

HOSPITAL MANAGEMENT SYSTEM

A MINI PROJECT REPORT

Submitted by

LOKESHWAR S 220701146

MOHITH S 220701170



In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE(AUTONOMOUS)

THANDALAM

CHENNAI-602105

2024-2025

BONAFIDE CERTIFICATE

Certified that this project report “HOSPITAL MANAGEMENT SYSTEM” is the Bonafide work of MOHITH S (220701170) and LOKESHWAR S (220701146) who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE

Dr.R.SABITHA

**Professor and II Year Academic Head
Computer Science and Engineering
Rajalakshmi Engineering College
(Autonomous),**

SIGNATURE

Ms.D.KALPANA

**Assistant Professor (SG),
Computer Science and Engineering
Rajalakshmi Engineering College
(Autonomous),**

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT:

- The Hospital Management System (HMS) is a comprehensive software application designed to streamline and enhance the administrative and operational functions of a healthcare institution. Developed using Python and MySQL, the system offers an efficient and user-friendly interface for managing patient, doctor, and staff information.
- The HMS allows patients to book appointments with doctors, reserve general or ICU beds, select doctors by specialty, request emergency ambulance services, and cancel appointments if necessary. Doctors can log in to view their appointments and inpatients, discharge patients while calculating bills, and update their outpatient department (OPD) lists. Staff members can log in to access billing information, manage emergency lists, and assist with bed bookings.
- The system relies on a MySQL database to store and manage patient records, doctor and staff details, and comprehensive patient stay information, including billing and discharge data. By integrating these essential hospital functions into a single platform, the HMS enhances efficiency, accuracy, and user experience, thereby improving the overall quality of patient care. This project demonstrates the effective use of Python and MySQL to create a scalable and robust solution for hospital management.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 INTRODUCTION

1.2 OBJECTIVES

1.3 MODULES

2. SURVEY OF TECHNOLOGIES

2.1 SOFTWARE DESCRIPTION

2.2 LANGUAGES

2.2.1 SQL

2.2.2 PYTHON

3. REQUIREMENTS AND ANALYSIS

3.1 REQUIREMENT SPECIFICATION

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

3.3 ARCHITECTURE DIAGRAM

3.4 ER DIAGRAM

3.5 NORMALIZATION

4. PROGRAM CODE

5. RESULTS AND DISCUSSION

6.CONCLUSION

7.REFERENCES

CHAPTER 1

1.1 THE INTRODUCTION

- In today's fast-paced and technologically driven world, the healthcare sector is continuously evolving to meet the increasing demands for better patient care and efficient hospital management. Hospitals face numerous challenges, including managing patient records, scheduling appointments, handling admissions, and ensuring effective communication between staff and patients. To address these challenges, the development of an integrated Hospital Management System (HMS) has become essential.
- This project work focuses on the creation of a Hospital Management System using Python and MySQL. The primary objective of this system is to streamline and automate the various administrative and operational tasks within a hospital, thereby improving the overall efficiency and quality of healthcare services

1.2 OBJECTIVES

To enhance the patient experience, the process of booking appointments and hospital beds should be streamlined, and patients should have easy access to a roster of available doctors, along with their specialties. The system should also allow for straightforward cancellation of appointments and prompt handling of emergency service requests. For improving doctor efficiency, the system should enable doctors to easily view their schedules, manage appointments, and update Outpatient Department (OPD) lists post consultations. Additionally, it should support accurate patient discharge processes and billing calculations. To aid hospital staff, tools for managing patient billing, emergency requests, bed bookings, and patient records are essential. Lastly, a centralized database management system is crucial for maintaining an extensive database of patients, doctors, and staff, ensuring secure, efficient data storage and retrieval. This comprehensive approach aims to streamline hospital operations and improve overall service quality.

Methodology:

- The Hospital Management System is developed using Python, a versatile and widely-used programming language, and MySQL, a powerful relational database management system. Python provides a robust framework for implementing the core application logic, while MySQL ensures reliable and scalable data storage.

Key functionalities of the system include:

- **Patient Management:** Booking appointments and beds, selecting doctors, and emergency service requests.
- **Doctor Management:** Viewing appointments, discharging patients, and updating OPD lists.
- **Staff Management:** Accessing billing information, managing emergencies, and assisting with bed bookings.
- **Database Integration:** Storing and managing comprehensive hospital data securely.

1.3 MODULES

- **User Interface Module**
- **Patient Management Module**
- **Doctor Management Module**
- **Staff Management Module**
- **Database Management Module**
- **Billing Module**
- **Security and Authentication Module**
- **Utility Functions Module**

CHAPTER 2

2.1 SOFTWARE DESCRIPTION:

The Hospital Management System (HMS) described in the provided code is a comprehensive software application designed to streamline various administrative and clinical processes in a hospital. The system is built using Python and MySQL, providing functionalities for patient management, doctor and staff operations, appointment scheduling, bed booking, and emergency services. Below is a detailed description of the software's modules and functionalities.

1. Patient Management

Features:

Appointment Booking: Patients can book appointments with doctors by providing their personal details and selecting a suitable date.

Bed Booking: Patients can book general or ICU beds by providing their admission details.

Doctor Search: Patients can find the best doctor by department.

Emergency Services: Patients can request an ambulance by providing their location and contact details.

Appointment Cancellation: Patients can cancel their appointments using their unique ID.

Workflow:

Patients enter their details and select a service (appointment or bed booking).

The system

generates a unique ID for each patient.

The patient's information is stored in the MySQL database.

2. Doctor Management

Features:

Login: Doctors can log in using their unique ID.

View Appointments: Doctors can view their scheduled appointments and in-hospital patients.

Patient Discharge: Doctors can discharge patients by calculating the bill based on the bed type and duration of stay.

OPD List Update: Doctors can update their outpatient department list by

removing checked patients.

Workflow:

Doctors log in and access their dashboard.

They can manage appointments, discharge patients, and update their patient list.

The system calculates the patient's bill based on their stay duration and bed type.

3. Staff Management

Features:

Login: Staff members can log in using their unique ID.

Billing: Staff can view the bill details of patients.

Emergency List: Staff can access the list of emergency cases.

