Rajalakshmi Engineering College

Name: GAYATHRI MOHANRAAJ
Email: 240701143@rajalakshmi.edu.in
Roll no: 2116240701143
Phone: 8610319532
Branch: REC
Department: CSE - Section 9
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 : 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 Inventory Management System that handles runtime input to manage items in an inventory. The system should allow users to:

Add a new item (item ID, name, quantity, price).

Restock an item by increasing its quantity.

Reduce the stock of an item, ensuring sufficient quantity.

Display all items in the inventory in a sorted order by item ID.

Exit the application.

Half of the code is given here; Only the remaining part should be completed.

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 items table has already been created with the following structure:

Table Name: items

Input Format

The first line of input consists of an integer choice, representing the operation to be performed (1 for Add Item, 2 for Restock item, 3 for reduce item, 4 for Display, 5 for Exit).

For choice 1 (Add Item):

- The second line consists of an integer item_id.
- The third line consists of a string name.
- The fourth line consists of an integer quantity.
- The fifth line consists of a double price.

For choice 2 (Restock Item):

- The second line consists of an integer item_id.
- The third line consists of an integer quantity_to_add (must be positive).

For choice 3 (Reduce Stock):

- The second line consists of an integer item_id.
- The third line consists of an integer quantity_to_remove (must be positive).

For choice 4 (Display Inventory):

- No additional inputs are required.

For choice 5 (Exit):

- No additional inputs are required.

Output Format

For choice 1 (Add Item):

- Print "Item added successfully" if the item was added.
- Print "Failed to add item." if the insertion failed.

For choice 2 (Restock Item):

- Print "Item restocked successfully" if the restock was successful.
- Print "Item not found." if the specified item ID does not exist.

For choice 3 (Reduce Stock):

- Print "Stock reduced successfully" if the stock reduction was successful.
- Print "Not enough stock to remove." if there is insufficient quantity.
- Print "Item not found." if the specified item ID does not exist.

For choice 4 (Display Inventory):

- Display each item on a new line in the format:
- ID | Name | Quantity | Price
- If no items are available, print nothing (or handle with an appropriate message if desired).

For choice 5 (Exit):

- Print "Exiting Inventory Management System."

For invalid input:

- Print "Invalid choice. Please try again."

Sample Test Case

Input: 1
101
Laptop
50
1200.00
4
5
Output: Item added successfully
ID | Name | Quantity | Price
101 | Laptop | 50 | 1200.00
Exiting Inventory Management System.

Answer

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

class InventoryManagementSystem {
   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:
                    addItem(conn, scanner);
                    break;
                case 2:
                    restockItem(conn, scanner);
                    break;
                case 3:
                    reduceStock(conn, scanner);
                    break;
                case 4:
                    displayInventory(conn);
                    break;
                case 5:
                    System.out.println("Exiting Inventory Management System.");
                    running = false;
                    break;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
public static void addItem(Connection conn, Scanner scanner) {
    int itemId = scanner.nextInt();
    scanner.nextLine();

    String name = scanner.nextLine();

    int quantity = scanner.nextInt();

    double price = scanner.nextDouble();

    String insertQuery = "INSERT INTO items (item_id, name, quantity, price)
VALUES (?, ?, ?, ?)";
    try (PreparedStatement stmt = conn.prepareStatement(insertQuery)) {
        stmt.setInt(1, itemId);
        stmt.setString(2, name);
```

```
            stmt.setInt(3, quantity);
            stmt.setDouble(4, price);

            int rowsInserted = stmt.executeUpdate();
            System.out.println(rowsInserted > 0 ? "Item added successfully" : "Failed
to add item.");
        } catch (SQLException e) {
            System.out.println("Error adding item: " + e.getMessage());
        }
    }

    public static void restockItem(Connection conn, Scanner scanner) {
        int itemId = scanner.nextInt();

        int quantityToAdd = scanner.nextInt();

        // Check if the quantity is positive
        if (quantityToAdd <= 0) {
            System.out.println("Quantity to add must be positive.");
            return;
        }

        String updateQuery = "UPDATE items SET quantity = quantity + ? WHERE
item_id = ?";
        try (PreparedStatement stmt = conn.prepareStatement(updateQuery)) {
            stmt.setInt(1, quantityToAdd);
            stmt.setInt(2, itemId);

            int rowsUpdated = stmt.executeUpdate();
            System.out.println(rowsUpdated > 0 ? "Item restocked successfully" :
"Item not found.");
        } catch (SQLException e) {
            System.out.println("Error during restock: " + e.getMessage());
        }
    }

    public static void reduceStock(Connection conn, Scanner scanner) {
        int itemId = scanner.nextInt();

        int quantityToRemove = scanner.nextInt();

        // Check if the quantity is positive
```

```
        if (quantityToRemove <= 0) {
            System.out.println("Quantity to remove must be positive.");
            return;
        }

        String checkQuantityQuery = "SELECT quantity FROM items WHERE item_id
= ?";
        String updateQuery = "UPDATE items SET quantity = quantity - ? WHERE
item_id = ?";

        try (PreparedStatement checkStmt =
conn.prepareStatement(checkQuantityQuery)) {
            checkStmt.setInt(1, itemId);
            ResultSet rs = checkStmt.executeQuery();

            if (rs.next()) {
                int currentQuantity = rs.getInt("quantity");

                if (currentQuantity >= quantityToRemove) {
                    try (PreparedStatement stmt =
conn.prepareStatement(updateQuery)) {
                        stmt.setInt(1, quantityToRemove);
                        stmt.setInt(2, itemId);

                        int rowsUpdated = stmt.executeUpdate();
                        System.out.println(rowsUpdated > 0 ? "Stock reduced
successfully" : "Failed to reduce stock.");
                    }
                } else {
                    System.out.println("Not enough stock to remove.");
                }
            } else {
                System.out.println("Item not found.");
            }
        } catch (SQLException e) {
            System.out.println("Error during stock reduction: " + e.getMessage());
        }
    }

    public static void displayInventory(Connection conn) {
        String displayQuery = "SELECT * FROM items ORDER BY item_id";
        try (Statement stmt = conn.createStatement();
```

```
         ResultSet rs = stmt.executeQuery(displayQuery)) {

            System.out.println("ID | Name | Quantity | Price");
            while (rs.next()) {
                System.out.printf("%d | %s | %d | %.2f%n",
                        rs.getInt("item_id"),
                        rs.getString("name"),
                        rs.getInt("quantity"),
                        rs.getDouble("price"));
            }
        } catch (SQLException e) {
            System.out.println("Error displaying inventory: " + e.getMessage());
        }
    }

}
```

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