**Hospital Management System**

Atharva Salitri (12310120), Sai Jadhav(12310331), Rudrani Sarangdhar (12310103), Rutuja Chaudhari(12311905)

***Abstract*—The Hospital Management System aims to streamline and digitize essential hospital operations such as patient registration, appointment booking, and medical record management. Implemented using MySQL for the backend database and Java/Swing for the frontend and logic layers, the system ensures efficient data handling and user-friendly interaction. Patients and doctors access the system via distinct login portals, enabling functionalities like appointment scheduling, medical history review, and prescription management. Built to prevent scheduling conflicts, the system checks doctor availability and breaks before confirming appointments. Doctors can access and update patient records, improving continuity of care. The ER diagram models key entities such as Patients, Doctors, Appointments, Medical History, and Schedules, capturing their relationships and attributes. This project enhances hospital workflow efficiency, reduces administrative overhead, and improves the patient experience through secure and centralized digital record management.**

***Keywords—Hospital, Database, MySQL, Java, Relational Database, GUI, Swing***

# Introduction

In today’s fast-paced world, hospitals manage a vast influx of patients daily, resulting in the generation and management of large amounts of sensitive data. This includes not only patient records but also appointment scheduling, medical histories, and treatment plans. Traditional, manual handling of such records often leads to inefficiencies, human errors, delays in patient care, and a significant administrative burden on hospital staff [1]. This issue is especially critical in healthcare settings, where data accuracy and timely access to information can directly affect patient outcomes and the quality of care provided [2]. Furthermore, the complexity of healthcare systems has increased, with diverse departments, treatments, and patient needs, making manual systems increasingly unsustainable [3].

As a solution to these challenges, a digitized Hospital Management System (HMS) has become essential. An HMS streamlines hospital operations, automating routine processes, and centralizing data management, leading to greater efficiency and accuracy [4]. By minimizing human error and reducing administrative tasks, a well-designed system allows healthcare providers to focus more on patient care and less on paperwork. It also enhances communication between departments, which is crucial for coordinated patient management [5].

The proposed Hospital Management System is designed to address these inefficiencies by providing a centralized, automated solution for managing appointments, patient records, medical histories, and more. Built using MySQL for robust backend data management, along with Java and Python for frontend and logic development, the system ensures high performance, data security, and seamless user interaction. The choice of MySQL for data management ensures reliable data storage and fast query processing, essential for handling large datasets common in hospital environments [6].

The system features separate login portals for patients and doctors, ensuring role-based access and secure authentication. This ensures that sensitive patient information is protected, and only authorized personnel can access certain records or functionalities [7]. Patients can book, update, or cancel appointments based on real-time doctor availability, which is cross-verified with the doctor's existing schedule and break times to prevent overlapping or missed slots. Doctors, on the other hand, can manage their appointments, access patient medical histories, record diagnoses, and issue prescriptions directly through the system [8]. This significantly reduces administrative work and minimizes the risk of scheduling errors.

A comprehensive Entity-Relationship (ER) model has been implemented to represent the major components of the system, including Patients, Doctors, Appointments, Medical History, and Schedules. The interrelations between these entities support efficient data flow and accurate information tracking within the system [9]. This model ensures that all relevant data, such as patient demographics, doctor availability, and treatment histories, are effectively integrated, providing a holistic view of each patient's medical journey. The ER model serves as a blueprint for organizing and connecting data, ensuring consistency and accuracy in patient management.

By automating routine processes and ensuring effective communication between doctors, patients, and administrative staff, the Hospital Management System improves overall healthcare delivery and operational efficiency. It reduces waiting times, improves data accuracy, and enhances the overall patient experience, leading to better outcomes and higher satisfaction levels for both patients and healthcare providers [10].

# Proposed Methodolgy

The proposed Hospital Management System is developed using a modular and database-centric approach, ensuring efficient handling of hospital operations. The system uses MySQL, a powerful Relational Database Management System (RDBMS), to store and manage structured data. The database schema is designed using Entity-Relationship modeling, with normalization up to the third normal form (3NF) to eliminate data redundancy and maintain consistency. Each core module—Patients, Doctors, Appointments, Medical History, Diagnosis—is implemented as a relational table with clearly defined primary keys and foreign key constraints to preserve referential integrity.

The system supports complex SQL queries and joins for retrieving and updating interrelated data efficiently. Transactions are implemented to ensure the ACID properties (Atomicity, Consistency, Isolation, Durability), providing reliable data operations even during simultaneous access by multiple users. The appointment scheduling mechanism incorporates logical checks on doctor availability, break times, and existing bookings to prevent double-booking. These conditions are evaluated dynamically before inserting or updating appointment records.

**Appointment Validity Condition:**

Valid\_Appointment = (Doctor\_Available) ∧ (Time\_Slot\_Not\_Booked) ∧ (Within\_Working\_Hours) …(i)

This methodology ensures a secure, scalable, and conflict-free environment for managing real-time hospital activities.

# Implementation details

The implementation of the Hospital Management System is structured into distinct modules, each corresponding to key functionalities depicted in the ER diagram and functional requirements. The project uses MySQL as the backend database to store, manage, and query patient, doctor, appointment, and medical history data. The frontend and business logic are developed using Java (Swing framework) for the graphical interface and Python for supplementary data handling and backend processes. This combination ensures a smooth, secure, and responsive user experience.

### 

### User Authentication Module

This module provides secure login and authentication functionality. It includes two types of users: **Patients** and **Doctors**, each with separate login portals. The system verifies credentials using hashed passwords stored in the database. Role-based access control ensures users can only access their respective dashboards and features.

*Patient Dashboard Module*

Once logged in, patients can:

* View available doctors and appointment slots.
* Book new appointments based on doctor availability.
* View, update, or cancel existing appointments.
* Access their profile and medical history. This module interacts with the Appointments and Medical History tables, enforcing rules to avoid scheduling overlaps and maintain data consistency.

