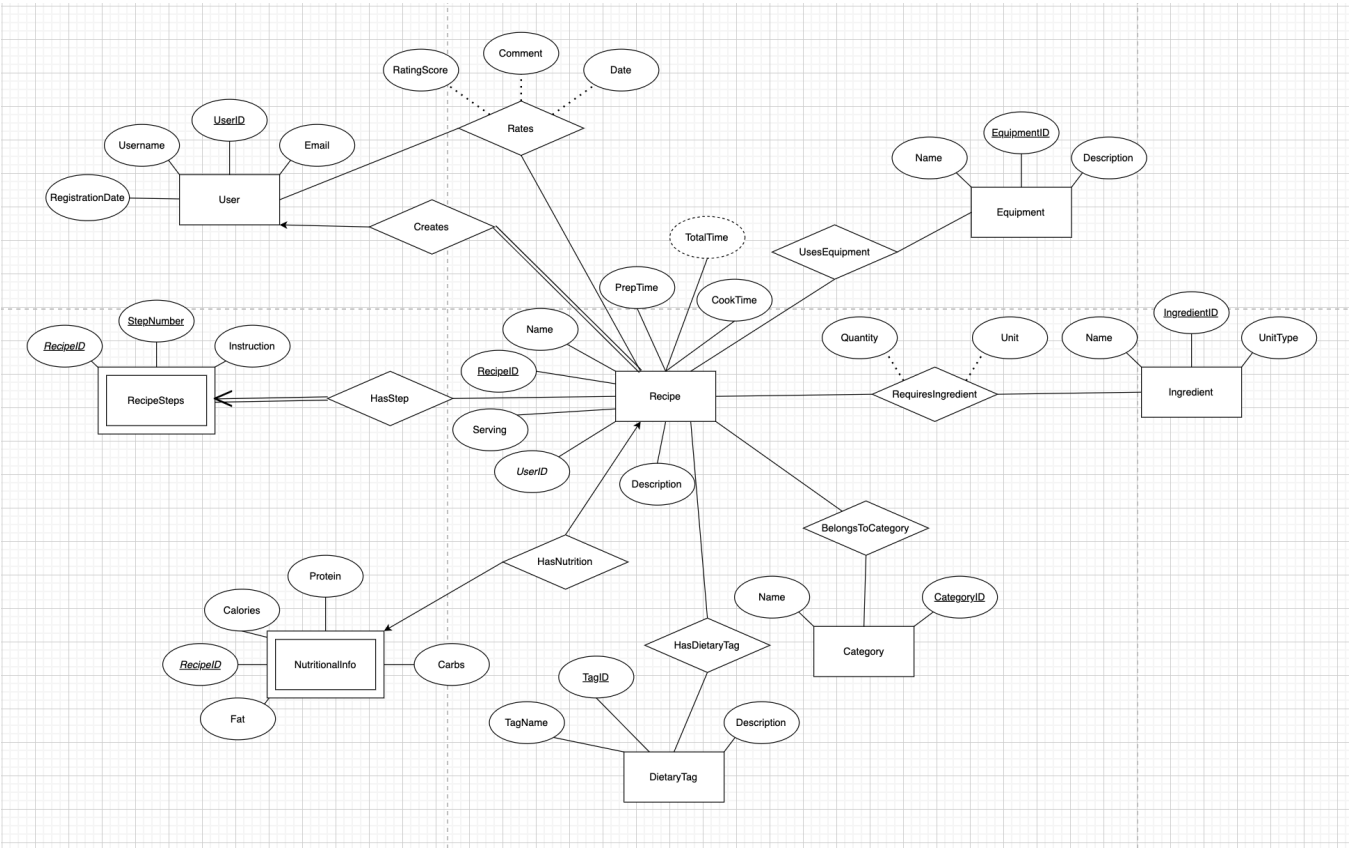
CSI 508- Database Systems
Project Milestone-1

Sai Satwik Bikumandla - 001655014
Hari Pavan Reddy Bojjam - 001605887
Rishikesh Sirisilla - 001661448
Akshitha Janapana - 001615320

Part 1: E-R Diagram

Entities & Relationships:



Assumptions:

1. A recipe must have at least one ingredient and one category.
2. Nutritional information is mandatory for every recipe.

Entity Sets:

Entity	Type	Description
User	Strong	Represents users who create recipes.
Recipe	Strong	Stores recipe details (name, prep time, etc.). Depends on User.
Ingredient	Strong	Unique ingredients (e.g., "Flour", "Sugar").
Category	Strong	Recipe categories (e.g., "Dessert", "Italian").
Equipment	Strong	Cooking tools (e.g., "Oven", "Blender").
DietaryTag	Strong	Reusable dietary tags (e.g., "Vegetarian", "Gluten-Free").
RecipeSteps	Weak	Steps for a recipe. Existence depends on Recipe.
NutritionalInfo	Weak	Nutritional data for a recipe. Existence depends on Recipe.

