Hospital Patient Record & Billing System (CLI Application)

Manish Kumar (2633550)

MAY 22, 2025

MPHASIS LTD
PUNE, MAHARASTRA

1. Introduction

In today’s healthcare environment, managing patient records, doctor schedules, services, and billing manually can be time-consuming and error- prone. The Hospital Patient Record & Billing System is a command-line interface (CLI) application developed using Core Python and MySQL Workbench to address these challenges. It provides a digital solution for hospitals to manage their administrative operations efficiently.

This system is designed to be lightweight and modular, making it ideal for small to mid-sized hospitals or clinics. It automates key processes such as patient registration, appointment scheduling, service tracking, and billing, while ensuring data accuracy and security.

In today’s fast-paced healthcare environment, the efficient management of patient records, doctor schedules, medical services, and billing is critical to delivering timely and quality care. However, many healthcare facilities, especially small to mid-sized hospitals and clinics, still rely on manual processes or outdated systems. These methods are not only time-consuming but also prone to human error, data loss, and inefficiencies that can impact patient care and administrative productivity.

To address these challenges, the **Hospital Patient Record & Billing System** was developed as a **command-line interface (CLI) application** using **Core Python** and **MySQL Workbench**. This system provides a comprehensive digital solution that automates and streamlines hospital administrative operations. It is designed to be lightweight, modular, and easy to deploy, making it an ideal choice for healthcare institutions with limited technical infrastructure or budget.

2. Project Objectives

The main objectives of the Hospital Patient Record & Billing System project are as follows:

- Automation: Replace traditional paper-based or manual record-keeping systems with a fully digital solution that automates patient registration, appointment scheduling, service tracking, and billing.
- Efficiency: Streamline hospital workflows by reducing repetitive administrative tasks, thereby saving time and allowing staff to focus more on patient care.
- Accuracy: Ensure high data integrity through input validation, error handling, and prevention of duplicate entries. This reduces the chances of billing errors, misdiagnoses, or lost records.
- Accessibility: Provide a lightweight, command-line interface (CLI) that can run on any system with Python installed, without the need for high-end hardware or graphical environments.
- Scalability: Design the system using a modular architecture that allows for easy expansion. Future enhancements such as a graphical user interface (GUI), cloud integration, or mobile app support can be added without major restructuring.
- Security: Although basic in this version, the system lays the groundwork for implementing secure access controls, ensuring that sensitive patient data is protected from unauthorized access.
- Portability: The application is platform-independent and can be deployed on any operating system that supports Python and MySQL, making it suitable for diverse IT environments.
- Maintainability: The codebase is organized into separate modules for each functionality (patients, doctors, appointments, billing, etc.), making it easier to debug, update, and maintain.
- Reporting and Analytics: Enable the generation of daily reports and summaries, which can be exported in CSV format for further analysis or archival purposes.

3. Key Features Explained

Patient Management

- Add, update, delete, and search patient records.
- Calculate the number of days a patient has been admitted.
- Track historical appointments and diagnoses.

Doctor Management

- Maintain a list of doctors with their specializations and contact details.
- Assign doctors to patients during appointment creation.

Appointment Scheduling

- Schedule appointments by linking patients and doctors.
- Record diagnosis details.
- Filter appointments by date (e.g., today, last 7 days).

Billing System

- Automatically calculate total charges based on services used and consultation fees.
- Generate and save invoices in text format.
- Store billing records in the database with timestamps.

Reporting and Export

- Generate daily reports for appointments and billing.
- Export summaries to CSV files for administrative use.

Smart ID System

- Uses formatted IDs like PAT-XXXXXX for patients and APT-XXXXXX for appointments to ensure uniqueness and readability.

4. Technical Stack

- Language: Python 3.x
- Database: MySQL Workbench
- Libraries:
    - mysql.connector – for database connectivity
    - datetime – for date calculations
    - pandas – for using Datasets
    - re – for file handling and input validation

5. System Architecture & Folder Structure

The project is organized into modular Python files, each responsible for a specific functionality. This promotes clean code, easy debugging, and future scalability.

```
Hospital_management_system/
├── db_config.py               # Database connection logic
├── patients.py               # Patient class and logic
├── appointments.py           # Appointment handling
├── doctors.py                # Doctor management
├── services.py               # Medical services logic
├── patient_billing_tracking.py   # Billing and invoice generation
├── hospital_main.py          # Main CLI dashboard
├── validations.py            # Input validation functions
├── requirements.txt          # Required libraries
├── output/
│   └── invoices/           # Generated invoice files
│   └── CSVs/               # Exported CSV summaries
└── README.md               # Project documentation
```

6. Implementation Details

Step 1: Environment Setup

- Install Python 3.x
- Install required libraries using:
- pip install mysql.connector

Step 2: Database Configuration

- Create a database named hospital.
- Create the following tables:
    - patients(patient_id, name, age, gender, admission_date, contact_no)
    - doctors(doctor_id, name, specialization, contact_no)
    - services(service_id, service_name, cost)
    - appointments(appt_id, patient_id, doctor_id, date, diagnosis)
    - billing(bill_id, patient_id, total_amount, billing_date)
    - billing_services(bill_id,service_id)

1. `patients` Table

Purpose: Stores personal and admission details of patients.

