HOSPITAL PATIENT RECORD & BILLING SYSTEM

DOCUMENTATION

-By

Kadali Harshavardhan

2633272

Table of Contents

HOSPITAL PATIENT RECORD & BILLING SYSTEM

1. SYSTEM OVERVIEW

The Hospital Patient Record & Billing System is designed to streamline the management of patient appointments and billing in a healthcare setting. It leverages a MySQL database for persistent storage and provides robust validation, reporting, and error handling to ensure data integrity and operational efficiency.

2. ARCHITECTURE

- **Programming Language**: Python
- **Database**: MySQL
- **Modularity**: The system is organized into separate modules for appointment and billing management.
- **Database Connectivity**: All database operations are performed using a get_connection() function from a shared db_config.py module.

3. DEPENDENCIES

- Python 3.x
- MySQL database
- mysql-connector-python for database connectivity
- CSV module for export functionality

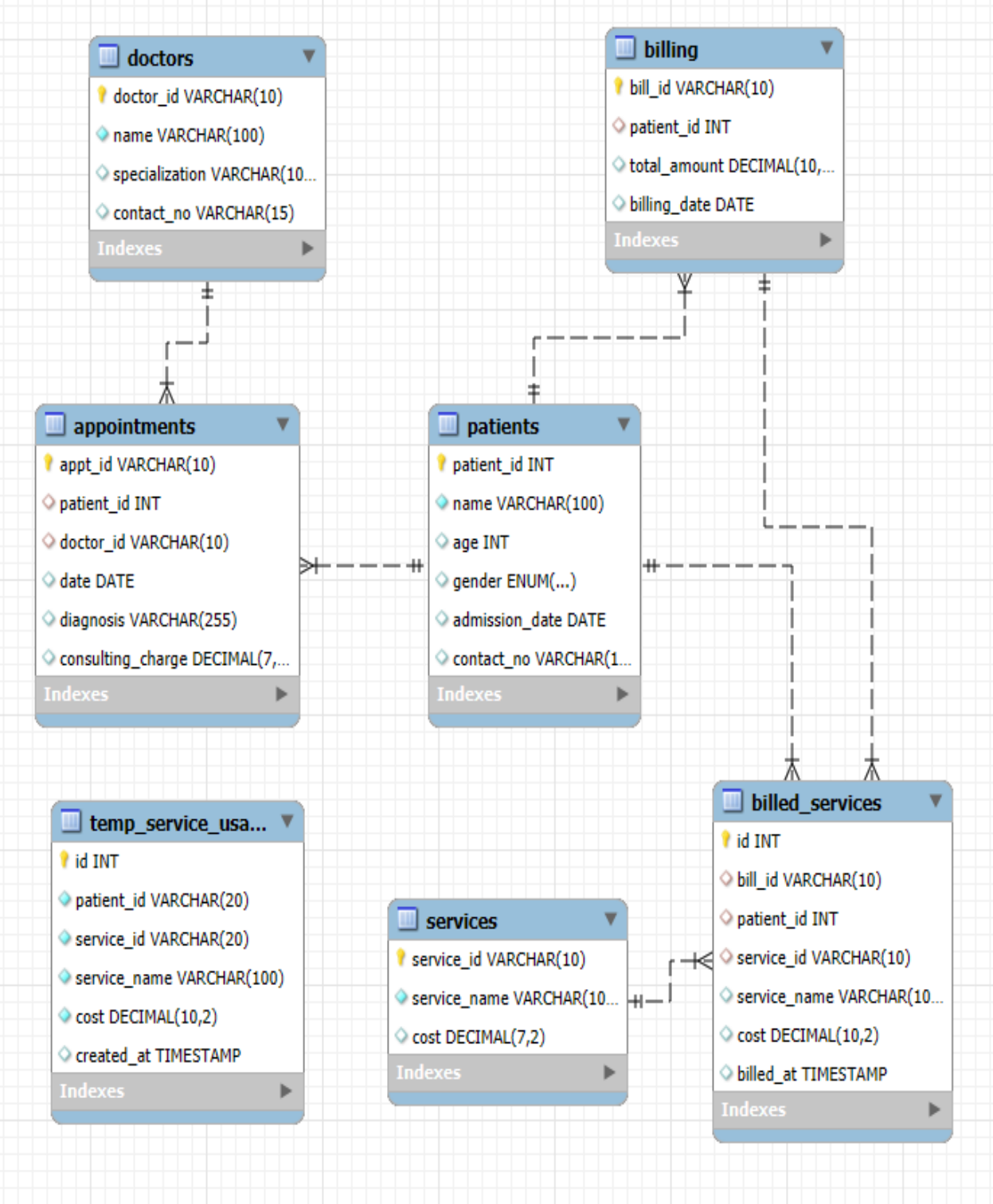
Notes

- The db_config.py file must be present and correctly configured for database access.
- The ServiceUsageDB class (used in billing) is expected to be defined in a separate service.py module.

