Rajalakshmi Engineering College

Name: Neha R N
Email: 240701354@rajalakshmi.edu.in
Roll no: 240701354
Phone: 9080137196
Branch: REC
Department: CSE - Section 7
Batch: 2028
Degree: B.E - CSE

Scan to verify results

2024_28_III_OOPS Using Java Lab

REC_2028_OOPS using Java_Week 11

Attempt : 1
Total Mark : 20
Marks Obtained : 10

Section 1 : Project

1. Problem Statement

Create a JDBC-based Hospital Management System that handles runtime input to manage patient records. The system should allow users to:

Add a new patient (patient ID, name, age, status).

Update a patient's status.

View a specific patient's record by patient ID.

Display all patient records in the database.

Exit the application.

The system should connect to a MySQL database using the following default credentials:

DB URL: jdbc:mysql://localhost/ri_db

USER: test

PWD: test123

The patients table has already been created with the following structure:

Table Name: patients

Input Format

The first line of input consists of an integer choice, representing the operation to be performed:

(1 for Add Patient, 2 for Update Patient Status, 3 for View Patient Record, 4 for Display All Patients, 5 for Exit)

For choice 1 (Add Patient):

- The second line consists of an integer patient_id.
- The third line consists of a string name.
- The fourth line consists of an integer age.
- The fifth line consists of a string status.

For choice 2 (Update Patient Status):

- The second line consists of an integer patient_id.
- The third line consists of a string new_status.

For choice 3 (View Patient Record):

- The second line consists of an integer patient_id.

For choice 4 (Display All Patients):

- No additional inputs are required.

For choice 5 (Exit):

- No additional inputs are required.

Output Format

For choice 1 (Add Patient):

- Print "Patient added successfully" if the patient was added.
- Print "Failed to add patient." if the insertion failed.

For choice 2 (Update Patient Status):

- Print "Patient status updated successfully" if the update was successful.
- Print "Patient not found." if the specified patient ID does not exist.

For choice 3 (View Patient Record):

- Display the patient details in the format:
- ID: [patient_id] | Name: [name] | Age: [age] | Status: [status]
- Print "Patient not found." if the specified patient ID does not exist.

For choice 4 (Display All Patients):

- Display each patient on a new line in the format:
- ID | Name | Age | Status
- If no records are available, print nothing (or handle it with an appropriate message if desired).

For choice 5 (Exit):

- Print "Exiting Hospital Management System."

For invalid input:

- Print "Invalid choice. Please try again."

Sample Test Case

Input: 1
101
John Doe
45
Admitted
4

5
Output: Patient added successfully
ID | Name | Age | Status
101 | John Doe | 45 | Admitted
Exiting Hospital Management System.

Answer

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

class HospitalManagementSystem {
   public static void main(String[] args) {
      try (Connection conn = DriverManager.getConnection("jdbc:mysql://
localhost/ri_db", "test", "test123");
          Scanner scanner = new Scanner(System.in)) {

         boolean running = true;

         while (running) {

            int choice = scanner.nextInt();

            switch (choice) {
               case 1:
                  addPatient(conn, scanner);
                  break;
               case 2:
                  updatePatientStatus(conn, scanner);
                  break;
               case 3:
                  viewPatientRecord(conn, scanner);
                  break;
               case 4:
                  displayAllPatients(conn);
                  break;
               case 5:
                  System.out.println("Exiting Hospital Management System.");
                  running = false;
                  break;
               default:
                  System.out.println("Invalid choice. Please try again.");
            }
```

```
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

public static void addPatient(Connection conn, Scanner scanner) {
        int id = scanner.nextInt();
        String name = scanner.next();
        String fullName = name;
        String temp = scanner.next();
        while (!temp.matches("\\d+")) {
            fullName += " " + temp;
            temp = scanner.next();
        }
        int age = Integer.parseInt(temp);
        String status = scanner.next();

        String sql = "INSERT INTO patients (patient_id, name, age, status) VALUES
(?, ?, ?, ?)";

        try {
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setInt(1, id);
            ps.setString(2, fullName);
            ps.setInt(3, age);
            ps.setString(4, status);

            int res = ps.executeUpdate();
            if (res > 0) System.out.println("Patient added successfully");
            else System.out.println("Failed to add patient.");
        } catch (Exception e) {
            System.out.println("Failed to add patient.");
        }
    }

    public static void updatePatientStatus(Connection conn, Scanner scanner) {
        int id = scanner.nextInt();
        String status = scanner.next();

