Stage 2: Database Design

Bennett Wu [bwu40]

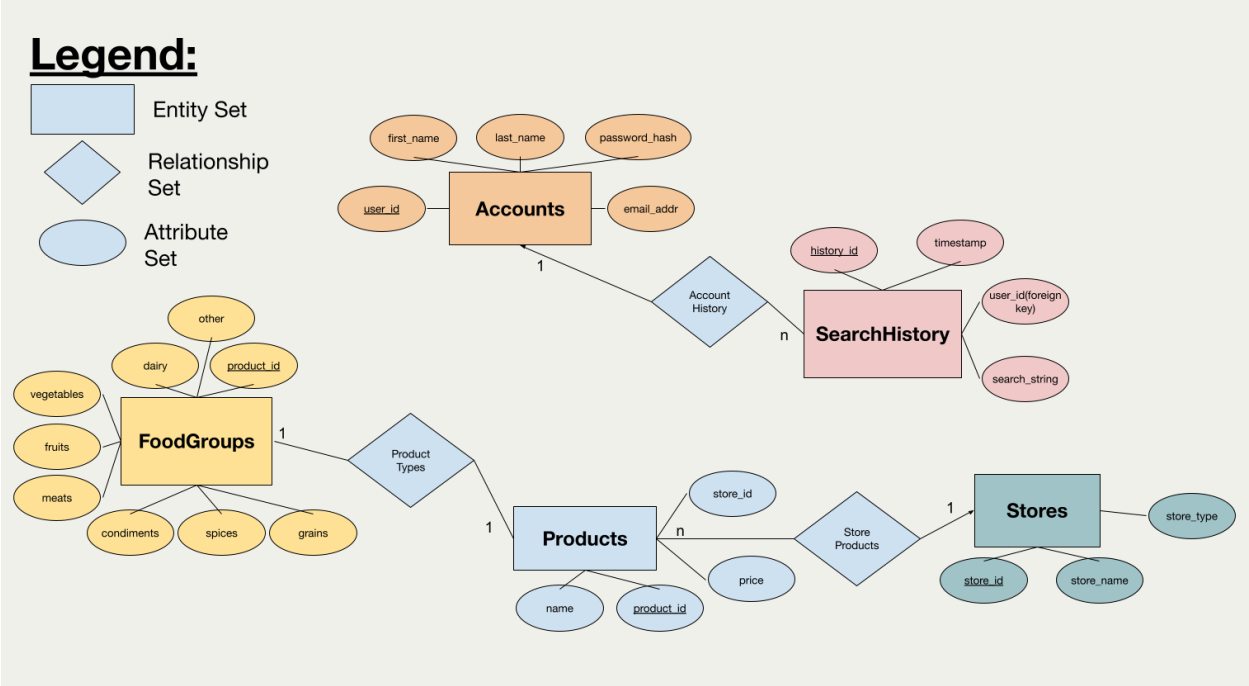
Jeremy Lee [jeremyl6]

Paul Jeong [paulj3]

Shreya Singh [shreya15]

Entity Relation Diagram

The following is the Entity-Relation Diagram used to illustrate the layout and relationships between the various entities present within the database's design.



Assumptions - Entities

This section details assumptions made for each entity in the model. The assumptions are split up per-entity, and include an explanation of why each entity is modeled as an entity, and not attributes of another table.

FoodGroups

The FoodGroups entity is primarily centered around the categorization of the food products we may encounter within the datasets.

a. **Entity vs Attribute:**
    i. FoodGroups is an entity as opposed to attributes of the Products entity to prevent unnecessary complexity and improve readability - with the addition of the FoodGroups flags into the Products entity, the categorization of products would be increasingly difficult to identify amongst the other information present within the Products entity.

b. Assumptions:

i. Every product is some type of consumable object.
   - Due to the application's focus on grocery prices, the datasets and tables will have any non-food products and items removed.

ii. There are enough food groups to encompass the entirety of modern household groceries.
   - Although groceries have a wide variety of categories they can fall under, this entity should be able to capture the various products that are made available by grocery stores.
   - While most of the food groups are targeted towards various nutritional categories, the addition of other food types such as *condiments, spices,* and *other* should be able to categorize the remainder of any items that do not fall under any specific nutritional category. For example: ketchup, chocolate, and chips.

