Data Modeling

Intro

For today, you'll be planning out a data model based on the scenario given.

Part 1: Conceptual Planning - Word/Google/Pages Doc

Brainstorming:

1. **Users:**
   - Email
   - Password
   - Username
   - Profile picture
   - Saved recipes
   - Grocery lists
   - Created occasions
2. **Recipes:**
   - Title
   - Description
   - Ingredients
   - Instructions
   - Public/Private status
   - Creator (user)
   - Occasion (if assigned)
3. **Ingredients:**
   - Name
   - Quantity
   - Measurement unit
4. **Occasions:**
   - Title
   - Date
   - Description
   - Associated recipes
5. **Grocery Lists:**
   - Title
   - Items
   - Checked off status
   - Associated user
6. **User Interaction Data:**
   - Likes on recipes
   - Comments on recipes
   - Views on recipes

7. **Security:**
    - Authentication tokens
    - Permissions (admin, regular user)

User Flows:

1. **Sign-Up Flow:**
    - Collect email, password, username, and profile picture.
2. **Recipe Creation Flow:**
    - Input recipe title, description, ingredients, and instructions.
    - Choose public or private visibility.
3. **Recipe Viewing Flow:**
    - Display recipe details including title, description, ingredients, and instructions.
    - Show creator's username and profile picture.
4. **Grocery List Management Flow:**
    - Add ingredients from recipes to a grocery list.
    - Check off purchased items.
5. **Occasion Creation Flow:**
    - Create new occasions with title, date, and description.
    - Assign recipes to occasions.
6. **User Interaction Flow:**
    - Like, comment, and view recipes.
    - View notifications for recipe interactions.

Table Ideas:

1. **Users:**
    - Description: This table will store information about app users.
    - Fields:
        - user_id (Primary Key)
        - email
        - password
        - username
        - profile_picture_url
2. **Recipes:**
    - Description: This table will store information about recipes created by users.
    - Fields:
        - recipe_id (Primary Key)
        - title
        - description
        - ingredients (JSON/Text)
        - instructions (Text)
        - visibility (Public/Private)
        - user_id (Foreign Key referencing Users table)
        - occasion_id (Foreign Key referencing Occasions table, optional)

3. **Ingredients:**
   - Description: This table will store information about ingredients used in recipes.
   - Fields:
     - ingredient_id (Primary Key)
     - name
     - quantity
     - unit
     - recipe_id(Foreign Key referencing recipes table, optional)
4. **Occasions:**
   - Description: This table will store information about occasions created by users.
   - Fields:
     - occasion_id (Primary Key)
     - title
     - date
     - description
     - user_id (Foreign Key referencing Users table)
5. **GroceryLists:**
   - Description: This table will store information about user's grocery lists.
   - Fields:
     - list_id (Primary Key)
     - title
     - items (JSON/Text)
     - checked_off_status (JSON/Text)
     - user_id (Foreign Key referencing Users table)
6. **UserInteractions:**
   - Description: This table will store information about user interactions with recipes.
   - Fields:
     - interaction_id (Primary Key)
     - user_id (Foreign Key referencing Users table)
     - recipe_id (Foreign Key referencing Recipes table)
     - action (Like/Comment/View)
     - timestamp
7. **AuthenticationTokens:**
   - Description: This table will store authentication tokens for users.
   - Fields:
     - token_id (Primary Key)
     - user_id (Foreign Key referencing Users table)
     - token
     - expiry_date

8 **Favorites:**

- Description: This table will store information about users' favorite recipes.
- Fields:
    - favorite_id (Primary Key)
    - user_id (Foreign Key referencing Users table)
    - recipe_id (Foreign Key referencing Recipes table)

9 **RecipeOccasions:**

- Description: This table will represent the many-to-many relationship between recipes and occasions.
- Fields:
    - recipe_occasion_id (Primary Key)
    - recipe_id (Foreign Key referencing Recipes table)
    - occasion_id (Foreign Key referencing Occasions table)

Relationships:

One-to-One:

1. **Users - AuthenticationTokens:**
    - Explanation: Each user can have only one authentication token at a time, and each token is associated with only one user. This relationship ensures secure authentication for each user.

