HOSPITAL PATIENT RECORD & BILLING SYSTEM

PROJECT DOCUMENTATION

Table of Contents

ABSTRACT

The Hospital Patient Record & Billing System (or) Hospital Management System (HMS) is a comprehensive, menu-driven application designed to streamline and automate the administrative and operational tasks within a hospital environment. Built using Core Python and MySQL, this system offers an integrated platform to manage essential hospital functions including patient registration, doctor information, service tracking, appointments, and billing operations.

The system employs object-oriented architecture, with modular classes like Patient, Doctor, Appointment, Service, and Billing, each encapsulating relevant business logic and database interactions. The database operations are executed securely using MySQL connector with structured queries and validation checks to maintain data integrity.

Key features include:

- Auto-ID Generation for patients, doctors, and bills.
- Input Validation and error handling to ensure data consistency.
- Invoice Generation with detailed consultation and service charges.
- CSV Exporting for billing and appointment summaries.
- Command-Line Interface (CLI) with intuitive menu options for users.

This system enhances the efficiency of hospital operations by minimizing manual errors, improving data accessibility, and ensuring organized storage of records. It is suitable for small to medium-sized hospitals looking to digitize their core processes without a web-based overhead.

HOSPITAL PATIENT RECORD & BILLING SYSTEM

System overview

The Hospital Patient Record & Billing System is built to efficiently manage patient appointments and billing operations within a healthcare facility. It uses a MySQL database for reliable and persistent data storage and incorporates strong validation, reporting, and error-handling mechanisms to ensure accuracy and smooth system performance.

Architecture

- **Programming Language:** Python
- **Database:** MySQL
- **Modular Design:** The system is structured into distinct modules, separating appointment management and billing functionalities.
- **Database Connectivity:** All database interactions are handled through a shared get_connection() function located in the db_config.py module.

Dependencies

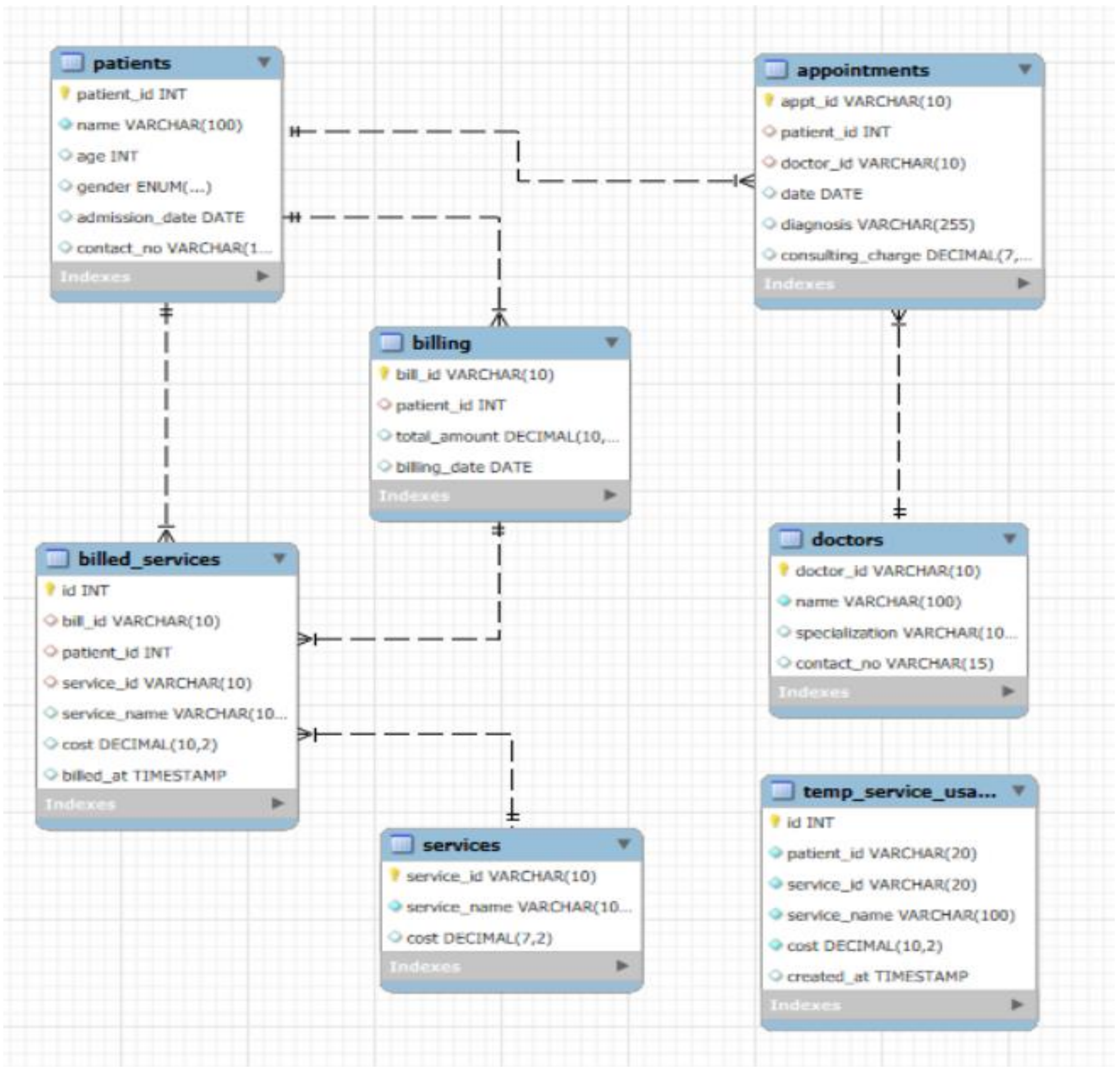
- Python 3.x
- Mysql Database
- Mysql-connector-python for database communication
- Python's built-in csv module for data export
- Note:

  Ensure the db_config.py file is correctly set up for database access.

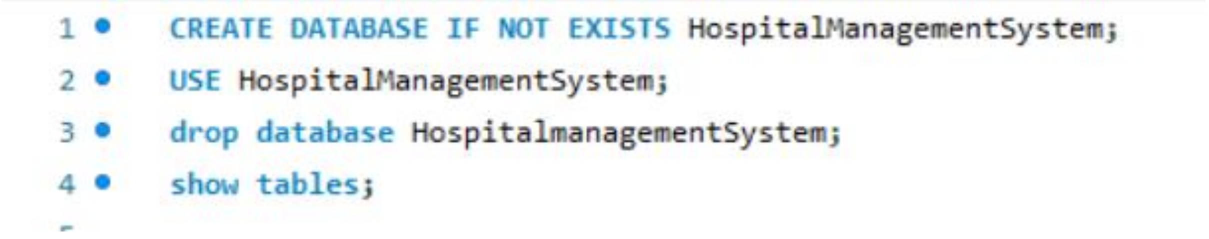
  The serviceusagedb class, which supports billing operations, should be implemented in a separate module named service.py.

DATABASE SCHEMA OVERVIEW

- The system uses the following key tables (not exhaustive):
- appointments – Stores details such as appointment ID, patient, doctor, date, diagnosis, and charges.
- patients – Contains master data for all patients.
- doctors – Contains master data for all doctors.
- billing – Stores bill records including bill ID, patient ID, total amount, and billing date.
- services – Lists all available hospital services.
- billed_services – Tracks services billed under each bill.
- temp_service_usage – Temporarily holds service usage data before the billing process is completed.



The Hospital Management System database schema created, which consists of multiple interrelated tables to manage patients, doctors, services, appointments, and billing in a hospital system.

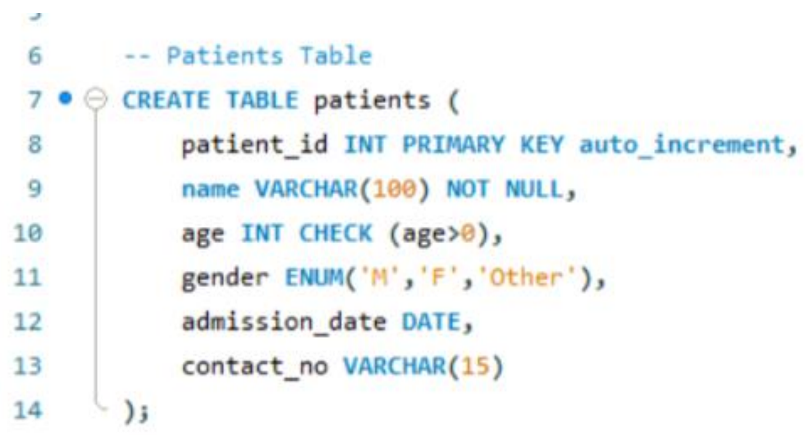


```sql
CREATE DATABASE IF NOT EXISTS HospitalManagementSystem;
USE HospitalManagementSystem;
drop database HospitalmanagementSystem;
show tables;
```

- Creates a database named HospitalManagementSystem if it doesn't already exist.
- USE sets the context to this database for subsequent operations.
- Deletes the entire database and all its tables.
- Should be used with caution (typically for reset/testing purposes).

1. patients Table

- Stores basic information about hospital patients.



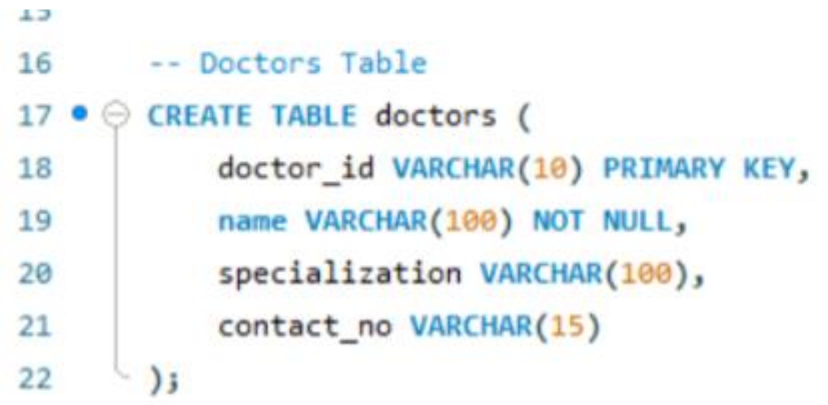
```sql
-- Patients Table
CREATE TABLE patients (
    patient_id INT PRIMARY KEY auto_increment,
    name VARCHAR(100) NOT NULL,
    age INT CHECK (age>0),
    gender ENUM('M','F','Other'),
    admission_date DATE,
    contact_no VARCHAR(15)
);
```

Fields:

- patient_id: Auto-incremented unique ID.
- name: Patient's full name (mandatory).
- age: Must be a positive number (using CHECK constraint).
- gender: Must be one of 'M', 'F', or 'Other'.
- admission_date: Date of admission.
- contact_no: Phone number (up to 15 characters).

2. doctors Table

Stores doctor information.



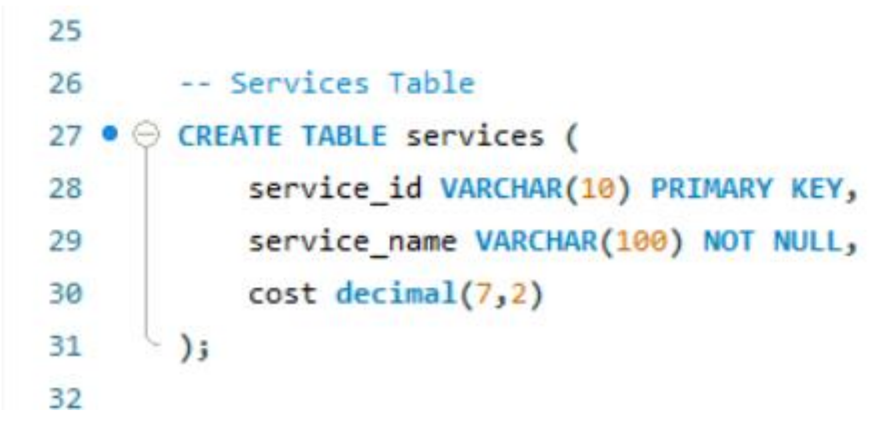
```sql
-- Doctors Table
CREATE TABLE doctors (
    doctor_id VARCHAR(10) PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    specialization VARCHAR(100),
    contact_no VARCHAR(15)
);
```

Fields:

- doctor_id: Unique doctor identifier.
- name: Doctor's name (mandatory).
- specialization: Area of expertise (e.g., Cardiology).
- contact_no: Phone number.

3. services Table

- Stores information about hospital services.



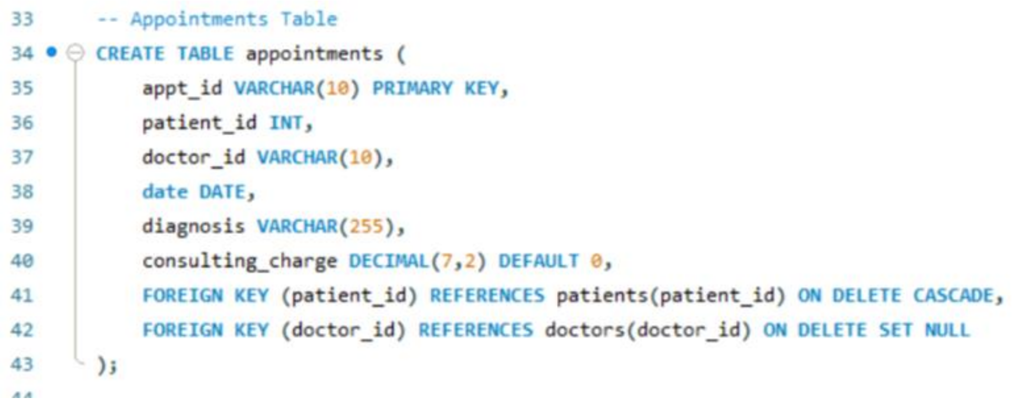
```sql
-- Services Table
CREATE TABLE services (
    service_id VARCHAR(10) PRIMARY KEY,
    service_name VARCHAR(100) NOT NULL,
    cost decimal(7,2)
);
```

Fields:

- service_id: Unique ID for each service.
- service_name: Describes the service (mandatory).
- cost: Service cost, allows up to 99999.99 (7 total digits, 2 after decimal).

4. appointments Table

Manages appointments between patients and doctors.



```sql
-- Appointments Table
CREATE TABLE appointments (
    appt_id VARCHAR(10) PRIMARY KEY,
    patient_id INT,
    doctor_id VARCHAR(10),
    date DATE,
    diagnosis VARCHAR(255),
    consulting_charge DECIMAL(7,2) DEFAULT 0,
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id) ON DELETE CASCADE,
    FOREIGN KEY (doctor_id) REFERENCES doctors(doctor_id) ON DELETE SET NULL
);
```

Fields:

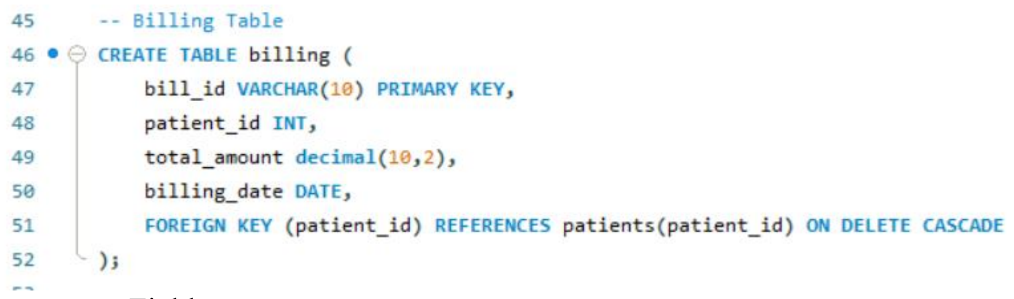
- appt_id: Unique appointment ID.
- patient_id: Foreign key referencing patients.
- doctor_id: Foreign key referencing doctors.
- date: Appointment date.
- diagnosis: Diagnosis from consultation.
- consulting_charge: Charge for the consultation (defaults to 0).

Constraints:

- ON DELETE CASCADE: Deletes appointments if the patient is deleted.
- ON DELETE SET NULL: If a doctor is removed, the field is set to NULL.

5. billing Table

- Stores billing details for a patient.



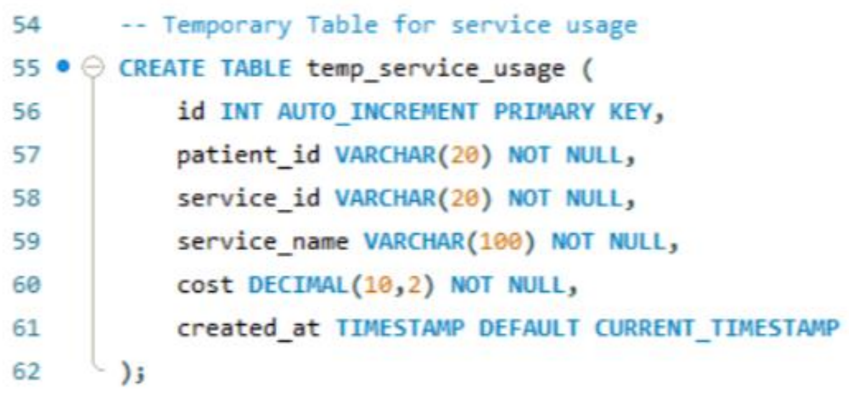
```sql
-- Billing Table
CREATE TABLE billing (
    bill_id VARCHAR(10) PRIMARY KEY,
    patient_id INT,
    total_amount decimal(10,2),
    billing_date DATE,
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id) ON DELETE CASCADE
);
```

Fields:

- bill_id: Unique ID for the bill.
- patient_id: Patient being billed.
- total_amount: Sum total of all charges.
- billing_date: Date the bill was generated.

6. temp_service_usage Table

- Stores services used by a patient temporarily before generating the final bill



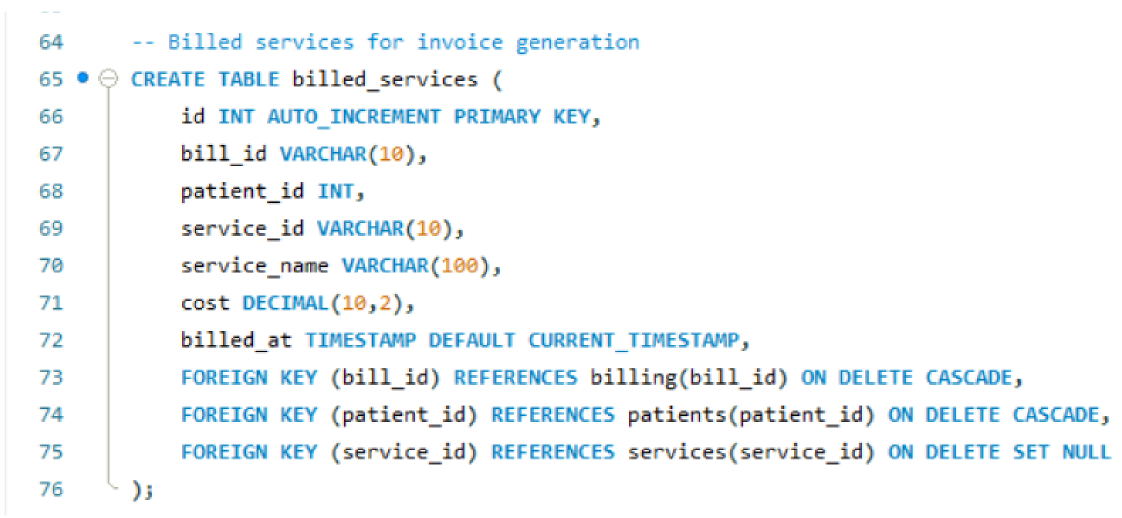
```sql
-- Temporary Table for service usage
CREATE TABLE temp_service_usage (
    id INT AUTO_INCREMENT PRIMARY KEY,
    patient_id VARCHAR(20) NOT NULL,
    service_id VARCHAR(20) NOT NULL,
    service_name VARCHAR(100) NOT NULL,
    cost DECIMAL(10,2) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Use Case:

- Temporary log of services a patient used (like a cart).
- Later moved to billed_services upon billing.

7. billed_services Table

- Final log of services billed to the patient.



```sql
-- Billed services for invoice generation
CREATE TABLE billed_services (
    id INT AUTO_INCREMENT PRIMARY KEY,
    bill_id VARCHAR(10),
    patient_id INT,
    service_id VARCHAR(10),
    service_name VARCHAR(100),
    cost DECIMAL(10,2),
    billed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (bill_id) REFERENCES billing(bill_id) ON DELETE CASCADE,
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id) ON DELETE CASCADE,
    FOREIGN KEY (service_id) REFERENCES services(service_id) ON DELETE SET NULL
);
```

Purpose:

- Tracks which services were billed in a specific invoice.
- Useful for generating printable bills or reports.

Constraints:

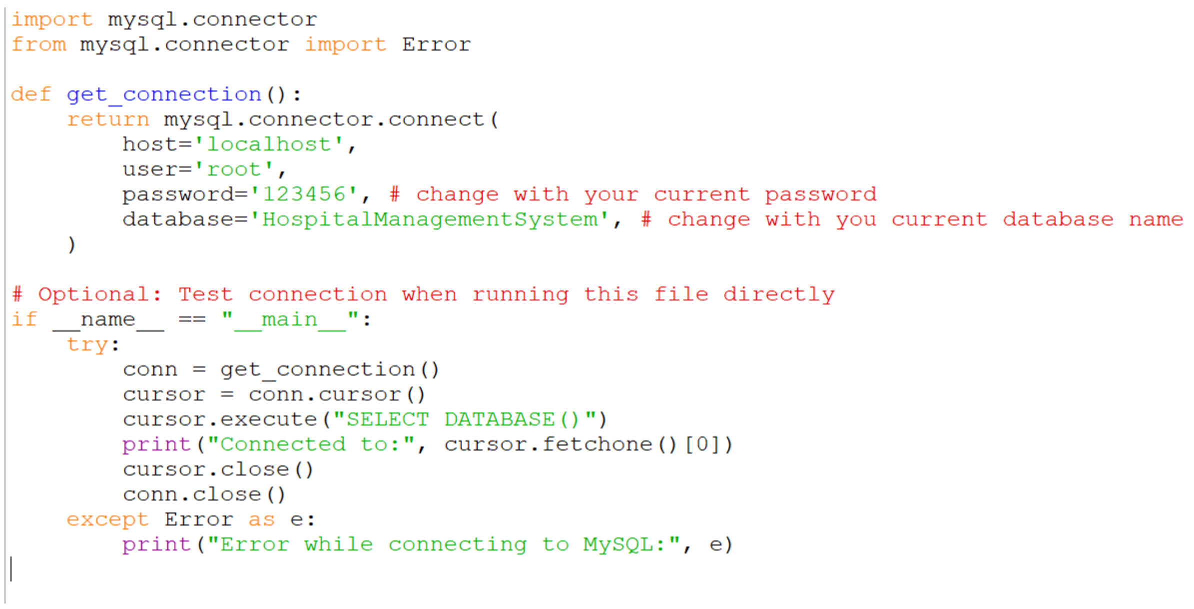
- Deletion of a patient or bill also removes the related billed services.
- If a service is removed, it will set service_id to NULL but retain billing data.

Table Relationships

- patients ↔ appointments: One-to-many.
- doctors ↔ appointments: One-to-many.
- patients ↔ billing: One-to-many.
- billing ↔ billed_services: One-to-many.
- services ↔ billed_services: Many-to-one.
- temp_service_usage: Temporary staging table not directly linked by foreign keys.

DB connection Module (db_config.py)

This script provides a reusable method for establishing a connection to a MySQL database. It also includes a basic test routine to verify the connection when the script is executed directly.



