**Hospital Management System**

Atharva Salitri, Sai Jadhav, Rudrani Sarangdhar, Rutuja Chaudhari

Vishwakarma Institute of Technology, Pune

*Abstract*— The Hospital Management System is designed to streamline and digitize essential hospital operations such as patient registration, appointment booking, and medical record handling using Object-Oriented Programming principles in Java. The system utilizes Java classes and objects to represent core components such as Patients, Doctors, Appointments, and Medical Records, promoting modularity and reusability. Implemented using Java Swing for the graphical user interface and core Java for business logic, the system provides a user-friendly experience through distinct login portals for patients and doctors. It incorporates functionality for appointment scheduling, conflict resolution by checking doctor availability and breaks, as well as features for reviewing and updating medical histories. Through the use of encapsulation, inheritance, and polymorphism, the system ensures maintainable and scalable code architecture. This object-oriented approach not only enhances the efficiency of hospital workflows but also simplifies system upgrades and maintenance, resulting in reduced administrative overhead and an improved patient care experience.

*Keywords—* *Hospital Management, Java, Object-Oriented Programming, Classes, Objects, GUI, Swing, Encapsulation, Inheritance, Polymorphism*

# Introduction

In today’s fast-paced world, hospitals manage a vast influx of patients daily, resulting in the generation and handling 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, with the increasing complexity of healthcare operations—spanning multiple departments, patient needs, and medical services—manual systems are no longer sustainable or scalable [3].

To address these challenges, an automated and modular Hospital Management System (HMS) designed using Object-Oriented Programming (OOP) in Java has become an essential solution. The use of OOP facilitates better system organization through features like encapsulation, inheritance, and polymorphism, allowing the design to mirror real-world entities and behaviors such as patients, doctors, appointments, and medical records [4]. By automating repetitive administrative tasks and streamlining data access, an OOP-based HMS significantly reduces human error and allows healthcare professionals to focus more on patient care [5]. It also enhances inter-departmental communication through structured class interactions, promoting coordinated and efficient healthcare delivery [6].

The proposed Hospital Management System is built entirely in Java, using Java Swing for the graphical user interface and core Java classes for backend logic. The system provides distinct login portals for doctors and patients, ensuring role-based access and secure authentication. This protects sensitive patient information and allows users to interact with only the functionalities relevant to their role [7]. Patients can book, modify, or cancel appointments in real-time, based on doctor availability that is verified against predefined schedules and break times to prevent conflicts. Doctors can manage their appointments, review patient histories, record diagnoses, and generate prescriptions through the system interface [8]. These functionalities are implemented using polymorphic methods, promoting code reusability and extensibility.

The core design includes well-defined Java classes representing key system entities such as Patient, Doctor, Appointment, MedicalHistory, and Schedule. These classes encapsulate data and behaviors relevant to each component, interacting with each other through class associations and method calls, simulating real-world hospital workflows [9]. The system architecture supports scalability and maintainability, allowing new features or entities to be added with minimal changes to existing code.

By automating routine processes and enabling structured interaction between patients, doctors, and hospital administration, the Object-Oriented Hospital Management System improves data accuracy, reduces delays, and enhances the overall healthcare experience. This contributes to better patient outcomes and increased satisfaction for both patients and providers [10].

# Literature Review

Hospital Management Systems (HMS) are crucial for managing and automating healthcare operations, enhancing efficiency in patient management, administrative tasks, and resource allocation. Java, as a robust and versatile programming language, paired with MySQL for data storage, has become a common choice for developing such systems.

Khan et al. (2019) developed a hospital management system using Java and MySQL, focusing on real-time tracking of patient records and streamlined administrative workflows. The authors demonstrated the system's ability to manage a large volume of data and provide user-friendly interfaces for hospital staff. However, the system's lack of mobile support for remote access limited its scalability in modern healthcare environments [1].

Patel and Desai (2020) proposed a modular HMS architecture that utilized MySQL’s relational database management capabilities to handle patient data securely. The system was praised for its scalability, especially in handling large datasets. However, scalability issues arose when large numbers of concurrent users accessed the system simultaneously, leading to delays in data retrieval and slower performance [2].

Singh et al. (2021) focused on real-time data updates in an HMS, integrating Java Swing for the front-end interface. This approach was effective for dynamic data updates but faced limitations regarding user adoption due to the complex and non-intuitive interface. Despite this, the system demonstrated the power of Java for backend logic and the efficiency of MySQL in data storage and retrieval [3].

The proposed Hospital Management System is developed using an object-oriented, modular approach, ensuring the reusability, maintainability, and scalability of code. The system is implemented in Java, following core OOP principles such as encapsulation, inheritance, and polymorphism to model real-world hospital entities effectively.

Each core component—Patients, Doctors, Appointments, Medical History, Diagnosis—is represented as a Java class with well-defined attributes and methods. These classes encapsulate their respective data and expose functionality through public methods, ensuring data integrity and controlled access.

Inheritance is used to define common behaviors; for instance, both Doctor and Patient classes inherit from a common User superclass that handles shared attributes like name, email, and login credentials. Polymorphism allows flexibility in managing different user interactions via method overriding where necessary.

Key scheduling logic is implemented through Java methods that evaluate doctor availability, break timings, and existing bookings before confirming an appointment. The system uses conditional logic and time comparison functions in Java to prevent scheduling conflicts.

# Implementation details

The implementation of the Hospital Management System follows a class-based, modular structure that aligns with object-oriented design patterns. Each module corresponds to real-world hospital operations and is modeled using Java classes and interfaces.