Bed Booking: Staff can book beds for patients.

Workflow:

Staff log in and access their functionalities.

They can manage billing, view emergency cases, and assist in bed booking for patients.

4. Database Management

Tables:

patient: Stores patient details and appointment information.

admission: Stores details of patients admitted to the hospital.

final: Maintains a summary of patient details, including billing and stay duration.

doc: Stores doctor details and their departments.

staff: Stores staff details.

emergency2: Logs emergency ambulance requests.

Database Operations:

Insertion, deletion, and updating of records based on user interactions.

Generation of unique IDs for patients to track their records.

Calculation and updating of patient bills during discharge.

5. User Interface

The user interface is command-line based, guiding users through various options and capturing their inputs. The main menu provides access to different modules for patients, doctors, and staff. Each module further presents relevant options to the user.

Menus and Navigation:

Main Menu: Offers options for Patient, Doctor, Staff, and information about the hospital.

Patient Menu: Options for booking appointments, booking beds, finding doctors, requesting ambulances, and canceling appointments.

Doctor Menu: Options for logging in, viewing appointments, discharging

patients, and updating OPD lists.

Staff Menu: Options for logging in, viewing patient bills, accessing emergency lists, and booking beds.

2.2 LANGUAGES:

Languages Used in the Hospital Management System

The Hospital Management System (HMS) described in the provided code utilizes the following programming languages and technologies:

1. Python

Role:

Python serves as the primary programming language for the application logic and database interactions. It handles user inputs, processes data, and communicates with the MySQL database to perform various operations like storing patient details, booking appointments, managing doctor schedules, and more.

Advantages:

Readability and Simplicity: Python's syntax is clear and easy to understand, which makes the code more maintainable and less error-prone.

Extensive Libraries: Python has a vast collection of libraries and frameworks, such as `mysql.connector` for database connectivity, which simplifies the development process.

Rapid Development: Python allows for quick development cycles, which is beneficial for prototyping and iterating over software features.

Key Libraries:

`mysql.connector`: A library that allows Python to connect to and interact with MySQL databases. It is used for executing SQL queries and handling database operations.

2. SQL (Structured Query Language)

Role:

SQL is used for creating and managing the database schema, as well as for querying and manipulating the data stored in the MySQL database. It facilitates data definition, data control, and data manipulation.

Advantages:

Data Management: SQL is specifically designed for managing and manipulating structured data stored in relational databases.

Standardized Language: SQL is a standard language for relational database management, which ensures compatibility and interoperability with various database systems.

Complex Queries: SQL can handle complex queries involving multiple tables and

conditional logic, making it ideal for applications like HMS that require detailed data retrieval and reporting

Examples of SQL Usage in the HMS:

Creating tables to store patient, doctor, staff, and emergency information.

Inserting new records when patients book appointments or beds.

Updating records when doctors discharge patients and calculate bills.

Deleting records when patients cancel appointments.

Integration of Python and SQL:

The integration of Python and SQL in the HMS allows for a seamless interaction between the application logic and the database. Here's how these technologies work together in the system:

Database Connection: Python uses the `mysql.connector` library to establish a connection to the MySQL database.

Executing Queries: Python scripts execute SQL queries to perform CRUD (Create, Read, Update, Delete) operations on the database. For instance, when a patient books an appointment, a SQL `INSERT` query is executed to add the patient's details to the database.

Data Retrieval: Python retrieves data from the database using SQL `SELECT` queries. This data is then processed and displayed to the user, such as listing available doctors or showing the bill details of a patient.

Transaction Management: Python handles database transactions to ensure data consistency and integrity. For example, when a doctor discharges a patient and updates the bill, Python ensures that all related database updates are completed successfully.

CHAPTER 3

REQUIREMENT AND ANALYSIS

Functional Requirements

Patient Management:

Appointment Booking: Patients should be able to book appointments with doctors by providing their personal details and selecting a suitable date.

Bed Booking: Patients should be able to book general or ICU beds by providing their admission details.

Doctor Search: Patients should be able to search for the best doctor by department. **Emergency Services:** Patients should be able to request an ambulance by providing their location and contact details.

Appointment Cancellation: Patients should be able to cancel their appointments using their unique ID.

Doctor Management:

Login: Doctors should be able to log in using their unique ID.

View Appointments: Doctors should be able to view their scheduled appointments and in-hospital patients.

Patient Discharge: Doctors should be able to discharge patients by calculating the bill based on the bed type and duration of stay.

OPD List Update: Doctors should be able to update their outpatient department list by removing checked patients.

Staff Management:

Login: Staff members should be able to log in using their unique ID.

Billing: Staff should be able to view the bill details of patients.

Emergency List: Staff should be able to access the list of emergency cases.

Bed Booking: Staff should be able to book beds for patients.

3. Non-Functional Requirements

Performance:

The system should handle multiple user requests simultaneously without significant delays.

Database queries should be optimized for fast retrieval and update operations.

Usability:

The command-line interface should be intuitive and user-friendly, guiding users

through various options and capturing their inputs efficiently.

Security:

User authentication should be implemented to ensure that only authorized personnel can access sensitive information and perform certain operations. Sensitive data, such as patient information, should be securely stored in the database.

Reliability:

The system should ensure data integrity and consistency, especially during transactions such as appointment booking, patient discharge, and billing.

Scalability:

The system should be designed to accommodate future expansions, such as adding new departments, increasing the number of users, and integrating additional functionalities.

4. System Analysis

User Roles and Permissions:

Patients: Can book appointments, book beds, search for doctors, request emergency services, and cancel appointments.

Doctors: Can log in, view appointments, discharge patients, and update their OPD lists.

Staff: Can log in, view patient bills, access emergency lists, and book beds for patients.

Database Design:

The database should have tables to store patient details, doctor details, staff details, appointment information, bed booking information, and emergency requests. Relationships between tables should be defined to ensure data consistency and integrity.

Key Tables:

patient: Stores patient details and appointment information.

admission: Stores details of patients admitted to the hospital.

final: Maintains a summary of patient details, including billing and stay duration.

doc: Stores doctor details and their departments.

staff: Stores staff details.

emergency2: Logs emergency ambulance requests.

