Coursework 2: Group Report

24978 97395

May 10, 2019

Task 1: Schema Design

Schema

The schema was arranged into `Person, Forum, Topic, Post, PostLikes` and `TopicLikes`, and the tables were kept as simple as possible so that space could be optimised. A Post belongs to a Topic, which belongs to a Forum. This arrangement meant the Post contained a foreign key to the Topic, which contained a foreign key to the Forum. This made the most sense, as a Forum has a 1-to-many relationship with Topic, and Topic has a 1-to-many relationship with Post. In this way, we could keep the forum table compact with only 2 columns. The Topic and Post also contained a foreign key to the person who created the associated post/topic. The Post contained a message and a `timePosted` variable. This was important to include as it indicated which topic was posted into most recently, which was displayed on the forum view page.

The PostLikes contained a foreign key to the person and the post. This is because there can be many likes to 1 post. 1 like is also associated to 1 person. This was likewise with the TopicLikes table. All the variables in the schema were defined as `NOT NULL` except the stuId. This could be null depending on whether the Person was a student or not. All other variables were set to not null because they were required to create a valid input.

Normalisation

The schema is in 1NF as none of the tables contain collection valued attributes. The Person table has two candidate keys, id and username. As both these candidate keys contain a single attribute, there cannot exist any non-trivial partial functional dependencies. The Forum table again contains two candidate keys but for the previously stated reason there cannot exist any non-trivial functional dependencies. The topic table contains only one candidate key, `id`. This is justified as none of the other parameters are guaranteed to be unique, i.e. two topics can exist with the same `title`, `message`, `forumId` and `personId`. This candidate key only contains a single attribute so again there can be no non-trivial partial functional dependencies. The same arguments can be used to deduce that the Post, PostLikes and TopicLikes tables all contain no non-trivial partial functional dependencies. Therefore, the schema is in 2NF.

The schema was not in 3NF as it held no transitive dependencies. For example in the `Post`, all the variables depend only on the id and nothing else. This means there cannot be any transitive dependencies as this requires a chain of dependencies. This is true for all other tables. This fact also makes the schema in BCNF. All the dependencies are dependant on the super key of the table - the id.

Furthermore, the schema was separated into separate tables so that there were no multivalued dependencies, which meant the schema was also in 4NF. I.e. the likes were not added to a Topic/Post/Person table, but created as a separate table so that each insert was unique to that Topic/Post and `personId`. If the likes were added to the Post table, there would have been repeats in the person Id if a person had liked a lot of different posts/topics.

Finally, the schema is NOT in 5NF. This format means there cannot be lossless decomposition into smaller tables. We decided to add an an auto-incremented primary key for each of the tables as these could always be a unique key. Therefore, all of the variables in the table depended on this primary key. This was done because for many of the tables, the inputs did not necessarily have to be unique.

Task 2: Method Implementation

In general, the methods used a Prepared statement to execute queries, which were sent to the API connector. These queries used placeholders to prevent SQL injections. After the query was executed, the `PrepareStatement` was

closed using the `close` method, and the Connection c was committed, using the `commit`. This allowed the database to be updated correctly.

To handle errors, all the prepare statements were placed in a try statement. This allowed capture of database errors, which returned a fatal result. Within the "create" methods, it was necessary to further investigate errors within the catch statement to ensure the database did not get filled incorrectly. Here, we used the connection rollback method inside another try statement.

getUsers

For this method the database is queried to return all the pairs of names and usernames in the database. These pairs are then iterated through and added to the hash map which is then returned.

getPersonView

This method returned a PersonView object, which required a username, name (NOT NULL) and student ID (NULL) to create an instance. For this, a query was made which selected name and stuId based on the input username. The ResultSet checked if there was a result and filled the parameters to be imputed to the PersonView instance. Else, a failure was made that the user does not exits.

addNewPerson

First, it was important to check that all the inputs were not empty, and username and name were not null. Then, an insert query was formulated which inserted each input into the appropriate columns using setString.

getSimpleForums

This method required a returned arraylist of SimpleForumSummaryView's. This object required a title and an id to create an instance. An array list was created, and all id's and names were selected from the forum table inside the prepare statement. The result set iterated through the contents of the table where the id and name were retrieved and added to the SimpleForumSumarryView array list and returned as success when the iteration was complete.

createForum

This method required and insert into the Forum table. First, the input title was checked so it was not null/empty. The first prepare statement was executed to check the title exists already, which would return a Result.failure if true. Following, an insert query was made in and the string was set to the input title. A connection rollback method was implemented inside the catch statement, so that if a database error occurred, it would not fill up the table incorrectly.

