Rajalakshmi Engineering College

Name: Rupashri S K
Email: 240701445@rajalakshmi.edu.in
Roll no: 240701445
Phone: 6382173059
Branch: REC
Department: CSE - Section 10
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 : 2
Total Mark : 20
Marks Obtained : 20

Section 1 : Project

1. 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

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

class Employee{


    private int employeeId;
```

```
    private String name;
    private String department;
    private double salary;

    // Constructor
    public Employee(int employeeId, String name, String department, double
salary)

{


        this.employeeId = employeeId;
        this.name = name;
        this.department = department;
        this.salary = salary;


}

    // Getters and Setters
    public int getEmployeeId()

{

 return employeeId;

}
    public void setEmployeeId(int employeeId)

{

 this.employeeId = employeeId;

}

    public String getName()

{

 return name;

}
```

```
    public void setName(String name)
{
 this.name = name;
}
    public String getDepartment()
{
 return department;
}
    public void setDepartment(String department)
{
 this.department = department;
}
    public double getSalary()
{
 return salary;
}
    public void setSalary(double salary)
{
 this.salary = salary;
}
}
class EmployeeManagementSystem
```

```
{


    // Add Employee
    public static void addEmployee(Connection conn, Scanner scanner)

{


        int employeeId = scanner.nextInt();
        scanner.nextLine();  // Consume newline
        String name = scanner.nextLine();
        String department = scanner.nextLine();
        double salary = scanner.nextDouble();

        if (salary < 30000)

{


            System.out.println("Salary must be at least 30000.");
            return;


}

        // Create an Employee POJO object
        Employee employee = new Employee(employeeId, name, department,
salary);

        String insertQuery = "INSERT INTO employees (employee_id, name,
department, salary) VALUES (?, ?, ?, ?)";
        try (PreparedStatement stmt = conn.prepareStatement(insertQuery))

{


            stmt.setInt(1, employee.getEmployeeId());
            stmt.setString(2, employee.getName());
            stmt.setString(3, employee.getDepartment());
```

```
            stmt.setDouble(4, employee.getSalary());

            int rowsInserted = stmt.executeUpdate();
            System.out.println(rowsInserted > 0 ? "Employee added successfully" :
"Failed to add employee.");


} catch (SQLException e)

{


            System.out.println("Error adding employee: " + e.getMessage());

}


}

    // Update Salary
    public static void updateSalary(Connection conn, Scanner scanner)

{


        int employeeId = scanner.nextInt();
        double newSalary = scanner.nextDouble();

        if (newSalary < 30000)

{


            System.out.println("Salary must be at least 30000.");
            return;


}

        String updateQuery = "UPDATE employees SET salary = ? WHERE
employee_id = ?";
```

```
        try (PreparedStatement stmt = conn.prepareStatement(updateQuery))
{

            stmt.setDouble(1, newSalary);
            stmt.setInt(2, employeeId);

            int rowsUpdated = stmt.executeUpdate();
            System.out.println(rowsUpdated > 0 ? "Salary updated successfully" :
"Employee not found.");


} catch (SQLException e)
{

            System.out.println("Error updating salary: " + e.getMessage());


}


}

    // View Employee Record
    public static void viewEmployeeRecord(Connection conn, Scanner scanner)
{


        int employeeId = scanner.nextInt();
        String selectQuery = "SELECT * FROM employees WHERE employee_id = ?";

        try (PreparedStatement stmt = conn.prepareStatement(selectQuery))

{


            stmt.setInt(1, employeeId);
            ResultSet rs = stmt.executeQuery();
```

```
        if (rs.next())
{

            Employee employee = new Employee(
                rs.getInt("employee_id"),
                rs.getString("name"),
                rs.getString("department"),
                rs.getDouble("salary")
            );
            System.out.printf("ID: %d | Name: %s | Department: %s | Salary: %.2f%n",
                employee.getEmployeeId(),
                employee.getName(),
                employee.getDepartment(),
                employee.getSalary());


} else

{


            System.out.println("Employee not found.");


}

} catch (SQLException e)

{


        System.out.println("Error retrieving employee record: " + e.getMessage());


}

}
```

```
    // Display All Employees
    public static void displayAllEmployees(Connection conn)

{


        String displayQuery = "SELECT * FROM employees";

        try (Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(displayQuery))

{


            System.out.println("ID | Name | Department | Salary");
            while (rs.next())

{


                Employee employee = new Employee(
                        rs.getInt("employee_id"),
                        rs.getString("name"),
                        rs.getString("department"),
                        rs.getDouble("salary")
                );
                System.out.printf("%d | %s | %s | %.2f%n",
                        employee.getEmployeeId(),
                        employee.getName(),
                        employee.getDepartment(),
                        employee.getSalary());


}


} catch (SQLException e)

{
```

```
            System.out.println("Error displaying employees: " + e.getMessage());

        }


    }
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost/ri_db";
        String username = "test";
        String password = "test123";

        try (Connection conn = DriverManager.getConnection(url, username,
password);
             Scanner scanner = new Scanner(System.in)) {

            int choice;
            do {
                choice = scanner.nextInt();

                switch (choice) {
                    case 1 -> addEmployee(conn, scanner);
                    case 2 -> updateSalary(conn, scanner);
                    case 3 -> viewEmployeeRecord(conn, scanner);
                    case 4 -> displayAllEmployees(conn);
                    case 5 -> System.out.println("Exiting Employee Management
System.");
                    default -> System.out.println("Invalid choice. Please try again.");
                }

            } while (choice != 5);

