Architecture solution for commenting tasks

Mentorship System

by Bartek Pacia Google Code-in 2019

Systers' Mentorship System is currently in development, which means the technical specification is not finalized yet. This stage gives us, developers, the possibility to rapidly add new functionalities, test them and even alter project architecture – without worrying about maintaining backwards compatibility.
In this document I will describe my vision on how the database could be restructured so that task commenting functionality could be easily added to the system. These are *very* breaking changes and are very likely to require dropping all current tables.

Overview of the current architecture

Server for Mentorship System is written in Python using Flask microframework with a lot of extensions. To store data, SQLite database is used and to make working with it easier, we use SQLAlchemy (very popular Python SQL toolkit and Object Relational Mapper).

Current structure of the database is not very flexible. It was experienced by some GCI participants when they tried to implement task commenting functionality[1].
After some research, I came to a conclusion that this inflexibility is caused by some data not being structured like it would be expected to in a typical SQL database. What I mean is that tasks_list implementation is not very clear and doesn't make much sense. Furthermore, there is a thing called JsonCustomType which, in my opinion, is unnecessary. We could easily replace it with native, so to say, SQL implementation.

Overview of the updated structure

That's why I decided to design a new database structure which you can see here. (I'm not including it here because it would be very inconvenient – the spreadsheet is too large). I have designed it with task commenting feature in mind.

To start, I will list tables I have created and describe them briefly.

[1] Zulip conversation link

1. Mentorship relations
   The only change I made in this table is deleting “tasks_list” field, because as you’ll see below, it becomes unnecessary. The rest remains the same – the most noteworthy fact is that the user can be in only 1 relation at one time.
2. Tasks
   I created a new table called “tasks”. It holds all tasks of all users. They are in one-to-many relationship with mentorship relations, because a mentorship relation can have many tasks.
   Task contains data like task title, description and creation + completion timestamp.
3. Comments
   This table holds info about a comment regarding some specific task. Comments are in one-to-many relationship with tasks, because every task can have many comments.
   A comment holds also information about creation date, so it’s easy to sort them, either on server or client side.
4. Reactions
   This is something that is getting more and more popular in all kinds of apps where between users happen. A reaction is just an emoticon that the user can add to certain task. The user can add only as many *different* reactions as they want to each comment. Reactions are in one-to-many relationship with comments, because every comment can receive many reactions (from different users).
   In my design, reactions are stored as strings (VARCHARS) representing Unicode code of a reaction. I am aware that it is not the best idea, but I haven’t been able to come up with anything better.

To sum up, changes I made include unification of the way in which we store data. Each unique entity (such as task, comments and so on) now has got its own table. Those entities are organized in one-to-many relationships. It makes retrieving specific data we want easy, because it’s structured into logical chunks, so to say.

Friends feature

I have also a simple idea on how to implement friends feature in the system. Table structure for this is at the bottom of the spreadsheet.
I would create a separate table called “friends” containing information about friendship relations. This would work similarly to mentorship relations:

- user 1 sends a friendship request to user 2
- user 2 can either accept or reject the friendship request

If user 2 accepts the friendship request, friendship’s status is changed to ACCEPTED and users are considered to be friends. Friendships could be used in

some kind of a news feed feature, where users would see recent accomplishments of their friends, tasks they have done and could comment on those tasks.
with an exception that when the user removes another user from friends, the friendship is completely removed from the database (where cancelling a mentorship relation only changes its state).

Alternatives

Alternative solution to everything listed above would be to migrate the database to a NoSQL one, for example to Cloud Firestore, part of the Firebase (which is part of the Google Cloud Platform). It is a cloud-hosted database, so we wouldn't have to worry about maintaining servers or doing maintenance work. Firebase offers a free tier which is enough for testing and small-scale services. When the demand for resources grows, there's also a pay-as-you-go tier, whose pricing can be found here. I have been using Firebase for more than 3 years now and I can say that the free tier would be perfectly enough for the current user base (assuming only Cloud Firestore would be used).

Redundancy, which is generally advised against in relational databases, is perfectly normal in NoSQL databases. In NoSQL databases, there are no relations – instead objects are nested. This leads to "completeness" of data being downloaded from the server. One of advantages of such behavior is greatly simplified implementation of client-side caching.

In case of a relational database, in order to cache all data that can be referenced by one relation, several requests to the server would have to be made. It is still totally possible, but requires more work and more code, which equals more bugs, at least at the beginning.

Wrapping up

I hope that changes suggested by me will help to improve the design of Mentorship System database. Overall, the current database structure is good, but we should always strive for the best possible solutions.