  
  
  
  
CS 353

Database Systems  
Project Proposal  
  
Project Name: Daily Food

Group 1

Pınar Bayata – 21401687

Özgür Can Erdoğan – 21300586

Kübra Nur Güzel – 21400946

Arif Can Terzioğlu - 21302061

Table of Contents

[1. Revised E/R Model 4](#_Toc498907812)

[2. Relation Schemas 7](#_Toc498907813)

[2.1. Account 7](#_Toc498907814)

[2.2. Message 8](#_Toc498907815)

[2.3. Member 9](#_Toc498907816)

[2.4. GoldMember 10](#_Toc498907817)

[2.5. Admin 11](#_Toc498907818)

[2.6. Category 12](#_Toc498907819)

[2.7. SubCategory 13](#_Toc498907820)

[2.8. Ingredient 14](#_Toc498907821)

[2.9. Comment 15](#_Toc498907822)

[2.10. Recipe 16](#_Toc498907823)

[2.11. Share 18](#_Toc498907824)

[2.12. Create 19](#_Toc498907825)

[2.13. Has 20](#_Toc498907826)

[2.14. Rate 21](#_Toc498907827)

[2.15. Follow 22](#_Toc498907828)

[2.16. Favorite 23](#_Toc498907829)

[3. Functional Dependencies and Normalization of Tables 24](#_Toc498907830)

[4. Functional Components 25](#_Toc498907831)

[4.1. Use Cases / Scenarios 25](#_Toc498907832)

[4.1.1. Admin 25](#_Toc498907833)

[4.1.2. Member and Gold Member 26](#_Toc498907834)

[4.2. Algorithms 27](#_Toc498907835)

[4.2.1. Rate Related Sorting Algorithms 27](#_Toc498907836)

[4.2.2. Search Algorithms 27](#_Toc498907837)

[4.2.3. Calorie Count Algorithm 28](#_Toc498907838)

[4.2.4. Logical Requirements 28](#_Toc498907839)

[4.3. Data Structures 28](#_Toc498907840)

[5. User Interface Design and Corresponding SQL Statements 29](#_Toc498907841)

[5.1. Homepage Screen 29](#_Toc498907842)

[5.1.1. Non-registered 29](#_Toc498907843)

[5.1.2. Registered – User 31](#_Toc498907844)

[5.2. Login and Signup 32](#_Toc498907845)

[5.3. Recipe Add Comment 33](#_Toc498907846)

[5.4. Profiles of Other Members 34](#_Toc498907847)

[5.5. Send Message 36](#_Toc498907848)

[5.6. User own Profile 37](#_Toc498907849)

[5.7. Search Screen 38](#_Toc498907850)

[5.8. Recipe View Screen 40](#_Toc498907851)

[5.9. Add Recipe Page 41](#_Toc498907852)

[5.10. Profile Edit 42](#_Toc498907853)

[6. Advanced Database Components 43](#_Toc498907854)

[6.1. Views 43](#_Toc498907855)

[6.1.1. Displaying the top 4 Recipes for Unregistered User 43](#_Toc498907856)

[6.1.2. Displaying the top 4 Recipes for Registered User 43](#_Toc498907857)

[6.1.3. Showing Other Members Favorites 44](#_Toc498907858)

[6.1.4. Chat Between Two Users 44](#_Toc498907859)

[6.1.5. Favorites of User’s own Profile 45](#_Toc498907860)

[6.1.6. Result of Search Screen 45](#_Toc498907861)

[6.2. Stored Procedures 46](#_Toc498907862)

[6.3. Reports 46](#_Toc498907863)

[6.3.1. Monthly Added Recipe Report By Members 46](#_Toc498907864)

[6.3.2. Total Number of Subcategories in Each Category 47](#_Toc498907865)

[6.4. Triggers 47](#_Toc498907866)

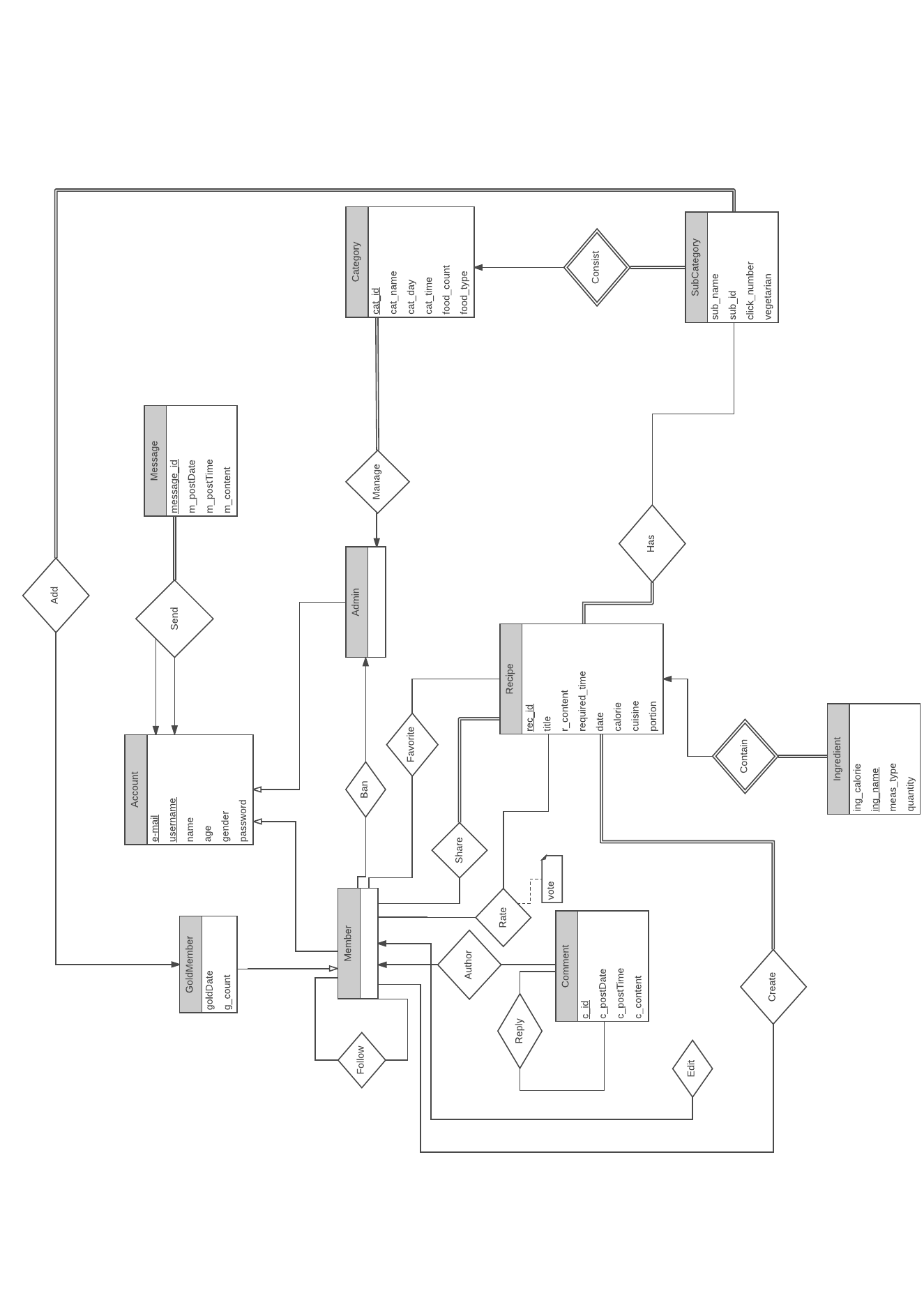
[6.5. Constraints 48](#_Toc498907867)

