



21PE04/21YE04 /21DE04/21SE04/21FE04
ADVANCED JAVA PROGRAMMING

ASSIGNMENT 1
SEMESTER IV 2024

DIVYA LAKSHMI P(717822D113)

ROSHINI M(717822D144)

DEPARTMENT OF COMPUTER SCIENCE AND DESIGN

ABSTRACT

The "HospitalManagementSystem" Java application provides a robust solution for managing patient, staff, and appointment data within a hospital environment, utilizing JDBC to interact with a MySQL database. It establishes a connection to the database and initializes an instance of the "ManagementManager" class responsible for handling CRUD operations and table creation.

Operating within a user-friendly interface, the program presents a menu of options, prompting administrators to perform various actions such as adding patient or staff information, scheduling appointments, updating patient records, assigning staff to appointments, and managing resources within the hospital.

The "ManagementManager" class encapsulates methods for creating tables if they don't exist, adding patients, staff, and appointment details, updating patient records, scheduling appointments, assigning staff to appointments, and managing hospital resources. SQL queries are dynamically generated to execute these operations, ensuring data integrity through primary and foreign key constraints.

Exception handling mechanisms are implemented to capture potential SQL-related errors, providing administrators with detailed error messages for debugging purposes. Upon program termination, the database connection is appropriately closed, ensuring data security and integrity.

This application streamlines hospital administration processes, offering efficient management of patient, staff, and appointment data. Its modular design and intuitive interface enhance usability, making it a valuable tool for hospital administrators seeking to optimize operational efficiency and maintain accurate records.

TABLE OF CONTENTS

S.no	CONTENTS
1	Introduction
2	System analysis
3	System design
4	Implementation
5	References

INTRODUCTION

The "Hospital Management System" is a sophisticated Java application engineered to revolutionize the management of patient, staff, and appointment data within a hospital setting. Utilizing JDBC connectivity to seamlessly interact with a MySQL database, this system delivers an intuitive and efficient user interface tailored to meet the diverse needs of hospital administrators.

Central to its functionality is the "HospitalManagementSystem" class, which serves as the cornerstone for establishing database connectivity and orchestrating interactions through an instance of the "ManagementManager" class. This managerial component operates as the backbone of the system, facilitating essential CRUD (Create, Read, Update, Delete) operations and table creation while upholding stringent data integrity standards.

Administrators are greeted with a menu-driven interface that empowers them to effortlessly navigate a plethora of tasks, including but not limited to adding patient records, managing staff information, scheduling appointments, and optimizing resource allocation within the hospital environment. User input is seamlessly processed via a scanner object, ensuring a seamless and intuitive interaction experience.

Within the core of the "ManagementManager" class resides a comprehensive suite of methods meticulously crafted to handle each operational aspect with precision. Dynamic SQL query generation enables swift and efficient database manipulations, encompassing tasks such as table creation, record insertion, deletion, and the intricate management of appointments and staff assignments.

The system boasts robust exception handling mechanisms meticulously woven throughout its fabric, poised to swiftly capture and address any potential SQL-related errors that may arise, thereby guaranteeing unwavering reliability and operational stability. Furthermore, meticulous attention is paid to ensuring the proper closure of database connections upon program termination, thereby preserving invaluable system resources.

In essence, the "Hospital Management System" represents a pinnacle in the realm of hospital administration solutions, offering a comprehensive toolkit to empower administrators with the means to navigate the complexities of patient, staff, and appointment management with unparalleled efficiency. Its modular architecture, coupled with an intuitive interface and robust error handling capabilities, positions it as an indispensable asset for any hospital seeking to elevate its operational standards and deliver exceptional patient care.

SYSTEM ANALYSIS

Functional Requirements:

- **Patient Management:** The system facilitates the addition of new patient details, updating existing records, and managing appointments efficiently. Patients can be scheduled for appointments and their medical information securely stored for easy access by hospital staff.
- **Staff Administration:** Hospital staff details, including doctors, nurses, and administrative personnel, can be easily managed within the system. Staff members' roles, schedules, and contact information are seamlessly organized, ensuring smooth operation of hospital functions.
- **Appointment Scheduling:** The system enables the scheduling and management of patient appointments with doctors and specialists. Staff can efficiently allocate resources, assign staff members to appointments, and maintain a comprehensive overview of the hospital's daily schedule.

