Part 1

brainstorming:
-users email and password
-grocery lists for different users
-recipes for your ingredients
-if recipes are private or public
-who follows who

Table Ideas:
users -- will hold information about the users email and password
ingredients_list -- will hold information about users list of ingredients
recipes -- will hold recipes that are based off users list of ingredients
followers -- will hold information about which users follow which other user
Recommendations -- will information on what recipe recommendations users should receive
User_profiles -- will hold information about users and their favorite recipes and ingredients, if public

Relationships:
one to one:
users/ingredients_list, because one user only has one ingredient list and an ingredient list only belongs to one user.
one to many:
-followers/users, because one user can have many followers but one follower is always one user
many to many:
ingredients_list/recipes, many ingredients can belong to one recipe and many recipes can involve one ingredient.

Part 2

Columns:
-users

- Id, serial primary key
- Email, VARCHAR(255): chose to store the data to allow users to log in and chose Varchar so that they can change the input
- Password, VARCHAR(255): same reasons as email

-ingredients_list

- Id, serial primary key
- User_id, referencing id for users table: this way we know who the list belongs to
- Ingredients, text: chose to store this so that we can see what ingredients are on the list

-recipes

- Id, serial primary key
- Ingredients_id, references id from ingredients table: so that we know what recipes the user can make based on ingredients
- User_id, references if from users table: so we know what user we are accessing
- Name, varchar(255): so we know the name of the recipe
- Public, boolean: so we know if the recipe is public or private
- Instructions, text: so we can tell the user how to use the recipe
- Description, text: so the user can get an idea of what the recipe makes

-followers

- Id, serial primary key
- User_id1, references id from users: so we know the first user that is following/ being followed by user 2
- User_id2, references if from users: so we know the second user being followed/ following user 1

-recommendations

- Id, serial primary key
- User_id, references id for users table: chose this so we know which user we are recommending recipes to
- Recipe_id, references id from recipes table: chose this so we can pull the right recipe to recommend
- ingredients_id , references id from ingredients table: so that we can tell what to recommend based on ingredients

-user_profiles

- Id, serial primary key
- User_id, references id from users table: this allows us to know which user profile to view
- recipe_id , references if from recipe table: to know what recipes are being displayed on the users page

Part 3

```
-- CREATE TABLE users (
-- users_id SERIAL PRIMARY KEY,
-- email VARCHAR(100),
-- password VARCHAR(100)
-- );

-- CREATE TABLE ingredients (
-- ingredients_id SERIAL PRIMARY KEY,
-- user_id INTEGER NOT NULL REFERENCES users(users_id),
-- ingredients_list TEXT
```

```
-- );

-- CREATE TABLE recipes (
-- recipe_id SERIAL PRIMARY KEY,
-- ingredients_id INTEGER NOT NULL REFERENCES ingredients(ingredients_id),
-- user_id INTEGER NOT NULL REFERENCES users(user_id),
-- recipe_name VARCHAR(100),
-- recipe_instructions TEXT,
-- recipe_description TEXT,
-- public BOOLEAN
-- );

-- CREATE TABLE followers(
-- followers_id SERIAL PRIMARY KEY,
-- user_id1 INTEGER NOT NULL REFERENCES users(user_id),
-- user_id2 INTEGER NOT NULL REFERENCES users(user_id)
-- );

-- CREATE TABLE recommendations(
-- recommendations_id SERIAL PRIMARY KEY,
-- user_id INTEGER NOT NULL REFERENCES users(user_id),
-- recipe_id INTEGER NOT NULL REFERENCES recipes(recipe_id),
-- ingredients_id INTEGER NOT NULL REFERENCES ingredients(ingredients_id)
-- );

-- CREATE TABLE user_profiles(
-- user_profiles_id SERIAL PRIMARY KEY,
-- user_id INTEGER NOT NULL REFERENCES users(user_id),
-- recipe_id INTEGER NOT NULL REFERENCES recipes(recipe_id)
-- );
```