[7. Implementation Plan 48](#_Toc498907868)

# Revised E/R Model

In this section, we described our new E/R model. We added some features and fixed some confusions such as table names, relations according to the review. The changes are as follows:

* The ‘Context’ entity’s name is changed to ‘Recipe’.
* We changed the ‘Topics’ entity to Category entity and we added c\_id, c\_name, c\_day, c\_time, food\_count, food\_type as attributes.
* We altered the ‘Edit’ relation between Admin and Category entities as one to many.
* To represent the appetizers, desserts, main food, beverages and their subcategories we created ‘SubCategory’ entity which has a weak relationship ‘Consist’ with ‘Category’ entity. Therefore, we deleted the entities such as ‘Main Food’.
* In our new model, we added also the ‘Has’ relationship between ‘Recipe’ and ‘Subcategory’ entities which is a many-to-many relationship.
* We created a new entity called ‘Comment’ which has c\_id, c\_postDate, c\_postTime, c\_content as attributes. In order to see the replied comment, we created also a relation named ‘Reply’ which is unary.
* In addition, we added a relationship called ‘Author’ between comment and member which is a one-to-many relationship.
* We changed the ‘Ban’ relationship to many-to-one.
* We made a isA relationship by adding GoldMember entity under a Member entity. This GoldMember entity has goldDate and g\_count attributes. Also, only gold members can add subcategories, so we added an ‘Add’ relationship between ‘GoldMember’ and ‘SubCategory’ entities which is one-to-many. GoldMember inherits Member and SubCategory inherits Category.
* We created ‘Message’ entity as a new entity which has a relationship with ‘Account’ entity named ‘Send’. In order to see who send the message and who is sending to whom, we made the relationship ternary.
* We added ‘vote’ attribute to ‘Rate’ relationship which is between ‘Member’ and ‘Recipe’.
* We created a new entity called ‘Ingredient’ in order to store a foods calorie, quantity, and so forth with the attributes ing\_calorie, ing\_name, meas\_type and quantity. This ‘Ingredient’ entity has a weak relationship with ‘Recipe’ entity named ‘Contain’.
* We added a relation between comment and Recipe named ‘belongs’ since Recipes have comments.
* There is a new relation between Member and Recipe named ‘Edit’.



# Relation Schemas

## Account

**Relational Model:**

Account (e\_mail, username, name, age, gender, password)

**Functional Dependencies:**

e\_mail, username -> name age gender password

**Candidate Keys:**

{(e\_mail, username)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE account (

e-mail varchar(20) PRIMARY KEY,

username varchar(20) NOT NULL,

name varchar(40) NOT NULL,

age int NOT NULL,

gender varchar(8) NOT NULL,

password varchar(20) NOT NULL,

);

## Message

**Relational Model:**

Message(message\_id, m\_postDate, m\_postTime, m\_content, rec-e\_mail, rec-username, send-e\_mail, send-username)

**Functional Dependencies:**

message\_id -> m\_postDate,m\_postTime,m\_content,rec-email,rec-username,send-e\_mail,send-username

**Candidate Keys:**

{(message\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE Message (

message\_id int PRIMARY KEY AUTO\_INCREMENT,

m\_postDate date NOT NULL,

m\_postTime time NOT NULL,

m\_content varchar(240) NOT NULL,

rec-e\_mail varchar(20) NOT NULL,

rec-username varchar(20) NOT NULL,

send-e\_mail varchar(20) NOT NULL,

send-username varchar(20) NOT NULL,

FOREIGN KEY (rec-e\_mail) REFERENCES Account,

FOREIGN KEY (rec-username) REFERENCES Account

FOREIGN KEY (send-e\_mail) REFERENCES Account

FOREIGN KEY (send-username) REFERENCES Account)

Engine = InnoDB;

## Member

**Relational Model:**

Member(me-mail, musername, fol\_e-mail, fol\_username)

**Functional Dependencies:**

me-mail, musername -> fol\_e-mail,fol\_username

**Candidate Keys:**

{( me-mail, musername)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE Member (

me-mail varchar(20) PRIMARY KEY,

musername varchar(20) PRIMARY KEY,

fol\_e-mail varchar(20) NOT NULL,

fol\_username varchar(20) NOT NULL,

FOREIGN KEY (me-mail) REFERENCES Account,

FOREIGN KEY (musername) REFERENCES Account,

FOREIGN KEY (fol\_e-mail) REFERENCES Account,

FOREIGN KEY (fol\_username) REFERENCES Account)

ENGINE = InnoDB;

## GoldMember

**Relational Model:**

GoldMember(gme-mail, gmusername, goldDate, g\_count)

**Functional Dependencies:**

gme-mail, gmusername -> goldDate, g\_count

**Candidate Keys:**

{( gme-mail, gmusername)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE GoldMember (

gme-mail varchar(20) PRIMARY KEY,

gmusername varchar(20) PRIMARY KEY,

goldDate DATE,

g\_count int NOT NULL,

FOREIGN KEY (gme-mail) REFERENCES Account,

FOREIGN KEY (gmusername) REFERENCES Account)

ENGINE = InnoDB;

## Admin

**Relational Model:**

Admin(ade-mail, adusername)

**Functional Dependencies:**

NONE

**Candidate Keys:**

{( ade-mail, adusername)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE `Admin` (

ade-mail varchar(20) PRIMARY KEY,

adusername varchar(20) PRIMARY KEY,

FOREIGN KEY (ade-mail) REFERENCES Account,

FOREIGN KEY (adusername) REFERENCES Account)

ENGINE = InnoDB;

## Category

**Relational Model:**

Category(cat\_id, cat\_name, cat\_day, cat\_time, food\_count, food\_type, ade\_mail, adusername)

**Functional Dependencies:**

cat\_id , -> cat\_name, cat\_day,cat\_time, food\_count ,food\_type, ade\_mai, adusername

**Candidate Keys:**

{( cat\_id,)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE Category(

cat\_id int PRIMARY KEY AUTO\_INCREMENT,

cat\_name varchar(32) NOT NULL,

cat\_day date NOT NULL,

cat\_time time NOT NULL,

food\_count int NOT NULL,

food\_type varchar(32) NOT NULL,

ade-mail varchar(20),

adusername varchar(420),

FOREIGN KEY (ade-mail) REFERENCES Account,

FOREIGN KEY (adusername) REFERENCES Account)

ENGINE = InnoDB;

## SubCategory

**Relational Model:**

SubCategory(cat\_id, sub\_id, sub\_name, click\_number, vegetarian)

**Functional Dependencies:**

cat\_id, sub\_id -> click\_number, vegetarian

**Candidate Keys:**

{( cat\_id, sub\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE `SubCategory` (

cat\_id int PRIMARY KEY,

sub\_name varchar(32),

sub\_id int PRIMARY KEY AUTO INCREMENT,

click\_number int,

vegetarian varchar(20),

FOREIGN KEY (cat\_id) REFERENCES Category)

ENGINE = InnoDB;

## Ingredient

**Relational Model:**

Ingredient(ing\_name, ing\_calorie, meas\_type, quantity, rec\_id)