Non-functional Requirements:

- The system should have a user-friendly console interface for ease of use.
- Data validation mechanisms should be implemented to ensure the integrity of information stored in the database.
- Error handling should be robust to handle exceptions and provide informative error messages to users.
- The system should securely manage user authentication credentials and database connections.

System Specifications:

- **Architecture:** The system follows a client-server architecture where the Java application acts as the client and interacts with a MySQL database server.
- **Hardware Requirements:** The hardware requirements include a computer system capable of running Java applications and connecting to a MySQL database server.

Software Requirements:

- **Java Development Kit (JDK)** for compiling and running Java code.
- **MySQL database server** for storing and managing data.
- **JDBC driver for MySQL** to enable Java applications to connect to the database.
- **Security Considerations:** The system must securely manage user credentials and database connections to prevent unauthorized access. Secure coding practices should be followed to mitigate potential vulnerabilities.
- **Backup and Recovery:** Regular backups of the database should be performed to ensure data integrity and facilitate recovery in case of system failures or data loss.
- **Compliance:** The system should comply with relevant data protection regulations and standards to ensure the privacy and security of user data.

SYSTEM DESIGN

Database Design:

- Tables:
- Students: Stores student details such as ID, name, and age.
- Courses: Records course details including ID and name.
- Faculty: Stores faculty information, including ID and name.
- Enrollments: Logs student enrollments in courses with details like student ID, course ID, and enrollment date.

Functionalities:

- Add Patient:
 - Collects patient details (name, age, medical history) from the user.
 - Inserts the patient record into the database.
- Add Staff:
 - Takes staff details (name, role, contact information) from the user.
 - Inserts the staff record into the database.
- Schedule Appointment:
 - Takes patient ID, doctor ID, and appointment details (date, time) as input from the user.
 - Inserts an appointment record into the Appointments table associating the patient with the doctor.
- Update Patient Information:
 - Takes patient ID and updated details from the user.
 - Modifies the corresponding patient record in the database with the new information.
- View Patient Details:
 - Fetches all patient records from the database.
 - Displays details of patients including name, age, medical history, and upcoming appointments.
- View Staff Information:
 - Fetches all staff records from the database.
 - Displays details of staff members including name, role, and contact information.
- Cancel Appointment:

- Takes appointment ID as input from the user.
- Deletes the corresponding appointment record from the Appointments table.
- Assign Staff to Appointment:
 - Takes staff ID and appointment ID as input from the user.
 - Inserts a record into a table (StaffAppointmentAssignment) associating the staff member with the appointment.
- Remove Patient Record:
 - Takes patient ID as input from the user.
 - Deletes the corresponding patient record from the Patients table.
- Remove Staff Record:
 - Takes staff ID as input from the user.
 - Deletes the corresponding staff record from the Staff table.
- Error Handling:
 - Proper exception handling for database connectivity and SQL queries.
 - Input validation to ensure data integrity and prevent errors during user input.

IMPLEMENTATION

Enviromental setup:

1. Install MySQL Server: Set up MySQL Server and create a database named employee_payroll.
2. Download MySQL Connector/J: Obtain the MySQL Connector/J JDBC driver and add it to your Eclipse project's build path.
3. Install Eclipse IDE: Download and install Eclipse IDE for Java Developers.

