National Economics University

School of Advanced Education Program

Hospital Patient Manager

Nguyen Tuan Anh - 11247260
Tran Tuan Anh - 11247264
Tran Minh Hoang - 11247292
Vu Quoc Huy - 11247301

Supervisor: Tran Duc Minh

December 6, 2025

Contents

Chapter 1

OVERVIEW AND REQUIREMENTS ANALYSIS

1.1 Business Context and Problem Description

In the modern context, patient data management in hospitals faces increasing complexity due to the volume of patients, the variety of services, and the need for both clinical and administrative data to be properly captured. Many hospitals, especially small or medium-sized ones, still store patient information using spreadsheets, manual logs, or loosely structured systems.

These approaches lead to several major issues:

- **Data Redundancy and Inconsistency**: The same patient data may be entered in multiple files, leading to potential conflicts and inconsistencies.
- **Difficulties in Tracking**: It becomes challenging to track a patient's full history across visits, departments, and billing records.
- **Limited Analytical Capability**: Management struggles to generate meaningful reports (e.g., revenue per department, patient load per doctor, unpaid bills).
- **Risk of Human Error**: Manual inputs and updates increase the probability of errors in critical information (e.g., billing, appointments).

The *Hospital Patient Manager* project is designed to resolve these challenges by implementing a properly structured relational database. This system ensures data consistency, supports core hospital operations, and provides a reliable foundation for later expansion into full hospital information systems (HIS).

The system focuses on:

- **Patients**: Basic personal and contact data.
- **Doctors and Administrative Staff**: Their roles, departments, and contact details.
- **Departments**: Organizational units in the hospital.
- **Appointments**: Scheduling and managing patient visits.
- **Medical Records**: Storing diagnoses and treatments.

- **Billing**: Handling payments and outstanding balances.

By building a 3NF-compliant schema and connecting it to a functional application, the project demonstrates how structured database design can improve hospital data quality, reduce redundancy, and support better decision-making in healthcare operations.

1.2 Project Scope

The project scope covers the following elements:

- **Patients**: Storing sensitive demographic and contact information.
- **Doctors and Administrative Staff**: Managing the workforce.
- **Departments**: Defining the organizational hierarchy.
- **Appointments**: Acting as the central operational event.
- **Medical Records and Billing**: Handling the clinical and financial outcomes of appointments.

The system also includes:

- A MySQL database implementing the 3NF design.
- Views, Stored Procedures, and Triggers for operational and analytic tasks.
- A Python application (with visualization) that leverages the database for CRUD and reporting.

Chapter 2

DATABASE ANALYSIS AND DESIGN

2.1 Input Data (UNF Table) and Functional Dependencies (FDs)

2.1.1 Input Data (Universal Relation / UNF)

Based on the business context, we assume a universal relation that aggregates all necessary attributes of the hospital operations into a single relation before normalization. The attributes are logically grouped by entity but are initially stored in a single UNF (Unnormalized Form) dataset.

The set of attributes R is defined as follows:

$R = \{$department_id, department_name, dept_location, head_of_department,
doctor_id, doctor_full_name, doctor_specialization, doctor_phone_number, doctor_email,
patient_id, patient_full_name, patient_dob, patient_gender, patient_phone_number, patient_address,
appointment_id, appointment_date, appointment_reason, appointment_status,
record_id, diagnosis, prescription, treatment_notes, follow_up_date,
bill_id, amount_due, amount_paid, payment_date, payment_status, payment_method,
staff_id, staff_full_name, staff_position, staff_phone_number, staff_email, staff_assigned_dept$\}$.

2.1.2 Functional Dependencies (FDs)

Based on the business rules outlined in the project requirements, the following Functional Dependencies (FDs) are identified. A Functional Dependency $X \rightarrow Y$ means that the value of X uniquely determines the value of Y.

1. Department Dependencies

FD1: department_id $\rightarrow$ department_name, dept_location, head_of_department

FD2: department_name $\rightarrow$ department_id

Explanation: Each department is uniquely identified by `department_id`, which determines its name, location, and the head of department. Assuming department names are unique, `department_name` can also determine `department_id`.

2. Doctor Dependencies

FD3: doctor_id → doctor_full_name, doctor_specialization, doctor_phone_number, doctor_email, department
doctor_email → doctor_id

Explanation: `doctor_id` uniquely identifies each doctor and determines their contact details and associated department. Each `doctor_email` is unique and thus serves as an alternate key.

3. Staff Dependencies

FD4: staff_id → staff_full_name, staff_position, staff_phone, staff_email, staff_assigned_dept
staff_email → staff_id

Explanation: The staff ID determines the staff member's personal details and the department they support. Staff email is unique, making it an alternate key.

5. Appointment Dependencies

FD5: appointment_id → patient_id, doctor_id, appointment_date, reason, status

Explanation: A specific appointment ID links a patient to a doctor for a specific date and reason.

6. Medical Record Dependencies

FD6: record_id → appointment_id, diagnosis, prescription, treatment_notes, follow_up_date

Explanation: The medical record ID identifies the clinical details (diagnosis, treatment) resulting from a specific appointment.

7. Billing Dependencies

FD7: bill_id → patient_id, appointment_id, amount_due, amount_paid, payment_date, payment_status,
Observed: appointment_id → patient_id

Explanation: The bill ID uniquely identifies the financial transaction details, payment status, and the associated patient/appointment. One appointment belongs to exactly one patient.

2.2 Normalization Process (UNF → 1NF → 2NF → 3NF)

The goal of data normalization is to organize the database structure into well-defined tables that minimize redundancy and avoid update anomalies. In this project, we follow the standard sequence of normal forms and aim to achieve Third Normal Form (3NF), which is commonly sufficient for most transactional systems.

2.2.1 Understanding the Normal Forms

The normalization process is hierarchical:

1NF (First Normal Form): All attributes are atomic (no repeating groups or multi-valued attributes). Each field contains only one value and each record is unique.

2NF (Second Normal Form): The table is already in 1NF, and every non-key attribute is fully functionally dependent on the entire Primary Key (no partial dependency on a subset of a composite key).

3NF (Third Normal Form): The table is in 2NF, and no non-key attribute depends transitively on the Primary Key (no transitive dependencies).

2.2.2 Transitive Dependency Explained

A transitive dependency exists when an attribute C depends on another non-key attribute B, which, in turn, depends on the Primary Key A:

$$A \to B \quad \text{and} \quad B \to C \quad \Rightarrow \quad A \to C$$

(Transitive Dependency)

To achieve 3NF, the dependency $B \to C$ must be removed by separating these attributes into a new table where B becomes the new Primary Key.

2.2.3 Normalization of Proposed Entities

Based on the proposed entities and Functional Dependencies derived from the business context, we analyze the current status of each table.

DEPARTMENT Table

Attributes: `department_id, department_name, location, head_of_department`

Functional Dependencies (FDs):

- `department_id` $\to$ `department_name, location, head_of_department`
- `department_name` $\to$ `department_id`

Analysis:

- All attributes are atomic.
- The primary key (`department_id`) is single-column (non-composite).
- No non-key attribute depends on another non-key attribute.
- There is no transitive dependency that violates 3NF.

Normal Form	Status	Justification
1NF	Achieved	All attributes are atomic.
2NF	Achieved	PK is non-composite (department_id), so no partial dependency exists.
3NF	Achieved	No transitive dependency is present. All non-key attributes depend directly on department_id.
Conclusion	Achieved 3NF	

Table 2.1: Table 1 – Normalization status of DEPARTMENT table

DOCTOR Table

Attributes: `doctor_id, full_name, specialization, phone_number, email, department_id`

FDs:

- `doctor_id` → `full_name, specialization, phone_number, email, department_id`
- `email` → `doctor_id` (assuming email is unique).

Analysis:

- All fields are atomic.
- The primary key (`doctor_id`) is non-composite.
- Other attributes depend directly on `doctor_id` and not on each other.