Field Name	Data Type	Description
`patient_id`	VARCHAR	Unique identifier for each patient (e.g., PAT-100001).
`Name`	VARCHAR	Full name of the patient.
Age	INT	Age of the patient.

Gender	Varchar	Gender of the patient (MaleFemale/Other).
admission_date	DATE	Date when the patient was admitted.
contact_no	VARCHAR	Patient's contact number (validated for format).

---

2. doctors Table

Purpose: Stores information about doctors available in the hospital.

Field Name	Data Type	Description
doctor_id	VARCHAR	Unique identifier for each doctor (e.g., DOC-001).
Name	VARCHAR	Full name of the doctor.
specialization	VARCHAR	Medical specialty (e.g., Cardiology, Neurology).
contact_no	VARCHAR	Doctor's contact number.

---

3. services Table

Purpose: Contains a list of medical services offered by the hospital.

Field Name	Data Type	Description
service_id	VARCHAR	Unique ID for each service (e.g., SER-001).
service_name	VARCHAR	Name of the service (e.g., Blood Test, X-Ray).
cost	DECIMAL	Cost of the service in local currency.

---

4. `appointments` Table

Purpose: Records appointments between patients and doctors, including diagnosis.

Field Name	Data Type	Description
`appt_id`	VARCHAR	Unique appointment ID (e.g., APT-0001).
`patient_id`	VARCHAR	Foreign key referencing `patients.patient_id`.
`doctor_id`	VARCHAR	Foreign key referencing `doctors.doctor_id`.
`date`	DATE	Date of the appointment.
`diagnosis`	TEXT	Diagnosis or notes from the doctor.

---

5. `billing` Table

Purpose: Stores billing records for patients.

Field Name	Data Type	Description
`bill_id`	VARCHAR	Unique billing ID (e.g., BILL-0001).
`patient_id`	VARCHAR	Foreign key referencing `patients.patient_id`.
`total_amount`	DECIMAL	Total amount billed to the patient.
`billing_date`	DATE	Date when the bill was generated.

---

6. `billing_services` Table

Purpose: A junction table to map multiple services to a single bill (many-to-many relationship).

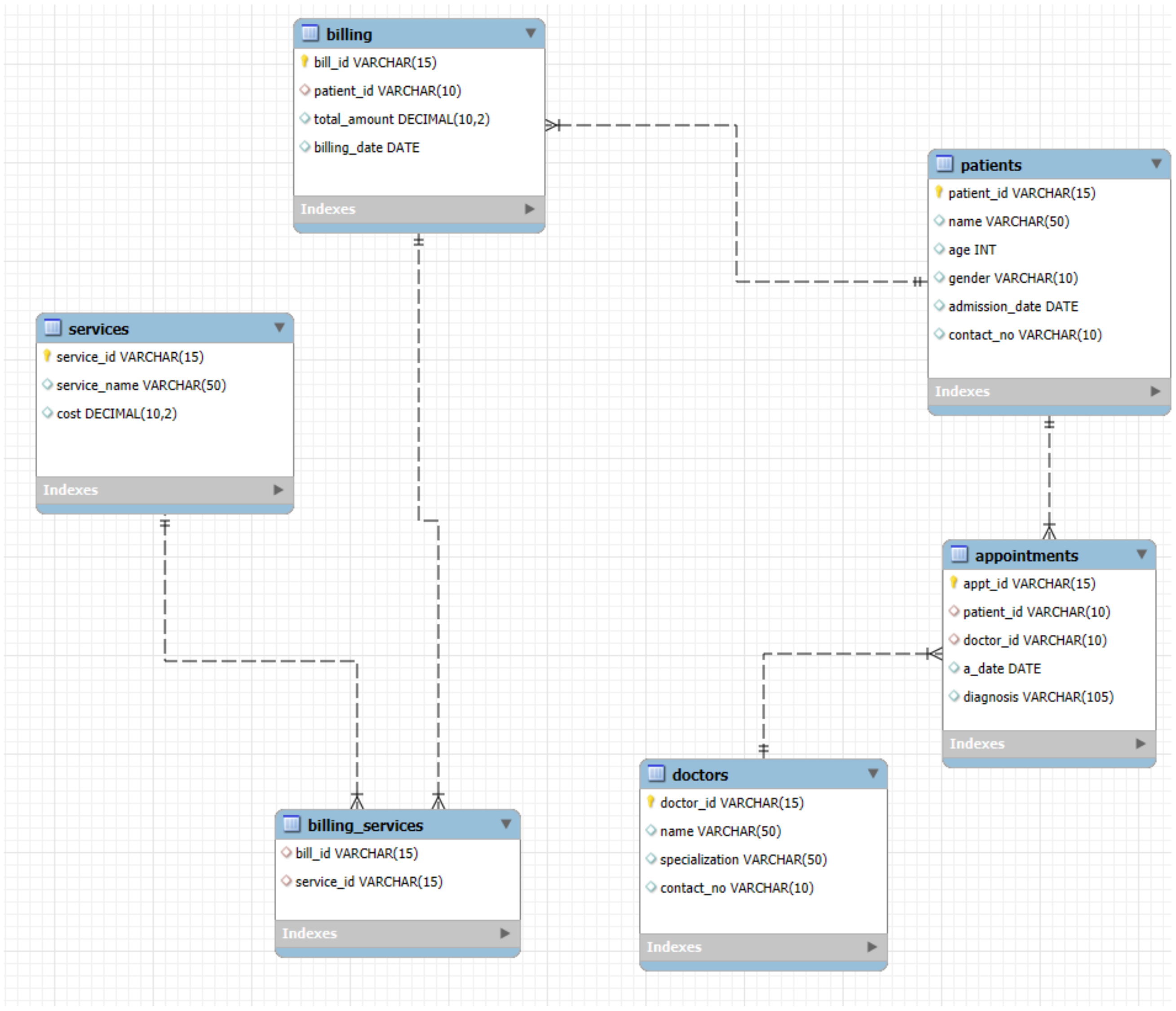
Field Name	Data Type	Description
`bill_id`	VARCHAR	Foreign key referencing `billing.bill_id`.
`service_id`	VARCHAR	Foreign key referencing `services.service_id`.

