine for rendering dynamic data into HTML pages.
Visualization	Chart.js / Matplotlib	Library for rendering Dashboard charts.
Data Processing	Pandas	Used for generating and exporting CSV reports.

3.3 Project Structure

The source code is organized into a modular structure to enhance maintainability and scalability. The repository structure is as follows:

- `app/`: The main application directory.
  - `db/`: Contains database scripts (`schema.sql`, `seed.sql`) and the connection module.
  - `models/`: Contains Python modules for **CRUD** operations (`patient_model.py`, `doctor_model.py`, etc.).
  - `services/`: Contains business logic such as **KPI** calculations (`dashboard_service.py`).
  - `ui/`: Contains frontend assets (`templates/` for HTML and `static/` for CSS/JS).
  - `main.py`: The entry point of the **Flask** application defining all routes.
- `docs/`: Documentation files (Report and Slides).
- `.env`: Stores sensitive environment variables (DB credentials).
- `requirements.txt`: Lists all dependencies for easy installation.

3.4 Backend Implementation

Database Connection

Secure connection to the **MySQL** database is established using the `mysql-connector-python` library. To ensure **security and portability**, database credentials (host, user, password) are loaded from an **environment variable file** (`.env`) rather than being hard-coded into the source code. A dedicated module manages connection pooling and error handling.

Data Access Objects (DAO)

For each entity (`Patients`, `Doctors`, `Treatments`, `Sessions`), a specific model file was created. These files contain Python functions that execute parameterized `SQL` queries.

- **Prevention of SQL Injection: Parameterized queries** are used for all user inputs to prevent **SQL injection attacks**.
- **Transaction Management:** Operations that modify data (`INSERT`, `UPDATE`, `DELETE`) are explicitly committed to ensure data persistence.

Business Logic Services

Complex calculations, such as determining the "Average Treatment Cost" or counting "High-Cost Treatments" for the Dashboard, are encapsulated in the **service layer**. This separation ensures that the controller (`main.py`) remains clean and focused on routing.

3.5 Frontend Implementation (GUI)

Layout and Responsiveness

A base template (`layout.html`) creates a consistent look and feel across the application. It includes a navigation bar and a footer. All other pages inherit from this layout using **Jinja2's inheritance mechanism**. **Bootstrap 5 grid system** is used to ensure the application is **responsive** on different screen sizes.

Dynamic Forms and Tables

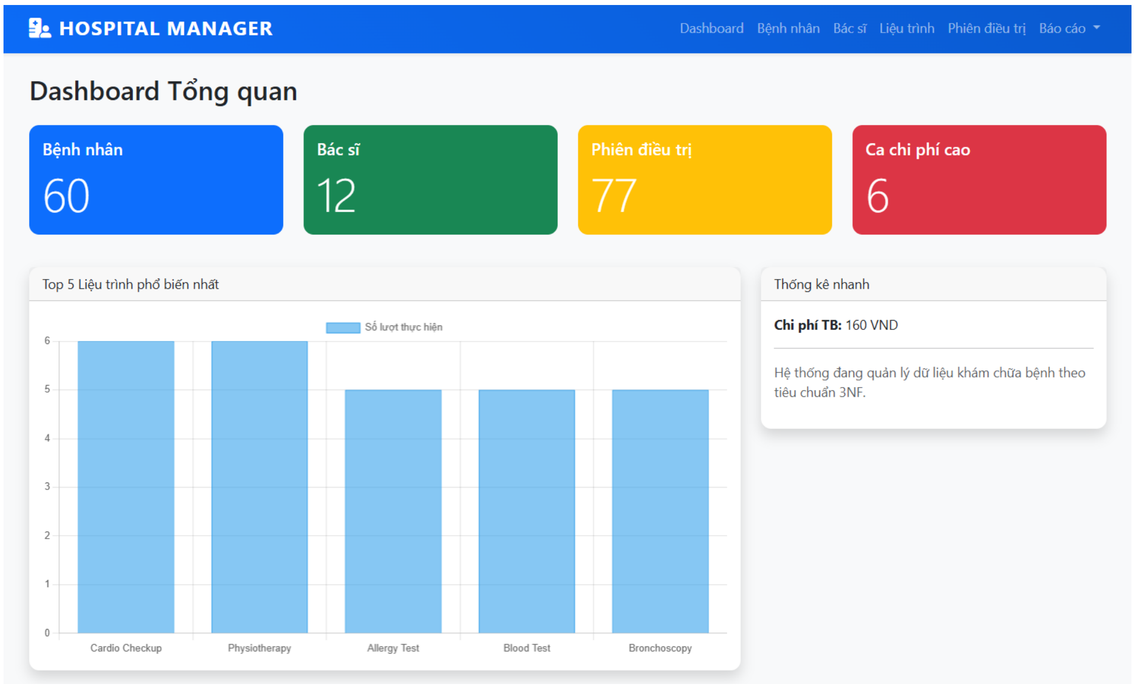
Data retrieved from the backend is rendered dynamically into **HTML tables**.

- **Session Management:** When creating a new treatment session, **dropdown menus are dynamically populated** with existing `Patients` and `Doctors` from the database, preventing foreign key errors.
- **Flash Messages:** The system provides immediate feedback to the user (e.g., "Patient added successfully" or "Error deleting record") using **Flask's flash messaging system**.

3.5.1 Key Features Description

1. Real-time Dashboard

The landing page features a dashboard that provides an instant overview of hospital operations. It displays **Key Performance Indicators (KPIs)** such as Total Patients, Total Doctors, and Total Sessions. A visual chart illustrates the distribution of treatment types or revenue trends.



