Hospital Management System

Abstract

This project involves the development of a Hospital Management System (HMS) using Java, aimed at streamlining the operations within a hospital. The HMS provides an integrated platform for managing patient information, doctor schedules, medical records, billing, and inventory of medical supplies. This system addresses the need for efficiency, accuracy, and quick access to patient data, ultimately improving patient care and hospital management.

Key technologies used in this project

 **JDBC (Java Database Connectivity)**:

* java.sql.Connection, java.sql.DriverManager, java.sql.PreparedStatement, java.sql.ResultSet, java.sql.Statement: These classes are fundamental for establishing database connections, executing SQL queries, and processing query results. JDBC allows Java applications to interact with relational databases such as SQLite (hospital\_management.db in this case).

 **SQLite**:

* SQLite is a lightweight, embedded relational database management system. It's used here to store and manage data related to patients, doctors, and appointments. SQLite databases are file-based and do not require a separate server process.

 **SQL (Structured Query Language)**:

* SQL statements (CREATE TABLE, INSERT INTO, SELECT \* FROM) are used to define database schema (CREATE TABLE statements), insert data (INSERT INTO statements), and query data (SELECT \* FROM statements).

 **Exception Handling**:

* try-catch blocks are used to handle exceptions (java.lang.Exception) that may occur during database operations (SQLException), ensuring proper error handling and graceful recovery.

 **Basic Java Syntax**:

* Standard Java language features such as methods (createTables, addPatient, addDoctor, addAppointment, displayPatients, displayDoctors, displayAppointments), variables, loops (while loop for iterating over ResultSet), and conditional statements (if statement`) are used throughout the code.

 **Data Modeling**:

* Tables (patients, doctors, appointments) are defined with appropriate fields (id, name, age, gender, contact, specialty, date, time) to model patient, doctor, and appointment data. Relationships between tables (FOREIGN KEY constraints) ensure data integrity.

 **Structured Programming**:

* The code is organized into methods for better modularity, readability, and reusability. Each method (createTables, addPatient, addDoctor, addAppointment, displayPatients, displayDoctors, displayAppointments) encapsulates a specific functionality related to database operations.

 **Console Output**:

* System.out.println() statements are used to display retrieved data (patients, doctors, appointments) on the console for demonstration purposes, providing feedback to the user.

Source code

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.Statement;

public class HospitalManagementSystem {

private static final String DB\_URL = "jdbc:sqlite:hospital\_management.db";

public static void main(String[] args) {

try (Connection conn = DriverManager.getConnection(DB\_URL)) {

if (conn != null) {

createTables(conn);

// Sample data

addPatient(conn, "John Doe", 30, "Male", "1234567890");

addDoctor(conn, "Dr. Smith", "Cardiology", "0987654321");

addAppointment(conn, 1, 1, "2024-06-24", "10:00 AM");

// Display all records

displayPatients(conn);

displayDoctors(conn);

displayAppointments(conn);

}

} catch (Exception e) {

e.printStackTrace();

}

}

private static void createTables(Connection conn) throws Exception {

String createPatientsTable = "CREATE TABLE IF NOT EXISTS patients ("

+ "id INTEGER PRIMARY KEY AUTOINCREMENT,"

+ "name TEXT NOT NULL,"

+ "age INTEGER,"

+ "gender TEXT,"

+ "contact TEXT"

+ ")";

String createDoctorsTable = "CREATE TABLE IF NOT EXISTS doctors ("

+ "id INTEGER PRIMARY KEY AUTOINCREMENT,"

+ "name TEXT NOT NULL,"

+ "specialty TEXT,"

+ "contact TEXT"

+ ")";

String createAppointmentsTable = "CREATE TABLE IF NOT EXISTS appointments ("

+ "id INTEGER PRIMARY KEY AUTOINCREMENT,"

+ "patient\_id INTEGER,"

+ "doctor\_id INTEGER,"

