
Entity 1: User

We need a **User** entity to store authentication details(email, password) and user-specific data(like their default location). This information does not belong as attributes of any other entity (such as Location or Product) because each user has unique credentials, and those credentials do not describe a location or product directly.

Assumptions

1. Users can log in from anywhere, but each user has a **default location** stored in location_id.
2. Each user must have **exactly one** location in the system (**but many users may share the same location record**).
3. **A user can only have one grocery list.**

Relationships

1. **User** ↠ **Location**: Many-to-one (One location can be referenced by multiple Users).
2. **User** ↠ **GroceryProduct**: One-to-many (Each user can have many grocery products, and each GroceryProduct belongs to exactly one user).

Entity 2: Location

Locations can be shared by multiple Users. Instead of duplicating location attributes inside User, a separate Location table avoids redundancy and ensures consistency.

Assumptions

1. One Location entry can be linked to multiple users and multiple products.

Entity 3: Product

Every distinct food/produce item is stored here. Treating Product as an entity makes sure that products can appear on multiple grocery lists or be referenced independently.

Assumptions

1. There must be at least one product per importer in the table.
2. Each product is imported by exactly one importer (i.e., importer_id is a foreign key in the Product table).
    - If a product has multiple importers, they are separate row entries with different product_id values.
3. Product information and Environmental Cost data should be stored separately for clarity.

Relationships

1. **Product** ↠ **Location**: Many-to-one (A single location can be home to many different products, but each product is found at exactly one location).
2. **Product** ↠ **EnvironmentalCost**: One-to-one (Each product maps to a specific Environmental_Cost entry).
3. **GroceryProduct** ↠ **Product**: Many-to-one (Many entries in Grocery_Product can reference the same Product).

Entity 4 : GroceryProduct

GroceryProduct is a table containing the products in each user's grocery lists. We use this entity to store each item in a user’s grocery list by mapping combinations of UserIds and ProductIds for every item in a user’s grocery list.

Entity 5: EnvironmentalCost

We want to capture the individual environmental footprint of each product. We store this as an entity separate from the Product table for clarity.

4. Normalize your database. Apply BCNF or 3NF to your schema or show that your schema adheres to one of these normal forms.

We look at the Formal Dependencies of each entity in order to determine if it follows BCNF:

1. **User Table**:
    - UserId ↠ {FirstName, LastName, EmailId, Password, UserLocationId}
    - UserLocationId ↠ LocationId (a user is linked to only one location)
    - **User Table**: UserId is the primary key and superkey and determines all attributes.
2. **Location Table**:
    - LocationId ↠ {Latitude, Longitude, City, Country}
    - **Location Table**: LocationId is the primary key and superkey and determines all attributes.
3. **Product Table**:
    - ProductId ↠ {ProductName, FoodGroup, ImporterId}
    - LocationId ↠ LocationId (Foreign key that references Location Table)
    - **Product Table**: ProductId is the primary key and determines all attributes.
4. **Grocery Product Table**:
    - {UserId, ProductId} ↠ {Quantity}
        i. UserIdand ProductId make up the primary key
    - **Grocery Product Table**: In BCNF, as (UserId and ProductId) uniquely determines the tuple and determines the remaining attribute.
5. **Environmental Cost Table**:
    - EC_Id ↠ {ProductId, CarbonFootPrint, LandUse, WaterUse, Eutrophication, LandGH, AnimalGH, FarmGH, ProcessingGH, TransportGH, PackagingGH, RetailingGH, TotalEmissions}

    - ProductId ↠ ProductId (Foreign key that references Product Table)
    - **Environmental Cost Table**: In BCNF, as ProductId uniquely determines all attributes.

Because in every table, each non trivial functional dependency has a superkey on the left side, each relation satisfies BCNF. Therefore, the entire database is compliant with BCNF (and therefore with 3NF as well).

5. Convert your conceptual database design (ER/UML) to the logical design (relational schema). Note that a relational schema is NOT an SQL DDL command.

User(UserId: INT [PK], FirstName: VARCHAR(255), LastName: VARCHAR(255), Email: VARCHAR(255), Password: VARCHAR(255), UserLocationId: INT [FK to Location.LocationId])

Location(LocationId: INT [PK], LocLat: DECIMAL(8,5), LocLong: DECIMAL(8,5), City: VARCHAR(255), LocCountry: VARCHAR(255))

Product(ProductId: INT [PK], ProductName: VARCHAR(255), FoodGroup: VARCHAR(255), LocationId: INT [FK to Location.LocationId])

Grocery_Product(UserId: INT [FK to User.UserId], ProductId: INT [FK to Product.ProductId], Quantity: INT, [PK(UserId, ProductId)])

Environmental_Cost(EC_Id: INT [PK], ProductId: INT [FK to Product.ProductId], CarbonFoot: DECIMAL(16,2), LandUse: DECIMAL(16,2), WaterUse: DECIMAL(16,2), Eutrophication: DECIMAL(6,2), LandGH: DECIMAL(18,3), AnimalGH: DECIMAL(18,3), FarmGH: DECIMAL(18,3), ProcessingGH: DECIMAL(18,3), TransportGH: DECIMAL(18,3), PackagingGH: DECIMAL(18,3), RetailingGH: DECIMAL(18,3), TotalEmissions: DECIMAL(18,3))