One-to-Many:

1. **Users - Recipes:**
    - Explanation: One user can create multiple recipes, but each recipe is created by only one user. This relationship links users to their created recipes.
2. **Recipes - Ingredients:**
    - Explanation: Each recipe can have multiple ingredients, but each ingredient belongs to only one recipe. This relationship allows recipes to list all necessary ingredients.
3. **Users - Occasions:**

    - Explanation: A user can create multiple occasions, but each occasion is created by only one user. This relationship links users to the occasions they've created.
4. **Users - GroceryLists:**
    - Explanation: Each user can have multiple grocery lists, but each grocery list is associated with only one user. This relationship ensures that each user can manage their own lists.
5. **Recipes - UserInteractions:**
    - Explanation: A recipe can have multiple user interactions (likes, comments, views), but each interaction is associated with only one recipe. This relationship tracks user engagement with recipes.

Many-to-Many:

1. **Recipes - Occasions:**
    - Explanation: A recipe can be associated with multiple occasions, and an occasion can have multiple recipes. This relationship allows users to assign recipes to various occasions and vice versa.
2. **Users - Recipes (Favorites):**
    - Explanation: A user can favorite multiple recipes, and each recipe can be favorited by multiple users. This relationship enables users to save recipes they like for easy access later.

Part 2: Table Planning - DB Designer & Word/Google/Pages Doc

Step 1 - DB Designer ---

Step 2

List out each table's respective columns in the table's sub-section and explain for each column:

- *why you'll be storing that data*
- *and why you chose the data type you did*

Columns:

Users:

- **user_id (Primary Key):**
    - Storing this data: To uniquely identify each user in the system and serve as the primary key for the Users table.
    - Data type chosen: INT or SERIAL (auto-incrementing integer) in PostgreSQL.
    - Explanation: INT or SERIAL is commonly used for primary keys as it efficiently stores integer values and ensures uniqueness.

- **email:**
    - Storing this data: To store the email address of users, which is used for authentication and communication purposes.
    - Data type chosen: VARCHAR
    - Explanation: VARCHAR is suitable for storing variable-length strings like email addresses. The length can be adjusted based on the maximum expected length of an email address.
- **password_hash:**
    - Storing this data: To securely store hashed passwords for user authentication.
    - Data type chosen: VARCHAR
    - Explanation: Hashed passwords are stored as strings. VARCHAR is used to accommodate the hashed password's variable length. Length should be chosen to accommodate the output length of the chosen hash algorithm.
- **username:**
    - Storing this data: To store the username chosen by the user for identification.
    - Data type chosen: VARCHAR
    - Explanation: VARCHAR is suitable for storing usernames as they can vary in length. Adjust the length based on the maximum allowed length for usernames in the system.
- **profile_picture_url:**
    - Storing this data: To store the URL of the user's profile picture.
    - Data type chosen: VARCHAR
    - Explanation: VARCHAR is suitable for storing URLs as they are variable-length strings. Adjust the length based on the maximum expected length of a URL.

Recipes:

- **recipe_id (Primary Key):**
    - Storing this data: To uniquely identify each recipe in the system and serve as the primary key for the Recipes table.
    - Data type chosen: INT or SERIAL (auto-incrementing integer) in PostgreSQL.
    - Explanation: INT or SERIAL is commonly used for primary keys as it efficiently stores integer values and ensures uniqueness.
- **title:**
    - Storing this data: To store the title of the recipe.
    - Data type chosen: VARCHAR
    - Explanation: VARCHAR is suitable for storing variable-length strings like recipe titles. Adjust the length based on the maximum expected length of a recipe title.
- **description:**
    - Storing this data: To provide additional information about the recipe.
    - Data type chosen: TEXT
    - Explanation: TEXT is suitable for storing large amounts of text data. Use TEXT for fields where the length may vary significantly, such as recipe descriptions.