User Interface Design:

The command-line interface should present a clear and organized menu

structure, allowing users to navigate through various options. Each menu option should prompt the user for necessary inputs and provide appropriate feedback.

Process Flow:

Appointment Booking: Patients provide their details, select a doctor, and choose a suitable date.

The system generates a unique ID and stores the appointment information in the database.

Bed Booking: Patients or staff provide admission details and select the type of bed. The system generates a unique ID and stores the booking information in the database.


Patient Discharge: Doctors calculate the bill based on the patient's stay duration and bed type, update the final bill in the database, and remove the patient's admission record.

Emergency Services: Patients provide their location and contact details. The system logs the request and ensures an ambulance is dispatched.

3.1 HARDWARE AND SOFTWARE REQUIREMENTS


Hardware Requirements

Client Machines:

 Processor: Intel Core i3 or equivalent (minimum)


 RAM: 4 GB (minimum)

 Storage: 100 GB HDD or SSD

 Display: 1024x768 resolution (minimum)


 Network Interface: Ethernet or Wi-Fi capability

Server Machine:

 Processor: Intel Xeon or equivalent (recommended)

 RAM: 16 GB (minimum)

 Storage: 1 TB HDD or SSD (minimum)

 Network Interface: High-speed Ethernet

 Backup Device: External HDD or Cloud Backup service

 Power Supply: Uninterruptible Power Supply (UPS)

Software Requirements

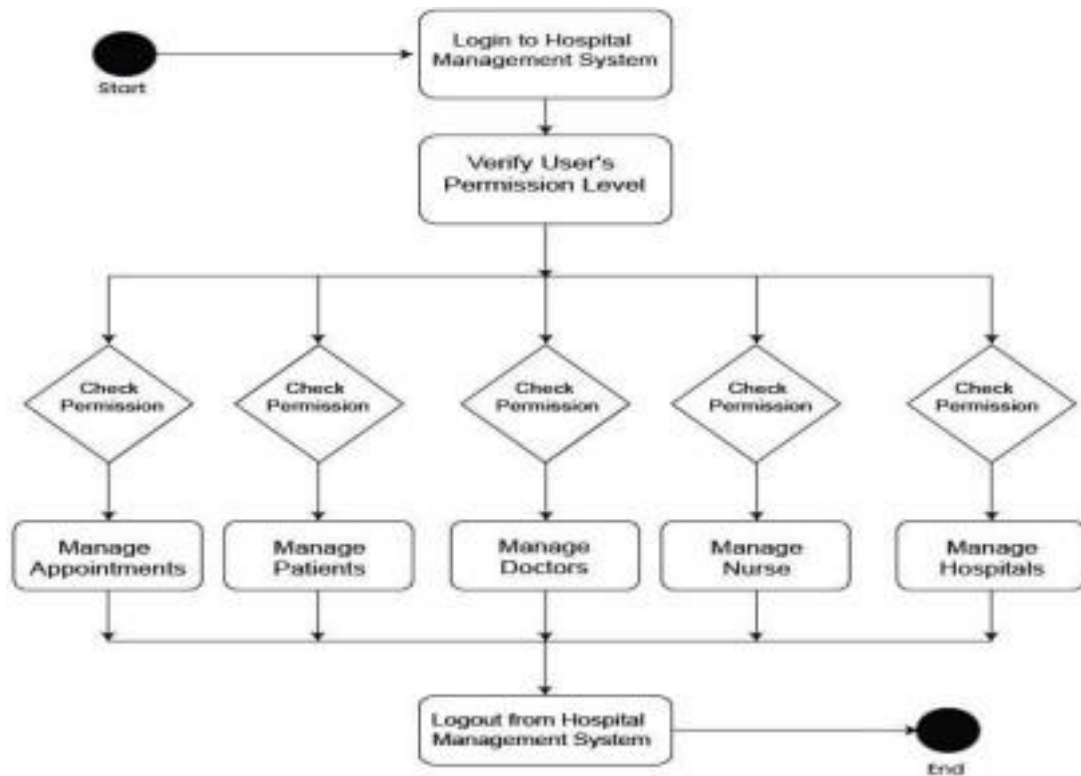
Client-Side Software:

- Operating System: Windows 10, macOS, or Linux
- Python Interpreter: Python 3.6 or higher
- MySQL Connector for Python: mysql-connector-python
- Text Editor or IDE: PYTHON IDE
- Terminal or Command Prompt: For executing scripts and commands/
mysql Workbench.

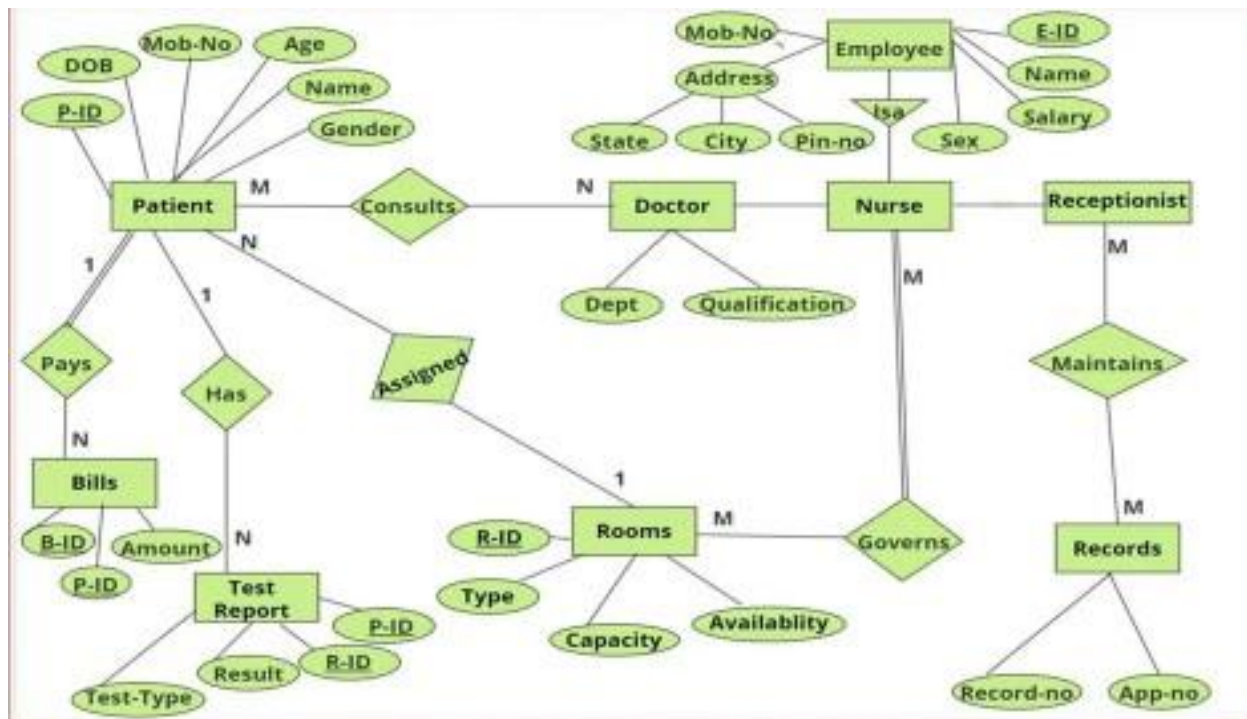
Server-Side Software:

- Operating System: Windows Server, Ubuntu Server, or any other Linux distribution
- Database Management System: MySQL Server 8.0 or higher
- Python Interpreter: Python 3.6 or higher
- MySQL Connector for Python: mysql-connector-python
- Development Tools:
 - Version Control System: Git , Github
 - Documentation Tools: Microsoft Word, Google Docs, or any other documentation tool.
- Dependencies and Libraries
 - mysql-connector-python: Library to connect Python applications to MySQL database.
- Network Requirements:
 - Internet Connection: Stable internet connection for initial setup and updates.
 - Internal Network: Secure and fast local area network (LAN) for communication between client and server machines.

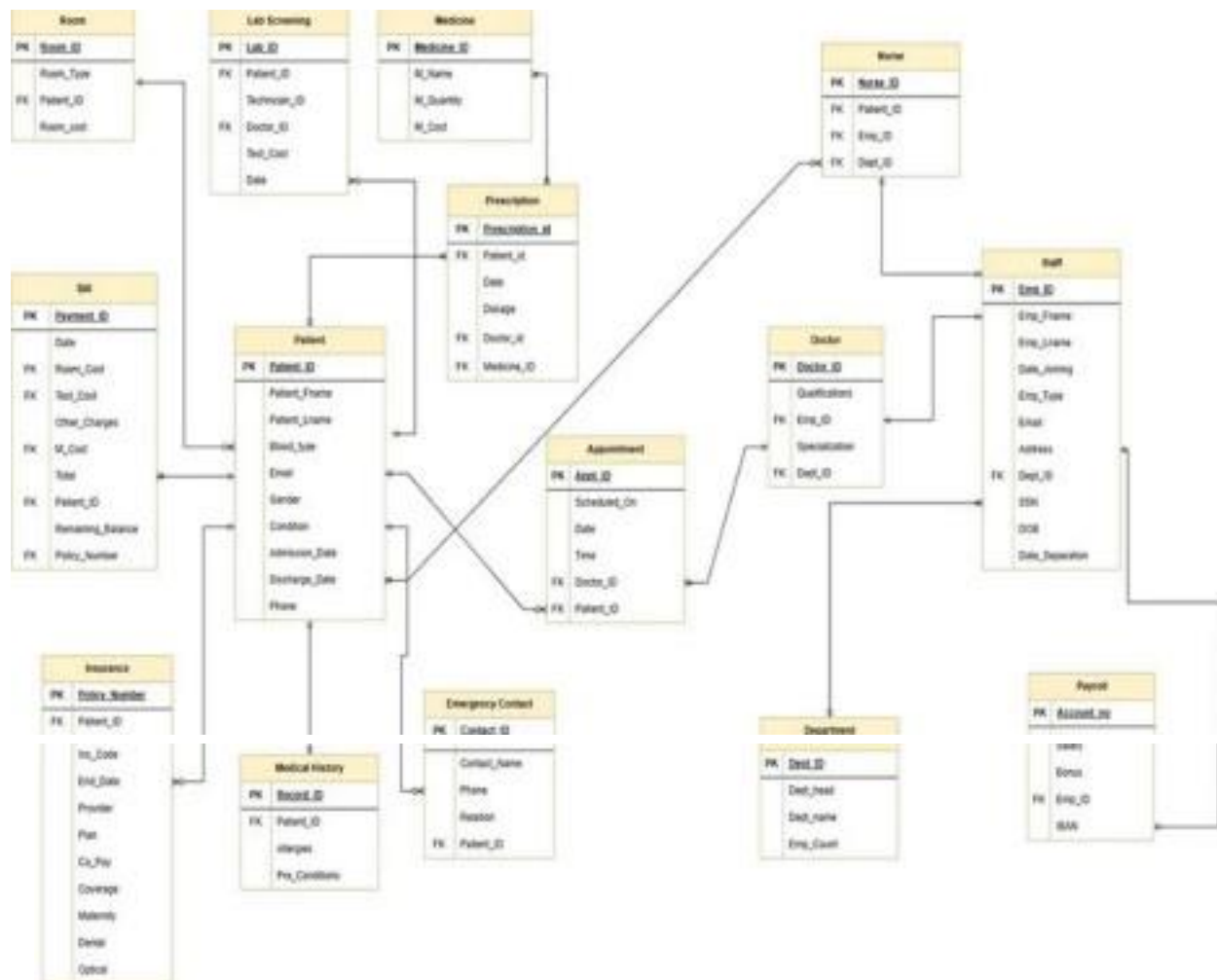
3.3 ARCHITECTURE DIAGRAM



3.4 ER DIAGRAM



3.5 NORMALISATION



CHAPTER 4

4.1 PROGRAM CODE

HOSPITAL MANAGEMENT SYSTEM USING PYTHON AND MYSQL

PYTHON IDLE CODE

PROGRAM:

```
import tkinter as tk
```

```
from tkinter import messagebox, simpledialog
```

```
import random
```

```
import mysql.connector as a
```

```
con = a.connect(host="localhost", user="root", passwd="Lokeshwar2004",  
database="hos")
```

```
c = con.cursor()
```

```
doc_id = ["9854", "4356", "8709", "9865", "9855", "4214", "2854", "7721",  
"3211", "3290"]
```

```
staff_id = ["1311", "3490", "5263", "1119", "0678"]
```

```
class HospitalApp:
```

```
    def __init__(self, root):
```

```
        self.root = root
```

```
        self.root.title("GRC Hospital")
```

```
        self.main_menu()
```

```
    def main_menu(self): self.clear_window()
```

```
tk.Label(self.root, text=" -----WELCOME TO GRC  
HOSPITAL ----- ").pack()
```

```
tk.Label(self.root, text="Main menu ----- ").pack()
```

```
tk.Button(self.root, text="1. Patient",  
command=self.patient_menu).pack()
```

```
tk.Button(self.root, text="2. Doctor",  
command=self.doctor_menu).pack()
```

```
tk.Button(self.root, text="3. Staff", command=self.staff_menu).pack()
```

```
tk.Button(self.root, text="4. Know more",  
command=self.know_more).pack()
```

```
def clear_window(self):
```

```
    for widget in self.root.winfo_children():
```

```
        widget.destroy()
```

```
def know_more(self):
```

```
    message = (
```

```
        "GRC Hospital, established in 2001, is committed to delivering premium  
quality healthcare within reach of everyone at affordable costs. "
```

```
        "Our mission is to provide international standard care combined with  
exceptional value for money. "
```

"With a team of highly qualified doctors, state-of-the-art facilities, and compassionate staff, we ensure the best possible treatment and comfort for our patients.\n\n"

"At GRC Hospital, we believe in the care you can trust. We offer a wide range of medical services, including cardiology, neurology, dermatology, surgery, and gastroenterology, supported by advanced diagnostic and therapeutic technologies."

"Our 24-hour emergency services and ambulance facilities ensure that help is always available when you need it the most.\n\n"

"GRC Hospital is not just about treatment; it's about creating a nurturing environment where patients and their families feel supported and confident in their care journey. "

"Our patient-centric approach ensures personalized treatment plans tailored to individual needs.\n\n"

