FEBRUARY 17, 2014

ASSIGNMENT 2 REPORT

IAT 352

CASSANDRA DE GIT
301166220
cdegit@sfu.ca

Project Overview

{ Tutor } is a tool to help connect people who want to learn programming and those who want to teach it. Contributors post Lessons; blog post style content designed to teach some programming Topic. Learners are able to view these Lessons, and follow Contributors or Topics. By grouping content into specific Topics that users can search, { Tutor } makes it easier for Learners to find content they are interested in.

Unlike the suggested project, most of { Tutor }'s focus is on the posts themselves rather than the users creating them, as well as the organizing system for the posts, called Topics. This shaped a great deal of the functionality and design of the system.

Database Design

In the early stages of designing the database, differentiating between the two different types of users, Contributors and Learners, was a big focus. The original database design came from looking at how these two user types would interact with the system. The original ER diagram is shown in Figure 1. A user class would contain the data shared between both Learners and Contributors, their username, email, and password. Subclasses were defined for Learners and Contributors; Contributors had attributes for their bio, as well as their Facebook, Twitter and Flicker accounts. Contributors were also able to write posts, while Learners were not. Learners didn't have any attributes associated with them, but could follow Contributors and Topics. Posts had id, author, title, and content as attributes, and could be given multiple topics.

