Sezvo-ndinemwi Mpfunya (mpfunya), Aidan Kucharski (akuchars), Nikita Adekar (adekar)

CMPUT 291 Mini-Project 1 Report

Overview

This project was an opportunity to learn how to use SQL in a host programming language using groups to facilitate the nature of teamwork in a real-life situation. The primary goal was to build a system that keeps the enterprise data in a database and provide services to users.

Mini-Project-1 is a python command-line application with an SQLite database using the sqlite3 python package. Using this program, users can log in and out of the database, post a question, search for posts and perform post actions such as adding an answer, adding a vote and, for privileged users adding a mark, giving a badge, adding a tag and editing a post. All commands are executed through the command-line. Users are given options to choose from which require input and they are notified when the input is invalid.

The database schema for this project is defined as:

- users(uid, name, pwd, city, crdate)
- privileged(uid)
- badges(bname, type)
- ubadges(uid, bdate, bname)
- posts(pid, pdate, title, body, poster)
- tags (pid, tag)
- votes(pid, vno, vdate, uid)
- questions(pid, theaid)
- answers(pid, qid)

User guide/Detailed design

Main - The main function acts as the point of execution for the program. Database input is located here and specifies actions to perform for each input type. The application is opened using the command “mini_project1.py -i or -d <./db name>” where the -i flag is used to create a new database given by the name <db name>, and -d opens an existing database given by <db name>. Once this is successful the system will immediately give the user a prompt to log in to such a database.

CMPUT 291 Mini-Project-1 Report

Sezvo-ndinemwi Mpfunya (mpfunya), Aidan Kucharski (akuchars), Nikita Adekar (adekar)

Account creation and login – Once opened the user is asked to log in. Users are asked to specify whether they are a returning user or a new user. If the user is returning the user id is verified and if it is already in use a message is given and the user is asked to log in again. The password is also verified by matching it to the one stored in the system (password is hidden by built-in python library getpass). If the user is new, the user must input a user id, name, city, password and this data is added to the database. The login prompt loops until a valid username/password combination are specified. The user id of the login user is returned. Once a log-in is completed they are shown menu options allowing them to logout, quit, search for a post or add a question.

Menu Option breakdown

logout - User is required to input 'logout'. When selected, the user is taken back to the login prompt and follows the same instructions for logging in.

quit – User is required to input 'quit'. When selected the main loop ends and the program stops running.

post a question - User is required to input 'question'. The user is able to post a question by providing title and body texts. This data is added to the corresponding table in the database. A unique post id (pid) is assigned by the system, the post date is set to the current date and the poster is set to the user posting it.

search_posts - User is required to input 'search'. Given user input(s) this function returns a list of posts that include the searched keywords. For each entered keyword all posts that include that word in the title, body or tag are obtained. For each matching post, the columns of the posts table, the number of votes (adds 0 if no votes exist), and the number of answers if the post is a question (or zero if the question has no answers) are displayed.

When matching posts are found the user is offered a list of actions to perform on the searched post. These options include adding an answer and adding a vote. A user-id check is performed to identify privileged users. Privileged users are able to add a mark, give a badge, add a tag and edit the post.

Post-action Menu Option breakdown

post_action_answer - User is required to input ' answer question'. If the selected post is a question, the user is able to post an answer for the question by providing title and body texts. The data is added to the corresponding table in the database. A unique post id (pid) is assigned to the answer by the system. The post date is set to the current date and the poster is set to the user posting it. Lastly, the answer is linked to the question.

post_action_vote - User is required to input ' vote '. The user is able to vote on the post (if not voted already on the same post). The vote is recorded in the database with a vno assigned by the system. The vote date is set to the current date and the user id is set to the current user.

Sezvo-ndinemwi Mpfunya (mpfunya), Aidan Kucharski (akuchars), Nikita Adekar (adekar)

Privileged user Post-action Menu Option breakdown

Privileged users can perform the menu options stated above including others specified below. These actions are only displayed when the user id is verified in the privileged table.

post_action_mark_as_the_accepted - User is required to input ‘ add accepted ’. The user is able to mark the post (if it is an answer) as the accepted answer. If the question already has an accepted answer, the user is prompted if they want to change the accepted answer. The user can select to change the accepted answer or leave it unchanged.

post_action_give_badge - User is required to input ‘ give badge ’. The user is prompted to give a badge to the poster by providing a badge name. The information is recorded in the database with the badge date set to the current system date.

add_tag - User is required to input ‘ add a tag’. The user is prompted to add a tag to the searched post. They are required to enter the name of the tag. They are continually prompted to add a tag and must input ‘n’ to exit the prompt.

edit_post - User is required to input ‘ edit title ’or ‘ edit body text’ .The user is able to edit the title and/or the body of the post. Other fields are not updated when a post is edited. The user is shown the previous title/ body text and once the update is made they are shown the edited post.

Testing Strategy

Our testing strategy was to use data inserted into our tables that could cover most of the possible scenarios. This included

- Verifying the correct uid chosen for post actions.

- Verifying that the data was added into each table correctly.

- Ensuring that unique ids were added for specific commands

- Input testing was added for each command. Returning a message when the input was invalid.

The code was modified to accommodate these needs.

Group Work Strategy

Aidan Kucharski (akuchars)

• Hours: ~15 hours

• Progress: Provided most of the groundwork and organization for the project. Defined the project structure and developed functionality for system logging, posting a question and searching for a post.

Nikita Adekar (adekar)

• Hours: ~15 hours

• Progress: Developed functionality for adding an answer, adding a vote, adding a mark and giving a badge. Also contributed by fixing bugs and writing unit tests for the system.·

Sezvo-ndinemwi M'pfunya (mpfunya)

• Hours: ~14 hours

• Progress: ~ Developed functionality for adding a tag and editing a post. Also contributed by writing the project report.

Work Coordination

We coordinated through GitHub. An organization was set up for the project (Group-CMPUT-291-Databases) and each member added their contributions with a new file to track progress. Members notified the group when updates were being made and this created a good workflow with minimal issues. Contributions from each member were compiled into one py file.

Communication was performed via Slack and members used this platform to ask questions and to clarify additions to the work. All the work was done remotely since members were in different locations.

Design decisions

All members were familiar with python and found it to be the most suitable for executing the project. Using python allowed the use of libraries such as datetime which were convenient for setting the times for posts. Pytest allowed us to continually debug the work as it was being built which made identifying key flaws easier.

Figuring out how to access the pids of the searched post was our first area of concern. We thus agreed to make search_posts return a list of lists that way the pid for corresponding posts could be easily accessed and passed to other functions. We also created a function to check whether a user was a privileged user which was necessary for displaying their Post actions. Another unanimous decision was to have the user enter the command line "mini_project.py -i -d <./db name>" where the -i flag is used to create a new database given by the name <db name>, and -d opens an existing database given by <db name>. This was the easiest way to get the program opened.