Dashboard Tổng quan

Bệnh nhân 60

Bác sĩ 12

Phiên điều trị 77

Ca chi phí cao 6

Thống kê nhanh

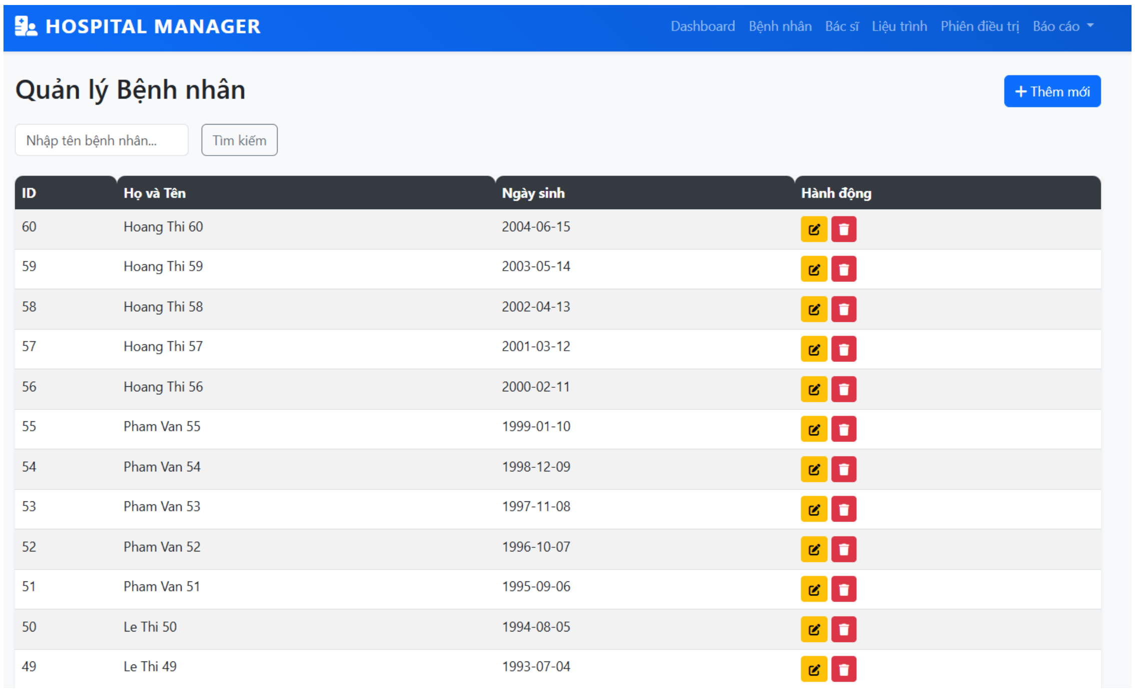
Chi phí TB: 160 VND

Hệ thống đang quản lý dữ liệu khám chữa bệnh theo tiêu chuẩn 3NF.

Figure 3.1: Application Dashboard with KPIs and Charts

2. CRUD Operations

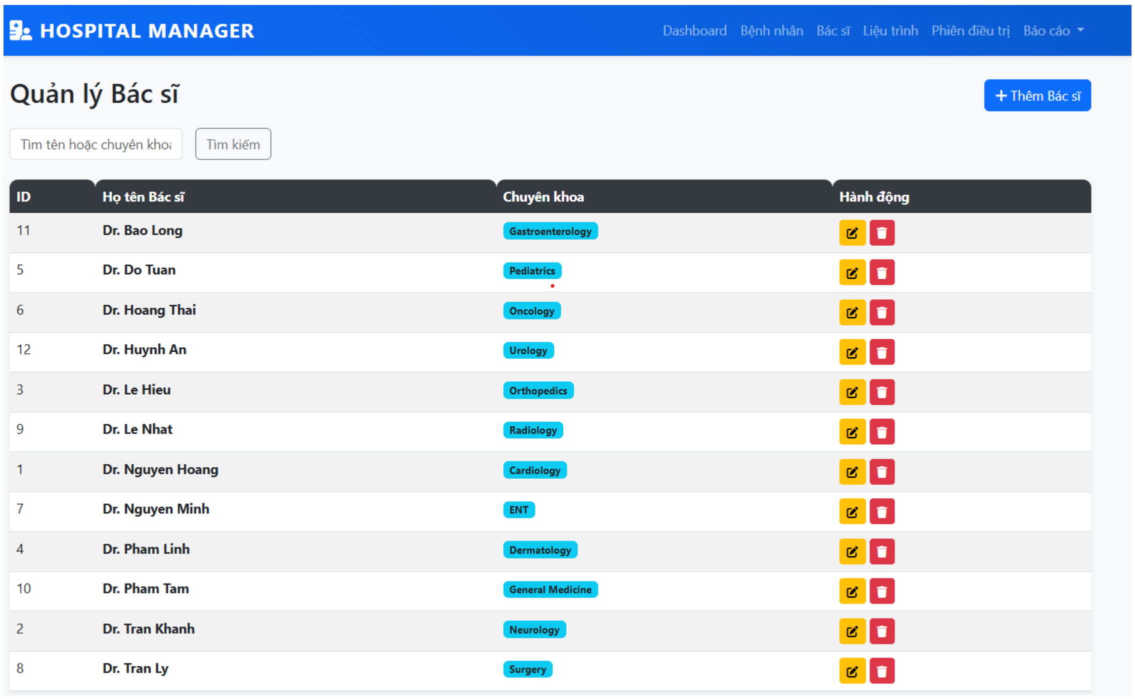
The system supports full **Create, Read, Update, and Delete (CRUD)** operations for all master data. **Input validation** is implemented both on the client-side (HTML5 required attributes) and server-side to ensure data integrity.



Quản lý Bệnh nhân

ID	Họ và Tên	Ngày sinh	Hành động
60	Hoang Thi 60	2004-06-15	✎ 🗑
59	Hoang Thi 59	2003-05-14	✎ 🗑
58	Hoang Thi 58	2002-04-13	✎ 🗑
57	Hoang Thi 57	2001-03-12	✎ 🗑
56	Hoang Thi 56	2000-02-11	✎ 🗑
55	Pham Van 55	1999-01-10	✎ 🗑
54	Pham Van 54	1998-12-09	✎ 🗑
53	Pham Van 53	1997-11-08	✎ 🗑
52	Pham Van 52	1996-10-07	✎ 🗑
51	Pham Van 51	1995-09-06	✎ 🗑
50	Le Thi 50	1994-08-05	✎ 🗑
49	Le Thi 49	1993-07-04	✎ 🗑