Products

The Products entity is primarily centered around the identification of products made available by major grocery stores.

a. Entity vs Attribute:

i. Products is an entity rather than attributes of another entity due to the fact that it is commonly referenced - Products contains all of the product information we will be referencing, and thus should be represented by its own group of items (as its own entity).

b. Assumptions:

i. Product IDs are completely unique
   - In order to be able to uniquely identify products across stores, each product will be assigned a unique Product ID.

ii. New products possess unique Product IDs
   - Any new products added to the application will be given a completely unique Product ID, to avoid any identification issues that have the potential to arise.

iii. Product Names may not necessarily be unique, but the product will still be able to be identified via unique Product ID

- Stores may utilize identical names based on information given by suppliers - however, these products may not necessarily be the same, and thus they will continue to possess a unique Product ID.

Stores

The Stores entity is primarily centered around identification of stores to which the products belong.

a. **Entity vs Attribute:**
    i. The reason we chose this to be an entity as opposed to an attribute is because we want the store name to be easily changeable, and storing the name as an entity, as opposed to an attribute, would make the change easier and more efficient.
b. **Assumptions:**
    i. The StoreId will be unique to a particular store name. For example, we can map all Walmart stores to 1.
        - StoreId will primarily be used as an identifier to aid with product comparison between various stores.
        - This will also make it easier for us to add / update any stores in the future, if we find more databases with relevant information.

SearchHistory

The SearchHistory entity is primarily centered around keeping record of search inputs to a user.

a. **Entity vs Attribute:**
    i. SearchHistory is an entity rather than an attribute because it is expected that there will be multiple instances, that is searches, per user. Additionally, it is expected to query and report the search history, which is practically done on an entity.
b. **Assumptions:**
    i. Each Search String is automatically saved, with a unique History Id attached to the string for identification purposes.
        - In order to differentiate searches between users and instances, each Search String will be automatically saved so that users may be able to recall their previous history for comparison purposes.

ii. Each Search String is limited to 32 words with an average character count of 4 characters per word.
    - To limit excessive Search Strings, the maximum length of a string will be set to a maximum of 32 words with an average of 4 characters per word.

Accounts

The Accounts entity is primarily centered around identifying the various users of the application, along with their personal information.

a. Entity vs Attribute:

i. Accounts is an entity rather than attributes of another entity due to reasons similar to Products - the entity is commonly referenced, and is intended to provide identifying information related to objects (people) in the real world. Thus, Account should be made an entity.

b. Assumptions:

i. Each account must have a unique email address and unique user_id to fit the criteria of being unique. We will add the unique constraint to it.
    - We do not want to limit people to one account, but in order to uniquely identify accounts and logins, we add the additional constraint of unique email addresses to further identify our accounts.

Assumptions - Relationships

This section details assumptions made for each relationship in the model. The assumptions are split up per-relationship

ProductTypes

The ProductTypes Relationship represents the food groups that each product in Products falls under.

Assumptions:

1. Every product has at least one type of Food Group the product can be categorized under, and only one category.
    - Every food product must be able to fall under at least one category so that it can be sorted later. The number of categories should be all-encompassing enough that it can categorize modern groceries.

StoreProducts

The StoreProducts relationship represents the various products each store possesses.

Assumptions:

1. Every product belongs to a store, and one store ONLY
    - Every product that is drawn from a dataset must belong to a store. Otherwise, the application will not be able to draw up proper comparisons when comparing a product's cost in comparison to other products and stores.
    - Each product is uniquely identified by its product id, but for further identification the application must be able to refer to additional identifying information to differentiate products with similar names, outside of their product ids.
2. Every store contains at least one product, but will likely contain multiple products
    - The application will only include stores that it has product data for. Thus, every store mentioned must have at least one product, and each store can have as many products as it needs.

AccountHistory

The AccountHistory relationship represents the various search queries associated with each account present within the database.

Assumptions:

1. Each account may have multiple searches
    - In order to properly represent the interests and history of each account, the application will not limit each account to a single instance of search history.
    - Each search will be uniquely identified by its history id, along with its search string and timestamp, subsequently associated with an account so that an account's entire search history may be brought up if necessary

Cardinality of Relationships

The following section details the cardinality of each relationship previously mentioned, further elaborating on their specific cardinalities between attributes.

