**RECIPE RECOMMENDATION SYSTEM**

**ABSTRACT:**

This project is a **Console-Based Recipe Recommendation System** designed to recommend recipes to users based on ingredients they have on hand. Implemented in **Core Java** with **MySQL** as the database, the system enables users to input available ingredients, retrieving a list of possible recipes that can be prepared. It leverages Object-Oriented Programming (OOP) principles, database management, and multithreading to enhance functionality and performance.

**Key Components and Requirements**

**Java Application**

1. **Core Classes and Inheritance**:
   * **Classes** like Recipe, Ingredient, User, and Recommendation will represent primary entities.
   * **Subclasses** VegetarianRecipe and NonVegetarianRecipe inherit from Recipe, using polymorphism to handle different types of recipes.
2. **Interface and Method Implementation**:
   * An interface, Recommendable, will define essential methods:
     + findRecipes(): retrieves recipes based on user-provided ingredients.
     + updateInventory(): updates available ingredients for users.
     + viewRecipeDetails(): displays detailed information about a selected recipe.
3. **Collections and Data Management**:
   * Use **collections** to store the user’s ingredient list and recipe data in memory, optimizing search and recommendation processes.
4. **Custom Exceptions and Input Validation**:
   * Define a custom exception, IngredientNotAvailableException, to handle cases where the necessary ingredients are unavailable.
   * Validate user inputs to ensure accurate ingredient entries, minimizing errors in the recommendation process.
5. **File Handling and Logging**:
   * Implement file handling to log each user’s recommended recipes, supporting a record of previous recommendations.
6. **Database Interaction with JDBC**:
   * Use **JDBC** to interact with the MySQL database for persisting recipes, ingredient details, and user information.
7. **Multithreading for Concurrency**:
   * Utilize **multithreading** to handle concurrent recipe recommendation requests, improving system responsiveness and performance.

**Database (MySQL) Requirements**

1. **Database Structure and Relationships**:
   * Define normalized tables for Recipe, Ingredient, User, and a **junction table** RecipeIngredient to link recipes with their required ingredients.
   * Implement **primary and foreign keys** to ensure data integrity and relational consistency.
2. **CRUD Operations and Stored Procedures**:
   * Provide **CRUD operations** for managing recipes, ingredients, and user data.
   * Create **stored procedures** for adding and updating recipes, ensuring streamlined and consistent data manipulation.
3. **Advanced SQL Features**:
   * Use **JOIN queries** to retrieve recipes based on available ingredients efficiently.
   * Define **triggers** to monitor inventory updates, keeping track of ingredient availability in real-time.
4. **ACID Compliance and Performance Optimization**:
   * Ensure **ACID properties** (Atomicity, Consistency, Isolation, Durability) to maintain consistent recipe and ingredient data.
   * Add indexes on frequently accessed columns, such as recipe\_id and ingredient\_id, for optimized data retrieval.
5. **Views and Functions**:
   * Create a **view** (AvailableRecipes) to display recipes based on the user’s available ingredients.
   * Implement **functions** to calculate total preparation time, factoring in the ingredient list for each recipe.

**CODE:**

**Java :**

package recipyrecommendation;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class RecipeRecommendationSystem{

public static void main(String[] args) {

try (Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/recipe\_db", "root", "gaya@#1234@#")) {

RecipeService recipeService = new RecipeService(connection);

try (Scanner scanner = new Scanner(System.in)) {

System.out.println("Enter the number of ingredients you have:");

int ingredientCount = scanner.nextInt();

scanner.nextLine();

List<Ingredient> userIngredients = new ArrayList<>();

for (int i = 0; i < ingredientCount; i++) {

System.out.println("Enter ingredient name:");

String name = scanner.nextLine();

System.out.println("Enter ingredient quantity:");

double quantity = scanner.nextDouble();

scanner.nextLine();

userIngredients.add(new Ingredient(name, quantity));

}

try {

List<Recipe> recipes = recipeService.findRecipes(userIngredients);

System.out.println("Recommended Recipes:");

for (Recipe recipe : recipes) {

System.out.println(recipe);

}

} catch (IngredientNotAvailableException e) {

System.out.println(e.getMessage());