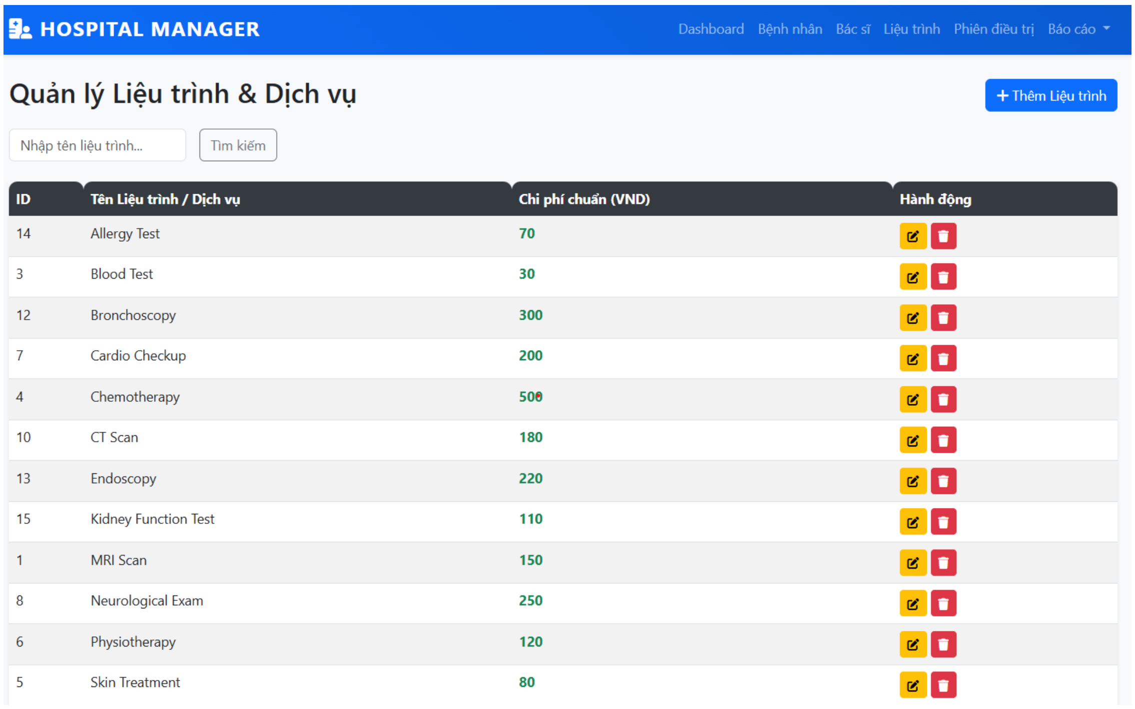
Figure 3.2: Patients Management Interface (List and Add Form)



Quản lý Bác sĩ

ID	Họ tên Bác sĩ	Chuyên khoa	Hành động
11	**Dr. Bao Long**	Gastroenterology	✎ 🗑
5	**Dr. Do Tuan**	Pediatrics	✎ 🗑
6	**Dr. Hoang Thai**	Oncology	✎ 🗑
12	**Dr. Huynh An**	Urology	✎ 🗑
3	**Dr. Le Hieu**	Orthopedics	✎ 🗑
9	**Dr. Le Nhat**	Radiology	✎ 🗑
1	**Dr. Nguyen Hoang**	Cardiology	✎ 🗑
7	**Dr. Nguyen Minh**	ENT	✎ 🗑
4	**Dr. Pham Linh**	Dermatology	✎ 🗑
10	**Dr. Pham Tam**	General Medicine	✎ 🗑
2	**Dr. Tran Khanh**	Neurology	✎ 🗑
8	**Dr. Tran Ly**	Surgery	✎ 🗑

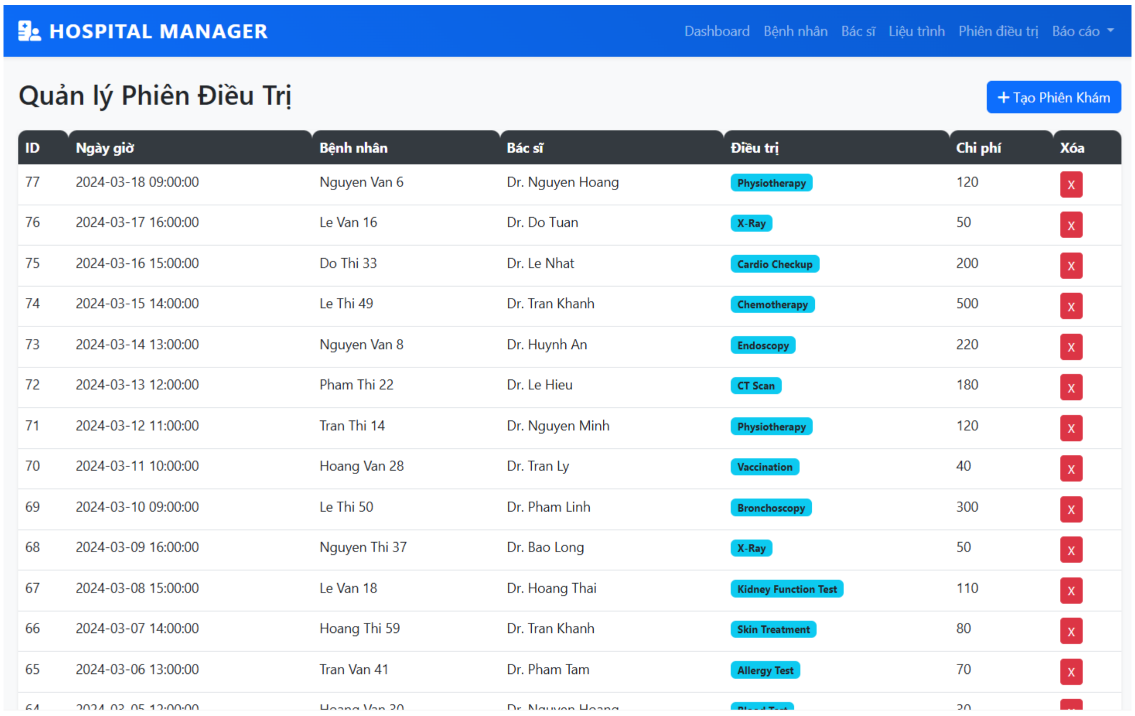
Figure 3.3: Doctors Management Interface (List and Add Form)



Quản lý Liệu trình & Dịch vụ

ID	Tên Liệu trình / Dịch vụ	Chi phí chuẩn (VND)	Hành động
14	Allergy Test	70	✎ 🗑
3	Blood Test	30	✎ 🗑
12	Bronchoscopy	300	✎ 🗑
7	Cardio Checkup	200	✎ 🗑
4	Chemotherapy	500	✎ 🗑
10	CT Scan	180	✎ 🗑
13	Endoscopy	220	✎ 🗑
15	Kidney Function Test	110	✎ 🗑
1	MRI Scan	150	✎ 🗑
8	Neurological Exam	250	✎ 🗑
6	Physiotherapy	120	✎ 🗑
5	Skin Treatment	80	✎ 🗑