4. DATABASE SCHEMA OVERVIEW

The system expects the following tables (not exhaustive):

- appointments: Stores appointment details (ID, patient, doctor, date, diagnosis, charge).
- patients: Patient master records.
- doctors: Doctor master records.
- billing: Bill master records (ID, patient, amount, date).
- services: Hospital services available.
- billed_services: Services billed under each bill.
- temp_service_usage: Temporary records of services used by patients before billing.


doctors
doctor_id VARCHAR(10)
name VARCHAR(100)
specialization VARCHAR(10...
contact_no VARCHAR(15)
Indexes
billing
bill_id VARCHAR(10)
patient_id INT
total_amount DECIMAL(10,...
billing_date DATE
Indexes
appointments
appt_id VARCHAR(10)
patient_id INT
doctor_id VARCHAR(10)
date DATE
diagnosis VARCHAR(255)
consulting_charge DECIMAL(7,...
Indexes
patients
patient_id INT
name VARCHAR(100)
age INT
gender ENUM(...)
admission_date DATE
contact_no VARCHAR(1...
Indexes
temp_service_usa...
id INT
patient_id VARCHAR(20)
service_id VARCHAR(20)
service_name VARCHAR(100)
cost DECIMAL(10,2)
created_at TIMESTAMP
Indexes
services
service_id VARCHAR(10)
service_name VARCHAR(10...
cost DECIMAL(7,2)
Indexes
billed_services
id INT
bill_id VARCHAR(10)
patient_id INT
service_id VARCHAR(10)
service_name VARCHAR(10...
cost DECIMAL(10,2)
billed_at TIMESTAMP
Indexes

5. SETUP INSTRUCTIONS

Follow the steps below to install and run the Hospital Patient Record & Billing System on your local machine.

5.1. Prerequisites

Ensure the following are installed on your system:

- **Python 3.7+**
- **MySQL Server**
- **pip** (Python package manager)

5.2. Clone the Repository

If you are using version control (e.g., GitHub), clone the project:

```
git clone https://github.com/your-username/hospital-mgmt.git
cd hospital-mgmt
```

5.3. Install Required Python Packages

Install all dependencies listed in the requirements.txt file:

```
pip install -r requirements.txt
```

Required packages include:

- mysql-connector-python
- tabulate

5.4. Configure the Database Connection

Open the db_config.py file and update it with your MySQL credentials:

```
# db_config.py
import mysql.connector
def get_connection():
    return mysql.connector.connect(
        host="localhost",
        user="your_mysql_user",
        password="your_mysql_password",
        database="your_database_name"
    )
```

5.5. Create the Database & Tables

Start your MySQL server, and create the required database and tables. Use the provided schema or SQL script, or manually create these tables:

- patients
- doctors
- appointments
- services
- billing
- billed_services
- temp_service_usage

Optionally, create indexes or foreign key constraints as needed.

5.6. Run the Application

Start the application using the main script:

```
python hospital_main.py
```

Follow the on-screen instructions to manage patients, appointments, billing, and services.

Notes

- Ensure MySQL is running before launching the app.
- Use the tabulate library for better output formatting.
- Regularly export your data for backup and reporting.

6. Main Modules

6.1 Patient Records (patient.py)

The patient.py module is responsible for managing patient records in the Hospital Appointment & Billing Management System. It provides functionality to add, update, delete, and retrieve patient information, ensuring data integrity and seamless integration with other modules such as appointments and billing.

Class: Patient

Purpose

Encapsulates all attributes and operations related to a patient's record.

Attributes

- patient_id: Unique identifier for the patient (string or integer, usually auto generated).
- name: Full name of the patient (string).
- age: Age of the patient (integer).
- gender: Gender of the patient (string, e.g., 'Male', 'Female', 'Other').
- address: Residential address (string).
- phone: Contact phone number (string).
- email: Email address (string, optional).