**User Authentication Module**

This module manages secure login functionality for two user roles—Doctors and Patients. The LoginManager class handles credential validation. Passwords are stored in hashed format and compared using hashing libraries in Java for security. Role-based access is implemented using polymorphic login behavior based on user type [3].

**Patient Dashboard Module**

Upon successful login, the patient is directed to a dashboard that allows:

* Viewing available doctors and slots.
* Booking or canceling appointments.
* Accessing personal medical history.

The logic is handled by the PatientDashboard class, which interfaces with the AppointmentManager and MedicalHistory classes. Data validation is performed before each operation to ensure correctness.

**Doctor Dashboard Module**

Doctors interact with the system through a dedicated dashboard allowing them to:

* View and manage appointments.
* Update diagnoses and medical histories.
* Access individual patient records.

Security constraints are enforced in the DoctorDashboard class, ensuring only the assigned doctor can update a patient’s records, preserving data confidentiality [4].

**Appointment Management Module**

This module includes classes like Appointment, Schedule, and AppointmentManager. Features include:

* Real-time appointment conflict checks.
* Time-slot validation methods.
* Appointment rescheduling or cancellation handling.

Scheduling logic uses Java’s LocalDateTime and Duration classes for accurate time computations.

**Medical History and Diagnosis Module**

The MedicalHistory and Diagnosis classes manage:

* Detailed chronological records.
* Entries linked to doctors and timestamps.
* Controlled access and accountability tracking.

Encapsulation ensures that medical data can only be modified via authorized methods, protecting patient data [5].

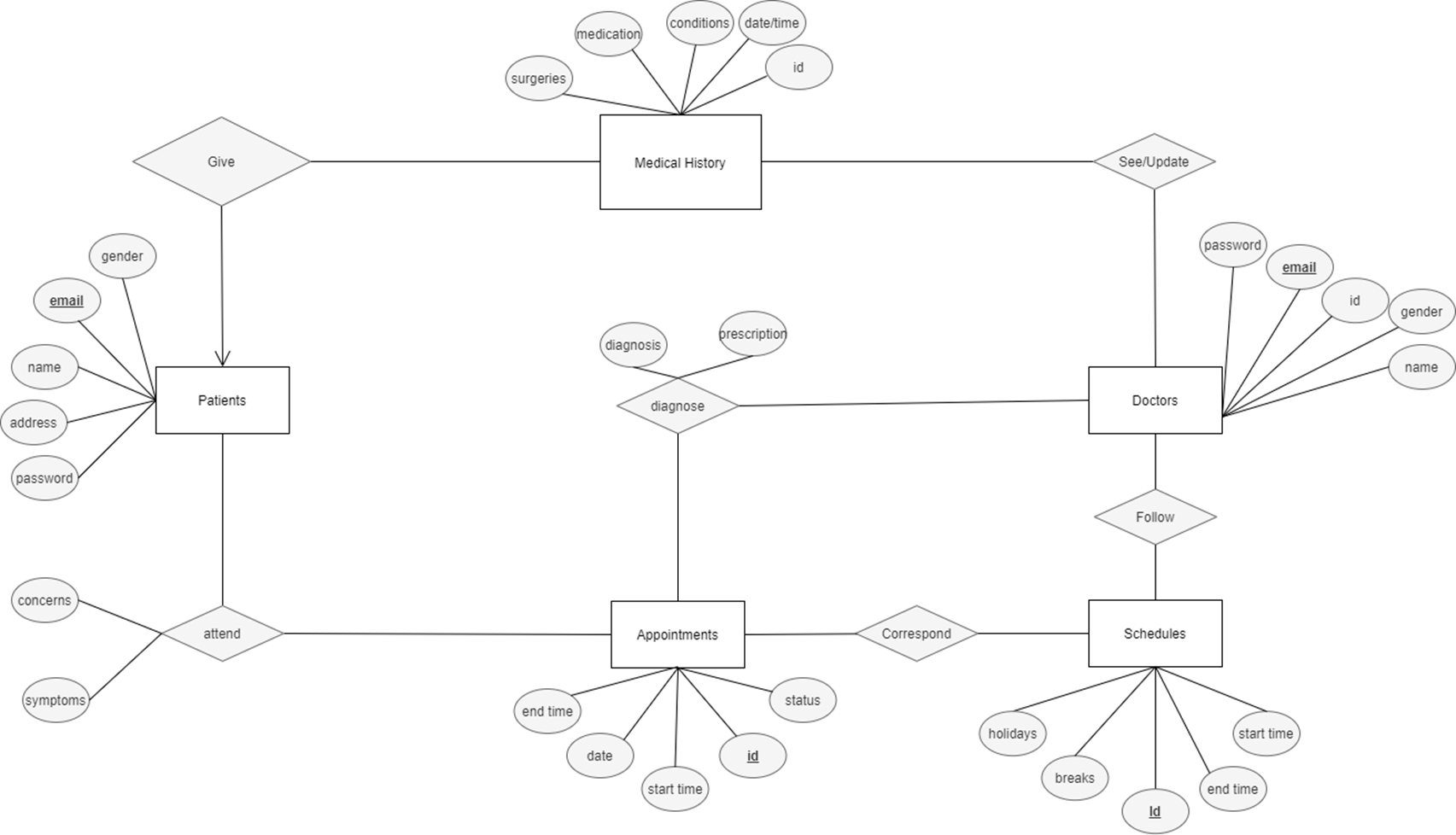
**Doctor Schedule Management Module**

The Schedule class maintains doctor availability by:

* Defining working hours and breaks.
* Updating the doctor’s calendar dynamically.
* Validating time slots during appointment booking.