```python
import mysql.connector
from mysql.connector import Error

def get_connection():
    return mysql.connector.connect(
        host='localhost',
        user='root',
        password='123456', # change with your current password
        database='HospitalManagementSystem', # change with you current database name
    )

# Optional: Test connection when running this file directly
if __name__ == "__main__":
    try:
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT DATABASE()")
        print("Connected to:", cursor.fetchone()[0])
        cursor.close()
        conn.close()
    except Error as e:
        print("Error while connecting to MySQL:", e)
```

Code Breakdown & Explanation

- **import mysql.connector**: Imports the MySQL Connector module which enables Python to interact with MySQL databases using a Pythonic API.
- **from mysql.connector import Error**: Specifically imports the Error class to handle exceptions that occur during database operations.

Function: get_connection()

- This function returns a **MySQL database connection object** using mysql.connector.connect().
- It uses the following **connection parameters**:
    - host='localhost': Connects to the database server on the local machine.
    - user='root': Uses the MySQL root user (typically an administrative account).
    - password='123456': Password for the MySQL root user (should be secured in production environments).
    - database='HospitalManagementSystem': Connects to the specific database used for the project.

Purpose: Centralizes database connection logic to ensure **code reuse** and **maintainability** across different modules of the application

Test Connection Block

- **if __name__ == "__main__":**
    - Ensures the code block only runs when this file is executed directly, not when imported as a module.
- **conn = get_connection()**
    - Calls the get_connection() function to establish a connection to the database.
- **cursor = conn.cursor()**
    - Creates a cursor object used to execute SQL statements.
- **cursor.execute("SELECT DATABASE()")**
    - Executes an SQL query to retrieve the name of the currently connected database.
- **print("Connected to:", cursor.fetchone()[0])**
    - Fetches and prints the result of the query (i.e., database name).
- **cursor.close() and conn.close()**
    - Closes the cursor and connection to free up resources.
- **except Error as e:**
    - Catches and prints any database connection or query execution errors for debugging purposes.

Use Case in Project

This module is intended to be imported into other components of **Hospital Management System**, such as:

- patient.py
- billing.py
- doctor.py
- service.py
- appointment.py

Security Tip

- Never hardcode database credentials (like root and 123456) in production.
- Use environment variables or configuration files with proper access control.

Main CLI Controller (hospital_main.py)

This is the **main menu-driven controller** for the Hospital Management System. It provides a Command Line Interface (CLI) that lets users access various modules including patient records, doctor details, services, appointments, billing, and export functionality.

Imports

- **db_config**: Imports the get_connection function used internally in each module for database interaction.
- **Patient, Doctor, Service, Appointment, Bill**: These are custom classes/modules that encapsulate logic and menus for their respective data entities.
- **tabulate**: (Imported but unused here) A third-party library for displaying tabular data in a formatted CLI table (likely used in submodules).

export_menu() Function

This function displays a sub-menu for exporting reports in CSV format.

Options:

1. **Export Billing Summary:** Prompts the user for a filename and exports billing details using Bill.export_billing_summary_to_csv().
2. **Export Appointment Summary:** Exports appointment data to CSV using Appointment.export_appointment_summary_to_csv().
3. **Return to Main Menu:** Exits the export menu loop.

Looped Input Handling: Keeps running until the user chooses to return to the main menu.

Validation: Ensures proper file extension (.csv) and default naming when no input is provided.

main_menu() Function

The main interactive menu for navigating the system.

Option	Action	Method Called
1	Patient Records	Patient.patient_menu()
2	Doctor Records	Doctor.doctor_menu()
3	Service Records	Service.service_menu()
4	Appointment Records	Appointment.appointment_menu()
5	Billing Records	Bill.billing_menu()
6	Export Records	export_menu()
7	Exit System	Breaks the loop and exits

Input Validation:

- Only accepts valid numbered options.
- Displays a message for invalid entries.

User Flow:

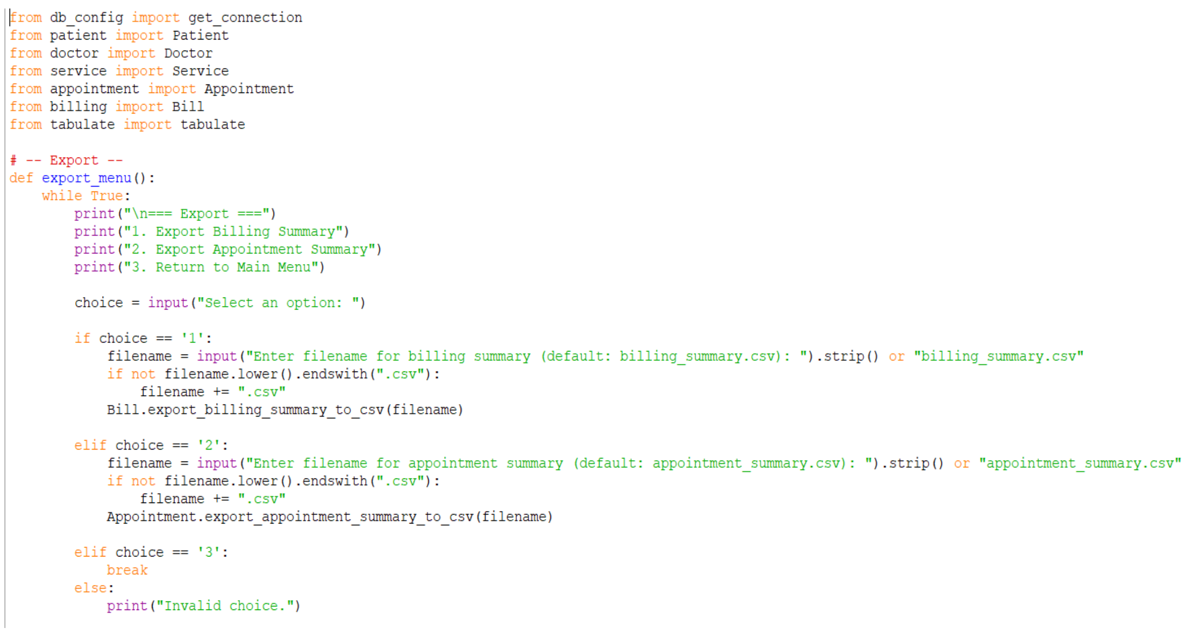
- Menu keeps displaying until user chooses to exit.

Entry Point

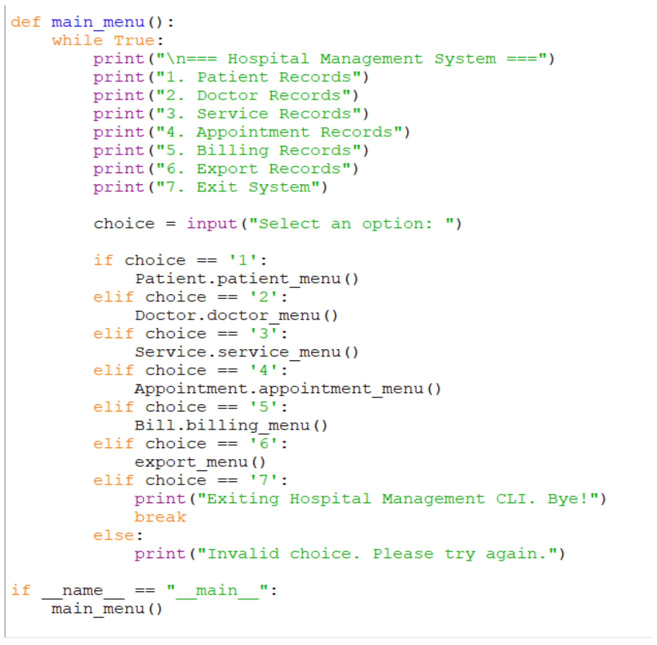
if __name__ == "__main__":

main_menu()

This block ensures the main menu is launched only when this script is run directly (not when imported).



```python
from db_config import get_connection
from patient import Patient
from doctor import Doctor
from service import Service
from appointment import Appointment
from billing import Bill
from tabulate import tabulate

# -- Export --
def export_menu():
    while True:
        print("\n=== Export ===")
        print("1. Export Billing Summary")
        print("2. Export Appointment Summary")
        print("3. Return to Main Menu")

        choice = input("Select an option: ")

        if choice == '1':
            filename = input("Enter filename for billing summary (default: billing_summary.csv): ").strip() or "billing_summary.csv"
            if not filename.lower().endswith(".csv"):
                filename += ".csv"
            Bill.export_billing_summary_to_csv(filename)

        elif choice == '2':
            filename = input("Enter filename for appointment summary (default: appointment_summary.csv): ").strip() or "appointment_summary.csv"
            if not filename.lower().endswith(".csv"):
                filename += ".csv"
            Appointment.export_appointment_summary_to_csv(filename)

        elif choice == '3':
            break
        else:
            print("Invalid choice.")
```



```python
def main_menu():
    while True:
        print("\n=== Hospital Management System ===")
        print("1. Patient Records")
        print("2. Doctor Records")
        print("3. Service Records")
        print("4. Appointment Records")
        print("5. Billing Records")
        print("6. Export Records")
        print("7. Exit System")

        choice = input("Select an option: ")

        if choice == '1':
            Patient.patient_menu()
        elif choice == '2':
            Doctor.doctor_menu()
        elif choice == '3':
            Service.service_menu()
        elif choice == '4':
            Appointment.appointment_menu()
        elif choice == '5':
            Bill.billing_menu()
        elif choice == '6':
            export_menu()
        elif choice == '7':
            print("Exiting Hospital Management CLI. Bye!")
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main_menu()
```

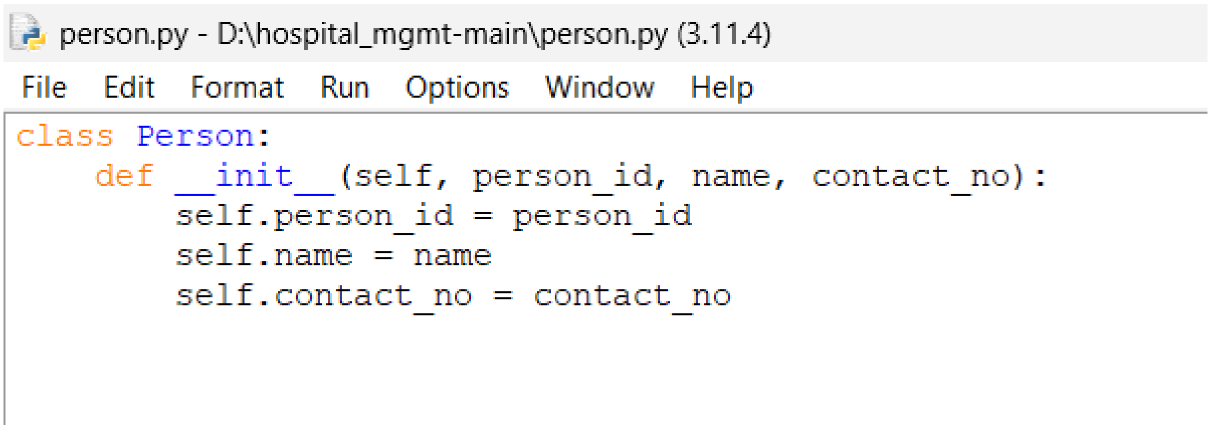
Person Module (person.py)

The Python class you've shared is a foundational OOP (Object-Oriented Programming) representation of a person.

Component	Explanation
class Person:	Declares a new class named Person.
__init__ method	Constructor that gets called when a new object is created.
person_id	Unique identifier for the person (could represent a patient or doctor ID).
name	Name of the person.
contact_no	Contact number of the person.
self	Refers to the current instance of the class.

This class could as a base class for specific types of people in your system, such as:

- Patient
- Doctor


person.py - D:\hospital_mgmt-main\person.py (3.11.4)
File Edit Format Run Options Window Help

```
class Person:
    def __init__(self, person_id, name, contact_no):
        self.person_id = person_id
        self.name = name
        self.contact_no = contact_no
```

Patient Module (patient.py)

This class manages patient records in a hospital management system and inherits from a Person class (assumed to provide name and contact_no attributes). Key features include:

Functionalities:

- Auto-generate patient ID
- CRUD operations: Add, View, Update, Delete
- Search by name
- Calculate admission duration
- Validate input (name, age, gender, contact number, and admission date)

Imports

```
from db_config import get_connection
from datetime import datetime, date
from person import Person
import re
import mysql.connector
from mysql.connector import IntegrityError, Error
from tabulate import tabulate
```

- **get_connection:** Custom function to connect to the MySQL database.
- **datetime, date:** For handling date comparisons.
- **Person:** A parent class providing shared attributes like name and contact_no.
- **re:** Used for regex validation.
- **mysql.connector:** For interacting with MySQL.
- **tabulate:** Formats output in tabular form for better readability.

Auto-Increment Logic

```
def auto_patient_id():
    from db_config import get_connection
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT MAX(patient_id) FROM patients")
    row = cursor.fetchone()
    cursor.close()
    conn.close()
    max_id = row[0]
    if max_id is None:
        return 1001
    return int(max_id) + 1
```

This function fetches the maximum patient_id from the database and increments it to generate a unique new ID. If no record exists, it starts at 1001.

Patient Class

Inherits from Person and initializes patient-specific details like age, gender, and admission_date.

```
class Patient(Person):
    def __init__(self, patient_id, name, age, gender, admission_date, contact_no):
        super().__init__(patient_id, name, contact_no)
        self.patient_id = patient_id
        self.age = age
        self.gender = gender
        self.admission_date = admission_date
```

Menu Navigation

```
def patient_menu():
    while True:
        print("\n=== Patient Records ===")
        print("1. Find Patient Details")
        print("2. Register New Patient")
        print("3. Display all Patients")
        print("4. Modify Patient Record")
        print("5. Delete Patient Record")
        print("6. Check Patient Admission Days")
        print("7. Return to Main Menu")

        choice = input("Choose an option: ")

        if choice == '1':
            name = input("Enter part or full patient name: ")
            Patient.search_by_name(name)

        elif choice == '2':
            patient_id = auto_patient_id()
            print(f"Patient ID: {patient_id}")
            name = input("Enter Name: ")
            age = input("Enter Age: ")
            gender = input("Enter Gender (M/F/Other): ")
            admission_date = input("Enter Admission Date (YYYY-MM-DD): ")
            contact_no = input("Enter Contact Number: ")
            patient = Patient(patient_id, name, age, gender, admission_date, contact_no)
            result = patient.add()
            print("Patient added successfully." if result else "Patient was not added.")

        elif choice == '3':
            Patient.view()

        elif choice == '4':
            patient_id = input("Enter Patient ID to update: ")
            existing = Patient.get_by_id(patient_id)
            if not existing:
                print("Patient not found.")
            else:
                print("Leave field blank to keep existing value.")
                name = input(f"Enter Name [{existing.name}]: ") or existing.name
                age = input(f"Enter Age [{existing.age}]: ") or existing.age
                gender = input(f"Enter Gender (M/F/Other) [{existing.gender}]: ") or existing.gender
                admission_date = input(f"Enter Admission Date (YYYY-MM-DD) [{existing.admission_date}]: ") or existing.admission_date
                contact_no = input(f"Enter Contact Number [{existing.contact_no}]: ") or existing.contact_no
                Patient(patient_id, name, age, gender, admission_date, contact_no).update()


        elif choice == '5':
            patient_id = input("Enter Patient ID to delete: ")
            Patient.delete(patient_id)

        elif choice == '6':
            patient_id = input("Enter Patient ID: ")
            Patient.days_admitted(patient_id)

        elif choice == '7':
            break

        else:
            print("Invalid Choice. Please try again.")
```

Displays a command-line interface to navigate through patient-related operations:

1. Search patient by name
2. Register new patient
3. View all patients
4. Modify patient data

5. Delete patient record
6. Calculate days admitted
7. Return to main menu

CRUD Methods

add()

Validates and inserts a new patient record.

Validation includes:

- Name must be alphabetic and contain only letters, periods, or spaces.
- Age must be between 0–120.
- Gender must be one of 'M', 'F', or 'Other'.
- Admission date must follow the YYYY-MM-DD format.
- Contact number must be numeric and at least 10 digits.

Handles integrity errors (e.g., duplicate patient_id) and commits valid data to the patients table.

```
# CRUD Operations

def add(self):
    if not self.name or not re.match(r'^[A-Za-z. ]+$', self.name):
        print("Invalid Name.")
        return False
    try:
        age = int(self.age)
        if age < 0 or age > 120:
            print("Invalid Age. Must be between 0 and 120.")
            return False
    except ValueError:
        print("Invalid Age. Must be a number.")
        return False
    if self.gender not in ['M', 'F', 'Other']:
        print("Invalid Gender. Choose from M, F, Other.")
        return False
    if not re.match(r'^\d{4}-\d{2}-\d{2}$', self.admission_date):
        print("Invalid Admission Date. Use YYYY-MM-DD format.")
        return False
    if not self.contact_no.isdigit() or len(self.contact_no) < 10:
        print("Invalid Contact Number. Must be at least 10 digits.")
        return False
    try:
        conn = get_connection()
        cursor = conn.cursor()
        sql = "INSERT INTO patients (patient_id, name, age, gender, admission_date, contact_no) VALUES (%s, %s, %s, %s, %s, %s)"
        cursor.execute(sql, (self.patient_id, self.name, age, self.gender, self.admission_date, self.contact_no))
        conn.commit()
        return True
    except mysql.connector.errors.IntegrityError as e:
        if "PRIMARY" in str(e):
            print(f"Error: Duplicate Patient ID '{self.patient_id}'. Please use a unique ID.")
        else:
            print("Database integrity error: ", e)
        return False
    except Exception as e:
        print("Error while adding patient:", e)
        return False
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

update()

Allows updating an existing record. Validation is similar to the add() method. Converts date objects into string if necessary and performs update query. Confirms success based on affected rows.

```
def update(self):
    if not self.name or not re.match(r'^[A-Za-z. ]+$', self.name):
        print("Invalid Name.")
        return False
    try:
        age = int(self.age)
        if age < 0 or age > 120:
            print("Invalid Age. Must be between 0 and 120.")
            return False
    except ValueError:
        print("Invalid Age.")
        return False
    if self.gender not in ['M', 'F', 'Other']:
        print("Invalid Gender. Choose from M, F, Other.")
        return False
    try:
        if isinstance(self.admission_date, date):
            self.admission_date = self.admission_date.strftime("%Y-%m-%d")
        else:
            datetime.strptime(self.admission_date, "%Y-%m-%d")
    except ValueError:
        print("Invalid Admission Date. Use YYYY-MM-DD format.")
        return False
    if not self.contact_no.isdigit() or len(self.contact_no) < 10:
        print("Invalid Contact Number. Must be at least 10 digits.")
        return False
    try:
        conn = get_connection()
        cursor = conn.cursor()
        sql = "UPDATE patients SET name=%s, age=%s, gender=%s, admission_date=%s, contact_no=%s WHERE patient_id=%s"
        cursor.execute(sql, (self.name, age, self.gender, self.admission_date, self.contact_no, self.patient_id))
        conn.commit()
        if cursor.rowcount == 0:
            print(f"No updates found in ID '{self.patient_id}'.")
            return False
        else:
            print("Sucessfully updated the patient details.")
            return True
    except mysql.connector.errors.IntegrityError as e:
        print("Integrity error in database: ", e)
        return False
    except Exception as e:
        print("Error while updating patient:", e)
        return False
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

get_by_id(patient_id)

Fetches a patient record based on patient_id. Returns a Patient object if found, else returns None.

```
@staticmethod
def get_by_id(patient_id):
    try:
        conn = get_connection()
        cursor = conn.cursor(dictionary=True)
        cursor.execute("SELECT * FROM patients WHERE patient_id = %s", (patient_id,))
        row = cursor.fetchone()
        if row:
            return Patient(**row)
        else:
            return None
    except Exception as e:
        print("Error retrieving patient record:", e)
        return None
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

delete(patient_id)

Deletes the patient record matching the patient_id. Outputs confirmation based on deletion success.

```
@staticmethod
def delete(patient_id):
    try:
        conn = get_connection()
        cursor = conn.cursor()
        sql = """DELETE FROM patients WHERE patient_id=%s"""
        cursor.execute(sql, (patient_id,))
        conn.commit()
        if cursor.rowcount == 0:
            print(f"No patient found with ID '{patient_id}'.")
            return False
        else:
            print("Successfully deleted the patient")
            return True
    except Error as e:
        print("Error while deleting patient:", e)
        return False
    except Exception as e:
        print("Error while deleting patient:", e)
        return False
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

view()

Displays all patient records in a clean table format using the tabulate library.

```
@staticmethod
def view():
    try:
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM patients")
        rows = cursor.fetchall()
        headers = ["Patient ID", "Name", "Age", "Gender", "Admission Date", "Contact Number"]
        print(tabulate(rows, headers=headers, tablefmt="grid"))
    except Error as e:
        print("Error while viewing patients:", e)
    except Exception as e:
        print("Error while viewing patients:", e)
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

days_admitted(patient_id)

Calculates how many days a patient has been admitted by subtracting the admission_date from the current date.

```
@staticmethod
def days_admitted(patient_id):
    conn = get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("SELECT admission_date FROM patients WHERE patient_id=%s", (patient_id,))
        row = cursor.fetchone()
        if row and row[0]:
            admission_date = row[0]
            today = date.today()
            days = (today - admission_date).days
            print(f"Patient {patient_id} has been admitted for {days} days.")
            return days
        else:
            print("Patient not found or admission date missing.")
            return None
    except Exception as e:
        print("Error calculating days admitted:", e)
        return None
    finally:
        cursor.close()
        conn.close()
```

search_by_name(name_substring)

Searches for patients whose name contains the given substring (case-insensitive). Useful for partial name lookups.

```
@staticmethod
def search_by_name(name_substring):
    conn = get_connection()
    cursor = conn.cursor()
    try:
        search_pattern = f"%{name_substring}%"
        cursor.execute("SELECT * FROM patients WHERE name LIKE %s", (search_pattern,))
        rows = cursor.fetchall()
        if rows:
            headers = ["Patient ID", "Name", "Age", "Gender", "Admission Date", "Contact Number"]
            print(tabulate(rows, headers=headers, tablefmt="grid"))
        else:
            print("No patients found")
    except Exception as e:
        print("Error searching patients:", e)
    finally:
        cursor.close()
        conn.close()
```

Error Handling

- Catches exceptions like IntegrityError, invalid input formats, and connection errors.
- Ensures that database connections and cursors are closed after every operation using finally.

Doctor Module (doctor.py)

This module provides functionality to manage doctor records in the Hospital Management System. It includes creating, updating, viewing, deleting, and searching doctor information using a menu-driven CLI.

Imports

```
import re
from db_config import get_connection
from person import Person
import mysql.connector
from mysql.connector import IntegrityError, Error
from tabulate import tabulate
```

- **get_connection**: Handles MySQL database connections.
- **Person class**: Base class inherited by Doctor.
- **re**: For validating name and specialization formats.
- **tabulate**: Pretty-prints results in a table format on the CLI.

Function: auto_doctor_id()

Generates a new unique doctor ID in the format D01, D02, etc.

Logic:

- Fetches all doctor IDs starting with D.
- Extracts the numeric part.
- Returns the next available ID formatted with zero padding.

```
def auto_doctor_id():
    from db_config import get_connection
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT doctor_id FROM doctors WHERE doctor_id LIKE 'D%'")
    ids = [int(row[0][1:]) for row in cursor.fetchall() if row[0][1:].isdigit()]
    cursor.close()
    conn.close()
    next_num = max(ids) + 1 if ids else 1
    return f"D{next_num:02d}"
```

Class: Doctor

- Inherits doctor_id, name, and contact_no from Person.
- Applies formatting to the name to ensure it begins with "Dr.".
- Stores specialization.

```
class Doctor(Person):
    def __init__(self, doctor_id, name, specialization, contact_no):
        super().__init__(doctor_id, name, contact_no)
        self.doctor_id = doctor_id
        self.specialization = specialization
        self.name = self._format_name(name)
```

Private Method: _format_name(name)

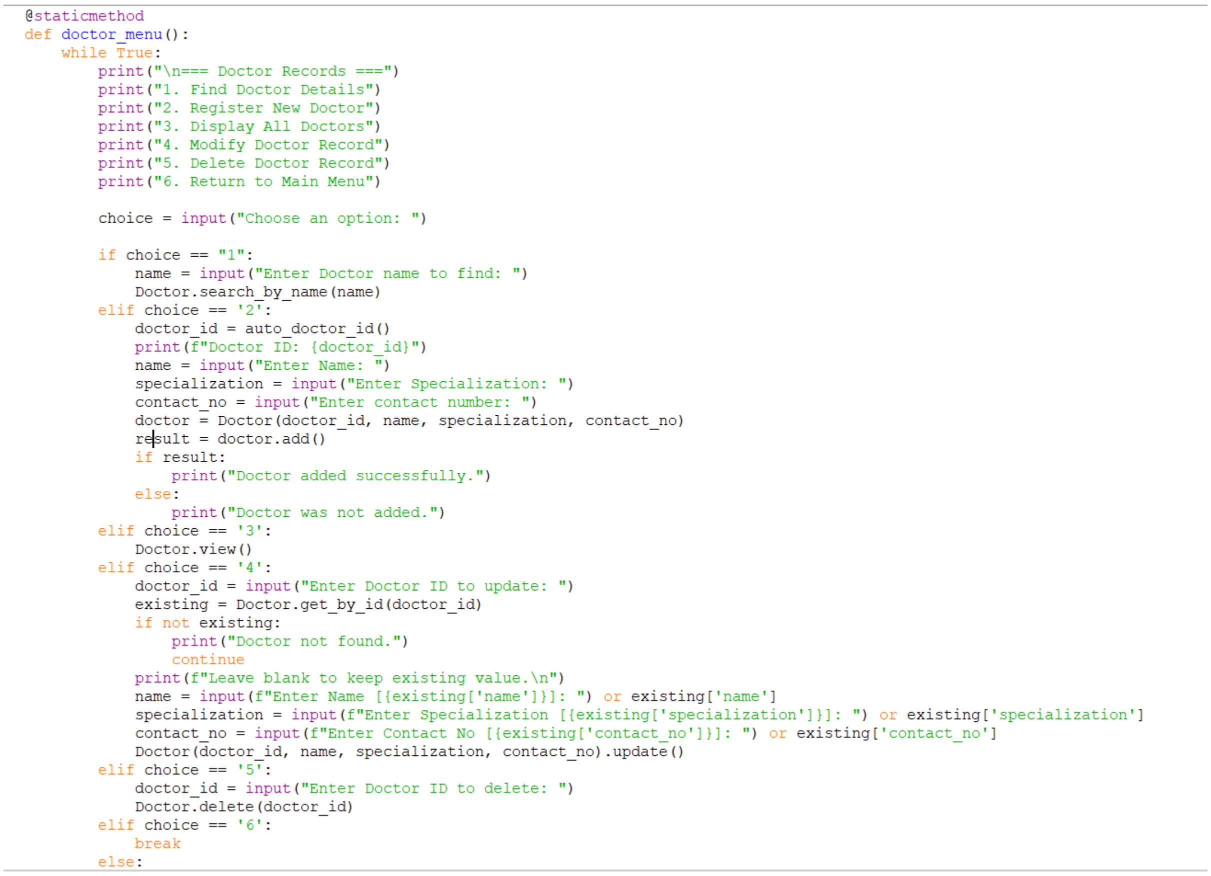
Ensures all doctor names are consistently formatted with "Dr." as a prefix.

```
    def _format_name(self, name):
        name = name.strip()
        if not name.lower().startswith("dr."):
            return "Dr. " + name
        return name
```

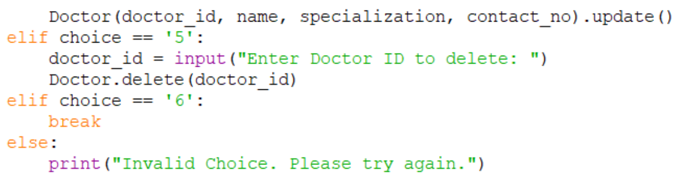
CLI Interface: doctor_menu()

Provides a user-driven interface to interact with the doctor module.

Option	Action
1	Search for doctor by name
2	Add/register a new doctor
3	View all doctors
4	Modify doctor details
5	Delete a doctor by ID
6	Return to main menu



```
@staticmethod
def doctor_menu():
    while True:
        print("\n=== Doctor Records ===")
        print("1. Find Doctor Details")
        print("2. Register New Doctor")
        print("3. Display All Doctors")
        print("4. Modify Doctor Record")
        print("5. Delete Doctor Record")
        print("6. Return to Main Menu")

        choice = input("Choose an option: ")

        if choice == "1":
            name = input("Enter Doctor name to find: ")
            Doctor.search_by_name(name)
        elif choice == '2':
            doctor_id = auto_doctor_id()
            print(f"Doctor ID: {doctor_id}")
            name = input("Enter Name: ")
            specialization = input("Enter Specialization: ")
            contact_no = input("Enter contact number: ")
            doctor = Doctor(doctor_id, name, specialization, contact_no)
            result = doctor.add()
            if result:
                print("Doctor added successfully.")
            else:
                print("Doctor was not added.")
        elif choice == '3':
            Doctor.view()
        elif choice == '4':
            doctor_id = input("Enter Doctor ID to update: ")
            existing = Doctor.get_by_id(doctor_id)
            if not existing:
                print("Doctor not found.")
                continue
            print(f"Leave blank to keep existing value.\n")
            name = input(f"Enter Name [{existing['name']}]: ") or existing['name']
            specialization = input(f"Enter Specialization [{existing['specialization']}]: ") or existing['specialization']
            contact_no = input(f"Enter Contact No [{existing['contact_no']}]: ") or existing['contact_no']
            Doctor(doctor_id, name, specialization, contact_no).update()
        elif choice == '5':
            doctor_id = input("Enter Doctor ID to delete: ")
            Doctor.delete(doctor_id)
        elif choice == '6':
            break
        else:
```



```
    Doctor(doctor_id, name, specialization, contact_no).update()
elif choice == '5':
    doctor_id = input("Enter Doctor ID to delete: ")
    Doctor.delete(doctor_id)
elif choice == '6':
    break
else:
    print("Invalid Choice. Please try again.")
```

add(self)

Inserts a new doctor record into the database after validation.