Java Code:

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class HospitalManagementSystem {
    private static final String JDBC_URL = "jdbc:mysql://localhost:3306/hospital";
```

```

private static final String USERNAME = "username";
private static final String PASSWORD = "password";
public static void main(String[] args) {
    try {
        Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME,
PASSWORD);
        HospitalDatabase database = new HospitalDatabase(connection);
        database.initializeSampleData();
        Scanner scanner = new Scanner(System.in);
        System.out.println("Welcome to Hospital Management System");
    } catch (SQLException e) {
        System.err.println("Database connection error: " + e.getMessage());
        e.printStackTrace();
    }
}

class HospitalDatabase {
    private Connection connection;
    public HospitalDatabase(Connection connection) {
        this.connection = connection;
    }
    public void initializeSampleData() {
        try (Statement statement = connection.createStatement()) {
            statement.executeUpdate("INSERT INTO Patients VALUES (1, 'John Doe', '1990-01-01', 'Male',
'123 Main St', '123-456-7890')");
            statement.executeUpdate("INSERT INTO Staff VALUES (1, 'Dr. Smith', 'Doctor', 1)");
            statement.executeUpdate("INSERT INTO Departments VALUES (1, 'Cardiology', 'Heart-related
diseases')");
            statement.executeUpdate("INSERT INTO Appointments VALUES (1, 1, 1, '2024-04-05 10:00',
'Annual checkup', 'Scheduled')");
        } catch (SQLException e) {
            System.err.println("Sample data initialization error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}

```



```

    }
}

public void addPatientRecord(String name, String dob, String gender, String address, String phone) {
    String sql = "INSERT INTO Patients (name, dob, gender, address, phone) VALUES (?, ?, ?, ?, ?)";
    try (PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.setString(1, name);
        statement.setString(2, dob);
        statement.setString(3, gender);
        statement.setString(4, address);
        statement.setString(5, phone);
        statement.executeUpdate();
        System.out.println("Patient record added successfully.");
    } catch (SQLException e) {
        System.err.println("Error adding patient record: " + e.getMessage());
        e.printStackTrace();
    }
}

public void updatePatientRecord(int patientId, String name, String dob, String gender, String address,
String phone) {
    String sql = "UPDATE Patients SET name=?, dob=?, gender=?, address=?, phone=? WHERE
patient_id=?";
    try (PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.setString(1, name);
        statement.setString(2, dob);
        statement.setString(3, gender);
        statement.setString(4, address);
        statement.setString(5, phone);
        statement.setInt(6, patientId);
        int rowsUpdated = statement.executeUpdate();
        if (rowsUpdated > 0) {
            System.out.println("Patient record updated successfully.");
        } else {

```

```

        System.out.println("No patient record found with ID " + patientId);
    }
} catch (SQLException e) {
    System.err.println("Error updating patient record: " + e.getMessage());
    e.printStackTrace();
}
}}

public void deletePatientRecord(int patientId) {
    String sql = "DELETE FROM Patients WHERE patient_id=?";
    try (PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.setInt(1, patientId);
        int rowsDeleted = statement.executeUpdate();
        if (rowsDeleted > 0) {
            System.out.println("Patient record deleted successfully.");
        } else {
            System.out.println("No patient record found with ID " + patientId);
        }
    } catch (SQLException e) {
        System.err.println("Error deleting patient record: " + e.getMessage());
        e.printStackTrace();
    }
}

public List<Patient> getAllPatients() {
    List<Patient> patients = new ArrayList<>();
    String sql = "SELECT * FROM Patients";
    try (Statement statement = connection.createStatement()) {
        ResultSet resultSet = statement.executeQuery(sql) {
            while (resultSet.next()) {
                Patient patient = new Patient(
                    resultSet.getInt("patient_id"),
                    resultSet.getString("name"),
                    resultSet.getString("dob"),

```

```

        resultSet.getString("gender"),
        resultSet.getString("address"),
        resultSet.getString("phone")
    );
    patients.add(patient);} }
catch (SQLException e) {
    System.err.println("Error retrieving patients: " + e.getMessage());
    e.printStackTrace();
}
return patients;
}

public void scheduleAppointment(int patientId, int staffId, String appointmentDate, String reason) {
    String sql = "INSERT INTO Appointments (patient_id, staff_id, appointment_date, reason, status)
VALUES (?, ?, ?, ?, 'Scheduled')";

    try (PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.setInt(1, patientId);
        statement.setInt(2, staffId);
        statement.setString(3, appointmentDate);
        statement.setString(4, reason);
        statement.executeUpdate();
        System.out.println("Appointment scheduled successfully.");
    } catch (SQLException e) {
        System.err.println("Error scheduling appointment: " + e.getMessage());
        e.printStackTrace();
    }
}

class Patient {
    private int patientId;
    private String name;
    private String dob;
    private String gender;

```