- **ingredients:**
  - Storing this data: To store the list of ingredients required for the recipe.
  - Data type chosen: JSON or TEXT
  - Explanation: JSON or TEXT can be used to store structured data like lists of ingredients. Choose JSON if you need to query or manipulate individual elements of the ingredient list.
- **instructions:**
  - Storing this data: To provide step-by-step instructions for preparing the recipe.
  - Data type chosen: TEXT
  - Explanation: TEXT is suitable for storing large amounts of text data. Use TEXT for fields where the length may vary significantly, such as recipe instructions.
- **visibility:**
  - Storing this data: To determine whether the recipe is public or private.
  - Data type chosen: BOOLEAN
  - Explanation: BOOLEAN is suitable for storing binary data (true/false) representing the visibility status of the recipe.
- **user_id (Foreign Key):**
  - Storing this data: To establish a relationship between recipes and their creators (users).
  - Data type chosen: INT
  - Explanation: INT is used to store the user ID of the recipe creator. It references the user_id column in the Users table.
- **occasion_id (Foreign Key):**
  - Storing this data: To associate recipes with occasions (optional).
  - Data type chosen: INT (nullable)
  - Explanation: INT is used to store the occasion ID if the recipe is associated with an occasion. It references the occasion_id column in the Occasions table. Nullable to allow recipes without occasions.

Ingredients:

- **ingredient_id (Primary Key):**
  - Storing this data: To uniquely identify each ingredient in the system and serve as the primary key for the Ingredients table.
  - Data type chosen: INT or SERIAL (auto-incrementing integer) in PostgreSQL.
  - Explanation: INT or SERIAL is commonly used for primary keys as it efficiently stores integer values and ensures uniqueness.
- **name:**
  - Storing this data: To store the name of the ingredient.
  - Data type chosen: VARCHAR
  - Explanation: VARCHAR is suitable for storing variable-length strings like ingredient names. Adjust the length based on the maximum expected length of an ingredient name.
- **quantity:**
  - Storing this data: To store the quantity of the ingredient required in a recipe.
  - Data type chosen: NUMERIC

    - Explanation: NUMERIC is suitable for storing precise numeric values like quantities. It allows for decimal values if needed.
- **unit:**
    - Storing this data: To store the unit of measurement for the ingredient quantity.
    - Data type chosen: VARCHAR
    - Explanation: VARCHAR is suitable for storing variable-length strings like units of measurement. Adjust the length based on the maximum expected length of a unit.
- **recipe_id (Foreign Key):**
    - Storing this data: To establish a relationship between ingredients and the recipes they belong to (optional).
    - Data type chosen: INT (nullable)
    - Explanation: INT is used to store the recipe ID if the ingredient is associated with a recipe. It references the recipe_id column in the Recipes table. Nullable to allow ingredients not associated with any recipe.

Occasions:

- **occasion_id (Primary Key):**
    - Storing this data: To uniquely identify each occasion in the system and serve as the primary key for the Occasions table.
    - Data type chosen: INT or SERIAL (auto-incrementing integer) in PostgreSQL.
    - Explanation: INT or SERIAL is commonly used for primary keys as it efficiently stores integer values and ensures uniqueness.
- **title:**
    - Storing this data: To store the title of the occasion.
    - Data type chosen: VARCHAR
    - Explanation: VARCHAR is suitable for storing variable-length strings like occasion titles. Adjust the length based on the maximum expected length of an occasion title.
- **date:**
    - Storing this data: To store the date of the occasion.
    - Data type chosen: DATE
    - Explanation: DATE is suitable for storing dates without time components. It efficiently stores date values.
- **description:**
    - Storing this data: To provide additional information about the occasion.
    - Data type chosen: TEXT
    - Explanation: TEXT is suitable for storing large amounts of text data. Use TEXT for fields where the length may vary significantly, such as occasion descriptions.
- **user_id (Foreign Key):**
    - Storing this data: To establish a relationship between occasions and their creators (users).
    - Data type chosen: INT

    - Explanation: INT is used to store the user ID of the occasion creator. It references the user_id column in the Users table.