Figure 3.4: Treatments Management Interface (List and Add Form)



Quản lý Phiên Điều Trị

ID	Ngày giờ	Bệnh nhân	Bác sĩ	Điều trị	Chi phí	Xóa
77	2024-03-18 09:00:00	Nguyen Van 6	Dr. Nguyen Hoang	Physiotherapy	120	X
76	2024-03-17 16:00:00	Le Van 16	Dr. Do Tuan	X-Ray	50	X
75	2024-03-16 15:00:00	Do Thi 33	Dr. Le Nhat	Cardio Checkup	200	X
74	2024-03-15 14:00:00	Le Thi 49	Dr. Tran Khanh	Chemotherapy	500	X
73	2024-03-14 13:00:00	Nguyen Van 8	Dr. Huynh An	Endoscopy	220	X
72	2024-03-13 12:00:00	Pham Thi 22	Dr. Le Hieu	CT Scan	180	X
71	2024-03-12 11:00:00	Tran Thi 14	Dr. Nguyen Minh	Physiotherapy	120	X
70	2024-03-11 10:00:00	Hoang Van 28	Dr. Tran Ly	Vaccination	40	X
69	2024-03-10 09:00:00	Le Thi 50	Dr. Pham Linh	Bronchoscopy	300	X
68	2024-03-09 16:00:00	Nguyen Thi 37	Dr. Bao Long	X-Ray	50	X
67	2024-03-08 15:00:00	Le Van 18	Dr. Hoang Thai	Kidney Function Test	110	X
66	2024-03-07 14:00:00	Hoang Thi 59	Dr. Tran Khanh	Skin Treatment	80	X
65	2024-03-06 13:00:00	Tran Van 41	Dr. Pham Tam	Allergy Test	70	X
64	2024-03-05 12:00:00	Hoang Van 30	Dr. Nguyen Hoang	Blood Test	30	X

Figure 3.5: Sessions Management Interface (List and Add Form)

3. Advanced Reporting and Export

The "Reports" module allows users to execute complex queries without writing SQL. It includes:

- **Inner Join Report:** Shows details of completed sessions.
- **Left Join Report:** Lists all patients, including those without treatment history.
- **High-Cost Analysis:** Filters treatments exceeding the average cost.

Users can export these report results to **CSV format** using the `Pandas` library for further analysis in Excel.

Chapter 4

Query Analysis

This chapter brings all the previous concepts together. We will synthesize the full address translation process, evaluate the strengths and weaknesses of the IA-32's hybrid memory model (Segmentation and Paging), and analyze the specific hardware mechanisms that make "Protected Mode" a secure environment.

4.1 Operational Report: Active Treatment Sessions (Inner Join)

4.1.1 Purpose

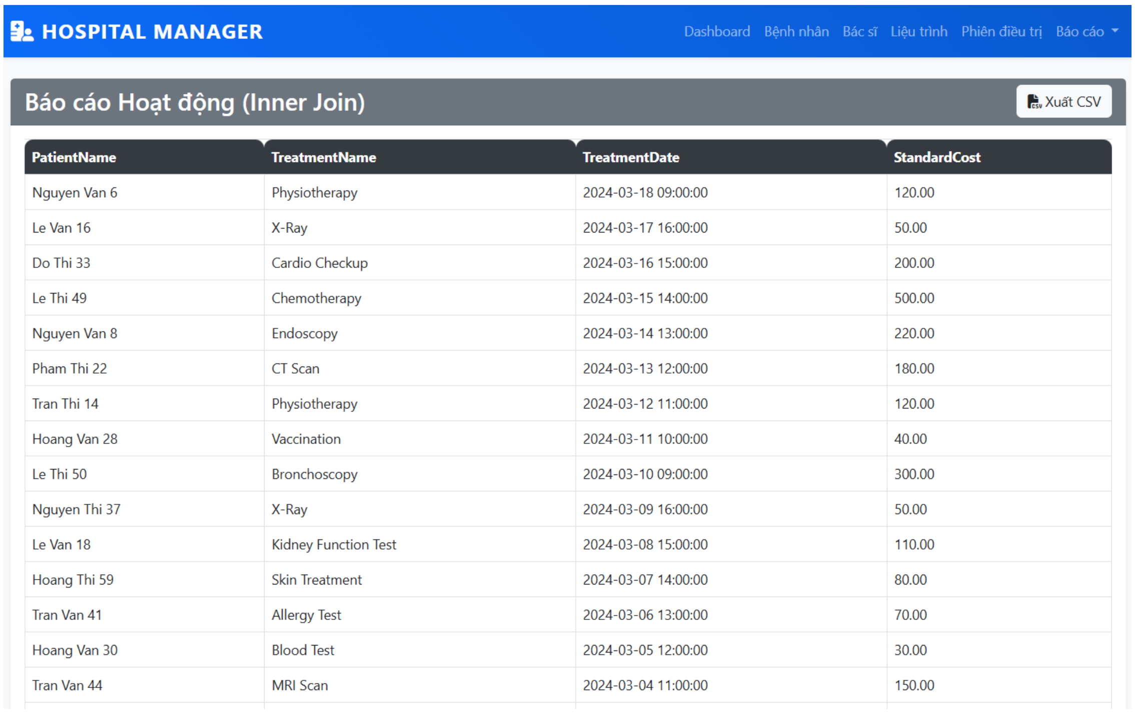
The primary function of the hospital management system is to track ongoing medical activities. The **"Active Treatment Sessions"** report aims to generate a list of all completed or scheduled treatment sessions, linking patients to the specific treatments they have received.

4.1.2 Query Logic

This query utilizes an **INNER JOIN** operation. It retrieves records from the `TreatmentSessions` table and joins them with the `Patients` and `Treatments` tables based on their respective Primary and Foreign Keys.

- **Tables Involved:** `TreatmentSessions`, `Patients`, `Treatments`.
- **Selection Criteria:** Only records where a match exists in all joined tables are returned. This ensures that only valid sessions with fully defined patient and treatment data are displayed.
- **Attributes Displayed:** Patient Name, Treatment Name, Date, and Standard Cost.