Normal Form	Status	Justification
1NF	Achieved	No repeating groups.
2NF	Achieved	PK (doctor_id) is non-composite.
3NF	Achieved	No non-key attribute depends on another non-key attribute.
Conclusion	Achieved 3NF	

Table 2.2: Table 2 – Normalization status of DOCTOR table

PATIENT Table

Attributes: `patient_id, full_name, gender, date_of_birth, phone_number, email, address, emergency_contact, date_registered`

FDs:

- `patient_id` → `full_name, gender, date_of_birth, phone_number, email, address, emergency_contact, date_registered`
- `email` → `patient_id` (assuming email is unique).

Analysis:

- All attributes are atomic.
- The primary key is non-composite (`patient_id`).
- All non-key attributes depend solely on `patient_id`, with no transitive dependencies.

Normal Form	Status	Justification
1NF	Achieved	No repeating groups.
2NF	Achieved	PK (patient_id) is non-composite.
3NF	Achieved	All non-key attributes depend directly on patient_id.
Conclusion	Achieved 3NF	

Table 2.3: Table 3 – Normalization status of PATIENT table

APPOINTMENT Table

Attributes: `appointment_id, patient_id, doctor_id, appointment_date, reason, status`

FDs:

- `appointment_id` → `patient_id, doctor_id, appointment_date, reason, status`
- `(patient_id, doctor_id, appointment_date)` → `appointment_id` (candidate key).

Analysis:

- All fields are atomic.
- The primary key is non-composite (`appointment_id`).
- Each non-key attribute is fully dependent on `appointment_id`.
- No attribute depends on other non-key attributes.

Normal Form	Status	Justification
1NF	Achieved	No repeating groups.
2NF	Achieved	PK (appointment_id) is non-composite.
3NF	Achieved	No non-key attribute depends on another non-key attribute.
Conclusion	Achieved 3NF	

Table 2.4: Table 4 – Normalization status of APPOINTMENT table

MEDICAL_RECORD Table

Attributes: `record_id, appointment_id, diagnosis, prescription, treatment_notes, follow_up_date`

FDs:

- `record_id` → `appointment_id, diagnosis, prescription, treatment_notes, follow_up_date`
- `appointment_id` → `diagnosis, prescription, treatment_notes, follow_up_date` (one appointment results in one medical record).

Analysis:

- The primary key is `record_id`.
- All non-key attributes are dependent on `record_id`.
- The relationship with `appointment_id` does not introduce a transitive dependency that violates 3NF.

Normal Form	Status	Justification
1NF	Achieved	No repeating groups; each record row is unique.
2NF	Achieved	PK (record_id) is non-composite.
3NF	Achieved	Non-key attributes depend directly on record_id.
Conclusion	Achieved 3NF	

Table 2.5: Table 5 – Normalization status of MEDICAL_RECORD table

BILLING Table

Attributes: `bill_id, patient_id, appointment_id, amount_due, amount_paid, payment_date, payment_status, payment_method`

FDs:

- `bill_id` → `patient_id, appointment_id, amount_due, amount_paid, payment_date, payment_status, payment_method`
- Observed: `appointment_id` → `patient_id` (one appointment belongs to one patient).

Analysis:

- `bill_id` is the primary key.
- All non-key attributes are directly dependent on `bill_id`.
- The relationship with `appointment_id` and `patient_id` does not break 3NF.

Normal Form	Status	Justification
1NF	Achieved	No repeating billing entries within a single row.
2NF	Achieved	PK (bill_id) is non-composite.
3NF	Achieved	All non-key attributes depend on bill_id; no transitive dependencies.
Conclusion	Achieved 3NF	

Table 2.6: Table 6 – Normalization status of BILLING table

STAFF Table

Attributes: `staff_id, full_name, position, phone_number, email, assigned_department`

FDs:

- `staff_id` → `full_name, position, phone_number, email, assigned_department`
- `email` → `staff_id` (assuming email is unique).

Analysis:

- The primary key is non-composite (`staff_id`).
- All non-key attributes depend directly and only on `staff_id`.
- There are no transitive dependencies.

Normal Form	Status	Justification
1NF	Achieved	All fields contain atomic values.
2NF	Achieved	PK (staff_id) is non-composite.
3NF	Achieved	Non-key attributes only depend on staff_id.
Conclusion	Achieved 3NF	

Table 2.7: Table 7 – Normalization status of STAFF table

Summary

All proposed entities meet the criteria for Third Normal Form (3NF).

2.3 Final Entity-Relationship Diagram (ERD)

The database design for the Hospital Patient Manager system is centered around representing the core entities involved in clinical and administrative workflows. The model is built around seven core entities: `DEPARTMENT`, `DOCTOR`, `PATIENT`, `STAFF`, `APPOINTMENT`, `MEDICAL_RECORD`, and `BILLING`.

The entities are connected through the following one-to-many (1:N) and one-to-one (1:1) relationships, enforced by foreign key constraints:

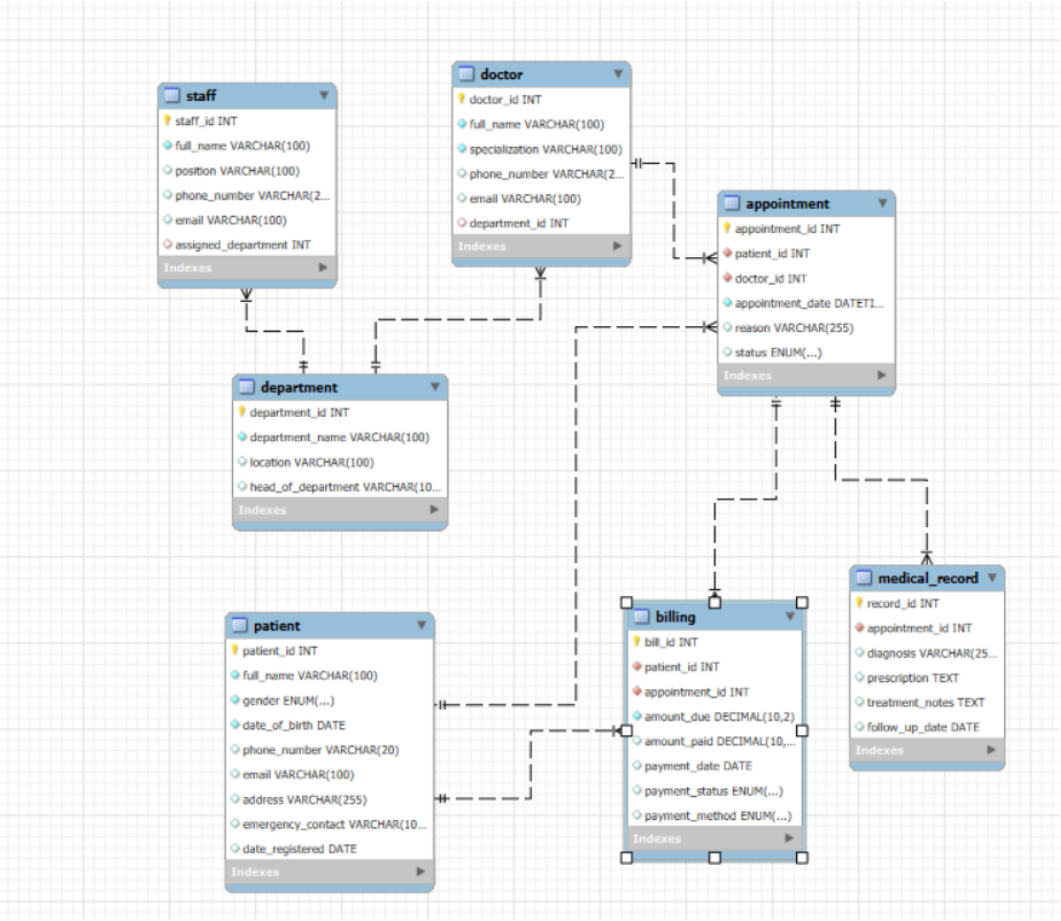

Figure 2.1: Screenshot of Entity Relationship Diagram

- **Doctor/Staff to Department (1:N)**: One DEPARTMENT can have many DOCTORs. One DEPARTMENT can also have many STAFF members (e.g., nurses, administrative staff).
- **Patient to Appointment (1:N)**: A PATIENT can have multiple APPOINTMENTs over time. Each APPOINTMENT is associated with exactly one PATIENT.
- **Doctor to Appointment (1:N)**: A DOCTOR can be associated with many APPOINTMENTs. Each APPOINTMENT is handled by one DOCTOR.
- **Appointment to Medical_Record (1:1)**: Each APPOINTMENT may generate one MEDICAL_RECORD that stores diagnosis and treatment. Each MEDICAL_RECORD links back to exactly one APPOINTMENT.
- **Appointment to Billing (1:N)**: An APPOINTMENT can generate multiple BILLING records (e.g., partial payments or multiple service charges tied to the same visit).
- **Patient to Billing (1:N)**: A PATIENT can be linked to multiple BILLING records across different appointments.

2.4 Final List of Tables with Primary and Foreign Keys

1. Table: DEPARTMENT

Column Name	Data Type	Description	Key
department_id	INT AUTO_INCREMENT	Department ID	PK
department_name	VARCHAR(100)	Department Name	
location	VARCHAR(100)	Location	
head_of_department	VARCHAR(100)	Head of Department	

Table 2.8: Table 1: DEPARTMENT

Keys:

- Primary Key (PK): department_id
- Foreign Key (FK): (None)

2. Table: DOCTOR

Column Name	Data Type	Description	Key
doctor_id	INT AUTO_INCREMENT	Doctor ID	PK
full_name	VARCHAR(100)	Full Name	
specialization	VARCHAR(100)	Specialization	
phone_number	VARCHAR(20)	Phone Number	
email	VARCHAR(100) UNIQUE	Email	
department_id	INT	Department ID	FK

Table 2.9: Table 2: DOCTOR

Keys:

- Primary Key (PK): doctor_id
- Foreign Key (FK): department_id references Department.department_id

3. Table: PATIENT

Column Name	Data Type	Description	Key
patient_id	INT AUTO_INCREMENT	Patient ID	PK
full_name	VARCHAR(100)	Full Name	
gender	VARCHAR(10)	Gender	
date_of_birth	DATE	Date of Birth	
phone_number	VARCHAR(20)	Phone Number	
email	VARCHAR(100) UNIQUE	Email	
address	VARCHAR(255)	Address	
emergency_contact	VARCHAR(100)	Emergency Contact	
date_registered	DATE	Date Registered	

Table 2.10: Table 3: PATIENT

Keys:

- Primary Key (PK): patient_id
- Foreign Key (FK): (None)

4. Table: APPOINTMENT

Column Name	Data Type	Description	Key
appointment_id	INT AUTO_INCREMENT	Appointment ID	PK
patient_id	INT	Patient ID	FK
doctor_id	INT	Doctor ID	FK
appointment_date	DATE	Appointment Date	
reason	VARCHAR(255)	Reason	
status	VARCHAR(50)	Status	

Table 2.11: Table 4: APPOINTMENT

Keys:

- Primary Key (PK): appointment_id
- Foreign Key (FK):
  - patient_id references Patient.patient_id

– doctor_id references Doctor.doctor_id

5. Table: MEDICAL_RECORD

Column Name	Data Type	Description	Key
record_id	INT AUTO_INCREMENT	Medical Record ID	PK
appointment_id	INT	Appointment ID	FK
diagnosis	VARCHAR(255)	Diagnosis	
prescription	TEXT	Prescription	
treatment_notes	TEXT	Treatment Notes	
follow_up_date	DATE	Follow-up Date	

Table 2.12: Table 5: MEDICAL_RECORD

Keys:

- Primary Key (PK): record_id
- Foreign Key (FK): appointment_id references Appointment.appointment_id

6. Table: BILLING

Column Name	Data Type	Description	Key
bill_id	INT AUTO_INCREMENT	Bill ID	PK
patient_id	INT	Patient ID	FK
appointment_id	INT	Appointment ID	FK
amount_due	DECIMAL(10,2)	Amount Due	
amount_paid	DECIMAL(10,2)	Amount Paid	
payment_date	DATE	Payment Date	
payment_status	VARCHAR(50)	Payment Status	
payment_method	VARCHAR(50)	Payment Method	

Table 2.13: Table 6: BILLING

Keys:

- Primary Key (PK): bill_id
- Foreign Key (FK):
  - patient_id references Patient.patient_id
  - appointment_id references Appointment.appointment_id

7. Table: STAFF

Column Name	Data Type	Description	Key
staff_id	INT AUTO_INCREMENT	Staff ID	PK
full_name	VARCHAR(100)	Full Name	
position	VARCHAR(100)	Position	
phone_number	VARCHAR(20)	Phone Number	
email	VARCHAR(100) UNIQUE	Email	
assigned_department	INT	Department ID	FK

Table 2.14: Table 7: STAFF

Keys:

- Primary Key (PK): staff_id
- Foreign Key (FK): assigned_department references Department.department_id

Chapter 3

IMPLEMENTATION ON MYSQL

3.1 SQL Code for Table Creation and Constraints (schema.sql)

```sql
CREATE DATABASE hospital_manager;
USE hospital_manager;
CREATE TABLE DEPARTMENT (
    department_id INT AUTO_INCREMENT PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL UNIQUE,
    location VARCHAR(100),
    head_of_department VARCHAR(100)
);

CREATE TABLE STAFF (
    staff_id            INT AUTO_INCREMENT PRIMARY KEY,
    full_name           VARCHAR(100) NOT NULL,
    position            VARCHAR(100) NOT NULL,
    phone_number        VARCHAR(20) UNIQUE,
    email               VARCHAR(100) UNIQUE,
    assigned_department INT,
    FOREIGN KEY (assigned_department) REFERENCES DEPARTMENT(department_id)
);
```

Figure 3.1: Screenshot of schema.sql execution

The `schema.sql` file contains the Data Definition Language (DDL) commands used to establish the database structure based on the 3NF model (7 tables).

Operation: `CREATE TABLE` statements.

Purpose: To define the structure (schema) of the `hospital_db` database, create all base tables (`DEPARTMENT`, `DOCTOR`, `PATIENT`, `STAFF`, `APPOINTMENT`, etc.) along with setting up critical constraints.

Key Focus: Enforcing data integrity by specifying:

- **Primary Keys (PK)**: (`department_id`, `doctor_id`, etc.) to uniquely identify each record.
- **Foreign Keys (FK)**: (e.g., `DOCTOR.department_id` references `DEPARTMENT.department_id`) to maintain relationships between tables.

- **Constraints**: Including `NOT NULL` for required fields, `UNIQUE` for attributes that must be unique (e.g., emails, phone numbers), and appropriate data types (`INT`, `VARCHAR`, `DATE`, `DECIMAL`).

