Site Database

Cameron Norton

These are my thoughts on a database for the project. I don't know anything about ER diagrams or the like, so I will just list what I'm thinking of. I want the database to basically implement the minimum that we can reasonably get away with, for our own sanity's sake. The project spec also wants us to create a logging table in the database, but I don't really want to think about that right now, so it's not here.

There are issues regarding how we are going to store the database. When the Docker images start running, the database files are dropped in the mysql directory. These files are the same as if the database were running on your machine *i.e.* they are owned by an administrator account and you don't have permission to screw with them. Currently, all files in the mysql directory are .gitignore'd and it would be unwise to change that.

So how do we provide a database if we can't put it in the git repository? Maybe we could provide a SQL script that creates the database alongside some example data, then we can run the script from the phpMyAdmin page when the site is first set up. A SQL script is definitely safe to version control.

Also, we need to figure out how to actually interact with the database from the site pages. Technically, the Apache server and the MariaDB database are on seperate Docker virtual machines, and I'm not sure how to make that work. I know that it is possible to communicate between the images, but we need to learn how to actually do it.

But anyway, here's the schema that I came up with:

# Accounts

This table represents the data required to establish and identify an account. We could also maybe store registration date but I'm not sure what we'd do with that information.

| field | type | info |
| --- | --- | --- |
| user\_id | int | primary key, auto increment |
| username | varchar(25) | unique |
| password | char(128) | sha512 hash, could be binary? |

# Users

This table represents data about the people associated with an account. User data should be stored separately from account data because there can be accounts that are not associated with a real-life person (like the root account). user\_id is the primary key.

| field | type | info |
| --- | --- | --- |
| user\_id | int | foreign key pointing to Accounts.user\_id |
| first\_name | varchar(25) |  |
| last\_name | varchar(25) |  |
| email | varchar(50) | unique |

# Items

This table represents store items. We should be able to dynamically fetch the name, description, image, and price of items to build the store page. Let's ignore silly things like stock and availability because that's complicated and we don't want that.

| field | type | info |
| --- | --- | --- |
| item\_id | int | primary key, auto increment |
| name | varchar(25) |  |
| short\_desc | varchar(100) |  |
| image | varchar(50) | stores picture filename |
| price | decimal(5,2) | up to $999.99 |

# Events

This represents events. Like above, we should be able to build the events page from information stored in this table alongside the attendance table.

| field | type | info |
| --- | --- | --- |
| event\_id | int | primary key, auto increment |
| name | varchar(50) |  |
| short\_desc | varchar(100) |  |
| start\_time | datetime |  |
| end\_time | datetime |  |

# Attendance

This table represents important attendees to an event. The big features should go here. I guess these people have to have an account too. Both fields together are the primary key.

| field | type | info |
| --- | --- | --- |
| user\_id | int | foreign key pointing to Accounts.user\_id |
| event\_id | int | foreign key pointing to Events.event\_id |