Reasoning for code

Specials ← getSpecialForRecipe($recipeId, $zipCode)

Contract

Given a recipeId (corresponding to that in the database) + a zip code,
returns the specials that correspond to the ingredients of the recipe.

Class: Special_Model

Invariant: Database is in a consistent state [i.e data in db is valid]

Why? ↔
- Special_Model class abstracts away the process of communicating and answering the results with mysql database. As such, controllers use Abstract model (in particular) Special.php (ontroller and use it to get all the information.
- If the invariant fails or is not met, then pretty much the whole class operation fails

How is the invariant always kept satisfied?

All the functions in the Special_Model class do NOT modify the database since they use "select" query statements. However this relies on the following assumptions:

- The library connecting to mysql database does not corrupt the database on "select" queries.
- The database itself does not corrupt itself on "select" statements.
- ☆ All the arguments in the function calls are "prepped" or "cast" to have no SQL injection statements that can cause data corruption.

Let's come back to the function.

We need to reason that given a recipe Id and zipcode we will get all specials that match ~~the~~ one or more ingredients of the recipe.

Moreover when no zip code is given, it returns all the specials and if it is given it returns ONLY those specials that are in stores at the given zipcode and also returns the store information.

Proof statement: Given data in the database, function will return the correct number of specials that match with the ingredients + the zipcode

Proof by cases

a) when zip code is NOT null

Induction on the number of specials match ingredients of a recipe with recipeId as recipe's id.

Base case: No specials match

The join of the tables occurs in the following way

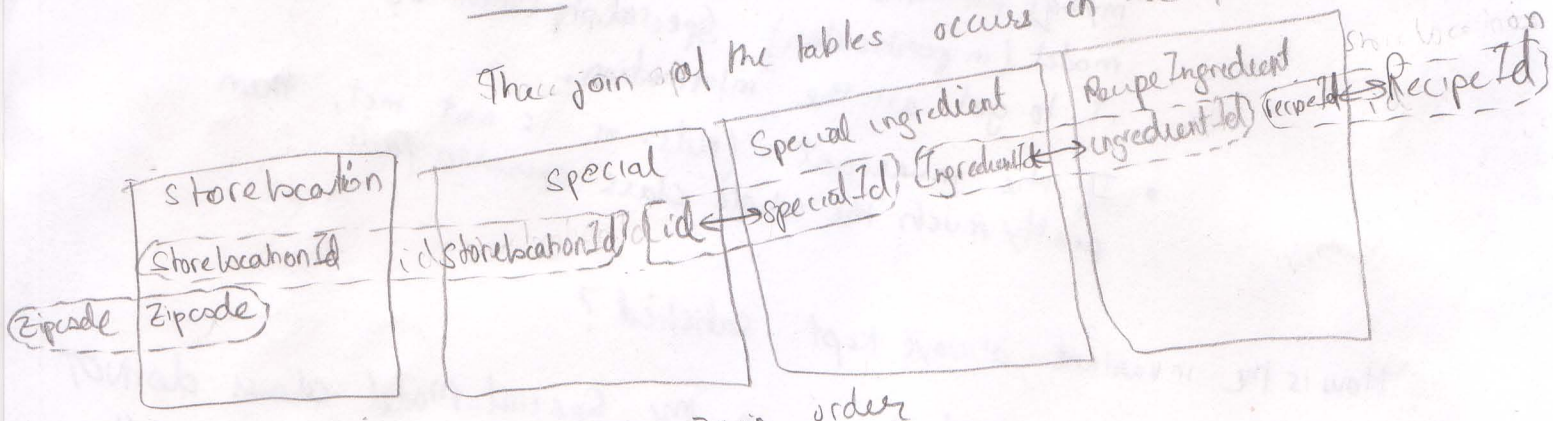

Figure 1. Join order

Hence each row returned by the db will have the above matched. Now since we know no special match i.e. IngredientId ≠ SpecialId on special ingredient will be NULL & no specials will be returned

Induchue hypothesis: let's say for any k, where $k \in \mathbb{Z}^+ \cup \{0\}$

a recipeId matches k ingredients

Induchue step: given k specials that match with the ingredients, plus one more special with the same name as ingredient

To prove: The function returns $k+1$ matches

→ Already k distinct matches in the table

Since $\text{Str special}_{k+1} = \text{Str ingredient}_{k+1}$ & $\text{zipcode}_{k+1} = \text{zipcode of storeId}$

Also (zipcode, Id_j) tuple exists

This assumption → $\{\Rightarrow$ (ingredient Id_{k+1}, special Id_{k+1}) exists in Special Ingredient

NEW TUPLE

As such a new row will be included in the join

looking like (

Storelocation	Special	SpecialIngredient	RecipeIngredient
zipcode/Id_j	storelocationId_j	(SpecialId$_{k+1}$, IngredientId$_{k+1}$)	IngredientId$_{k+1}$ → recipeId

Moreover (SpecialId$_{k+1}$, IngredientId)$_{k+1}$ exists

$\Rightarrow$ id_{k+1} in Special table exists

because of Foreign key constraints b/w SpecialIngredients special table on specialId & id

$\Rightarrow$ storelocation Id exists (in table special)

$\Rightarrow$ $\exists$ storelocation Id in storelocation exists

FK contraint

$\Rightarrow$ zipcode exists and matches

b/c of initial assumption that

$\exists s \in$ storelocation, that has the special

$\Rightarrow$ This row is matched!

$\Rightarrow$ row ~~is~~ existing k results since one more special assumption

$\Rightarrow$ $k+1$ results are returned

Moreover the proof for the case of when zip code is null is handled with the ~~case~~ of exception that store location & special tables are not chosen.

Again invariant stays the same assuming episode AND recipe7d DONOT contain any "weird" sql statements themselves. (

The MAIN underlying assumption needed still to be proven is the parser or database input ther script correctly inserts data in the db.

Most of our other code is even simpler than mine since it involves simple for loops, looping through results.