Methods

1. add ()

- Purpose: Adds a new patient record to the database.
- Validation:
    - Checks for valid name, age, gender, and contact details.
    - Ensures no duplicate patient ID or phone number.

- Database Operation: Inserts a new row into the patient's table.

2. update ()

- Purpose: Updates details for an existing patient.
- Validation:
  - Checks if the patient exists.
  - Validates updated fields.
- Database Operation: Updates the row in the patients table where patient_id matches.

3. delete(patient_id)

- Purpose: Deletes a patient record by ID.
- Validation:
  - Checks if the patient exists.
  - Ensures no dependent records in appointments or billing (or handles cascading).
- Database Operation: Deletes the row from the patient's table.

4. get_by_id(patient_id)

- Purpose: Retrieves a patient's details with their ID.
- Database Operation: Fetches the row from the patient's table.

5. view ()

- Purpose: Lists all patients in the system.
- Database Operation: Selects all rows from the patients table and displays them in a readable format.

6. search_by_name(name)

- Purpose: Finds patients by partial or full name.
- Database Operation: Performs a LIKE query on the name field.

7. export_patients_to_csv(filename="patients.csv")

- Purpose: Exports all patient records to a CSV file for reporting or backup.

Field	Type	Description
patient_id	varchar/int	Primary Key, unique ID
name	varchar	Patient's full name
age	int	Age
gender	varchar	Gender
address	varchar	Address
phone	varchar	Contact number (unique)
email	varchar	Email address

Error Handling & Validation

- Ensure all mandatory fields are provided and valid.
- Handles database integrity errors (e.g., duplicate phone numbers).
- Provides clear error messages for user actions.

Integration with Other Modules

- Appointments: Patient ID is referenced in the appointments table.
- Billing: Patient ID is referenced in the billing table.
- Services: Patient service usage may be tracked for billing.

Best Practices

- Always validate user input before database operations.
- Use parameterized queries to prevent SQL injections.
- Handle exceptions and provide user-friendly error messages.

6.2. Doctor Records (doctor.py)

The doctor.py module is dedicated to managing doctor records within the Hospital Appointment & Billing Management System. It provides all necessary functionality to add, update, delete, and retrieve doctor information, and ensures smooth integration with appointment scheduling and billing modules.

Class: Doctor

Purpose

Encapsulates all attributes and operations related to a doctor's record.

Attributes

- doctor_id: Unique identifier for the doctor (string, e.g., "D001", auto-generated or user-defined).
- name: Full name of the doctor (string).
- specialization: Medical specialty (string, e.g., "Cardiology", "Pediatrics").
- phone: Contact phone number (string).
- email: Email address (string, optional).
- department: Department or unit in the hospital (string, optional).

Methods

1. add()

- Purpose: Adds a new doctor record to the database.
- Validation:
    - Ensures all required fields are present and valid.
    - Checks for duplicate doctor ID or phone/email.
- Database Operation: Inserts a new row into the doctors table.

2. update()

- Purpose: Updates details for an existing doctor.
- Validation:
    - Checks if the doctor exists.
    - Validates updated fields.
- Database Operation: Updates the row in the doctors table where doctor_id matches.

3. delete(doctor_id)

- Purpose: Deletes a doctor record by ID.
- Validation:
    - Checks if the doctor exists.
    - Ensures no dependent records in appointments (or handles cascading).
- Database Operation: Deletes the row from the doctors table.

4. get_by_id(doctor_id)

- Purpose: Retrieves a doctor's details by their ID.
- Database Operation: Fetches the row from the doctors table.

5. view ()

- Purpose: Lists all doctors in the system.
- Database Operation: Selects all rows from the doctors table and displays them in a readable format.

6. search_by_specialization(specialization)

- Purpose: Finds doctors by their specialization.
- Database Operation: Performs a LIKE query on the specialization field.

7. export_doctors_to_csv(filename="doctors.csv")

- Purpose: Exports all doctorrecords to a CSV file for reporting or backup.

Field	Type	Description
doctor_id	varchar	Primary Key, unique ID
name	varchar	Doctor's full name
specialization	varchar	Medical specialty
phone	varchar	Contact number (unique)
email	varchar	Email address
department	varchar	Department/unit

Error Handling & Validation

- Ensure all mandatory fields are provided and valid.
- Handles database integrity errors (e.g., duplicate phone numbers or emails).
- Provides clear error messages for user actions.

Integration with Other Modules

- Appointments: Doctor ID is referenced in the appointments table.
- Billing/Invoices: Doctor details are included in invoices for patient consultations.
- Reporting: Doctor data can be exported and used in hospital analytics.