"Explore our specialized departments and find the best healthcare professionals dedicated to improving your health and well-being. With GRC Hospital, you are in safe hands.")

```
messagebox.showinfo("ABOUT GRC ",message)
```

```
def patient_menu(self):
```

```
    self.clear_window()
```

```
    tk.Label(self.root, text="Patient ----- ").pack()
```

```
    tk.Button(self.root, text="1. Book an appointment",  
command=self.book_appointment).pack()
```

```
tk.Button(self.root, text="2. Book a bed",  
command=self.book_bed).pack()
```

```
tk.Button(self.root, text="3. Find your best doctor",  
command=self.find_best_doctor).pack()
```

```
tk.Button(self.root, text="4. 24 hrs ambulance",  
command=self.book_ambulance).pack()
```

```
tk.Button(self.root, text="5. Cancel an appointment",  
command=self.cancel_appointment).pack()
```

```
tk.Button(self.root, text="6. Return to Main menu",  
command=self.main_menu).pack()
```

```
def book_appointment(self):
```

```
name = simpledialog.askstring("Input", "Enter Name:") age =
```

```
simpledialog.askstring("Input", "Enter Age:") gender =
```

```
simpledialog.askstring("Input", "Enter Gender:")
```

```
doctor_id = simpledialog.askstring("Input", "Enter Doctor's ID you  
wanna appoint:")
```

```
if doctor_id not in doc_id:
```

```
messagebox.showerror("Error", "Enter a valid doctor ID")return
```

```
date = simpdialog.askstring("Input", "Enter your suitable date in  
YYYY-MM-DD format:")
```

```
contact = simpdialog.askstring("Input", "Enter your contact no:")
```

```
unique_id = self.uid()
```

```
messagebox.showinfo("Success", f"Your unique generated ID:  
{unique_id}\n\nCongratulations Your appointment has been booked!")
```

```
data1 = (unique_id, name, age, gender, date, contact, doctor_id)sql1 =
```

```
"insert into patient values(%s,%s,%s,%s,%s,%s,%s)" c.execute(sql1,
```

```
data1)
```

```
con.commit()
```

```
data2 = (unique_id, name, age, doctor_id, date, 0, 0) sql2 =
```

```
"insert into final values(%s,%s,%s,%s,%s,%s,%s)"
```

```
c.execute(sql2, data2)
```

```
con.commit()
```

```

def book_bed(self):

    name = simpdialog.askstring("Input", "Enter Name:") age =

    simpdialog.askstring("Input", "Enter Age:") gender =

    simpdialog.askstring("Input", "Enter Gender:")

    doctor_id = simpdialog.askstring("Input", "Enter Doctor's ID underwhich
you want to take admission:")

    if doctor_id not in doc_id:

        messagebox.showerror("Error", "Enter a valid doctor ID")return

    date = simpdialog.askstring("Input", "Enter your suitable date youwant to
take admission in YYYY-MM-DD format:")

    unique_id = self.uid()

    messagebox.showinfo("Success", f"Your unique generated ID:
{unique_id}")

    bed_type = simpdialog.askstring("Input", "Book a bed ----- \n1. General
bed\n2. ICU bed\nEnter your choice:")

    if bed_type == "1":

```

```

        bed_type = "GEN"

elif bed_type == "2":

    bed_type = "ICU"

else:

    messagebox.showerror("Error", "Enter a valid choice")return

data3 = (unique_id, name, age, gender, doctor_id, bed_type, date, 0)sql3 =

"insert into admission values(%s,%s,%s,%s,%s,%s,%s,%s)" c.execute(sql3,

data3)

con.commit()


data6 = (unique_id, name, age, doctor_id, date, 0, 0) sql6 =

"insert into final values(%s,%s,%s,%s,%s,%s,%s)"

c.execute(sql6, data6)

con.commit()