1. **ProductTypes:** The ProductTypes relationship will possess a **One-to-One Relationship** between the FoodGroups and Products entities. Each unique ProductId in Products will be mapped to a matching ProductId in FoodGroups, making it a one-to-one relationship - we will use the mapped ProductId and its corresponding Food Group to further identify and categorize each product.
2. **StoreProducts:** This Stores table is going to have a **Many-to-One relationship** with the Products table. Each unique ProductId is going to be mapped to a certain store. This is why multiple ProductIds will be mapped to the to only one value in the Stores table. For example, if we say that Walmart's store_id is 1 then all the products in the products table with different ProductIds (many) will be mapped to one store (one.)
3. **AccountHistory:** The Accounts table is going to have a **Many-to-One** relationship with the SearchHistory table. Multiple records in the SearchHistory table are going to be mapped to the same userId. The same user can have multiple search records, which will all be stored in the SearchHistory table. For example, a user_id with value 1, can make multiple searches, with different history_ids, all stored in the SearchHistory table.

Normalization of Database

Instead of using Boyce-Codd Normal Form (BCNF), the database will implement Third Normal Form (3NF) to achieve decomposition and remove redundancies. The database uses 3NF over BCNF because 3NF gives us dependency preservation - when refining our schema, we want to ensure that any dependencies we want to establish remain intact after decomposition. Even though BCNF gives us less redundancy, it does not leave all dependencies intact - thus to eliminate the risk of removing any essential dependencies, the database will implement 3NF instead.

The process for decomposition into 3NF is as follows:

1. Get a *"Minimal Basis" G* of given FDs
2. For each FD $A \rightarrow B$ in the minimal basis *G*, use *AB* as the schema of a new relation
3. If none of the schemas from Step 2 is a superkey, add another relation whose schema is a key for the original relation

Using the following method, we can generate our refined schema. We first list our Functional Dependencies (FDs) in order to determine our minimal basis:

Functional Dependencies:

1. product_id → store_id
2. product_id → dairy
3. product_id → vegetables
4. product_id → fruits
5. product_id → meats
6. product_id → condiments
7. product_id → spices
8. product_id → grains
9. product_id → others
10. history_id → user_id
11. history_id → SearchString

Based on the FDs, we can see that no reduction can be performed to reduce redundancy within the FDs, and thus that the Minimal Basis Set is the set of FDs

previously listed. Thus, we decompose our relations (with arbitrary and unique one-character names) into the following:

Relations:

1. A(product_id, store_id)
2. B(product_id, vegetables)
3. C(product_id, fruits)
4. D(product_id, meats)
5. E(product_id, condiments)
6. F(product_id, spices)
7. G(product_id, grains)
8. J(product_id, others)
9. H(history_id, user_id)
10. I(history_id, search_string)

However, we can see that the relations above does *not* include a Superkey - in order to fully enclose every attribute within the database, we determine the closure of our two Left-Hand-Side attributes, product_id and history_id, to determine what attributes need to be added to the Superkey:

product_id+={ product_id, name, price, store_id, store_name, vegetables, fruits, meats, condiments, spices, grains, others }

history_id+={ history_id, saved, user_id, search_string, first_name, last_name, password_hash, email_addr)

By determining the closure of product_id and history_id, we can see that these two attributes can individually reach every other attribute in their respective related entities - thus, we generate our Superkey with a relation involving these two elements, as so:

SUPERKEY: J{ product_id, history_id }

With the addition of the Superkey, we conclude our schema refinement.

Conceptual Design to Logical Design

The following schemas represent the conversion of our Conceptual Database Design to Logical Designs, with tables for each of the aforementioned entities:

```
Accounts (
        user_id : INT [PK],
        first_name : CHAR(256),
        last_name : CHAR(256),
        password_hash : CHAR(256),
        email_addr : CHAR(256)
)
```

```
SearchHistory (
        history_id : INT [PK],
        user_id : INT [FK to Accounts.user_id],
        search_string : CHAR(256),
        timestamp : INT
)
```

```
FoodGroups (
        product_id : INT [PK, FK to Products.product_id],
        dairy : BOOLEAN,
        vegetables : BOOLEAN,
        fruits : BOOLEAN,
        meats : BOOLEAN,
        condiments : BOOLEAN,
        spices : BOOLEAN,
        grains : BOOLEAN,
        other : BOOLEAN
)
```

```
Stores (
        store_id : INT [PK],
        store_name : CHAR(256),
        store_type : CHAR(256)
)
```

```
Products (
        product_id : INT [PK],
        store_id : INT [FK to Stores.store_id]
        name : CHAR(256),
        price : REAL,
)
```