**Functional Dependencies:**

ing\_name ->ing\_calorie, meas\_type, quantity, rec\_id

**Candidate Keys:**

{( ing\_name)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE Ingredient (

ing\_name varchar(32) PRIMARY KEY,

ing\_calorie int NOT NULL,

meas\_type varchar(32),

quantity int,

rec\_id int NOT NULL,

FOREIGN KEY (rec\_id) REFERENCES Recipe)

ENGINE = InnoDB;

## Comment

**Relational Model:**

Comment(c\_id,c\_postDate, c\_postTime, c\_content, replyc\_id, me-mail,musername,rec\_id)

**Functional Dependencies:**

c\_id -> c\_postDate, c\_postTime, c\_content, replyc\_id, me-mail, musername, rec\_id

**Candidate Keys:**

{( c\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE Comment (

c\_id int PRIMARY KEY AUTO\_INCREMENT,

c\_postDate date NOT NULL,

c\_postTime time NOT NULL,

c\_content varchar(80) NOT NULL,

replyc\_id int,

me-mail varchar(20) NOT NULL,

rec\_id int NOT NULL,

musername varchar(20) NOT NULL,

FOREIGN KEY(me-mail) REFERENCES member,

FOREIGN KEY(rec\_id) REFERENCES Recipe,

FOREIGN KEY(musername) REFERENCES member)

ENGINE = InnoDB;

## Recipe

**Relational Model:**

Recipe(rec\_id, title, r\_content, required\_time, date, calorie, cuisine, portion, me-mail,musername, sub-id)

**Functional Dependencies:**

rec\_id -> title,r\_content,required\_time,date,calorie,cuisine,portion, me-mail, musername

rec\_id -> sub\_id

**Candidate Keys:**

{(rec\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE Recipe (

rec\_id int PRIMARY KEY AUTO\_INCREMENT,

title varchar(32) NOT NULL,

r\_content varchar(800) NOT NULL,

required\_time int NOT NULL,

date date,

calorie int,

cuisine varchar(16),

portion int,

me-mail varchar(20) NOT NULL,

musername varchar(20) NOT NULL,

sub\_id int NOT NULL,

FOREIGN KEY(me-mail) REFERENCES member,

FOREIGN KEY(musername) REFERENCES member,

FOREIGN KEY(sub\_id) REFERENCES SubCategory)

ENGINE = InnoDB;

## Share

**Relational Model:**

Share(me-mail, musername, rec-id)

**Functional Dependencies:**

NONE

**Candidate Keys:**

{( e-mail, username, rec-id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE Share (

me-mail varchar(20) PRIMARY KEY NOT NULL,

musername varchar(20) PRIMARY KEY NOT NULL,

rec\_id int PRIMARY KEY NOT NULL,

FOREIGN KEY(me-mail) REFERENCES Member,

FOREIGN KEY(musername) REFERENCES Member,

FOREIGN KEY(rec\_id) REFERENCES Recipe)

ENGINE = InnoDB;

## Create

**Relational Model:**

Create(me-mail, musername, rec-id)

**Functional Dependencies:**

NONE

**Candidate Keys:**

{( e-mail, username, rec-id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE Create (

me-mail varchar(20) PRIMARY KEY NOT NULL,

musername varchar(20) PRIMARY KEY NOT NULL,

rec\_id int PRIMARY KEY NOT NULL,

FOREIGN KEY(me-mail) REFERENCES Member,

FOREIGN KEY(musername) REFERENCES Member,

FOREIGN KEY(rec\_id) REFERENCES Recipe)

ENGINE = InnoDB;

## Has

**Relational Model:**

Has(rec-id, sub\_id,cat\_id,)

**Functional Dependencies:**

NONE

**Candidate Keys:**

{( rec-id, sub\_id, cat\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE Has (

rec\_id int PRIMARY KEY NOT NULL,

cat\_id int PRIMARY KEY NOT NULL,

sub\_id int PRIMARY KEY NOT NULL,

FOREIGN KEY(rec\_id) REFERENCES Recipe,

FOREIGN KEY(cat\_id) REFERENCES Category,

FOREIGN KEY(sub\_id) REFERENCES SubCategory)

ENGINE = InnoDB;

## Rate

**Relational Model:**

Rate(rec-id, sub\_id, vote)

**Functional Dependencies:**

rec-id, sub-id -> vote

**Candidate Keys:**

{( rec-id, sub\_id )}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE Rate (

rec\_id int PRIMARY KEY NOT NULL,

sub-id int PRIMARY KEY NOT NULL,

vote int NOT NULL,

FOREIGN KEY(rec\_id) REFERENCES Recipe,

FOREIGN KEY(sub\_id) REFERENCES SubCategory)

ENGINE = InnoDB;

## Follow

**Relational Model:**

Follow(me-mail, musername, fole-mail, folusername)

**Functional Dependencies:**

NONE

**Candidate Keys:**

{( me-mail, musername, fole-mail, folusername)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE Follow (

me-mail varchar(20) PRIMARY KEY NOT NULL,

musername varchar(20) PRIMARY KEY NOT NULL,

fole-mail varchar(20) PRIMARY KEY NOT NULL,

folusername varchar(20) PRIMARY KEY NOT NULL,

FOREIGN KEY(me-mail) REFERENCES Member,

FOREIGN KEY(musername) REFERENCES Member,

FOREIGN KEY(fole-mail) REFERENCES Member,

FOREIGN KEY(folusername) REFERENCES SubCategory)

ENGINE = InnoDB;

## Favorite

**Relational Model:**

Favorite(me-mail, musername, rec-id)

**Functional Dependencies:**

NONE

**Candidate Keys:**

{( e-mail, username, rec-id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE Favorite (

me-mail varchar(20) PRIMARY KEY NOT NULL,

musername varchar(20) PRIMARY KEY NOT NULL,

rec\_id int PRIMARY KEY NOT NULL,

FOREIGN KEY(me-mail) REFERENCES Member,

FOREIGN KEY(musername) REFERENCES Member,

FOREIGN KEY(rec\_id) REFERENCES Recipe)

ENGINE = InnoDB;

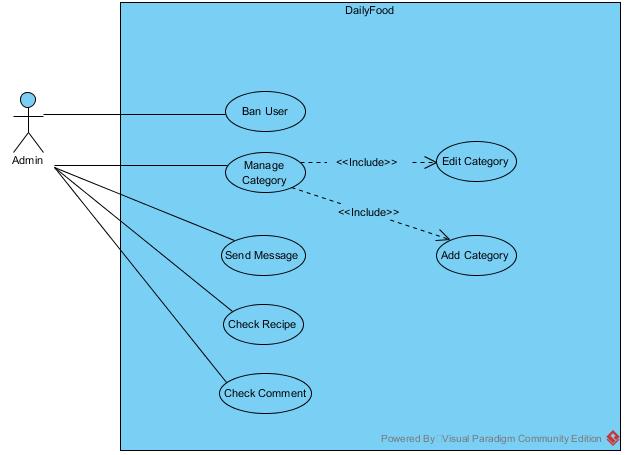
# Functional Dependencies and Normalization of Tables

The Relation Schemas part of our design report contains all the functional dependencies and normal forms. Since the relations are all in Boyce-Codd Normal Form (BCNF), there is no need for any decomposition nor normalization.