}

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

class Recipe {

private int id;

private String name;

private String instructions;

public Recipe(int id, String name, String instructions) {

this.id = id;

this.name = name;

this.instructions = instructions;

}

@Override

public String toString() {

return "Recipe ID: " + id + ", Name: " + name + ", Instructions: " + instructions;

}

}

class Ingredient {

private String name;

private double quantity;

public Ingredient(String name, double quantity) {

this.name = name;

this.quantity = quantity;

}

public String getName() {

return name;

}

public double getQuantity() {

return quantity;

}

}

class IngredientNotAvailableException extends Exception {

public IngredientNotAvailableException(String message) {

super(message);

}

}

interface Recommendable {

List<Recipe> findRecipes(List<Ingredient> userIngredients) throws IngredientNotAvailableException;

}

class RecipeService implements Recommendable {

private Connection connection;

public RecipeService(Connection connection) {

this.connection = connection;

}

@Override

public List<Recipe> findRecipes(List<Ingredient> userIngredients) throws IngredientNotAvailableException {

List<Recipe> recommendedRecipes = new ArrayList<>();

StringBuilder ingredientNames = new StringBuilder();

for (Ingredient ingredient : userIngredients) {

ingredientNames.append("'").append(ingredient.getName()).append("', ");

}

if (ingredientNames.length() > 0) {

ingredientNames.setLength(ingredientNames.length() - 2); // Remove last comma and space

}

String query = "SELECT DISTINCT r.recipe\_id, r.name, r.instructions " +

"FROM Recipe r " +

"JOIN RecipeIngredient ri ON r.recipe\_id = ri.recipe\_id " +

"JOIN Ingredient i ON ri.ingredient\_id = i.ingredient\_id " +

"WHERE i.name IN (" + ingredientNames + ") " +

"GROUP BY r.recipe\_id";

try (Statement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(query)) {

while (rs.next()) {

int recipeId = rs.getInt("recipe\_id");

String recipeName = rs.getString("name");

String instructions = rs.getString("instructions");

Recipe recipe = new Recipe(recipeId, recipeName, instructions);

recommendedRecipes.add(recipe);

logRecommendation(recipe);

}

} catch (SQLException e) {

e.printStackTrace();

}

if (recommendedRecipes.isEmpty()) {

throw new IngredientNotAvailableException("No recipes found with the provided ingredients.");

}

return recommendedRecipes;

}

private void logRecommendation(Recipe recipe) {

System.out.println("Logging recommendation: " + recipe);

}

}

**SQL CODE:**

DROP TABLE IF EXISTS RecipeIngredient;

DROP TABLE IF EXISTS Ingredient;

DROP TABLE IF EXISTS Recipe;

CREATE TABLE Recipe (

recipe\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(255) NOT NULL,

instructions TEXT

);

CREATE TABLE Ingredient (

ingredient\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(255) UNIQUE NOT NULL

);

CREATE TABLE RecipeIngredient (

recipe\_id INT,

ingredient\_id INT,

quantity DOUBLE,

FOREIGN KEY (recipe\_id) REFERENCES Recipe(recipe\_id),

FOREIGN KEY (ingredient\_id) REFERENCES Ingredient(ingredient\_id)

);

INSERT INTO Recipe (name, instructions) VALUES

('Pasta', '1. Boil water in a large pot.\n2. Add pasta and cook until tender.\n3. Drain and set aside.\n4. In a saucepan, heat the tomato sauce with garlic and basil.\n5. Combine pasta with sauce, and mix well before serving.'),

('Salad', '1. Chop the lettuce and place it in a large bowl.\n2. Add diced tomatoes, cucumber, and cheese.\n3. Toss with olive oil, salt, and pepper.\n4. Serve fresh, topped with optional croutons.'),

('Grilled Cheese Sandwich', '1. Spread butter on one side of each bread slice.\n2. Place cheese slices between two slices of bread (butter side out).\n3. Heat a skillet and place the sandwich on it.\n4. Grill each side until golden brown and cheese is melted.'),

('Omelette', '1. Crack eggs into a bowl and whisk until fluffy.\n2. Add a pinch of salt and pepper.\n3. Heat a pan with a bit of oil or butter.\n4. Pour the eggs into the pan and add diced tomatoes and cheese.\n5. Fold the omelette and serve hot.'),