        } catch (SQLException e) {
            System.out.println("Database Error: " + e.getMessage());
        }
    }
}
```

Status : Correct **Marks : 10/10**

2. Problem Statement

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

Add a new student (student ID, name, grade level, GPA).

Update a student's GPA, ensuring the GPA value is within the valid range (0.0 - 4.0).

View a specific student’s record by student ID.

Display all students 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 students table has already been created with the following structure:

Table Name: students

Input Format

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

(1 for Add Student, 2 for Update GPA, 3 for View Student Record, 4 for Display All Students, 5 for Exit)

For choice 1 (Add Student):

- The second line consists of an integer student_id.
- The third line consists of a string name.
- The fourth line consists of a string grade_level.

- The fifth line consists of a double gpa (must be between 0.0 and 4.0).

For choice 2 (Update GPA):

- The second line consists of an integer student_id.
- The third line consists of a double new_gpa (must be between 0.0 and 4.0).

For choice 3 (View Student Record):

- The second line consists of an integer student_id.

For choice 4 (Display All Students):

- No additional inputs are required.

For choice 5 (Exit):

- No additional inputs are required.

Output Format

The output displays:

For choice 1 (Add Student):

- Print "Student added successfully" if the student was added.
- Print "Failed to add student." if the insertion failed.

For choice 2 (Update GPA):

- Print "GPA updated successfully" if the GPA update was successful.
- Print "Student not found." if the specified student ID does not exist.
- Print "GPA must be between 0.0 and 4.0." if the provided GPA is out of the valid range.

For choice 3 (View Student Record):

- Display the student details in the format:
- ID: [student_id] | Name: [name] | Grade Level: [grade_level] | GPA: [gpa]
- Print "Student not found." if the specified student ID does not exist.

For choice 4 (Display All Students):

- Display each student on a new line in the format:
- ID | Name | Grade Level | GPA
- If there are no records, print nothing (or handle with an appropriate message if desired).

For choice 5 (Exit):

- Print "Exiting School Management System."

For invalid input:

- Print "Invalid choice. Please try again."

Sample Test Case

Input: 1
101
Alice Johnson
10
3.8
5
Output: Student added successfully
Exiting School Management System.

Answer

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

class SchoolManagementSystem {
    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:
                        addStudent(conn, scanner);
```

```
                break;
            case 2:
                updateGrades(conn, scanner);
                break;
            case 3:
                viewStudentRecord(conn, scanner);
                break;
            case 4:
                displayAllStudents(conn);
                break;
            case 5:
                System.out.println("Exiting School Management System.");
                running = false;
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
      }
    } catch (SQLException e) {
       e.printStackTrace();
    }
  }
public static void addStudent(Connection conn, Scanner scanner){



    int studentId = scanner.nextInt();
    scanner.nextLine(); // Consume newline

    String name = scanner.nextLine();

    String gradeLevel = scanner.nextLine();

    double gpa = scanner.nextDouble();

    String insertQuery = "INSERT INTO students (student_id, name, grade_level,
gpa) VALUES (?, ?, ?, ?)";
    try (PreparedStatement stmt = conn.prepareStatement(insertQuery))
{
```

```
            stmt.setInt(1, studentId);
            stmt.setString(2, name);
            stmt.setString(3, gradeLevel);
            stmt.setDouble(4, gpa);

            int rowsInserted = stmt.executeUpdate();
            System.out.println(rowsInserted > 0 ? "Student added successfully" :
"Failed to add student.");


} catch (SQLException e)

{

            System.out.println("Error adding student: " + e.getMessage());


}


}

    public static void updateGrades(Connection conn, Scanner scanner)

{


        int studentId = scanner.nextInt();

        double newGpa = scanner.nextDouble();

        // Validate GPA range
        if (newGpa < 0.0 || newGpa > 4.0)

{


            System.out.println("GPA must be between 0.0 and 4.0.");
            return;
```

```
}
        String updateQuery = "UPDATE students SET gpa = ? WHERE student_id = ?";
        try (PreparedStatement stmt = conn.prepareStatement(updateQuery))

{


            stmt.setDouble(1, newGpa);
            stmt.setInt(2, studentId);

            int rowsUpdated = stmt.executeUpdate();
            System.out.println(rowsUpdated > 0 ? "GPA updated successfully" :
"Student not found.");


} catch (SQLException e)

{


            System.out.println("Error updating GPA: " + e.getMessage());


}

}
    public static void viewStudentRecord(Connection conn, Scanner scanner)

{


        int studentId = scanner.nextInt();

        String selectQuery = "SELECT * FROM students WHERE student_id = ?";
        try (PreparedStatement stmt = conn.prepareStatement(selectQuery))
{
```

```
        stmt.setInt(1, studentId);

        ResultSet rs = stmt.executeQuery();
        if (rs.next())

{


            System.out.printf("ID: %d | Name: %s | Grade Level: %s | GPA: %.2f%n",
                rs.getInt("student_id"),
                rs.getString("name"),
                rs.getString("grade_level"),
                rs.getDouble("gpa"));


} else

{


            System.out.println("Student not found.");


}


} catch (SQLException e)

{


        System.out.println("Error retrieving student record: " + e.getMessage());


}


}

    public static void displayAllStudents(Connection conn)
```

```
{
        String displayQuery = "SELECT * FROM students ";
        try (Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(displayQuery))
{


            System.out.println("ID | Name | Grade Level | GPA");
            while (rs.next())
{

                System.out.printf("%d | %s | %s | %.2f%n",
                        rs.getInt("student_id"),
                        rs.getString("name"),
                        rs.getString("grade_level"),
                        rs.getDouble("gpa"));


}

} catch (SQLException e)
{

            System.out.println("Error displaying students: " + e.getMessage());


}


}
}
```

Status : Correct **Marks : 10/10**