# Functional Components

## Use Cases / Scenarios

## Admin



* **Ban User:** Admin can ban other users in case of an any inappropriate content or any security issues.
* **Manage Category:** Admins can manager categories; admin can edit a category and also add a category. This includes adding and editing categories. Only an admin is capable of doing so.
* **Send Message:** Admins can send messages to other users.
* **Check Recipe:** Admin can check recipes, it’s description, names of ingredients etc., in case of any inappropriate content.
* **Check Comment:** Admin can check comments if there is any bad language or other bad usage of comments.

## Member and Gold Member

* **Create Recipe:** Every member can create their own recipes. They need to enter ingredients, categories, time of day it should be eaten, time it takes to make and number of portions. Members can pick ingredients from the saved list or they can add their own ingredients. Adding a new ingredient requires it to have a name and calories for each serving.
* **Share Recipe:** Members can share their recipes on social media.
* **Search Recipe:** Members can search for recipes using the search page. Recipes can be search based on their names, categories, and ingredients.
* **Rate Recipe:** Members can rate other members recipes by a “5 Star Rating System” which allows members to click on stars at the top of each recipe to rate it.
* **Send Message:** Members can send messages to each other and reply to received messages.
* **Make Comment:** Members can comment on other recipes and reply to already made comments.
* **Follow Member:** Members can follow other members.
* **Modify Personal Information:** Members can modify their personal informations from their profiles.
* **Favorite Recipe:** Members can favorite a recipe and add it to “Favorites” section on their profiles.
* **Add Subcategory:** Gold members are allowed to add subcategories to the Daily Food database.

## Algorithms

## Rate Related Sorting Algorithms

The system provides 3 different sorting algorithms. First one, the system sorts the recipes by their rates. This method search through all members of the system and finds highest rated recipes which belongs the previous day. Second one, the system more likely to provide member’s “following friends’” recipe on the daily feed home page. The last one, system provides “try this recipe” which has a higher rate, which the user have never clicked on before; it will be a new recipe for the user.

## Search Algorithms

User can search for recipes based on their categories, subcategories, ingredients and cuisine. The system will search through methods and comes of with results based on selected properties. User can only search recipes by selecting different options; type-to-search is not usable in this system. The system can sort the recipes according to following state, if the user search for a recipe, and that recipe is in the recipe book of the user’s “following friends”, those recipes have priority in search method. Rest will be listed by their “Rate Related Sorting Algorithm” principles.

## Calorie Count Algorithm

Each recipe has a total calorie according to its ingredients. Calories of each individual ingredient will be summed up to a total calorie of the whole recipe. Then those calories will be divided into the number of portions to show the calories per portion.

## Logical Requirements

Preventing logical errors are key for the system, to have true information. The system should be designed in a way such that it decides if recipes components are not something sensitive or something unrelated. Logical errors are most likely to occur during a member adds a recipe because that member can enter descriptions and ingredients freely where the member can enter something unrelated.

Ingredients can be based on grams or quantity. System should be designed in a way to give user those 2 options to enter ingredients with.

## Data Structures

The relation schemas we created used Numeric Types, String Types, Date and Time Types.

Numeric Types are used to store numeric data, like id, age (of account), rate, cat\_id, food\_count, sub\_id, click\_number, ing\_calorie, quantity, rec\_id, c\_id, calorie, portion. For the numeric types we use **INT** and **DOUBLE.**

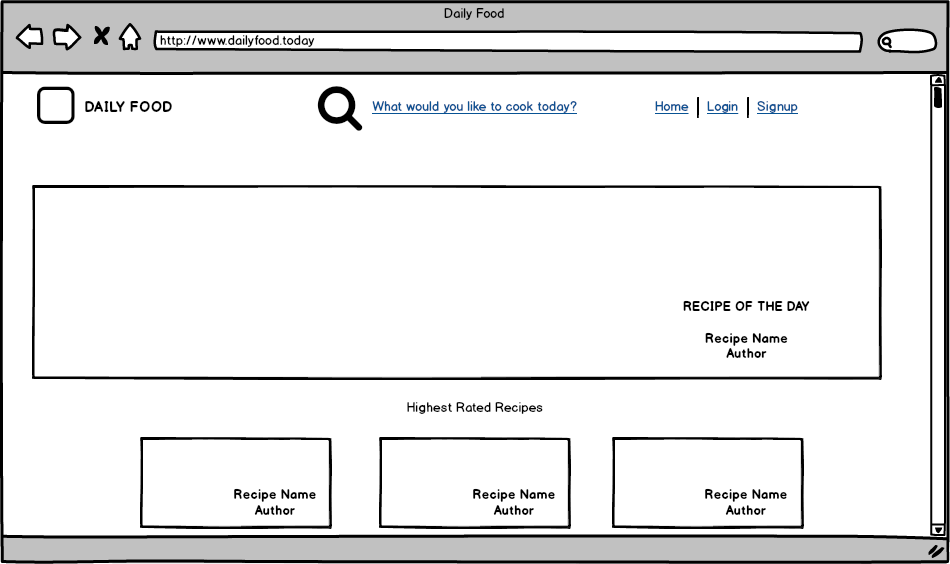
String types are required to store any attributes composed of characters such as email, username, password, cat\_name, food\_type, cuisine. For the string stypes we use **VARCHAR**.

Date and Time types are used to keep time values such as cat\_day, cat\_time, date. For the Date and Time types we use **DATE** and **TIME**.

# User Interface Design and Corresponding SQL Statements

## Homepage Screen

* + 1. Non-registered



**Inputs: -**

Process: The homepage for DailyFood is displayed above. Non-registered users of the system will be welcomed with the Recipes with the highest ratings within a day will be demonstrated on the feed screen. On the hero image, highest rated food will be placed. Non –registered users can also click on the recipes and a general search bare to search for any recipe in the system. In order to comment and add recipes, users must sign-up or log-in to the system.

**SQL Statements**:

**Displaying the top 4 Recipes**

CREATE VIEW recipes\_top4 (rec\_id, title , rate, username)

AS SELECT rec\_id, title,rate, username

FROM

(SELECT rec\_id, sum(votes) AS rates

FROM (SELECT rec\_id FROM Recipe WHERE DATE(date)=CURDATE())

NATURAL JOIN Rate

GROUP BY rec\_id

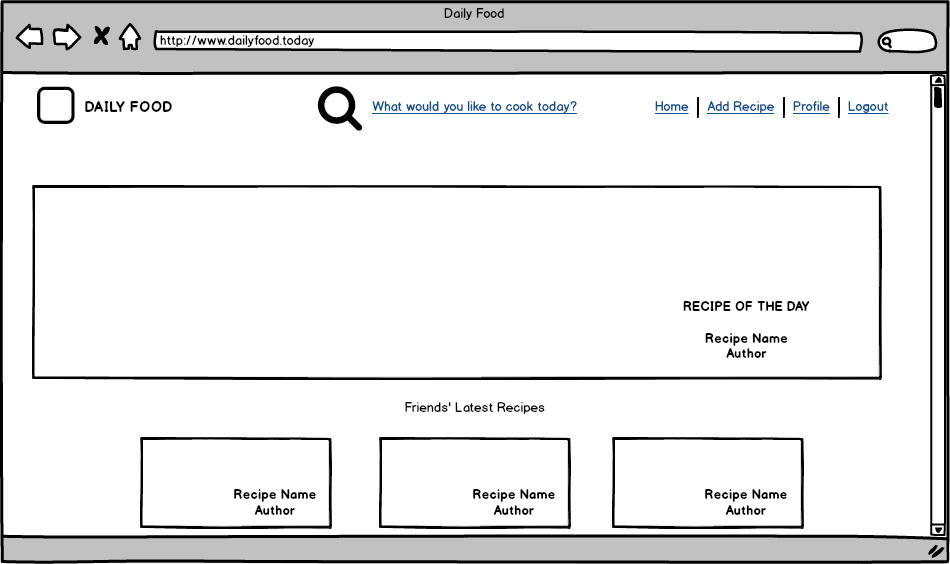
ORDER BY rates DESC

LIMIT 4)