*Doctor Dashboard Module*

Doctors have access to:

* View their daily and upcoming appointments.
* Cancel or reschedule appointments.
* Access patient medical profiles.
* Add diagnoses, prescriptions, and update patient history. The system ensures that only doctors assigned to a patient can modify their history, ensuring medical data integrity and privacy.

*Appointment Management Module*

The core of the system, this module handles:

* Real-time appointment booking with conflict detection.
* Integration with the **Schedule** table to avoid booking during breaks or pre-existing appointments.
* Updating or deleting appointments which automatically frees slots for other patients.

*Medical History and Diagnosis Module*

This module allows doctors to maintain detailed medical histories for each patient. It supports:

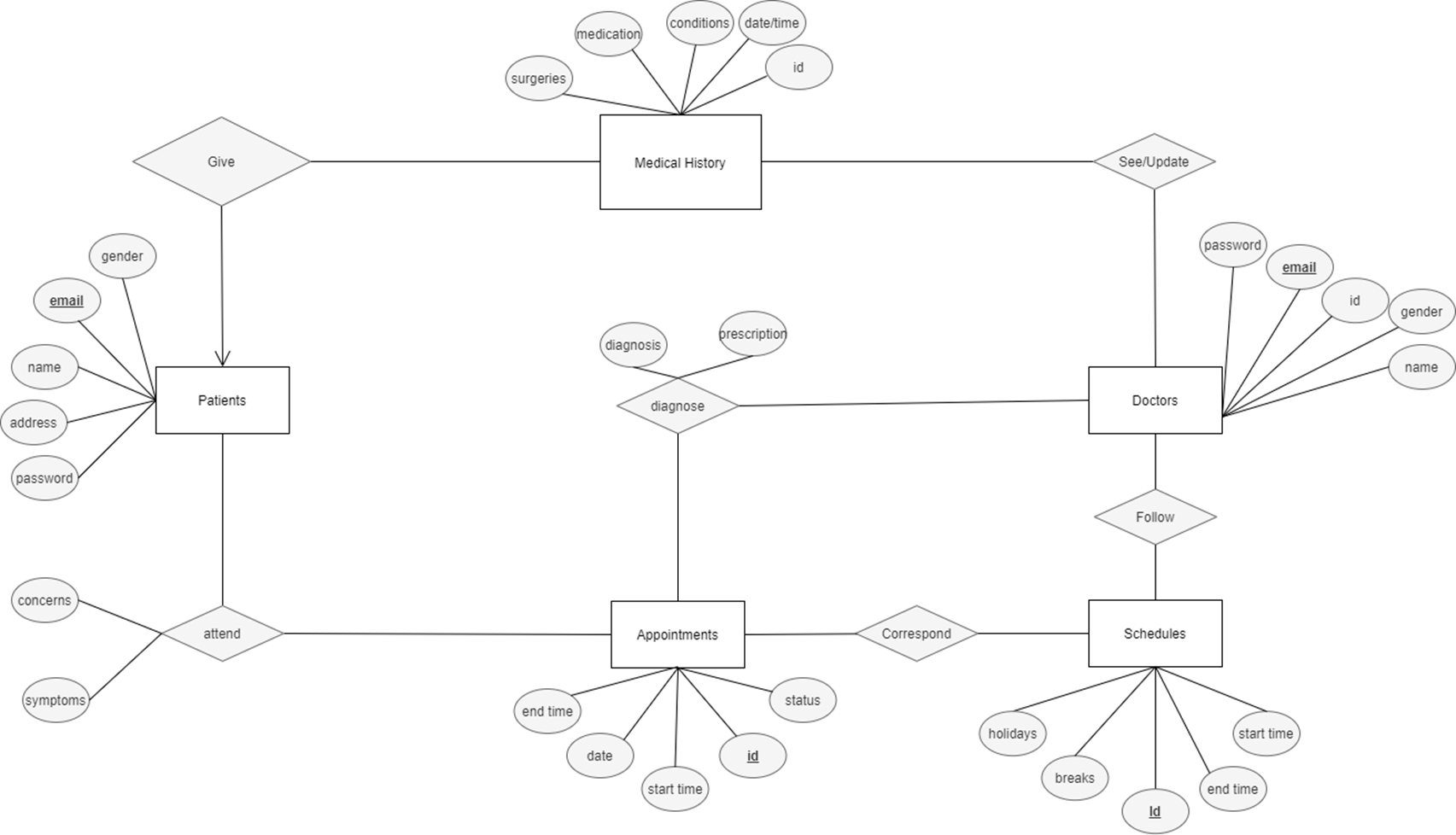
* Recording of symptoms, diagnosis, prescriptions, and past treatments.
* Timestamped entries to maintain chronological order.
* Linking to relevant doctors for accountability and future reference.

### Doctor Schedule Management Module

Each doctor has a configurable schedule stored in the **Schedule** table. The system:

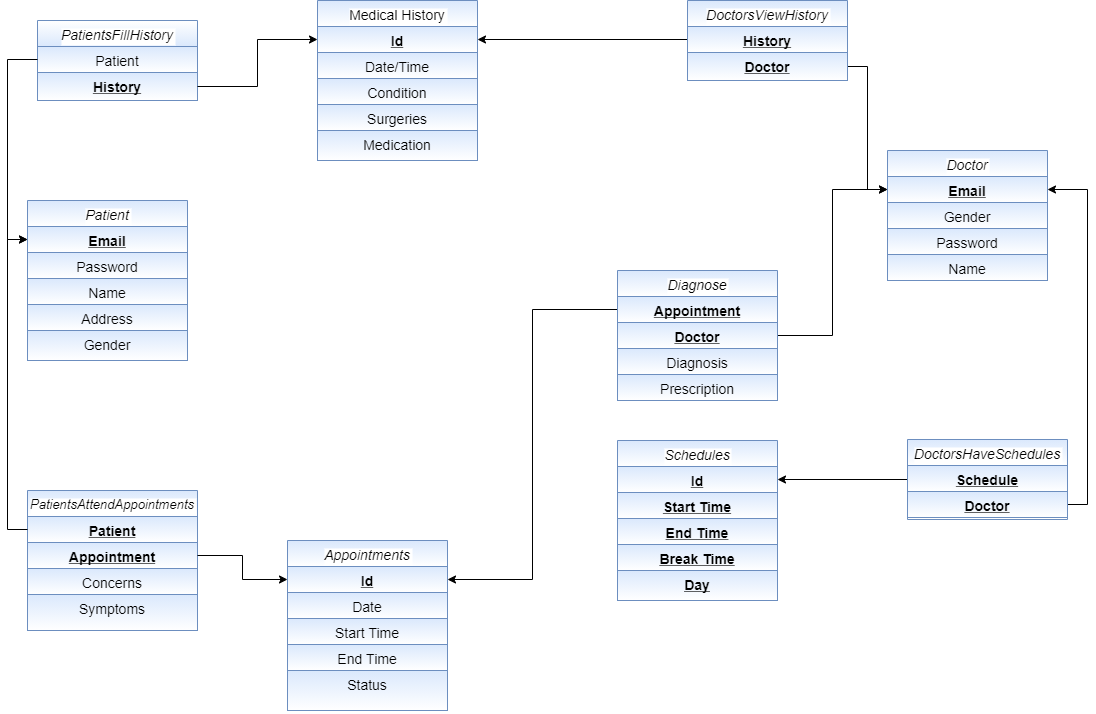
* Stores working hours, break times, and off days.
* Dynamically checks availability during appointment booking.
* Prevents double-booking and ensures smooth scheduling.

Fig. 1 shows the ER diagram of the project, which is a visual representation of how different entities (objects, concepts) in our system relate to each other.



1. ER Diagram of the project.

Fig. 2 shows the Relational Schema, which is a compelling visual representation of our database system's structure and organization.



1. ER Diagram of the project.

Table 1 which is the Entity and Attribute Mapping table provides a structured overview of the key components of the Hospital Management System. It highlights each entity's core attributes, associated relationships, and specific roles within the system.

This mapping ensures clarity in database design, supports efficient data flow, and strengthens the overall system architecture.

1. ENTITY AND ATTRIBUTE MAPPING

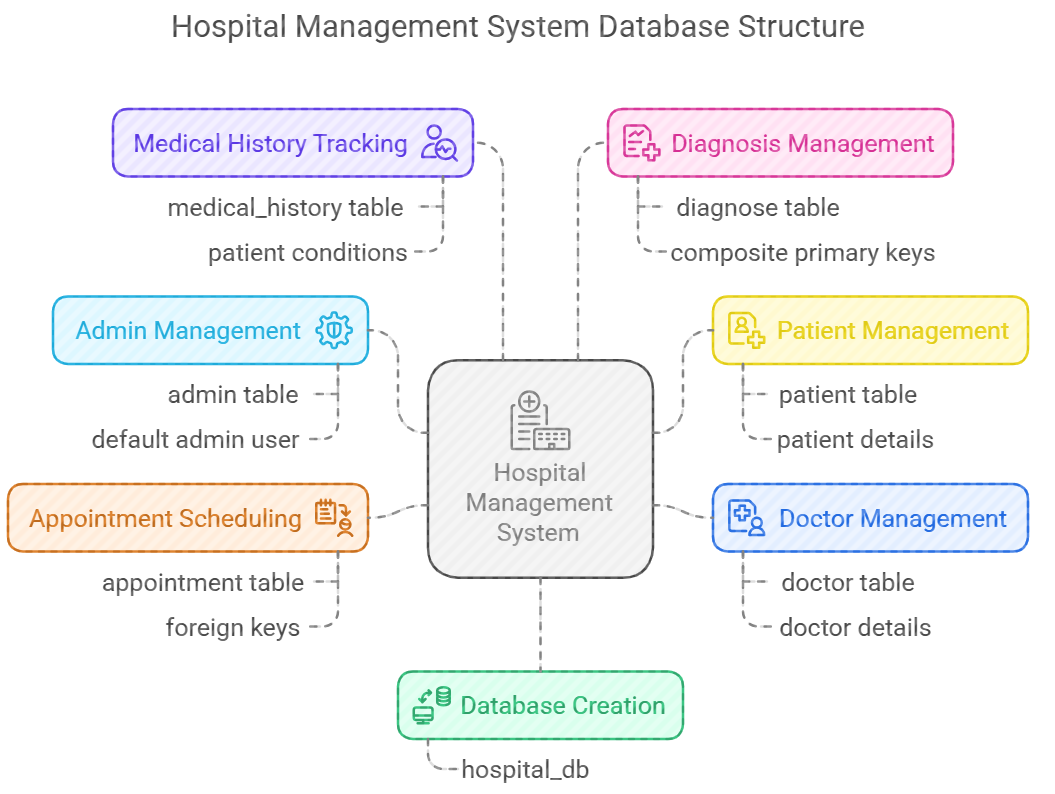
| **Table Name** | **Attributes Associated** | | |
| --- | --- | --- | --- |
| ***Purpose*** | ***Associated Tables*** | ***Attributes*** |
| Patients | Stores patient details and links them to medical records and appointments. | Appointments, Medical History | name, email, gender, address, password |
| Doctors | Maintains doctor profiles, appointments, and allows medical updates | Appointments, Medical History, Schedules | id, name, email, gender, password |
| Appointments | Handles patient-doctor meetings with status tracking and scheduling logic. | Patients, Doctors | id, date, time, status |
| Medical History | Maintains detailed patient medical records updated by admin. | Patients | id,conditions,medication,surgeries,date/time |
| Diagnose | Manages diagnosis of patients updated by doctors. | Patients,Doctors | id, date,condition,surgeries,medication |