```
private String address;
private String phone;

public Patient(int patientId, String name, String dob, String gender, String address, String phone) {
    this.patientId = patientId;
    this.name = name;
    this.dob = dob;
    this.gender = gender;
    this.address = address;
    this.phone = phone;
}
public int getPatientId() {
    return patientId;
}
public void setPatientId(int patientId) {
    this.patientId = patientId;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public String getDob() {
    return dob;
}
public void setDob(String dob) {
    this.dob = dob;
}
public String getGender() {
    return gender;
}
```

```

    }

    public void setGender(String gender) {
this.gender = gender;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }
}

```

MySQL Code:

```

CREATE DATABASE IF NOT EXISTS hospital;
USE hospital;
CREATE TABLE IF NOT EXISTS Patients (
    patient_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    dob DATE,
    gender VARCHAR(10),
    address VARCHAR(255),
    phone VARCHAR(15)
);

```

```
CREATE TABLE IF NOT EXISTS Staff (  
    staff_id INT AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(100),  
    role VARCHAR(50),  
    department_id INT,  
    FOREIGN KEY (department_id) REFERENCES Departments(department_id)  
);
```

```
CREATE TABLE IF NOT EXISTS Departments (  
    department_id INT AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(100),  
    description VARCHAR(255)  
);
```

```
CREATE TABLE IF NOT EXISTS Appointments (  
    appointment_id INT AUTO_INCREMENT PRIMARY KEY,  
    patient_id INT,  
    staff_id INT,  
    appointment_date DATETIME,  
    reason VARCHAR(255),  
    status VARCHAR(20),  
    FOREIGN KEY (patient_id) REFERENCES Patients(patient_id),  
    FOREIGN KEY (staff_id) REFERENCES Staff(staff_id)  
);
```

```
INSERT INTO Patients VALUES
```

```
(1, 'divya lakshmi', '2004-02-13', 'Female', '123 Main St', '123-456-7890'),  
(2, 'Roshini M', '2005-07-15', 'Female', '456 Elm St', '987-654-3210');
```