GroceryLists:

- **list_id (Primary Key):**
    - Storing this data: To uniquely identify each grocery list in the system and serve as the primary key for the GroceryLists table.
    - Data type chosen: INT or SERIAL (auto-incrementing integer) in PostgreSQL.
    - Explanation: INT or SERIAL is commonly used for primary keys as it efficiently stores integer values and ensures uniqueness.
- **title:**
    - Storing this data: To store the title or name of the grocery list.
    - Data type chosen: VARCHAR
    - Explanation: VARCHAR is suitable for storing variable-length strings like list titles. Adjust the length based on the maximum expected length of a list title.
- **items:**
    - Storing this data: To store the list of items needed for the grocery list.
    - Data type chosen: JSON or TEXT
    - Explanation: JSON or TEXT can be used to store structured data like lists of items. Choose JSON if you need to query or manipulate individual elements of the item list.
- **checked_off_status:**
    - Storing this data: To store the status of each item on the grocery list (checked off or not).
    - Data type chosen: JSON or TEXT
    - Explanation: JSON or TEXT can be used to store structured data like the checked-off status of items. Choose JSON if you need to query or manipulate individual elements of the status list.
- **user_id (Foreign Key):**
    - Storing this data: To establish a relationship between grocery lists and their owners (users).
    - Data type chosen: INT
    - Explanation: INT is used to store the user ID of the grocery list owner. It references the user_id column in the Users table.

UserInteractions:

- **interaction_id (Primary Key):**
    - Storing this data: To uniquely identify each user interaction in the system and serve as the primary key for the UserInteractions table.
    - Data type chosen: INT or SERIAL (auto-incrementing integer) in PostgreSQL.
    - Explanation: INT or SERIAL is commonly used for primary keys as it efficiently stores integer values and ensures uniqueness.

- **user_id (Foreign Key):**
  - Storing this data: To establish a relationship between user interactions and the users involved.
  - Data type chosen: INT
  - Explanation: INT is used to store the user ID of the user involved in the interaction. It references the user_id column in the Users table.
- **recipe_id (Foreign Key):**
  - Storing this data: To establish a relationship between user interactions and the recipes being interacted with.
  - Data type chosen: INT
  - Explanation: INT is used to store the recipe ID of the recipe being interacted with. It references the recipe_id column in the Recipes table.
- **action:**
  - Storing this data: To store the type of interaction (e.g., Like, Comment, View).
  - Data type chosen: VARCHAR
  - Explanation: VARCHAR is suitable for storing variable-length strings like interaction types. Adjust the length based on the maximum expected length of an interaction type.
- **timestamp:**
  - Storing this data: To store the timestamp of when the interaction occurred.
  - Data type chosen: TIMESTAMP or TIMESTAMP WITH TIME ZONE
  - Explanation: TIMESTAMP is suitable for storing date and time information. Adjust for timezone handling based on your application's requirements.

AuthenticationTokens:

- **token_id (Primary Key):**
  - Storing this data: To uniquely identify each authentication token in the system and serve as the primary key for the AuthenticationTokens table.
  - Data type chosen: INT or SERIAL (auto-incrementing integer) in PostgreSQL.
  - Explanation: INT or SERIAL is commonly used for primary keys as it efficiently stores integer values and ensures uniqueness.
- **user_id (Foreign Key):**
  - Storing this data: To establish a relationship between authentication tokens and the users they belong to.
  - Data type chosen: INT
  - Explanation: INT is used to store the user ID of the user associated with the authentication token. It references the user_id column in the Users table.
- **token:**
  - Storing this data: To store the actual authentication token value.
  - Data type chosen: VARCHAR or TEXT
  - Explanation: VARCHAR or TEXT is suitable for storing the token value. The length may vary based on the length of the token generated by the authentication system.
- **expiry_date:**
  - Storing this data: To store the expiry date of the authentication token.