+ "date TEXT,"

+ "time TEXT,"

+ "FOREIGN KEY(patient\_id) REFERENCES patients(id),"

+ "FOREIGN KEY(doctor\_id) REFERENCES doctors(id)"

+ ")";

try (Statement stmt = conn.createStatement()) {

stmt.execute(createPatientsTable);

stmt.execute(createDoctorsTable);

stmt.execute(createAppointmentsTable);

}

}

private static void addPatient(Connection conn, String name, int age, String gender, String contact) throws Exception {

String sql = "INSERT INTO patients (name, age, gender, contact) VALUES (?, ?, ?, ?)";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, name);

pstmt.setInt(2, age);

pstmt.setString(3, gender);

pstmt.setString(4, contact);

pstmt.executeUpdate();

}

}

private static void addDoctor(Connection conn, String name, String specialty, String contact) throws Exception {

String sql = "INSERT INTO doctors (name, specialty, contact) VALUES (?, ?, ?)";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, name);

pstmt.setString(2, specialty);

pstmt.setString(3, contact);

pstmt.executeUpdate();

}

}

private static void addAppointment(Connection conn, int patientId, int doctorId, String date, String time) throws Exception {

String sql = "INSERT INTO appointments (patient\_id, doctor\_id, date, time) VALUES (?, ?, ?, ?)";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setInt(1, patientId);

pstmt.setInt(2, doctorId);

pstmt.setString(3, date);

pstmt.setString(4, time);

pstmt.executeUpdate();

}

}

private static void displayPatients(Connection conn) throws Exception {

String sql = "SELECT \* FROM patients";

try (Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

System.out.println("Patient ID: " + rs.getInt("id"));

System.out.println("Name: " + rs.getString("name"));

System.out.println("Age: " + rs.getInt("age"));

System.out.println("Gender: " + rs.getString("gender"));

System.out.println("Contact: " + rs.getString("contact"));

System.out.println();

}

}

}

private static void displayDoctors(Connection conn) throws Exception {

String sql = "SELECT \* FROM doctors";

try (Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

System.out.println("Doctor ID: " + rs.getInt("id"));

System.out.println("Name: " + rs.getString("name"));

System.out.println("Specialty: " + rs.getString("specialty"));

System.out.println("Contact: " + rs.getString("contact"));

System.out.println();

}

}

}

private static void displayAppointments(Connection conn) throws Exception {

String sql = "SELECT \* FROM appointments";

try (Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

System.out.println("Appointment ID: " + rs.getInt("id"));

System.out.println("Patient ID: " + rs.getInt("patient\_id"));

System.out.println("Doctor ID: " + rs.getInt("doctor\_id"));

System.out.println("Date: " + rs.getString("date"));

System.out.println("Time: " + rs.getString("time"));

System.out.println();

}

}

}

}

Output:

Patient ID: 1

Name: John Doe

Age: 30

Gender: Male

Contact: 1234567890

Doctor ID: 1

Name: Dr. Smith

Specialty: Cardiology

Contact: 0987654321

Appointment ID: 1

Patient ID: 1

Doctor ID: 1

Date: 2024-06-24

Time: 10:00 AM

Conclusion

Designing a hospital management system in Java involves creating a robust application that integrates various functionalities crucial for efficient hospital operations. The system typically includes modules for patient management, doctor scheduling, inventory management, billing, and administrative tasks.

Key considerations include scalability, security, and usability to ensure the system can handle a large volume of data securely while being user-friendly for both staff and patients. Using object-oriented principles and design patterns helps in structuring the codebase for maintainability and extensibility.

In conclusion, developing a hospital management system in Java requires careful planning, adherence to industry standards, and a focus on meeting the diverse needs of healthcare professionals and patients. It plays a crucial role in enhancing operational efficiency, improving patient care, and streamlining administrative tasks within a healthcare facility.

Top of Form

Bottom of Form