Validations:

- **Name**: Alphabet, dot, or space only.
- **Specialization**: Letters and spaces only.
- **Contact number**: Must be digits and at least 10 characters.
- Checks for unique contact number in the database.

```python
def add(self):
    if not self.name or not re.match(r'^[A-Za-z. ]+$', self.name):
        print("Invalid Name.")
        return False
    if not self.specialization or not all(x.isalpha() or x.isspace() for x in self.specialization):
        print("Invalid Specialization.")
        return False
    if not self.contact_no.isdigit() or len(self.contact_no) < 10:
        print("Invalid Contact Number.")
        return False

    try:
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT 1 FROM doctors WHERE contact_no = %s", (self.contact_no,))
        if cursor.fetchone():
            print(f"Contact number '{self.contact_no}' is already in use. Please provide a unique number.")
            return False

        sql = "INSERT INTO doctors (doctor_id, name, specialization, contact_no) VALUES (%s, %s, %s, %s)"
        cursor.execute(sql, (self.doctor_id, self.name, self.specialization, self.contact_no))
        conn.commit()
        return True
    except mysql.connector.errors.IntegrityError as e:
        if "unique_contact_no" in str(e):
            print(f"Error: Contact number '{self.contact_no}' already exists. Please use a unique contact number.")
        elif "PRIMARY" in str(e):
            print(f"Error: Duplicate Doctor ID '{self.doctor_id}'. Please use a unique ID.")
        else:
            print("Database integrity error: ", e)
        return False

    except Exception as e:
        print("Error while adding doctor:", e)
        return False
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

update(self)

Updates an existing doctor record based on doctor_id.

- Performs similar validations as add.
- Uses an UPDATE query to modify doctor data.
- Confirms update success via rowcount.

```
def update(self):
    if not self.name or not re.match(r'^[A-Za-z. ]+$', self.name):
        print("Invalid Name.")
        return False
    if not self.specialization or not all(x.isalpha() or x.isspace() for x in self.specialization):
        print("Invalid Specialization.")
        return False
    if not self.contact_no.isdigit() or len(self.contact_no) < 10:
        print("Invalid Contact Number.")
        return False

    try:
        conn = get_connection()
        cursor = conn.cursor()
        sql = "UPDATE doctors SET name=%s, specialization=%s, contact_no=%s WHERE doctor_id=%s"
        cursor.execute(sql, (self.name, self.specialization, self.contact_no, self.doctor_id))
        conn.commit()
        if cursor.rowcount == 0:
            print(f"No updates found.")
            return False
        else:
            print("Doctor updated successfully.")
            return True
    except Error as e:
        print("Database error while updating doctor:", e)
        return False
    except Exception as e:
        print("Error while updating doctor:", e)
        return False
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

get_by_id(doctor_id)

Fetches a doctor record using the given doctor ID.

- Returns a dictionary of the record or None if not found.

```
@staticmethod
def get_by_id(doctor_id):
    try:
        conn = get_connection()
        cursor = conn.cursor(dictionary=True)
        cursor.execute("SELECT * FROM doctors WHERE doctor_id = %s", (doctor_id,))
        record = cursor.fetchone()
        return record
    except Exception as e:
        print("Error fetching doctor:", e)
        return None
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

delete(doctor_id)

Deletes a doctor from the database using the doctor_id.

- Confirms deletion based on affected rows.

```
@staticmethod
def delete(doctor_id):
    try:
        conn = get_connection()
        cursor = conn.cursor()
        sql = "DELETE FROM doctors WHERE doctor_id=%s"
        cursor.execute(sql, (doctor_id,))
        conn.commit()
        if cursor.rowcount == 0:
            print(f"Doctor ID '{doctor_id}' not found.")
            return False
        else:
            print("Doctor deleted successfully.")
            return True
    except Error as e:
        print("Database error while deleting doctor:", e)
        return False
    except Exception as e:
        print("Error while deleting doctor:", e)
        return False
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

view()

Displays all doctor records in a tabular format using tabulate.

- Columns: Doctor ID, Name, Specialization, Contact Number.

```
@staticmethod
def view():
    try:
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM doctors")
        rows = cursor.fetchall()
        headers = ["Doctor ID", "Name", "Specialization", "Contact Number"]
        print(tabulate(rows, headers=headers, tablefmt="grid"))

    except Error as e:
        print("Database error while viewing doctors:", e)
    except Exception as e:
        print("Error while viewing doctors:", e)
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

search_by_name(name_substring)

Searches for doctors whose names match or contain the input substring.

- Uses SQL LIKE pattern match.
- Displays results in tabular format.

```
@staticmethod
def search_by_name(name_substring):
    conn = get_connection()
    cursor = conn.cursor()
    try:
        search_pattern = f"%{name_substring}%"
        cursor.execute("SELECT * FROM doctors WHERE name LIKE %s", (search_pattern,))
        rows = cursor.fetchall()
        if rows:
            headers = ["Doctor ID", "Name", "Specialization", "Contact Number"]
            print(tabulate(rows, headers=headers, tablefmt="grid"))
        else:
            print("No doctors found matching that name.")
    except Exception as e:
        print("Error searching doctors:", e)
    finally:
        cursor.close()
        conn.close()
```

Service Module (service.py)

The code is a Python module designed to manage services in a Hospital Management System using a MySQL database. It handles:

- Registering and managing services (like tests, procedures).
- Assigning those services to patients.
- Viewing, updating, and deleting service records.
- Temporary service usage tracking for billing or treatment history.

Imports

```
import re
from db_config import get_connection
import mysql.connector
from mysql.connector import IntegrityError, Error
from tabulate import tabulate
```

- re: Used for regex-based input validation.
- get_connection: Imports a function from db_config to connect to the MySQL database.
- mysql.connector: MySQL database driver.
- IntegrityError, Error: Specific exceptions for handling database errors.
- tabulate: For formatting outputs in table format.

Method auto_service_id() Function

```
def auto_service_id():
    from db_config import get_connection
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT service_id FROM services WHERE service_id LIKE 'S%'")
    ids = [int(row[0][1:]) for row in cursor.fetchall() if row[0][1:].isdigit()]
    cursor.close()
    conn.close()
    next_num = max(ids) + 1 if ids else 1
    return f"S{next_num:02d}"
```

Auto-generates a new service ID (like S01, S02, …).

- Fetch all service_ids starting with S.
- Strip the S, convert the numeric part to an integer.
- Find the max number and add 1.
- Return the new ID as SXX.

Service Class

Manages all operations related to hospital services.

```
class Service:
    def __init__(self, service_id, service_name, cost):
        self.service_id = service_id
        self.service_name = service_name
        self.cost = cost
```

service_menu() – CLI Menu

Allows the user to:

1. Register a new service.
2. View all services.
3. Modify existing services.
4. Delete a service.
5. Exit menu.

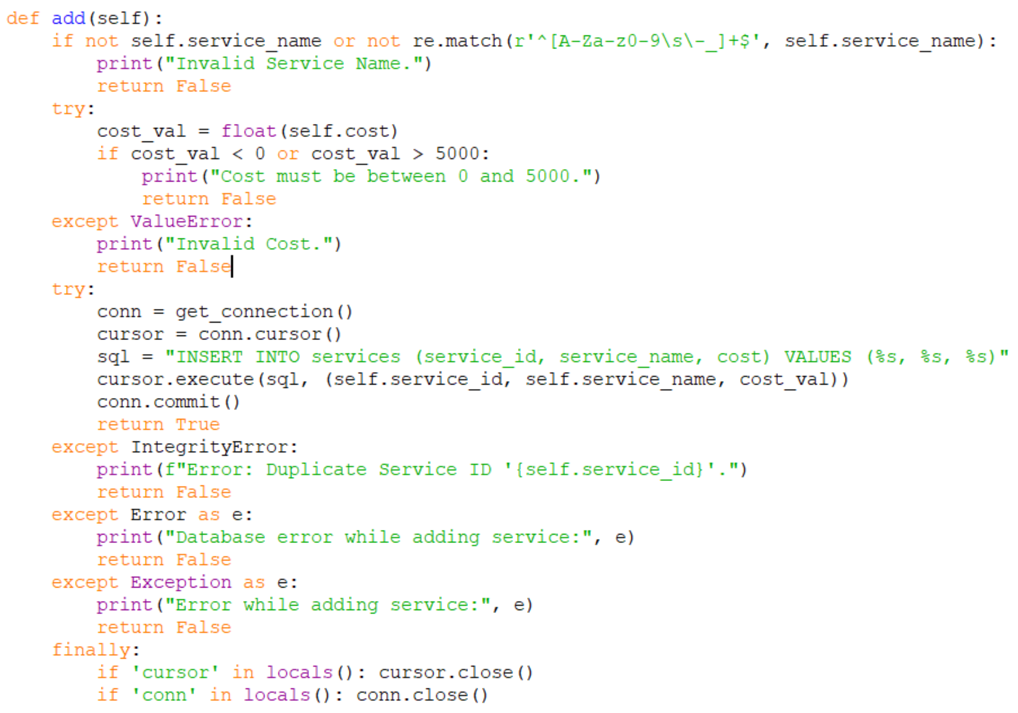
This is a user interface loop using standard input and output.

```python
@staticmethod
def service_menu():
    while True:
        print("\n=== Service Records ===")
        print("1. Register New Service")
        print("2. Display All Services")
        print("3. Modify Service")
        print("4. Remove Service")
        print("5. Return to Main Menu")
        choice = input("Select an option: ")
        if choice == '1':
            service_id = auto_service_id()
            print(f"Auto-generated Service ID: {service_id}")
            service_name = input("Enter Service Name: ")
            cost = input("Enter Cost: ")
            service = Service(service_id, service_name, cost)
            result = service.add()
            if result:
                print("Service added successfully.")
            else:
                print("Service was not added.")
        elif choice == '2':
            Service.view()
        elif choice == '3':
            service_id = input("Enter Service ID to update: ")
            existing = Service.get_by_id(service_id)
            if not existing:
                print("Service not found.")
                continue
            print("Leave input blank to keep current value.")
            name = input(f"Enter Service Name [{existing['service_name']}]: ") or existing['service_name']
            cost_input = input(f"Enter Cost [{existing['cost']}]: ")
            try:
                cost = float(cost_input) if cost_input.strip() else float(existing['cost'])
            except ValueError:
                print("Invalid cost input. Please try again.")
                continue
            Service(service_id, name, cost).update()
        elif choice == '4':
            service_id = input("Enter Service ID to delete: ")
            Service.delete(service_id)
        elif choice == '5':
            break
        else:
            print("Invalid Choice. Please try again.")
```

add(self)

Adds a new service to the database.

- Validates:
  - service_name must match allowed characters (alphanumeric, hyphen, underscore).
  - cost must be numeric and within 0–5000.
- If validation passes:
  - Inserts record into services table.
- Handles exceptions:
  - Duplicate IDs, invalid cost, database errors.

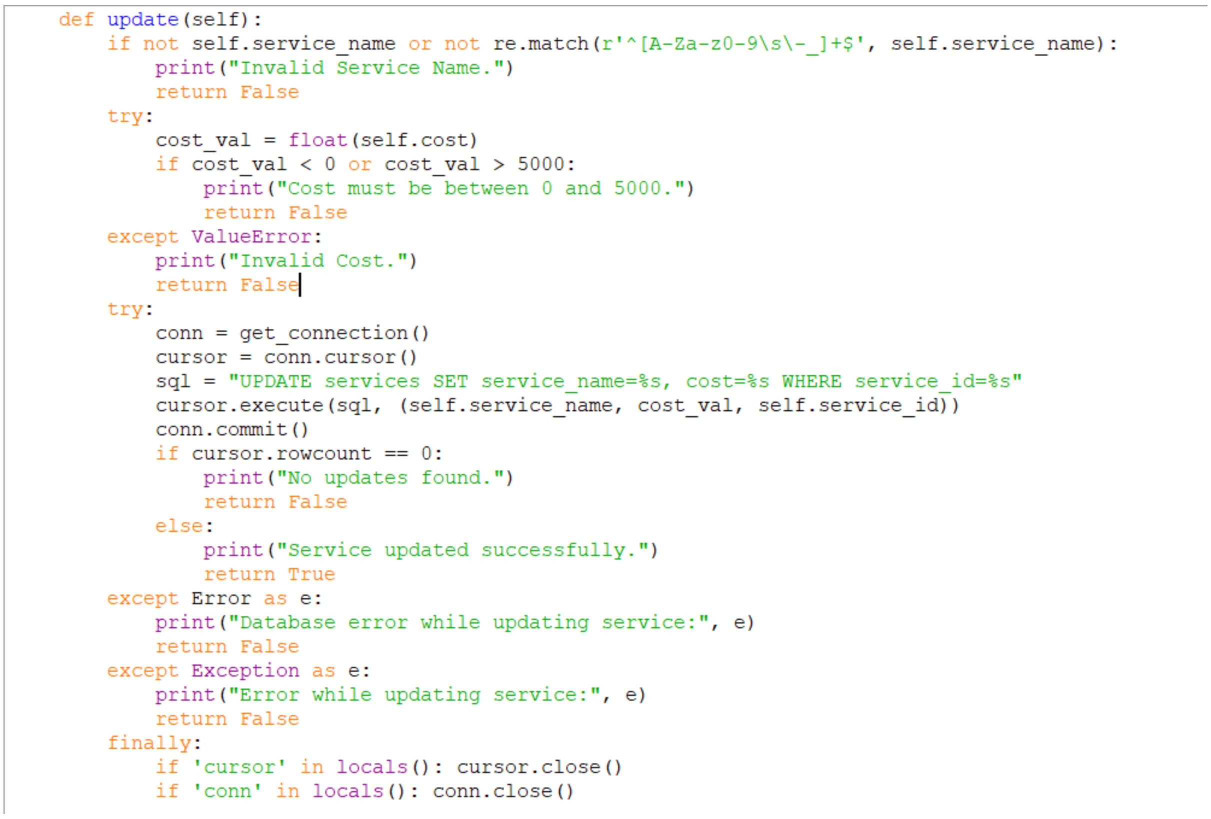


```
def add(self):
    if not self.service_name or not re.match(r'^[A-Za-z0-9\s\-_]+$', self.service_name):
        print("Invalid Service Name.")
        return False
    try:
        cost_val = float(self.cost)
        if cost_val < 0 or cost_val > 5000:
            print("Cost must be between 0 and 5000.")
            return False
    except ValueError:
        print("Invalid Cost.")
        return False
    try:
        conn = get_connection()
        cursor = conn.cursor()
        sql = "INSERT INTO services (service_id, service_name, cost) VALUES (%s, %s, %s)"
        cursor.execute(sql, (self.service_id, self.service_name, cost_val))
        conn.commit()
        return True
    except IntegrityError:
        print(f"Error: Duplicate Service ID '{self.service_id}'.")
        return False
    except Error as e:
        print("Database error while adding service:", e)
        return False
    except Exception as e:
        print("Error while adding service:", e)
        return False
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

update(self)

Updates an existing service.

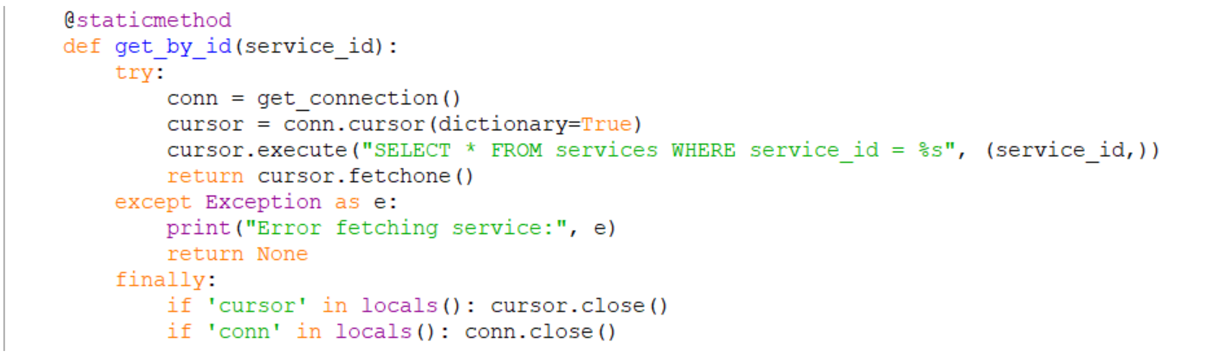
- Validates updated name and cost.
- Updates the record where service_id matches.
- Displays success/failure.



```
def update(self):
    if not self.service_name or not re.match(r'^[A-Za-z0-9\s\-_]+$', self.service_name):
        print("Invalid Service Name.")
        return False
    try:
        cost_val = float(self.cost)
        if cost_val < 0 or cost_val > 5000:
            print("Cost must be between 0 and 5000.")
            return False
    except ValueError:
        print("Invalid Cost.")
        return False
    try:
        conn = get_connection()
        cursor = conn.cursor()
        sql = "UPDATE services SET service_name=%s, cost=%s WHERE service_id=%s"
        cursor.execute(sql, (self.service_name, cost_val, self.service_id))
        conn.commit()
        if cursor.rowcount == 0:
            print("No updates found.")
            return False
        else:
            print("Service updated successfully.")
            return True
    except Error as e:
        print("Database error while updating service:", e)
        return False
    except Exception as e:
        print("Error while updating service:", e)
        return False
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

get_by_id(service_id)

- Fetches a service row by service_id as a dictionary using cursor(dictionary=True).
- Used during modification to retrieve current values.



```
@staticmethod
def get_by_id(service_id):
    try:
        conn = get_connection()
        cursor = conn.cursor(dictionary=True)
        cursor.execute("SELECT * FROM services WHERE service_id = %s", (service_id,))
        return cursor.fetchone()
    except Exception as e:
        print("Error fetching service:", e)
        return None
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

delete(service_id)

- Deletes a service from the services table by ID.
- Checks cursor.rowcount to confirm if deletion occurred.

```
@staticmethod
def delete(service_id):
    try:
        conn = get_connection()
        cursor = conn.cursor()
        sql = "DELETE FROM services WHERE service_id=%s"
        cursor.execute(sql, (service_id,))
        conn.commit()
        if cursor.rowcount == 0:
            print("Service ID not found.")
            return False
        else:
            print("Service deleted successfully.")
            return True
    except Error as e:
        print("Database error while deleting service:", e)
        return False
    except Exception as e:
        print("Error while deleting service:", e)
        return False
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

view()

- Displays all services using tabulate() for readability.

```
@staticmethod
def view():
    try:
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM services")
        rows = cursor.fetchall()
        headers = ["Service ID", "Service Name", "Cost"]
        print(tabulate(rows, headers=headers, tablefmt="grid"))
    except Error as e:
        print("Database error while viewing services:", e)
    except Exception as e:
        print("Error while viewing services:", e)
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

ServiceUsageDB Class

Handles tracking of which services have been used by which patients. These are **temporary records**, probably used before billing.

Method add_service_for_patient(patient_id, service)

Function: Adds a service for a given patient.

- Validates patient ID and service fields.
- Inserts record into temp_service_usage.

Used during patient treatment for service logging.

```
class ServiceUsageDB:
    @staticmethod
    def add_service_for_patient(patient_id, service):
        if not re.match(r'^[A-Za-z0-9]+$', patient_id):
            print("Invalid Patient ID.")
            return
        if not re.match(r'^[A-Za-z0-9]+$', service.service_id):
            print("Invalid Service ID.")
            return
        if not re.match(r'^[A-Za-z0-9\s\-_]+$', service.service_name):
            print("Invalid Service Name.")
            return
        try:
            cost = float(service.cost)
            if cost < 0 or cost > 5000:
                print("Invalid Cost.")
                return
        except ValueError:
            print("Invalid Cost.")
            return

        try:
            conn = get_connection()
            cursor = conn.cursor()
            sql = "INSERT INTO temp_service_usage (patient_id, service_id, service_name, cost) VALUES (%s, %s, %s, %s)"
            cursor.execute(sql, (patient_id, service.service_id, service.service_name, cost))
            conn.commit()
            print(f"Added {service.service_name} (ID: {service.service_id}, Cost: {cost}) for patient {patient_id}")
        except IntegrityError:
            print(f"Error: Duplicate service usage entry for patient {patient_id} and service {service.service_id}.")
        except Error as e:
            print("Database error while adding service usage:", e)
        except Exception as e:
            print("Error while adding service usage:", e)
        finally:
            if 'cursor' in locals(): cursor.close()
            if 'conn' in locals(): conn.close()
```

get_services_for_patient(patient_id)

Returns a list of all services used by a patient from temp_service_usage.

Useful for:

- Viewing patient billing summary.
- Checking services used during treatment.

```python
@staticmethod
def get_services_for_patient(patient_id):
    try:
        conn = get_connection()
        cursor = conn.cursor()
        sql = "SELECT service_id, service_name, cost FROM temp_service_usage WHERE patient_id=%s"
        cursor.execute(sql, (patient_id,))
        rows = cursor.fetchall()
        return rows
    except Error as e:
        print("Database error while fetching services:", e)
        return []
    except Exception as e:
        print("Error while fetching services:", e)
        return []
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

clear_services_for_patient(patient_id)

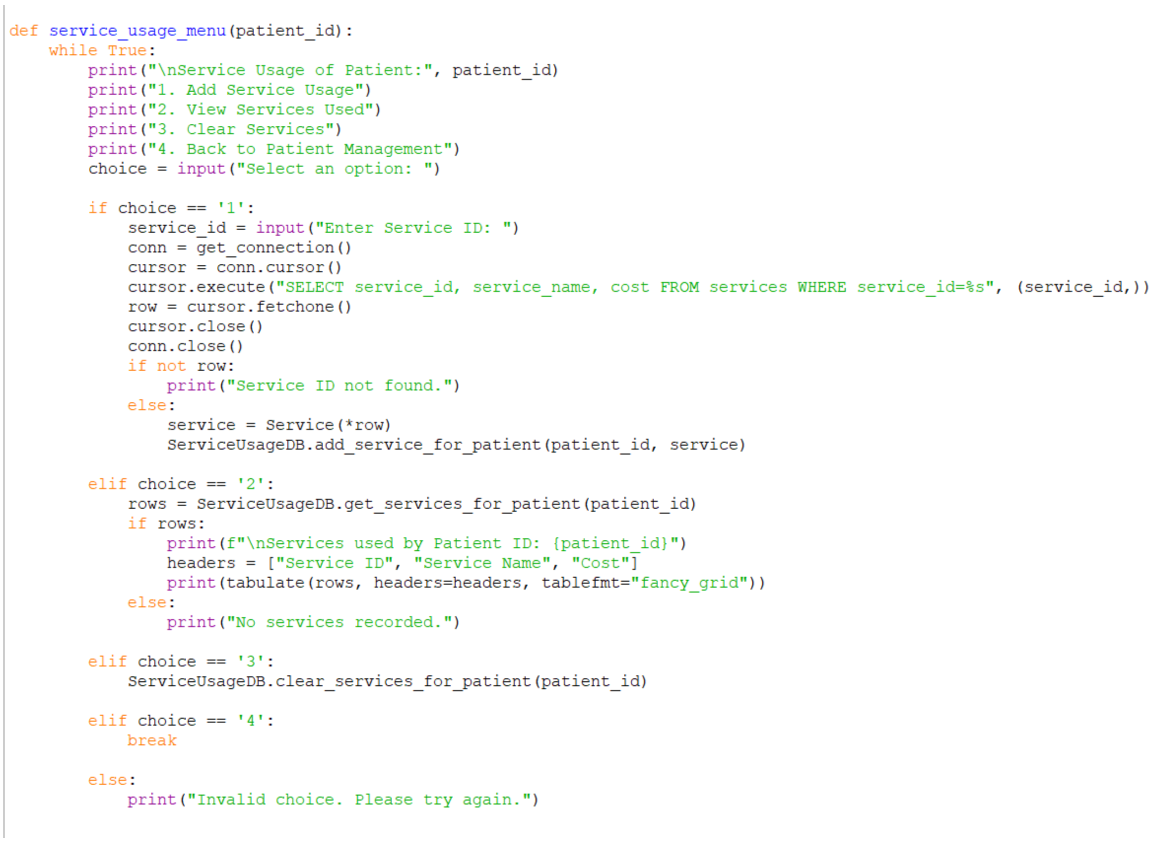
Deletes all temp service records for a patient. Usually done after billing is finalized.

```python
@staticmethod
def clear_services_for_patient(patient_id):
    try:
        conn = get_connection()
        cursor = conn.cursor()
        sql = "DELETE FROM temp_service_usage WHERE patient_id=%s"
        cursor.execute(sql, (patient_id,))
        conn.commit()
        print(f"Cleared services for patient {patient_id}")
    except Error as e:
        print("Database error while clearing services:", e)
    except Exception as e:
        print("Error while clearing services:", e)
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

service_usage_menu(patient_id)

Menu to manage services used by a patient:

1. **Add Service Usage** – Prompts for a service ID, fetches service data, and logs it.
2. **View Services Used** – Shows a formatted table of services.
3. **Clear Services** – Removes all services used by a patient.
4. **Back to Patient Menu** – Exits.



```python
def service_usage_menu(patient_id):
    while True:
        print("\nService Usage of Patient:", patient_id)
        print("1. Add Service Usage")
        print("2. View Services Used")
        print("3. Clear Services")
        print("4. Back to Patient Management")
        choice = input("Select an option: ")

        if choice == '1':
            service_id = input("Enter Service ID: ")
            conn = get_connection()
            cursor = conn.cursor()
            cursor.execute("SELECT service_id, service_name, cost FROM services WHERE service_id=%s", (service_id,))
            row = cursor.fetchone()
            cursor.close()
            conn.close()
            if not row:
                print("Service ID not found.")
            else:
                service = Service(*row)
                ServiceUsageDB.add_service_for_patient(patient_id, service)

        elif choice == '2':
            rows = ServiceUsageDB.get_services_for_patient(patient_id)
            if rows:
                print(f"\nServices used by Patient ID: {patient_id}")
                headers = ["Service ID", "Service Name", "Cost"]
                print(tabulate(rows, headers=headers, tablefmt="fancy_grid"))
            else:
                print("No services recorded.")

        elif choice == '3':
            ServiceUsageDB.clear_services_for_patient(patient_id)

        elif choice == '4':
            break

        else:
            print("Invalid choice. Please try again.")
```

Appointment Module (appointment.py)

This Appointment class in Python is part of a Hospital Management System. It handles appointment scheduling, modification, deletion, viewing, searching, CSV exporting, and date-based operations using MySQL as the backend and OOP principles in Core Python.

Imports

```
import re
from db_config import get_connection
import mysql.connector
from mysql.connector import IntegrityError, Error
import csv
from tabulate import tabulate
```

- re: For regular expressions (used in validating the date format).
- get_connection: Custom method from db_config.py to get a connection to the MySQL database.
- mysql.connector: Python library to interact with MySQL.
- csv: For exporting data to CSV files.
- tabulate: For displaying tabular data in a pretty format in the console.

auto_appt_id() Function

Automatically generate a new unique appointment ID in format A001, A002, etc.