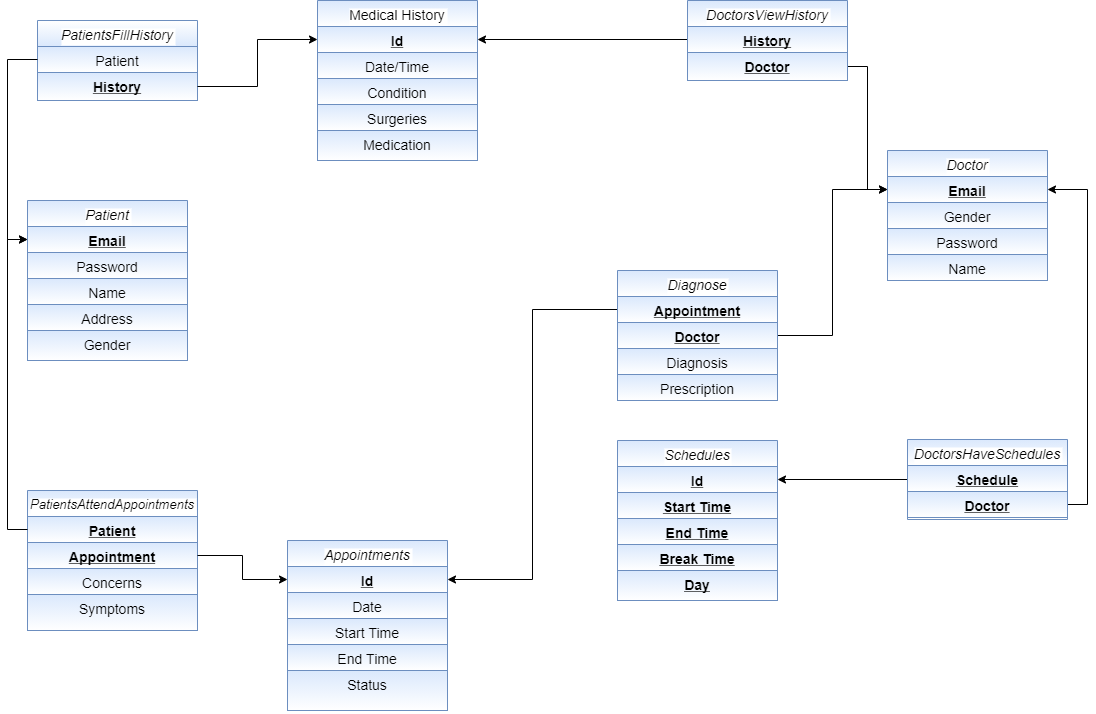
The system ensures synchronization between the doctor’s availability and patient requests through real-time checks and exception handling mechanisms.

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

| **Class Name** | **Role in System** | **Key Attributes & Methods** |
| --- | --- | --- |
| Patient | Manages patient details, appointments, and profile | name, email, bookAppointment() |
| Doctor | Handles appointments, diagnosis, and history updates | specialization, viewAppointments(), diagnosePatient() |
| Appointment | Represents patient-doctor meetings | appointmentDate, timeSlot, status |
| MedicalHistory | Records and stores patient treatment and diagnosis history | addEntry(), getHistory() |
| Diagnosis | Manages diagnosis records | condition, prescription, timestamp |
| LoginManager | Handles login validation and session control | authenticateUser() |

**1. Encapsulation:**

* **Definition**: Each class in the system (like Admin, Patient, Doctor, etc.) encapsulates its data using private attributes, ensuring that access and modification of data is controlled through public methods.
* **Usage**: All attributes such as id, name, contact are kept private, and access is provided through public getter and setter methods.
* **Importance**: Encapsulation protects the internal state of an object from unintended interference and misuse, ensuring data integrity and security.

**2. Abstraction:**

* **Definition**: The system abstracts real-world hospital entities into classes like Patient, Doctor, Appointment, and so on.
* **Usage**: Each class only exposes relevant information and behaviors, hiding the internal implementation details from the user.
* **Importance**: It reduces complexity and allows developers to work with higher-level concepts while managing underlying functionality effectively.

**3. Class and Object Design:**

* **Definition**: Java classes are designed to represent entities, and their instances (objects) represent individual records.
* **Usage**: Classes like Patient, Doctor, and MedicalHistory serve as blueprints for objects created in the system.
* **Importance**: Object-oriented design supports modular development, reusability, and scalability of the system.

**4. Composition (Has-A Relationship):**

* **Definition**: Certain classes contain objects of other classes to represent complex structures and relationships.
* **Usage**: The Appointment class includes references to both Doctor and Patient classes, representing a real-world interaction.
* **Importance**: It models real-world relationships and allows for better organization and reuse of code.

**5. Association:**

* **Definition**: Logical connections are established between different classes to reflect real-world associations.
* **Usage**: A Doctor is associated with multiple Appointments, and a Patient is linked to their MedicalHistory.
* **Importance**: Association helps represent relationships among entities in a realistic and meaningful way.

**6. Constructors:**

* **Definition**: Constructors are used to create and initialize objects with required values at the time of instantiation.
* **Usage**: Each class includes constructors to set essential attributes like id, name, and specialization.
* **Importance**: They ensure proper object initialization and allow control over object creation.