  - Data type chosen: TIMESTAMP or TIMESTAMP WITH TIME ZONE
  - Explanation: TIMESTAMP is suitable for storing date and time information. Adjust for timezone handling based on your application's requirements.

Favorites:

- **favorite_id (Primary Key):**
  - Storing this data: To uniquely identify each favorite entry in the system and serve as the primary key for the Favorites table.
  - Data type chosen: INT or SERIAL (auto-incrementing integer) in PostgreSQL.
  - Explanation: INT or SERIAL is commonly used for primary keys as it efficiently stores integer values and ensures uniqueness.
- **user_id (Foreign Key):**
  - Storing this data: To establish a relationship between favorite entries and the users who favorited them.
  - Data type chosen: INT
  - Explanation: INT is used to store the user ID of the user who favorited the recipe. It references the user_id column in the Users table.
- **recipe_id (Foreign Key):**
  - Storing this data: To establish a relationship between favorite entries and the recipes that are favorited.
  - Data type chosen: INT
  - Explanation: INT is used to store the recipe ID of the favorited recipe. It references the recipe_id column in the Recipes table.

RecipeOccasions:

- **recipe_occasion_id (Primary Key):**
  - Storing this data: To uniquely identify each entry in the RecipeOccasions table and serve as the primary key.
  - Data type chosen: INT or SERIAL (auto-incrementing integer) in PostgreSQL.
  - Explanation: INT or SERIAL is commonly used for primary keys as it efficiently stores integer values and ensures uniqueness.
- **recipe_id (Foreign Key):**
  - Storing this data: To establish a relationship between recipes and the occasions they belong to.
  - Data type chosen: INT
  - Explanation: INT is used to store the recipe ID. It references the recipe_id column in the Recipes table.
- **occasion_id (Foreign Key):**
  - Storing this data: To establish a relationship between occasions and the recipes associated with them.

- Data type chosen: INT
- Explanation: INT is used to store the occasion ID. It references the occasion_id column in the Occasions table.

These column definitions establish the necessary relationships between entities and store relevant data for each table in the database. Adjustments can be made based on specific application requirements and database design considerations.

Part 3: Create Tables in SQL – pgAdmin

Here are the SQL statements to create the tables:

Users table:

```
CREATE TABLE Users (
   user_id SERIAL PRIMARY KEY,
   email VARCHAR(255) UNIQUE NOT NULL,
   password_hash VARCHAR(128) NOT NULL,
   username VARCHAR(50) UNIQUE NOT NULL,
   profile_picture_url VARCHAR(255)
);
```

Recipes tables:

```
CREATE TABLE Recipes (
   recipe_id SERIAL PRIMARY KEY,
   title VARCHAR(255) NOT NULL,
   description TEXT,
   ingredients JSONB,
   instructions TEXT,
   visibility BOOLEAN NOT NULL,
   user_id INT REFERENCES Users(user_id)
);
```

Ingredients table:

```
CREATE TABLE Ingredients (
   ingredient_id SERIAL PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   quantity NUMERIC,
   unit VARCHAR(50),
   recipe_id INT REFERENCES Recipes(recipe_id)
);
```

Occasions table:

```
CREATE TABLE Occasions (
    occasion_id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    date DATE,
    description TEXT,
    user_id INT REFERENCES Users(user_id)
);
```

GroceryLists table:

```
CREATE TABLE GroceryLists (
    list_id SERIAL PRIMARY KEY,
    title VARCHAR(255),
    items JSONB,
    checked_off_status JSONB,
    user_id INT REFERENCES Users(user_id)
);
```

