

Rajalakshmi Engineering College

Name: Mukesh Raghavendhar N
Email: 241501120@rajalakshmi.edu.in
Roll no:
Phone: 9342823404
Branch: REC
Department: AI & ML - Section 4
Batch: 2028
Degree: B.E - AI & ML

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 : 0

Section 1 : Project

1. 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();

    String name = scanner.nextLine();

    String gradeLevel = scanner.nextLine();

    double gpa = scanner.nextDouble();

    if (gpa < 0.0 || gpa > 4.0) {
        ;
        return;
    }

    String query = "INSERT INTO students (student_id, name, grade_level, gpa)
VALUES (?, ?, ?, ?)";
    try (PreparedStatement stmt = conn.prepareStatement(query)) {
        stmt.setInt(1, studentId);
        stmt.setString(2, name);
        stmt.setString(3, gradeLevel);
        stmt.setDouble(4, gpa);
        int rowsAffected = stmt.executeUpdate();
        if (rowsAffected > 0) {
            System.out.println("Student added successfully");
        } else {
            System.out.println("Failed to add student.");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public static void updateGrades(Connection conn, Scanner scanner) {

    int studentId = scanner.nextInt();

```

```

double newGpa = scanner.nextDouble();

if (newGpa < 0.0 || newGpa > 4.0) {

    return;
}

String query = "UPDATE students SET gpa = ? WHERE student_id = ?";
try (PreparedStatement stmt = conn.prepareStatement(query)) {
    stmt.setDouble(1, newGpa);
    stmt.setInt(2, studentId);
    int rowsAffected = stmt.executeUpdate();
    if (rowsAffected > 0) {
        System.out.println("GPA updated successfully");
    } else {
        System.out.println("Student not found.");
    }
} catch (SQLException e) {
    e.printStackTrace();
}
}

public static void viewStudentRecord(Connection conn, Scanner scanner) {
    System.out.println("Enter student ID:");
    int studentId = scanner.nextInt();

    String query = "SELECT * FROM students WHERE student_id = ?";
    try (PreparedStatement stmt = conn.prepareStatement(query)) {
        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) {
        e.printStackTrace();
    }
}

```

```

    }
}

public static void displayAllStudents(Connection conn) {
    String query = "SELECT * FROM students";
    try (Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(query)) {
        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) {
        e.printStackTrace();
    }
}
}

```

Status : Wrong

Marks : 0/10

2. Problem Statement

In Café Central, the menu is cataloged and stored in a database.

To efficiently manage the restaurant's menu using Java and JDBC, you must build a Restaurant Management System that supports:

Adding new menu items

Updating menu item prices

Viewing details of a menu item

Displaying all menu items in sorted order

You are given two files:

File 1: MenuItem.java (POJO Class)

This class represents the MenuItem entity.

A MenuItem contains the following details:

Field Description

itemId Unique Menu Item ID (Integer)

name Item Name (String)

category Item Category (String)

price Item Price (Double)

Students must write code in the marked area:

```
class MenuItem {  
    private int itemId;  
    private String name;  
    private String category;  
    private double price;  
  
    public MenuItem() {}  
  
    public MenuItem(int itemId, String name, String category, double price) {  
        // 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: MenuItemDAO.java (Data Access Layer)

This class handles all database operations using JDBC.

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

```
class MenuItemDAO {

    public void addItem(Connection conn, MenuItem menuItem)
    throws SQLException {
        // write your code here
    }

    public void updateItemPrice(Connection conn, int itemId, double
    newPrice) throws SQLException {
        // write your code here
    }

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

    public MenuItem viewItemDetails(Connection conn, int itemId) throws
    SQLException {
        // write your code here
    }

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

    private MenuItem mapToMenuItem(ResultSet rs) throws SQLException {
        return new MenuItem(
```

```
        // 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 MenuItem objects using mapToMenuItem().

Return a List<MenuItem> where required.

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 menu table has already been created with the following structure:

Table Name: menu

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 Menu Item):

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

For choice 2 (Update Item Price):

- The second line consists of an integer item_id.
- The third line consists of a double new_price.

For choice 3 (View Item Details):

- The second line consists of an integer item_id.

For choice 4 (Display All Menu Items):

- No additional inputs are required.

For choice 5 (Exit):

- No additional inputs are required.

Output Format

For choice 1 (Add Menu Item):

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

For choice 2 (Update Item Price):

- Print "Item price updated successfully" if the price update was successful.
- Print "Item not found." if the specified item ID does not exist.

For choice 3 (View Item Details):

- Display the item details in the format:
- ID: [item_id] | Name: [name] | Category: [category] | Price: [price]
- Print "Item not found." if the specified item ID does not exist.

For choice 4 (Display All Menu Items):

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

For choice 5 (Exit):

- Print "Exiting Restaurant Management System."

For invalid input:

- Print "Invalid choice. Please try again."

Sample Test Case

Input: 1

11

Margherita Pizza

Main Course

12.99

4

5

Output: Menu item added successfully

ID | Name | Category | Price

11 | Margherita Pizza | Main Course | 12.99

Exiting Restaurant Management System.

Answer