4.1.3 Result Interpretation



Báo cáo Hoạt động (Inner Join)

PatientName	TreatmentName	TreatmentDate	StandardCost
Nguyen Van 6	Physiotherapy	2024-03-18 09:00:00	120.00
Le Van 16	X-Ray	2024-03-17 16:00:00	50.00
Do Thi 33	Cardio Checkup	2024-03-16 15:00:00	200.00
Le Thi 49	Chemotherapy	2024-03-15 14:00:00	500.00
Nguyen Van 8	Endoscopy	2024-03-14 13:00:00	220.00
Pham Thi 22	CT Scan	2024-03-13 12:00:00	180.00
Tran Thi 14	Physiotherapy	2024-03-12 11:00:00	120.00
Hoang Van 28	Vaccination	2024-03-11 10:00:00	40.00
Le Thi 50	Bronchoscopy	2024-03-10 09:00:00	300.00
Nguyen Thi 37	X-Ray	2024-03-09 16:00:00	50.00
Le Van 18	Kidney Function Test	2024-03-08 15:00:00	110.00
Hoang Thi 59	Skin Treatment	2024-03-07 14:00:00	80.00
Tran Van 41	Allergy Test	2024-03-06 13:00:00	70.00
Hoang Van 30	Blood Test	2024-03-05 12:00:00	30.00
Tran Van 44	MRI Scan	2024-03-04 11:00:00	150.00

Figure 4.1: Execution Result of Inner Join Query

As shown in Figure 4.1, the result set provides a chronological list of patient visits. This report is essential for the front-desk staff to verify daily activities and for the billing department to calculate daily revenue based on the displayed costs.

4.2 Patient Overview: Comprehensive List (Left Join)

4.2.1 Purpose

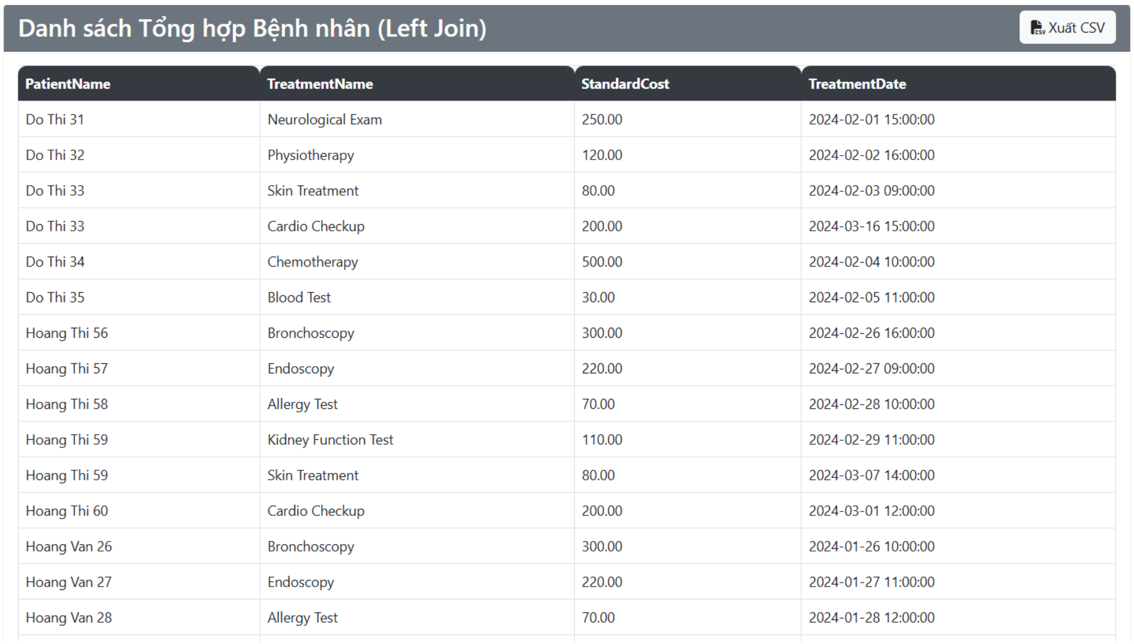
While the **Inner Join** focuses on active sessions, the hospital also needs to manage its entire **patient registry**, including those who have registered but not yet received any treatment (e.g., new registrations or cancelled appointments). This report ensures no patient record is "lost" simply because of inactivity.

4.2.2 Query Logic

This query employs a **LEFT JOIN** strategy, starting with the `Patients` table (the "Left" table).

- **Mechanism:** It retrieves *all* rows from the `Patients` table and matches them with records in `TreatmentSessions` and `Treatments`.
- **Handling Nulls:** If a patient has no corresponding session record, the columns for Treatment Name and Cost will return `NULL` values instead of excluding the patient row entirely.

4.2.3 Result Interpretation



Danh sách Tổng hợp Bệnh nhân (Left Join)

PatientName	TreatmentName	StandardCost	TreatmentDate
Do Thi 31	Neurological Exam	250.00	2024-02-01 15:00:00
Do Thi 32	Physiotherapy	120.00	2024-02-02 16:00:00
Do Thi 33	Skin Treatment	80.00	2024-02-03 09:00:00
Do Thi 33	Cardio Checkup	200.00	2024-03-16 15:00:00
Do Thi 34	Chemotherapy	500.00	2024-02-04 10:00:00
Do Thi 35	Blood Test	30.00	2024-02-05 11:00:00
Hoang Thi 56	Bronchoscopy	300.00	2024-02-26 16:00:00
Hoang Thi 57	Endoscopy	220.00	2024-02-27 09:00:00
Hoang Thi 58	Allergy Test	70.00	2024-02-28 10:00:00
Hoang Thi 59	Kidney Function Test	110.00	2024-02-29 11:00:00
Hoang Thi 59	Skin Treatment	80.00	2024-03-07 14:00:00
Hoang Thi 60	Cardio Checkup	200.00	2024-03-01 12:00:00
Hoang Van 26	Bronchoscopy	300.00	2024-01-26 10:00:00
Hoang Van 27	Endoscopy	220.00	2024-01-27 11:00:00
Hoang Van 28	Allergy Test	70.00	2024-01-28 12:00:00

Figure 4.2: Execution Result of Left Join Query showing patients with NULL treatments

The output (Figure 4.2) displays the complete patient database. Notably, patients such as "Do Thi 31" appear with empty fields in the treatment columns, indicating they are inactive. This insight allows the marketing department to target these specific individuals for follow-up or health check-up reminders.