- Fetch all IDs starting with 'A'.
- Extract the numeric part, convert it to int.
- Increment the maximum number and return formatted ID like A004.

```
def auto_appt_id():
    from db_config import get_connection
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT appt_id FROM appointments WHERE appt_id LIKE 'A%'")
    ids = [int(row[0][1:]) for row in cursor.fetchall() if row[0][1:].isdigit()]
    cursor.close()
    conn.close()
    next_num = max(ids) + 1 if ids else 1
    return f"A{next_num:03d}"
```

Class Appointment

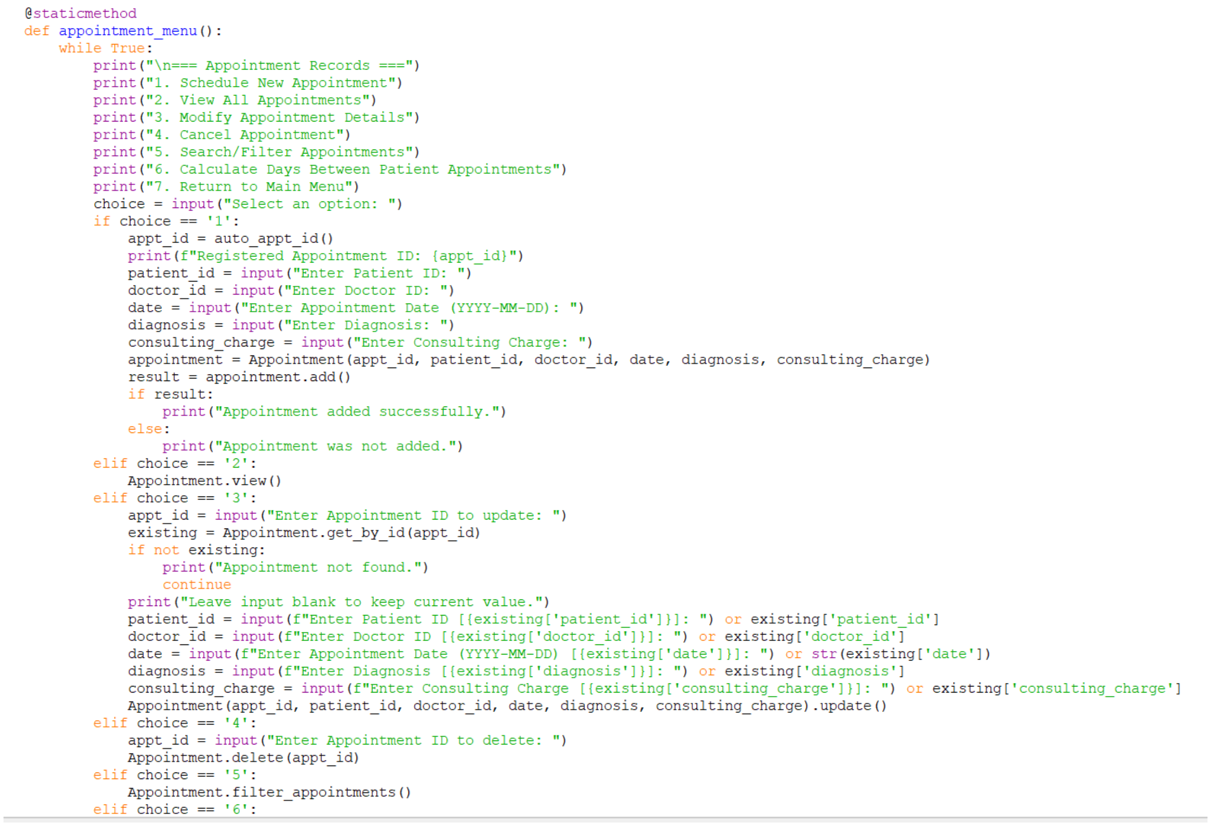
```
class Appointment:
    def __init__(self, appt_id, patient_id, doctor_id, date, diagnosis, consulting_charge):
        self.appt_id = appt_id
        self.patient_id = patient_id
        self.doctor_id = doctor_id
        self.date = date
        self.diagnosis = diagnosis
        self.consulting_charge = consulting_charge
```

This encapsulates all functionality related to appointment records. Stores appointment details as instance variables.

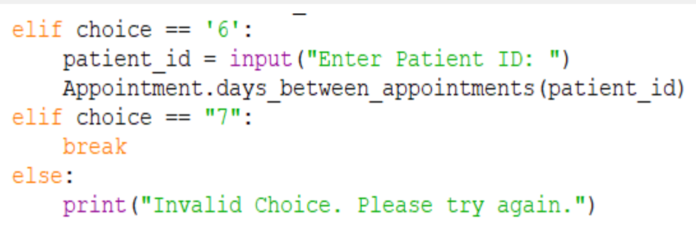
appointment_menu() :

Menu-driven interface to access appointment features.

- Schedule a new appointment.
- View all appointments.
- Modify an appointment.
- Cancel an appointment.
- Search/filter appointments by date range.
- Calculate days between a patient's appointments.
- Exit.



```python
@staticmethod
def appointment_menu():
    while True:
        print("\n=== Appointment Records ===")
        print("1. Schedule New Appointment")
        print("2. View All Appointments")
        print("3. Modify Appointment Details")
        print("4. Cancel Appointment")
        print("5. Search/Filter Appointments")
        print("6. Calculate Days Between Patient Appointments")
        print("7. Return to Main Menu")
        choice = input("Select an option: ")
        if choice == '1':
            appt_id = auto_appt_id()
            print(f"Registered Appointment ID: {appt_id}")
            patient_id = input("Enter Patient ID: ")
            doctor_id = input("Enter Doctor ID: ")
            date = input("Enter Appointment Date (YYYY-MM-DD): ")
            diagnosis = input("Enter Diagnosis: ")
            consulting_charge = input("Enter Consulting Charge: ")
            appointment = Appointment(appt_id, patient_id, doctor_id, date, diagnosis, consulting_charge)
            result = appointment.add()
            if result:
                print("Appointment added successfully.")
            else:
                print("Appointment was not added.")
        elif choice == '2':
            Appointment.view()
        elif choice == '3':
            appt_id = input("Enter Appointment ID to update: ")
            existing = Appointment.get_by_id(appt_id)
            if not existing:
                print("Appointment not found.")
                continue
            print("Leave input blank to keep current value.")
            patient_id = input(f"Enter Patient ID [{existing['patient_id']}]: ") or existing['patient_id']
            doctor_id = input(f"Enter Doctor ID [{existing['doctor_id']}]: ") or existing['doctor_id']
            date = input(f"Enter Appointment Date (YYYY-MM-DD) [{existing['date']}]: ") or str(existing['date'])
            diagnosis = input(f"Enter Diagnosis [{existing['diagnosis']}]: ") or existing['diagnosis']
            consulting_charge = input(f"Enter Consulting Charge [{existing['consulting_charge']}]: ") or existing['consulting_charge']
            Appointment(appt_id, patient_id, doctor_id, date, diagnosis, consulting_charge).update()
        elif choice == '4':
            appt_id = input("Enter Appointment ID to delete: ")
            Appointment.delete(appt_id)
        elif choice == '5':
            Appointment.filter_appointments()
        elif choice == '6':
```



```python
        elif choice == '6':
            patient_id = input("Enter Patient ID: ")
            Appointment.days_between_appointments(patient_id)
        elif choice == "7":
            break
        else:
            print("Invalid Choice. Please try again.")
```

add()

- Validates inputs: Checks for empty/invalid patient ID, doctor ID, date format, diagnosis, charge.
- Inserts a new row into the appointments table.
- Handles errors:
    - Duplicate ID.
    - Database error.

```
def add(self):
    if not self.patient_id or not str(self.patient_id).isdigit():
        print("Invalid Patient ID.")
        return False
    if not self.doctor_id or not isinstance(self.doctor_id, str):
        print("Invalid Doctor ID.")
        return False
    if not self.date or not re.match(r'^\d{4}-\d{2}-\d{2}$', self.date):
        print("Invalid Date. Use YYYY-MM-DD format.")
        return False
    if not self.diagnosis or not isinstance(self.diagnosis, str):
        print("Invalid Diagnosis.")
        return False
    try:
        charge = float(self.consulting_charge)
        if charge < 0:
            print("Consulting charge cannot be negative.")
            return False
    except ValueError:
        print("Invalid consulting charge. Enter a valid number.")
        return False
    try:
        conn = get_connection()
        cursor = conn.cursor()
        sql = "INSERT INTO appointments (appt_id, patient_id, doctor_id, date, diagnosis, consulting_charge) VALUES (%s, %s, %s, %s, %s, %s)"
        cursor.execute(sql, (self.appt_id, self.patient_id, self.doctor_id, self.date, self.diagnosis, charge))
        conn.commit()
        return True
    except mysql.connector.errors.IntegrityError as e:
        if "PRIMARY" in str(e):
            print(f"Error: Duplicate Appointment ID '{self.appt_id}'.")
        else:
            print("Database integrity error: ", e)
        return False
    except Exception as e:
        print("Error while adding appointment:", e)
        return False
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

update()

- Similar to add(), but updates existing appointment details.
- SQL: UPDATE appointments SET ... WHERE appt_id = %s
- Returns success or error message.

```
def update(self):
    if not self.patient_id or not str(self.patient_id).isdigit():
        print("Invalid Patient ID.")
        return False
    if not self.doctor_id or not isinstance(self.doctor_id, str):
        print("Invalid Doctor ID.")
        return False
    if not self.date or not re.match(r'^\d{4}-\d{2}-\d{2}$', self.date):
        print("Invalid Date. Use YYYY-MM-DD format.")
        return False
    if not self.diagnosis or not isinstance(self.diagnosis, str):
        print("Invalid Diagnosis.")
        return False
    try:
        charge = float(self.consulting_charge)
        if charge < 0 or charge > 10000:
            print("Consulting charge must be between 0 and 10000.")
            return False
    except ValueError:
        print("Invalid Consulting Charge. Enter a valid number.")
        return False

    try:
        conn = get_connection()
        cursor = conn.cursor()
        sql = "UPDATE appointments SET patient_id=%s, doctor_id=%s, date=%s, diagnosis=%s, consulting_charge=%s WHERE appt_id=%s"
        cursor.execute(sql, (self.patient_id, self.doctor_id, self.date, self.diagnosis, charge, self.appt_id))
        conn.commit()
        if cursor.rowcount == 0:
            print("No updates found.")
            return False
        else:
            print("Appointment updated successfully.")
            return True
    except Error as e:
        print("Database error while updating appointment:", e)
        return False
    except Exception as e:
        print("Error while updating appointment:", e)
        return False
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

get_by_id()

Fetches appointment row by ID and returns it as a dictionary (thanks to dictionary=True).

```
@staticmethod
def get_by_id(appt_id):
    try:
        conn = get_connection()
        cursor = conn.cursor(dictionary=True)
        cursor.execute("SELECT * FROM appointments WHERE appt_id = %s", (appt_id,))
        return cursor.fetchone()
    except Exception as e:
        print("Error fetching appointment:", e)
        return None
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

delete()

- Deletes appointment by ID.
- Returns whether the operation was successful.

```
@staticmethod
def delete(appt_id):
    try:
        conn = get_connection()
        cursor = conn.cursor()
        sql = "DELETE FROM appointments WHERE appt_id=%s"
        cursor.execute(sql, (appt_id,))
        conn.commit()
        if cursor.rowcount == 0:
            print("Appointment ID not found.")
            return False
        else:
            print("Appointment cancelled successfully.")
            return True
    except Error as e:
        print("Database error while deleting appointment:", e)
        return False
    except Exception as e:
        print("Error while deleting appointment:", e)
        return False
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

view()

Fetches and displays all appointments using tabulate().

```
@staticmethod
def view():
    try:
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM appointments")
        rows = cursor.fetchall()
        headers = ["Appointment ID", "Patient ID", "Doctor ID", "Date", "Diagnosis", "Consulting Charge"]
        print(tabulate(rows, headers=headers, tablefmt="grid"))
    except Error as e:
        print("Database error while viewing appointments:", e)
    except Exception as e:
        print("Error while viewing appointments:", e)
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

filter_appointments()

- User inputs a date range.
- Fetches appointments between those dates.
- Displays them in a fancy table format.

```
@staticmethod
def filter_appointments():
    try:
        conn = get_connection()
        cursor = conn.cursor()
        start_date = input("Enter start date(YYYY-MM-DD): ")
        end_date = input("Enter end date(YYYY-MM-DD):")
        cursor.execute("SELECT * FROM appointments WHERE date BETWEEN %s and %s", (start_date, end_date))
        rows = cursor.fetchall()
        if rows:
            headers = ["Appointment ID", "Patient ID", "Doctor ID", "Date", "Diagnosis", "Consulting Charge"]
            print("\nAppointments from", start_date, "to", end_date)
            print(tabulate(rows, headers=headers, tablefmt="fancy_grid"))
        else:
            print("No appointments found in the given date range.")
    except Error as e:
        print("Database error while fetching appointments for given dates:", e)
    except Exception as e:
        print("Error while fetching appointments for given dates:", e)
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

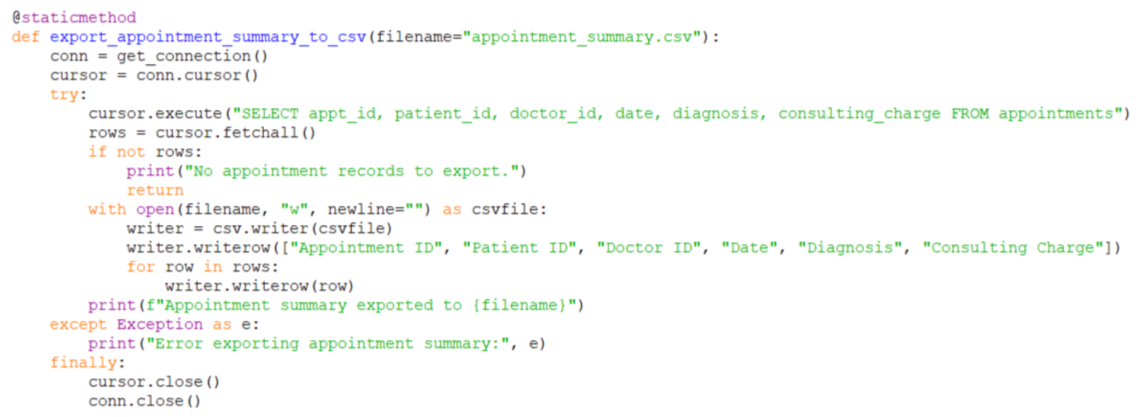
days_between_appointments()

- Calculates and prints the number of days between each pair of sequential appointments for a patient.
- Returns a list of days differences.

```
@staticmethod
def days_between_appointments(patient_id):
    conn = get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("SELECT date FROM appointments WHERE patient_id=%s ORDER BY date", (patient_id,))
        rows = cursor.fetchall()
        dates = [row[0] for row in rows if row[0]]
        if len(dates) < 2:
            print("Not enough appointments to calculate days between.")
            return []
        days_between = []
        for i in range(1, len(dates)):
            days = (dates[i] - dates[i-1]).days
            days_between.append(days)
        for idx, days in enumerate(days_between, 1):
            print(f"Days between appointment {idx} and {idx+1}: {days}")
        return days_between
    except Exception as e:
        print("Error calculating days between appointments:", e)
        return []
    finally:
        cursor.close()
        conn.close()
```

export_appointment_summary_to_csv()

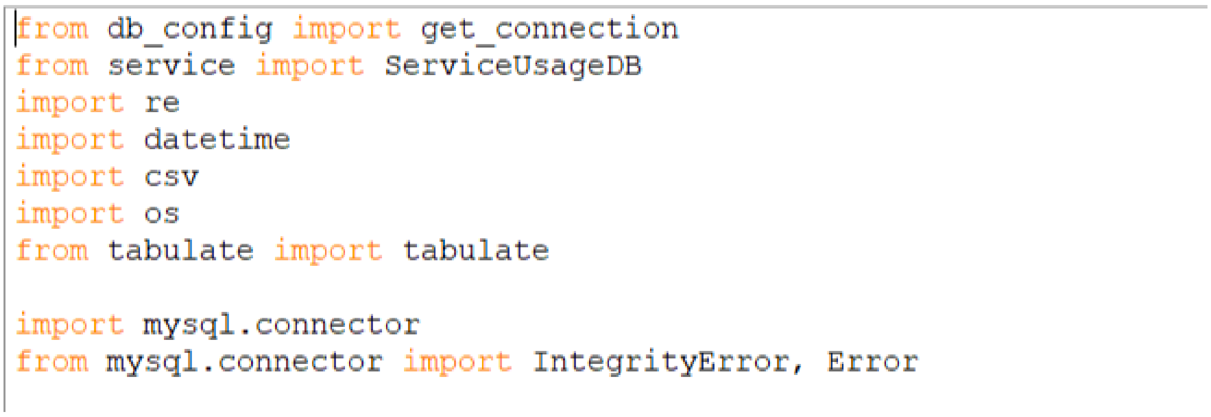
- Writes all appointments to a CSV file.
- Headers: Appointment ID, Patient ID, Doctor ID, Date, Diagnosis, Consulting Charge.



```
@staticmethod
def export_appointment_summary_to_csv(filename="appointment_summary.csv"):
    conn = get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("SELECT appt_id, patient_id, doctor_id, date, diagnosis, consulting_charge FROM appointments")
        rows = cursor.fetchall()
        if not rows:
            print("No appointment records to export.")
            return
        with open(filename, "w", newline="") as csvfile:
            writer = csv.writer(csvfile)
            writer.writerow(["Appointment ID", "Patient ID", "Doctor ID", "Date", "Diagnosis", "Consulting Charge"])
            for row in rows:
                writer.writerow(row)
        print(f"Appointment summary exported to {filename}")
    except Exception as e:
        print("Error exporting appointment summary:", e)
    finally:
        cursor.close()
        conn.close()
```

Billing Module (billing.py)

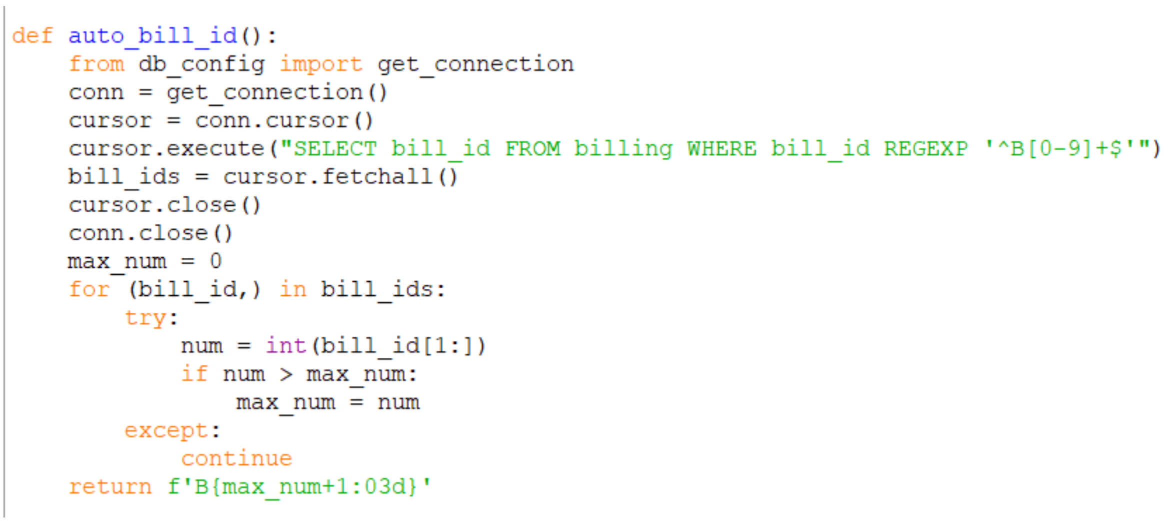
This module handles bill creation, modification, deletion, viewing, invoice generation, and exporting billing summaries. It also calculates total charges per patient by aggregating services and consultation fees.



```python
from db_config import get_connection
from service import ServiceUsageDB
import re
import datetime
import csv
import os
from tabulate import tabulate

import mysql.connector
from mysql.connector import IntegrityError, Error
```

- **Database Access**: get_connection() establishes a connection to the MySQL database.
- **ServiceUsageDB**: Temporary service usage logic for billing.
- **tabulate**: For formatted CLI output.
- **CSV & OS**: For file export and invoice creation.
- **Regex & datetime**: For validation and date processing.

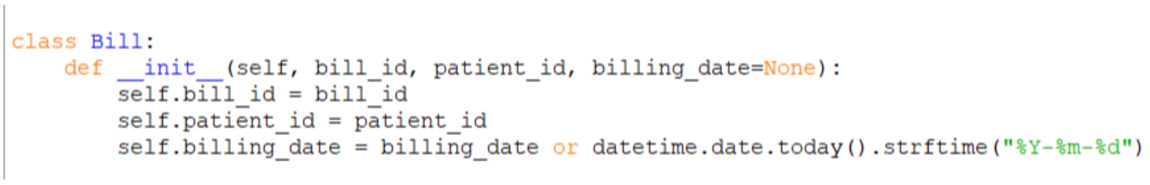
auto_bill_id()



```python
def auto_bill_id():
    from db_config import get_connection
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT bill_id FROM billing WHERE bill_id REGEXP '^B[0-9]+$'")
    bill_ids = cursor.fetchall()
    cursor.close()
    conn.close()
    max_num = 0
    for (bill_id,) in bill_ids:
        try:
            num = int(bill_id[1:])
            if num > max_num:
                max_num = num
        except:
            continue
    return f'B{max_num+1:03d}'
```

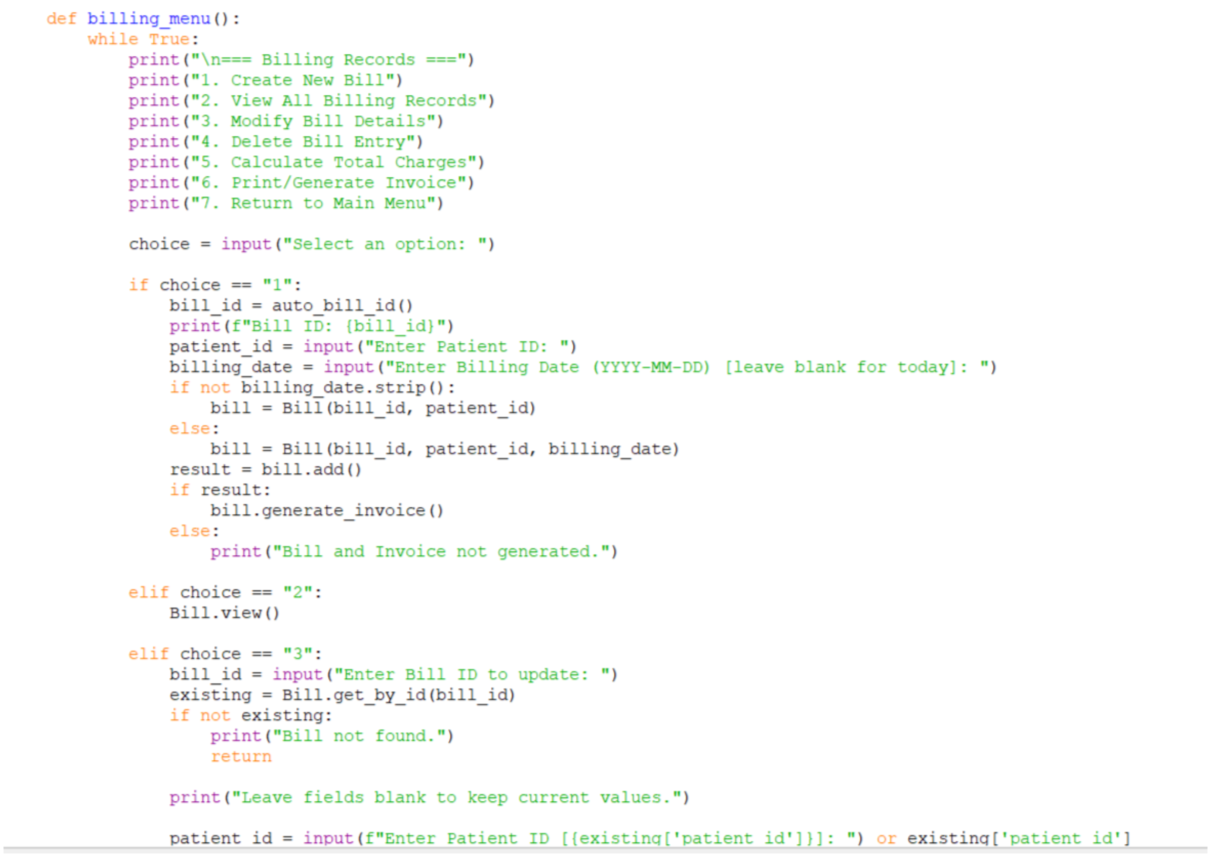
Generates a unique bill ID in the format B001, B002, ..., based on the max bill_id found in the billing table using a REGEXP.