📌 *Why this table is needed*:
A patient may use multiple services during a visit, and each service must be linked to the bill. This table allows tracking of all services billed under a single invoice

SQL Queries:

```
CREATE DATABASE Hospital;

USE Hospital;

CREATE TABLE Patients (

   patient_id VARCHAR(10) PRIMARY KEY,

   name VARCHAR(50),

   age INT,

   gender VARCHAR(10),

   admission_date DATE,

   contact_no VARCHAR(10)

);


CREATE TABLE Doctors (

   doctor_id VARCHAR(10) PRIMARY KEY,

   name VARCHAR(50),

   specialization VARCHAR(50),
   contact_no VARCHAR(10)
);

CREATE TABLE Services (
   service_id VARCHAR(10) PRIMARY KEY,
   service_name VARCHAR(50),
   cost DECIMAL(10, 2)
);

CREATE TABLE Appointments (
   appt_id VARCHAR(10) PRIMARY KEY,
   patient_id VARCHAR(10),
   doctor_id VARCHAR(10),
   date DATE,
   diagnosis VARCHAR(105),
   FOREIGN KEY (patient_id) REFERENCES Patients(patient_id),
   FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id)
);
CREATE TABLE billing (
   bill_id VARCHAR(10) PRIMARY KEY,
   patient_id VARCHAR(10),
   total_amount DECIMAL(10, 2),
   billing_date DATE,
   FOREIGN KEY (patient_id) REFERENCES Patients(patient_id)
);
```

```
CREATE TABLE billing_services (
        bill_id VARCHAR(15),
  service_id VARCHAR(15),
  FOREIGN KEY (bill_id) REFERENCES billing(bill_id),
  FOREIGN KEY (service_id) REFERENCES services(service_id)
);
```



Step 3: Running the Application

- Launch the CLI app using hospital_main.py.
- Use the menu to navigate between modules and perform operations.

Launch the CLI Application

Open your terminal or command prompt.

Navigate to the project directory where hospital_main.py is located.

Run the application using the command:

python hospital_main.py

Ensure that all required dependencies are installed beforehand (e.g., via requirements.txt if provided).

7. Output & File Management

- Invoices: Automatically generated and saved in output/invoices/ with filenames like invoice_PAT-100001.txt.
- CSV Reports: Appointment and billing summaries are exported to output/CSVs/ for administrative use.
- Custom Paths: The system allows users to specify custom paths for saving files.

Here are two-line definitions for each menu item:

---

1. Patients
Manage patient records including personal details, medical history, and admission status.

2. Doctors
Maintain doctor profiles, specializations, and assign them to patient appointments.

3. Services
Define and manage hospital services like lab tests, surgeries, and consultations.

4. Appointments
Schedule, update, and track appointments between patients and doctors.

5. Billing
Generate and manage invoices based on services and consultations provided.

6. Patient Billing Tracking
View and track individual patient billing history and outstanding payments.

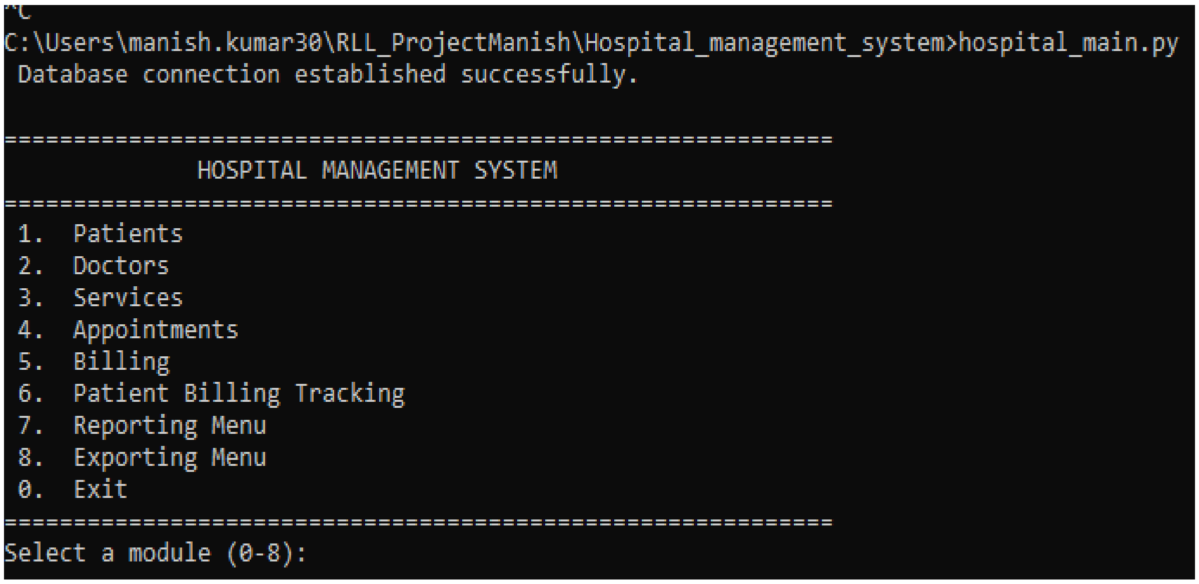
7. Reporting Menu
Generate reports on appointments, billing, and hospital performance metrics.

8. Exporting Menu
Export data and reports to CSV or Excel for administrative or backup use.

0. Exit
Safely close the application and terminate all active sessions.

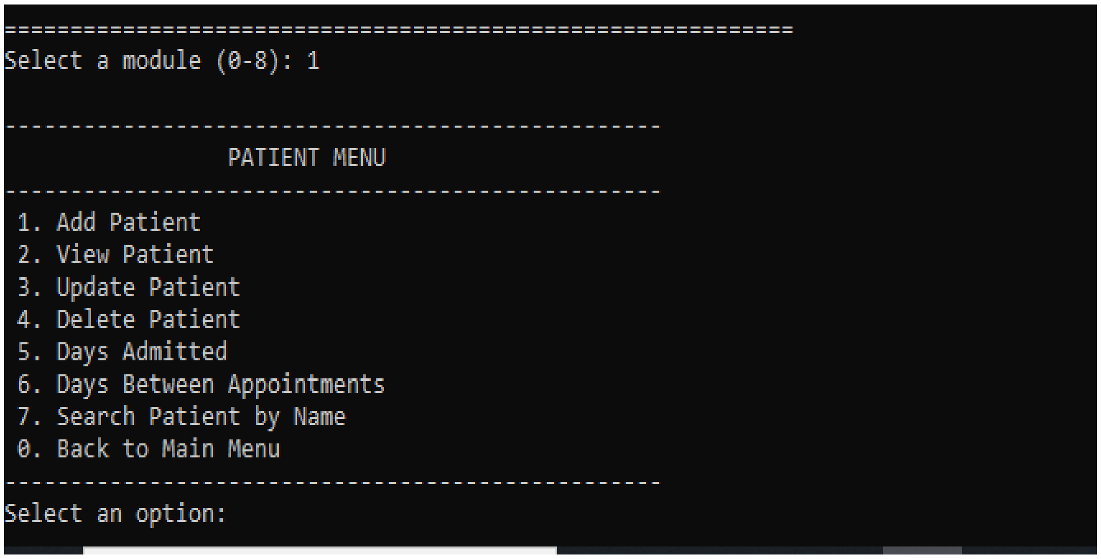
Outputs

Main Menu:



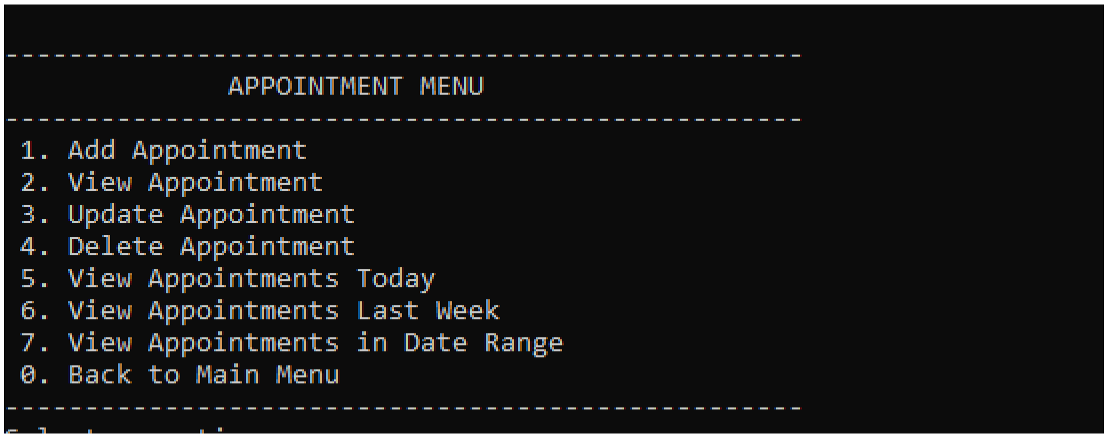
```
^C
C:\Users\manish.kumar30\RLL_ProjectManish\Hospital_management_system>hospital_main.py
 Database connection established successfully.

============================================================
              HOSPITAL MANAGEMENT SYSTEM
============================================================
 1.  Patients
 2.  Doctors
 3.  Services
 4.  Appointments
 5.  Billing
 6.  Patient Billing Tracking
 7.  Reporting Menu
 8.  Exporting Menu
 0.  Exit
============================================================
Select a module (0-8):
```

Patient Menu:



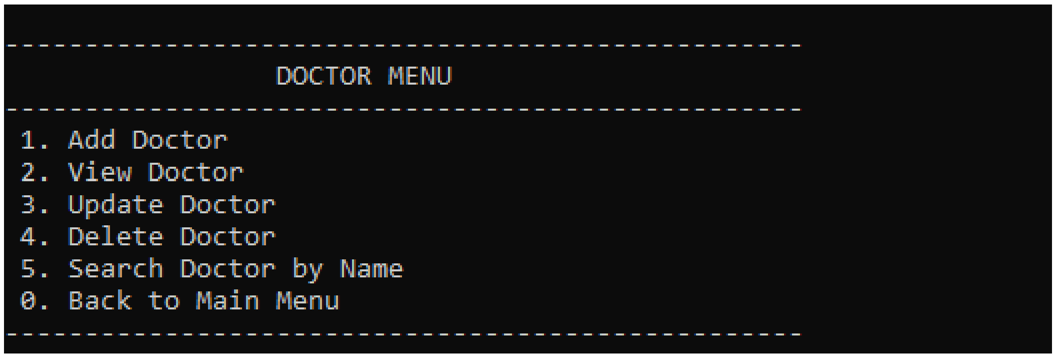
```
============================================================
Select a module (0-8): 1

--------------------------------------------------
                 PATIENT MENU
--------------------------------------------------
 1. Add Patient
 2. View Patient
 3. Update Patient
 4. Delete Patient
 5. Days Admitted
 6. Days Between Appointments
 7. Search Patient by Name
 0. Back to Main Menu
--------------------------------------------------
Select an option:
```

Appointments Menu:



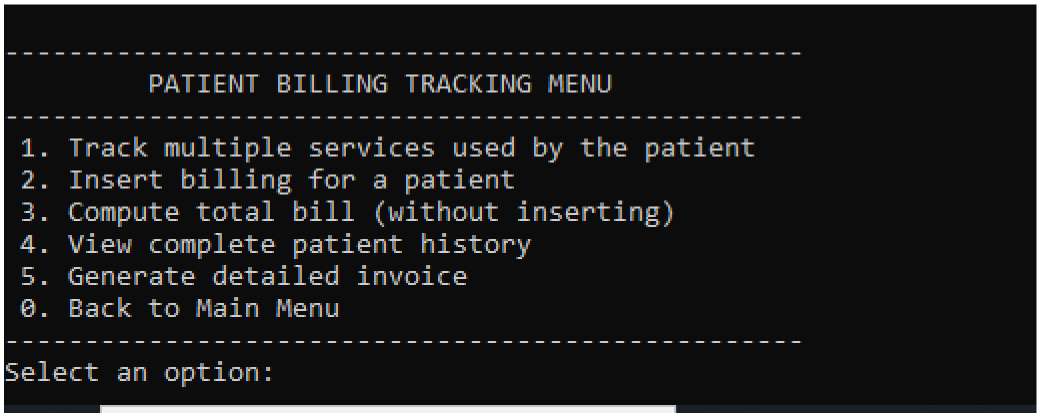
```
--------------------------------------------------
              APPOINTMENT MENU
--------------------------------------------------
 1. Add Appointment
 2. View Appointment
 3. Update Appointment
 4. Delete Appointment
 5. View Appointments Today
 6. View Appointments Last Week
 7. View Appointments in Date Range
 0. Back to Main Menu
--------------------------------------------------
```

