1 Introduction

2 Assumptions and Explanations

2.1 Product, ProductProperty, Expiry and Label

In this design I have chose to go for a more flexible approach instead of making a design that enforces the law, since if the design would look something like:

Although this design where you have the attributes on the front and the back in the respective tables enforces the current law, any change in the law would need a change in the database. For example, if law made it mandatory to store the amount of Whey Protein then this design would need to change and if this field is marked "Not Null" then the old products (not manufactured any more) for which whey protein was not stored will need to be filled with dummy values. This database will also fail to be a good historical reference since it will only enforce "current" law and any historical reference would be lost with the necessary change of design with change of laws, and I think it is safe to assume there will be a script used to enter the data into the database which would allow use to code in order to enforce the current law allowing the database to be more flexible. Moreover, if a script is not available then the Database Admin who is entering the data should have easy access to the law, and the data that would be measured or given to him/her would only be measured if required.

I have chosen a very general design where a Product is associated with optionally many rows of ProductProperty (which is a supertype of disjoint double line subtypes) and optionally many instances of Label (I chose optionally many because some basic products don't need to have food labelling and so don't need to have stored properties and labels, but my design still allows them to have food labelling). The Label for a Product has LabelType which may be front, back etc and a SubSection which may be left or upper-left etc. Every Label is connected to many instances of ProductProperty and a ProductProperty can be displayed on many instances of Label. So we have a n-m relationship which resolved to an associative entity called LabelContent. The LabelContent has a ValidFrom and a ValidTo allowing us to use the same Label multiple times (meaning the law changes and reverts back, then we can use historical data). Label doesn't have a ValidFrom and ValidTo since if any ProductProperty has a ValidTo of NULL int LabelContent then the Label is currently in use, but if the CEO so wants a ValidFrom and ValidTo could be added to the Label as well, the trade-off being increased data redundancy and increase in complexity.

Now, every Product has atleast one Expiry associated with it, meaning that we need to store either 'UseBy' or 'BestBefore' for every product (this is good practice but the relationship can be made optional if the expiry of things like flour don't need to be stored). Any product can have only two expiry intances associated to it due to the ENUM corresponding to 'UseBy' and 'BestBefore' respectively. Every Expiry is optionally connects to the associative entity for the n-m relationship between Expiry and Label, again this has been left optional due to basic goods not requiring food labelling.

Also, I allow for the Product to have a change in ProductProperty without a change in the Barcode (The barcode is part of Primary Key since a Product should have a unique barcode, I have a ProducedFrom which is also part of the Primary Key since it allows us to differentiate between different iterations of the same product). The fact, that a ProductProperty can change (For example, the use of seasonal fruits) without changing barcode gives S.T.A.R industries control over when they want to give different barcodes to products (but barcodes should be changed in order to comply with law). We can view current product properties by checking for ValidTo of the ProductProperty to be NULL.

2.2 Supertype and subtype

A lot of properties of the product can be used as a separate entity for the purpose of normalisation. These normalised entities (Manufacture, Rating, Nutrient , Ingredient, Warning, Serving and TotalWeight) have been grouped under the supertype ProductProperty. This supertpe-subtype relationship is disjoint and has a = meaning that every ProductProperty must be of only one subtype. I will explain why it makes sense to have such a relationship for each of the subtypes below:

Manufacture - My design allows a product to have many manufacturers i.e. places where a type of product is made. I store the address in a very general table for addresses but make MinorMunicipality to be Suburb, MajorMunicipality to be City and Governing District to be State since we were told Batmania is similar to Australia.

Ingredient - The design allows a Product to have multiple ingredients (as it should) and also allows the manufacturer to display any flattering details on the front as well as the back. For example if Mitchell's Soup contains high-quality tomatoes then the company can display that on both the sides.

All of the 'amounts' stored in my design will be the total amount of that ProductProperty (with Unit associated ofcourse).

TotalWeight, Serving - Both of these properties need to be displayed on the labels and making them ProductProperty subtypes allows us the flexibility to do so. I am assuming here that NumberOfServing and ServingSize are displayed together most of the time, but if it's not the case the design can be easily modified to accomodate this.

Nutrient - Every Product will have many instances of type Nutrient, a Nutrient has a NutrientType (eq Protein, Carbohydrates). Every NutrientType can be of the type of another NutrientType (sugar being a type of carbohydrate). I store the amount of every Nutrient since calculating the amount using nutrients of that type may not give us the exact answer (due to them not being exhaustive). Moreover every Nutrient associated to a Product may have a NutrientCategorisation (like high/low, I allow the NutrientCategoryName to be NULL to accomodate for nothing being displayed for medium) which is a time-varying property of the NutrientType. I store the lower threshold and the upper threshold (can be NULL for high) for a category corresponding to a nutrient type in order to be able to use that data in the future or in calculations. I also have a table called DailyIntake which is also a time-varying property of the NutrientType and stores the average daily intake of that NutrientType (this helps in percentage related calculations). I am assuming here, we can treat Energy as a Nutrient although it's strictly not a Nutrient because the corresponding table for energy would be the same as the Nutrient table (if not then a design change would be straightforward).

Rating, Warning - A product can have many ratings if the law changes or if the traffic light system is replaced by star system (the ValidFrom and ValidTo of ProductProperty helps in storing this kind of information). My design allows the company to store all possible warnings (maybe, provided by the government) in WarningType and then associate many instances of WarningType to Warning which will store the Warning associated with a Product (the design obviously allows for one Product to have many warnings).

2.3 Some other remarks

I have a DisplayStyle associated with every row of LabelContent which allows us to make boolean values true if a specific ProductProperty was displayed in a specific manner on the Label. The DisplayStyleID can be left out for things that don't have amounts associated with them. This allows me to store the total amount of nutrients and a boolean which tells us what style was used to display the Nutrient, the actual value on the packaging is just a simple calculation away (for example, we can find all the nutrients per serving by creating a new table

which is basically TotalAmountOfNutrient/NumberOfServings both this information is easily available to us and SQL is more than capable of doing this sort of calculation). I avoid a lot of data redundancy by doing this, although there is a trade-off with speed.

I have ProductName, Website and MadeInMessage as part of Product in the interest of De-normalisation, since having those three as separate entities would make the design more complicated without adding much value to it. I have booleans in my Label in order to store whether they are displayed or not.

The value ExpiryPeriod in Expiry stores in a VARCHAR, data about how long the shelf life of a product is (for example, 30 days from packaging). MadeInMessage may contain "Made Proudly in Batmania", but the Country can also be accessed from the Address table.