Best Practices

- Always validate user input before database operations.
- Use parameterized queries to prevent SQL injections.
- Handle exceptions and provide user-friendly error messages.

6.3. Service Records(service.py)

The service.py module manages hospital services and tracks their usage by patients. It allows administrators to add, update, delete, and view available services, as well as record and retrieve which services have been used by which patients. This module is critical for accurate billing and reporting.

Main Components

Class: Service

Purpose

Represents a single hospital service (e.g., X-ray, Blood Test, MRI) and provides methods for CRUD (Create, Read, Update, Delete) operations on the services table.

Attributes

- service_id: Unique identifier for the service (string, e.g., "S001", auto-generated or user-defined).
- service_name: Name of the service (string).
- cost: Cost of the service (float).

Methods

1. **add()**
    - Adds a new service to the database after validating inputs.
2. **update()**
    - Updates the details (name, cost) of an existing service.
3. **delete(service_id)**
    - Deletes a service from the database by its ID.
4. **get_by_id(service_id)**
    - Retrieves details of a specific service.

5. **view()**
    - Displays all available services in a readable format.
6. **search_by_name(service_name)**
    - Finds services by partial or full name.
7. **export_services_to_csv(filename="services.csv")**
    - Exports all service records to a CSV file.

Field	Type	Description
service_id	varchar	Primary Key, unique ID
service_name	varchar	Name of the service
cost	float	Cost of the service

Class: ServiceUsageDB

Purpose

Tracks which services have been used by each patient (before billing). This is referenced in the billing process to generate accurate bills.

Methods

1. **add_service_for_patient(patient_id, service_id, service_name, cost)**
    - Records that a patient has used a particular service. Adds an entry to the temp_service_usage table.
2. **get_services_for_patient(patient_id)**
    - Retrieves all unbilled services used by a patient (returns a list of tuples: (service_id, service_name, cost)).
3. **clear_services_for_patient(patient_id)**
    - Removes all unbilled service usage records for a patient (typically called after billing is complete).

4. **view_service_usage()**

    - Displays all current (unbilled) service usages for all patients.

Field	Type	Description
patient_id	varchar	Foreign Key to patients
service_id	varchar	Foreign Key to services
service_name	varchar	Name of the service
cost	float	Cost of the service

Error Handling & Validation

- Ensures all service fields are valid before database operations.
- Handles duplicate service IDs and integrity errors gracefully.
- Provides clear error messages for invalid operations.

Integration with Other Modules

- Billing: The billing module fetches unbilled services for a patient from ServiceUsageDB and clears them after billing.
- Appointments: Services may be associated with appointments for certain workflows.

Best Practices

- Always validate input data for service name and cost.
- Use parameterized queries to prevent SQL injection.
- Handle exceptions and provide user-friendly error messages.
- Regularly export service data for backup and reporting.

6.4. Appointment Management (appointment.py)

The appointment.py module manages all operations related to patient appointments in the Hospital Management System. It provides functionalities for creating, updating, deleting, viewing, filtering, and exporting appointments. The module ensures data validation and integrates with a MySQL database for persistent storage.

Function: auto_appt_id()

Purpose:

Automatically generates the next available unique appointment ID in the format A### (e.g., A001, A002).

How it works:

- Connects to the database.
- Retrieves all appointment IDs starting with 'A'.
- Extracts the numeric part, finds the maximum, and increments it.
- Returns the next appointment ID as a string.

Class: Appointment

Attributes

- appt_id: Appointment ID (string, e.g., "A001").
- patient_id: Patient ID (integer as string).
- doctor_id: Doctor ID (string).
- date: Appointment date (string, format: YYYY-MM-DD).
- diagnosis: Diagnosis description (string).
- consulting_charge: Consultation charge (float).

Methods

1. add(self)

- Purpose: Adds a new appointment to the database.
- Validation:
    - Checks for valid patient ID (numeric), doctor ID (string), date (YYYY-MM-DD), diagnosis (string), and consulting charge (positive float).
- Operation:
    - Inserts a new record into the appointments table.
    - Handles duplicate appointment IDs and other integrity errors.

2. update(self)

- Purpose: Updates an existing appointment in the database.
- Validation:
    - Similar to add, but also ensures consulting charge is within a reasonable range (0–10,000).
- Operation:
    - Updates the record in the appointments table for the given appt_id.