        String sql = "UPDATE patients SET status=? WHERE patient_id=?";

        try {
```

```
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, status);
            ps.setInt(2, id);

            int res = ps.executeUpdate();
            if (res > 0) System.out.println("Patient status updated successfully");
            else System.out.println("Patient not found.");
        } catch (Exception e) {
            System.out.println("Patient not found.");
        }
    }

    public static void viewPatientRecord(Connection conn, Scanner scanner) {
        int id = scanner.nextInt();
        String sql = "SELECT * FROM patients WHERE patient_id=" + id;

        try {
            Statement st = conn.createStatement();
            ResultSet rs = st.executeQuery(sql);

            if (rs.next()) {
                System.out.println("ID: " + rs.getInt("patient_id")
                        + " | Name: " + rs.getString("name")
                        + " | Age: " + rs.getInt("age")
                        + " | Status: " + rs.getString("status"));
            } else {
                System.out.println("Patient not found.");
            }
        } catch (Exception e) {
            System.out.println("Patient not found.");
        }
    }

    public static void displayAllPatients(Connection conn) {
        String sql = "SELECT * FROM patients";

        try {
            Statement st = conn.createStatement();
            ResultSet rs = st.executeQuery(sql);

            System.out.println("ID | Name | Age | Status");
```

```
            while (rs.next()) {
                System.out.println(
                        rs.getInt("patient_id") + " | " +
                        rs.getString("name") + " | " +
                        rs.getInt("age") + " | " +
                        rs.getString("status")
                );
            }
        } catch (Exception e) {

        }
    }
}
```

Status : Correct *Marks : 10/10*

2. Problem Statement

In ABC Corporation, employee records are stored in a database.

To efficiently manage employee details using Java and JDBC, you are tasked with building an Employee Management System that supports the following functionalities:

Adding a new employee

Updating an employee's salary

Viewing an employee's details

Displaying all employees

You are given two files:

File 1: Employee.java (POJO Class)

This class represents the Employee entity.

An Employee contains the following details:

Field Description

employeeId Unique Employee ID (Integer)

name Employee Name (String)

department Employee Department (String)

salary Employee Salary (Double)

Students must write code in the marked area:

```
class Employee {
    private int employeeId;
    private String name;
    private String department;
    private double salary;

    public Employee() {}

    public Employee(int employeeId, String name, String department, double
salary) {
        // write your code here
    }

    // Include getters and setters
}
```

Expected in this part:

Assign parameter values to instance variables inside the constructor.

Add getters and setters for all attributes.

File 2: EmployeeDAO.java (Data Access Layer)

This class handles all database operations using JDBC.

Students must complete the missing JDBC logic in the following methods:

```
class EmployeeDAO {

    public void addEmployee(Connection conn, Employee employee) throws SQLException {
        // write your code here
    }

    public void updateSalary(Connection conn, int employeeId, double newSalary) throws SQLException {
        // write your code here
    }

    public void deleteEmployee(Connection conn, int employeeId) throws SQLException {
        // write your code here
    }

    public Employee viewEmployeeRecord(Connection conn, int employeeId) throws SQLException {
        // write your code here
    }

    public List<Employee> displayAllEmployees(Connection conn) throws SQLException {
        // write your code here
    }

    private Employee mapToEmployee(ResultSet rs) throws SQLException {
        return new Employee(
            // write your code here
        );
    }
```

}

Expected in this part:

Write SQL queries for INSERT, UPDATE, DELETE, SELECT.

Execute queries using PreparedStatement or Statement.

Map ResultSet rows to Employee objects using mapToEmployee().

Return a List<Employee> where required.

The system should connect to a MySQL database using the following default credentials:

DB URL: jdbc:mysql://localhost/ri_dbUsername: testPassword: test123

The employees table has already been created with the following structure:

Input Format

The first line of input consists of an integer choice, representing the operation to be performed:

(1 for Add Employee, 2 for Update Salary, 3 for View Employee Record, 4 for Display All Employees, 5 for Exit)

For choice 1 (Add Employee):

1. The second line consists of an integer employee_id.
2. The third line consists of a string name.
3. The fourth line consists of a string department.
4. The fifth line consists of a double salary (must be at least 30000).

For choice 2 (Update Salary):

1. The second line consists of an integer employee_id.
2. The third line consists of a double new_salary (must be at least 30000).

For choice 3 (View Employee Record):

1. The second line consists of an integer employee_id.

For choice 4 (Display All Employees).

For choice 5 (Exit).

Output Format

For choice 1 (Add Employee),

1. Print "Employee added successfully" if the employee was added.

For choice 2 (Update Salary),

1. Print "Salary updated successfully" if the salary update was successful.
2. Print "Employee not found." if the specified employee ID does not exist.
3. Print "Salary must be at least 30000." if the provided salary is below the minimum.

For choice 3 (View Employee Record),

1. Display the employee details in the format:
2. ID: [employee_id] | Name: [name] | Department: [department] | Salary: [salary]
3. Print "Employee not found." if the specified employee ID does not exist.

For choice 4 (Display All Employees),

1. Display each employee on a new line in the format:
2. ID | Name | Department | Salary

For choice 5 (Exit),

1. Print "Exiting Employee Management System."

For invalid input:

1. Print "Invalid choice. Please try again."

Sample Test Case

Input: 1
101
Alice Johnson
Engineering
31000.75
4
6
5

Output: Employee added successfully
ID | Name | Department | Salary
101 | Alice Johnson | Engineering | 31000.75
Invalid choice. Please try again.
Exiting Employee Management System.

Answer

-

Status : Skipped ***Marks : 0/10***