DB CW2 TASK 1: Group 10 Report

Database Schema:

Prior to the creation of any database tables a schema using crows foot notation was designed, to ensure that construction of the database was optimised. Subsequently, to populate the database a create.sql script was produced. It was decided to have a separate table for each type of data structure. Therefore, the tables created are: Person, Forum, Topic, Post, LikeTopic, LikePost. The tables act in a trickle-down fashion meaning a Forum has a Topic and a Topic has a Post. Even though each table has a candidate key (see normalization table) that could be used as a primary key, an auto incrementing ID value was implemented acting as the primary key, to ensure that even if data is updated, the reference to that data remains constant. This helped to keep the schema consistent, especially when using foreign keys between tables. Throughout the schema it has been assumed that titles for topics and posts are not unique. Below are detailed explanations for each table.

Person:

This table holds a list of users. A user on the site is required to sign up and this information will be stored within the person table. Each row of this table relates to an individual user, storing id, username, name, and an optional student id dependent on whether the user is a student or a member of staff. Other tables refer to this table if information is needed about an individual. A person has a unique username.

Forum:

This table holds a list of all the forums within the database with their corresponding title. This title has to exists, i.e. not null and unique. A forum is allowed to exist without any topics within. Each record within this table is unique as the primary key is an auto incrementing integer and the title has to be unique. In the designed schema a forum does not need to be created by a signed in user and as such can be created holding no content. But to populate a forum with topics a person must be signed it. A forum holds only a reference itself in the form of an id and title.

Topic:

This table holds a list of all topics within a forum. When a topic is created it needs a title, the id of the creator/author and the id of the forum it belongs to. It was decided that an author is a person and hence a foreign key was implemented referring to the person table. The same stands for the forum ID, which refers to the forum table. However, if a person was deleted from the database the table would still keep track of the author ID and therefore some logic could be implemented to cope with this. Although deletion is not implemented in this project as it currently stands it is a logical progression and thus should be considered.

Part 2 B.2 required alteration of how we store data within our database. It required an extension of the topic table to include information of the time of creation of a topic. This was a simple fix achieved through the addition of the new column postedAt which was of type Timestamp and was automatically filled upon successful creation of a topic.

Post:

This table holds the detailed information for a particular post within a topic. A post needs a title, an author, it's content, the relating topic ID. Multiple topics can have the same title, but the ID is auto incrementing and therefore at no point two identical records can be created. Additionally, a timestamp is auto generated keeping track of the date and time the post was created. A post cannot be created without any content and therefore content cannot be null. Again, the author references a person and therefore a foreign key was implemented. This also holds true for the topic. Similar to topic there is an associated timestamp for each post allowing for easy tracking of the order of post updates and the most recent post within a topic.

LikeTopic:

This table keeps track of the likes for a particular post. A person can only like a topic once and therefore the person ID is a column referring to the person table. However, a topic can be liked by multiple, different people and therefore the table can have multiple records with the same topic ID but the person ID must not be the same. Therefore, the candidate key is a combination of the topic ID and the person ID.

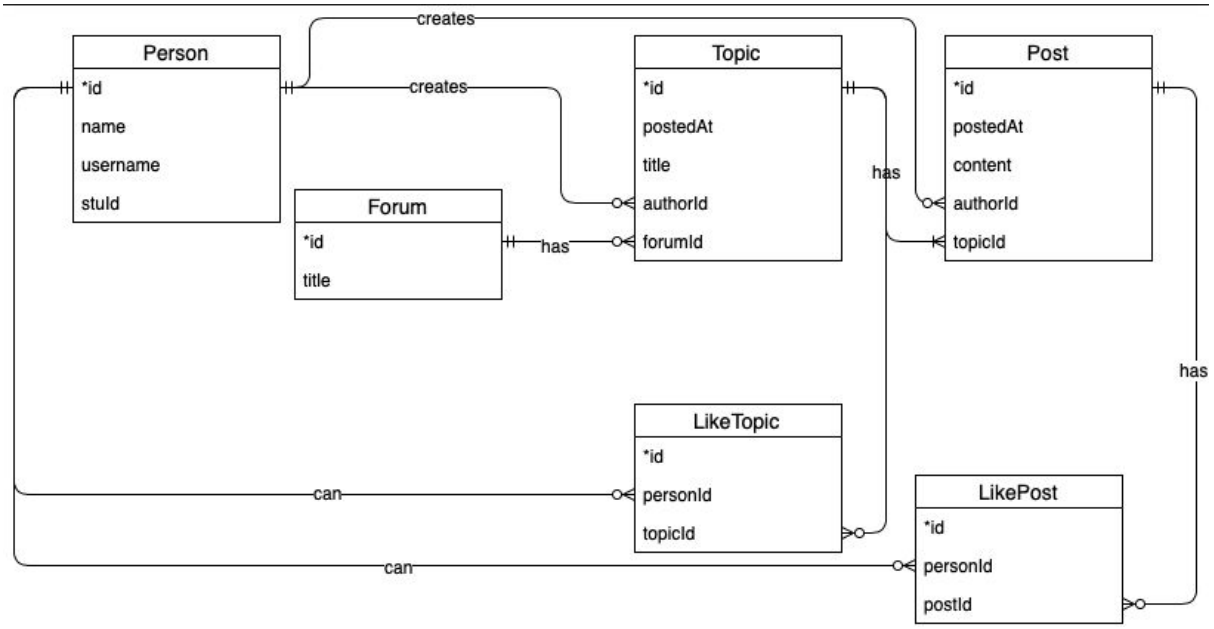
LikePost:

This table keeps track of which person liked which post. It therefore needs an author, referring to the person table and a post ID referring to the post table. The purpose of this table is to know whether a user has like a certain table or not. A person can like multiple posts and therefore a separate table was created to avoid repetition in the other tables. It was chosen to normalize the database as much as possible.

LikeTopic and LikePost are separate tables even though they contain similar information. It was considered to initially combine them in one single table, however because LikePost needed a foreign key constraint to the Post table and LikeTopic to the Topic table, a generic 'parentLikedID' field was not possible.

When testing the application a data.sql script was created to fill the database with dummy data. Later on, the database was cleared and the web app tested by manually adding forums, users, topics and posts.

Modelling



There is exactly one to many relationship between person and post. As a user is allowed to create multiple posts, but each individual post can only be created by a single user and for a topic to exist a post must have been created. This is the same argument for the relationship seen between person and topic. A forum is only directly associated with the topics held within it as it can be created by someone not logged into a user account. A forum can be empty or hold one or more topics but for a topic to exist it must be found in exactly one forum.
A person is also able to like topics and posts, this is separate from the other relationship as it is not a creation relationship. Rather a person can like any ones post or topic and as such it is a many to many relationship where a topic or post can be like by no-one or many people. This connection is mediated by joining tables which hold a reference to the person who liked the topic or post and a reference to the topic or post. If a post or topic is unliked, rather than being stored in a table the information is removed from the database.

The explanation for the normalization procedure is described in the table below. It should be noted that all tables are in BCNF form. The reason for using a database schema with separate tables for each section keeps residing information together. However, because of this some SQL queries resulted in joining several tables together. On one hand this increased the difficulty when writing the SQL queries in the API class. On the other hand, this may enable easier API expansion as the schema is clear and easy to understand. It would have been possible to use multiple smaller SQL queries to pull out the information from each table separately, however it was decided against this to avoid an unnecessary number of database requests.

Normalization table

TABLE	CANDIDATE KEYS	KEY ATTRIBUTES	NON-KEY ATTRIBUTES	1NF	2NF	3NF	BCNF
				Each table cell should contain a single value. *Each record needs to be unique.*	*Rule 1- Be in 1NF* *Rule 2- Single Column Primary Key (SCPK)* *no {all keys but not superkey}->{a non-key}*	*Rule 1- Be in 2NF* *Rule 2- Has no transitive functional dependencies* *no {contains non-key} -> {non-key} where LHS is not a superkey*	*no {!candidate key} -> {key}*
Person	{id} {username}	id, username	name, stuId	Y - unique id and username ensures each record is unique	Y - both id and username are SCPKs	Y - With {id} and {username} as candidate keys, any dependency that contains it becomes a superkey. As name and stuid are non unique there are no dependencies where a non-key depends on another.	Y - All dependencies contain a candidate key
Forum	{id} {title}	id, title		Y - unique id and title ensures each record is unique	Y - both id and title are SCPKs	Y - there are no non-key attributes	Y - All dependencies contain a candidate key
Topic	{id}	id	potedAt, title, authorId, forumId	Y - unique id ensures each record is unique	Y - id is SCPK	Y - We have assumed that postedAt is not a Candidate key because it is complex and, although unlikely, could be non-unique. With {id} as the only candidate key, any dependency that contains it becomes a superkey. Therefore, it is 3NF because there are no non-key attributes or combinations of that have a functional dependency on another non-key attributes.	Y - All dependencies contain a candidate key
Post	{id}	id	potedAt, authorId, content, topicId	Y - unique id ensures each record is unique	Y - id is SCPK	Y - same as above	Y - All dependencies contain a candidate key
LikeTopic	{id} {topicId, personId}	id, topicId, personId		Y - unique id ensures each record is unique	Y - id is SCPK	Y - there are no non-key attributes	Y - All dependencies contain a candidate key
LikePost	{id} {postId, personId}	id, postId, personId		Y - unique id ensures each record is unique	Y - id is SCPK	Y - there are no non-key attributes	Y - All dependencies contain a candidate key