```
import java.sql.*;
```

```
import java.util.Scanner;
```

```
class RestaurantManagementSystem {
```

```
    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:
```

```
                            addMenuItem(conn, scanner);
```

```
                            break;
```

```
                        case 2:
```

```
                            updateItemPrice(conn, scanner);
```

```
                            break;
```

```

        case 3:
            viewItemDetails(conn, scanner);
            break;
        case 4:
            displayAllMenuItems(conn);
            break;
        case 5:
            System.out.println("Exiting Restaurant Management System.");
            running = false;
            break;
        default:
            System.out.println("Invalid choice. Please try again.");
    }
}
} catch (SQLException e) {
    e.printStackTrace();
}
}

```

// You are using Java

public void addItem(Connection conn, MenuItem menuItem) throws
SQLException {

String sql = "INSERT INTO menu (itemId, name, category, price) VALUES
(?, ?, ?, ?)";

```

    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, menuItem.getItemId());
        pstmt.setString(2, menuItem.getName());
        pstmt.setString(3, menuItem.getCategory());
        pstmt.setDouble(4, menuItem.getPrice());
        int rowsAffected = pstmt.executeUpdate();
        if (rowsAffected > 0) {
            System.out.println("Menu item added successfully");
        } else {
            System.out.println("Failed to add item.");
        }
    }
}

```

// Method to update the price of a menu item

public void updateItemPrice(Connection conn, int itemId, double newPrice)
throws SQLException {

String sql = "UPDATE menu SET price = ? WHERE itemId = ?";

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

```

        pstmt.setDouble(1, newPrice);
        pstmt.setInt(2, itemId);
        int rowsAffected = pstmt.executeUpdate();
        if (rowsAffected > 0) {
            System.out.println("Item price updated successfully");
        } else {
            System.out.println("Item not found.");
        }
    }
}

```

```

// Method to delete a menu item
public void deleteMenuItem(Connection conn, int itemId) throws
SQLException {
    String sql = "DELETE FROM menu WHERE itemId = ?";
    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, itemId);
        pstmt.executeUpdate();
    }
}

```

```

// Method to view details of a menu item
public MenuItem viewItemDetails(Connection conn, int itemId) throws
SQLException {
    String sql = "SELECT * FROM menu WHERE itemId = ?";
    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, itemId);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return mapToMenuItem(rs);
        } else {
            System.out.println("Item not found.");
            return null;
        }
    }
}

```

```

// Method to display all menu items
public List<MenuItem> displayAllMenuItems(Connection conn) throws
SQLException {
    List<MenuItem> menuItems = new ArrayList<>();
    String sql = "SELECT * FROM menu ORDER BY name";
}

```

```

        try (Statement stmt = conn.createStatement(); ResultSet rs =
stmt.executeQuery(sql)) {
            while (rs.next()) {
                menuItems.add(mapToMenuItem(rs));
            }
        }
        return menuItems;
    }

```

```

class MenuItem {
    private int itemId;
    private String name;
    private String category;
    private double price;
}

```

```

}

```

```

}

```

```

// Getter for itemId
public int getItemId() {
    return itemId;
}

```

```

// Setter for itemId
public void setItemId(int itemId) {
    this.itemId = itemId;
}

```

```

// Getter for name
public String getName() {
    return name;
}

```

```

// Setter for name
public void setName(String name) {
    this.name = name;
}

```

```
}

// Getter for category
public String getCategory() {
    return category;
}

// Setter for category
public void setCategory(String category) {
    this.category = category;
}

// Getter for price
public double getPrice() {
    return price;
}

// Setter for price
public void setPrice(double price) {
    this.price = price;
}

}

//
```

Status : Wrong

Marks : 0/10