I think it's a good starting point. It gives a chance to see more clearly what's missing. I think though that this structure lets you figure out what ingredients went into the product but not how much it cost to make it (not the Cost of Goods Sold). There are 2 problems – price and quantity.

With regard to price:

For example, looking at the Products table first, lets say there is a record for Cabbage Soup with a Product ID of P100 and it has a selling price of $2.

These parts work:

It's possible to work backwards and see which ingredients went into the product. Let's say one of them is cabbage and the Product\_Ingredients table has a record of that with a record P100 I200.

So, with the I200 you can find from Ingredients table that it has a name of "Cabbage".

But then you hit a problem:

When you look in the Ingredients\_Provider table to find the price of that cabbage, you find that there are several I200's (as there should be – one for each Provider and each at a different price) and you have no information to tell you which Provider's I200 you have used in the soup.

So, a solution would be to remove the "Ingredients\_Providers" table and put the ingredient\_price in the Ingredients table. This would mean there would be several rows containing the name cabbage and each would have its own ID and price.

You may be wondering that there might be too much repeating information if there's lots of things to record about all cabbages in general (such as their storage location or shelf life). I think this could be sold with another table called "Ingredient\_Type" (One Type for many Ingredients, with the Ingredients table having the field "product\_type" as a foreign key). I think it would be good idea, but not sure if you need this though for your project.

For this structure to work in the workplace it would require the employees or Gavin to carefully record which batch of cabbages is going into the finished "Product" being manufactured. I think that would be normal business practice – something already done. Or perhaps that is one of the things Gavin is having trouble with?

Actually, that brings another thought to mind – there should be many "Cabbage Soup" finished products – or batches of soup. Each batch having a different cost of goods sold (that is, connected with a different collection of ingredient prices). So, there should be a table between "Products" and "Product\_Ingredients" called "Batch". Batch should contain the date of production and a batch number and a foreign key of "product\_id". Products\_Ingredients should have "batchID" as a foreign key or composite key. So, one product has many batches, and one batch has many product\_ingredients.

And given that the price of the cabbage from Provider Farmer Joe will change over time, how to store that price for old batches so that it isn't lost? I see that you have an Ingredient\_Cost\_History table. This could be connected up with the Batch table (or the batch table and the Ingredients\_Cost\_History table could be combined into a single table.

Another idea is to not require an Ingredients\_Cost\_History table all but setting up a programming rule for your database. That is, when a Provider changes their price, it leads to the generation of a new ingredient\_id. So, if Farmer Joe with an ID of P500 supplies you cabbage through several seasons, the "Ingredients" table would contain several rows with "cabbage" and P500 each with a price, and a date, and a unique ingredient\_id. The advantage – you can eliminate the Ingredient\_Cost\_History table. The disadvantage: an ever growing "Ingredients " table. I'm having trouble thinking it through fully, but I think this would be a better solution. The database can have programming within it that is triggered when Gavin enters a new price for a Provider's ingredient, leading to the creation of a new "Ingredients " record.

The quantity.

Nowhere is quantity of ingredients recorded so cost cannot be figured out.

One idea is a recipe table (One Recipe, many Products, with Product table having a Recipe\_ID as a foreign key). A Recipe would contain a "Recipe Name" (and this may eliminate the need for a "Product Name" in the "Products" table – although it might be good to have a Name for both because it would give Gavin a chance the option of having one product called "Christmas Cabbage Soup" and another called "Special Cabbage Soup" which both of the same Recipe). The Recipe table would then several field ingredient1, qty1, ingredient2, qty2, etc. Should the ingredient1 values be names of the ingredients or ID's. I would have to see this drawn up as an ER diagram to get a clearer idea, but I think it should be an ingredient "name" (and as discussed below, I'm thinking too that there should be validation so that the ingredient Name must exist in the "Ingredients" table).

And now, without seeing what we have as an ER diagram, I think that if you were to go with what's suggested above, there would be one too many tables with Batches, Products, and Recipe. Perhaps recipe data could be stored in the products table, or the products data in the bathes table.

Another thought that adds more complexity:

A Cabbage Soup batch/product would probably contain cabbage from several suppliers. For example, 80% comes from Farmer Joe with a price of $1000 per tonne, while the rest was sourced from Farmer Beth at $1100 per tonne. This would indeed be the case. So how to cater for it?

Perhaps an IngredientQty% column in the "Products\_Ingredients" table, and some programming that validates for 100% quantity. The code would need to use the ingredientID to look into the "Ingredients" table to find the ingredient "name" (so that the qty for a "name" adds up to 100%).

Given what I was thinking above about Recipes, and now quantity%, I have described a situation where the ingredient name must be very carefully entered because misspellings (such as cabbage and cabbege) would throw everything out. So, perhaps it would be a good idea to create another table that maintains those names – a table called IngredientName that contains only a single column called "Ingredient\_Name" that is the primary key (and therefore must be unique). It wouldn't change often – only when Gavin decides to rework his product line, and he would have to be careful about what he enters. When initially creating the database, values would be inserted into the Ingredient\_Name table, and then Recipe and Ingredients would be validated against it.

9One ingredient might have 1 or many providers.

Providers could have 1 or many ingredients.

So, between ingredient table and providers table has ingredient\_providers table. Ingredient\_providers table has composite key which is ingredient ID and provider ID.

Ingredient table and product table has a relationship which is many \_to \_many. So, It will have product\_ingredient table which is between ingredient table and product table.

About time line of ingredient, I will create a table which is ingredient\_cost\_ history. This table will store ingredient’s price monthly. At finished month, it will take average price of ingredient.

For example, I have pumpkin’s cost history in May.

1st May pumpkins are bought with 85 cent per kg

15th May pumpkins are bought with $1 per kg

27th May it is bought with 90 cent per kg

When May is finished, pumpkin’s cost history will take average = (0.85+1+0.9)/3= 0.92. this means pumpkin’s cost history in May will save 92 cent per kg.