Bill Class

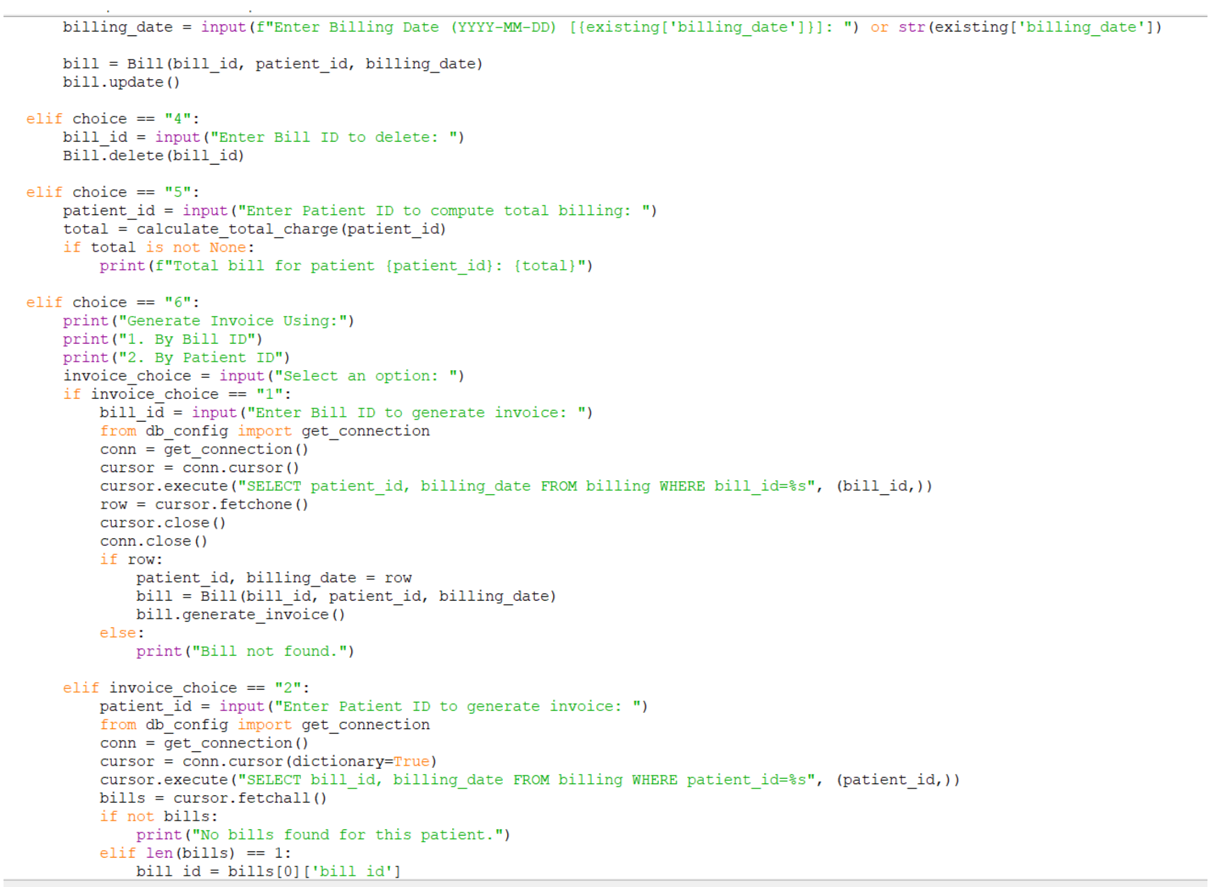


```
class Bill:
    def __init__(self, bill_id, patient_id, billing_date=None):
        self.bill_id = bill_id
        self.patient_id = patient_id
        self.billing_date = billing_date or datetime.date.today().strftime("%Y-%m-%d")
```

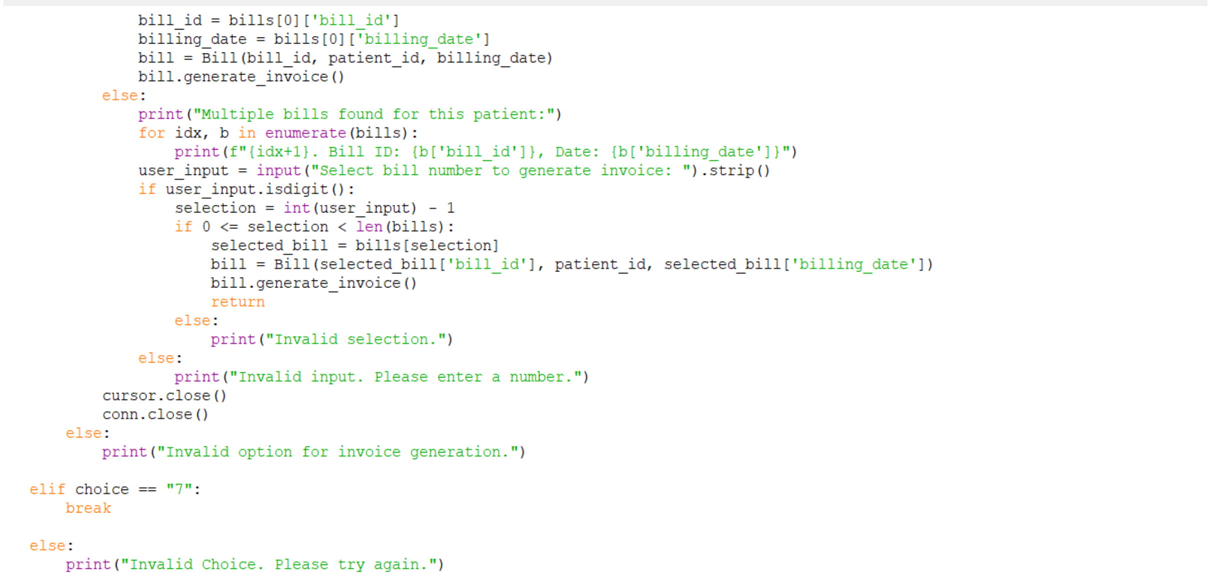
billing_menu() – Main CLI Interface



```
def billing_menu():
    while True:
        print("\n=== Billing Records ===")
        print("1. Create New Bill")
        print("2. View All Billing Records")
        print("3. Modify Bill Details")
        print("4. Delete Bill Entry")
        print("5. Calculate Total Charges")
        print("6. Print/Generate Invoice")
        print("7. Return to Main Menu")

        choice = input("Select an option: ")

        if choice == "1":
            bill_id = auto_bill_id()
            print(f"Bill ID: {bill_id}")
            patient_id = input("Enter Patient ID: ")
            billing_date = input("Enter Billing Date (YYYY-MM-DD) [leave blank for today]: ")
            if not billing_date.strip():
                bill = Bill(bill_id, patient_id)
            else:
                bill = Bill(bill_id, patient_id, billing_date)
            result = bill.add()
            if result:
                bill.generate_invoice()
            else:
                print("Bill and Invoice not generated.")

        elif choice == "2":
            Bill.view()

        elif choice == "3":
            bill_id = input("Enter Bill ID to update: ")
            existing = Bill.get_by_id(bill_id)
            if not existing:
                print("Bill not found.")
                return

            print("Leave fields blank to keep current values.")

            patient_id = input(f"Enter Patient ID [{existing['patient_id']}]: ") or existing['patient_id']
```



```
        billing_date = input(f"Enter Billing Date (YYYY-MM-DD) [{existing['billing_date']}]: ") or str(existing['billing_date'])

        bill = Bill(bill_id, patient_id, billing_date)
        bill.update()

    elif choice == "4":
        bill_id = input("Enter Bill ID to delete: ")
        Bill.delete(bill_id)

    elif choice == "5":
        patient_id = input("Enter Patient ID to compute total billing: ")
        total = calculate_total_charge(patient_id)
        if total is not None:
            print(f"Total bill for patient {patient_id}: {total}")

    elif choice == "6":
        print("Generate Invoice Using:")
        print("1. By Bill ID")
        print("2. By Patient ID")
        invoice_choice = input("Select an option: ")
        if invoice_choice == "1":
            bill_id = input("Enter Bill ID to generate invoice: ")
            from db_config import get_connection
            conn = get_connection()
            cursor = conn.cursor()
            cursor.execute("SELECT patient_id, billing_date FROM billing WHERE bill_id=%s", (bill_id,))
            row = cursor.fetchone()
            cursor.close()
            conn.close()
            if row:
                patient_id, billing_date = row
                bill = Bill(bill_id, patient_id, billing_date)
                bill.generate_invoice()
            else:
                print("Bill not found.")

        elif invoice_choice == "2":
            patient_id = input("Enter Patient ID to generate invoice: ")
            from db_config import get_connection
            conn = get_connection()
            cursor = conn.cursor(dictionary=True)
            cursor.execute("SELECT bill_id, billing_date FROM billing WHERE patient_id=%s", (patient_id,))
            bills = cursor.fetchall()
            if not bills:
                print("No bills found for this patient.")
            elif len(bills) == 1:
                bill_id = bills[0]['bill_id']
```



```
                bill_id = bills[0]['bill_id']
                billing_date = bills[0]['billing_date']
                bill = Bill(bill_id, patient_id, billing_date)
                bill.generate_invoice()
            else:
                print("Multiple bills found for this patient:")
                for idx, b in enumerate(bills):
                    print(f"{idx+1}. Bill ID: {b['bill_id']}, Date: {b['billing_date']}")
                user_input = input("Select bill number to generate invoice: ").strip()
                if user_input.isdigit():
                    selection = int(user_input) - 1
                    if 0 <= selection < len(bills):
                        selected_bill = bills[selection]
                        bill = Bill(selected_bill['bill_id'], patient_id, selected_bill['billing_date'])
                        bill.generate_invoice()
                        return
                    else:
                        print("Invalid selection.")
                else:
                    print("Invalid input. Please enter a number.")
            cursor.close()
            conn.close()
        else:
            print("Invalid option for invoice generation.")

    elif choice == "7":
        break

    else:
        print("Invalid Choice. Please try again.")
```

A static menu-driven CLI to handle billing tasks:

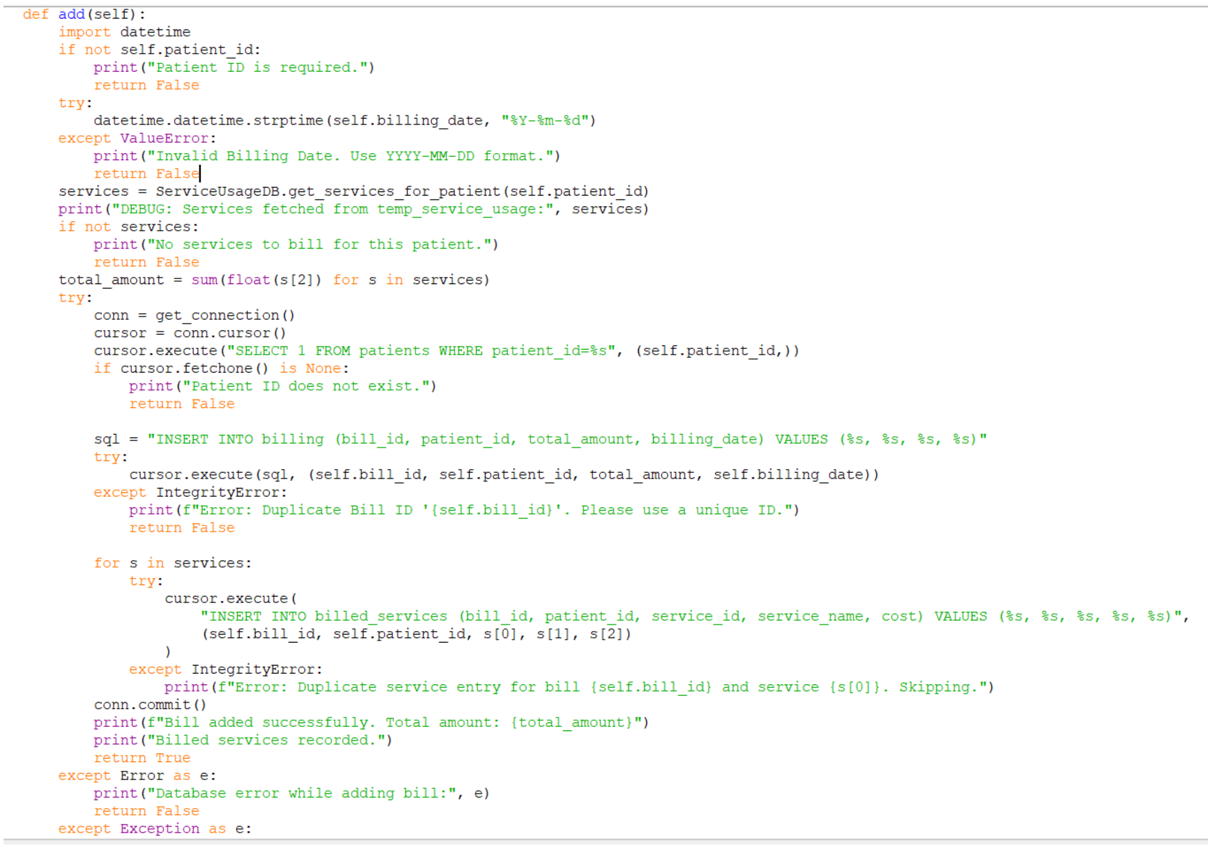
Option	**Action**
1	Create new bill and generate invoice
2	View all billing records (tabulated)
3	Modify bill details

Option	Action
4	Delete bill entry
5	Calculate total billing (services + consulting)
6	Generate invoice (by bill ID or patient ID)
7	Exit menu

add(self)

Adds a bill entry after validating and fetching service costs.

- Validates patient_id and billing_date.
- Fetches services used by patient from temp_service_usage.
- Calculates total and inserts into:
    - billing table.
    - billed_services table (item-wise).
- Commits transaction and clears the temporary service table.



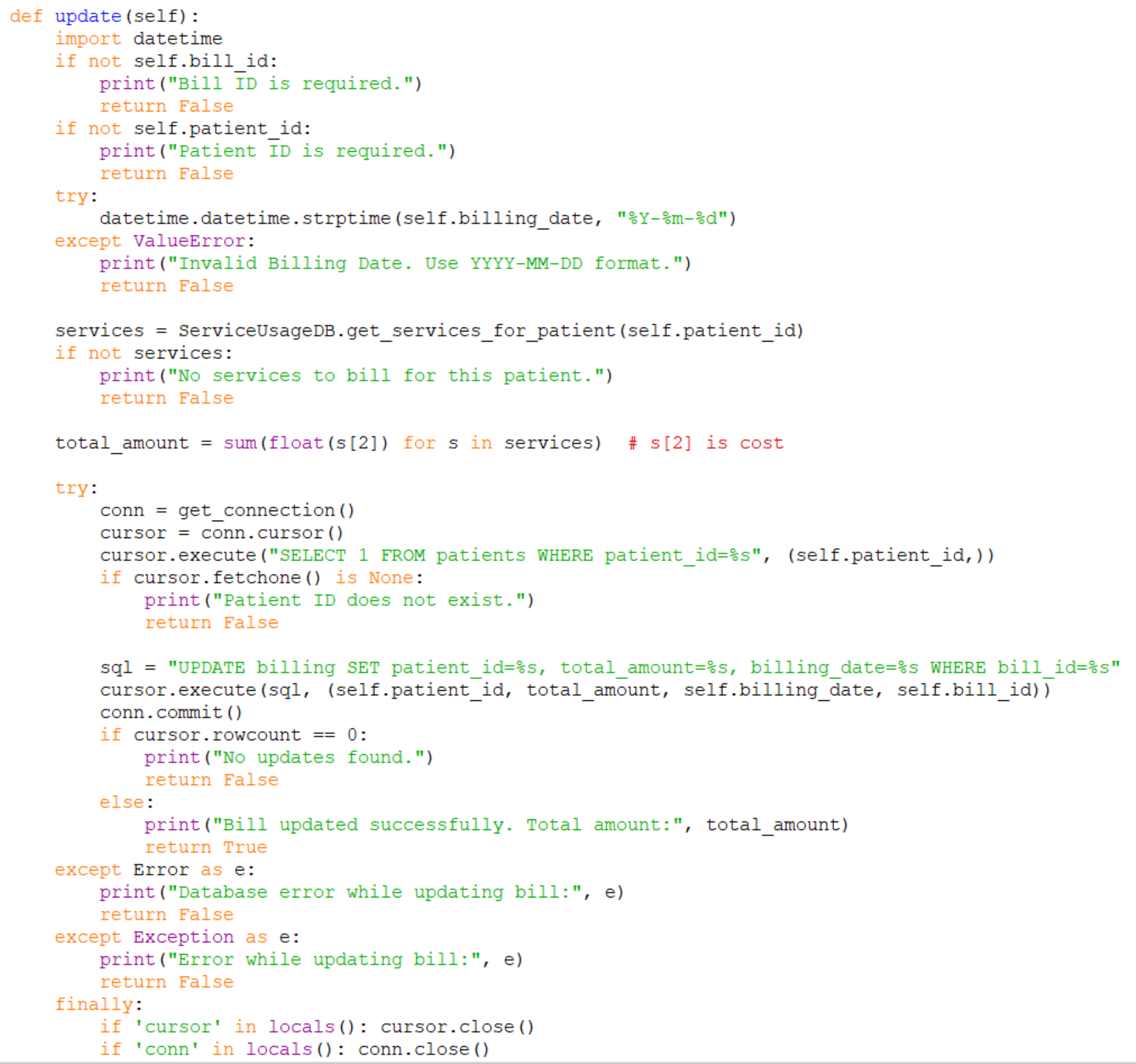
```
def add(self):
    import datetime
    if not self.patient_id:
        print("Patient ID is required.")
        return False
    try:
        datetime.datetime.strptime(self.billing_date, "%Y-%m-%d")
    except ValueError:
        print("Invalid Billing Date. Use YYYY-MM-DD format.")
        return False
    services = ServiceUsageDB.get_services_for_patient(self.patient_id)
    print("DEBUG: Services fetched from temp_service_usage:", services)
    if not services:
        print("No services to bill for this patient.")
        return False
    total_amount = sum(float(s[2]) for s in services)
    try:
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT 1 FROM patients WHERE patient_id=%s", (self.patient_id,))
        if cursor.fetchone() is None:
            print("Patient ID does not exist.")
            return False

        sql = "INSERT INTO billing (bill_id, patient_id, total_amount, billing_date) VALUES (%s, %s, %s, %s)"
        try:
            cursor.execute(sql, (self.bill_id, self.patient_id, total_amount, self.billing_date))
        except IntegrityError:
            print(f"Error: Duplicate Bill ID '{self.bill_id}'. Please use a unique ID.")
            return False

        for s in services:
            try:
                cursor.execute(
                    "INSERT INTO billed_services (bill_id, patient_id, service_id, service_name, cost) VALUES (%s, %s, %s, %s, %s)",
                    (self.bill_id, self.patient_id, s[0], s[1], s[2])
                )
            except IntegrityError:
                print(f"Error: Duplicate service entry for bill {self.bill_id} and service {s[0]}. Skipping.")
        conn.commit()
        print(f"Bill added successfully. Total amount: {total_amount}")
        print("Billed services recorded.")
        return True
    except Error as e:
        print("Database error while adding bill:", e)
        return False
    except Exception as e:
```

```
    except Exception as e:
        print("Error while adding bill:", e)
        return False
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()

    try:
        ServiceUsageDB.clear_services_for_patient(self.patient_id)
    except Exception as e:
        print("Error clearing temp service usage:", e)
```

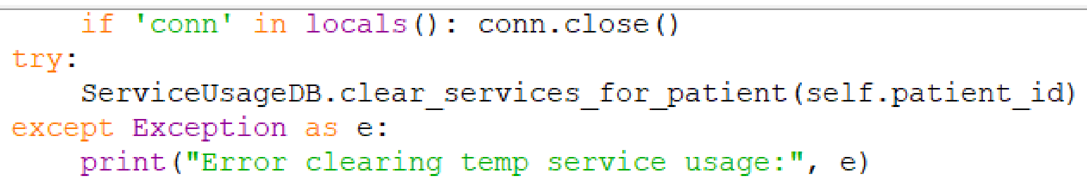
update(self)

Updates an existing bill:

- Recalculates total charges from current services.
- Updates the billing table.
- Clears and reinserts billed services (if required).



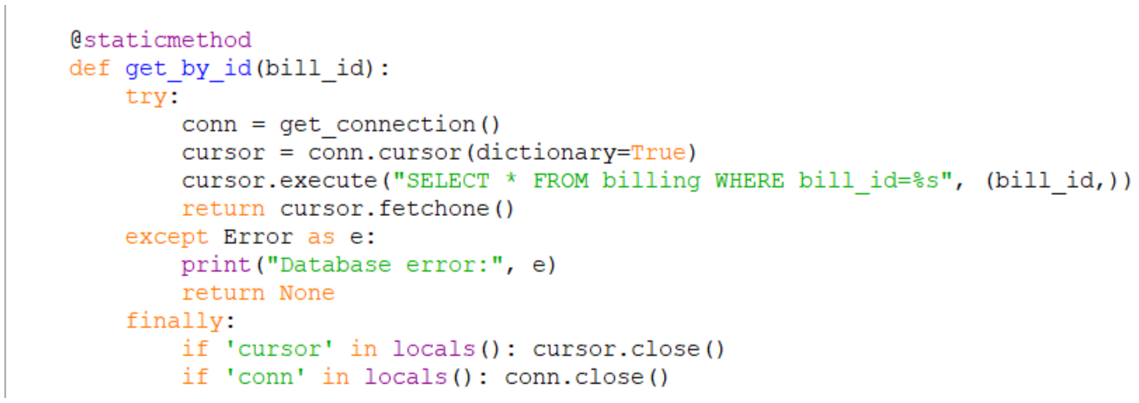
```
def update(self):
    import datetime
    if not self.bill_id:
        print("Bill ID is required.")
        return False
    if not self.patient_id:
        print("Patient ID is required.")
        return False
    try:
        datetime.datetime.strptime(self.billing_date, "%Y-%m-%d")
    except ValueError:
        print("Invalid Billing Date. Use YYYY-MM-DD format.")
        return False

    services = ServiceUsageDB.get_services_for_patient(self.patient_id)
    if not services:
        print("No services to bill for this patient.")
        return False

    total_amount = sum(float(s[2]) for s in services)  # s[2] is cost

    try:
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT 1 FROM patients WHERE patient_id=%s", (self.patient_id,))
        if cursor.fetchone() is None:
            print("Patient ID does not exist.")
            return False

        sql = "UPDATE billing SET patient_id=%s, total_amount=%s, billing_date=%s WHERE bill_id=%s"
        cursor.execute(sql, (self.patient_id, total_amount, self.billing_date, self.bill_id))
        conn.commit()
        if cursor.rowcount == 0:
            print("No updates found.")
            return False
        else:
            print("Bill updated successfully. Total amount:", total_amount)
            return True
    except Error as e:
        print("Database error while updating bill:", e)
        return False
    except Exception as e:
        print("Error while updating bill:", e)
        return False
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```



```
        if 'conn' in locals(): conn.close()
    try:
        ServiceUsageDB.clear_services_for_patient(self.patient_id)
    except Exception as e:
        print("Error clearing temp service usage:", e)
```

get_by_id(bill_id)

- Fetches a single bill record from the database using the bill_id.

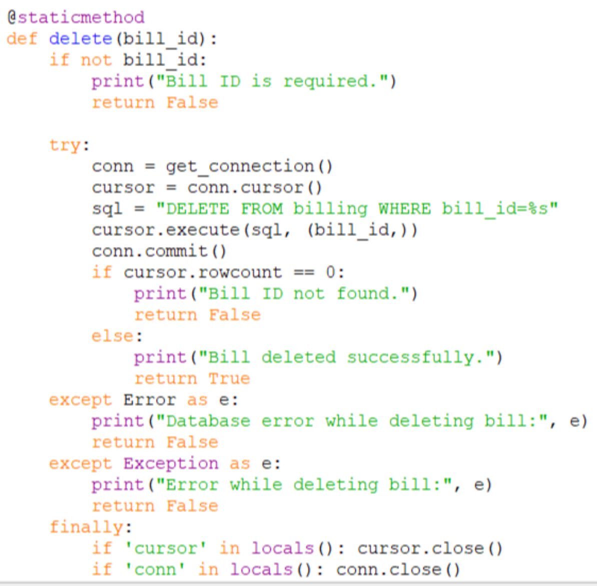


```python
@staticmethod
def get_by_id(bill_id):
    try:
        conn = get_connection()
        cursor = conn.cursor(dictionary=True)
        cursor.execute("SELECT * FROM billing WHERE bill_id=%s", (bill_id,))
        return cursor.fetchone()
    except Error as e:
        print("Database error:", e)
        return None
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

delete(bill_id)

Deletes a billing record from the database.

- Validates the bill ID.
- Deletes the record from the billing table.



```python
@staticmethod
def delete(bill_id):
    if not bill_id:
        print("Bill ID is required.")
        return False

    try:
        conn = get_connection()
        cursor = conn.cursor()
        sql = "DELETE FROM billing WHERE bill_id=%s"
        cursor.execute(sql, (bill_id,))
        conn.commit()
        if cursor.rowcount == 0:
            print("Bill ID not found.")
            return False
        else:
            print("Bill deleted successfully.")
            return True
    except Error as e:
        print("Database error while deleting bill:", e)
        return False
    except Exception as e:
        print("Error while deleting bill:", e)
        return False
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

view()

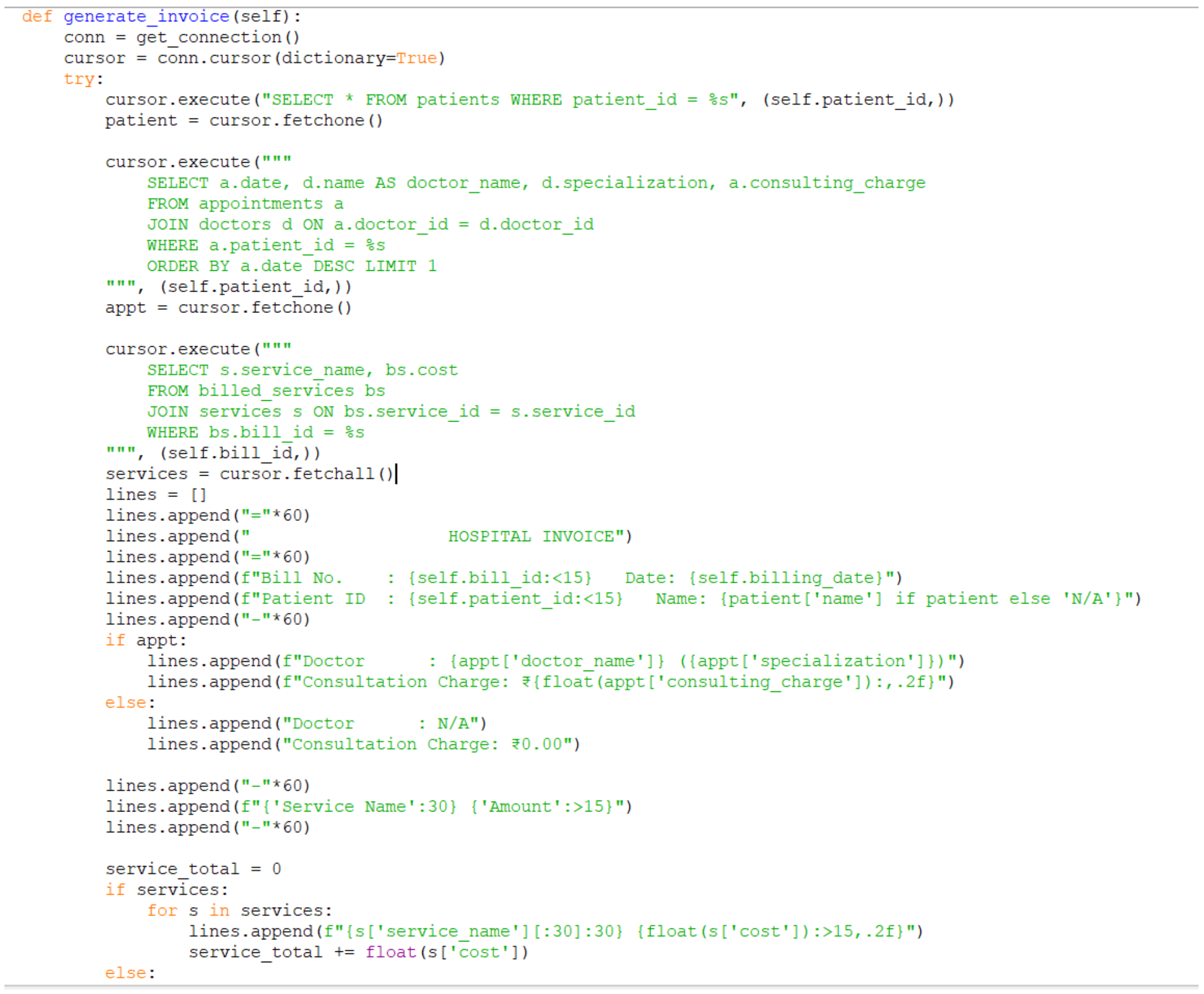
- Displays all bill records in a formatted table using tabulate

```
def view():
    try:
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM billing")
        rows = cursor.fetchall()
        headers = ["Bill ID", "Patient ID", "Total Amount", "Billing Date"]
        print(tabulate(rows, headers=headers, tablefmt="grid"))
    except Error as e:
        print("Database error while viewing bills:", e)
    except Exception as e:
        print("Error while viewing bills:", e)
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