)

NATURAL JOIN Recipes NATURAL JOIN Create;

## Registered – User



**Inputs:** @CurrentUsername, @CurrentUserEmail

**Process**: When user log into or sign up to system, welcome page similar to non-registered users will welcome the user. Instead of login and signup buttons, there will be edit profile, my recipes and add recipe buttons. Also instead of highest ratings, user will able to see her/his friends recipes lastly shared

**SQL Statements:**

CREATE VIEW friendsRecipe( rec\_id, title, username)

AS SELECT rec\_id, title,username

FROM

(SELECT me-mail, musername

FROM follow

WHERE fole-mail= @CurrentUserMail and folusername = @CurrentUserName

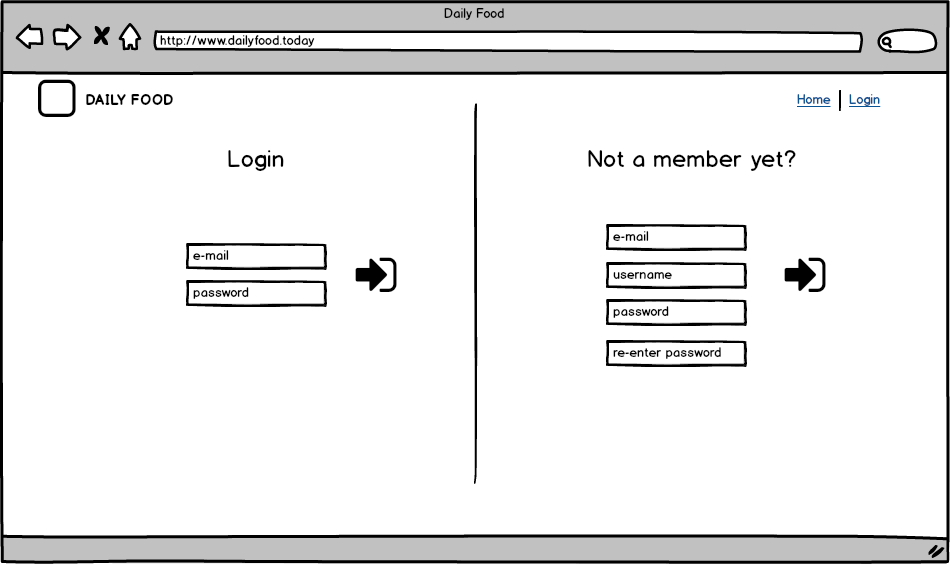
)

NATURAL JOIN share NATURAL JOIN recipe

ORDER BY date DESC;

Note: In the follow relation, “me-mail” and “musername” represent users who have been followed. “Fole-mail” and “folusername” represent users who followed the user.

## Login and Signup

****

**Inputs:** @e-mail, @password, @username, @password1, @password2.

**Process:** If user has an existing account, by entering e-mail/username and password he/she can enter the system. If do not, by providing necessary information that are email, username and a valid password

**SQL Statements:**

**Get User:**

Select \*

FROM member

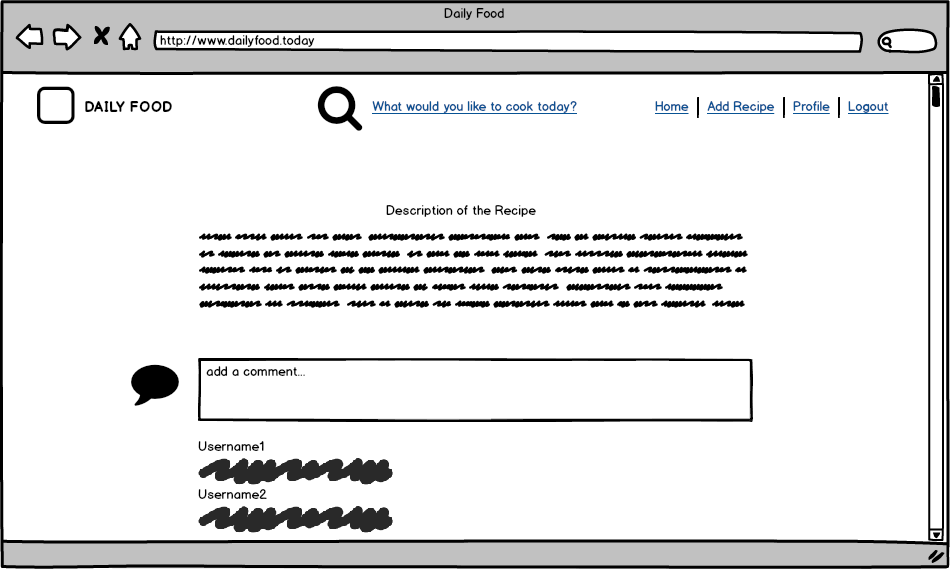
WHERE((username = @username) OR (e-mail = @e-mail) ) AND password = @password;

**Insert User:**

INSERT INTO Account (e-mail,username,password)

VALUES (@e-mail, @username, @password1);

## Recipe Add Comment

****

**Inputs**: @comment, @currentUsername,@currentUseremail, @CurrentRecipeID, @currentC\_id

**Process:** Registered user will enter their comment to current recipe screen or others comments.

**SQL Statements:**

**Insert Comment to recipe**

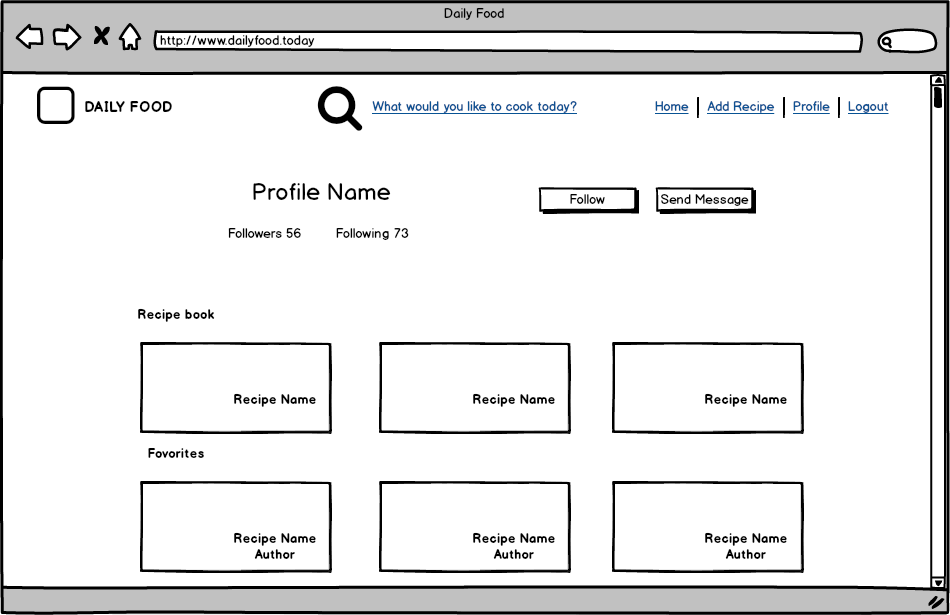
**Note:** C\_id is automatically incremented.