**7. Exception Handling:**

* **Definition**: Java handles unexpected conditions using exception mechanisms to prevent program crashes and ensure smooth execution.
* **Usage**: Errors during appointment booking, data entry, or file/database access are handled using try-catch blocks.
* **Importance**: It improves system reliability and user experience by managing runtime errors gracefully

**8. Java Swing and AWT:**

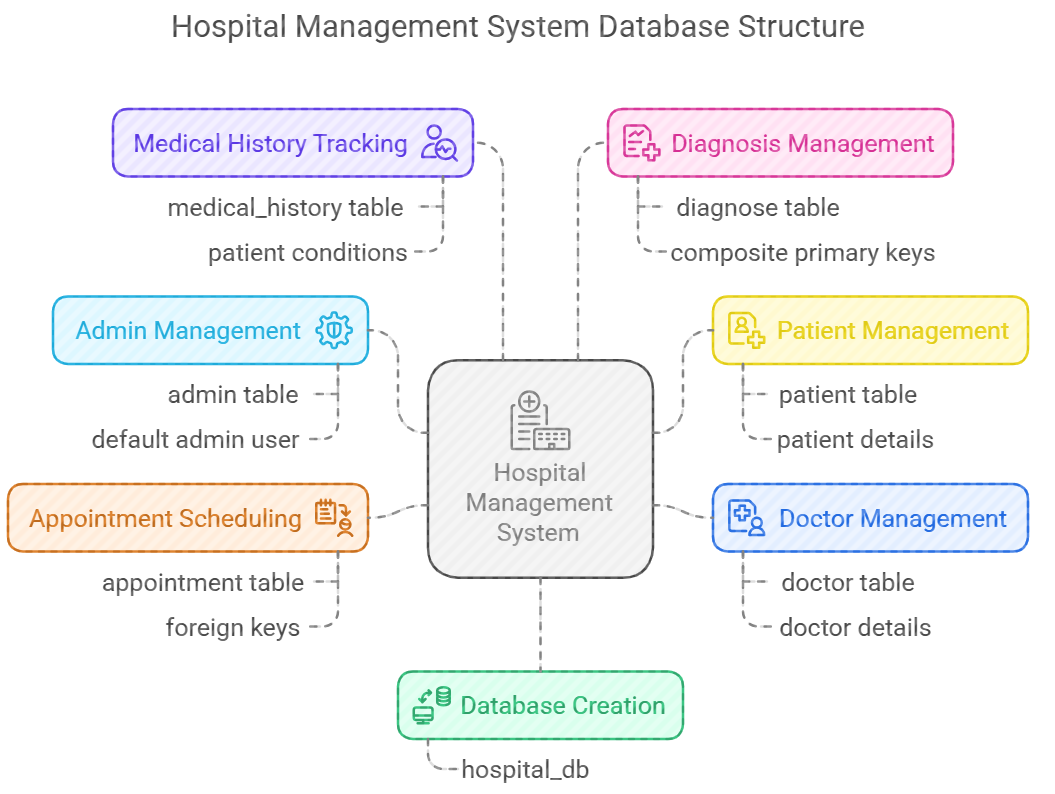
* The project employs Java Swing and Abstract Window Toolkit (AWT) for designing the graphical user interface (GUI).
* Java Swing provides a robust set of components to create user-friendly and interactive interfaces, while AWT handles the layout and component management.
* The combination of these technologies enables the development of a responsive and intuitive interface for hospital staff to manage patient records, appointments, and administrative tasks efficiently.
* Swing components such as buttons, text fields, tables, and combo boxes are used to ensure seamless user interaction.

**10. JDBC Connectivity:**

* For backend connectivity, Java Database Connectivity (JDBC) is utilized to connect the application to a MySQL database.
* JDBC facilitates secure and efficient data operations, including insert, update, delete, and retrieve operations, using SQL queries.
* MySQL serves as the relational database management system (RDBMS), storing patient details, medical history, and staff information.
* The use of JDBC ensures that the system is scalable, reliable, and capable of handling large datasets with high performance.