getForums

This method uses a single query that retrieves the information for all forums. A complex join was required to retrieve the topic within a forum which has had a post made in it most recently. The results are then iterated through and added to the list of ForumSummaryViews. The topicId was retrieved as a string instead of an integer. This is to deal with the case where a forum has no topics within it. If the topicId string is not null it is then converted to an integer.

getForum

This method required a return of the Forum View object. The Forum view contained a forum id, a forum title and a List of SimpleTopicSummaryView's (which require a topicId, forumId and a topic title). This was split into 2 prepare statements in order to send the correct errors relative to the appropriate table. The first statement got the forum title where the id = the input. The second got the id and title from the topic where the foreign key of the forum = input forum id. This allowed a list to be generated, by iterating through the result set and creating the SimpleTopicSummaryView arraylist. After completion the Forum View instance was set.

getSimpleTopic

This method returned a SimpleTopicView, which contained a topicid, title and arraylist of the SimplePostView object. This object contained a postNumber, author, text and time when posted. The input topicId is check using a PrepareStatement, which check an entry with that id exists. If true, and the result set has a next, the title is retrieved from the database and saved. For the next statement, a the post information is retrieved from the post table, and joined onto the person table using the Post tables foreign key. As the Posts are iterated through using a ResultSet, the relevant information is saved and inserted into the ArrayLists of SimplePostView instances. When complete the SimpleTopicView instance could be created and returned.

getLatestPost

A single query is used to get the information needed to construct the required PostView, joins are used to get the latest post by choosing the post with the largest time posted. Joins are also used to get the number of likes the post has as well as the post number.

createPost

First, this method checks that the text and username given as input are not null or empty. A query is used to check that the topic with the given ID exists and another query is used to check that a person with the given username exists. The post is then inserted into the database.

createTopic

createTopic begins by checking that the title and text arguments are not empty or null. A query is then used to retrieve the persons ID from there username. The topic is then inserted into the database. `RETURN_GENERATED_KEYS` is used to find the ID of the topic which has been inserted into the database. This key is then used when inserting the initial post of the topic into the Post table in the database.

countPostsInTopic

A count of posts is returned via an SQL query, whereby the count is selected from the Post table where the topic id is equal to the input. If there are no results in the result set, it means that there is no such topic id, which will return a failure. If not, the count will be returned successfully.

likeTopic

In order to like a topic, the schema requires an insert to be made into the TopicLikes table. For this, 2 queries are executed to check the topic id AND the username exists in the Topic and Person tables. Following, a query is set to select all the TopicLikes, to check if there are any likes associated to the input topic and person. If true and the like is set to true, an error is sent that the post has already been liked. If false and the like is set to false/dislike, then the row is deleted from the table. Finally, if there are no results with the associated topic or person AND like is set to true, an insert statement is prepared to create an entry of the input topic and person using the ids retrieved from the previous ResultSets. If the like is set to dislike, an error is sent whereby a topic cannot be disliked if it has not been liked.

likePost

The method checks that the person liking the post exists and gets the ID of this person if they exist. The existence of a topic with this ID is then checked. Then a query is done to check whether there is a database entry for this post being liked by this person. If there is and the `like` boolean is true then an error is returned as the post has already been liked. If there is and the `like` boolean is false then the entry is removed from the PostLikes table in the database. If there is no database entry, essentially the reverse is done, if `like` is true, an entry is inserted into the PostLikes table and if `like` is false then an error is returned.

getLikers

This method must return a list of PersonView objects which contains a name, username and student id. A PrepareStatement was created to check the input topic id exists to send the appropriate error. Another prepare statement was created to select the person information by joining the Person table onto the TopicLikes table, and ordered in terms of `personIds`. A resultSet is used to iterated through the list and add them to the array list of PersonView objects, and returned.

getTopic

This method begins by checking the existence of a topic with this ID in the database and if so retrieves the necessary information about the topic and the forum it is contained in. Another query is then used to get the posts contained within the topic, with a left outer join with the PostLikes table being used to get the number of likes each post has. The results are then iterated through and added to the list of PostViews.

getAdvancedPersonView

This method required complex SQL queries in order to avoid querying the database in a loop. In the first query two left outer joins are used to retrieve the number of likes posts and topics created by this user have received. Another lengthy query is then used to get the rest of the required information. Separate joins were required to get the information about the creation of the topic, the information about the most recent post within the topic, the number of topic likes and number of post likes for all the topics which have been liked by the user. These results are then iterated through and added to the list of TopicSummaryViews.