Programming Project, Database Technology

Krusty Kookies AB

John Taylor Fairbank - int13jf4@student.lu.se
Computer Science - University of Illinois

7/4/14

To run the program, go to https://github.com/jtfairbank/KrustyKookies and checkout the README, specifically the setup section.

Introduction

Krusty Kookies AB has contracted me to design a system to help them track their cookie production and delivery processes. It will integrate with existing manual systems, such as those for ordering raw materials and building loading orders. A database must be designed and a pilot program implemented. The pilot program will implement functionality for the cookie production process, including checking cookie pallets in / out of storage, blocking cookie pallets, and searching for pallets by a number of criteria.

For the project specifications, see the README's Specifications section.

Requirements

All requirements should have been fulfilled- a full database has been designed and the pilot implemented. The only note in this regard is that the pilot program uses a shortcut (hack) to determine if a pallet of cookies has been checked out / delivered. Specifically, instead of using the relationship between Loading Order Items and Pallets to do the check, an extra field *checked_out_on* is used to mark when a pallet is checked out of storage. This was done to save time and prevent feature creep in the pilot. I actually implemented the 'correct' sql code to do the check (you can see it commented out in the PalletController), but did not implement any functionality to link Pallets and Loading Order Items as the pilot doesn't have any concept of Loading Orders in its requested functionality.

System Outline

The code is written to be very self-explainable. The following high level outline describes the main components of the system and where they are located. This codebase is based on a typical setup for my many other web-dev projects, so if something isn't clear please let me know and I can fix it on all of them. :)

1. MySQL is used to store data in a meaningful context. Setting up the MySQL db is described in the README's setup section.

2. PHP is used to expose application functionality, respond to front-end requests,
    a. The PHP code is broken up into two main sections: Controllers are responsible for exposing CRUD-like functionality that interact with the database. Models are representations of database items in PHP. Note that the mapping is not necessarily 1-1: there is a Recipe and RecipeIngredient model (and tables in the DB), but only one RecipeController that interacts with both models.
        i. All application functionality is located in */build/dynamic/api/*.
        ii. PHP's PDO is used to safely perform MySQL operations.
    b. The PHP templating library Twig is used to generate the front-end pages. Templates are located in */build/dynamic/template/*.

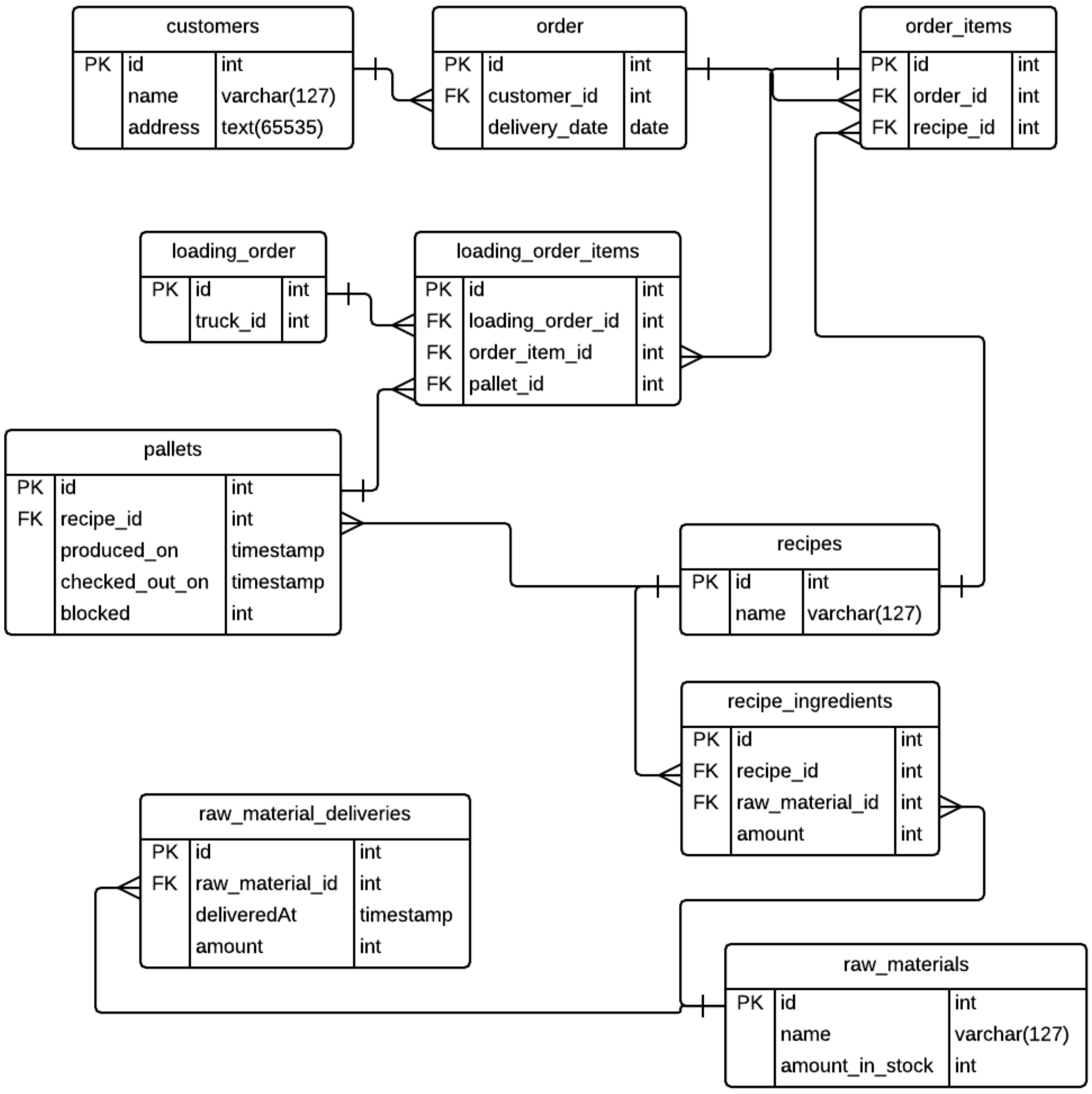
3. HTML and related front-end web technologies expose forms to the user so they can interact with the application. A traditional multi-page request-response model is used, as the front-end functionality was sufficiently simple to not require complicated javascript aided interactions.
    a. Front end files are accessed via the .php files in */build/*
    b. Front end resources can be found in */build/static/*.