4.3 Detailed Medical History (Multi-table Join)

4.3.1 Purpose

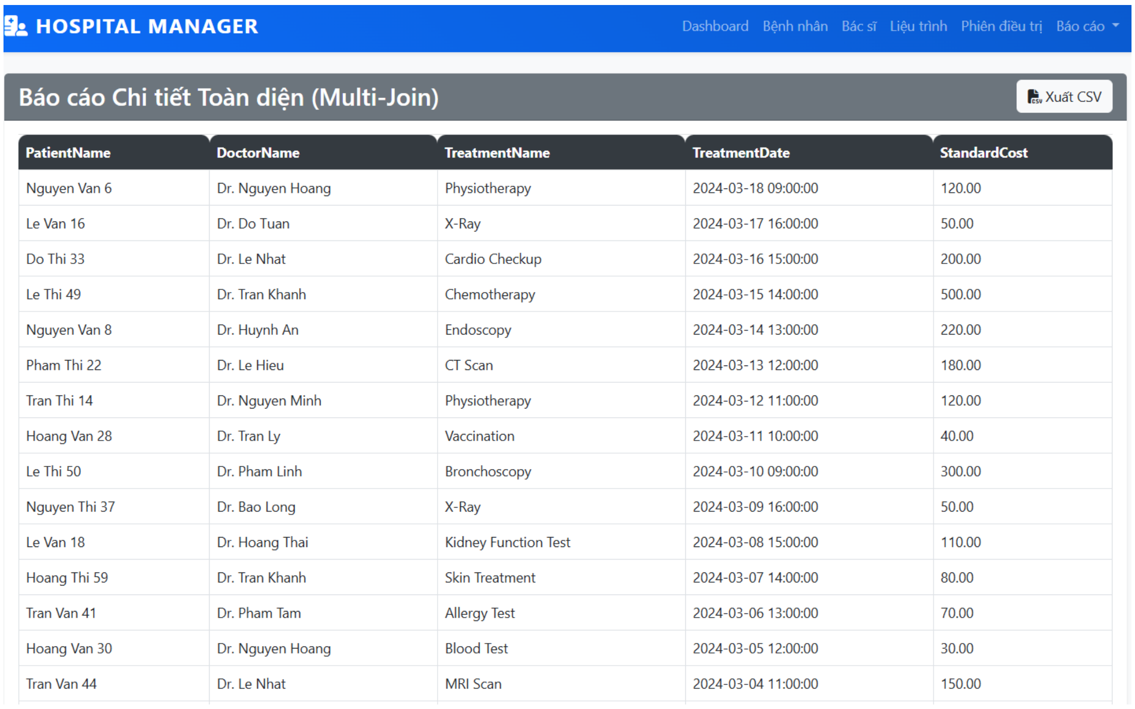
For clinical audits and detailed history tracking, a report combining all major entities is required. This query aims to provide a **360-degree view** of a medical event: "Who treated whom, with what procedure, when, and at what cost?".

4.3.2 Query Logic

This is the most complex **join operation** in the system, linking four distinct tables.

- **Join Chain:** `TreatmentSessions` ↔ `Patients` ↔ `Doctors` ↔ `Treatments`.
- **Complexity:** The query resolves multiple **Foreign Key constraints** simultaneously (`PatientID`, `DoctorID`, `TreatmentID`) to assemble a human-readable record from normalized ID numbers.

4.3.3 Result Interpretation


HOSPITAL MANAGER
Dashboard Bệnh nhân Bác sĩ Liệu trình Phiên điều trị Báo cáo
Xuất CSV

Báo cáo Chi tiết Toàn diện (Multi-Join)

PatientName	DoctorName	TreatmentName	TreatmentDate	StandardCost
Nguyen Van 6	Dr. Nguyen Hoang	Physiotherapy	2024-03-18 09:00:00	120.00
Le Van 16	Dr. Do Tuan	X-Ray	2024-03-17 16:00:00	50.00
Do Thi 33	Dr. Le Nhat	Cardio Checkup	2024-03-16 15:00:00	200.00
Le Thi 49	Dr. Tran Khanh	Chemotherapy	2024-03-15 14:00:00	500.00
Nguyen Van 8	Dr. Huynh An	Endoscopy	2024-03-14 13:00:00	220.00
Pham Thi 22	Dr. Le Hieu	CT Scan	2024-03-13 12:00:00	180.00
Tran Thi 14	Dr. Nguyen Minh	Physiotherapy	2024-03-12 11:00:00	120.00
Hoang Van 28	Dr. Tran Ly	Vaccination	2024-03-11 10:00:00	40.00
Le Thi 50	Dr. Pham Linh	Bronchoscopy	2024-03-10 09:00:00	300.00
Nguyen Thi 37	Dr. Bao Long	X-Ray	2024-03-09 16:00:00	50.00
Le Van 18	Dr. Hoang Thai	Kidney Function Test	2024-03-08 15:00:00	110.00
Hoang Thi 59	Dr. Tran Khanh	Skin Treatment	2024-03-07 14:00:00	80.00
Tran Van 41	Dr. Pham Tam	Allergy Test	2024-03-06 13:00:00	70.00
Hoang Van 30	Dr. Nguyen Hoang	Blood Test	2024-03-05 12:00:00	30.00
Tran Van 44	Dr. Le Nhat	MRI Scan	2024-03-04 11:00:00	150.00

Figure 4.3: Multi-table Join showing detailed doctor-patient interactions

The result set (Figure 4.3) serves as the primary **audit trail** for the hospital. It allows administrators to evaluate **doctor workload** (by seeing which doctor performed which session) and review **patient medical history** in a single unified view.

