• Which tasks were particularly easy/hard or interesting/boring?

Once understanding the use of PrepareStatments and placeholders, the first few methods were quite rhythmic to go through. It was a case of working out what the method needed to return, and how to create those objects using the data in the tables. The checks for each method took a long time and simply required checking each input, but it was necessary to create a robust piece of code.

Things started to get more interesting at the getForums method, whereby the forums had to be ordered according to the latest post. In the schema, we had designed it so that only the post contained the data relevant to the time it was posted. There were issues with the query as the latest topic that was posted to was not showing. We then created a join table where the MAX timePosted was selected from the Post table and grouped by forum id. This table was joined on the timePosted of the outer table, which used a right outer join for the Forum table and a topic join. This meant the correct topic was now showing in the table with the last post. This further consolidated my understanding of outer joins so they could be implemented correctly throughout the rest of the methods.

Another challenge was trying to understand how many separates prepare statements to use. This boiled down to how many checks we needed to make. After the queries were executed to make the relevant checks, and saved appropriate information, then a final query could be made.

Finally, it was important that there were no nested statements, which was tempting especially in section B2. Instead, complex joins had to be made to ensure all the data was retrieved and there were no queries executed within a ResultSet iteration.

LikeTopic and LikePost required us to rethink the schema. We decided to have each as its own table which contained the person and the associated topic/post.

• What did you learn during this coursework?

- If the return required a list, the ResultSet was useful for iterating through the tables.
- Using placeholders to prevent sql injections.
- When creating something new, thorough checks must be made to ensure all the input from the user is correct, and to ensure to always send the correct errors back.
- Using rollbacks to ensure the database is not filled incorrectly
- The use of commits to ensure the database was updated.
- Using outer joins to display a fuller dataset was useful when grouping columns.

• What is your opinion of JDBC now that you have worked with it for a while?

It is surprisingly straightforward to use SQL & java with this API. I like how you can directly use SQL statements inside prepare statements. I found SQL useful for retrieving relevant information, and java useful to send out the associating errors or data.

If you were designing your own database API what would you definitely do the same, or definitely do differently?

Due to my new experience, I would use JDBC again. This is because I like java and found it straightforward to retrieve the needed information from the database. I like how simple and neat the code was with prepare statements followed by the setString/setInt. I also like the result set whereby the query could be separated from the generated table to create a more readable piece of code.

I would probably try find a way to close the statement automatically, so if another one was run, or the result was returned, it would close. It may be a good idea to be a more error responsive API, that could create prepareStatement specific for error catching, which could display the associated error message.

• Anything else you would like to mention (relating to the coursework).

I will definitely be using this API again!