3. get_by_id(appt_id) *(staticmethod)*

- Purpose: Fetches a single appointment record by its ID.
- Operation:
    - Returns the appointment as a dictionary, or None if not found.

4. delete(appt_id) *(staticmethod)*

- Purpose: Deletes an appointment by its ID.
- Operation:

- Removes the record from the appointments table.
- Returns True if successful, False otherwise.

5. view() ***(staticmethod)***

- Purpose: Displays all appointments in the system.
- Operation:
  - Prints all appointments in a tabular format.

6. filter_appointments() ***(staticmethod)***

- Purpose: Filters and displays appointments within a user-specified date range.
- Operation:
  - Prompts for start and end dates.
  - Displays matching appointments.

7. days_between_appointments(patient_id) ***(staticmethod)***

- Purpose: Calculates the number of days between consecutive appointments for a given patient.
- Operation:
  - Fetches all appointment dates for the patient, ordered chronologically.
  - Calculates and prints the days between each appointment.

8. export_appointment_summary_to_csv(filename="appointment_summary.csv") ***(staticmethod)***

- Purpose: Exports all appointment records to a CSV file.
- Operation:
  - Writes appointment data to the specified CSV file.

Error Handling & Validation

- Validates all input fields before performing database operations.
- Handles database integrity errors, such as duplicate IDs.
- Provides clear, user-friendly error messages for invalid input or failed operations.
- Ensures database connections and cursors are closed after each operation.

Field	Type	Description
appt_id	varchar	Appointment ID (Primary Key)
patient_id	int	Patient ID (Foreign Key)
doctor_id	varchar	Doctor ID (Foreign Key)
date	date	Appointment date
diagnosis	varchar	Diagnosis description
consulting_charge	float	Consultation charge

Integration

- **Patient and Doctor Modules**: Uses patient_id and doctor_id as foreign keys; these must exist in their respective tables.
- **Billing Module**: Appointment data (including consulting charge) is referenced during invoice generation.

Best Practices

- Always use the auto_appt_id() function to generate new appointment IDs.
- Validate all user input before adding or updating records.
- Regularly export appointment data for backup and compliance.
- Handle exceptions gracefully and log errors for troubleshooting.

6.5. Billing Management (billing.py)

The billing.py module manages all billing operations in the Hospital Management System. It provides functionalities to create, update, delete, view, and retrieve bills, as well as generate detailed invoices for patients. The module integrates with the services and appointments modules, ensuring that all billed services and consultation charges are accurately recorded and reported.

Function: auto_bill_id()

Purpose:

Automatically generates the next available unique bill ID in the format B### (e.g., B001, B002).

How it works:

- Connects to the database.
- Retrieves all bill IDs starting with 'B' followed by digits.
- Extracts the numeric part, finds the maximum, and increments it.
- Returns the next bill ID as a string.

Class: Bill

Attributes

- bill_id: Bill ID (string, e.g., "B001").
- patient_id: Patient ID (string).
- billing_date: Billing date (string, format: YYYY-MM-DD, defaults to today if not provided).

Methods

1. add(self)

- Purpose: Adds a new bill to the database for a patient.
- Validation:
    - Checks for the presence of patient ID.

- Validates billing date format.
- Ensures the patient exists in the database.
- Fetches all unbilled services for the patient from temp_service_usage.

- Operation:
  - Calculates the total amount for all services.
  - Inserts a new record into the billing table.
  - Inserts each billed service into the billed_services table.
  - Clears temporary service usage for the patient after billing.
  - Handles duplicate bill IDs and other integrity errors.

2. update(self)

- Purpose: Updates an existing bill in the database.
- Validation:
  - Checks for the presence of bill ID and patient ID.
  - Validates billing date format.
  - Ensures the patient exists in the database.
  - Fetches all unbilled services for the patient from temp_service_usage.
- **Operation:**
  - Recalculates the total amount for all services.
  - Updates the record in the billing table for the given bill_id.
  - Clears temporary service usage for the patient after updating the bill.

3. get_by_id(bill_id) ***(staticmethod)***

- Purpose: Fetches a single bill record by its ID.
- Operation:
    - Returns the bill as a dictionary, or None if not found.

4. delete(bill_id) ***(staticmethod)***

- Purpose: Deletes a bill by its ID.
- Operation:
    - Removes the record from the billing table.
    - Returns True if successful, False otherwise.

5. view() ***(staticmethod)***

- Purpose: Displays all bills in the system.
- Operation:
    - Prints all bills in a tabular format.