Normalisation and Functional Dependencies

1. Admin Table:
   * 1NF: All attributes (id, username, password) are atomic, and there are no repeating groups.
   * 2NF: The primary key is a single attribute (id), ensuring no partial dependencies.
   * 3NF: There are no transitive dependencies; all non-key attributes (username, password) depend directly on the primary key.
2. Patient Table:
   * 1NF: Attributes like name, age, gender, contact, and address are atomic, with no repeating groups.
   * 2NF: The primary key (id) is a single attribute, so all non-key attributes are fully dependent on it.
   * 3NF: No transitive dependencies exist; all attributes directly depend on the primary key.
3. Doctor Table:  
   * 1NF: All fields (id, name, specialization, contact, available) are atomic.
   * 2NF: The primary key (id) ensures no partial dependencies as it is a single attribute.
   * 3NF: No transitive dependencies; each non-key attribute depends only on the primary key.
4. Appointment Table:  
   * 1NF: Attributes like patient\_id, doctor\_id, appointment\_date, and appointment\_time are atomic, with no repeating groups.
   * 2NF: The primary key (id) ensures that all non-key attributes are fully dependent on it. Foreign keys (patient\_id and doctor\_id) maintain referential integrity without introducing partial dependencies.
   * 3NF: No transitive dependencies exist; all non-key attributes depend directly on the primary key.
5. Medical History Table:  
   * 1NF: Attributes such as date, conditions, surgeries, and medication are atomic, with no repeating groups.
   * 2NF: The primary key (id) ensures full dependency for all non-key attributes.
   * 3NF: No transitive dependencies exist; all attributes directly depend on the primary key.
6. Diagnose Table:  
   * 1NF: All attributes (appointment\_id, doctor\_id, diagnosis, and prescription) are atomic, with no repeating groups.
   * 2NF: The composite primary key (appointment\_id and doctor\_id) ensures that all non-key attributes depend on both components of the key, avoiding partial dependencies.
   * 3NF: No transitive dependencies exist; each attribute depends directly on the composite primary key.

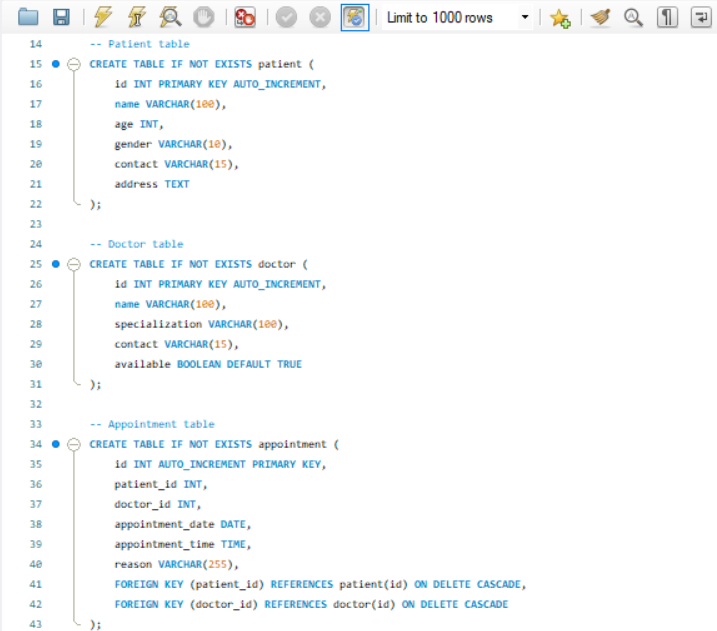
# FLOWCHART

The flowchart in Fig 3. represents the working process of Hospital Management System starting from database creation to medical diagnosis.



