Databases CW 2: Group 10 Report

# Database Schema:

In order to populate the database a create.sql script was created. This in turn acts as the database schema. It was decided to have a separate table for each section of the website. 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, which has a Post. Even though each table has a candidate key which could be used as a primary key, an auto incrementing ID value was implemented acting as the primary key. This helped to keep the schema consistent, especially when using foreign keys between tables. Throughout the schema it has been assumed that titles for forums, topics and posts are not unique. Below are detailed explanations for each table.

## Person:

This table holds a list of users. Any person created or deleted in the entire application is stored here and only here. 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.

## 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. This is a point for further discussion.

## 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.

## LikeTopic:

This tables 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.

# A screenshot of a cell phone Description automatically generatedSchema visualized

# 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 dependancy 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 depencies 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 depencies 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 dependancy 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 depencies 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 depencies 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 depencies 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 depencies contain a candidate key |

It should be pointed out that all tables are in BCNF due to the reasons given in the table above.

# Discussion

When implementing the API methods in task 2, there were several key stages that required reimplementation of code:

The addNewPerson() method refers to the person table. According to the schema the student ID is allowed to be NULL but in case it is NOT NULL it cannot be empty. A test was implemented that checks if the student ID is not null. In case it is not null it must not be empty and hence another check was carried out. A result.failure() is therefore returned when the student ID is NOT NULL and the student ID is empty. The student ID can have two forms; 1. NULL or 2. Not NULL && Not Empty.