```



```
messagebox.showinfo("Success", f"Congratulations Your {bed_type} Bed  
has been booked!")
```

```
def find_best_doctor(self):
```

```
    dept_code = simpdialog.askstring("Input", "Find Your best doctor  
----- \n1. Cardiology (CD)\n2. Neurology (NU)\n3. Dermatology  
(DM)\n4. Surgery (SG)\n5. Gastroentrology (GS)\nEnter department code:")
```

```
    c.execute(f"select name from doc where dept='{dept_code}'")
```

```
    result = c.fetchall()
```

```
    doctors = "\n".join([f"Dr. {self.clean(i)}" for i in result])
```

```
    messagebox.showinfo("Doctors", doctors)
```

```
def book_ambulance(self):
```

```
    room_no = simpdialog.askstring("Input", "Enter room no:") patient_name
```

```
    = simpdialog.askstring("Input", "Enter patient name:") contact_no =
```

```
    simpdialog.askstring("Input", "Enter contact number:") location =
```

```
    simpdialog.askstring("Input", "Enter location:")
```

```
    data3 = (room_no, patient_name, contact_no, location)
```

```
sql3 = "insert into emergency2 values(%s,%s,%s,%s)"
```

```
c.execute(sql3, data3)
```

```
con.commit()
```

```
messagebox.showinfo("Success", "Congrats!!! Your ambulance has been  
booked and will reach your location shortly")
```

```
def cancel_appointment(self):
```

```
    unique_id = simpledialog.askstring("Input", "Enter Your ID:")if
```

```
    self.patient_check(unique_id):
```

```
        c.execute(f"DELETE FROM patient WHERE uid={ unique_id}")con.commit()
```

```
        messagebox.showinfo("Success", "Your appointment is Cancelled!")else:
```

```
        messagebox.showerror("Error", "You do not have anyappointments!")
```

```
def staff_menu(self):
```

```
self.clear_window()
```

```
tk.Label(self.root, text="Staff-----").pack()
```

```
tk.Button(self.root, text="1. Login with ID",  
command=self.staff_login).pack()
```

```
tk.Button(self.root, text="2. Return to Main menu",  
command=self.main_menu).pack()
```

```
def staff_login(self):
```

```
    staff_id = simpledialog.askstring("Input", "Enter your ID Number:")if
```

```
    staff_id not in staff_id:
```

```
        messagebox.showerror("Error", "Enter a valid ID")return
```

```
    c.execute(f"select name from staff where ID={staff_id}")
```

```
    result = c.fetchall()
```

```
    staff_name = self.clean(result[0])
```

```
    messagebox.showinfo("Welcome", f"Hi! {staff_name}")
```

```
self.clear_window()
```

```
tk.Label(self.root, text="Login with ID ----- ").pack()
```

```
tk.Button(self.root, text="1. Show the bill of a patient",  
command=self.show_patient_bill).pack()
```

```
tk.Button(self.root, text="2. Show the emergency list",  
command=self.show_emergency_list).pack()
```

```
tk.Button(self.root, text="3. Book a bed for patient",  
command=self.book_bed).pack()
```

```
tk.Button(self.root, text="4. Return to Main menu",  
command=self.main_menu).pack()
```

```
def show_patient_bill(self):
```

```
    patient_id = simpledialog.askstring("Input", "Enter the patient's ID  
Number:")
```

```
    c.execute(f"select bill from final where ID={patient_id}")
```

```
    result = c.fetchall()
```

```
    bill = self.clean(result[0])
```

```
    messagebox.showinfo("Bill", f"Net bill of patient with ID {patient_id}:  
₹{bill}")
```

```
def show_emergency_list(self):
```

```

c.execute("select * from emergency")result =

c.fetchall()

emergencies = "\n".join([str(i) for i in result])

messagebox.showinfo("Emergency List", f"('Location', 'Contact
Number')\n{emergencies}")


def doctor_menu(self):

    self.clear_window()

    tk.Label(self.root, text="Doctor -----").pack()

    tk.Button(self.root, text="1. Login with ID",command=self.doctor_login).pack()

    tk.Button(self.root, text="2. Return to Main menu",
command=self.main_menu).pack()


def doctor_login(self):

    doctor_id = simpledialog.askstring("Input", "Enter your ID Number:")if

    doctor_id not in doc_id:

        messagebox.showerror("Error", "Enter valid ID")

    return

```

```

c.execute(f"select name from doc where ID={doctor_id}")

result = c.fetchall()

doctor_name = self.clean(result[0]) messagebox.showinfo("Welcome",
f"Hi! Dr. {doctor_name}")

self.clear_window()

tk.Label(self.root, text="Login with ID ----- ").pack()

tk.Button(self.root, text="1. Show my appointments",
command=self.show_appointments).pack()

tk.Button(self.root, text="2. Discharge a patient",
command=self.discharge_patient).pack()

tk.Button(self.root, text="3. Update my opd list",
command=self.update_opd_list).pack()

tk.Button(self.root, text="4. Return to Main menu",
command=self.main_menu).pack()

def show_appointments(self):

    doctor_id = simpdialog.askstring("Input", "Enter your ID Number:")

    messagebox.showinfo("Appointments", "Hello doctor, your today's OPD

```

appointments are ----- ")

```
c.execute(f"select name,uid from patient where dr={doctor_id}")result =
```

```
c.fetchall()
```

```
appointments = "\n".join([self.clean(i) for i in result])
```

```
messagebox.showinfo("Appointments", appointments)
```

messagebox.showinfo("Hospital Patients", "And, your today's patientsin
hospitals are ----- ")

```
c.execute(f"select name,ID from admission where dr={doctor_id}")result =
```

```
c.fetchall()
```

```
hospital_patients = "\n".join([self.clean(i) for i in result])
```

```
messagebox.showinfo("Hospital Patients", hospital_patients)
```

```
def discharge_patient(self):
```

```
    doctor_id = simpledialog.askstring("Input", "Enter your ID Number:")
```