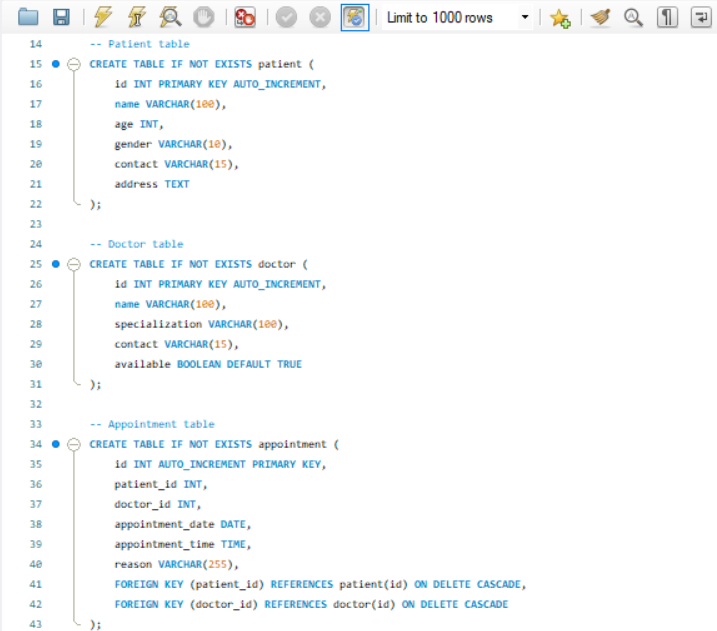
# Results

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



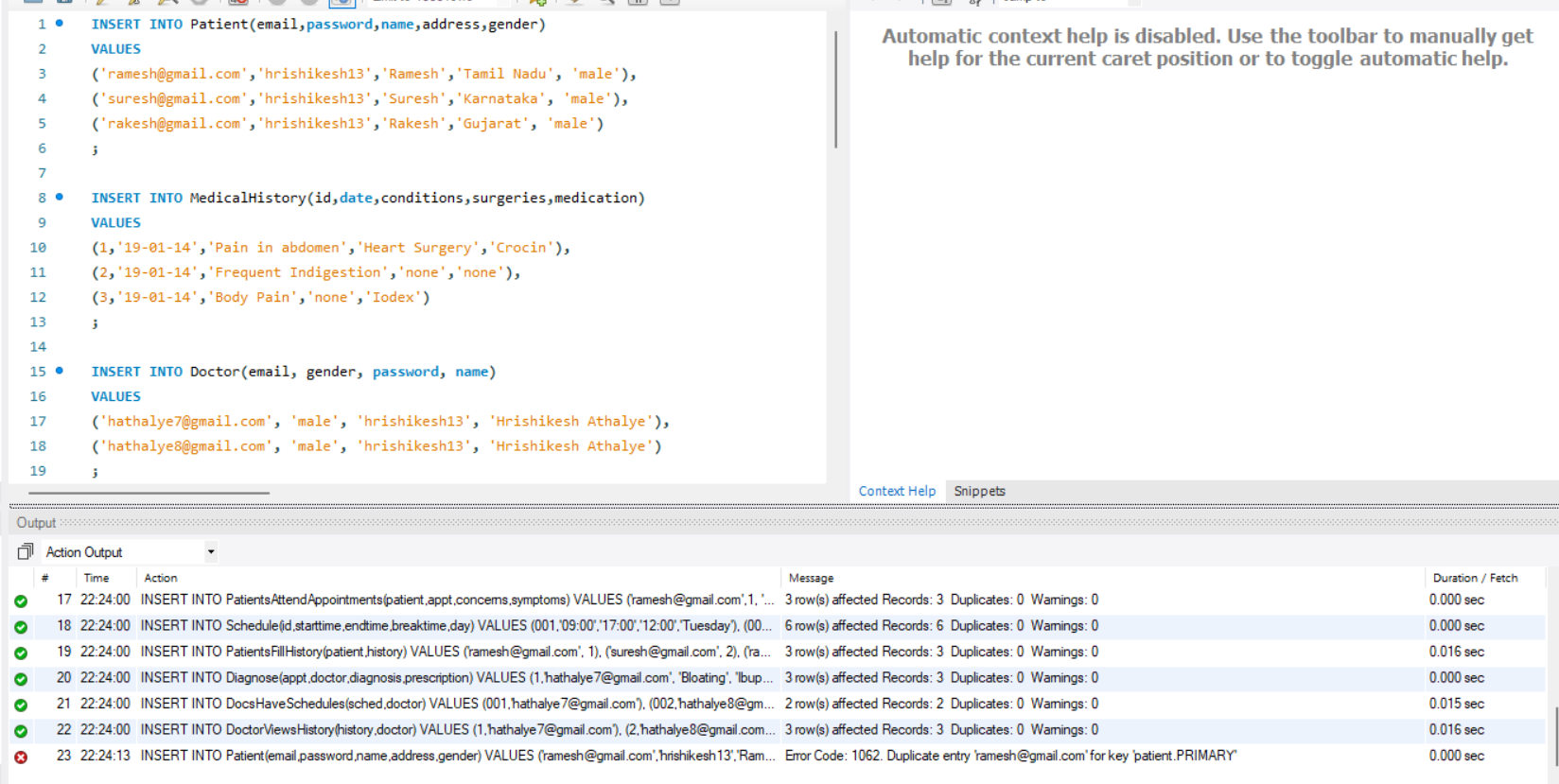
1. System Flowchart Diagram

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

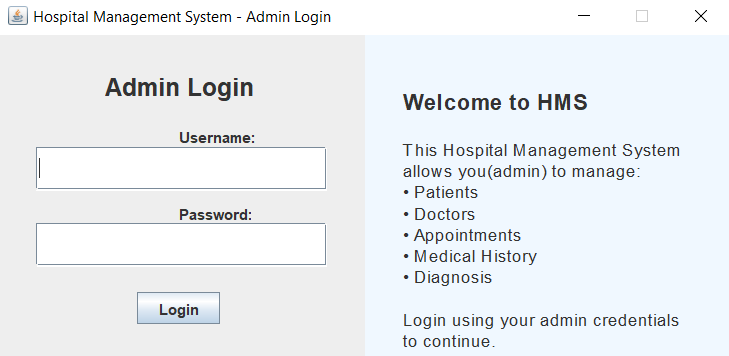
It displays the SQL schema creation for three relational tables: Diagnose, DocsHaveSchedules, and DoctorViewsHistory. Each table links the Doctor table using the email field as a foreign key and includes additional references to other tables such as Appointment, Schedule, and MedicalHistory. All foreign 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