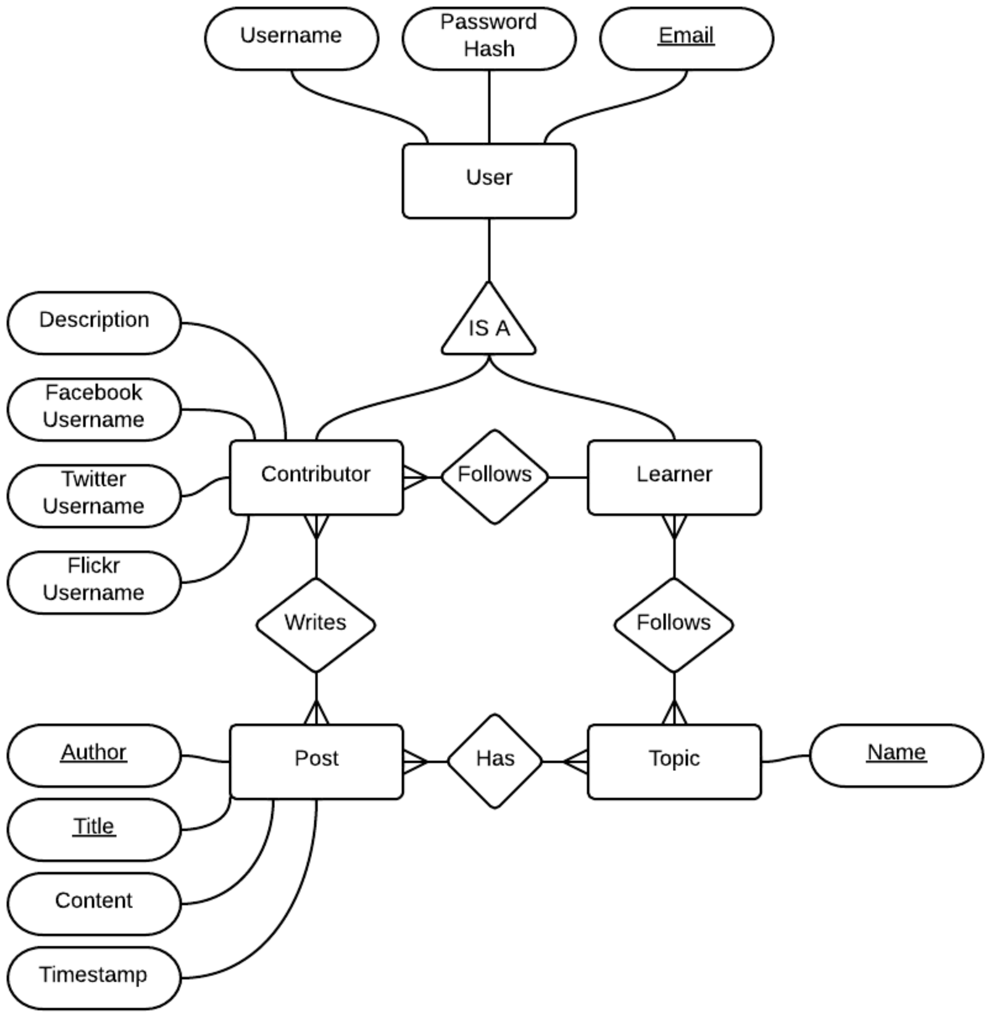

Figure 1: Initial ER Diagram

From this, creating the initial design of the database was fairly straightforward. This can be seen in Figure 2. The users table would contain the data from the Users superclass, while separate tables for learners and contributors would fill in the rest of the necessary data. Email would act as the primary key in users and the foreign key in contributors and learners, allowing the records to be easily combined. The posts table is also quite a simple translation from the original design. By having the posts table contain an author attribute, it is easy to determine which posts were written by each user without having to store that data with the user themselves.

The relationship between posts and topics is slightly more complicated. Originally, I considered simply having a list of topics as an attribute of a post. However, this would have violated First Normal Form, and would have made the database more difficult to use. With some advice from Srecko, I decided on using a new table to contain the information about this relationship. Each row of the post_topics table contains a unique id, a post id and a topic name. To give a post a particular topic, you simply need to create an entry in this table that contains that post's id as the postId, and the desired topic's name as topicName. To add another topic to that post, another row is added. This is a simpler solution that allows topics and posts to be quickly and easily pared.

An identical approach was used to capture the Learner's ability to follow users and topics.