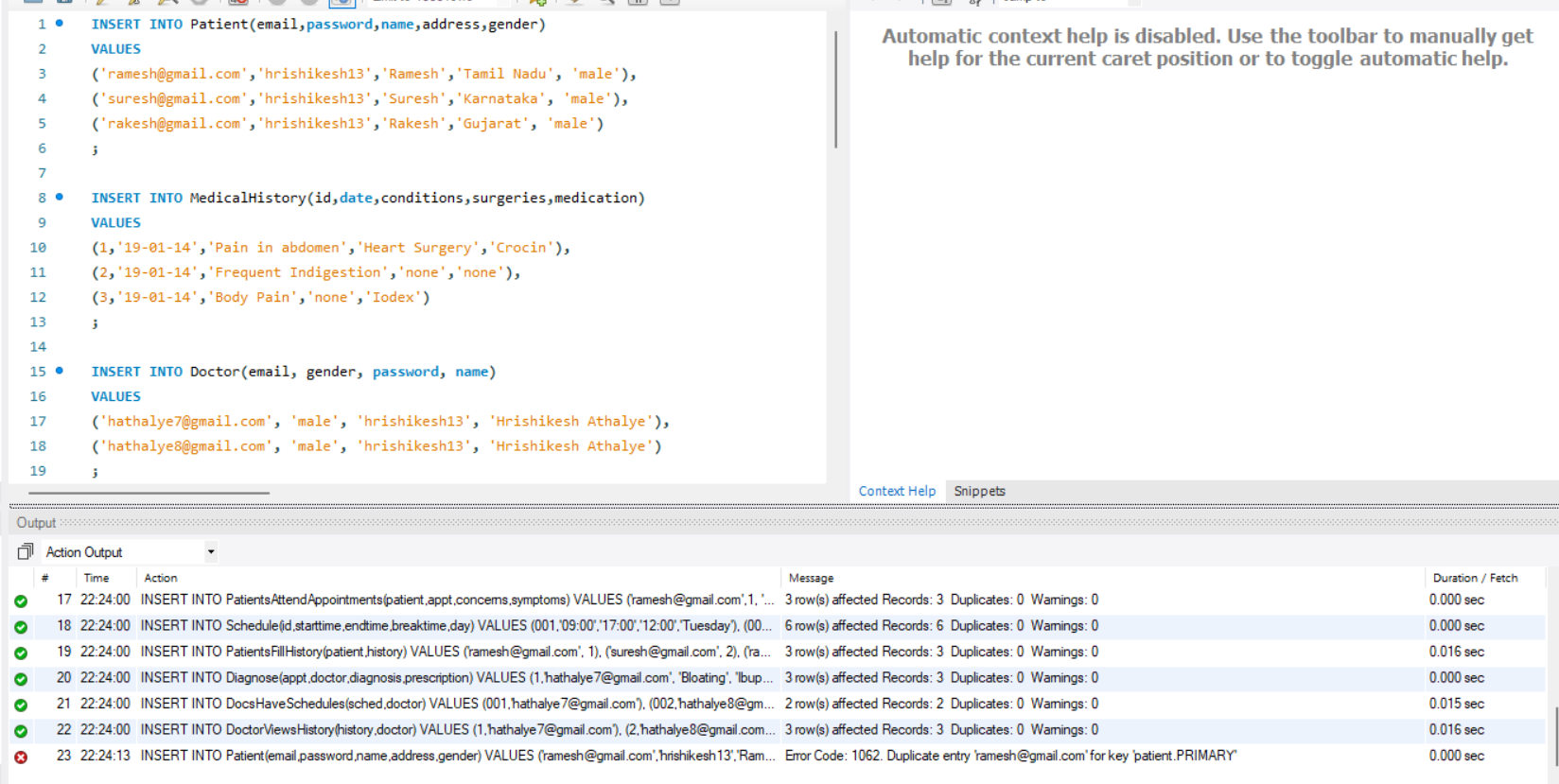
1. System Flowchart Diagram

This hospital management system is designed to streamline hospital operations by organizing essential data into structured tables. It includes modules for managing admin credentials, patient information, doctor details, appointment scheduling, medical history records, and doctor diagnoses. Each table is interconnected through relationships that ensure accurate tracking of patient visits and treatments. The system supports efficient data management, enabling hospitals to maintain comprehensive records of patients, doctors, appointments, and medical diagnoses in a centralized database.



1. Creation of tables for three main uses

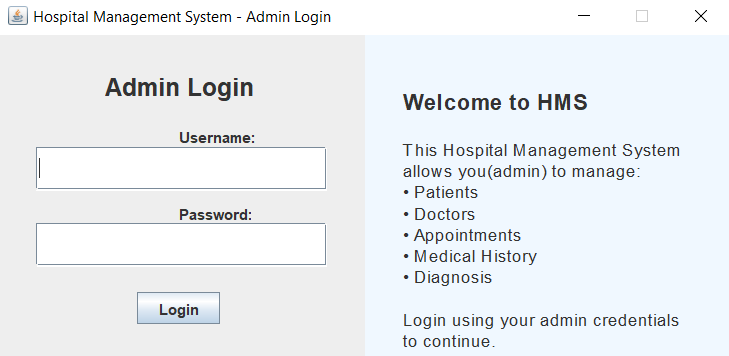
It displays the SQL schema creation for three relational tables: Diagnose, DocsHaveSchedules, and DoctorViewsHistory. Each table links the Doctor table using the email field as a foreign key and includes additional references to other tables such as Appointment, Schedule, and MedicalHistory. All foreign keys have cascading deletes, ensuring that any deletions in the referenced tables automatically remove associated rows. The primary keys are composite keys, made up of the foreign key pairs, to uniquely identify each relationship. This normalized design efficiently captures the relationships between doctors, their diagnoses, schedules, and the medical history they view.



1. Taking user inputs for patient information

It shows SQL INSERT statements for populating the Patient, MedicalHistory, and Doctor tables, along with the output window below displaying the results of various data insertions. The output log confirms successful insertions into multiple relationship tables like PatientsAttendAppointments, Schedule, PatientsFillHistory, and others, with each showing the number of rows affected. However, a duplicate entry error (Error Code: 1062) is reported for the Patient table due to an attempt to reinsert a record with an already existing email, which violates the primary key constraint. This error highlights the importance of ensuring unique primary key values during insertions.

Fig. 6. shows the login page of the Hospital Management System which was designed to provide a secure and intuitive interface for administrators. During testing, the system successfully validated user credentials against the database, ensuring robust access control. The graphical user interface (GUI) demonstrated seamless usability, with clear labels and responsive input fields for username and password. The inclusion of error messages for invalid or incomplete input enhanced user experience by guiding administrators to correct mistakes efficiently. Additionally, the integration of SQL queries ensured accurate authentication, while the modular design facilitated future scalability. This functionality forms a critical foundation for accessing the broader management system securely and efficiently.



1. Login Page Interface

Fig. 7. The dashboard of the Hospital Management System was successfully implemented to serve as a central hub for navigating various management functionalities. The design utilized a clear and organized layout, incorporating a header for branding, a central panel for navigation buttons, and a footer for copyright information. The use of a BorderLayout ensured efficient space management, while the GridLayout for buttons provided a structured and user-friendly interface. Each button was linked to specific management panels, such as patient, doctor, appointment, medical history, and diagnosis management, facilitating easy access to these critical system components. The inclusion of tooltips enhanced user experience by providing additional context for each button's functionality. Overall, the dashboard effectively streamlined access to the system's core features, improving usability and workflow efficiency.