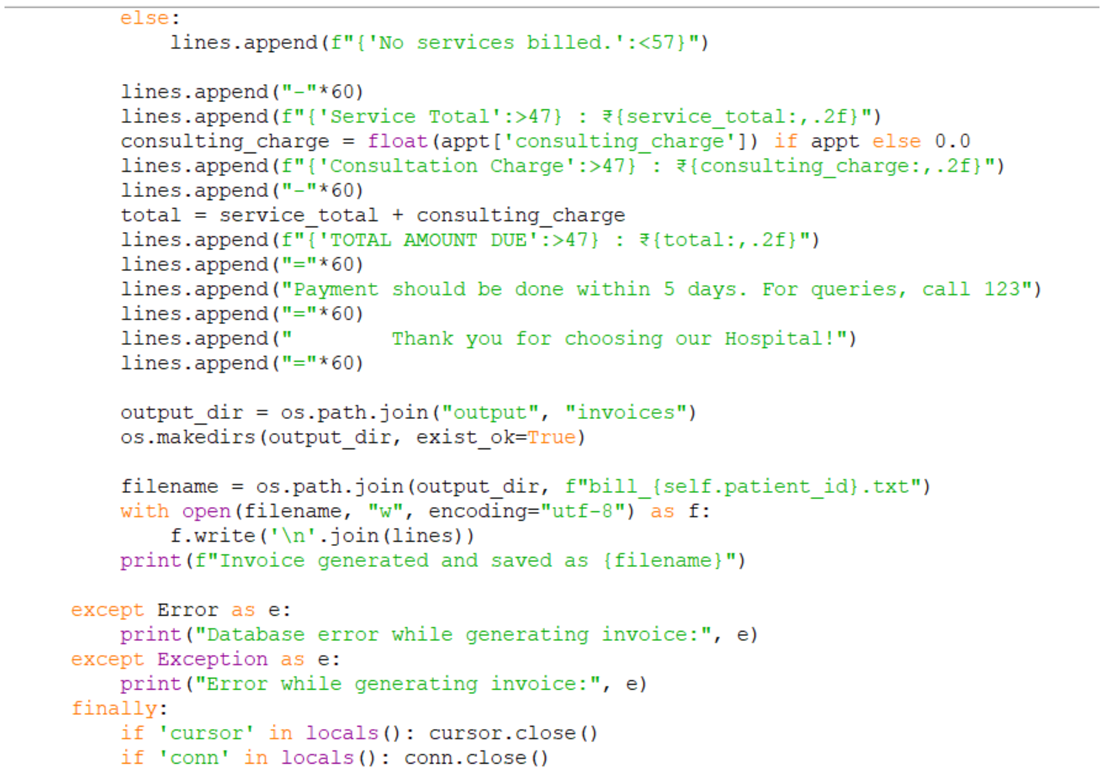
generate_invoice(self)

Creates a text invoice with patient, doctor, and service details:

- Fetches:
    - Patient name.
    - Latest doctor appointment and charge.
    - Services associated with the bill.
- Calculates totals and writes to a .txt file inside output/invoices/.
- Invoice format includes:
    - Hospital heading.
    - Bill and patient details.
    - Doctor consultation details.
    - Itemized service costs.
    - Total due amount.



```python
def generate_invoice(self):
    conn = get_connection()
    cursor = conn.cursor(dictionary=True)
    try:
        cursor.execute("SELECT * FROM patients WHERE patient_id = %s", (self.patient_id,))
        patient = cursor.fetchone()

        cursor.execute("""
            SELECT a.date, d.name AS doctor_name, d.specialization, a.consulting_charge
            FROM appointments a
            JOIN doctors d ON a.doctor_id = d.doctor_id
            WHERE a.patient_id = %s
            ORDER BY a.date DESC LIMIT 1
        """, (self.patient_id,))
        appt = cursor.fetchone()

        cursor.execute("""
            SELECT s.service_name, bs.cost
            FROM billed_services bs
            JOIN services s ON bs.service_id = s.service_id
            WHERE bs.bill_id = %s
        """, (self.bill_id,))
        services = cursor.fetchall()
        lines = []
        lines.append("="*60)
        lines.append("                   HOSPITAL INVOICE")
        lines.append("="*60)
        lines.append(f"Bill No.    : {self.bill_id:<15}   Date: {self.billing_date}")
        lines.append(f"Patient ID  : {self.patient_id:<15}   Name: {patient['name'] if patient else 'N/A'}")
        lines.append("-"*60)
        if appt:
            lines.append(f"Doctor      : {appt['doctor_name']} ({appt['specialization']})")
            lines.append(f"Consultation Charge: ₹{float(appt['consulting_charge']):,.2f}")
        else:
            lines.append("Doctor      : N/A")
            lines.append("Consultation Charge: ₹0.00")

        lines.append("-"*60)
        lines.append(f"{'Service Name':30} {'Amount':>15}")
        lines.append("-"*60)

        service_total = 0
        if services:
            for s in services:
                lines.append(f"{s['service_name'][:30]:30} {float(s['cost']):>15,.2f}")
                service_total += float(s['cost'])
        else:
```



```python
        else:
            lines.append(f"{'No services billed.':<57}")

        lines.append("-"*60)
        lines.append(f"{'Service Total':>47} : ₹{service_total:,.2f}")
        consulting_charge = float(appt['consulting_charge']) if appt else 0.0
        lines.append(f"{'Consultation Charge':>47} : ₹{consulting_charge:,.2f}")
        lines.append("-"*60)
        total = service_total + consulting_charge
        lines.append(f"{'TOTAL AMOUNT DUE':>47} : ₹{total:,.2f}")
        lines.append("="*60)
        lines.append("Payment should be done within 5 days. For queries, call 123")
        lines.append("="*60)
        lines.append("        Thank you for choosing our Hospital!")
        lines.append("="*60)

        output_dir = os.path.join("output", "invoices")
        os.makedirs(output_dir, exist_ok=True)

        filename = os.path.join(output_dir, f"bill_{self.patient_id}.txt")
        with open(filename, "w", encoding="utf-8") as f:
            f.write('\n'.join(lines))
        print(f"Invoice generated and saved as {filename}")

    except Error as e:
        print("Database error while generating invoice:", e)
    except Exception as e:
        print("Error while generating invoice:", e)
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

export_billing_summary_to_csv(filename)

Exports all billing data to a CSV file with columns:

- Bill ID
- Patient ID
- Total Amount
- Billing Date

```
@staticmethod
def export_billing_summary_to_csv(filename="billing_summary.csv"):
    conn = get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("SELECT bill_id, patient_id, total_amount, billing_date FROM billing")
        rows = cursor.fetchall()
        if not rows:
            print("No billing records to export.")
            return
        with open(filename, "w", newline="") as csvfile:
            writer = csv.writer(csvfile)
            writer.writerow(["Bill ID", "Patient ID", "Total Amount", "Billing Date"])
            for row in rows:
                writer.writerow(row)
        print(f"Billing summary exported to {filename}")
    except Exception as e:
        print("Error exporting billing summary:", e)
    finally:
        cursor.close()
        conn.close()
```

calculate_total_charge(patient_id)

Calculates the total bill amount for a patient by summing:

- Services used (from temp_service_usage).
- Consultation charges (from appointments).

Outputs the intermediate and final totals for clarity.

```
def calculate_total_charge(patient_id):
    try:
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute(
            "SELECT COALESCE(SUM(cost), 0) FROM temp_service_usage WHERE patient_id=%s",
            (patient_id,)
        )
        service_total = cursor.fetchone()[0] or 0

        cursor.execute(
            "SELECT COALESCE(SUM(consulting_charge), 0) FROM appointments WHERE patient_id=%s",
            (patient_id,)
        )
        consulting_total = cursor.fetchone()[0] or 0

        total_billing = service_total + consulting_total
        print(f"Service Total: {service_total}")
        print(f"Consulting Total: {consulting_total}")
        print(f"Total Billing: {total_billing}")
        return total_billing
    except Error as e:
        print("Database error while computing total billing:", e)
        return None
    except Exception as e:
        print("Error while computing total billing:", e)
        return None
    finally:
        if 'cursor' in locals(): cursor.close()
        if 'conn' in locals(): conn.close()
```

Import CSV module (import_csv.py)

Using Python along with the pandas and mysql.connector libraries to import multiple CSV files into a MySQL database called "hospitalmanagementsystem"

1. **Import Required Modules**

- pandas: For reading and processing CSV files.
- mysql.connector: To connect and interact with the MySQL database.
- os: To check file paths and get the current working directory.

2. **Print Current Working Directory**

Helps confirm that Python is executing from the expected directory (useful for debugging paths).

3. **Connect to MySQL Database**

- Establishes a connection to the MySQL database using credentials.
- Creates a cursor for executing SQL commands.

4. **Reusable Function: import_csv_to_table()**

A flexible function that:

- Takes the path to a CSV file
- Specifies which database table to insert into
- Accepts a list of column names and optional date columns
- Parses the CSV into a DataFrame
- Converts NaN to None (SQL NULL)
- Formats date columns properly
- Executes row-wise insertions into the database
- Catches and prints any row-specific errors

5. Call Function for Each CSV File

- Each call imports a different dataset

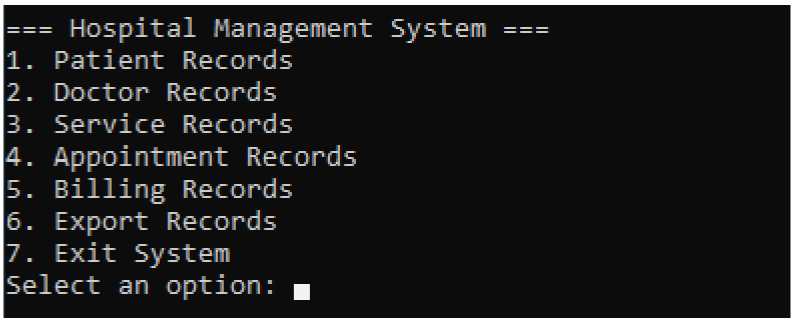
6. **Close the DB Connection**

```python
import pandas as pd
import mysql.connector
import os

print("Current working directory:", os.getcwd())

conn = mysql.connector.connect(
    host='localhost',
    user='root',
    password='123456',
    database='hospitalmanagementsystem'
)
cursor = conn.cursor()

def import_csv_to_table(csv_file, table_name, columns, date_columns=None):
    try:
        df = pd.read_csv(csv_file)

        df = df.where(pd.notnull(df), None)

        if date_columns:
            for col in date_columns:
                df[col] = pd.to_datetime(df[col], errors='coerce').dt.date

        placeholders = ','.join(['%s'] * len(columns))
        cols = ','.join(columns)
        sql = f"INSERT INTO {table_name} ({cols}) VALUES ({placeholders})"

        for index, row in df.iterrows():
            try:
                cursor.execute(sql, tuple(row[col] for col in columns))
            except Exception as row_err:
                print(f"Row {index + 1} skipped due to error: {row_err}")
                print("Row data:", row.to_dict())

        conn.commit()
        print(f"✅ Imported {len(df)} rows into '{table_name}' from '{os.path.basename(csv_file)}'")

    except Exception as e:
        print(f"❌ Error importing '{csv_file}' into table '{table_name}': {e}")
```

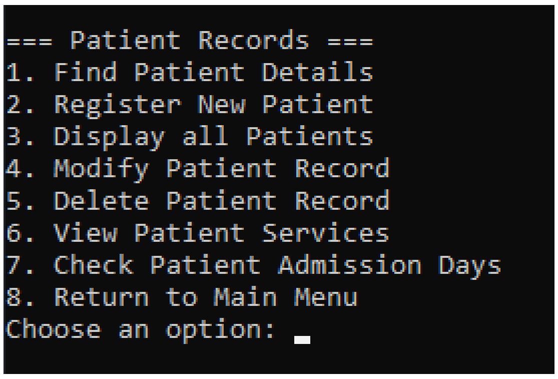
```python
import_csv_to_table(
    csv_file=r'C:/Users/kadal/Desktop/hospital_mgmt-main/CSVreports/patients_dataset.csv',
    table_name='patients',
    columns=['patient_id', 'name', 'age', 'gender', 'admission_date', 'contact_no'],
    date_columns=['admission_date']
)

import_csv_to_table(
    csv_file=r'C:/Users/kadal/Desktop/hospital_mgmt-main/CSVreports/doctors_dataset.csv',
    table_name='doctors',
    columns=['doctor_id', 'name', 'specialization', 'contact_no']
)

import_csv_to_table(
    csv_file=r'C:/Users/kadal/Desktop/hospital_mgmt-main/CSVreports/services_dataset.csv',
    table_name='services',
    columns=['service_id', 'service_name', 'cost']
)

import_csv_to_table(
    csv_file=r'C:/Users/kadal/Desktop/hospital_mgmt-main/CSVreports/appointments_dataset.csv',
    table_name='appointments',
    columns=['appt_id', 'patient_id', 'doctor_id', 'date', 'diagnosis', 'consulting_charge'],
    date_columns=['date']
)

import_csv_to_table(
    csv_file=r'C:/Users/kadal/Desktop/hospital_mgmt-main/CSVreports/billing_dataset.csv',
    table_name='billing',
    columns=['bill_id', 'patient_id', 'total_amount', 'billing_date'],
    date_columns=['billing_date']
)

# Close connection
cursor.close()
conn.close()
print("✅ All CSVs imported successfully and DB connection closed.")
```

Cli workflow images

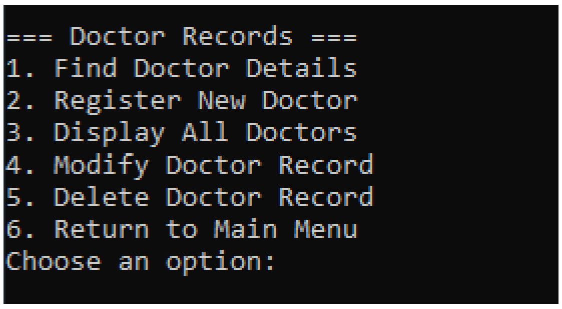
Main Menu of each record



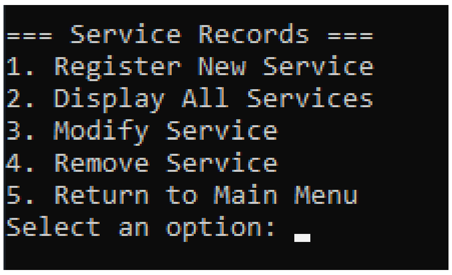
```
=== Hospital Management System ===
1. Patient Records
2. Doctor Records
3. Service Records
4. Appointment Records
5. Billing Records
6. Export Records
7. Exit System
Select an option:
```



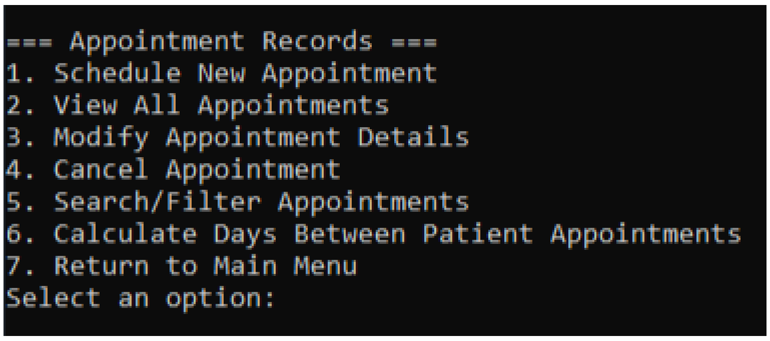
```
=== Patient Records ===
1. Find Patient Details
2. Register New Patient
3. Display all Patients
4. Modify Patient Record
5. Delete Patient Record
6. View Patient Services
7. Check Patient Admission Days
8. Return to Main Menu
Choose an option:
```



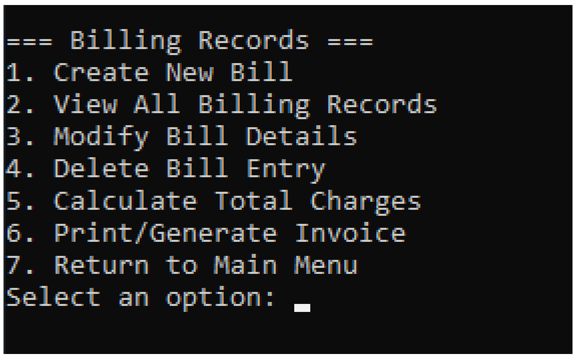
```
=== Doctor Records ===
1. Find Doctor Details
2. Register New Doctor
3. Display All Doctors
4. Modify Doctor Record
5. Delete Doctor Record
6. Return to Main Menu
Choose an option:
```



```
=== Service Records ===
1. Register New Service
2. Display All Services
3. Modify Service
4. Remove Service
5. Return to Main Menu
Select an option:
```

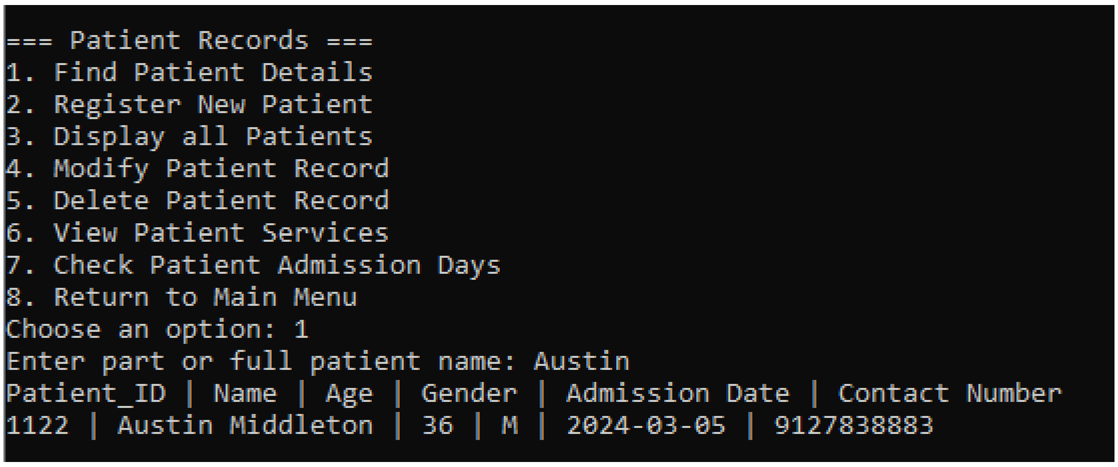


```
=== Appointment Records ===
1. Schedule New Appointment
2. View All Appointments
3. Modify Appointment Details
4. Cancel Appointment
5. Search/Filter Appointments
6. Calculate Days Between Patient Appointments
7. Return to Main Menu
Select an option:
```

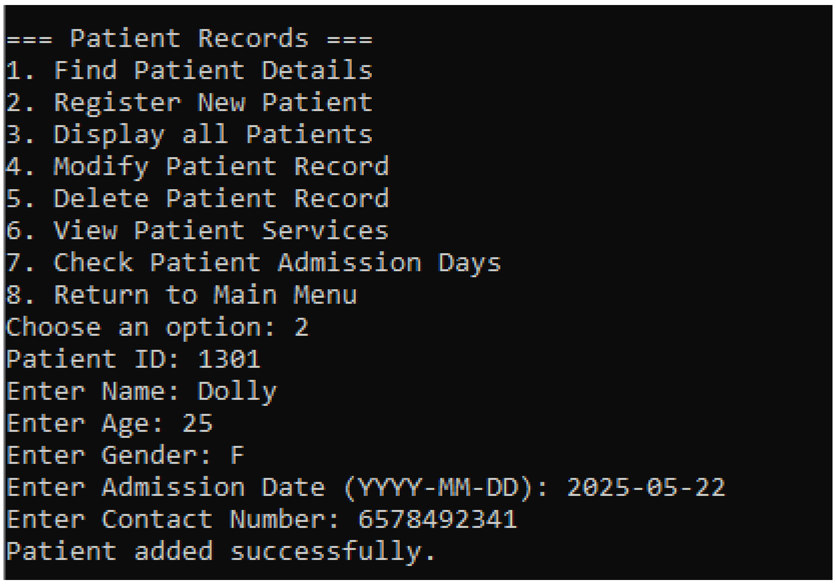


```
=== Billing Records ===
1. Create New Bill
2. View All Billing Records
3. Modify Bill Details
4. Delete Bill Entry
5. Calculate Total Charges
6. Print/Generate Invoice
7. Return to Main Menu
Select an option: _
```

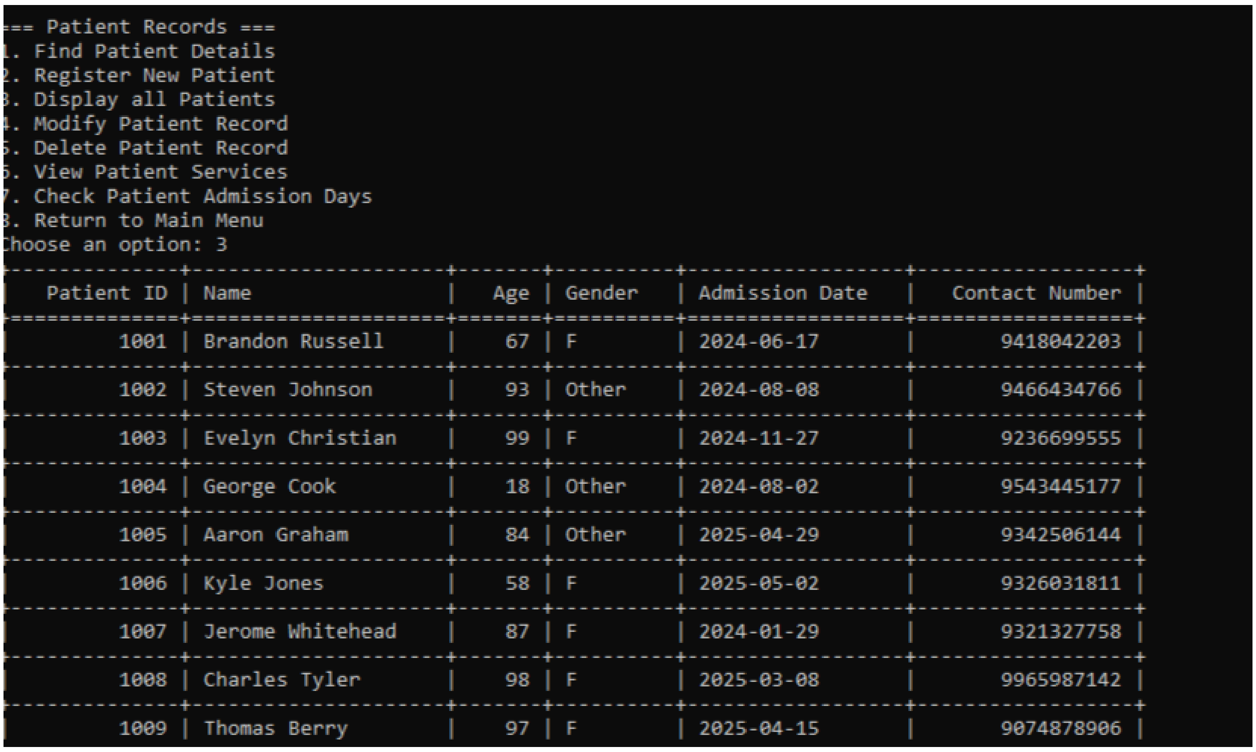
Patient Records



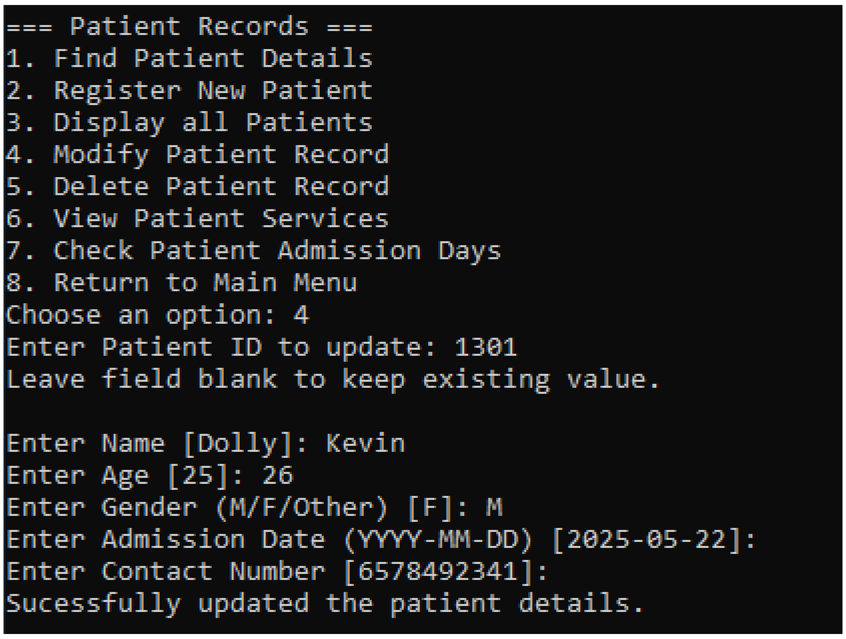
```
=== Patient Records ===
1. Find Patient Details
2. Register New Patient
3. Display all Patients
4. Modify Patient Record
5. Delete Patient Record
6. View Patient Services
7. Check Patient Admission Days
8. Return to Main Menu
Choose an option: 1
Enter part or full patient name: Austin
Patient_ID | Name | Age | Gender | Admission Date | Contact Number
1122 | Austin Middleton | 36 | M | 2024-03-05 | 9127838883
```



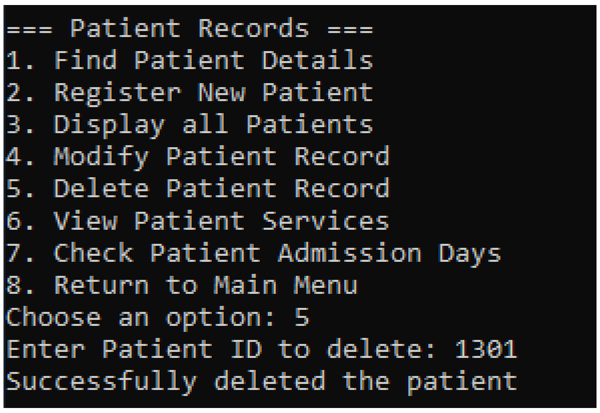
```
=== Patient Records ===
1. Find Patient Details
2. Register New Patient
3. Display all Patients
4. Modify Patient Record
5. Delete Patient Record
6. View Patient Services
7. Check Patient Admission Days
8. Return to Main Menu
Choose an option: 2
Patient ID: 1301
Enter Name: Dolly
Enter Age: 25
Enter Gender: F
Enter Admission Date (YYYY-MM-DD): 2025-05-22
Enter Contact Number: 6578492341
Patient added successfully.
```



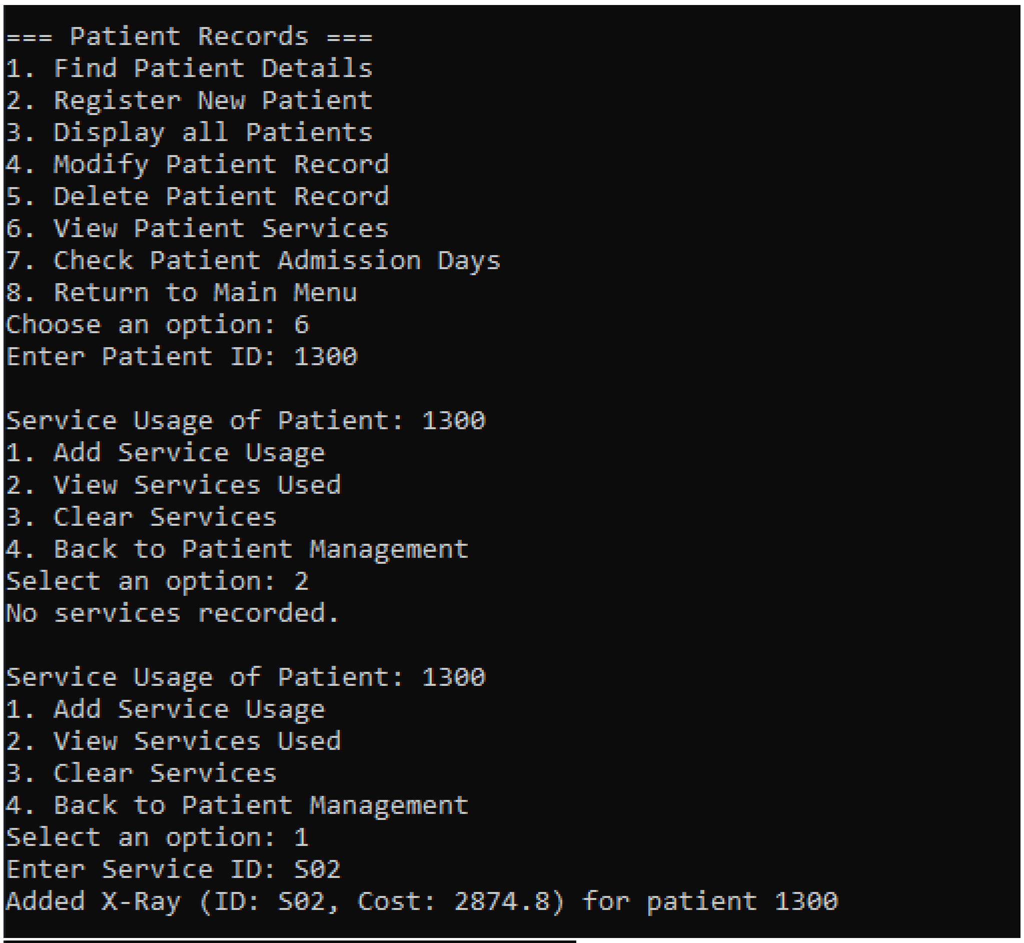
```
== Patient Records ===
l. Find Patient Details
2. Register New Patient
3. Display all Patients
4. Modify Patient Record
5. Delete Patient Record
5. View Patient Services
7. Check Patient Admission Days
3. Return to Main Menu
hoose an option: 3
-------------+--------------------+-------+---------+-----------------+-----------------+
 Patient ID | Name               |  Age | Gender  | Admission Date  |  Contact Number |
=============+====================+=======+=========+=================+=================+
       1001 | Brandon Russell    |   67 | F       | 2024-06-17      |      9418042203 |
-------------+--------------------+-------+---------+-----------------+-----------------+
       1002 | Steven Johnson     |   93 | Other   | 2024-08-08      |      9466434766 |
-------------+--------------------+-------+---------+-----------------+-----------------+
       1003 | Evelyn Christian   |   99 | F       | 2024-11-27      |      9236699555 |
-------------+--------------------+-------+---------+-----------------+-----------------+
       1004 | George Cook        |   18 | Other   | 2024-08-02      |      9543445177 |
-------------+--------------------+-------+---------+-----------------+-----------------+
       1005 | Aaron Graham       |   84 | Other   | 2025-04-29      |      9342506144 |
-------------+--------------------+-------+---------+-----------------+-----------------+
       1006 | Kyle Jones         |   58 | F       | 2025-05-02      |      9326031811 |
-------------+--------------------+-------+---------+-----------------+-----------------+
       1007 | Jerome Whitehead   |   87 | F       | 2024-01-29      |      9321327758 |
-------------+--------------------+-------+---------+-----------------+-----------------+
       1008 | Charles Tyler      |   98 | F       | 2025-03-08      |      9965987142 |
-------------+--------------------+-------+---------+-----------------+-----------------+
       1009 | Thomas Berry       |   97 | F       | 2025-04-15      |      9074878906 |
```



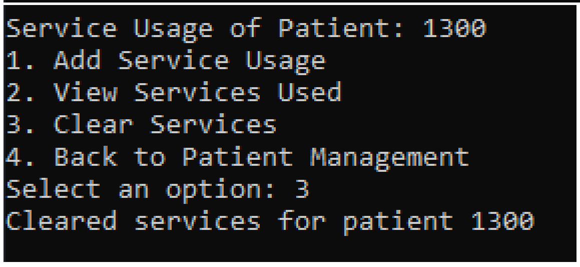
```
=== Patient Records ===
1. Find Patient Details
2. Register New Patient
3. Display all Patients
4. Modify Patient Record
5. Delete Patient Record
6. View Patient Services
7. Check Patient Admission Days
8. Return to Main Menu
Choose an option: 4
Enter Patient ID to update: 1301
Leave field blank to keep existing value.

