Stage 2 - Conceptual and Logical Database Design

Entity and relationship assumptions :

Figure : Relationship Diagram

1. User
   a. Attributes
      i. user_id INT (PK),
      ii. username String,
      iii. email String,
      iv. password String,
      v. created_at datetime
   b. Assumptions
      i. Each new user will be assigned a unique user_id
      ii. One-to-many relationship with scans: each user can perform multiple scans, but each scan is associated with only one user so that we can trace the scans back to the users
   c. Relations
      i. User → Scans: One-to-Many : A user can have multiple scans, but each scan has only one user.
      ii. User -> Allergen : Many-to-Many : A user can have multiple allergens. Multiple users can have the same allergens
      iii. User ->Diet : Many-to-Many: A user can have multiple diets. Multiple users can have the same diets
2. Scans
   a. Attributes
      i. Scan_id INT (PK),
      ii. user_id INT,
      iii. image_data String,   -- need base 64 format to paste into LLM
      iv. ocr_text String,   -- text extracted via OCR
      v. raw_llm_response String,   -- raw JSON/text response from the LLM (with name, description, price, etc.)
      vi. status String,   -- e.g., "pending", "completed"
      vii. created_at datetime
   b. Assumptions
      i. Once a picture is uploaded, we must encode it in a base 64 format in order to perform optical character recognition (OCR)
      ii. LLM Response: The large language model returns JSON/text that can be stored as raw_llm_response. We parse this later to create Menu_items.
      iii. Each scan is associated with exactly one user.
   c. Relations

        i. Scans → User: Many-to-One : One user can have many scans, but each scan must be associated with at most one user.
        ii. Scans → Menu_items: One-to-many : A single scan can produce multiple menu_items

3. Menu_item
    a. Attributes
        i. menu_item_id INT(PK),
        ii. scan_id INT,
        iii. dish_name String,
        iv. description String,
        v. price INT,
        vi. created_at datetime
    b. Assumptions
        i. We can derive zero or multiple allergens from each ingredient.
        ii. We can determine the kind of diet a menu item falls under based on the ingredients.
    c. Relations
        i. Menu_items -> Scans : Many-to-One : Many menu items can come from the same scan
        ii. Menu_item -> ingredient: A single scan can have multiple ingredients, and one ingredient (e.g. potatoes) can appear in multiple menu items.

4. Allergens
    a. Attributes
        i. allergen_id INT (PK)
        ii. String name
    b. Assumptions
        i. We will store this data from official sources
        ii. Allergens can be linked to Ingredients or Menu_items
    c. Relations
        i. Allergens -> Ingredients : Many-to-Many : An ingredient can have multiple allergens, and one allergen can be present in multiple ingredients.
        ii. Allergens -> User : Many-to-Many : Any user can have multiple allergens.

5. Diet
    a. Attributes
        i. Diet_id INT (PK)
        ii. Name String (e.g. “Vegan” , “Keto”)
    b. Assumptions
        i. A user can have multiple diets.
        ii. A menu item can fall under one or more diets.
    c. Relations

i. User -> Diet : Many-to-Many : one or more user can have one or more of the same diets.
ii. Diet-> Ingredients : Many-to-Many : one diet can restrict many ingredients, and one ingredient can be restricted by many diets

6. Ingredients
   a. Attributes
      i. Ingredient_id INT (PK)
      ii. String name
   b. Assumptions
      i. Each ingredient can be associated with zero or multiple allergens.
      ii. The same ingredient can appear in multiple menu_items
   c. Relations
      i. Ingredients -> Menu_Items : Many-to-Many
      ii. Ingredients -> Allergens : Many-to-Many
      iii. Ingredients -> Diet : Many to many

Justification of Database Normalization:
We are using 3NF normalization. In all of the tables, all functional dependencies have a key/superkey on the left side, and there are no transitive dependencies,

User Table:
- In all functional dependencies, the primary key (user_id) is on the left side. This satisfies the rule that all functional dependencies must have a key on the left side.
- None of the non-key attributes depend on another non-key attribute. They all depend directly on the primary key (user_id).

Menu_items:
- In all functional dependencies, the primary key (menu_item_id) is on the left side.
- None of the non-key attributes depend on another non-key attribute. They all depend directly on the primary key (menu_item_id).

Scan:
- In all functional dependencies, the primary key (scan_id) is on the left side.
- None of the non-key attributes depend on another non-key attribute. They all depend directly on the primary key (scan_id).

Ingredients:
- In all functional dependencies, the primary key (ingredient_id) is on the left side.
- None of the non-key attributes depend on another non-key attribute. They all depend directly on the primary key (ingredient_id).

Diet:
- In all functional dependencies, the primary key (diet_id) is on the left side.

- None of the non-key attributes depend on another non-key attribute. They all depend directly on the primary key (diet_id).

Allergens:

- The primary key (allergen_id) is on the left side of the functional dependency.
- There are no non-key attributes that depend on another non-key attribute. The non-key attribute (name) depends directly on the primary key (allergen_id).

Relational Schema:

User(user_id:INT [PK], username:VARCHAR(255), email:VARCHAR(255), password:VARCHAR(255), created_at:DATETIME)

Scan(scan_id:INT [PK], user_id:INT [FK to User.user_id], image_data:TEXT, ocr_text:TEXT, raw_llm_response:TEXT, status:VARCHAR(20), created_at:DATETIME)

Menu_Item(menu_item_id:INT [PK], scan_id:INT [FK to Scan.scan_id], dish_name:VARCHAR(255), description:TEXT, price:REAL, created_at:DATETIME)

Ingredient(ingredient_id:INT [PK], name:VARCHAR(255))

Allergen(allergen_id INT [PK], name:VARCHAR(255))

Diet(diet_id:INT [PK], name:VARCHAR(255))

User_Allergen(user_id:INT [PK] [FK to User.user_id], allergen_id INT [PK] [FK to Allergen.allergen_id])

User_Diet(user_id:INT [PK] [FK to User.user_id], diet_id:INT [PK] [FK to Diet.diet_id])

Menu_Item_Ingredient(menu_item_id:INT [PK] [FK to Menu_item.menu_item_id], ingredient_id:INT [PK] [FK to Ingredient.ingredient_id])

Ingredient_Allergen(ingredient_id:INT [PK] [FK to Ingredient.ingredient_id], allergen_id:INT [PK] [FK to Allergen.allergen_id])

Diet_Ingredient(diet_id:INT [PK] [FK to Diet.diet_id], ingredient_id:INT [PK] [FK to Ingredient.ingredient_id])