1. Management Dashboard Interface

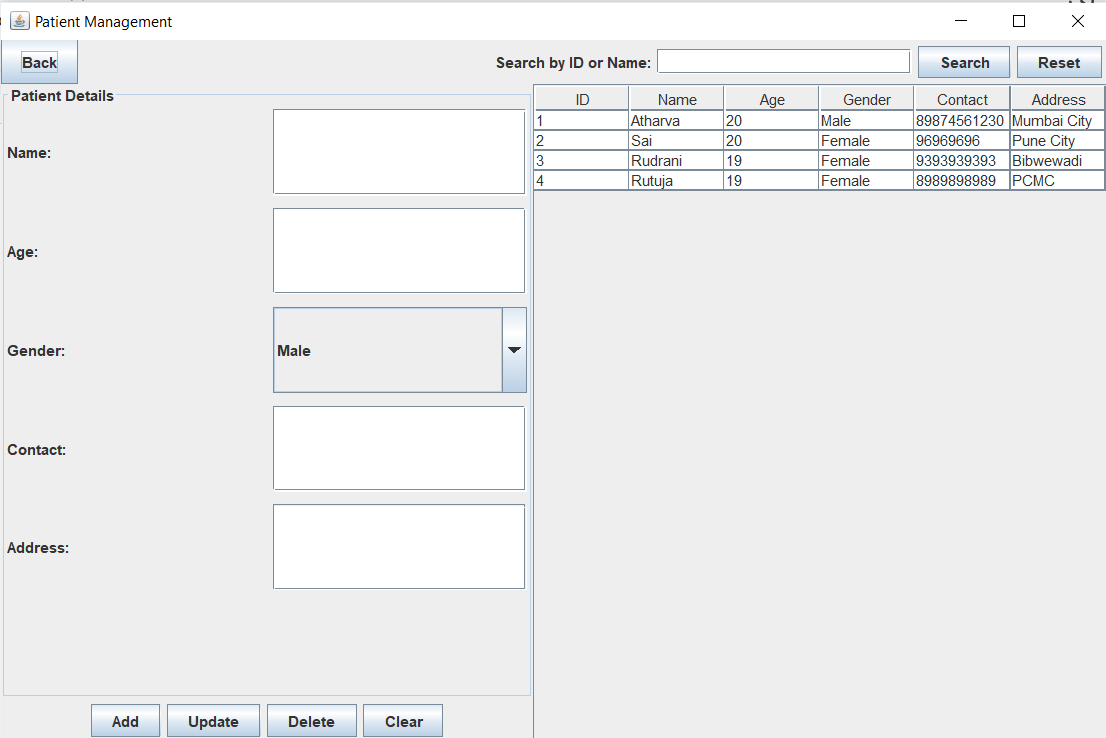
Fig. 8.(a-e) shows the results of the main 5 tables in GUI form. The Hospital Management System integrates comprehensive modules for managing Patients, Doctors, Appointments, Medical History, and Diagnosis, each featuring full CRUD (Create, Read, Update, Delete) capabilities along with advanced search and filter options. These modules collectively enhance hospital operations by enabling efficient and accurate data management. Users can easily create new records, update existing information, retrieve detailed data, and delete obsolete entries, all while maintaining data integrity through secure database interactions.

The search and filter functionalities allow quick access to specific records based on various criteria such as patient name, doctor specialization, appointment date, or diagnosis type, improving usability and reducing time spent on data retrieval.

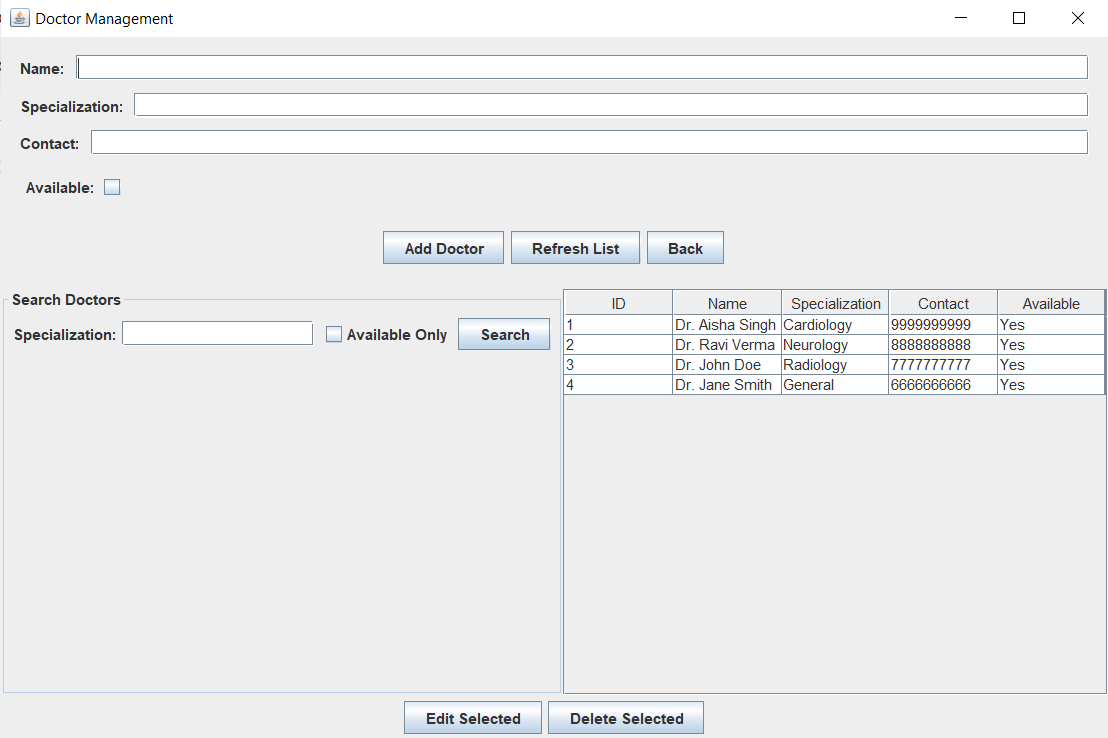
Furthermore, the modules are interconnected, ensuring seamless data flow—for instance, appointments link patients and doctors, while medical history ties closely with diagnosis records—facilitating comprehensive and coherent information management. The user interface is designed to be intuitive and responsive, with clear layouts and helpful tooltips that guide users through the system’s features.