('Fruit Smoothie', '1. Add yogurt, banana, and mixed berries to a blender.\n2. Pour in orange juice and honey.\n3. Blend until smooth and creamy.\n4. Pour into a glass and enjoy as a refreshing drink.');

INSERT INTO Ingredient (name) VALUES

('Pasta'),

('Tomato'),

('Lettuce'),

('Cheese'),

('Bread'),

('Butter'),

('Eggs'),

('Cucumber'),

('Olive Oil'),

('Banana'),

('Yogurt'),

('Mixed Berries'),

('Orange Juice'),

('Honey');

-- Insert Recipe-Ingredient relationships with quantities

INSERT INTO RecipeIngredient (recipe\_id, ingredient\_id, quantity) VALUES

(1, 1, 200),

(1, 2, 50),

(1, 9, 10),

(2, 3, 100),

(2, 2, 50),

(2, 4, 30),

(2, 8, 30),

(2, 9, 5),

(3, 5, 2),

(3, 4, 50),

(3, 6, 10),

(4, 7, 2),

(4, 2, 30),

(4, 4, 20),

(5, 10, 1),

(5, 11, 100),

(5, 12, 50),

(5, 13, 150),

(5, 14, 10);

SELECT DISTINCT r.recipe\_id, r.name, r.instructions

FROM Recipe r

JOIN RecipeIngredient ri ON r.recipe\_id = ri.recipe\_id

JOIN Ingredient i ON ri.ingredient\_id = i.ingredient\_id

WHERE i.name IN ('Pasta', 'Tomato')

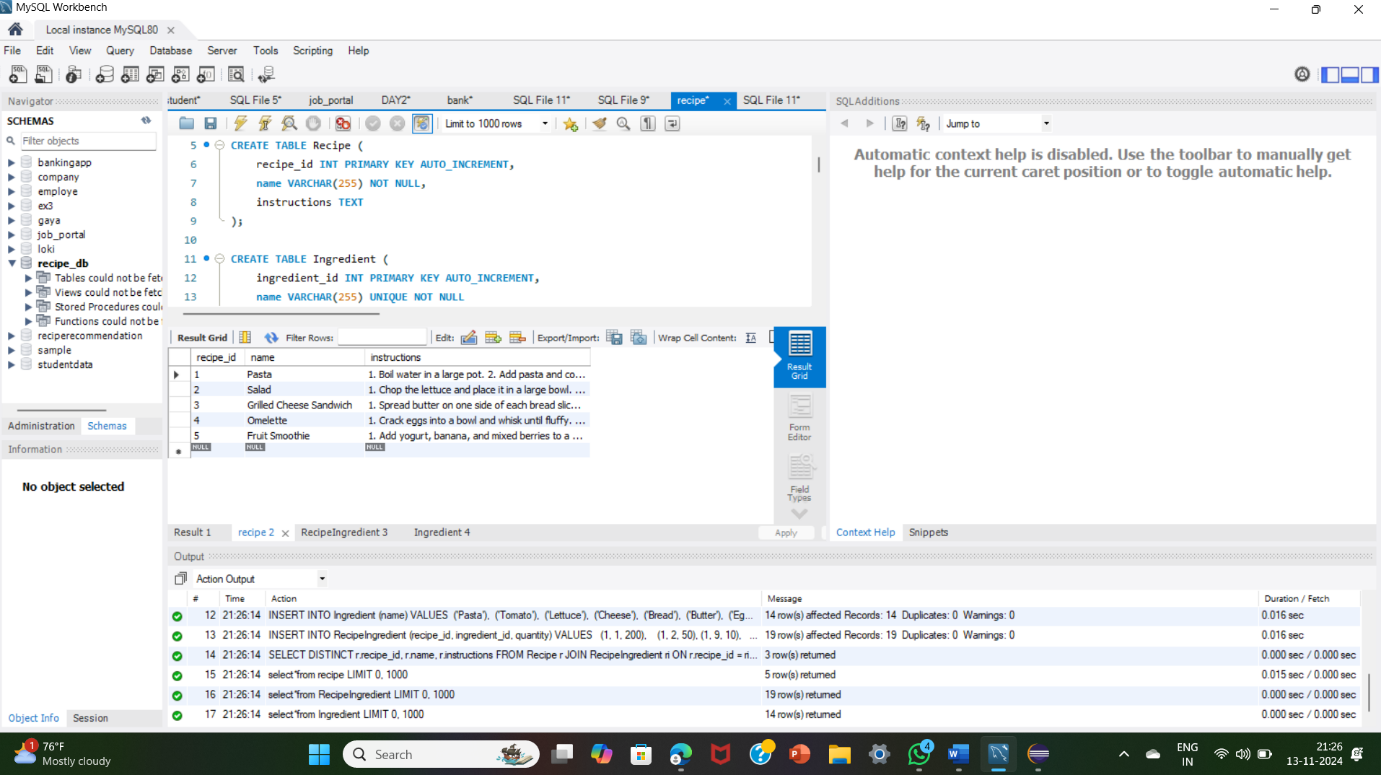
GROUP BY r.recipe\_id;