Doctors Menu:



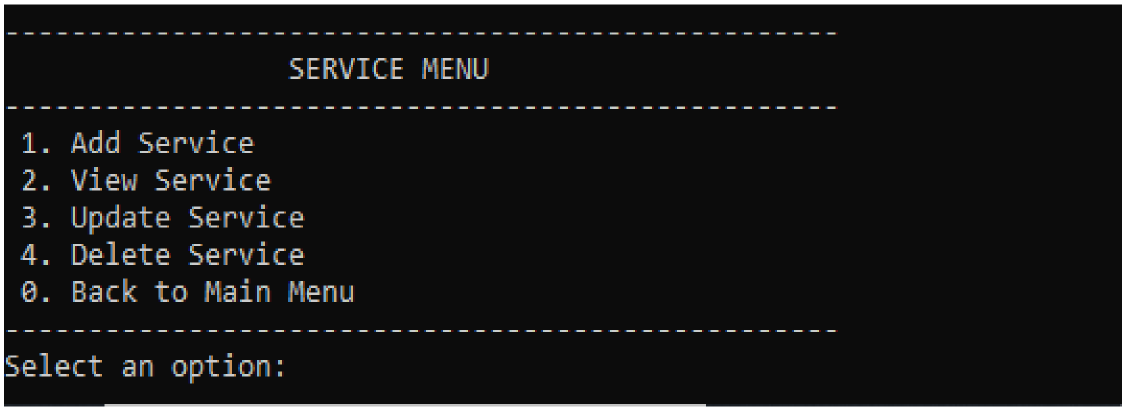
```
--------------------------------------------------
                 DOCTOR MENU
--------------------------------------------------
 1. Add Doctor
 2. View Doctor
 3. Update Doctor
 4. Delete Doctor
 5. Search Doctor by Name
 0. Back to Main Menu
--------------------------------------------------
```

Billing Tracking Menu:



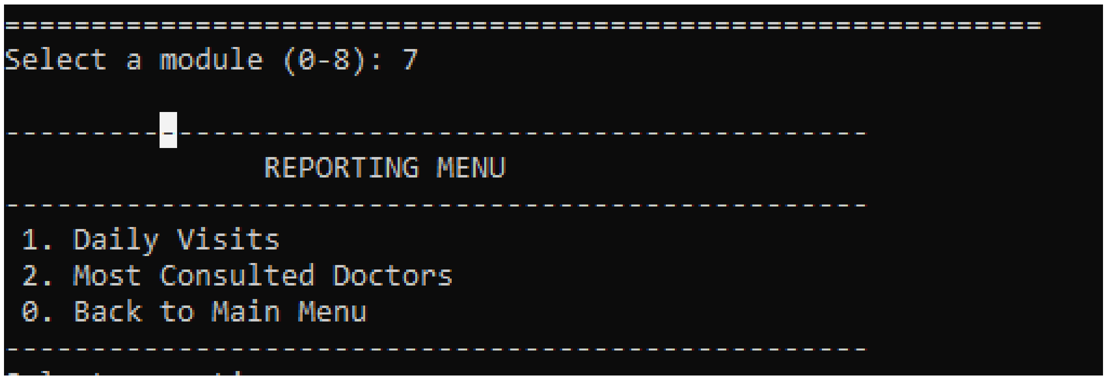
```
--------------------------------------------------
         PATIENT BILLING TRACKING MENU
--------------------------------------------------
 1. Track multiple services used by the patient
 2. Insert billing for a patient
 3. Compute total bill (without inserting)
 4. View complete patient history
 5. Generate detailed invoice
 0. Back to Main Menu
--------------------------------------------------
Select an option:
```

Services Menu:



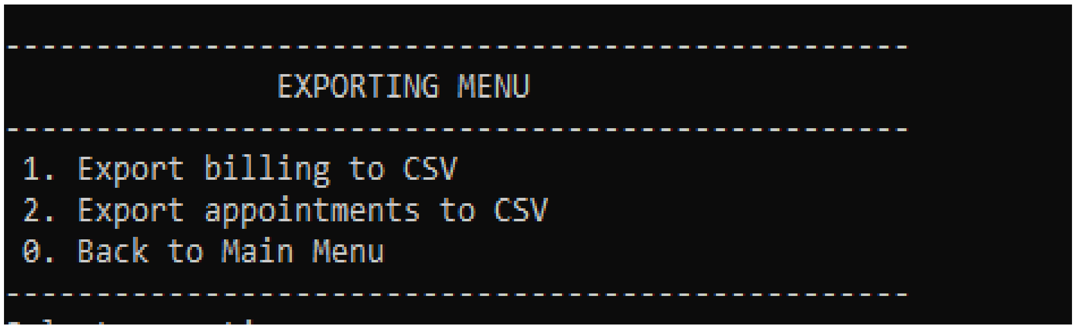
```
--------------------------------------------------
                 SERVICE MENU
--------------------------------------------------
 1. Add Service
 2. View Service
 3. Update Service
 4. Delete Service
 0. Back to Main Menu
--------------------------------------------------
Select an option:
```