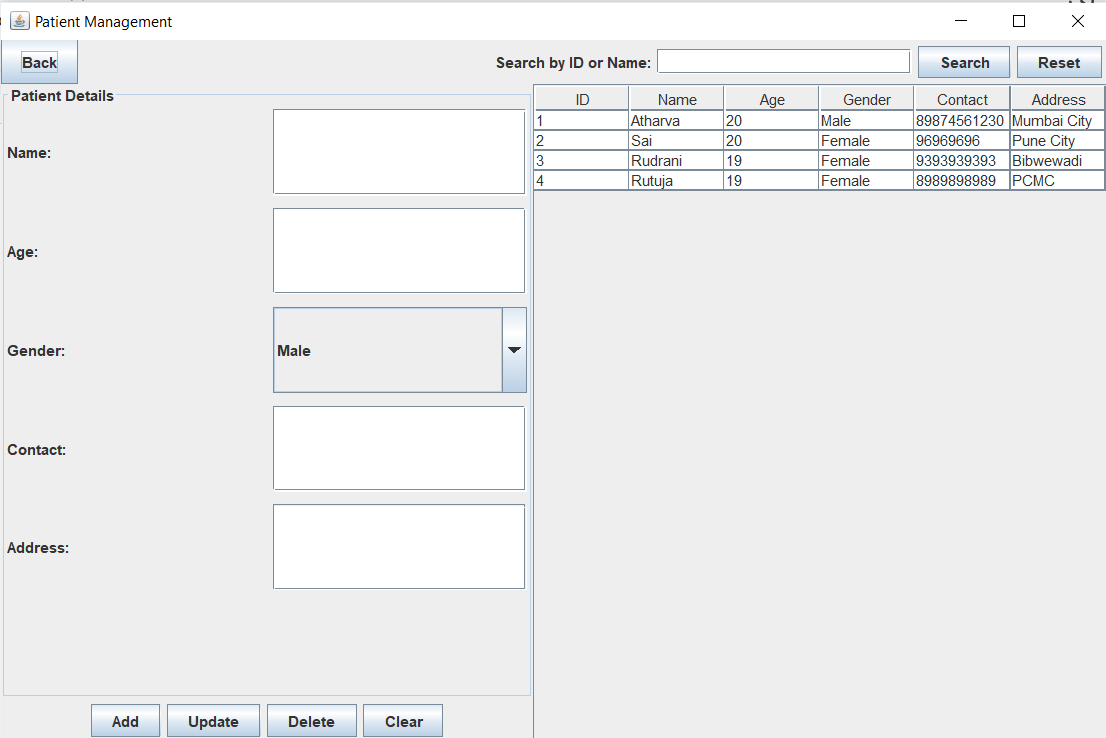
Fig. 8.(a-e) shows the results of the main 5 tables in GUI form. The Hospital Management System integrates comprehensive modules for managing Patients, Doctors, Appointments, Medical History, and Diagnosis, each featuring full CRUD (Create, Read, Update, Delete) capabilities along with advanced search 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.

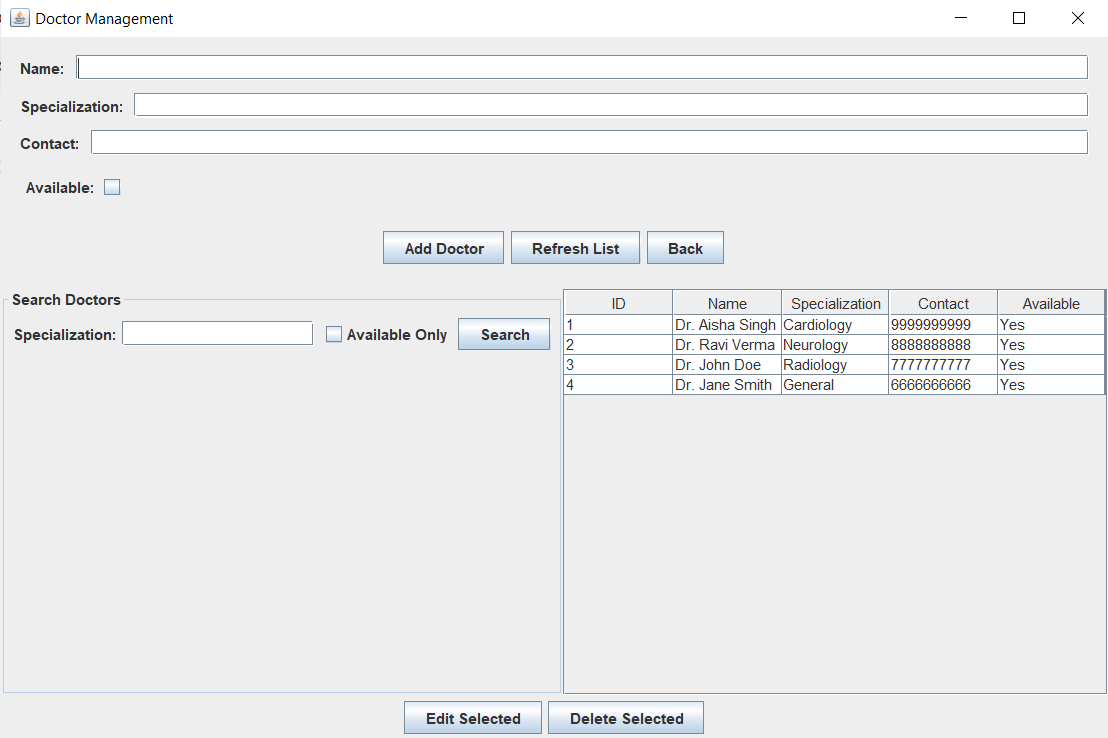
Furthermore, the modules are interconnected, ensuring seamless data flow—for instance, appointments link patients and doctors, while medical history ties closely with diagnosis records—facilitating comprehensive and coherent information management. The user interface is designed to be intuitive and responsive, 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.