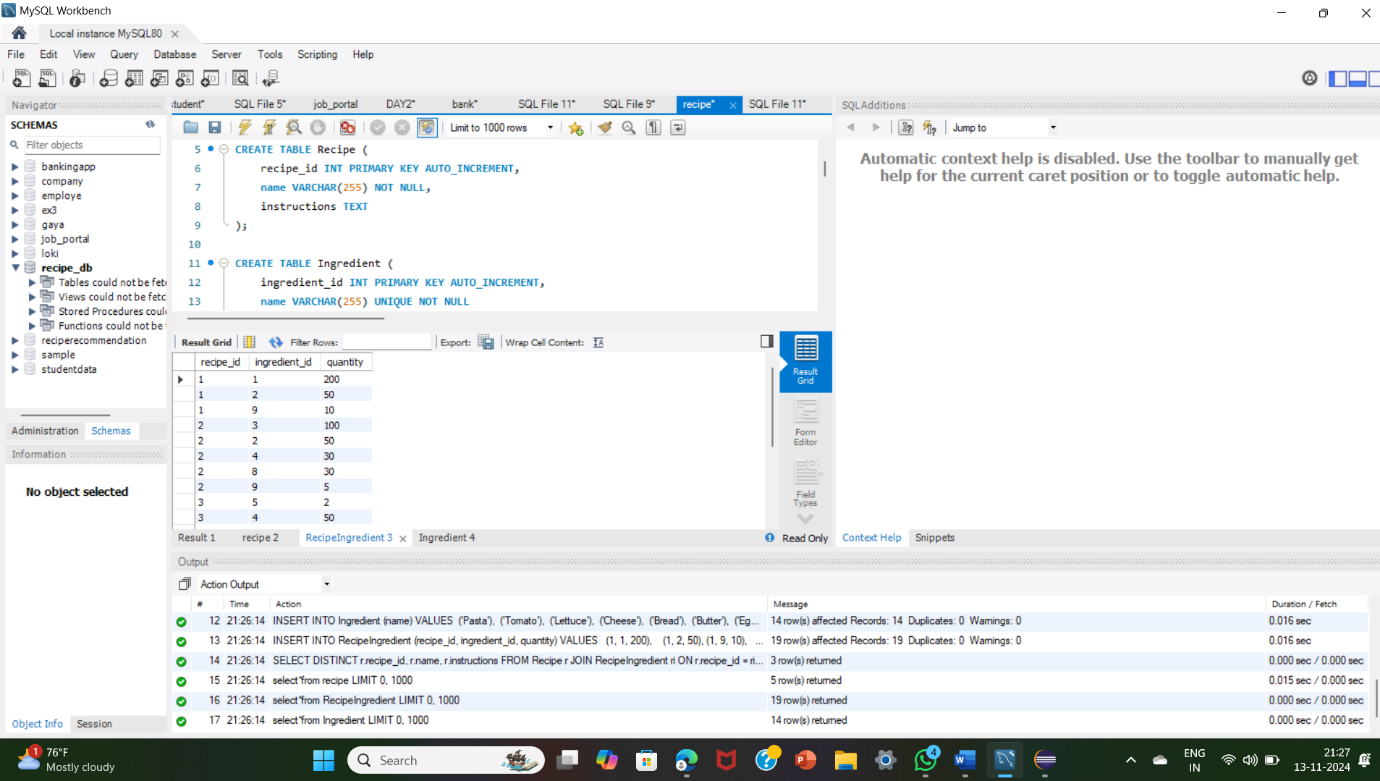
select\*from recipe;

select\*from RecipeIngredient;