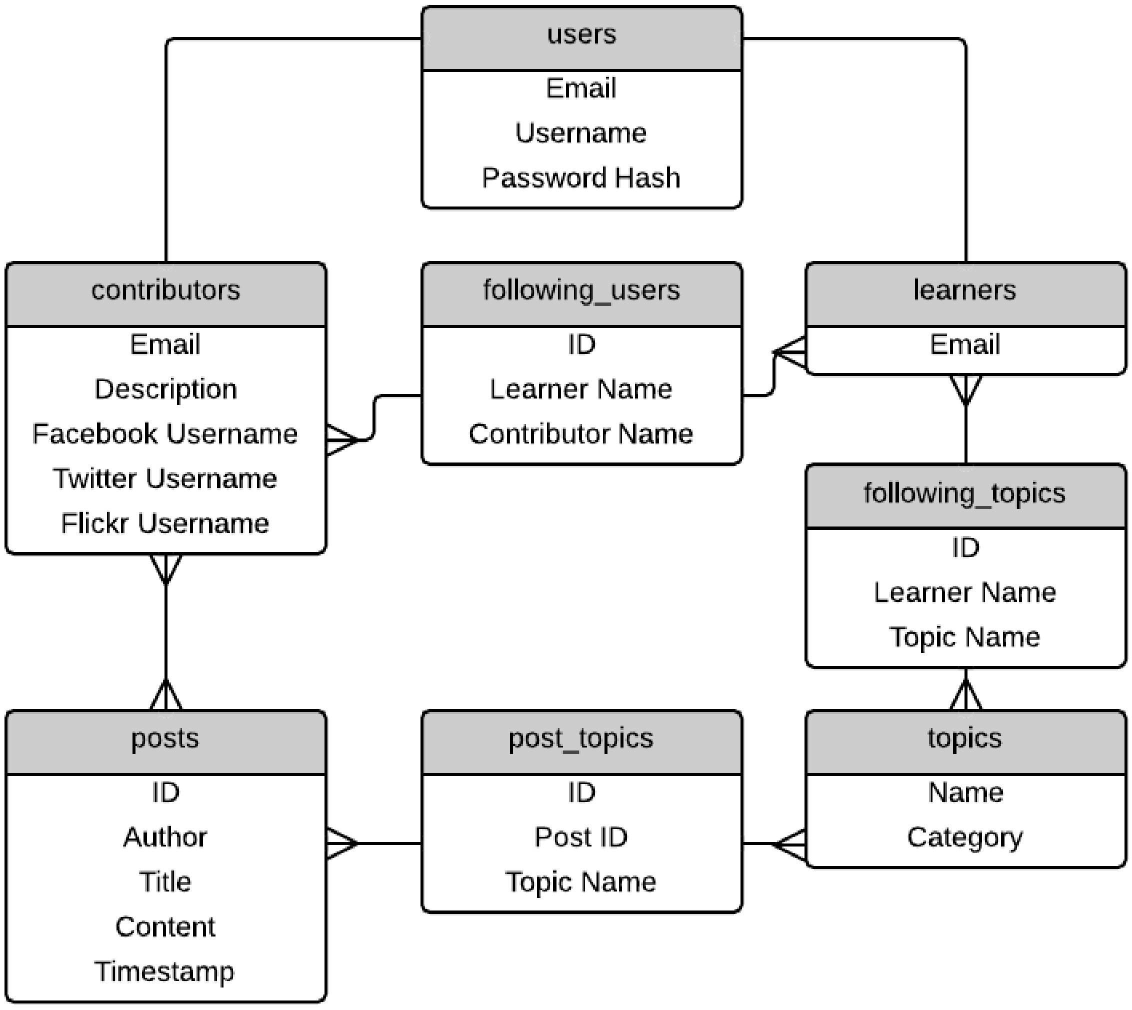

Figure 2: Initial Database Design

Upon beginning to implement the database, it became clear that having the separate tables for the user types would be more work than it was worth, given that learners actually had no unique data to be stored. The two user types were consolidated into one table called users. This is less efficient for storage, as all Learners have attributes that they don't actually need (description, Facebook, Twitter and Flicker), but is easier to use.

