Username:mz19460 COMSM0016: Databases Coursework 2

Task1 — Schema Design

Design Descriptions：

After understanding the corresponding functions required by the API, confirming the number of necessary tables and establish the relationship between the tables based on the object-oriented. In the process of creating the table relationship, as far as possible to achieve higher normalisation requirements. My design of the forum database framework illustrated in the figure:

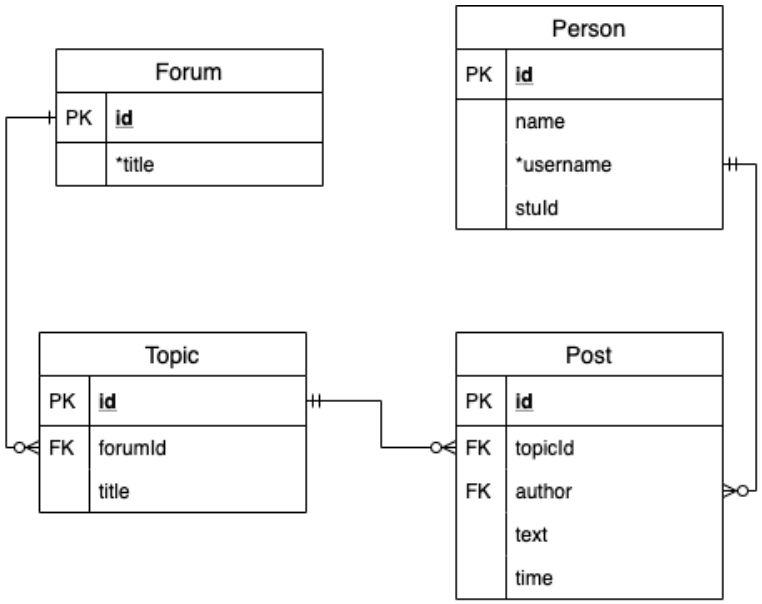

figure 1. Schema of Forum Application

1. Function Analysis

The central services of this forum application are user login, the release of the forum themes and corresponding subtopics, and the release of comments under the topics. Consequently, I divided the functions into four tables (objects): Person, Forum, Topic, Post. Table Person records user information. Table Forum distinguishes between different forum themes. Table Topic and Post use to implement the function of posting replies under subtopics.

2. Design Analysis

The purpose of this design is to give priority to facilitate API access to data. At the same time, the structure of the forum application has a clear hierarchical relationship. In order to reduce the duplicated storage of data, it is an efficient way to divide the table classification according to the hierarchy (fig.2).

figure 2. Hierarchy of Functions

Here are some examples of Java API methods connecting to database which showing advantages of schema design：

- The getUser () method focuses on quoting user information, which means that this method only needs to query the contents from the Person table.
- Another example is createTopic () method, which contains changing contents from two different tables. The execution of the two insert statements becomes very straightforward because of the independence of Topic Table and Post Table. At the same time, according to the connection relationship of the two tables, when the new topic is created, there must be an initial post, which also means that the information of the topic creator is potentially obtained (Foreign key connection of Post table and Person table).

3. Normalisation Analysis

From a normalisation point of view, each table that I created has its own candidate key, so the 1NF is satisfied. To conform to 2NF, every non-key attribute in each table not partly depend on candidate key. At the same time, it is no transitive dependencies in each table. Therefore, the table design meets 3NF.

Task 3 — "like" Functionality

Design Descriptions：

A new table is created to store "like" function relevant data to minimise the impact on existing tables (fig.3).