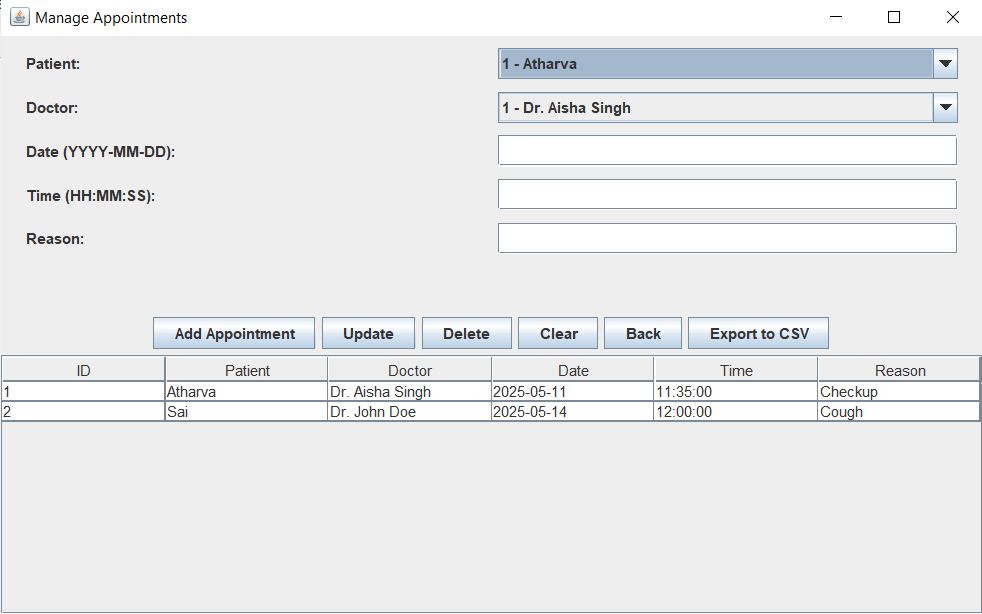
Overall, these modules form a robust foundation for managing critical healthcare information efficiently, supporting better decision-making and enhancing the quality of patient care.



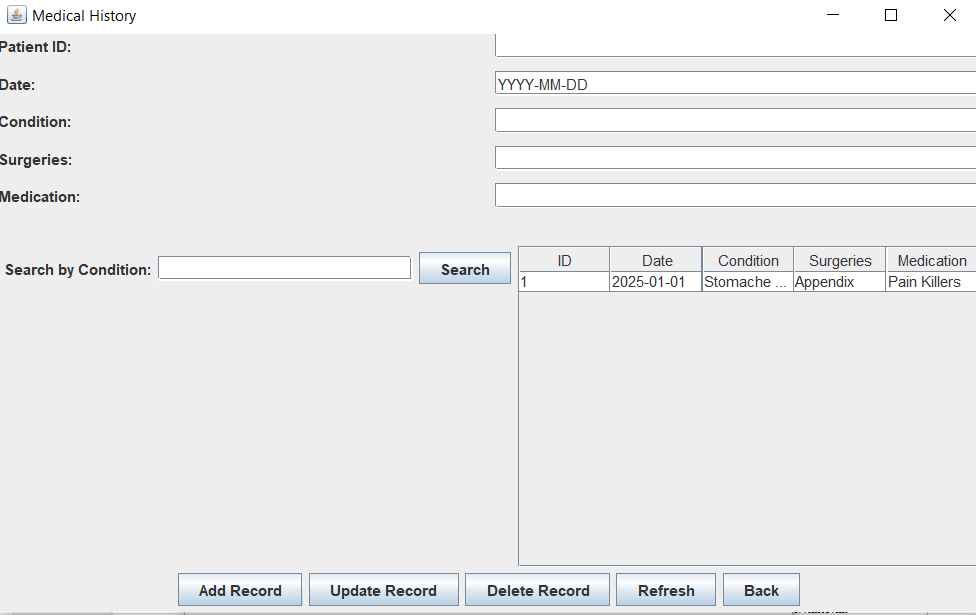
(a)