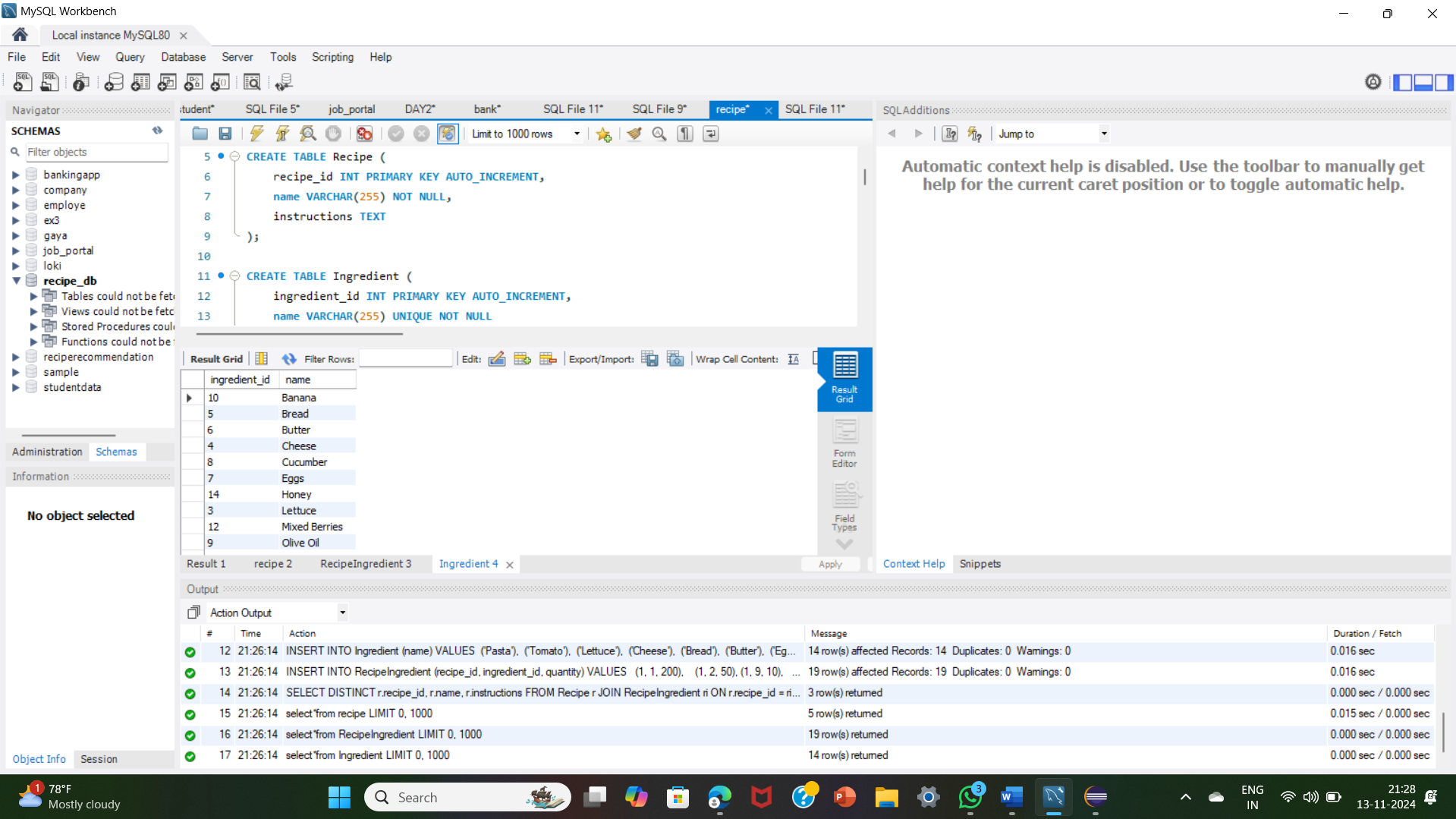
select\*from Ingredient;