4. Any third party code is included from */build/lib/*.

5. You can safely ignore everything in */src/* and */test/*, those are used for development purposes only.

E / R Diagram

I used LucidChart.com to generate this diagram but couldn't add annotations. It clearly shows the relationships between tables; the reason for those relations should be fairly self-explanatory. The *order_items*, *loading_order_items*, and *recipe_ingredients* tables all facilitate a many-to-many relationship.



Relations

In practice I find the best PKs are ids. Thus every table uses an *int(10)* id field as the PK. That is the first entry in every relation.

italicized entries are foreign keys

* => secondary key

- **Customers(**id, name*, address***)**
- **Order(**id, *customer_id*, delivery_date**)**
- **OrderItems(**id, *order_id*, *recipe_id***)**
- **Recipes(**id, name***)**
- **RecipeIngredients(**id, *recipe_id*, *raw_material_id*, amount**)**
- **RawMaterials(**id, name*, amount_in_stock**)**
- **RawMaterialDeliveries(**id, *raw_material_id*, deliveredAt, amount**)**
- **Pallets(**id, *recipe_id*, produced_on*, checked_out_on, blocked**)**
    - produced_on is a secondary key since each Pallet must be scanned into the freezer 1 at a time. Thus it will always be unique.
- **LoadingOrder(**id, *loading_order_id*, *order_item_id, pallet_id***)**
    - A weak entity that relates a pallet to the order item it fulfills, and both to the loading order that is going to fulfill the order.
- **LoadingOrderItems(**id, truck_id***)**
    - Note that truck_id is included in our system for demo purposes but relates to an external one.

SQL Statements

See https://github.com/jtfairbank/KrustyKookies/blob/master/setup/DB_setup.sql.

Final Notes

I'm actually working on two projects very similar to this (same technologies), but with more complex functionality. I'm quite excited to apply my knowledge gained during this course and final project to these. If this report seems a bit rushed, that's because it is. I worked really hard on the codebase to make it easy to read / understand, fast, etc. But all of that work is directly applicable to my real-world projects. Some examples of this in action:

1. I was able to test my project-setup scripts and adjust them as I worked on the project. These include things like automatically syntax-checking the php / javascript source code, and compiling various files into the output */build/* directory. I can now apply the same setup to my other projects, standardizing the directory structure, automation of tasks like

testing, etc.

2. I really tried to write efficient controllers. And it turns out that means having a lot of joins, so that I only select data once (rather than multiple SQL statements to create something like a Recipe model). Doing that in this project taught me what adjustments to make to my Models and Controllers, and how to think through the issues I faced. I look forward to refactoring my other projects to use this method.
    a. The LoadingOrderController, and LoadingOrder and LoadingOrderItem models, showcase this.
    b. A note: I realize I could do it even better by using views that tie together all the dependencies of an object, then just mass create the object with all its dependencies. I plan on doing that in my real-world projects but figured that efficient sql selectors per-model were good enough in this case.
    c. Same thing for stored procedures instead of using PHP's PDO to make sql queries directly.

Anyways, sorry this was a bit late and if the edges are a tad rough. I'm just a tad too excited to use what I've learned. ;)