3.2 SQL Code for Sample Data Insertion (seed.sql)

The `seed.sql` file contains the Data Manipulation Language (DML) commands used to populate the newly created tables with realistic sample records.

```
16    -- ============================
17    -- 2. DOCTOR (10 records)
18    -- ============================
19 •  INSERT INTO Doctor (full_name, specialization, phone_number, email, department_id)
20    VALUES
21    ('Dr. Nguyen Quang Anh', 'Cardiology', '0905111222', 'anh.nguyen@hospital.com', 1),
22    ('Dr. Tran Bao Long', 'Cardiology', '0912333444', 'long.tran@hospital.com', 1),
23    ('Dr. Le Thi Trang', 'Neurology', '0933444555', 'trang.le@hospital.com', 2),
24    ('Dr. Pham Minh Hieu', 'Neurology', '0988776655', 'hieu.pham@hospital.com', 2),
25    ('Dr. Hoang Van Phuc', 'Pediatrics', '0905111999', 'phuc.hoang@hospital.com', 3),
26    ('Dr. Nguyen Thu Ha', 'Pediatrics', '0912333555', 'ha.nguyen@hospital.com', 3),
27    ('Dr. Tran Hoai Nam', 'Orthopedics', '0933666777', 'nam.tran@hospital.com', 4),
28    ('Dr. Do Thi Mai', 'Orthopedics', '0988111223', 'mai.do@hospital.com', 4),
29    ('Dr. Pham Quang Huy', 'Dermatology', '0905222333', 'huy.pham@hospital.com', 5),
30    ('Dr. Le Thanh Tam', 'Dermatology', '0912444555', 'tam.le@hospital.com', 5);
```

Figure 3.2: Screenshot of seed.sql execution

Operation: `INSERT INTO` statements.

Purpose: To fill the database tables with sufficient and diverse data so that queries and application features (such as the Python visualization) can be demonstrated and tested effectively.

Key Focus: The script inserts the minimum required number of records per table (e.g., multiple departments, doctors, patients, and appointments) to ensure that joins, groupings, and aggregations produce meaningful reports.

3.3 Implementation of Views

In this section, the views that were created to support data analysis and application logic are described. Views help simplify complex joins and provide an aggregated or joined overview of specific hospital operations.

3.3.1 View 1: v_patient_appointments

One of the primary examples is `v_patient_appointments`, which joins `PATIENT`, `APPOINTMENT`, and `DOCTOR` information to display:

- Patient name
- Doctor name
- Appointment date

appointment...	appointment_date	status	patient_id	patient_name	doctor_id	doctor_name	specialization	department...	department_na...
1	2025-01-05 09:00:00	Completed	1	Nguyen Van A	1	Dr. Nguyen Quang Anh	Cardiology	1	Cardiology
2	2025-01-06 10:00:00	Completed	2	Tran Thi B	1	Dr. Nguyen Quang Anh	Cardiology	1	Cardiology
19	2025-01-23 09:00:00	Completed	19	Hoang Van S	1	Dr. Nguyen Quang Anh	Cardiology	1	Cardiology
20	2025-01-24 10:00:00	Completed	20	Pham Thi T	1	Dr. Nguyen Quang Anh	Cardiology	1	Cardiology
3	2025-01-07 11:00:00	Completed	3	Le Van C	2	Dr. Tran Bao Long	Cardiology	1	Cardiology
21	2025-01-24 11:00:00	Completed	2	Tran Thi B	2	Dr. Tran Bao Long	Cardiology	1	Cardiology
15	2025-01-20 14:00:00	Completed	15	Pham Van O	9	Dr. Pham Quang Huy	Dermatology	5	Dermatology
16	2025-01-21 15:30:00	Completed	16	Nguyen Thi P	9	Dr. Pham Quang Huy	Dermatology	5	Dermatology
25	2025-01-24 16:00:00	Completed	10	Pham Thi J	9	Dr. Pham Quang Huy	Dermatology	5	Dermatology
17	2025-01-22 10:00:00	Completed	17	Tran Van Q	10	Dr. Le Thanh Tam	Dermatology	5	Dermatology
18	2025-01-22 11:30:00	Completed	18	Le Thi R	10	Dr. Le Thanh Tam	Dermatology	5	Dermatology
26	2025-01-24 17:00:00	Completed	12	Tran Thi L	10	Dr. Le Thanh Tam	Dermatology	5	Dermatology
4	2025-01-08 09:30:00	Completed	4	Hoang Thi D	3	Dr. Le Thi Trang	Neurology	2	Neurology
5	2025-01-09 14:00:00	Completed	5	Pham Van E	3	Dr. Le Thi Trang	Neurology	2	Neurology
30	2025-01-25 14:30:00	Completed	20	Pham Thi T	3	Dr. Le Thi Trang	Neurology	2	Neurology
6	2025-01-10 15:00:00	Completed	6	Nguyen Thi F	4	Dr. Pham Minh Hieu	Neurology	2	Neurology
22	2025-01-24 13:00:00	Completed	3	Le Van C	4	Dr. Pham Minh Hieu	Neurology	2	Neurology
11	2025-01-16 16:00:00	Completed	11	Nguyen Van K	7	Dr. Tran Hoai Nam	Orthopedics	4	Orthopedics
12	2025-01-17 10:00:00	Completed	12	Tran Thi L	7	Dr. Tran Hoai Nam	Orthopedics	4	Orthopedics
24	2025-01-24 15:00:00	Completed	8	Le Thi H	7	Dr. Tran Hoai Nam	Orthopedics	4	Orthopedics
28	2025-01-25 11:00:00	Completed	16	Nguyen Thi P	7	Dr. Tran Hoai Nam	Orthopedics	4	Orthopedics
13	2025-01-18 09:30:00	Completed	13	Le Van M	8	Dr. Do Thi Mai	Orthopedics	4	Orthopedics
14	2025-01-19 11:00:00	Completed	14	Hoang Thi N	8	Dr. Do Thi Mai	Orthopedics	4	Orthopedics
27	2025-01-25 09:30:00	Completed	14	Hoang Thi N	8	Dr. Do Thi Mai	Orthopedics	4	Orthopedics
7	2025-01-12 08:00:00	Completed	7	Tran Van G	5	Dr. Hoang Van Phuc	Pediatrics	3	Pediatrics
8	2025-01-13 10:30:00	Completed	8	Le Thi H	5	Dr. Hoang Van Phuc	Pediatrics	3	Pediatrics
29	2025-01-25 13:30:00	Completed	18	Le Thi R	5	Dr. Hoang Van Phuc	Pediatrics	3	Pediatrics
9	2025-01-14 13:00:00	Completed	9	Hoang Van I	6	Dr. Nguyen Thu Ha	Pediatrics	3	Pediatrics
10	2025-01-15 09:00:00	Completed	10	Pham Thi J	6	Dr. Nguyen Thu Ha	Pediatrics	3	Pediatrics
23	2025-01-24 14:00:00	Completed	5	Pham Van E	6	Dr. Nguyen Thu Ha	Pediatrics	3	Pediatrics

Figure 3.3: Screenshot of view `v_patient_appointments`

- Reason
- Status

This view simplifies querying and is used by the application layer to quickly retrieve appointment-related data.

3.3.2 View 2: v_revenue_summary

department...	department_na...	total_appointme...	total_amount_d...	total_amount_p...	total_outstandi...
1	Cardiology	6	1750000.00	1700000.00	50000.00
2	Neurology	5	1150000.00	550000.00	600000.00
3	Pediatrics	6	1000000.00	1000000.00	0.00
4	Orthopedics	7	1900000.00	1500000.00	400000.00
5	Dermatology	6	1260000.00	1190000.00	70000.00

Figure 3.4: Screenshot of view `v_revenue_summary`

The view `v_revenue_summary` summarizes the financial performance of departments or time periods by aggregating data from `BILLING` and related tables.

It usually includes columns like:

- Department name
- Total amount due
- Total amount paid
- Outstanding balance

This view supports revenue analytics and helps administrators quickly evaluate departmental performance.