4.4 Strategic Analysis: High-Cost Treatments (Subquery & Aggregation)

4.4.1 Purpose

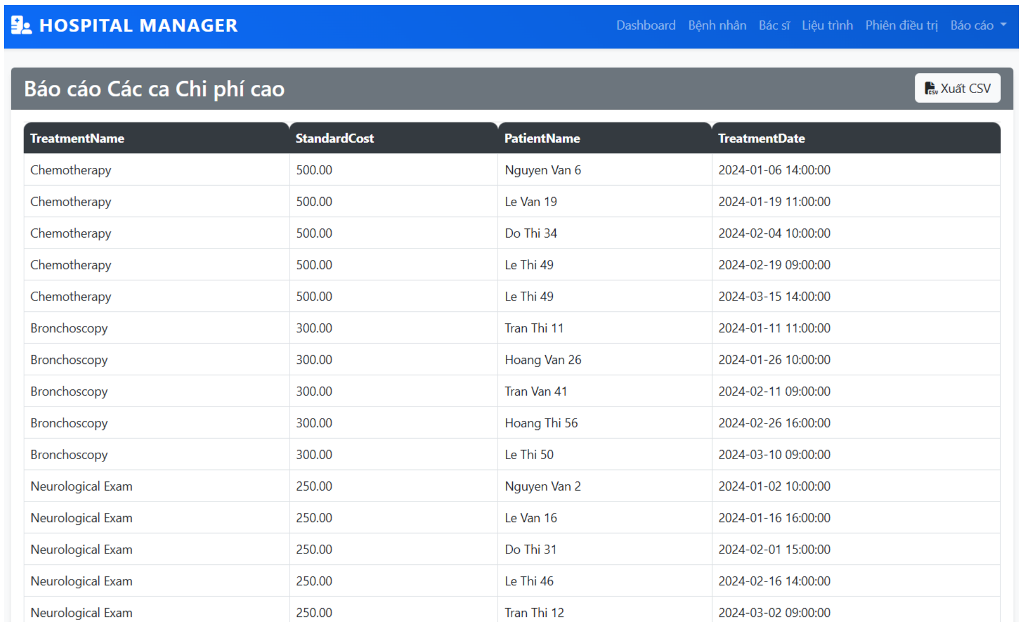
To support financial decision-making, the hospital management needs to identify **premium or high-cost procedures**. This report filters out standard treatments and highlights only those that **exceed the average cost** of all services provided.

4.4.2 Query Logic

This query introduces analytical logic using a **Subquery** and **Aggregation**.

1. **Step 1 (Inner Query):** Calculate the **average cost** of all treatments in the database using the `AVG(StandardCost)` function.
2. **Step 2 (Outer Query):** Select treatment names, costs, and associated patient names where the specific treatment cost is strictly greater than ($>$) the calculated average.

4.4.3 Result Interpretation



Báo cáo Các ca Chi phí cao

TreatmentName	StandardCost	PatientName	TreatmentDate
Chemotherapy	500.00	Nguyen Van 6	2024-01-06 14:00:00
Chemotherapy	500.00	Le Van 19	2024-01-19 11:00:00
Chemotherapy	500.00	Do Thi 34	2024-02-04 10:00:00
Chemotherapy	500.00	Le Thi 49	2024-02-19 09:00:00
Chemotherapy	500.00	Le Thi 49	2024-03-15 14:00:00
Bronchoscopy	300.00	Tran Thi 11	2024-01-11 11:00:00
Bronchoscopy	300.00	Hoang Van 26	2024-01-26 10:00:00
Bronchoscopy	300.00	Tran Van 41	2024-02-11 09:00:00
Bronchoscopy	300.00	Hoang Thi 56	2024-02-26 16:00:00
Bronchoscopy	300.00	Le Thi 50	2024-03-10 09:00:00
Neurological Exam	250.00	Nguyen Van 2	2024-01-02 10:00:00
Neurological Exam	250.00	Le Van 16	2024-01-16 16:00:00
Neurological Exam	250.00	Do Thi 31	2024-02-01 15:00:00
Neurological Exam	250.00	Le Thi 46	2024-02-16 14:00:00
Neurological Exam	250.00	Tran Thi 12	2024-03-02 09:00:00

Figure 4.4: Analytical Report of High-Cost Treatments

The output (Figure 4.4) focuses exclusively on **expensive procedures** (e.g., Surgeries, MRI Scans). This data is critical for **financial forecasting**, **insurance claim processing**, and **resource allocation planning**.

Chapter 5

Discussion and Analysis

Following the successful implementation and testing of the "Hospital Patient Manager" system, this chapter provides a **critical evaluation** of the project's outcomes. It discusses the **effectiveness of the normalized database design** compared to the legacy system, evaluates the application against the **initial project objectives**, and analyzes the system's **limitations and potential areas for future improvement**.

5.1 Comparison: UNF vs. 3NF System

The core motivation of this project was to address data anomalies found in the **Unnormalized Form (UNF)** storage method. The transition to the **Third Normal Form (3NF)** has yielded significant improvements in data integrity and storage efficiency.

Table 5.1: Comparative Analysis of Legacy vs. New System

Metric	Legacy System (UNF)	New System (3NF)
Data Redundancy	High. Patient and Doctor details are repeated for every visit.	Minimal. Master data is stored once; only IDs are referenced in sessions.
Update Anomalies	High risk. Changing a doctor's phone number requires updating hundreds of rows.	Eliminated. Updates are made in a single row in the `Doctors` table.
Data Integrity	Low. No constraints prevent entering invalid or inconsistent data.	High. Enforced by Primary Keys, Foreign Keys, and Data Types.
Query Efficiency	Slow for analytical queries due to table size and lack of indexing.	Optimized. Joins are efficient, and specific indexes speed up retrieval.

The analysis confirms that the **3NF design successfully eliminates the update, insertion, and deletion anomalies** defined in the Problem Statement (Chapter 1).

5.2 Evaluation against Objectives

The project aims and objectives outlined in Chapter 1 have been systematically addressed:

1. **Objective 1: Database Normalization.** *Status: Achieved.* The database was rigorously normalized to **3NF**, resulting in four distinct entities. The `ERD` (Chapter 2) verifies the correct relationship modeling.

2. **Objective 2: Application Development.** *Status: Achieved.* A **Python Flask** application with a complete **GUI** has been deployed. Users can interact with the database without writing `SQL` commands.

3. **Objective 3: Reporting Capabilities.** *Status: Achieved.* The system successfully generates operational reports using **complex joins and subqueries** (as demonstrated in Chapter 4), providing critical insights into hospital operations.

4. **Objective 4: Data Visualization.** *Status: Achieved.* The integrated **Dashboard** provides real-time visibility into **Key Performance Indicators (KPIs)**, aiding rapid decision-making.

5.3 System Limitations

Despite the successful implementation, the current version of the "Hospital Patient Manager" has certain **limitations** inherent to its scope as a course project:

- **Security and Authentication:** The system currently lacks a **login mechanism**. Any user with access to the application `URL` can view and modify sensitive patient data. **Role-based access control (RBAC)** is missing.

- **Scalability:** The application runs on a **local development server** (Flask built-in server). For a real-world hospital environment with concurrent users, a production-grade server (e.g., `Gunicorn/Nginx`) would be required.