Enter Name [Dolly]: Kevin
Enter Age [25]: 26
Enter Gender (M/F/Other) [F]: M
Enter Admission Date (YYYY-MM-DD) [2025-05-22]:
Enter Contact Number [6578492341]:
Sucessfully updated the patient details.
```



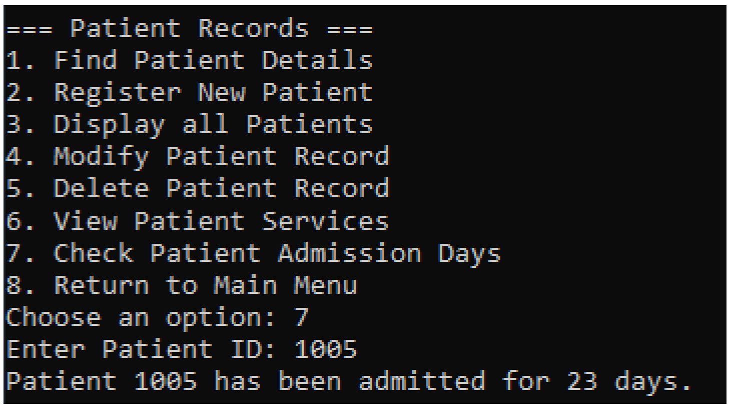
```
=== Patient Records ===
1. Find Patient Details
2. Register New Patient
3. Display all Patients
4. Modify Patient Record
5. Delete Patient Record
6. View Patient Services
7. Check Patient Admission Days
8. Return to Main Menu
Choose an option: 5
Enter Patient ID to delete: 1301
Successfully deleted the patient
```



```
=== Patient Records ===
1. Find Patient Details
2. Register New Patient
3. Display all Patients
4. Modify Patient Record
5. Delete Patient Record
6. View Patient Services
7. Check Patient Admission Days
8. Return to Main Menu
Choose an option: 6
Enter Patient ID: 1300

Service Usage of Patient: 1300
1. Add Service Usage
2. View Services Used
3. Clear Services
4. Back to Patient Management
Select an option: 2
No services recorded.

Service Usage of Patient: 1300
1. Add Service Usage
2. View Services Used
3. Clear Services
4. Back to Patient Management
Select an option: 1
Enter Service ID: S02
Added X-Ray (ID: S02, Cost: 2874.8) for patient 1300
```

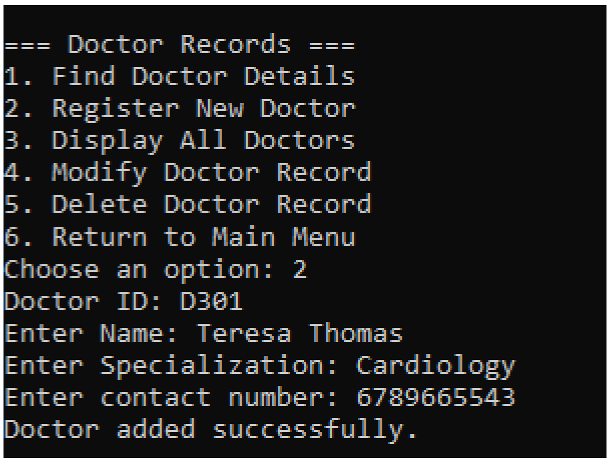


```
Service Usage of Patient: 1300
1. Add Service Usage
2. View Services Used
3. Clear Services
4. Back to Patient Management
Select an option: 3
Cleared services for patient 1300
```

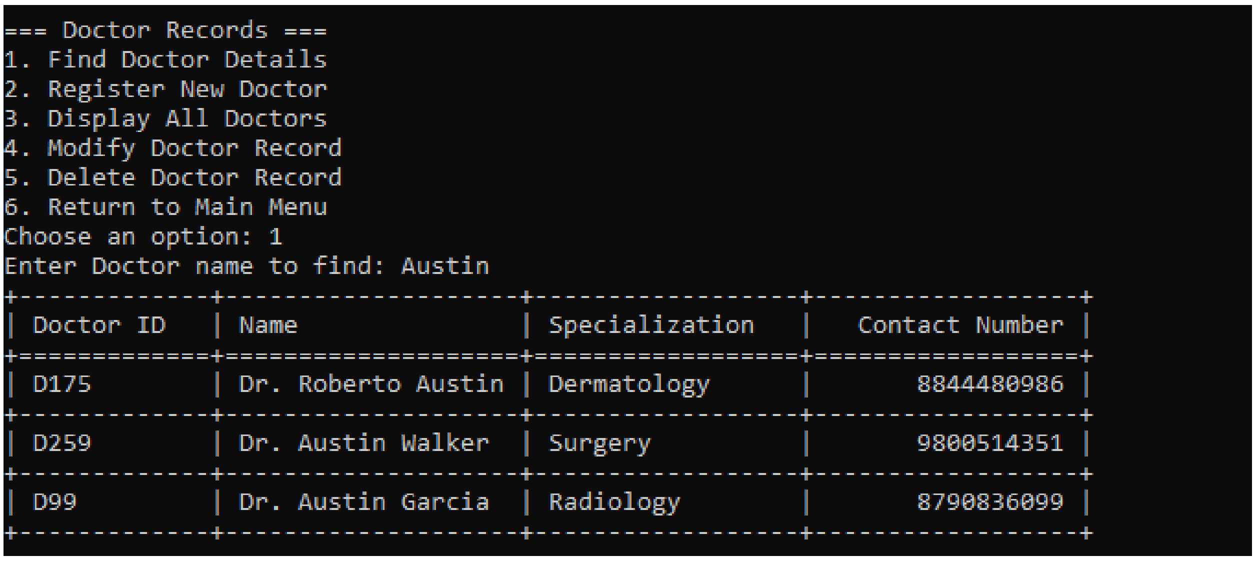


```
=== Patient Records ===
1. Find Patient Details
2. Register New Patient
3. Display all Patients
4. Modify Patient Record
5. Delete Patient Record
6. View Patient Services
7. Check Patient Admission Days
8. Return to Main Menu
Choose an option: 7
Enter Patient ID: 1005
Patient 1005 has been admitted for 23 days.
```

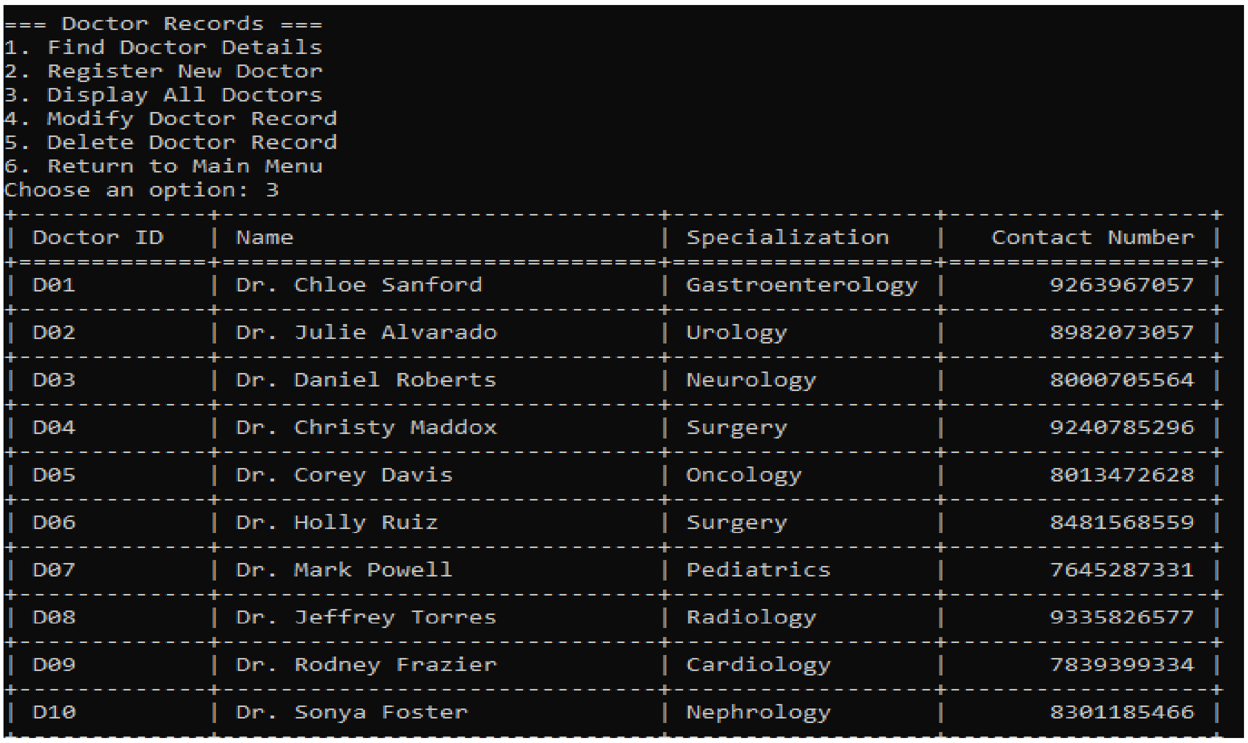
Doctor Records



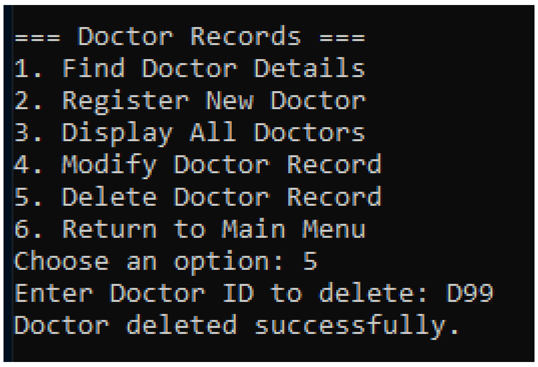
```
=== Doctor Records ===
1. Find Doctor Details
2. Register New Doctor
3. Display All Doctors
4. Modify Doctor Record
5. Delete Doctor Record
6. Return to Main Menu
Choose an option: 2
Doctor ID: D301
Enter Name: Teresa Thomas
Enter Specialization: Cardiology
Enter contact number: 6789665543
Doctor added successfully.
```



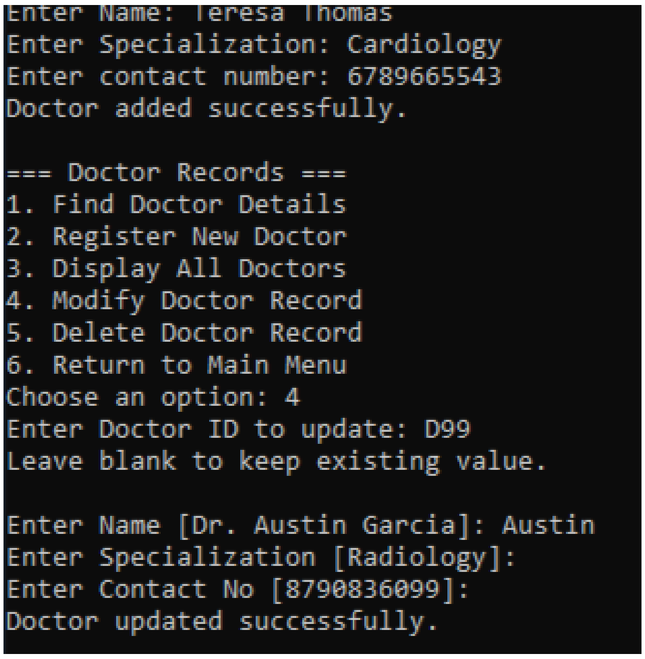
```
=== Doctor Records ===
1. Find Doctor Details
2. Register New Doctor
3. Display All Doctors
4. Modify Doctor Record
5. Delete Doctor Record
6. Return to Main Menu
Choose an option: 1
Enter Doctor name to find: Austin
+-------------+--------------------+------------------+------------------+
| Doctor ID   | Name               | Specialization   |   Contact Number |
+=============+====================+==================+==================+
| D175        | Dr. Roberto Austin | Dermatology      |       8844480986 |
+-------------+--------------------+------------------+------------------+
| D259        | Dr. Austin Walker  | Surgery          |       9800514351 |
+-------------+--------------------+------------------+------------------+
| D99         | Dr. Austin Garcia  | Radiology        |       8790836099 |
+-------------+--------------------+------------------+------------------+
```



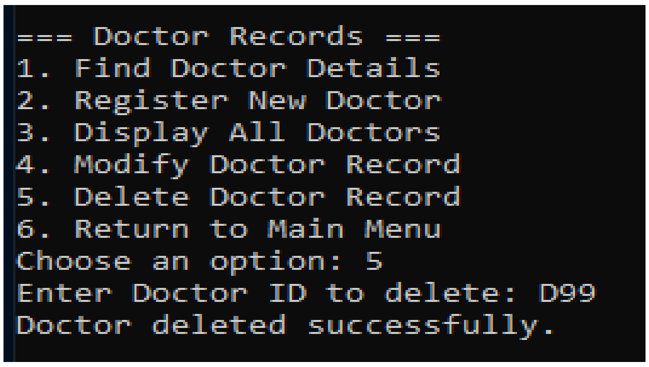
```
=== Doctor Records ===
1. Find Doctor Details
2. Register New Doctor
3. Display All Doctors
4. Modify Doctor Record
5. Delete Doctor Record
6. Return to Main Menu
Choose an option: 3
+-------------+------------------------------+------------------+------------------+
| Doctor ID   | Name                         | Specialization   |   Contact Number |
+=============+==============================+==================+==================+
| D01         | Dr. Chloe Sanford            | Gastroenterology |       9263967057 |
+-------------+------------------------------+------------------+------------------+
| D02         | Dr. Julie Alvarado           | Urology          |       8982073057 |
+-------------+------------------------------+------------------+------------------+
| D03         | Dr. Daniel Roberts           | Neurology        |       8000705564 |
+-------------+------------------------------+------------------+------------------+
| D04         | Dr. Christy Maddox           | Surgery          |       9240785296 |
+-------------+------------------------------+------------------+------------------+
| D05         | Dr. Corey Davis              | Oncology         |       8013472628 |
+-------------+------------------------------+------------------+------------------+
| D06         | Dr. Holly Ruiz               | Surgery          |       8481568559 |
+-------------+------------------------------+------------------+------------------+
| D07         | Dr. Mark Powell              | Pediatrics       |       7645287331 |
+-------------+------------------------------+------------------+------------------+
| D08         | Dr. Jeffrey Torres           | Radiology        |       9335826577 |
+-------------+------------------------------+------------------+------------------+
| D09         | Dr. Rodney Frazier           | Cardiology       |       7839399334 |
+-------------+------------------------------+------------------+------------------+
| D10         | Dr. Sonya Foster             | Nephrology       |       8301185466 |
```



```
=== Doctor Records ===
1. Find Doctor Details
2. Register New Doctor
3. Display All Doctors
4. Modify Doctor Record
5. Delete Doctor Record
6. Return to Main Menu
Choose an option: 5
Enter Doctor ID to delete: D99
Doctor deleted successfully.
```

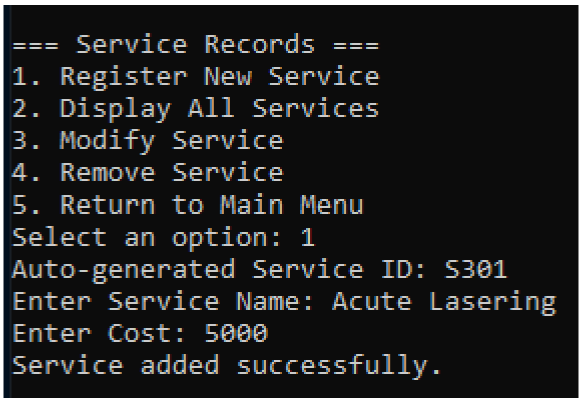


```
Enter Name: Teresa Thomas
Enter Specialization: Cardiology
Enter contact number: 6789665543
Doctor added successfully.

=== Doctor Records ===
1. Find Doctor Details
2. Register New Doctor
3. Display All Doctors
4. Modify Doctor Record
5. Delete Doctor Record
6. Return to Main Menu
Choose an option: 4
Enter Doctor ID to update: D99
Leave blank to keep existing value.

Enter Name [Dr. Austin Garcia]: Austin
Enter Specialization [Radiology]:
Enter Contact No [8790836099]:
Doctor updated successfully.
```

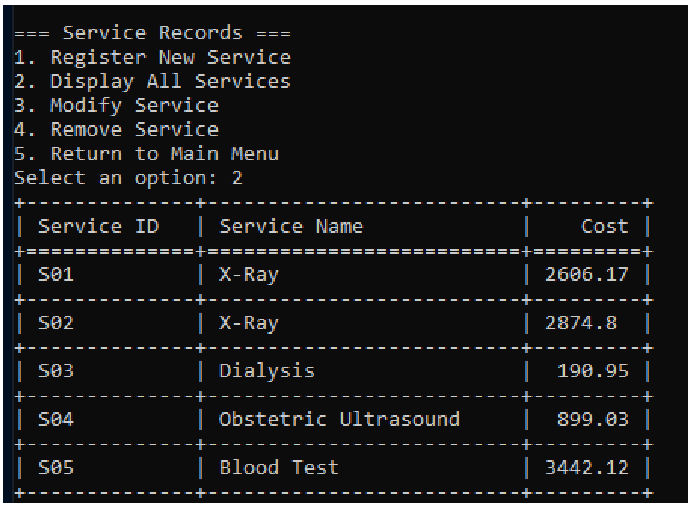


```
=== Doctor Records ===
1. Find Doctor Details
2. Register New Doctor
3. Display All Doctors
4. Modify Doctor Record
5. Delete Doctor Record
6. Return to Main Menu
Choose an option: 5
Enter Doctor ID to delete: D99
Doctor deleted successfully.
```

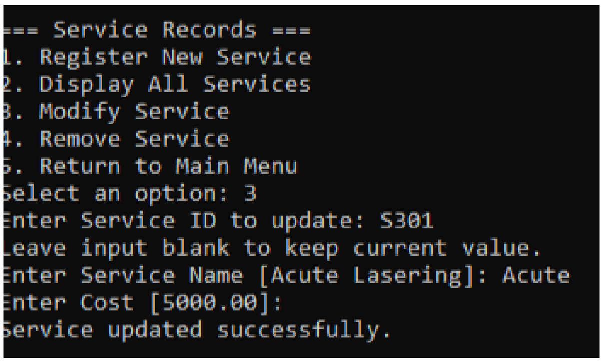
Service Records



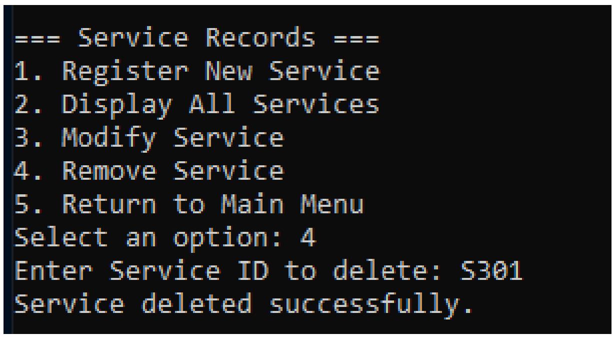
```
=== Service Records ===
1. Register New Service
2. Display All Services
3. Modify Service
4. Remove Service
5. Return to Main Menu
Select an option: 1
Auto-generated Service ID: S301
Enter Service Name: Acute Lasering
Enter Cost: 5000
Service added successfully.
```



```
=== Service Records ===
1. Register New Service
2. Display All Services
3. Modify Service
4. Remove Service
5. Return to Main Menu
Select an option: 2
+-------------+-------------------------+---------+
| Service ID  | Service Name            |    Cost |
+=============+=========================+=========+
| S01         | X-Ray                   | 2606.17 |
+-------------+-------------------------+---------+
| S02         | X-Ray                   | 2874.8  |
+-------------+-------------------------+---------+
| S03         | Dialysis                |  190.95 |
+-------------+-------------------------+---------+
| S04         | Obstetric Ultrasound    |  899.03 |
+-------------+-------------------------+---------+
| S05         | Blood Test              | 3442.12 |
+-------------+-------------------------+---------+
```

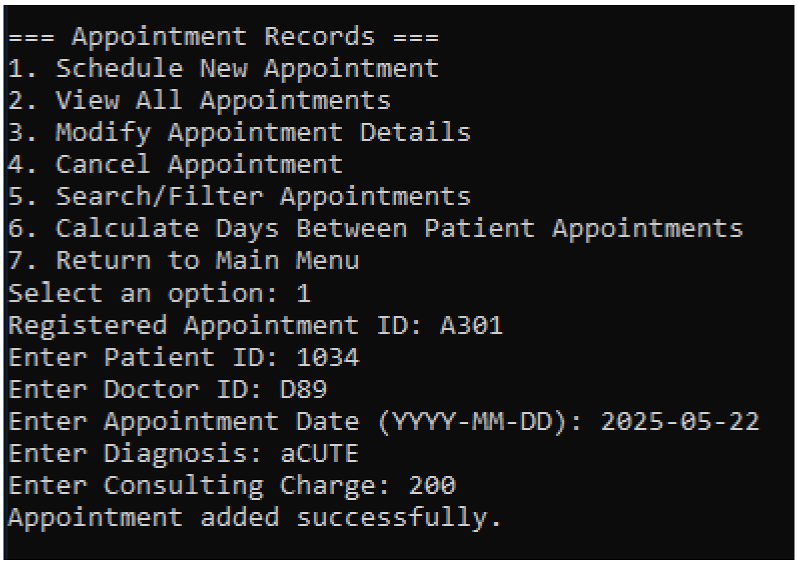