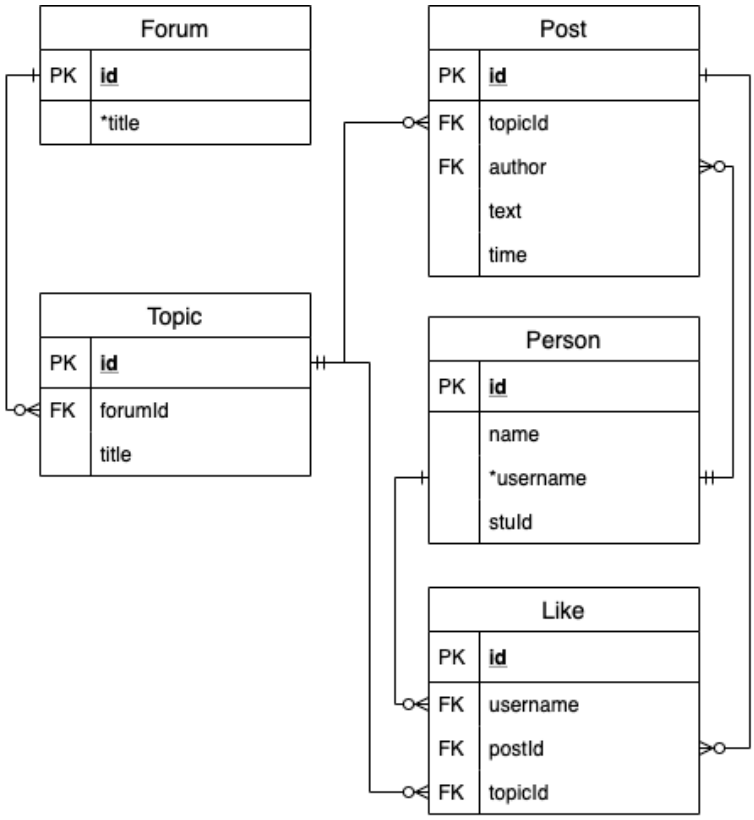

figure 3. Schema of adding “like” functionality

1. Function Analysis

The table of "like" will build correspondence with Table Person, Table Topic, and Table Post because of functional requirements.

There are three main points of "like" functionality:
1. The number of likes of a single Topic or Post can be obtained separately.
2. The names of the users who like the same topic can be obtained.
3. Users can get a list of Topics or Posts that have been liked by themselves.

To achieve these three points, Table like will use foreign keys connect with Table Person, Table Post, and Table Topic. For flexibility of store data, both attribute 'topicId' and attribute 'postId' in Table like can set NULL, while attribute 'username' cannot be NULL or empty.

2. Design Analysis

The most apparent drawback of adding new tables is data redundancy. The data of non-primary

attributes in Table 'like' have already existed in other tables, so the data storage of the 'like' table is only to achieve functional requirements rearrange the existing data.

Creating a new table for function requirements also has certain advantages, which will reduce the impact on the realized function. In other words, the impact on the API methods that Java has constructed will be minimised.

SQL Queries of New Schema：

```
MariaDB [bb]> Select * from Likes;
+----+----------+--------+---------+
| id | username | postId | topicId |
+----+----------+--------+---------+
|  1 | mz19460  |      1 |    NULL |
|  2 | mz19461  |      1 |    NULL |
|  3 | mz19460  |   NULL |       1 |
|  4 | mz19460  |   NULL |       2 |
|  5 | mz19461  |   NULL |       3 |
|  6 | mz19462  |   NULL |       1 |
|  7 | mz19460  |      2 |    NULL |
+----+----------+--------+---------+
```

figure 4. Table Likes

The 'LIKES' about Topic and Post are counted separately. They all can match the corresponding username. Query the 'Likes' list of Post or Topic just needs to know their Id. Query the 'likes' record of a specific user needs to know the username.

- **Getting the total number of topics and posts liked by a specific user.**

In other words, this operation is getting the total number of user's likes on topics and posts.

```
SELECT COUNT (*) FROM Likes
WHERE Likes.username = ?;
```

We can also get the number of liked topics or posts separately.

```
SELECT COUNT (postId) FROM Likes
WHERE username = ?;
```

```
SELECT COUNT (topicId) FROM Likes
WHERE username = ?;
```

- **Getting the names of all people who have liked a specific topic, ordered alphabetically.**

```
SELECT name FROM
(Likes INNER JOIN Topic on Likes.topicId = Topic.id)
INNER JOIN Person on Person.username = Likes.username
WHERE topicId = ? ORDER BY name;
```

This example shows how to get the name list of 'likes' of a specific topic. At the same time, it

also can get a name list of 'likes' of a particular post in the same way.

```
SELECT name FROM
(Likes INNER JOIN Post on Likes.postId = Post.id)
INNER JOIN Person on Person.username = Likes.username
WHERE postId = ? ORDER BY name;
```

Each specific topic or post has a particular publisher, so this feature can also be extended to record the number of likes of the topic or post creator.

Risk Assessment:

Adding functionality will change the basic structure of the database, which will bring modification risks to existing applications. In order to reduce the impact on existing functions, the new functionality 'likes' are recorded in the new table, which also means that it can be an independent object in the Java program.

If the corresponding API add into the Java program, the methods are rough as follows:

1. getUserTopicLikes (String username) return a specific user's list of topics
2. getUserPostLikes (String username) return a specific user's list of posts
3. setTopicLike (String username, int topicId)
4. setPostLike (String username, int postId)
5. countLikes (String username) return the total number of user's likes on topics and posts

These methods added by the new function, which has less impact on existing methods in the Java program. Therefore, in the process of updating the database structure, it should not be entirely restricted by normalization, but also consider the operational issues in JDBC.