3.3.3 View 3: v_unpaid_bills

bill_id	patient_id	patient_name	appointment...	appointment_date	amount_due	amount_paid	amount_outstandi...	payment_stat...	payment_meth...	doctor_id	doctor_name	department...	department_na...
3	3	Le Van C	3	2025-01-07 11:00:00	250000.00	200000.00	50000.00	Partially Paid	cash	2	Dr. Tran Bao Long	1	Cardiology
5	5	Pham Van E	5	2025-01-09 14:00:00	220000.00	0.00	220000.00	Unpaid	cash	3	Dr. Le Thi Trang	2	Neurology
6	6	Nguyen Thi F	6	2025-01-10 15:00:00	300000.00	100000.00	200000.00	Partially Paid	cash	4	Dr. Pham Minh Hieu	2	Neurology
11	11	Nguyen Van K	11	2025-01-16 16:00:00	400000.00	0.00	400000.00	Unpaid	cash	7	Dr. Tran Hoai Nam	4	Orthopedics
15	15	Pham Van O	15	2025-01-20 14:00:00	230000.00	200000.00	30000.00	Partially Paid	cash	9	Dr. Pham Quang Huy	5	Dermatology
22	3	Le Van C	22	2025-01-24 13:00:00	180000.00	0.00	180000.00	Unpaid	cash	4	Dr. Pham Minh Hieu	2	Neurology
25	10	Pham Thi J	25	2025-01-24 16:00:00	190000.00	150000.00	40000.00	Partially Paid	card	9	Dr. Pham Quang Huy	5	Dermatology

Figure 3.5: Screenshot of view listing unpaid or partially paid bills

One of the essential operational views is `v_unpaid_bills_details` (or equivalent naming), which lists all bills that are either *Unpaid* or *Partially Paid*.

It includes key details such as:

- Patient name
- Appointment date
- Amount due
- Amount paid
- Payment status

This view is especially useful for the billing department to follow up on outstanding payments and link them directly to the patient, the involved doctor, and the date of service.

3.4 Implementation of Stored Procedures

This section presents the stored procedures that were implemented and verified inside the database.

3.4.1 Procedure 1: sp_create_appointment

The first procedure, `sp_create_appointment`, encapsulates the logic for creating a new appointment. It takes input parameters such as:

- `p_patient_id`
- `p_doctor_id`
- `p_appointment_date`
- `p_reason`

The procedure executes an `INSERT INTO APPOINTMENT` statement, applying necessary checks to ensure that the patient and doctor IDs exist and that the appointment date is valid. This centralizes the creation of appointments and ensures data integrity, instead of spreading appointment insert logic across multiple places in the application code.

```
DELIMITER //
CREATE PROCEDURE sp_create_appointment(
    IN p_patient_id INT,
    IN p_doctor_id INT,
    IN p_appointment_date DATETIME,
    IN p_reason VARCHAR(255),
    OUT p_new_appointment_id INT
)
BEGIN
    INSERT INTO Appointment (patient_id, doctor_id, appointment_date, reason, status)
    VALUES (p_patient_id, p_doctor_id, p_appointment_date, p_reason, 'Scheduled');

    SET p_new_appointment_id = LAST_INSERT_ID();
END //
DELIMITER ;
```

Figure 3.6: Screenshot of stored procedures for creating appointment

```
DELIMITER //
CREATE PROCEDURE sp_monthly_revenue_report(
    IN p_year INT,
    IN p_month INT
)
BEGIN
    SELECT
        dept.department_name,
        COUNT(b.bill_id) AS total_bills,
        SUM(b.amount_paid) AS total_paid,
        SUM(b.amount_due - b.amount_paid) AS total_unpaid
    FROM Billing b
    JOIN Appointment a ON b.appointment_id = a.appointment_id
    JOIN Doctor d ON a.doctor_id = d.doctor_id
    JOIN Department dept ON d.department_id = dept.department_id
    WHERE YEAR(b.payment_date) = p_year
      AND MONTH(b.payment_date) = p_month
    GROUP BY dept.department_id;
END //
DELIMITER ;
```

Figure 3.7: Screenshot of stored procedures for `sp_monthly_revenue_report`

3.4.2 Procedure 2: sp_monthly_revenue_report

The second procedure displayed is `sp_monthly_revenue_report`, which accepts year and month as input parameters and returns aggregated financial data (e.g., total billed amount, total paid amount) for that period.

This procedure internally queries the `BILLING` table and possibly `v_revenue_summary` to compute:

- Total revenue in the selected month.
- Total amount still unpaid.
- Number of bills processed.

This enables monthly financial reporting without requiring manual SQL queries.

3.5 Implementation of Triggers

The final section features the trigger implementation used to maintain automatic updates and ensure data correctness.

```
1   DELIMITER //
2   CREATE TRIGGER trg_update_bill_status
3   BEFORE UPDATE ON Billing
4   FOR EACH ROW
5   BEGIN
6       IF NEW.amount_paid >= NEW.amount_due THEN
7           SET NEW.payment_status = 'Paid';
8           SET NEW.payment_date = CURDATE();
9       ELSEIF NEW.amount_paid > 0 THEN
10          SET NEW.payment_status = 'Partially Paid';
11      ELSE
12          SET NEW.payment_status = 'Unpaid';
13      END IF;
14  END //
15  DELIMITER ;
16
17  SHOW TRIGGERS;
```

Figure 3.8: Screenshot of trigger `trg_update_bill_status` (original Word: Picture 7 – a screenshot of computer code, description automatically generated)

The screenshot displays the creation and confirmation of a trigger named `trg_update_bill_status`.

This trigger activates whenever a billing record is updated. Its main purpose is to adjust the `payment_status` field according to the amount that has been paid.

The trigger compares the updated `amount_paid` with the bill's total amount:

- If the `amount_paid` is greater than or equal to the total `amount_due`, the bill is automatically marked as *Paid*, and the payment date is filled in.
- If some payment has been made but the amount is still less than the total, the bill is labeled *Partially Paid*.
- If no payment has been made, the status remains *Unpaid*.

By automating this logic, the trigger ensures that the billing table always reflects accurate payment information.

Chapter 4

PYTHON APPLICATION AND DATA ANALYSIS

4.1 Database Connection and Code Structure (Modular)

The Python application was developed using a modular structure, where the database connection logic is separated from the data processing and visualization components.

- **Connection Library**: The `mysql.connector` library is used to connect to the MySQL server (see the `get_connection()` function in `visuallize.py`).
- **Parameters**: Connection constants (`HOST`, `PORT`, `USER`, `PASSWORD`, `DATABASE`) are centralized at the top of the `visuallize.py` file for easy configuration and maintenance.
- **Processing Library**: The `pandas` library is utilized to convert query results into DataFrames, facilitating subsequent analysis and visualization.

4.2 Graphical User Interface (GUI) Implementation

The software was developed with the goal of providing an intuitive GUI for users (e.g., administrative staff) to perform basic data management operations.

GUI features typically include:

- Simple forms for entering patient, doctor, and appointment data.
- Tables or lists showing existing records.
- Buttons for creating, updating, and deleting entries.
- Integration with the reporting features for quickly accessing summary data.

4.3 Required Query/Report Pages

This section demonstrates the required SQL reports that were implemented in the database and integrated into the Python application. Each report showcases a specific type of JOIN or analytical query.