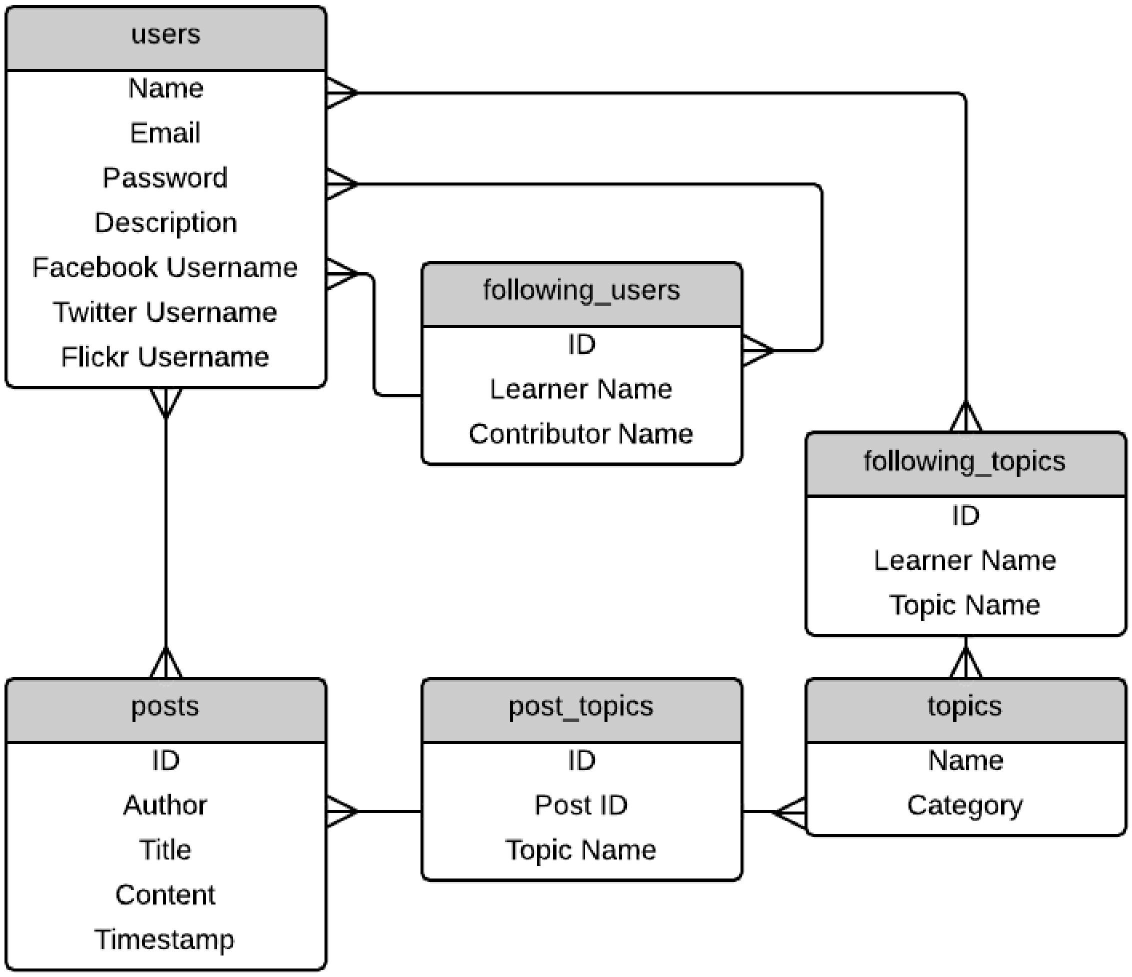

Figure 3: Revised Database Design with consolidated users table

Implementation

I used a simple controller architecture for the majority of the website. Depending on the contents of $_GET, the controller (controller.php) would call various functions in other files to achieve the requested result. The 'action' key in $_GET would determine which function to call, and other values in $GET were typically other data needed for that particular function. For example, controller.php?action=user&name=cdegit would display the user profile for the user named 'cdegit'. This architecture made it incredibly easy to add more functionality. It also allowed me to open up a single connection to the database for most pages, and simply pass the connection to the functions that needed it.

Not all functions were able to use the controller architecture, however. For form submissions, I generally used POST requests to avoid exposing data, making them unsuitable for this approach. These scripts were written as separate files, all with the prefix "process".

I wanted Topics for posts to be "moderated"; only certain, preset topics would be allowed, and users would not be able to create their own topics. While restrictive, I felt this was important for the information design integrity of the site. The current navigation system would become bogged down, and would be susceptible to having a large number of redundant topics. Instead, users would ideally request that a new topic be added, and this topic would be added by a moderator if approved. While the site doesn't currently have "moderator" type users, new topics and categories can be easily added to the database and then used.

One issue I ran into frequently was needing to get somewhat redundant data from the database within a single page. For example, due to the way in which I store topics and their categories, it was easiest to make a query to get all the categories, then for each category, write a query to get all of the topics within it. While not efficient, the queries are easier to understand and require less processing in PHP then trying to extract that data from a single query. This issue occurs in multiple places in the system; each time I opted for multiple queries for the increased ease and readability.

Future Changes

There are many changes that will need to be made in future versions. The most significant is having registration and log in occurring over HTTPS. Unfortunately, I was unable to get HTTPS to work on my computer, possibly due to issues with creating my own certificate.

Another major change is adding pagination. Currently, most pages will just display all posts or users at the same time. With a large number of posts or users, this quickly becomes next to impossible to navigate. For the next iteration, I'll implement simple pagination that will limit results to 12 per page, and allow users to navigate to other pages.

The last change doesn't affection functionality, but would require significant changes to the code, which is why I was unable to do it for this version. In the code, I refer to Posts both as Posts and Lessons, which has caused quite a few issues with my naming conventions. This will be fixed in future iterations to have much more readable code.