**OUTPUT :**

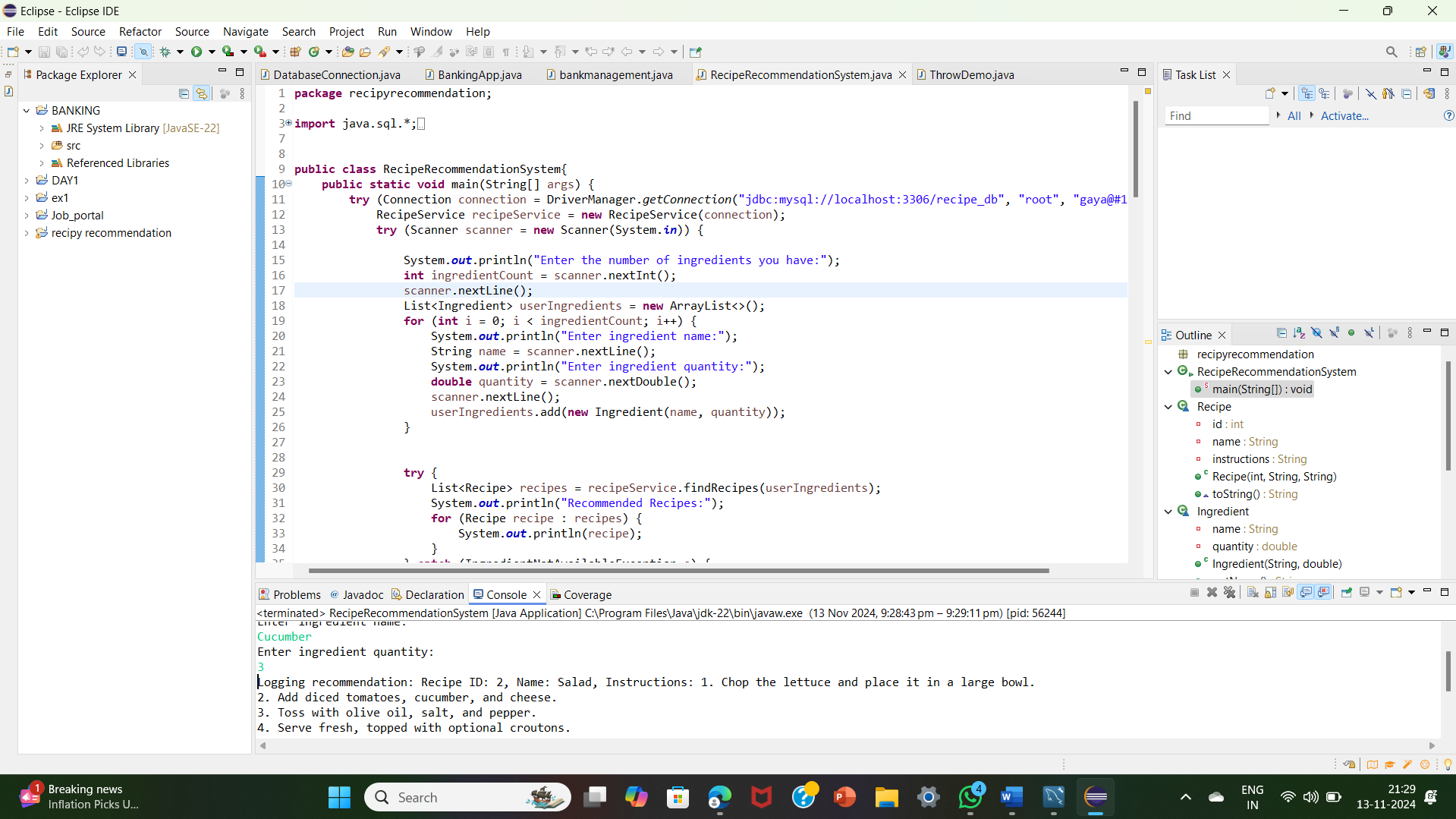
**Recipe table :**

**RecipeIngredient Table:**

**Ingredient:**

****

**RecipeRecommendationSystem:**

****