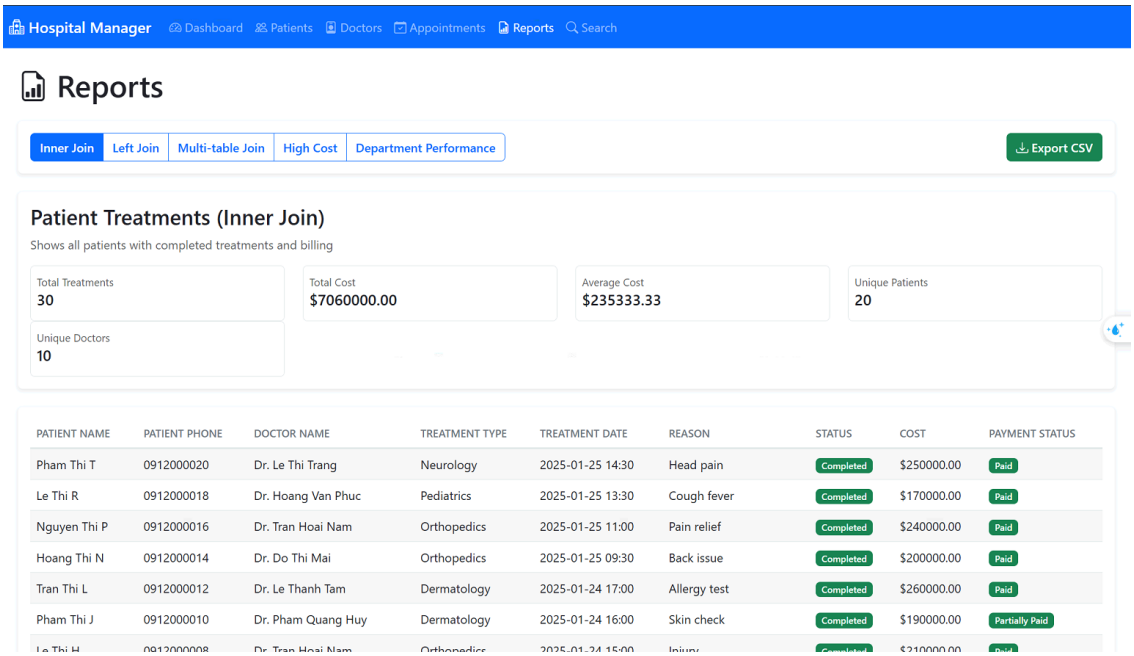

Figure 4.1: Screenshot of report/query pages (original Word: Picture 1 – a screenshot of a medical form, description automatically generated)

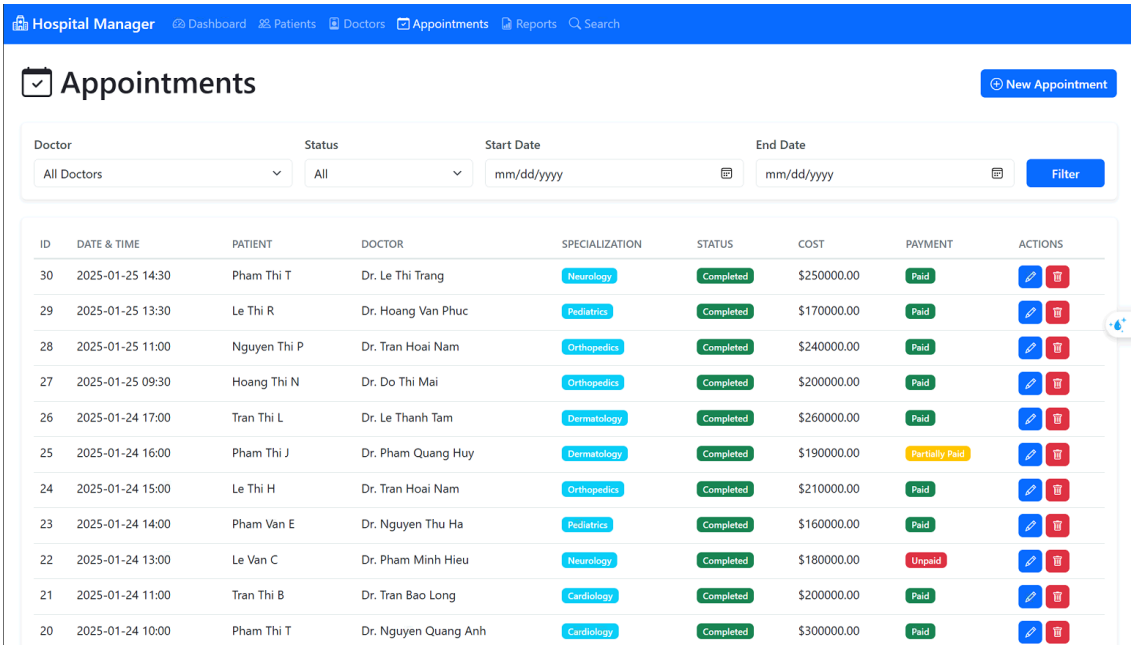

Figure 4.2: Screenshot of Appointment table

4.3.1 INNER JOIN: Patient name, treatment name, treatment date, and cost

This report performs an `INNER JOIN` query among the `PATIENT`, `APPOINTMENT`, and `BILLING` (or `TREATMENT`) tables to retrieve details about a patient's treatment cost and date.

The report shows:

- Patient name
- Treatment / service
- Treatment date
- Cost

4.3.2 LEFT JOIN: Show all patients and their treatments/costs

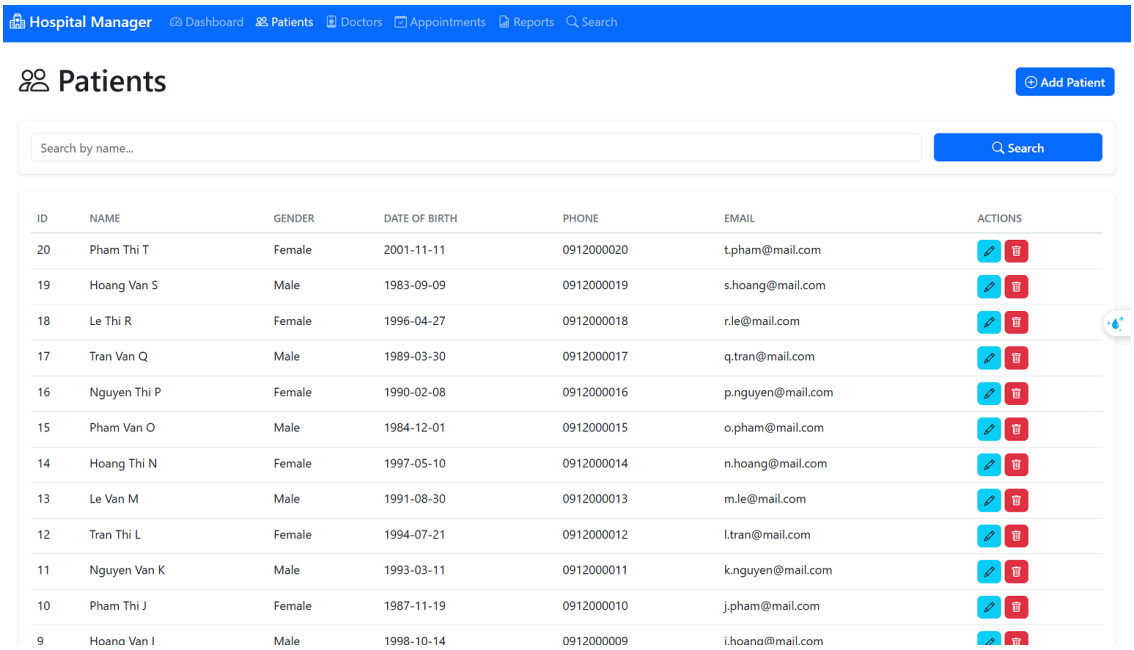

Figure 4.3: Screenshot of Patient table

This report uses a `LEFT JOIN` to display the complete list of patients, including those who do not have any appointments or bills, ensuring no records are missed.