Relationship Sets:

Relationship	Entities Involved	Description
Creates	User (1) – Recipe (N)	A user creates multiple recipes.
RequiresIngredient	Recipe (N) – Ingredient (M)	A recipe uses multiple ingredients.
BelongsToCategory	Recipe (N) – Category (M)	A recipe can belong to multiple categories.
UsesEquipment	Recipe (N) – Equipment (M)	A recipe requires multiple equipment items.

HasStep	Recipe (1) – RecipeSteps (N)	A recipe has multiple steps.
HasNutrition	Recipe (1) – NutritionalInfo (1)	A recipe has nutritional info (1:1).
HasDietaryTag	Recipe (N) – DietaryTag (M)	A recipe can have multiple dietary tags.
Rates	User (N) – Recipe (M)	Users can rate multiple recipes.

Note: Subclassing is not applicable for our project. Because, all entities are standalone with no hierarchical specialization (e.g., no subtypes like PremiumUser or AdminUser).

Attribute Types:

Entity	**Single-Valued**	**Multi-Valued**	**Derived**	**Composite**
User	UserID, Email, Username	None	None	None
Recipe	RecipeID, Name	None	TotalTime (PrepTime + CookTime)	None
Ingredient	IngredientID, Name	None	None	None
RecipeSteps	StepNumber	Instruction (stored as steps)	None	None
NutritionalInfo	Calories, Protein	None	None	None

Example for Derived Attribute:

```
-- TotalTime is not stored but derived from PrepTime + CookTime
SELECT Name, (PrepTime + CookTime) AS TotalTime FROM Recipe;
```

Cardinality:

Relationship	Cardinality	Description
Creates	User (1) – Recipe (N)	One user creates many recipes; each recipe has one user.
HasStep	Recipe (1) – RecipeSteps (N)	One recipe has many steps; each step belongs to one recipe.
HasNutrition	Recipe (1) – NutritionalInfo (1)	One-to-one relationship.
RequiresIngredient	Recipe (N) – Ingredient (M)	Many-to-many (via RecipeIngredient).
Rates	User (N) – Recipe (M)	Many-to-many (via UserRating).

Mandatory Relationships (Total Participation):

Entity	Mandatory	Justification
Recipe	Yes	A recipe must have a UserID (NOT NULL foreign key).
RecipeSteps	Yes	A recipe must have at least one step (modeled via weak entity dependency).
NutritionalInfo	Optional	Not all recipes require nutritional data.
UserRating	Optional	A recipe can exist without ratings.

Why there are no other Mandatory Total Participation Relations?

Ingredients, categories, and equipment are optional for flexibility (e.g., a recipe might not require special equipment). Dietary tags are optional to accommodate recipes with no dietary restrictions.

Design Justifications:

- **Weak Entities:** RecipeSteps and NutritionalInfo are not possible without a Recipe (total participation).
- **Normalisation:** Removed redundancy (e.g., reusable DietaryTag instead of recipe-specific tags).
- **Flexibility:** Optional relationships for ingredients/categories enable recipe drafts with missing ingredients.

Part 2: Schema Statements

Strong Entities

1. **User**(UserID, Username, Email, RegistrationDate)
2. **Recipe**(RecipeID, Name, Description, PrepTime, CookTime, Servings, UserID (FK- User))
3. **Ingredient**(IngredientID, Name, UnitType)
4. **Category**(CategoryID, Name)
5. **Equipment**(EquipmentID, Name, Description)
6. **DietaryTag**(TagID, TagName, Description)

Weak Entities

7. **RecipeSteps**(RecipeID (FK-Recipe), StepNumber, Instruction)
8. **NutritionalInfo**(RecipeID (FK- Recipe), Calories, Protein, Carbs, Fat)

Relationship Tables

9. **RecipeIngredient**(RecipeID (FK- Recipe), IngredientID (FK- Ingredient), Quantity, Unit)
10. **RecipeCategory**(RecipeID (FK- Recipe), CategoryID (FK- Category))
11. **RecipeEquipment**(RecipeID (FK- Recipe), EquipmentID (FK- Equipment))
12. **UserRating**(UserID (FK- User), RecipeID (FK- Recipe), RatingScore, Comment, Date)
13. **RecipeDietaryTag**(RecipeID (FK- Recipe), TagID (FK- DietaryTag))

Note: For clarity, we have decided to mention Foreign Keys within brackets that links to other entity.