INSERT INTO Comment Values( CURDATE(),HOUR(),@comment,null, , @currentUsername,@currentUseremail, @CurrentRecipeID);

**Insert Comment to Existing Comment**

INSERT INTO Comment Values( CURDATE(),HOUR(),@comment,@currentC\_id, , @currentUsername,@currentUseremail,@CurrentRecipeID);

## Profiles of Other Members

****

**Process:** When registered users enter the other users’ profiles, they can see the visited user recipes,his/her favourites. Also, user can follow users and send messages.

**Inputs:** @visitedUserEmail, @visitedUsername, @currentUserEmail, @currentUsername.

**SQL Statements**

**Recipe Book:**

SELECT \* FROM recipe WHERE me-mail = @visitedUserEmail and musername = visitedUsername;

**Favourites:**

Create VIEW user\_Fav

AS

SELECT rec-id,title

FROM favourite NATURAL JOIN Member WHERE me-mail = @visitedUserEmail AND musername= @visitedUsername;

**Follower Count:**

SELECT COUNT(folusername) FROM Follow WHERE me-mail = @ visitedUserEmail AND musername = @visitedUserEmail;

**Following Count**

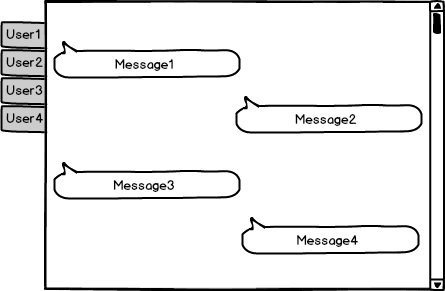
SELECT COUNT(folusername)

FROM Follow WHERE fole-mail = @ visitedUserEmail AND folusername = @visitedUserEmail;

**Follow Button**

INSERT INTO Follow Values (@visitedUserEmail, @visitedUsername, @currentUserEmail, @currentUsername);

## Send Message

****

**Inputs:** @Message

**Process:** Users can send message to each other. Messages can be seen via this screen.

INSERT INTO Message Values( CURDATE(),HOUR(),@visitedUserEmail, @visitedUsername, @currentUserEmail, @currentUsername);

**Taking of Specific Chat Between Two Users**

CREATE VIEW CHAT(message\_id, m\_postDate, m\_postTime, m\_content, rec-e\_mail, rec-

username, send-e\_mail, send-username))

AS

(SELECT \* FROM Message WHERE rec-e\_mail = @visitedUserEmail AND

rec-username=@visitedUsername AND send-e\_mail = @currentUserEmail AND

send-username= @currentUsername)

UNION

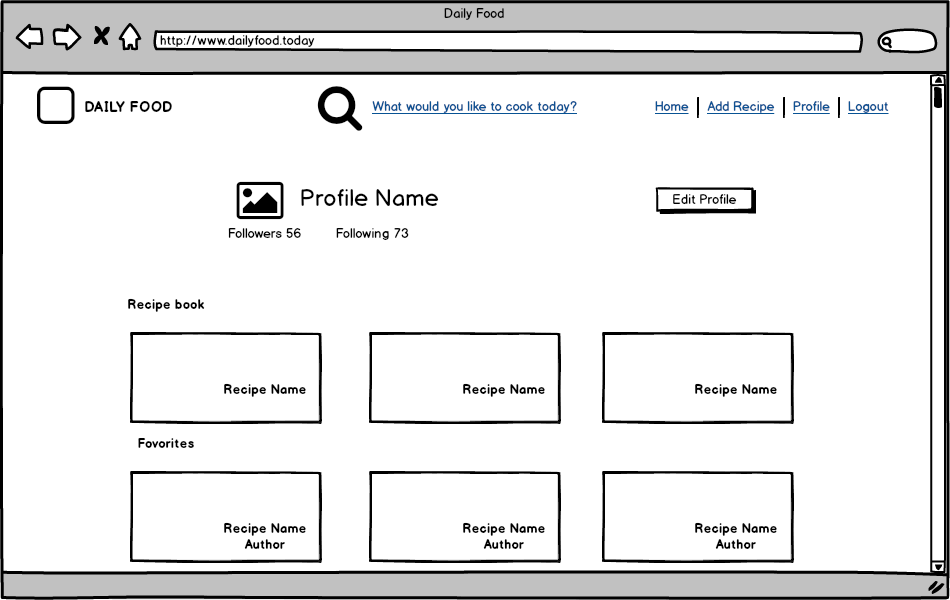
(SELECT \* FROM Message WHERE rec-e\_mail = @ currentUserEmail AND

rec-username=@ currentUsername AND send-e\_mail = @ visitedUserEmail AND

send-username= @ visitedUsername)

SELECT \* FROM [CHAT];

## User own Profile

****

**Inputs:** @currentUsername, @currentUserEmail

**Process:** User can display all the recipes he/she wrote or his/her favourite recipes.Also, by clicking edit profile button, he or she edit profile information.

**Recipe Book:**

SELECT \* FROM recipe

WHERE musername = @ currentUsername AND me-mail= @currentUserEmail;

**Favourites:**

CREATE View user\_Fav AS

SELECT rec-id, title

FROM favourite NATURAL JOIN Member WHERE me-mail = @currentUserEmail and musername= @currentUsername,

**Follower Count:**

Select COUNT(folusername)

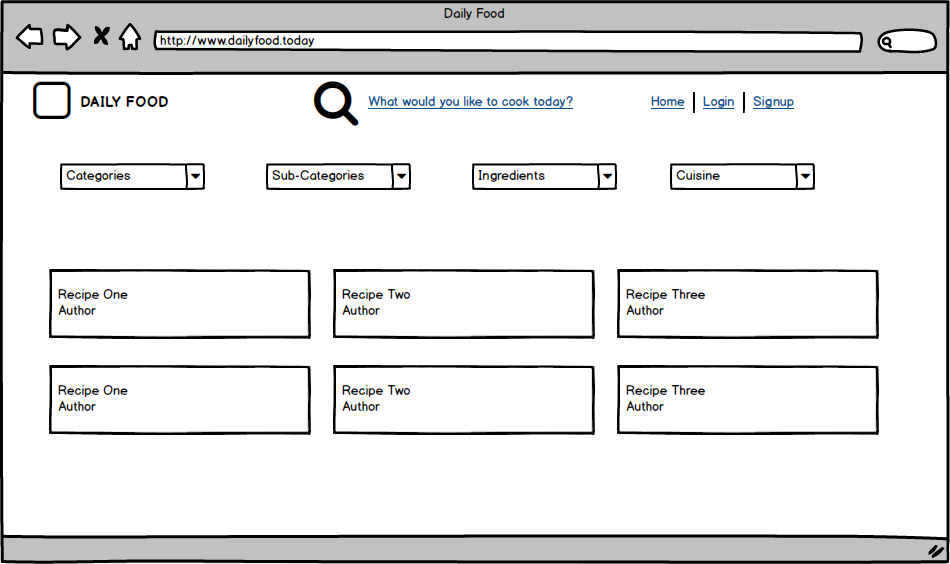
FROM Follow where me-mail = @ visitedUserEmail AND musername = @visitedUserEmail;

**Following Count**

SELECT COUNT(folusername)

FROM Follow WHERE fole-mail = @ currentUsername AND folusername = @ currentUserEmail;

## Search Screen

****

**Input:** @filterCategories, @filterSubCategories, @filterIngredients, @filterCuisine

**Process:** The search page for the Daily Food is displayed above. Registered and non-registered users of the system can use search function. When a user opens the search screen, there will be filters for categories, sub categories, ingredients, and cuisine. User can leave the filters blank. When user fills (select from list) filters. System brings top 6 rated recipes.