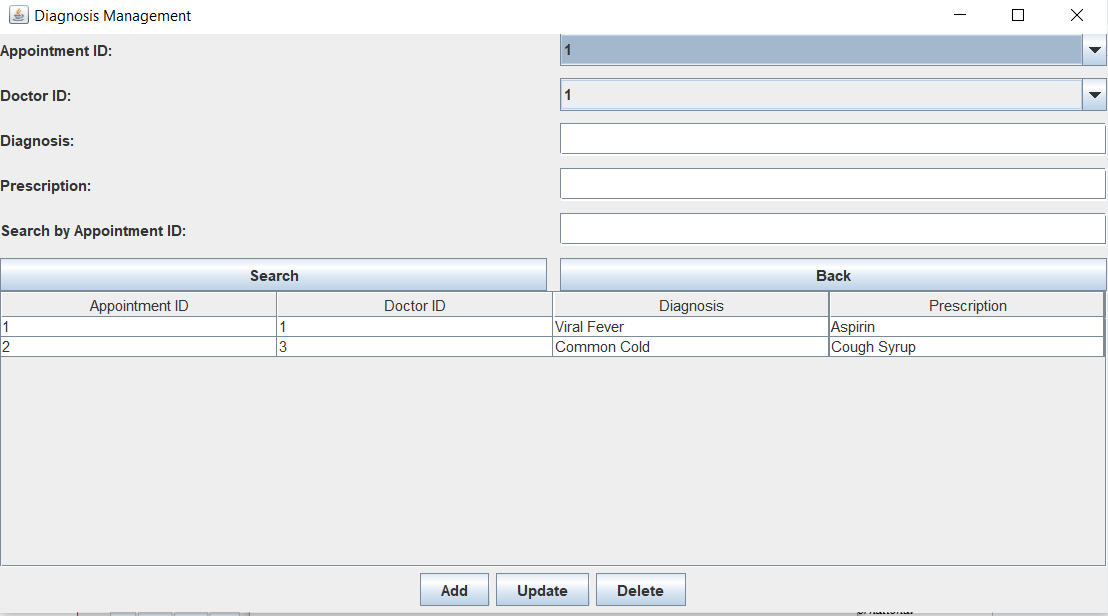
(b)



(c)



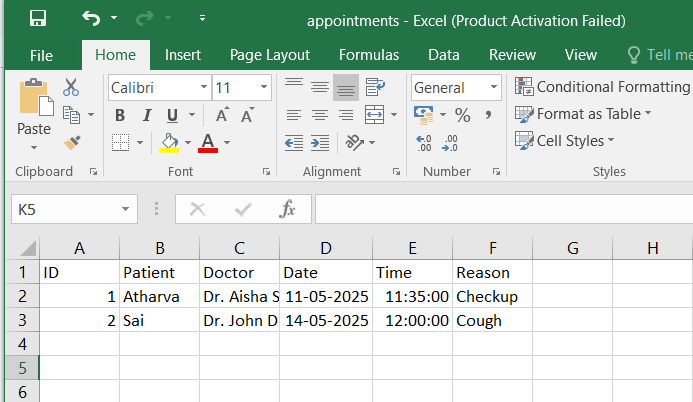
(d)



(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, designed to manage essential hospital operations such as appointment scheduling, patient record maintenance, and medical history tracking. The system incorporated dedicated interfaces for both patients and doctors, ensuring a secure, role-based user experience. The use of an Entity-Relationship (ER) model facilitated the clear definition of core entities and their interactions, contributing to an efficient and normalized database structure. Overall, the project successfully met its objectives by streamlining hospital workflows, minimizing manual errors, and improving communication between patients and healthcare professionals.

Despite its effectiveness in handling basic functionalities, the current version of the system is limited in terms of scalability and advanced capabilities. In the future, the system can be enhanced by integrating modules such as automated billing, insurance claim processing, laboratory report management, and pharmacy inventory. Incorporating AI-based diagnosis suggestions, chatbot-based appointment scheduling, and patient feedback mechanisms can further enrich the user experience. Additionally, migrating the application to a full-stack web architecture using frameworks like Django or Spring Boot, along with deploying the database on cloud platforms, can significantly improve system performance, security, and accessibility. Developing mobile applications for Android and iOS would also increase the system’s versatility, enabling patients and healthcare providers to interact with the system seamlessly from remote locations. With these enhancements, the Hospital Management System has the potential to evolve into a comprehensive and intelligent solution suitable for multi-hospital environments.

##### References

1. S. Khan et al., "Development of a Hospital Management System using Java and MySQL," International Journal of Computer Applications, vol. 179, no. 8, pp. 5-10, 2019.
2. N. Patel and S. Desai, "Modular Hospital Management System for Healthcare Institutions," Journal of Software Engineering, vol. 15, no. 3, pp. 21-28, 2020.
3. A. Singh et al., "Java-Based Hospital Management System: A Real-Time Data Management Approach," International Journal of Medical Informatics, vol. 144, pp. 50-58, 2021.
4. 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.
5. 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.
6. Raj, R. M., and S. P. Shukla. "Design of a Hospital Management System." *Journal of Computer Science and Engineering* 4.6 (2012): 88-92.
7. Kumar, Suraj, and Raghavendra Tiwari. "Database Management Systems for Healthcare Applications: A Survey." *International Journal of Computer Applications* 59.13 (2012): 23-27.
8. Jha, Sushant, et al. "Implementation of Hospital Information System Using DBMS." *International Journal of Computer Science and Mobile Computing* 3.3 (2014): 23-28.
9. Singh, Bhupinder, and Ranjit Kaur. "Development of an Effective Hospital Management System." *International Journal of Engineering Research and Applications* 4.3 (2014): 1-6.
10. 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.
11. 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.
12. 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.
13. Gupta, Vishal, and Sumit Bhargava. "Hospital Management System with SQL Database Integration." *International Journal of Engineering and Technology* 3.4 (2015): 56-60.
14. Madhusree, S., and S. A. Nair. "Hospital Management System: Concept and Design." *International Journal of Computer Science and Engineering* 3.1 (2011): 70-73.
15. 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.
16. Kaur, Navdeep, and Harpreet Kaur. "Analysis of Hospital Management System Using DBMS." *International Journal of Engineering and Technology* 5.7 (2014): 92-97.
17. Vijayakumar, M., and Preetha R. "An Overview of Hospital Management Systems and Technologies." *Journal of Computer Science* 5.2 (2013): 47-53.
18. 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