messagebox.showinfo("Patients", "Your all inhospital patients along
with their ID---- ")

```
c.execute(f"select name,ID from admission where dr={doctor_id}")result =
```

```
c.fetchall()
```

```
patients = "\n".join([self.clean(i) for i in result])messagebox.showinfo("Patients",  
  
patients)
```

```
patient_id = simpdialog.askstring("Input", "Enter the id of the patientyou  
want to discharge:")
```

```
c.execute(f"select incoming_date from final where ID={patient_id}")result  
  
= c.fetchall()
```

```
incoming_date = self.clean(result[0])
```

```
messagebox.showinfo("Admission Date", f"Your patient was admittedon  
{incoming_date}")
```

```
days_in_hospital = simpdialog.askstring("Input", "Enter the no of daysthe  
patient was in the hospital under you:")
```

```
c.execute(f"select bed_type from admission where ID={patient_id}")result =  
  
c.fetchall()
```

```
bed_type = self.clean(result[0])
```

```
if bed_type == 'GEN':
```

```
bill = (4500 + 2000 + 1000) * int(days_in_hospital)
```



```
elif bed_type == 'ICU':
```

```
    bill = (7500 + 3000 + 1500) * int(days_in_hospital)
```

```
    c.execute(f"update final set bill={bill} where ID={patient_id}")
```

```
    c.execute(f"DELETE FROM admission WHERE ID={patient_id}")
```

```
    c.execute(f"update final set days={days_in_hospital} where  
ID={patient_id}")
```

```
    con.commit()
```

```
    messagebox.showinfo("Discharge", "Discharged Successfully")
```

```
def update_opd_list(self):
```

```
    patient_id = simpledialog.askstring("Input", "Enter the id of the patient you  
have checked:")
```

```
    c.execute(f"DELETE FROM patient WHERE uid={patient_id}")
```

```
    con.commit()
```

```
    messagebox.showinfo("Update", "Your patient list Updated!")
```

```
def uid(self):
```

```
    return ''.join([str(random.randint(0, 9)) for _ in range(4)])
```

```
def clean(self, str1):
```

```
    return str1[0] if str1 else ""
```

```
def patient_check(self, str1):
```

```
    c.execute("select uid FROM patient")
```

```
    result = c.fetchall()
```

```
    result1 = [self.clean(i) for i in result]
```

```
    return str1 in result1
```

```
if __name__ == "__main__":
```

```
    root = tk.Tk()
```

```
    app = HospitalApp(root)
```

```
    root.mainloop()
```

SQL COMMANDS AND QUERIES:

```
create database hos;use
hos;
```

```
create table patient(uid varchar(20) ,name varchar(50), age varchar(20),gender
varchar(20),dnt date, ph varchar(50),dr varchar(50));
```

```
create table admission(ID varchar(50),name varchar(50),age varchar(50),gender
varchar(50),dr varchar(50),bed_type varchar(50),incoming_date
varchar(50),bill varchar(50));
```

```
create table final(ID varchar(50),name varchar(50),age
varchar(50),dr varchar(50),incoming_date varchar(50),days
varchar(50),bill varchar(5000));
```

```
create table doc(ID varchar(50),dept varchar(50), name
varchar(50));
```

```
insert into doc(ID,dept,name) values("9854","CD","DR.MOHITH");insert
into doc(ID,dept,name) values("4356","CD","DR.MANIKANDAN");
```

```
insert into doc(ID,dept,name) values("8709","NU","DR.MERVIN");
```

```
insert into doc(ID,dept,name) values("9865","NU","DR.LOGESH");
```

```
insert into doc(ID,dept,name) values("9855","DM","DR.LITHAN");
```

```
insert into doc(ID,dept,name) values("4214","DM","DR.LALIT"); insert
```

```
into doc(ID,dept,name) values("7721","SG","DR.LOGESHWARAN");
```

```
insert into doc(ID,dept,name)
values("2854","SG","DR.PRASHANTH");
```

```
insert into doc(ID,dept,name) values("3211","GS","DR.Mukesh");insert
```

```
into doc(ID,dept,name) values("3290","GS","DR.HARIVARSAN");
```

```
create table emergency2(room no int,name varchar(45),location
varchar(50),ph varchar(50));
```

```
create table staff(ID varchar(50),name varchar(50));insert
into staff(ID,name) values("1311","VASU"); insert into
staff(ID,name) values("3490","RAJ"); insert into
staff(ID,name) values("5263","PRIYA");
```

```
insert into staff(ID,name) values("1119","JULIE"); insert  
into staff(ID,name) values("0678","KANNAN");
```

///RUN THIS SQL COMMANDS AND QUERIES IN MYSQL WORKBENCHUNDER
THE CREATED DATABASE hos///

CHAPTER 5

Results:

Functional Hospital Management System:

- The implementation of the source code results in a functional HMS capable of managing various aspects of hospital operations, including patient appointments, bed bookings, doctor-patient interactions, staff management, and emergency services.

User-Friendly Interface:

- The system provides a user-friendly command-line interface (CLI) that allows users, including patients, doctors, and staff, to interact with the system intuitively.

Database Connectivity:

- The integration with MySQL database enables persistent storage of essential hospital data, ensuring data consistency and reliability across multiple sessions.

Appointment Booking and Bed Management:

- Patients can book appointments with doctors and reserve beds based on their requirements, enhancing the efficiency of hospital resource allocation and scheduling.

Staff Management and Emergency Services:

- The system facilitates staff login, enabling access to patient information, billing, and emergency service requests, thereby streamlining hospital operations and response to critical situations.

Discussion:

Data Integrity and Normalization:

- While the system provides essential functionality, there may be opportunities to improve data integrity through database normalization.
■ Analyzing the database schema and applying normalization techniques could minimize redundancy and dependency issues, further enhancing data consistency and reliability.

Scalability and Performance:

- As the HMS system evolves and accommodates a larger volume of data and users, scalability and performance considerations become crucial.
- Optimizing database queries, indexing frequently accessed fields, and implementing caching mechanisms can mitigate potential performance bottlenecks.

Security and Access Control:

- Ensuring data security and access control mechanisms is paramount to protect sensitive patient information and comply with privacy regulations.
- Implementing role-based access control (RBAC) and encryption techniques can safeguard data integrity and confidentiality.

