Group 12 Report

Schema Design

The original schema design was created from looking at the APIProvider. We looked at each of the methods and took the parameters of what needed to be created. This enabled us to have an initial schema.

We also created test data which we inserted into the tables so we could check whether the methods work when running the server. Then we also input some data by the web interface to see whether it is in the right place in the database.

We decided to create two join tables for the topic likes and post likes. This linked our topic and person tables, so one person could like a topic/post. This was so we were able to keep count of the number of likes that we needed to for likePost and likeTopic methods. Although this was updated later on when we were doing likePost and likeTopic.

Throughout the process of completing the methods, we needed to change the schema on a regular basis. A main change we made was adding in a function to display when the post/topic was created. Within the ‘post’ and ‘topic’ tables we used the ‘TIMESTAMP’ function to convert an expression to the date-time value. This gave yyyy-mm-dd hh:mm:ss.

Another issue we had to update was how we kept track of likes. Within ‘topicLike’ and ‘postLike’ methods there is a ‘like’ which is assigned to boolean. This is because if a topic/post is liked then this can either be ‘liked’ (1) or ‘not liked’ (0), making it easily identifiable. Interestingly, in the method ‘getLikers’ a LEFT JOIN is used to gather both the liked and unliked topics.

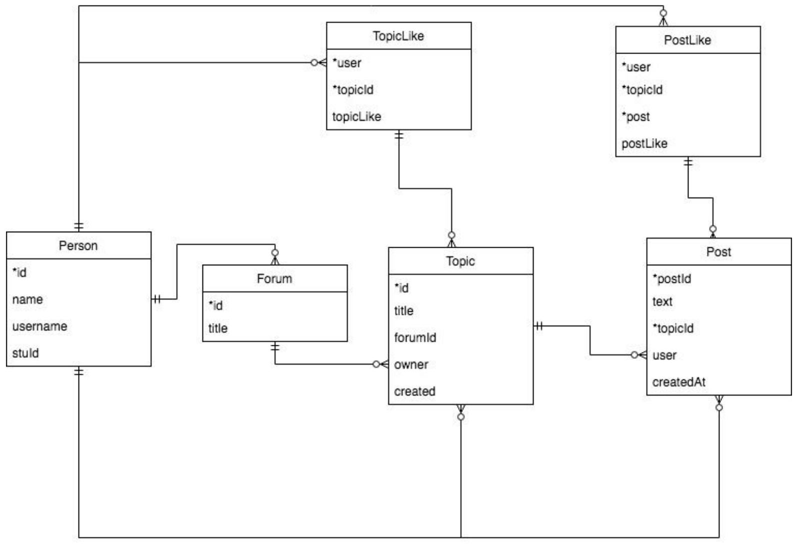
When designing the schema, we tried to reduce the amount of duplication so we would not break third NF. An example of this can be seen in ‘Post’, where ‘post’ contains ‘postId’ and ‘topicId’ but not ‘forumId’. This is because we are able to use a JOIN to link ‘Post’ to the ‘Forum’ via the ‘Topic’ table.

Another issue that we ran into was with ‘PostLike’. Here we needed to include ‘topicId’ and ‘postId’ because otherwise we would not be able to identify which topic each post like belonged to. Within ‘PostLike’ we need three foreign keys because otherwise we would not know which post is liked if just topicId and userId was used. We discussed doing this another way, which would involve having two candidate keys within ‘Post’ which would not be ideal.

One inefficiency we initially included was a text field in the Topic table, which duplicated the text used in the Post table. The only time this was required was during creation of a new post, after which we would copy the text field from the new topic into the new post when calling createPost from within createTopic. Instead we realised we could just call createPost using the “String text” parameter from the createPost method. Unfortunately we re-added the field as we ran out of time to test on the lab machine and weren’t prepared to risk breaking the database.

After completing all the methods necessary throughout sections A and B, the schema was checked and agreed upon (Figure. 1.). We used ant version 1.10.3 to create and test our methods, all of which work correctly upon hand in.

The schema is definitely 1NF as it doesn’t have collection-valued attributes, every cell only has one single value. It is also the 2NF because it has no partial FD from candidate keys to non-key attributes. It is a 3NF as well because there is not transitive FD between non-key attributes. But it is not the BCNF as every non-trivial FD is not a dependency on a superkey, for example, in table Person, the non-trivial FD of stuId->name does not depend on the superkey id.



Generally, the schema is well-structured after we proceeding through the task 2. And we adapted the schema into the lab machine to make sure everything works right and no error prompts out.

Fig. 1. Crows feet notation of created schema