**SQL Statements:**

CREATE VIEW resultPage(rec\_id, title, username)

SELECT rec\_id, title, musername

FROM

(SELECT rec\_id, sum(vote) as rates)

FROM (SELECT rec\_id FROM Recipe)

NATURAL JOIN Rate

GROUP BY rec\_id

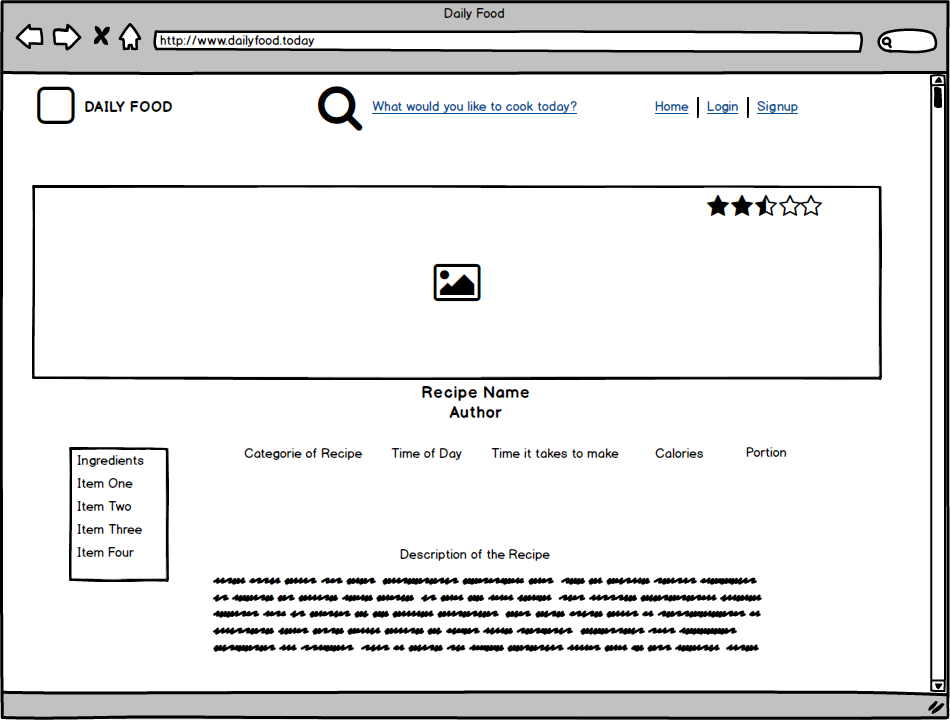
ORDER BY rates DESC

LIMIT 6)

NATURAL JOIN Create NATURAL JOIN Has NATURAL JOIN SubCategory NATURAL JOIN Ingredient

WHERE @filterCategories = cat\_id AND @filterSubCategories = sub\_id AND @filterIngredients = ing\_name AND @filterCuisine = cuisine;

## Recipe View Screen

****

**Input:** @selectedRecipe

**Process:** The recipe view page for the Daily Food is displayed above. Registered and non-registered users of the system can see the recipes. When a user opens recipe view screen, the system provides all the information about that recipe.

**SQL Statements:**

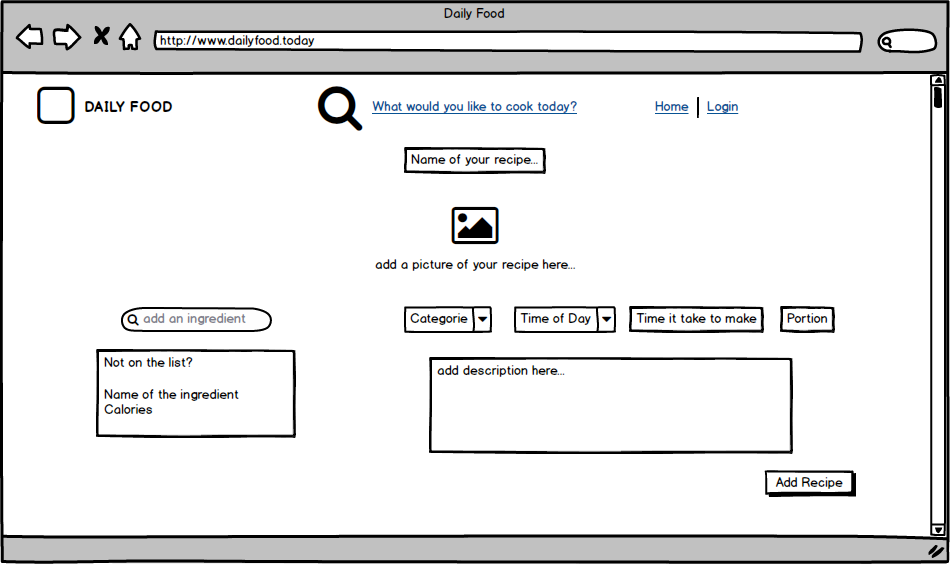
SELECT (\*)

FROM Recipe

NATURAL JOIN Ingredient NATURAL JOIN Has NATURAL JOIN Category NATURAL JOIN Comment

WHERE @selectedRecipe = rec\_id;

## Add Recipe Page

****

**Input:** @RecipeName, @SelectedCategory, @SelectedSubCategory ,@TimeOfDay, @TimeToMake, @Portion, @Ingredients, @Description, @Calorie, @Cuisine, @IngredientCalorie, @IngredientName, @Meas\_type, @Quantity, @Vegetarian

**Process:** The add recipe page for the Daily Food is displayed above. Registered users of the system can add new recipes. When a user opens add recipe page, he should fill the required fields which are name of the recipe, category of the recipe, sub-category of the recipe, time of the day, required time, portion number, ingredients, description, calorie, cuisine, ingredient’s calorie, and vegetarian info.