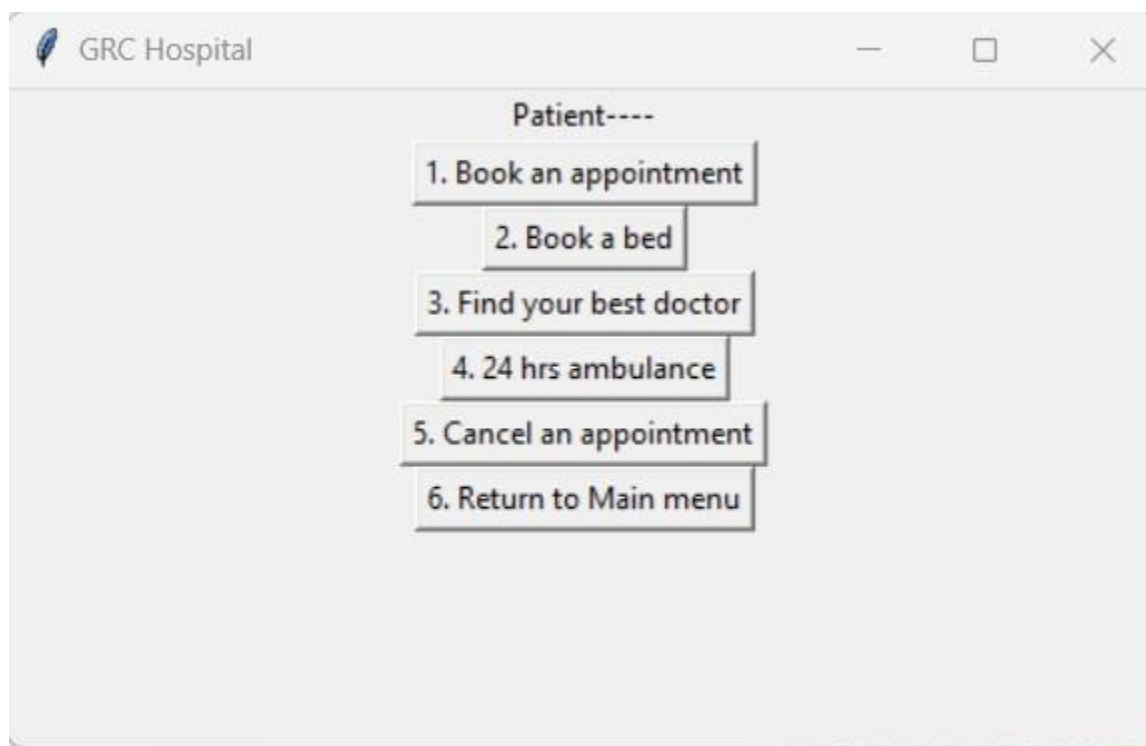
User Feedback and Iterative Development:

- Soliciting feedback from end-users, including hospital staff and administrators, is essential for identifying areas of improvement and iteratively refining the HMS system.
- Continuous development cycles can address user requirements, incorporate new features, and enhance system usability over time.

Integration and Interoperability:

- Integrating the HMS system with other healthcare IT systems, such as electronic health records (EHR) or laboratory information systems (LIS), can improve interoperability and data exchange within the healthcare ecosystem.
- In summary, the project work on the Hospital Management System based on the provided source code demonstrates a functional solution for managing hospital operations. However, ongoing efforts in data normalization, performance optimization, security enhancement, and user feedback integration are essential to ensure the system's effectiveness, reliability, and adaptability in meeting the dynamic needs of healthcare institutions.

OUTPUT SNAPSHOTS





GRC Hospital, established in 2001, is committed to delivering premium quality healthcare within reach of everyone at affordable costs. Our mission is to provide international standard care combined with exceptional value for money. With a team of highly qualified doctors, state-of-the-art facilities, and compassionate staff, we ensure the best possible treatment and comfort for our patients.

At GRC Hospital, we believe in the care you can trust. We offer a wide range of medical services, including cardiology, neurology, dermatology, surgery, and gastroenterology, supported by advanced diagnostic and therapeutic technologies. Our 24-hour emergency services and ambulance facilities ensure that help is always available when you need it the most.

GRC Hospital is not just about treatment; it's about creating a nurturing environment where patients and their families feel supported and confident in their care journey. Our patient-centric approach ensures personalized treatment plans tailored to individual needs.

Explore our specialized departments and find the best healthcare professionals dedicated to improving your health and well-being. With GRC Hospital, you are in safe hands.

OK

6.1 CONCLUSION

- In conclusion, the Hospital Management System (HMS) project, based on the provided source code, marks a significant advancement in healthcare management technology. Through its implementation, the system offers a robust platform for hospitals to streamline their operations, enhance patient care, and optimize resource utilization.
- Key highlights of the project include its functional capabilities, user friendly interface, and database connectivity. The system effectively handles tasks such as appointment scheduling, bed management, staff coordination, and emergency services, empowering hospital staff to efficiently manage daily operations.
- Moreover, the integration with MySQL database ensures reliable data storage and management, facilitating informed decision-making and improving data accuracy and accessibility. This strengthens the overall efficiency and effectiveness of hospital management processes.
- While the project demonstrates substantial progress, there is room for further refinement and enhancement. Areas such as data normalization, performance optimization, and security measures can be addressed to elevate the system's functionality, scalability, and security.
- In summary, the Hospital Management System project based on the provided source code presents a valuable solution for modern healthcare facilities, enabling them to deliver superior patient care, streamline operations, and adapt to evolving healthcare demands.

7.1 REFERENCES

1. Fitch K, Bernstein SJ, Aguilar MD, Burnand B, LaCalle JR, Lazaro P, et al. The RAND/UCLA Appropriateness Method User's Manual.
2. Iezzoni LI, Davis RB, Palmer RH, Cahalane M, Hamel MB, Mukamal K. et al. Does the Complications Screening Program flag cases with process of care problems? Using explicit criteria to judge processes. *Int J Qual Health Care*.
3. Davies S, Geppert J, McClellan M, McDonald KM, Romano PS, Shojania KG. Refinement of the HCUP Quality Indicators. Technical.
4. Balaraman, Premkumar, and Kalpana Kosalram. "E-Hospital Management & Hospital Information Systems-Changing Trends." *International Journal of Information Engineering & Electronic Business* 5.1 (2013).
5. Griffith, John R., and Kenneth R. White. "The revolution in hospital management." *Journal of Healthcare Management* 50.3 (2005).