Reporting Menu:



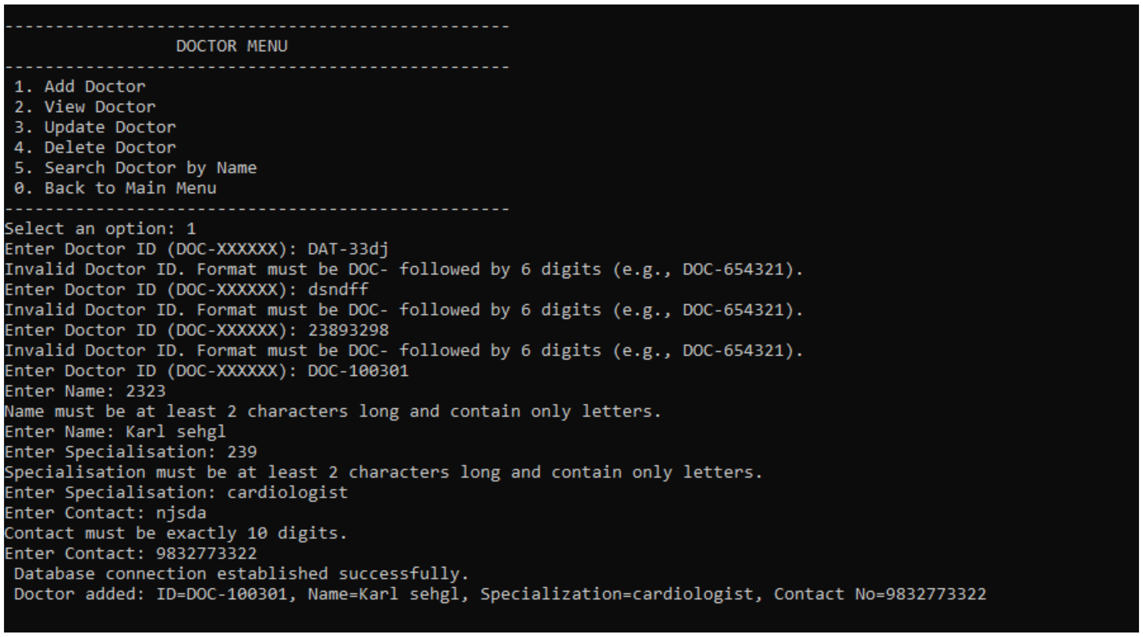
```
==============================================================
Select a module (0-8): 7

--------------------------------------------------
                REPORTING MENU
--------------------------------------------------
 1. Daily Visits
 2. Most Consulted Doctors
 0. Back to Main Menu
--------------------------------------------------
```

Exporting Menu:



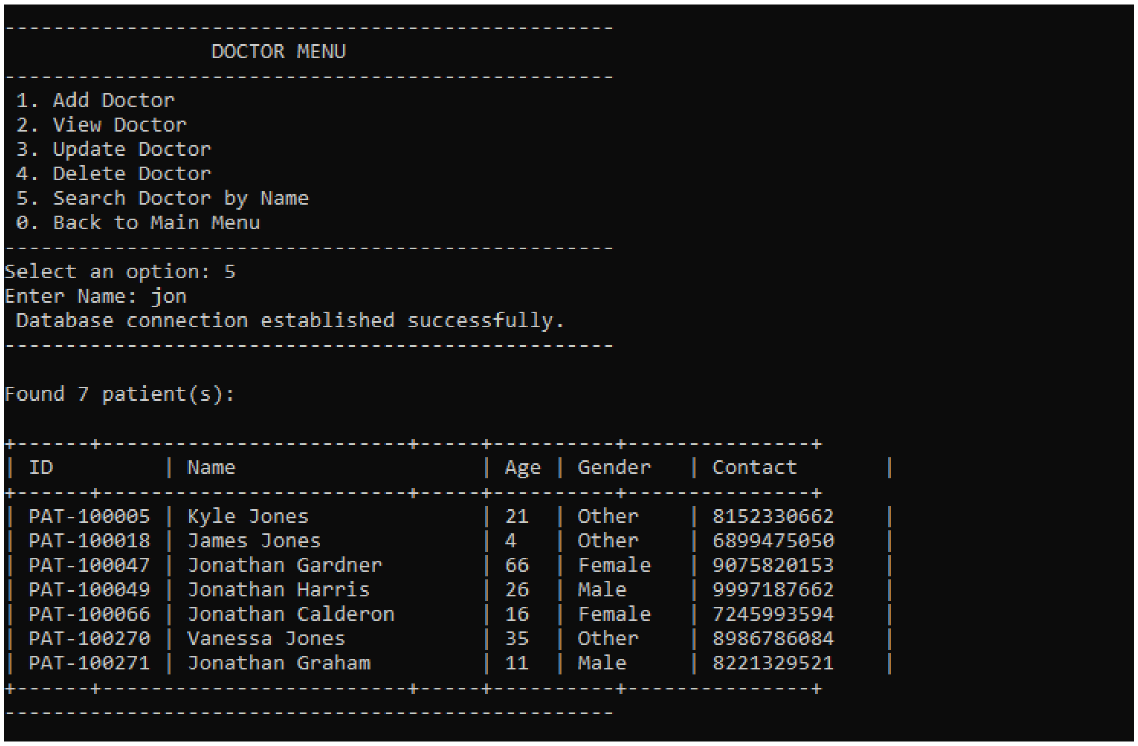
```
--------------------------------------------------
                EXPORTING MENU
--------------------------------------------------
 1. Export billing to CSV
 2. Export appointments to CSV
 0. Back to Main Menu
--------------------------------------------------
```