4.3.3 Multi-table JOIN: Patient name, doctor name, treatment, cost (3+ tables)

A complex multi-table JOIN combines 4 tables:

- `PATIENT`
- `APPOINTMENT`
- `DOCTOR`
- `BILLING` (or `MEDICAL_RECORD`/`TREATMENT`)

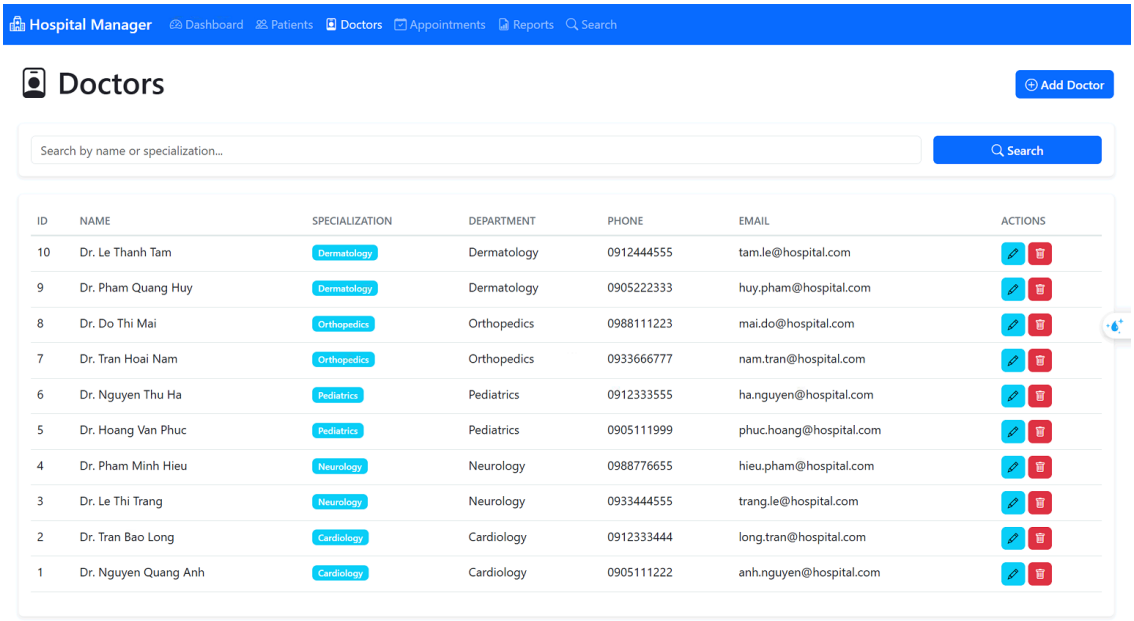

Figure 4.4: Screenshot of Doctor table

The output includes:

- Patient name
- Doctor name
- Service/treatment description
- Cost

4.3.4 High-cost treatments: Cost > average treatment cost

This query uses a nested query or subquery to filter out treatment sessions whose cost is higher than the average treatment cost across the entire hospital.

It is important for:

- Financial analysis
- Pricing review
- Identifying potential outliers in service charges

4.4 Dashboard and Data Visualization

This section uses the `visuallize.py` source code to demonstrate how the Python application connects and visualizes critical data using the `matplotlib` library.

The visualizations focus on three main aspects:

- Department revenue
- Appointment status distribution
- Monthly revenue trend

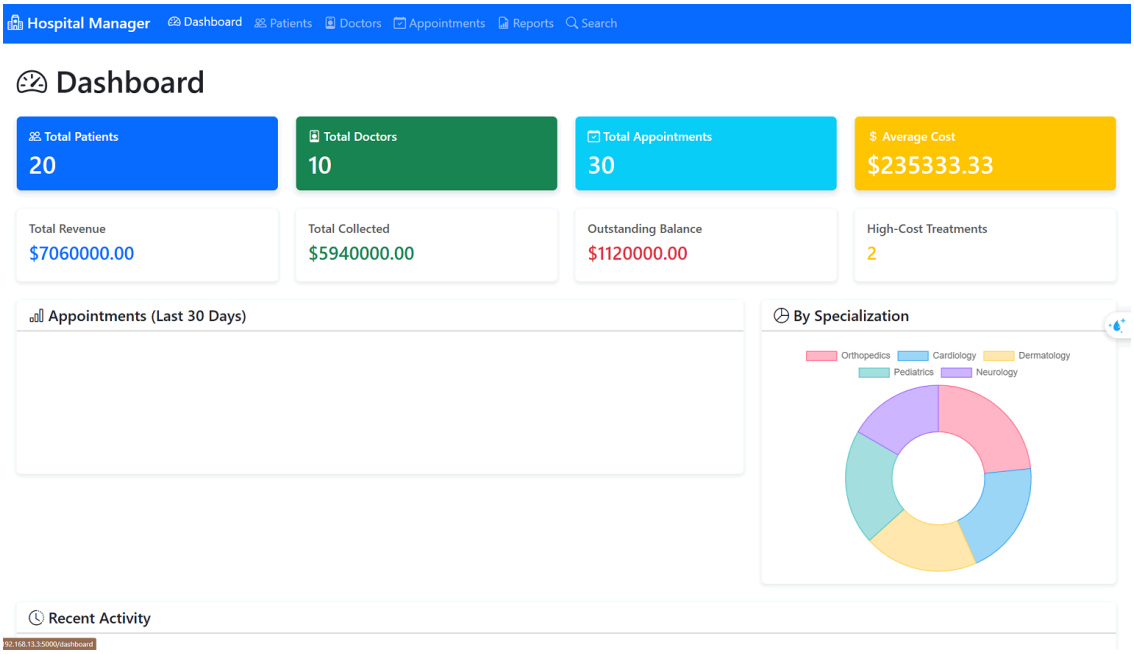

Figure 4.5: Screenshot of Dashboard

4.4.1 Department Revenue Visualization (View v_department_revenue)

Functionality: The `load_department_revenue()` function reads aggregated data from the view `v_department_revenue`.

Chart: The `plot_department_revenue(df)` function creates a bar chart that shows total revenue per department. This helps management identify departments with the highest financial performance.

4.4.2 Appointment Status Analysis

Functionality: The `load_appointment_stats()` function queries the total number of appointments by status.

Chart: The `plot_appointment_status(df)` function uses a bar chart to visualize the ratio of completed, scheduled, and cancelled appointments.

4.4.3 Monthly Revenue Report (Stored Procedure sp_monthly_revenue_report)

Functionality: The `call_sp_monthly_revenue(year, month)` function calls the stored procedure `sp_monthly_revenue_report` in the database, passing in year and month as input parameters.

Value: This demonstrates the flexible integration capability between the Python application and complex business logic predefined in the database.

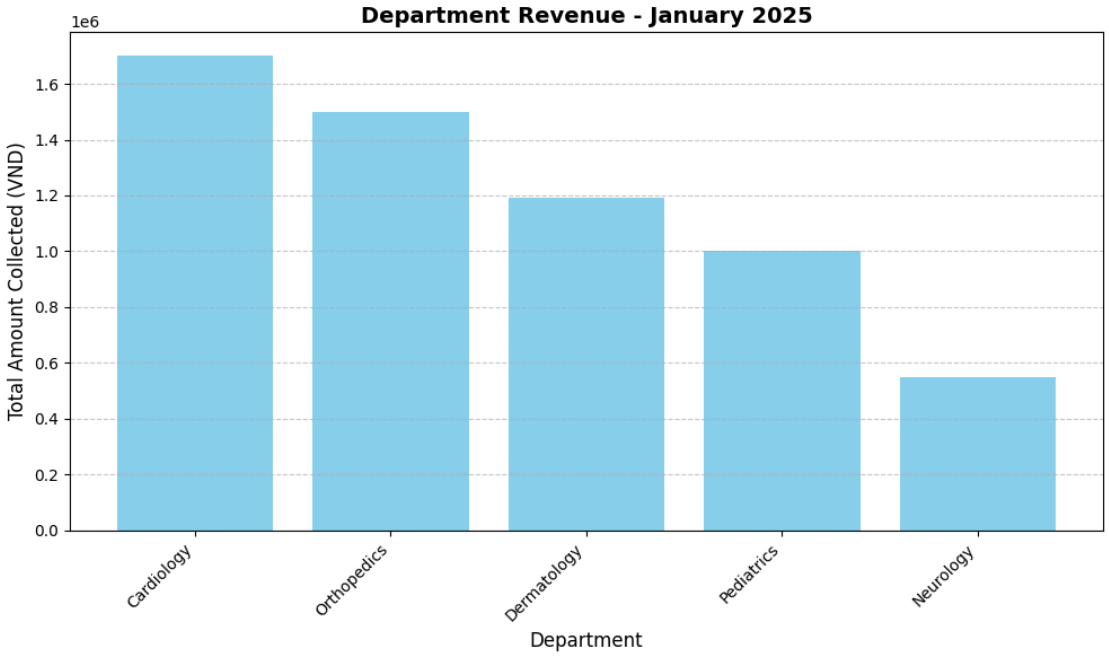

Figure 4.6: Department revenue bar chart (original Word: Picture 2 – a screenshot of a computer, description automatically generated)