Our schema statements strictly follow Part 2's grading rubric in translating the E-R diagram to a good relational database schema. We start the schema by declaring strong entities like User, Recipe, Ingredient, Category, Equipment, and DietaryTag with clear primary keys (e.g., UserID, RecipeID) and constraints. For instance, uniqueness is applied to Username and Email in the User table, and Recipe has a non-null foreign key (UserID) so that each recipe must be for an existing user, illustrating the total participation of the "creates" relationship. Strong entities are used to prevent redundancy, with Name and Description as single-valued attributes, following normalization rules.

The weak entities, RecipeSteps and NutritionalInfo, are identified by composite primary keys that include foreign keys to their strong parent entity, Recipe. For example, RecipeSteps has (RecipeID, StepNumber) as its primary key, so steps do not exist independently of a recipe, thus total participation is ensured. Likewise, NutritionalInfo also has RecipeID as its primary and foreign key, establishing a 1:1 relationship with Recipe with optional participation since not all recipes need nutritional information. Both these design decisions adhere to the dependencies and constraints of the E-R diagram.

Relationship sets are implemented by junction tables that control cardinality and participation. For instance, RecipeIngredient represents the M:N relationship between Recipe and Ingredient with a composite primary key (RecipeID, IngredientID) along with additional fields such as Quantity and Unit to hold recipe-specific information. Likewise, UserRating connects User and Recipe and includes extra fields such as RatingScore (with a constraint of 1–5) and Date.

All relationships are cardinality-constrained: 1:N relationships (e.g., User → Recipe) are managed using foreign keys, and M:N relationships (e.g., Recipe ↔ Category) employ junction tables with composite keys.

Integrity constraints are always enforced. Primary keys and foreign keys rigidly establish entity relations, e.g., RecipeEquipment to Recipe and Equipment. Unique constraints enforce uniqueness on important fields like Category.Name and DietaryTag.TagName, and NOT NULL constraints on key attributes like Recipe.Name and Instruction in RecipeSteps so that they are always filled. Derived attributes like TotalTime (which would be a sum of PrepTime + CookTime) are defined in Part 1 but not stored to prevent data redundancy.

Part 3: Normalization (3NF)

The schema created in Part 2 automatically conforms to Boyce-Codd Normal Form (BCNF), a strict normalization requirement that removes redundancy and has all functional dependencies controlled by candidate keys. The following is a detailed explanation of why the schema conforms to BCNF and why no additional decomposition is needed:

Functional Dependencies and BCNF Compliance

Every table in the schema is structured such that **all functional dependencies (FDs)** are explicitly tied to **candidate keys** (primary keys or unique identifiers). For Example:

- In the User table, UserID determines all the other columns (Username, Email, RegistrationDate) uniquely.
- In the Recipe table, RecipeID determines fields like Name, PrepTime, and CookTime with no transitive dependencies (e.g., PrepTime is independent of Servings).
- For relationship tables like RecipeIngredient, composite key (RecipeID, IngredientID) determines directly non-key fields Quantity and Unit with no partial or transitive dependencies.

Weak entities such as RecipeSteps and NutritionalInfo also obey BCNF. The composite key (RecipeID, StepNumber) of RecipeSteps ensures that each step's Instruction solely depends upon the recipe as well as its position number, and NutritionalInfo has RecipeID as its key, with a 1:1 dependency upon the parent entity Recipe.

Absence of BCNF Violations

BCNF requires that all determinants should be candidate keys. This is true for all tables:

- **No non-key determinants**: Columns like Username or Email in the User table are distinct but are not determinants of other columns.
- **No partial dependencies**: Composite keys like (RecipeID, IngredientID) in RecipeIngredient avoid non-key columns from relying on a subset instead of the entire key.
- **No transitive dependencies**: Columns like PrepTime and CookTime in Recipe rely only on RecipeID and not on intermediate columns.

Choice of BCNF Over 3NF

3NF eliminates transitive dependencies. BCNF, on the other hand, gives more guarantees as it ensures that each determinant is a candidate key. The schema achieves this inherently by design:

- **Strong entities** (e.g., User, Ingredient) are founded on atomic primary keys.
- **Weak entities** (e.g., RecipeSteps) make use of composite keys referencing their parent entities.
- **Relationship tables** (e.g., UserRating) enforce dependencies by means of composite keys.

Normalized Tables and Integrity Constraints

The schema tables are in BCNF already, and thus no further decomposition is needed, as all the functional dependencies adhere to candidate keys. Primary keys (e.g., UserID, RecipeID) ensure uniqueness between entities, while foreign keys (e.g., Recipe.UserID pointing to User.UserID) ensure referential integrity among related tables. Key fields like Username and Recipe.Name are restricted using UNIQUE and NOT NULL to prevent duplicates or null values, and ensure data consistency. This design naturally supports normalization requirements along with maintaining strong integrity constraints.