Entering Doctor with Validations:



```
--------------------------------------------------
                 DOCTOR MENU
--------------------------------------------------
 1. Add Doctor
 2. View Doctor
 3. Update Doctor
 4. Delete Doctor
 5. Search Doctor by Name
 0. Back to Main Menu
--------------------------------------------------
Select an option: 1
Enter Doctor ID (DOC-XXXXXX): DAT-33dj
Invalid Doctor ID. Format must be DOC- followed by 6 digits (e.g., DOC-654321).
Enter Doctor ID (DOC-XXXXXX): dsndff
Invalid Doctor ID. Format must be DOC- followed by 6 digits (e.g., DOC-654321).
Enter Doctor ID (DOC-XXXXXX): 23893298
Invalid Doctor ID. Format must be DOC- followed by 6 digits (e.g., DOC-654321).
Enter Doctor ID (DOC-XXXXXX): DOC-100301
Enter Name: 2323
Name must be at least 2 characters long and contain only letters.
Enter Name: Karl sehgl
Enter Specialisation: 239
Specialisation must be at least 2 characters long and contain only letters.
Enter Specialisation: cardiologist
Enter Contact: njsda
Contact must be exactly 10 digits.
Enter Contact: 9832773322
 Database connection established successfully.
 Doctor added: ID=DOC-100301, Name=Karl sehgl, Specialization=cardiologist, Contact No=9832773322
```

Function of searching for doctors:



```
--------------------------------------------------
                 DOCTOR MENU
--------------------------------------------------
 1. Add Doctor
 2. View Doctor
 3. Update Doctor
 4. Delete Doctor
 5. Search Doctor by Name
 0. Back to Main Menu
--------------------------------------------------
Select an option: 5
Enter Name: jon
 Database connection established successfully.
--------------------------------------------------

Found 7 patient(s):

+------+-------------------------+-----+----------+---------------+
| ID         | Name                      | Age | Gender    | Contact        |
+------+-------------------------+-----+----------+---------------+
| PAT-100005 | Kyle Jones                | 21  | Other     | 8152330662     |
| PAT-100018 | James Jones               | 4   | Other     | 6899475050     |
| PAT-100047 | Jonathan Gardner          | 66  | Female    | 9075820153     |
| PAT-100049 | Jonathan Harris           | 26  | Male      | 9997187662     |
| PAT-100066 | Jonathan Calderon         | 16  | Female    | 7245993594     |
| PAT-100270 | Vanessa Jones             | 35  | Other     | 8986786084     |
| PAT-100271 | Jonathan Graham           | 11  | Male      | 8221329521     |
+------+-------------------------+-----+----------+---------------+
--------------------------------------------------
```

8. Conclusion

This project successfully demonstrates how Python and SQL can be integrated to build a robust, real-world hospital management system. By automating critical administrative tasks such as patient registration, appointment scheduling, billing, and medical record management, the system significantly reduces human error and enhances operational efficiency.

The use of a Command-Line Interface (CLI) ensures accessibility in environments with limited computing resources or where graphical interfaces may not be practical. This makes the system especially valuable for rural or underfunded healthcare facilities, where simplicity and reliability are paramount.

From a technical perspective, Python provides a flexible and powerful programming environment for implementing business logic, while SQL ensures efficient and secure data storage and retrieval. Together, they form a scalable foundation that can be extended to include advanced features such as:

- **Role-based access control** to protect sensitive patient data.
- **Real-time reporting and analytics** to support decision-making.
- **Integration with external systems** like laboratory information systems or insurance databases.
- **Audit trails and logging** for compliance and accountability.

9. Future Scope

- GUI Integration: Develop a graphical interface using Tkinter or PyQt for better usability.
- Cloud Deployment: Host the system on a cloud platform for remote access and scalability.
- Security Enhancements: Implement user authentication and role-based access control.
- Analytics Dashboard: Add visual reports and charts for better insights.
- Multi-language Support: Enable localization for use in different regions.