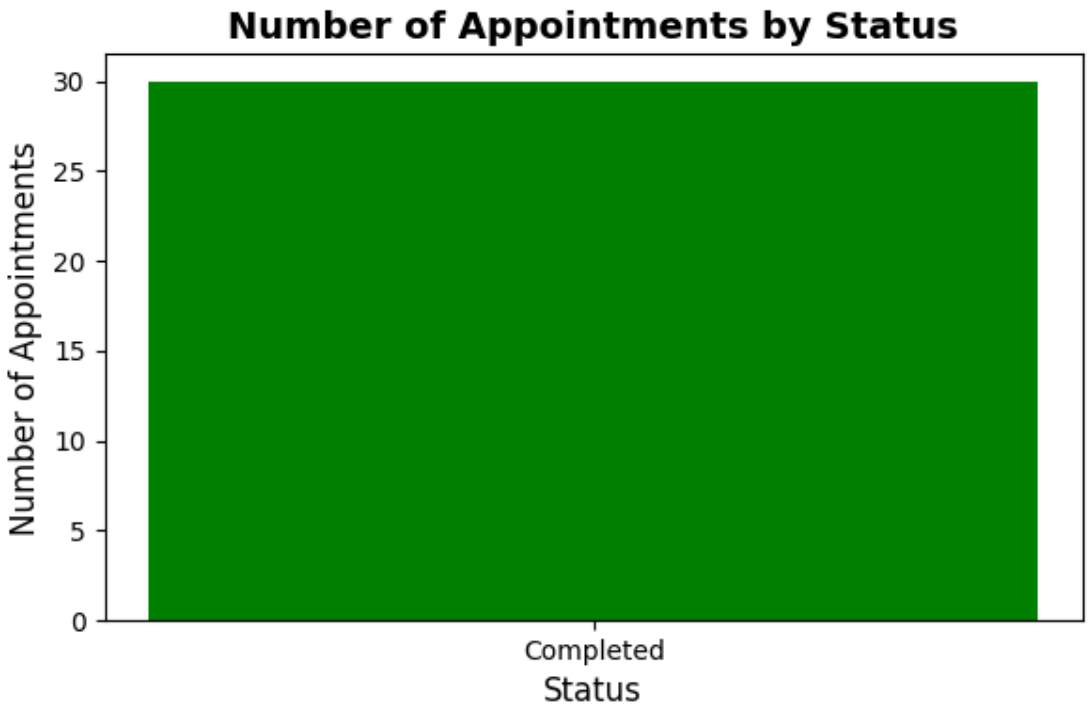

Figure 4.7: Appointment status distribution chart (original Word: Picture 7 – a graph of blue bars, description automatically generated)

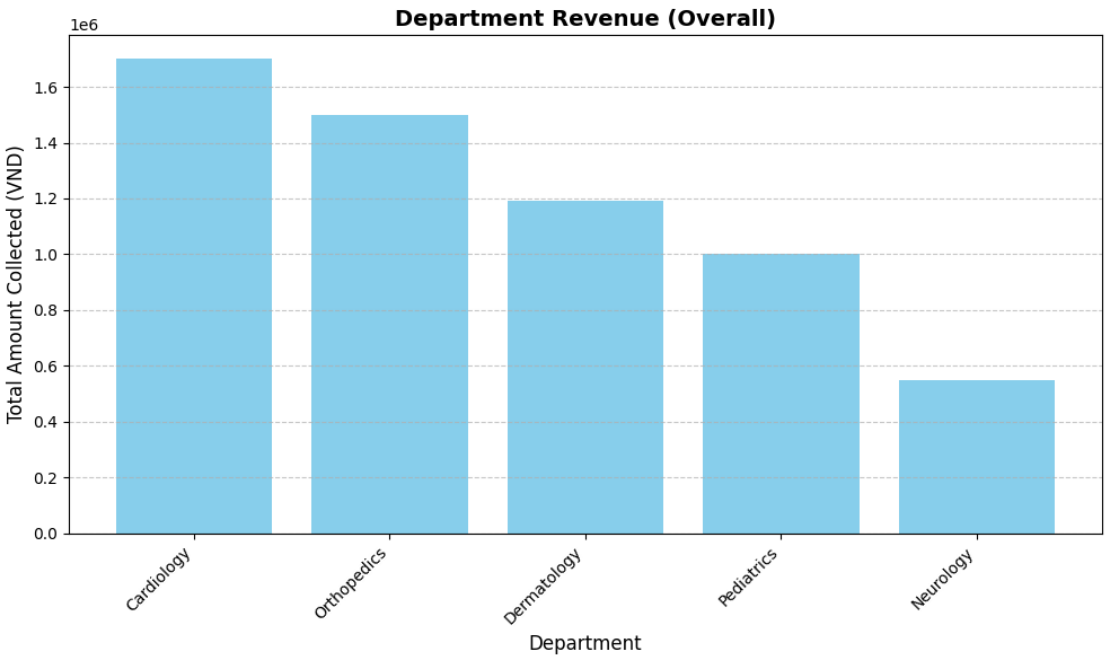

Figure 4.8: Monthly revenue visualization (original Word: Picture 9 – a green rectangular object with white text, description automatically generated)

Chapter 5

CONCLUSION AND LESSONS LEARNED

5.1 Project Summary and Achievements (Meeting Acceptance Criteria)

The project successfully achieved all objectives set for a basic Hospital Patient Management System:

- **Data Design**: Ensured data normalization to 3NF, with a clear ERD design and complete PK/FK constraints.
- **Logic Implementation**: Successfully utilized advanced SQL objects:
  - **Views**: For creating aggregated and complex reports.
  - **Stored Procedures**: For encapsulating business processes (e.g., creating appointments, periodic reports).
  - **Triggers**: For automating status updates (e.g., automatically updating bill payment status).
- **Practical Application**: The Python application demonstrated CRUD operations and data visualization, making the system ready for integration into a real-world working environment.

5.2 Lessons Learned and Team Contribution

Throughout the project, the team learned several important lessons:

- **Importance of Normalization**: The most significant lesson learned was the value of properly analyzing Functional Dependencies and systematically applying normal forms, especially the identification and resolution of transitive dependencies to achieve 3NF.
- **Efficiency of Stored Procedures**: Encapsulating complex queries inside stored procedures improves performance (as they are pre-compiled) and protects the integrity of the business logic.
- **Integration Skills**: The project enhanced the team's ability to integrate different layers: database (MySQL), application (Python), and visualization (matplotlib/pandas), ensuring synchronization and end-to-end data processing capability.

5.3 Limitations and Future Work

Limitations:

- **Scale**: The current sample data (seed data) is minimal and primarily for functional testing.
- **Security**: The database connection information (root/password) in the Python code may still be hard-coded; a safer secret management approach should be implemented.

Future Work:

- **Database Optimization**: Consider further normalization to BCNF (Boyce-Codd Normal Form) if additional dependencies are discovered.
- **Feature Expansion**: Develop stored procedures for more complex transactions and operations (e.g., managing drug inventory, detailed medical records).
- **Web API Development**: Build an API (e.g., using Flask/Django) to allow external systems (such as web or mobile apps) to access and use the data securely and efficiently.