6. generate_invoice(self)

- Purpose: Generates and prints a detailed invoice for the bill, including patient, doctor, appointment, and service details.
- Operation:
    - Fetches patient details.
    - Fetches the latest appointment and doctor details for the patient.
    - Fetches all billed services for the bill.
    - Calculates totals and formats the invoice for display.

Database Tables:

1. Table: billing

Field	Type	Description
bill_id	varchar	Bill ID (Primary Key)
patient_id	varchar	Patient ID (Foreign Key)
total_amount	float	Total billed amount
billing_date	date	Billing date

2. Table: billed_services (Optional)

Field	Type	Description
bill_id	varchar	Bill ID (Foreign Key)
patient_id	varchar	Patient ID (Foreign Key)
service_id	varchar	Service ID (Foreign Key)
service_name	varchar	Name of the service
cost	float	Cost of the service

3. Table: temp_service_usage (Optional)

Field	Type	Description
patient_id	varchar	Patient ID
service_id	varchar	Service ID
service_name	varchar	Name of the service
cost	float	Cost of the service

Error Handling & Validation

- Validates all input fields before performing database operations.
- Handles database integrity errors, such as duplicate IDs.

- Provides clear, user-friendly error messages for invalid input or failed operations.
- Ensures database connections and cursors are closed after each operation.

Integration

- **Service Module**: Uses ServiceUsageDB to fetch and clear unbilled services for a patient.
- **Patient and Doctor Modules**: References patient and doctor details for invoice generation.
- **Appointment Module**: Fetches latest appointment and consulting charge for invoice.

Best Practices

- Always use the auto_bill_id() function to generate new bill IDs.
- Validate all user input before adding or updating records.
- Regularly review and export billing data for backup and compliance.
- Handle exceptions gracefully and log errors for troubleshooting.

7. CORE WORKFLOWS

7.1. Appointment Workflow

- **Create Appointment**
  - System generates a unique appointment ID.
  - User provides patient ID, doctor ID, date, diagnosis, and consulting charge.
  - Input is validated and stored in the database.
- **Update Appointment**
  - User specifies the appointment ID and updated details.
  - System validates and updates the record.
- **Delete Appointment**
  - User specifies the appointment ID to remove.
  - System deletes the record if it exists.
- **View & Export**
  - All appointments can be viewed in a tabular format or exported as a CSV for reporting.
- **Date Filtering & Analysis**
  - Appointments can be filtered by date range.
  - Days between a patient's appointments can be calculated for analysis.

7.2. Billing Workflow

- **Create Bill**
  - System generates a unique bill ID.
  - User provides patient ID and (optionally) billing date.
  - System fetches all unbilled services used by the patient.
  - Total amount is calculated and stored.
  - Each service is recorded in the billed_services table.
  - Temporary service usage records are cleared.
- **Update Bill**
  - Bill details and amount can be updated as needed.
- **Delete Bill**
  - Bill can be removed by specifying its ID.
- **View Bills**
  - All bills can be listed for administrative review.
- **Generate Invoice**
  - Produces a formatted invoice showing:
    1. Bill and patient details
    2. Doctor and appointment information
    3. Itemized list of services and charges
    4. Consultation charge and total amount

8. Error Handling & Validation

- All user inputs are validated for correct format (IDs, dates, charges).
- Database integrity is enforced via exception handling.
- Duplicate IDs and missing records are reported with clear messages.
- Temporary service usage is cleared after successful billing to prevent duplicate charges.

9. Reporting & Export

- **Appointment Summaries** can be exported as CSV files for external analysis or compliance.
- **Invoices** are generated in a clear, human-readable format for printing or digital sharing.

10. Extensibility

- The system is modular and can be extended to include more features such as:
    - Patient registration and management
    - Advanced reporting and analytics
    - User authentication and access control

11. CLI WORFLOW 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 Record

```
=== 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 ===
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: 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 ===
1. Register New Service
2. Display All Services
3. Modify Service
4. Remove Service
5. Return to Main Menu
Select an option: 3
Enter Service ID to update: S301
Leave input blank to keep current value.
Enter Service Name [Acute Lasering]: Acute
Enter Cost [5000.00]:
Service 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
```

12. CONCLUSION

This system provides a robust backend for hospital appointments and billing management, ensuring data integrity, operational efficiency, and ease of reporting. It is suitable for integration into larger hospital management solutions or use as a standalone administrative tool.