UserInteractions table:

```
CREATE TABLE UserInteractions (
    interaction_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES Users(user_id),
    recipe_id INT REFERENCES Recipes(recipe_id),
    action VARCHAR(50) NOT NULL,
    timestamp TIMESTAMP
);
```

AuthenticationTokens table:

```
CREATE TABLE AuthenticationTokens (
    token_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES Users(user_id),
    token TEXT NOT NULL,
    expiry_date TIMESTAMP
);
```

Favorites table:

```
CREATE TABLE Favorites (
    favorite_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES Users(user_id),
    recipe_id INT REFERENCES Recipes(recipe_id)
);
```

RecipeOccasions table:

```
CREATE TABLE RecipeOccasions (
    recipe_occasion_id SERIAL PRIMARY KEY,
    recipe_id INT REFERENCES Recipes(recipe_id),
    occasion_id INT REFERENCES Occasions(occasion_id)
);
```

Intermediate

- Try inserting some data into your tables using `INSERT INTO` statements in the `psql` console.

1. **Inserting data into the Users table:**

```
INSERT INTO Users (email, password_hash, username, profile_picture_url)
VALUES ('example1@example.com', 'hashed_password1', 'user1', 'https://example.com/profile_pic1.jpg'),
       ('example2@example.com', 'hashed_password2', 'user2', 'https://example.com/profile_pic2.jpg');
```

2. **Inserting data into the Recipes table:**

```
INSERT INTO Recipes (title, description, ingredients, instructions, visibility, user_id)
VALUES ('Recipe 1', 'Description for Recipe 1', '{"ingredient1": "quantity1", "ingredient2": "quantity2"}',
'Instructions for Recipe 1', TRUE, 1),
       ('Recipe 2', 'Description for Recipe 2', '{"ingredient1": "quantity1", "ingredient2": "quantity2"}',
'Instructions for Recipe 2', FALSE, 2);
```

3. **Inserting data into the Ingredients table:**

```
INSERT INTO Ingredients (name, quantity, unit, recipe_id)
VALUES ('Ingredient 1', 100, 'grams', 1),
       ('Ingredient 2', 2, 'cups', 1),
       ('Ingredient 3', 250, 'milliliters', 2);
```

4. **Inserting data into the Occasions table:**

```
INSERT INTO Occasions (title, date, description, user_id)
VALUES ('Occasion 1', '2024-03-15', 'Description for Occasion 1', 1),
       ('Occasion 2', '2024-04-20', 'Description for Occasion 2', 2);
```

5. **Inserting data into the GroceryLists table:**

```
INSERT INTO GroceryLists (title, items, checked_off_status, user_id)
VALUES ('Grocery List 1', '{"item1": "quantity1", "item2": "quantity2"}', '{"item1": false, "item2":
true}', 1),
       ('Grocery List 2', '{"item3": "quantity3", "item4": "quantity4"}', '{"item3": false, "item4": false}', 2);
```

6. **Inserting data into the UserInteractions table:**

```
INSERT INTO UserInteractions (user_id, recipe_id, action, timestamp)
VALUES (1, 1, 'Like', CURRENT_TIMESTAMP),
       (2, 1, 'View', CURRENT_TIMESTAMP);
```

7. **Inserting data into the AuthenticationTokens table:**

```
INSERT INTO AuthenticationTokens (user_id, token, expiry_date)
VALUES (1, 'auth_token_1', '2024-06-01 12:00:00'),
       (2, 'auth_token_2', '2024-06-01 12:00:00');
```

8. **Inserting data into the Favorites table:**

```
INSERT INTO Favorites (user_id, recipe_id)
VALUES (1, 1),
       (2, 2);
```

9. **Inserting data into the RecipeOccasions table:**

```
INSERT INTO RecipeOccasions (recipe_id, occasion_id)
VALUES (1, 1),
       (2, 2);
```