This integration not only streamlines administrative workflows but also supports scalability, allowing the system to adapt to increasing data volumes without sacrificing performance.

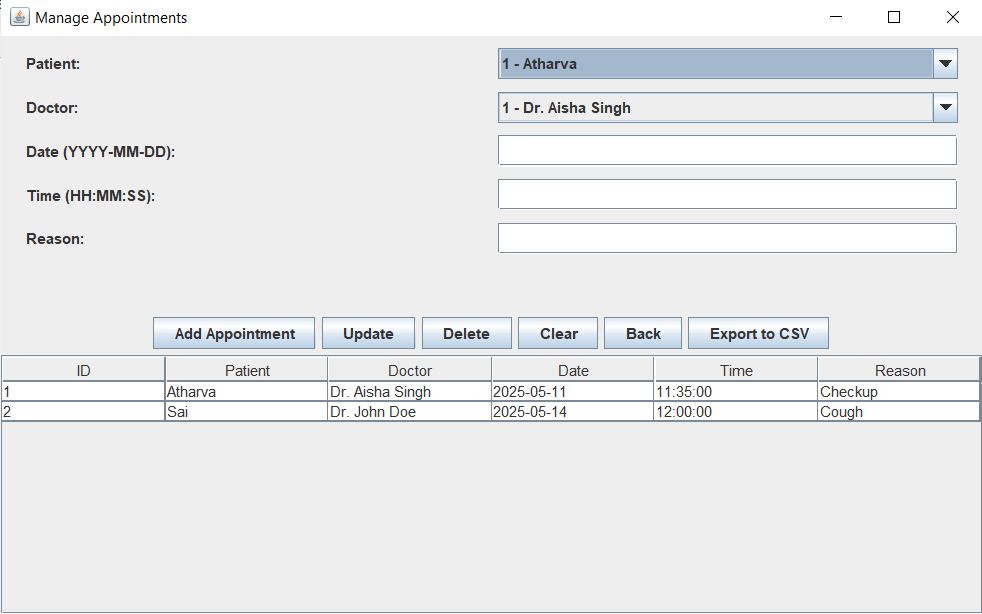
Overall, these modules form a robust foundation for managing critical healthcare information efficiently, supporting better decision-making and enhancing the quality of patient care.



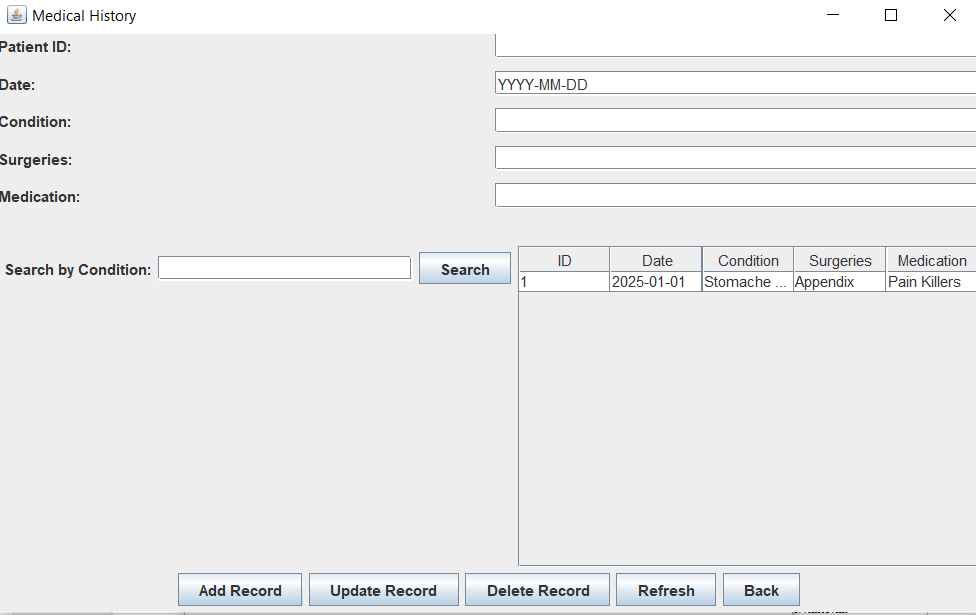
(a)



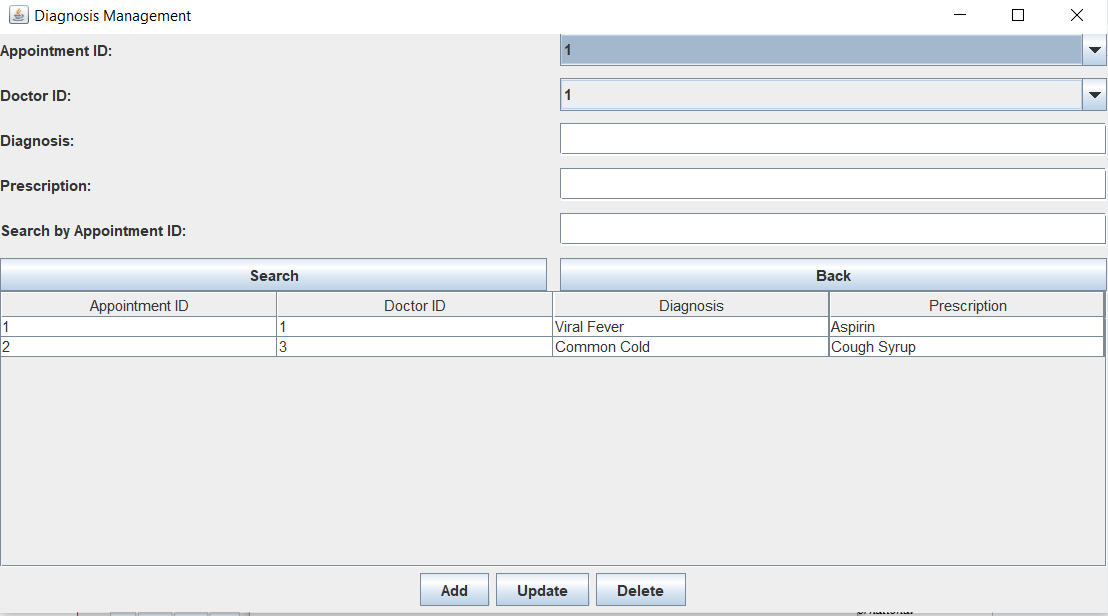
(b)