- **Concurrency Handling:** While **MySQL** handles transactions well, the current Python implementation does not fully implement **optimistic locking**, which could lead to **race conditions** if two staff members edit the same record simultaneously.

- **Input Validation:** While basic validation exists (e.g., required fields), **advanced validation** (e.g., checking for valid phone number formats or duplicate appointments) is limited.

Chapter 6

Conclusions and Future Work

6.1 Project Summary

The project set out to resolve the critical issues of **data redundancy** and **update anomalies** caused by the hospital's legacy Unnormalized Form (UNF) data storage. By applying rigorous database **normalization theories**, the team successfully designed a **Third Normal Form (3NF)** schema comprising four relational entities: *Patients, Doctors, Treatments, and TreatmentSessions*. To make this database accessible to non-technical hospital staff, a **full-stack web application** was developed using **Python Flask** and **MySQL**. The resulting system provides:

- A robust Backend that ensures **data integrity** through Foreign Key constraints.
- A user-friendly Frontend with a **real-time Dashboard** for monitoring hospital **KPIs**.
- A powerful **Reporting Module** capable of executing complex `SQL` joins to track medical history and financial performance.

The successful deployment and testing of the application confirm that all primary objectives defined in Chapter 1 have been met. The system effectively transforms raw, chaotic data into **structured, actionable information**.

6.2 Future work

To evolve the "Hospital Patient Manager" into a **commercial-grade product**, the following enhancements are proposed for the next development phase:

1. Security and Role-Based Access Control (RBAC)

The immediate priority is to implement a secure login system.

- **Authentication:** Use libraries like `Flask-Login` to handle user sessions and password hashing (e.g., `bcrypt`).
- **Authorization:** Define specific roles such as "Admin" (full CRUD access), "Doctor" (view-only patient records), and "Receptionist" (appointment scheduling only).

2. Cloud Deployment and Scalability

To support remote access for hospital staff, the system should be migrated to the cloud.

- **Database:** Migrate `MySQL` to a managed service like `AWS RDS` or `Azure SQL Database` for automated backups and high availability.
- **Application:** Containerize the application using **Docker** and orchestrate it with **Kubernetes** to handle varying loads.

3. Advanced Analytics and AI Integration

Leveraging the structured data now available in the **3NF schema**, advanced analytics can be introduced:

- **Predictive Modeling:** Use **Machine Learning** to predict patient influx peaks based on historical session data.
- **Revenue Forecasting:** Implement **linear regression models** to project future revenue trends based on treatment costs.

4. Mobile Application Interface

Developing a **RESTful API** would allow the creation of a mobile app for doctors, enabling them to view patient records and schedules on tablets while moving between wards.

Chapter 7

Reflection and Personal Growth

This chapter reflects on the educational journey and the practical experiences gained throughout the development of the "Hospital Patient Manager" project. While the previous chapters focused on the technical product and its performance, this section analyzes the development process itself. It highlights the technical challenges encountered, the effectiveness of the team's collaboration workflow, and the professional skills acquired during the transition from theoretical database concepts to a fully functioning software application.

7.1 Technical Challenges and Solutions

During the implementation phase, the team encountered several technical hurdles that required research and problem-solving skills to overcome.

Database Connectivity Issues

One of the most significant challenges was establishing a stable connection between the **Python Flask** application and the **MySQL** database server.

- **The Challenge:** The development environment (Windows) initially rejected connections due to **authentication protocol mismatches** and "Named Pipe" errors, causing the application to fail in retrieving data despite a correct schema.
- **The Solution:** By debugging the connection module and explicitly configuring the **TCP/IP loopback address (127.0.0.1)** instead of "localhost", we resolved the platform-specific driver issues. We also learned the importance of **environment variables** (`.env`) for managing credentials securely.

Data Synchronization

Maintaining consistency between the database state and the application view was critical.

- **The Challenge:** Initially, the dashboard displayed zero values even after creating tables.
- **The Solution:** We realized that creating the schema was not enough; **data seeding** was required. We developed a robust `seed.sql` script and ensured that **transaction commits** were executed properly in **MySQL Workbench** to make data visible to the application.

7.2 Lessons Learned

This project bridged the gap between academic theory and industrial practice.

1. **The Importance of Normalization:** We experienced firsthand how a **3NF design simplifies application logic**. Because the database was normalized, we did not have to write complex code to update a doctor's name in multiple places; a single `SQL UPDATE` sufficed.

2. **Full-Stack Integration:** We gained a comprehensive understanding of how the "frontend" talks to the "backend" and how the "backend" talks to the "database". Understanding the **flow of data** from a user's form input to a **MySQL table row** was a valuable insight.

3. **Debugging Skills:** We learned that error messages (e.g., in the terminal) are not roadblocks but **clues**. Reading **stack traces** and isolating variables (*Network* vs. *Code* vs. *Database*) became a daily practice.

7.3 Concluding Thoughts

The "Hospital Patient Manager" project was more than just a coding assignment; it was a **simulation of a real-world software engineering lifecycle**. Despite the initial difficulties with configuration and environment setup, the final outcome is a testament to the team's **persistence and collaboration**. We are confident that the knowledge gained here will serve as a strong foundation for our future careers in **Information Technology**.

Table 7.1: Team Contribution Matrix (Detailed Breakdown)

Member	**Role**	**Key Contributions**	**%**
Nguyen Xuan Kiet-Leader (Student ID: 11247306)	**Project Lead & Full-stack Dev**	▪ **Architecture:** Designed the complete MVC system architecture. ▪ **Coding:** Developed 100% of Backend (Python/Flask) and Frontend (HTML/Bootstrap). ▪ **Integration:** Handled DB connection and debugging. ▪ **Report:** Authored the main content of the Project Report.	**65%**
Nguyen Phung Hoa (Student ID: 11247290)	Database Designer	▪ Analyzed functional dependencies (UNF to 3NF). ▪ Designed ERD Schema. ▪ Provided initial SQL scripts for Schema and Seed data.	20%
Nguyen Huy Minh (Student ID: 11247322)	Documentation Support	▪ Designed the presentation slide layout. ▪ Formatted visual assets for slides. ▪ *Support:* Verified UI responsiveness.	7.5%
Hoang Thi Ngoc Han (Student ID: 11247286)	Documentation Support	▪ Summarized content for presentation points. ▪ Assisted in formatting the bibliography. ▪ *Support:* Checked spelling and grammar.	7.5%

If you have any question, please contact : kietnguyenxuan2401@gmail.com or scan this QR code