```
== Service Records ===
. Register New Service
. Display All Services
. Modify Service
. Remove Service
. Return to Main Menu
elect an option: 3
nter Service ID to update: S301
eave input blank to keep current value.
nter Service Name [Acute Lasering]: Acute
nter Cost [5000.00]:
ervice updated successfully.
```

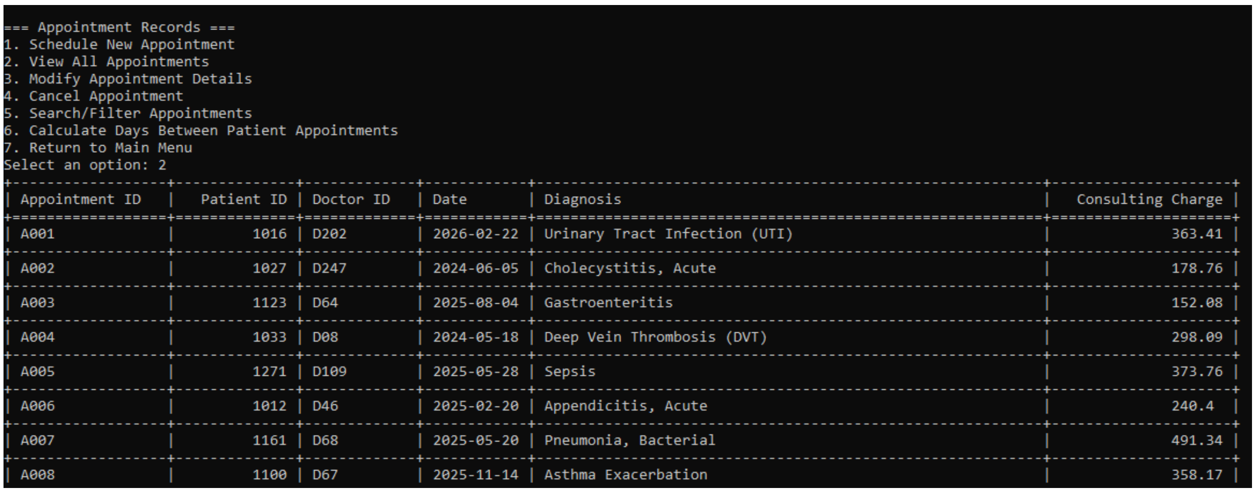


```
=== Service Records ===
1. Register New Service
2. Display All Services
3. Modify Service
4. Remove Service
5. Return to Main Menu
Select an option: 4
Enter Service ID to delete: S301
Service deleted successfully.
```

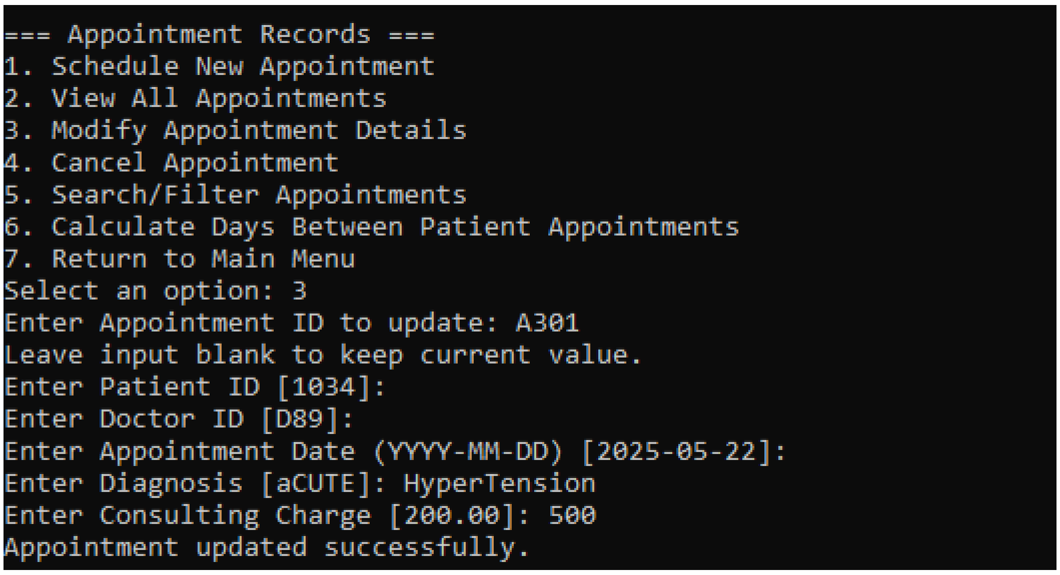
Appointment Records



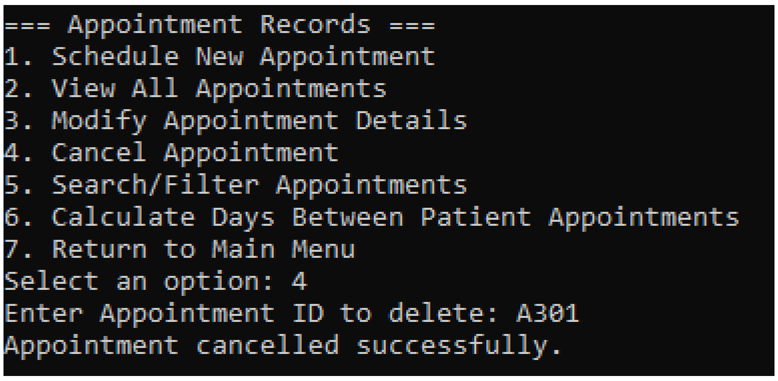
```
=== Appointment Records ===
1. Schedule New Appointment
2. View All Appointments
3. Modify Appointment Details
4. Cancel Appointment
5. Search/Filter Appointments
6. Calculate Days Between Patient Appointments
7. Return to Main Menu
Select an option: 1
Registered Appointment ID: A301
Enter Patient ID: 1034
Enter Doctor ID: D89
Enter Appointment Date (YYYY-MM-DD): 2025-05-22
Enter Diagnosis: aCUTE
Enter Consulting Charge: 200
Appointment added successfully.
```



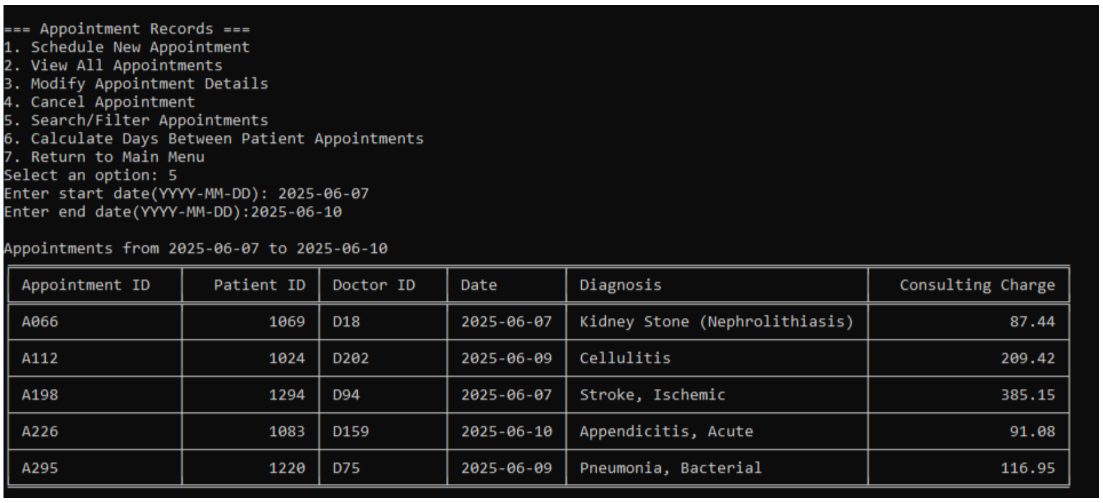
```
=== Appointment Records ===
1. Schedule New Appointment
2. View All Appointments
3. Modify Appointment Details
4. Cancel Appointment
5. Search/Filter Appointments
6. Calculate Days Between Patient Appointments
7. Return to Main Menu
Select an option: 2
+-------------------+---------------+-------------+------------+------------------------------------------------------------+---------------------+
| Appointment ID    |    Patient ID | Doctor ID   | Date       | Diagnosis                                                  |   Consulting Charge |
+===================+===============+=============+============+============================================================+=====================+
| A001              |          1016 | D202        | 2026-02-22 | Urinary Tract Infection (UTI)                              |              363.41 |
+-------------------+---------------+-------------+------------+------------------------------------------------------------+---------------------+
| A002              |          1027 | D247        | 2024-06-05 | Cholecystitis, Acute                                       |              178.76 |
+-------------------+---------------+-------------+------------+------------------------------------------------------------+---------------------+
| A003              |          1123 | D64         | 2025-08-04 | Gastroenteritis                                            |              152.08 |
+-------------------+---------------+-------------+------------+------------------------------------------------------------+---------------------+
| A004              |          1033 | D08         | 2024-05-18 | Deep Vein Thrombosis (DVT)                                 |              298.09 |
+-------------------+---------------+-------------+------------+------------------------------------------------------------+---------------------+
| A005              |          1271 | D109        | 2025-05-28 | Sepsis                                                     |              373.76 |
+-------------------+---------------+-------------+------------+------------------------------------------------------------+---------------------+
| A006              |          1012 | D46         | 2025-02-20 | Appendicitis, Acute                                        |              240.4  |
+-------------------+---------------+-------------+------------+------------------------------------------------------------+---------------------+
| A007              |          1161 | D68         | 2025-05-20 | Pneumonia, Bacterial                                       |              491.34 |
+-------------------+---------------+-------------+------------+------------------------------------------------------------+---------------------+
| A008              |          1100 | D67         | 2025-11-14 | Asthma Exacerbation                                        |              358.17 |
```



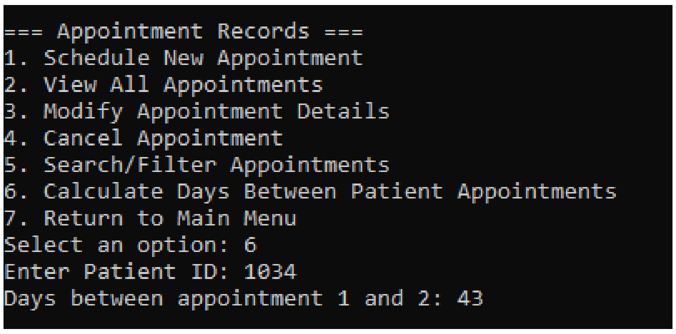
```
=== Appointment Records ===
1. Schedule New Appointment
2. View All Appointments
3. Modify Appointment Details
4. Cancel Appointment
5. Search/Filter Appointments
6. Calculate Days Between Patient Appointments
7. Return to Main Menu
Select an option: 3
Enter Appointment ID to update: A301
Leave input blank to keep current value.
Enter Patient ID [1034]:
Enter Doctor ID [D89]:
Enter Appointment Date (YYYY-MM-DD) [2025-05-22]:
Enter Diagnosis [aCUTE]: HyperTension
Enter Consulting Charge [200.00]: 500
Appointment updated successfully.
```



```
=== Appointment Records ===
1. Schedule New Appointment
2. View All Appointments
3. Modify Appointment Details
4. Cancel Appointment
5. Search/Filter Appointments
6. Calculate Days Between Patient Appointments
7. Return to Main Menu
Select an option: 4
Enter Appointment ID to delete: A301
Appointment cancelled successfully.
```

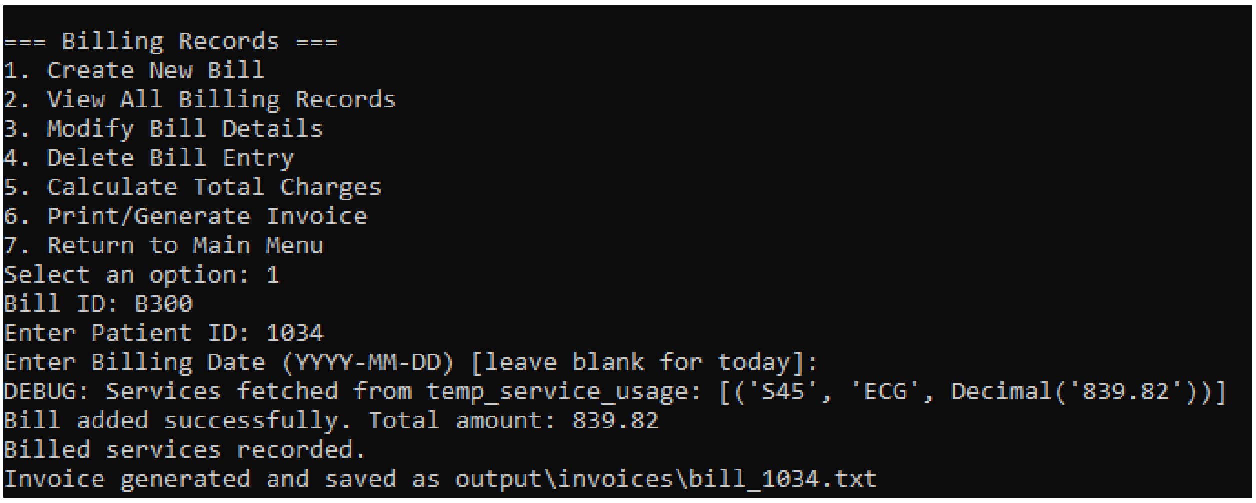


```
=== Appointment Records ===
1. Schedule New Appointment
2. View All Appointments
3. Modify Appointment Details
4. Cancel Appointment
5. Search/Filter Appointments
6. Calculate Days Between Patient Appointments
7. Return to Main Menu
Select an option: 5
Enter start date(YYYY-MM-DD): 2025-06-07
Enter end date(YYYY-MM-DD):2025-06-10

Appointments from 2025-06-07 to 2025-06-10
| Appointment ID | Patient ID | Doctor ID | Date       | Diagnosis                      | Consulting Charge |
| A066           |       1069 | D18       | 2025-06-07 | Kidney Stone (Nephrolithiasis) |             87.44 |
| A112           |       1024 | D202      | 2025-06-09 | Cellulitis                     |            209.42 |
| A198           |       1294 | D94       | 2025-06-07 | Stroke, Ischemic               |            385.15 |
| A226           |       1083 | D159      | 2025-06-10 | Appendicitis, Acute            |             91.08 |
| A295           |       1220 | D75       | 2025-06-09 | Pneumonia, Bacterial           |            116.95 |
```

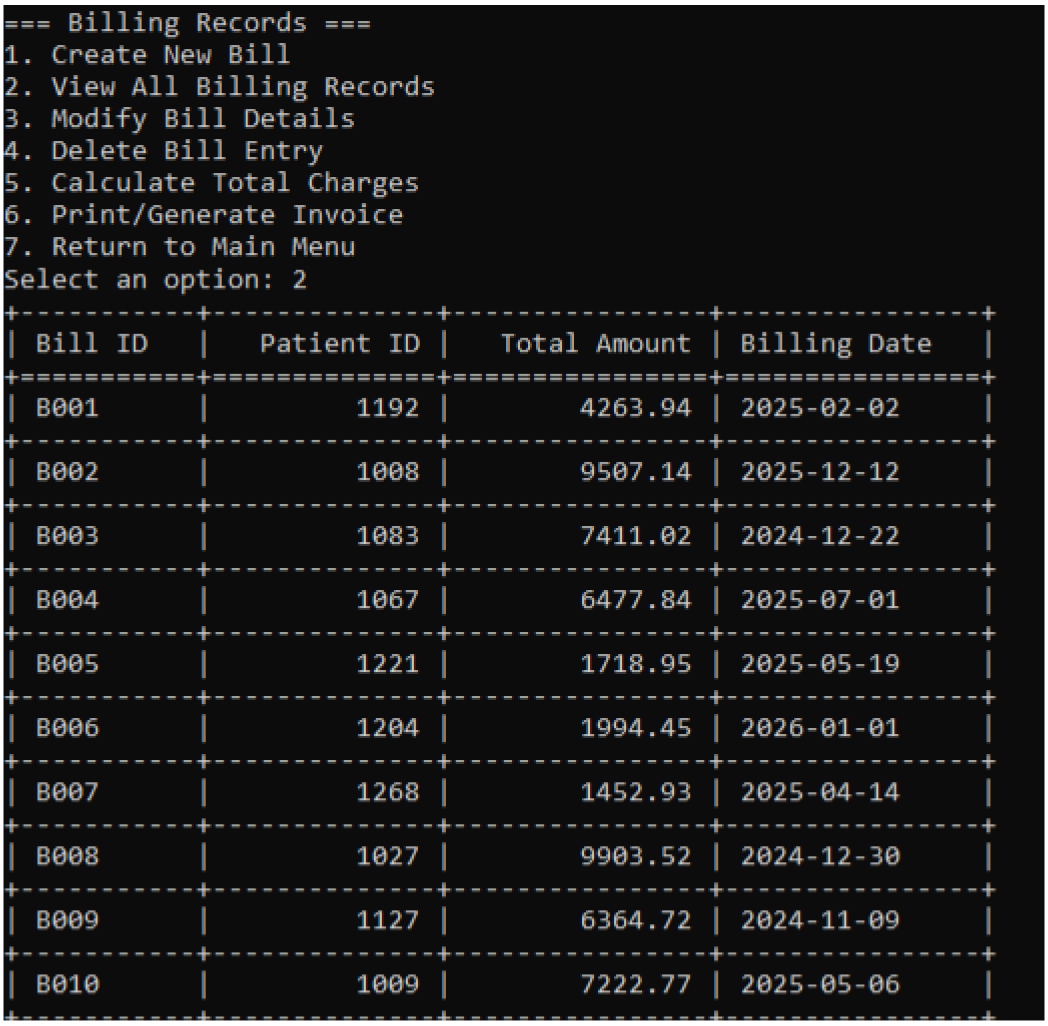


```
=== Appointment Records ===
1. Schedule New Appointment
2. View All Appointments
3. Modify Appointment Details
4. Cancel Appointment
5. Search/Filter Appointments
6. Calculate Days Between Patient Appointments
7. Return to Main Menu
Select an option: 6
Enter Patient ID: 1034
Days between appointment 1 and 2: 43
```

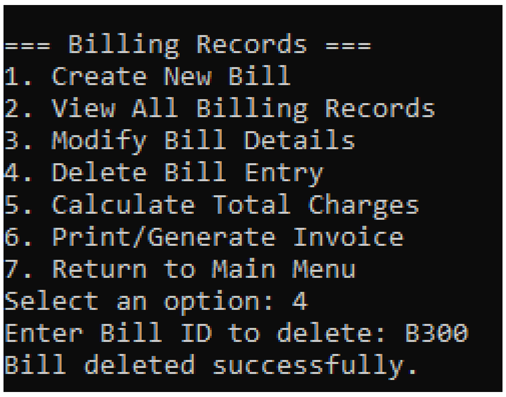
Billing Records



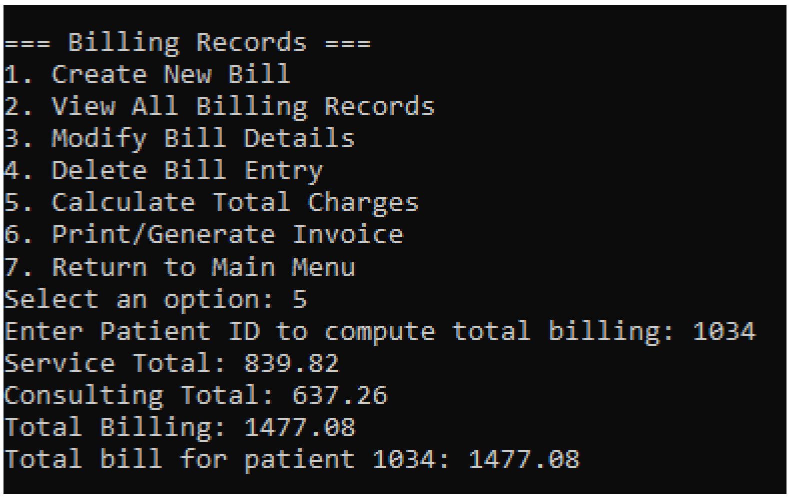
```
=== Billing Records ===
1. Create New Bill
2. View All Billing Records
3. Modify Bill Details
4. Delete Bill Entry
5. Calculate Total Charges
6. Print/Generate Invoice
7. Return to Main Menu
Select an option: 1
Bill ID: B300
Enter Patient ID: 1034
Enter Billing Date (YYYY-MM-DD) [leave blank for today]:
DEBUG: Services fetched from temp_service_usage: [('S45', 'ECG', Decimal('839.82'))]
Bill added successfully. Total amount: 839.82
Billed services recorded.
Invoice generated and saved as output\invoices\bill_1034.txt
```



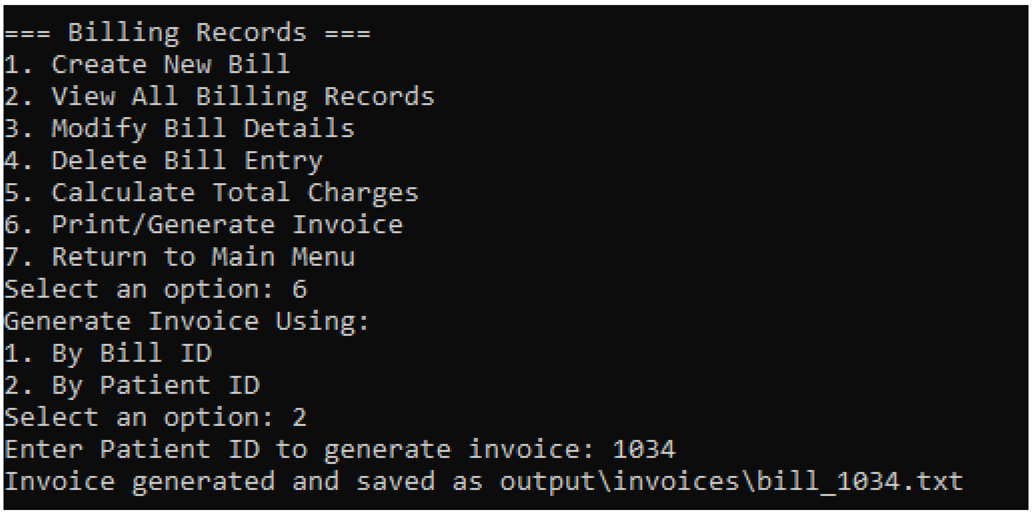
```
=== Billing Records ===
1. Create New Bill
2. View All Billing Records
3. Modify Bill Details
4. Delete Bill Entry
5. Calculate Total Charges
6. Print/Generate Invoice
7. Return to Main Menu
Select an option: 2
+-----------+--------------+----------------+----------------+
| Bill ID   |   Patient ID |   Total Amount | Billing Date   |
+===========+==============+================+================+
| B001      |         1192 |        4263.94 | 2025-02-02     |
+-----------+--------------+----------------+----------------+
| B002      |         1008 |        9507.14 | 2025-12-12     |
+-----------+--------------+----------------+----------------+
| B003      |         1083 |        7411.02 | 2024-12-22     |
+-----------+--------------+----------------+----------------+
| B004      |         1067 |        6477.84 | 2025-07-01     |
+-----------+--------------+----------------+----------------+
| B005      |         1221 |        1718.95 | 2025-05-19     |
+-----------+--------------+----------------+----------------+
| B006      |         1204 |        1994.45 | 2026-01-01     |
+-----------+--------------+----------------+----------------+
| B007      |         1268 |        1452.93 | 2025-04-14     |
+-----------+--------------+----------------+----------------+
| B008      |         1027 |        9903.52 | 2024-12-30     |
+-----------+--------------+----------------+----------------+
| B009      |         1127 |        6364.72 | 2024-11-09     |
+-----------+--------------+----------------+----------------+
| B010      |         1009 |        7222.77 | 2025-05-06     |
+-----------+--------------+----------------+----------------+
```



```
=== Billing Records ===
1. Create New Bill
2. View All Billing Records
3. Modify Bill Details
4. Delete Bill Entry
5. Calculate Total Charges
6. Print/Generate Invoice
7. Return to Main Menu
Select an option: 4
Enter Bill ID to delete: B300
Bill deleted successfully.
```

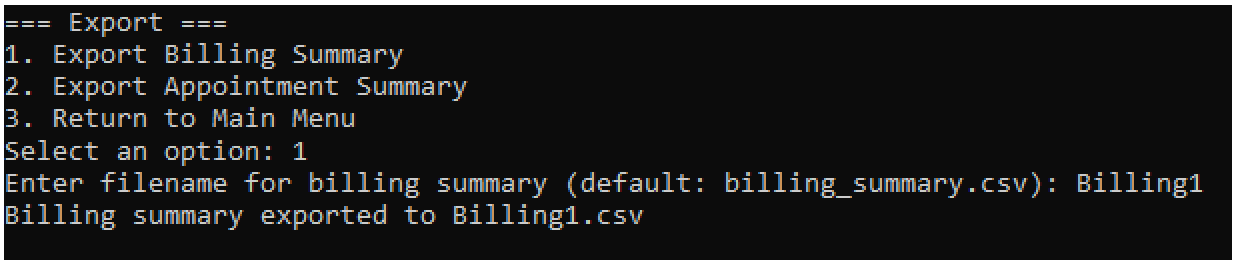


```
=== Billing Records ===
1. Create New Bill
2. View All Billing Records
3. Modify Bill Details
4. Delete Bill Entry
5. Calculate Total Charges
6. Print/Generate Invoice
7. Return to Main Menu
Select an option: 5
Enter Patient ID to compute total billing: 1034
Service Total: 839.82
Consulting Total: 637.26
Total Billing: 1477.08
Total bill for patient 1034: 1477.08
```

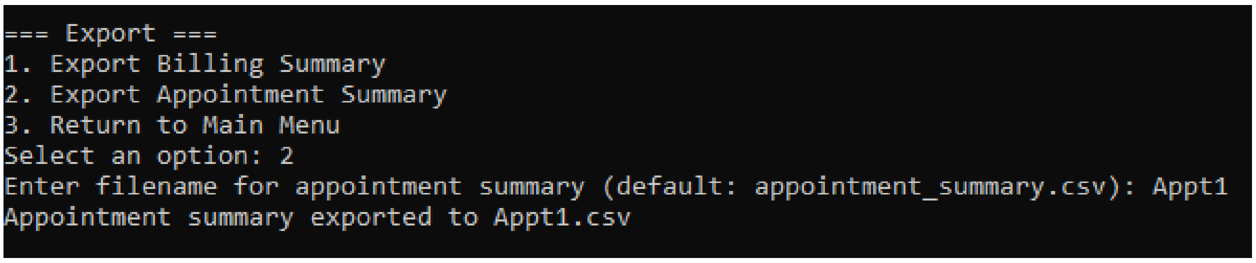


```
=== Billing Records ===
1. Create New Bill
2. View All Billing Records
3. Modify Bill Details
4. Delete Bill Entry
5. Calculate Total Charges
6. Print/Generate Invoice
7. Return to Main Menu
Select an option: 6
Generate Invoice Using:
1. By Bill ID
2. By Patient ID
Select an option: 2
Enter Patient ID to generate invoice: 1034
Invoice generated and saved as output\invoices\bill_1034.txt
```

Export Records



```
=== Export ===
1. Export Billing Summary
2. Export Appointment Summary
3. Return to Main Menu
Select an option: 1
Enter filename for billing summary (default: billing_summary.csv): Billing1
Billing summary exported to Billing1.csv
```



```
=== Export ===
1. Export Billing Summary
2. Export Appointment Summary
3. Return to Main Menu
Select an option: 2
Enter filename for appointment summary (default: appointment_summary.csv): Appt1
Appointment summary exported to Appt1.csv
```