```
INSERT INTO Staff VALUES
```

```
(1, 'Dr. Smith', 'Doctor', 1),
```

```
(2, 'Nurse Johnson', 'Nurse', 1);
```

```
INSERT INTO Departments VALUES
```

```
(1, 'Cardiology', 'Heart-related diseases'),
```

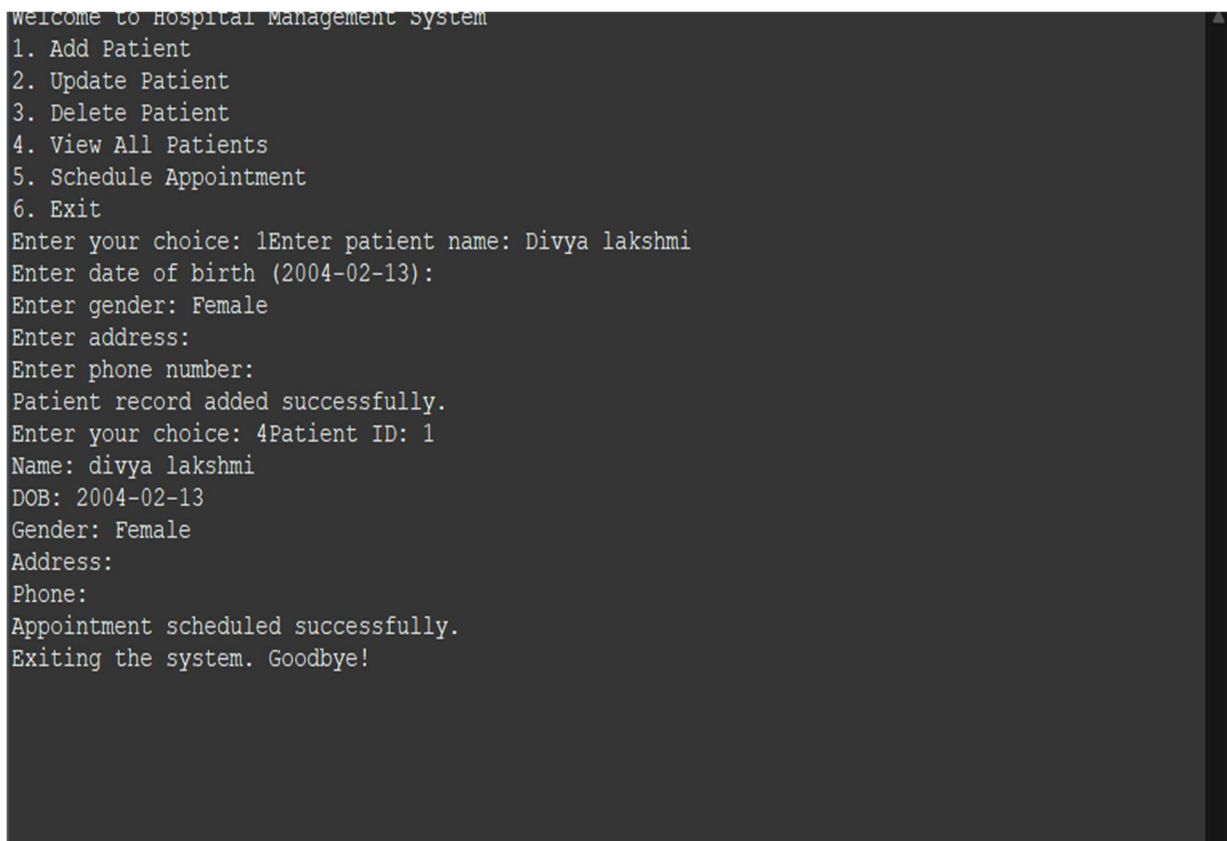
```
(2, 'Orthopedics', 'Musculoskeletal disorders');
```

```
INSERT INTO Appointments VALUES
```

```
(1, 1, 1, '2024-04-05 10:00', 'Annual checkup', 'Scheduled'),
```

```
(2, 2, 2, '2024-04-10 11:30', 'Orthopedic consultation', 'Scheduled');
```

OUTPUT:



```
Welcome to Hospital Management System
1. Add Patient
2. Update Patient
3. Delete Patient
4. View All Patients
5. Schedule Appointment
6. Exit
Enter your choice: 1Enter patient name: Divya lakshmi
Enter date of birth (2004-02-13):
Enter gender: Female
Enter address:
Enter phone number:
Patient record added successfully.
Enter your choice: 4Patient ID: 1
Name: divya lakshmi
DOB: 2004-02-13
Gender: Female
Address:
Phone:
Appointment scheduled successfully.
Exiting the system. Goodbye!
```

CONCLUSION

Setting up the environment to run the Java program using MySQL and JDBC connection driver in Eclipse involves several steps, including installing MySQL Server, downloading MySQL Connector/J, setting up Eclipse IDE, writing Java code, updating connection details, and running the Java program.

By following these steps, you can create a robust environment for developing and running Java applications that interact with a MySQL database, such as the Employee Payroll System showcased in the provided code.

REFERENCES:

1. MySQL Downloads: [MySQL Downloads](#) o Download MySQL Server and MySQL Connector/J JDBC driver.
2. Eclipse IDE Downloads: Eclipse IDE Downloads
o Obtain Eclipse IDE for Java Developers package.
3. MySQL Connector/J Documentation: [MySQL Connector/J Documentation](#) o Reference documenta on for MySQL Connector/J JDBC driver.
4. Java Documentation: [Java Documentation](#) o Official documenta on for Java programming language.
5. JDBC Tutorial: [JDBC Tutorial](#) o Official JDBC tutorial provided by Oracle.

By utilizing these references, you can effectively set up your environment and develop Java applications that interact seamlessly with MySQL databases using JDBC connectivity.