(c)



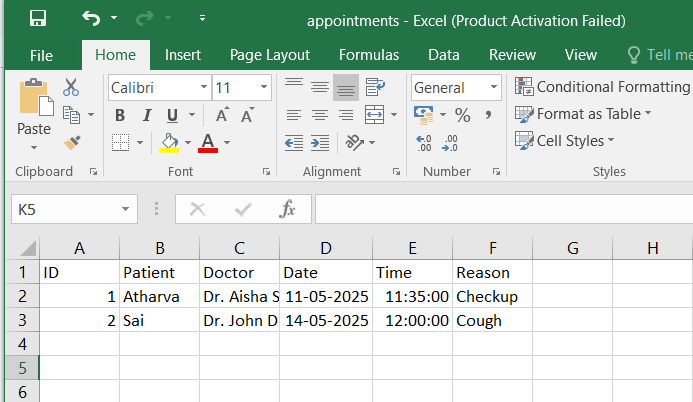
(d)



(e)

1. User Interfaces to perform CRUD and Search operations on various tables

Fig. 9. displays the result of exporting appointments to csv. The Appointment module of the Hospital Management System includes a convenient feature allowing users to export appointment data to a CSV file. This functionality is accessible via a dedicated button, providing a straightforward method for administrators to transfer appointment records into a widely compatible format. By exporting data to CSV, users can easily analyze appointment trends, manage schedules offline, or integrate the data into external systems for further processing. This export capability enhances the system's utility by facilitating data portability and supporting more flexible data analysis and reporting processes.



1. Exporting Appointments to CSV

# Conclusion and Future Scope

This project report presented the development of a Hospital Management System using MySQL, Java, and Python, addressing key hospital operations such as appointment scheduling, patient record management, and medical history tracking. The system includes distinct interfaces for patients and doctors, ensuring secure and role-specific functionality. Through the ER model, core entities and their interactions were clearly defined, contributing to a structured and efficient database design. The project successfully achieved its objectives by improving hospital workflows, reducing manual errors, and enhancing patient-doctor interaction. However, the current system is limited to basic functionalities and does not yet support advanced analytics or multi-hospital integration.

In terms of Future Scope, the system can be expanded with several advanced modules, such as automated billing, insurance claim processing, lab report management, and pharmacy integration. Features like patient feedback systems, AI-based diagnosis suggestions, and chatbot support for appointment booking can significantly enhance user experience. Migrating the platform to a full-stack web application using frameworks like Django or Spring Boot, along with a cloud-hosted database, can ensure better scalability, performance, and accessibility. Additionally, mobile app integration can make the system more versatile and patient-friendly.

##### References

1. Balaraman, Premkumar, and Kalpana Kosalram. "E-hospital management & hospital information systems-changing trends." *International Journal of Information Engineering and Electronic Business* 5.1 (2013): 50.
2. Sharma, Manoj, and Sunil Sharma. "Design and Implementation of Hospital Management System Using Database." *International Journal of Advanced Research in Computer Science and Software Engineering* 5.3 (2015): 112-118.
3. Raj, R. M., and S. P. Shukla. "Design of a Hospital Management System." *Journal of Computer Science and Engineering* 4.6 (2012): 88-92.
4. Kumar, Suraj, and Raghavendra Tiwari. "Database Management Systems for Healthcare Applications: A Survey." *International Journal of Computer Applications* 59.13 (2012): 23-27.
5. Jha, Sushant, et al. "Implementation of Hospital Information System Using DBMS." *International Journal of Computer Science and Mobile Computing* 3.3 (2014): 23-28.
6. Singh, Bhupinder, and Ranjit Kaur. "Development of an Effective Hospital Management System." *International Journal of Engineering Research and Applications* 4.3 (2014): 1-6.
7. Suresh, K. N., and Chandra S. Reddy. "A Review of Hospital Management System with Database Management." *International Journal of Computer Science and Technology* 5.2 (2014): 45-50.
8. Bari, Shyam S., and Deepak B. Joshi. "Hospital Management System: A Case Study." *International Journal of Computer Science and Information Technologies* 4.2 (2013): 112-117.
9. Patel, Vishal, and Mitali R. Mehta. "Design and Development of an Efficient Hospital Management System Using DBMS." *International Journal of Computer Applications* 40.10 (2012): 12-18.
10. Gupta, Vishal, and Sumit Bhargava. "Hospital Management System with SQL Database Integration." *International Journal of Engineering and Technology* 3.4 (2015): 56-60.
11. Madhusree, S., and S. A. Nair. "Hospital Management System: Concept and Design." *International Journal of Computer Science and Engineering* 3.1 (2011): 70-73.
12. Verma, Pawan, and Arun Kumar. "A Review on Hospital Management System with Database Management Techniques." *International Journal of Advanced Research in Computer Science* 3.2 (2015): 78-83.
13. Kaur, Navdeep, and Harpreet Kaur. "Analysis of Hospital Management System Using DBMS." *International Journal of Engineering and Technology* 5.7 (2014): 92-97.
14. Vijayakumar, M., and Preetha R. "An Overview of Hospital Management Systems and Technologies." *Journal of Computer Science* 5.2 (2013): 47-53.
15. Jain, Sandeep, and Nidhi Ghosh. "Design and Development of Hospital Management System Using DBMS: A Case Study." *International Journal of Computer Science and Applications* 4.6 (2014): 102-10