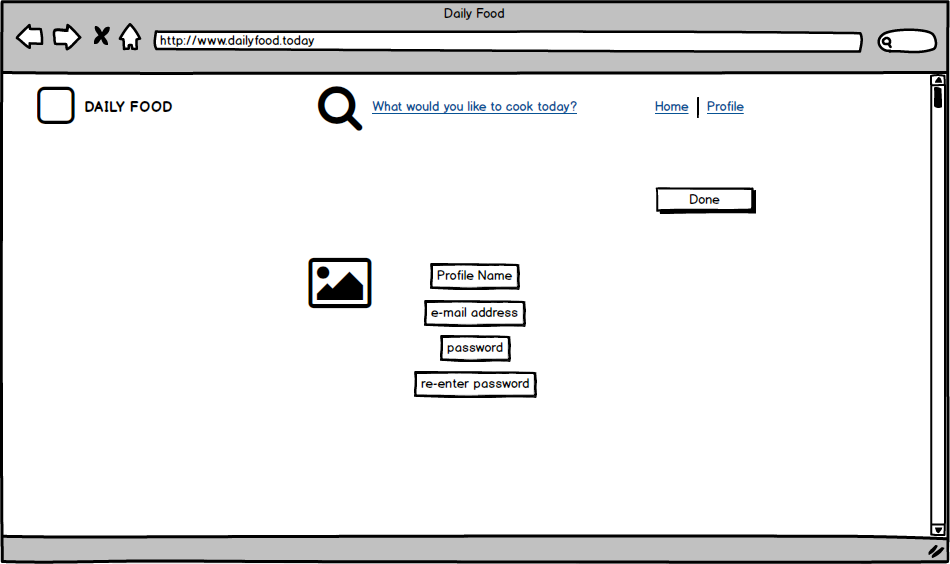
**SQL Statements:**

INSERT INTO Recipe VALUES (@RecipeName, @Description, @TimeToMake, @TimeOfDay, @Calorie, @Cuisine, @Portion);

INSERT INTO Ingredient VALUES(@IngredientName, @IngredientCalorie, @Meas\_type, @Quantity);

INSERT INTO SubCategory VALUES(@SelectedSubCategory, @Vegetarian);

## Profile Edit

****

**Input:** @ProfileName, @Email, @Password

**Process:** The profile edit page for the Daily Food is displayed above. Registered users of the system can change account information in the profile edit page.

**SQL Statements:**

UPDATE Account

SET e-mail = @Email , username = @ProfileName, password = @Password;

# Advanced Database Components

## Views

## Displaying the top 4 Recipes for Unregistered User

CREATE VIEW recipes\_top4 (rec\_id, title , rate, username)

AS SELECT rec\_id, title,rate, username

FROM

(SELECT rec\_id, sum(votes) AS rates

FROM (SELECT rec\_id FROM Recipe WHERE DATE(date)=CURDATE())

NATURAL JOIN Rate

GROUP BY rec\_id

ORDER BY rates DESC

LIMIT 4)

)

NATURAL JOIN Recipes NATURAL JOIN Create;

## Displaying the top 4 Recipes for Registered User

CREATE VIEW friendsRecipe( rec\_id, title, username)

AS SELECT rec\_id, title,username

FROM

(SELECT me-mail, musername

FROM follow

WHERE fole-mail= @CurrentUserMail and folusername = @CurrentUserName

)

NATURAL JOIN share NATURAL JOIN recipe

ORDER BY date DESC;

## Showing Other Members Favorites

Create VIEW user\_Fav

AS

SELECT rec-id,title

FROM favourite NATURAL JOIN Member WHERE me-mail = @visitedUserEmail AND musername= @visitedUsername;

## Chat Between Two Users

CREATE VIEW CHAT(message\_id, m\_postDate, m\_postTime, m\_content, rec-e\_mail, rec-

username, send-e\_mail, send-username))

AS

(SELECT \* FROM Message WHERE rec-e\_mail = @visitedUserEmail AND

rec-username=@visitedUsername AND send-e\_mail = @currentUserEmail AND

send-username= @currentUsername)

UNION

(SELECT \* FROM Message WHERE rec-e\_mail = @ currentUserEmail AND

rec-username=@ currentUsername AND send-e\_mail = @ visitedUserEmail AND

send-username= @ visitedUsername)

SELECT \* FROM [CHAT];

## Favorites of User’s own Profile

CREATE View user\_Fav AS

SELECT rec-id, title

FROM favourite NATURAL JOIN Member WHERE me-mail = @currentUserEmail and musername= @currentUsername,

## Result of Search Screen

CREATE VIEW resultPage(rec\_id, title, username)

SELECT rec\_id, title, musername

FROM

(SELECT rec\_id, sum(vote) as rates)

FROM (SELECT rec\_id FROM Recipe)

NATURAL JOIN Rate

GROUP BY rec\_id

ORDER BY rates DESC

LIMIT 6)

NATURAL JOIN Create NATURAL JOIN Has NATURAL JOIN SubCategory NATURAL JOIN Ingredient

WHERE @filterCategories = cat\_id AND @filterSubCategories = sub\_id AND @filterIngredients = ing\_name AND @filterCuisine = cuisine;

## Stored Procedures

We plan to use stored procedures when creating a recipe. Initially there will be no recipe and that means there will be no categories or subcategories associated with the recipe. For each recipe, there will be rec\_id to seperate the recipes from each other. However, initially, there is no recipe added. Typing the value by hand can raise some consistency issues. Therefore, the stored procedure is incrementing the rec\_id in the related recipe tuple every time a recipe is added.

When a recipe is added, it is looked whether it has a corresponding subcategory. While adding a recipe, a subcategory will be typed. If that subcategory is absence, adding the corresponding tuple into the subcategory table will be the stored procedure.

## Reports

## Monthly Added Recipe Report By Members

Members and admins can view a monthly report of the recipe along with the recipe title and the current values. Members may add different recipes by looking at this report. If there are no additions of a recipe under a subcategory, a goldmember may delete the subcategory by looking at this report. Also, a member can edit a category like creating a new one if there are so many recipes in one category.

CREATE VIEW monthly\_added\_recipe\_report

AS SELECT r.rec\_id, c.me-mail, c.musername, count(r.rec\_id) as Recipes

FROM create c NATURAL JOIN recipe r

WHERE DATEDIFF(CURDATE(), date) <= 30

GROUP BY c.me-mail, c.musername

## Total Number of Subcategories in Each Category

If there is no subcategory, under a category, admin may delete the category according to this report.

WITH categories\_and\_subcategories

AS (SELECT c.cat\_id, c.cat\_name, s.sub\_id

FROM Category c NATURAL JOIN Subcategory s)

SELECT cat\_id, cat\_name,

(CASE

WHEN EXISTS (SELECT \* FROM Category) THEN ‘YES’

ELSE ‘NO’

END) AS categoryType, count(\*)

FROM categories\_and\_subcategories

GROUP BY cat\_id

## Triggers

* When a member creates a recipe, sub category and the ingredients will be updated. The votes will be updated. The click number of that recipe in subcategory will be updated. The rec\_id will be updated automatically.
* Every member will have a counter that cannot be seen by the members. This counter will be updated every time when a member shares a recipe.
* When a subcategory is added, the food type in category will be updated automatically.
* When a recipe is deleted, the subcategory of that recipe and category of subcategory will be updated.

## Constraints

* The system cannot be used without logging in.
* Members cannot edit a category. The system will not allow it.
* A member can only ban one admin. The system will not allow such attempts.
* Only goldmember can add a subcategory.
* Messages can be seen only by accounts who messaged to. For example, if an admin messaged to a member, it can only be seen by that admin and member.
* An admin cannot edit a category if there is no category at all.
* Someone cannot edit an ingredient of a recipe. The ingredients are updated for once after the addition of a Recipe.
* A category cannot be removed if there is a subcategory under that category. In order to remove subcategories from the system, there should be no categories at all that is linked to such categories.

# Implementation Plan

For our system functionalities and user interface in our hypertext dictionary system, we are planning to use HTML, CSS, JavaScript, Asp.net. In order to manage the flow of data in our project, we are planning to use MySQL Server.