CMPUT 291 - File and Database Management (Fall 2020)

Mini-Project 1

CMPUT291 - Fall 2020
Mini Project I

(group project)

***Due**: Nov 4th at 5pm*

Clarifications:

You are responsible for monitoring the course news and discussion forums in eclass and this section of the project specification for more details and clarifications. No clarification will be posted after *5pm on Nov 2nd*.

- **Oct 29.** Here is a marking rubric with some details of the testing process.
- **Oct 28.** The same tag cannot be assigned to the same post even if the case is the different (e.g. if 'database' is assigned, 'Database' cannot be assigned to the same post).
- **Oct 27.** (1) A tag can include more than one word, e.g. "relational databases" is a valid tag. (2) When searching for a post, the search keyword can be part of another word. For example, the keyword "relation" will match a post that has "relational" in the title. (3) When ordering the results, the number of matching query keywords and not the number of occurrences are counted. For example, the same keyword that appears in both title and body is only counted once.
- **Oct 26.** (1) The given schema cannot be updated under any condition. This also means temporary tables should not be created. (2) Passwords cannot be encrypted or hashed, to allow an easy testing of the projects. (3) All votes in the project are upvotes (downvotes are not considered).
- **Oct 21**. (1) The user id and pwd will only contain alphanumeric characters. (2) Privileged users are assumed to be in the database and they cannot be added through the registration process.

Introduction

The goal of this assignment is twofolds: (1) to teach the use of SQL in a host programming language, and (2) to demonstrate some of the functionalities that result from combining SQL with a host programming language. Your job in this project is to build a system that keeps the enterprise data in a database and to provide services to users. You will be storing data in a SQLite database and will be writing code in Python (or similarly Java/JDBC, C, etc.) to access it. Your code will implement a simple command line interface. You are free to implement a GUI interface instead but there will be no support nor bonus for doing that. You are also free to write your code in Python, Java, C, C++, Perl or any other language that is suited for the task. If you decide to use any language other than Python, you should let the instructor know in advance.
Your project will be evaluated on the basis of 84% of the mark for implementing the functionalities listed in this specification; this component will be assessed in a demo session. Another 12% of the mark will be assigned for both the documentation and the quality of your source code. 4% of the mark is assigned for the quality of your group coordination and the project break-down between partners.

Group work policy

You will be doing this project with *one or two other partners* from the 291 class. Register your group at the group registration page. It is assumed that all group members contribute somewhat equally to the project, hence they would receive the same mark. In case of difficulties within a group and when a partner is not lifting his/her weight, make sure to document all your contributions. If there is a break-up, each group member will get credit only for his/her portion of the work completed (losing the mark for any work either not completed or completed by the partners).

Database Specification

You are given the following relational schema.

- 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)

These tables are derived from the specification of Assignment 1 and are identical to those in Assignment 2 except the tables *users*, which has now a field for password (referred to as pwd), and table *privileged*, which is new, keeping the id of privileged users. These users can perform more actions than ordinary users. The SQL commands to create the tables of the system are given here (right click to save as a file). Use the given schema in your project and do not change any table/column names.

Login Screen

The first screen of your system should provide options for both registered and unregistered users to login. Registered users should be able to login using a valid user id and password, respectively denoted with uid and pwd in table users. Unregistered users should be able to sign up by providing a unique uid and additionally a name, a city, and a password. Passwords are not encrypted in this project. The field crdate should be set by your system to the current date. After a successful login or signup, users should be able to perform the subsequent operations (possibly chosen from a menu) as discussed next.

Users should be able to logout, which directs them to the first screen of the system. There must be also an option to exit the program directly.

System Functionalities

After a successful login, users should be able to perform all of the following tasks.

1. *Post a question.*The user should be able to post a question by providing title and body texts. The post should be properly recorded in the database tables. A unique pid should be assigned by your system, the post date should be set to the current date and the poster should be set to the user posting it.
2. *Search for posts.*The user should be able to provide one or more keywords, and the system should retrieve all posts that contain at least one keyword either in title, body, or tag fields. For each matching post, in addition to the columns of posts table, the number of votes, and the number of answers if the post is a question (or zero if the question has no answers) should be displayed. The result should be ordered based on the number of matching keywords with posts matching the largest number of keywords listed on top. If there are more than 5 matching posts, at most 5 matches will be shown at a time, letting the user select a post or see more matches. The user should be able to select a post and perform a post action (as discussed next).
3. *Post action-Answer.* If the selected post is a question, the user should be able to post an answer for the question by providing title and body texts. The answer should be properly recorded in the database tables. A unique pid should be assigned by your system, the post date should be set to the current date and the poster should be set to the user posting it. The answer should be also linked to the question.
4. *Post action-Vote.The user should be able to vote on the post (if not voted already on the same post). The vote should be recorded in the database with a vno assigned by your system, the vote date set to the current date and the user id is set to the current user.*

Privileged users can perform the following post actions in addition to those that can be performed by ordinary users (as discussed above). These actions are not available to ordinary users.

1. *Post action-Mark as the accepted.* The user should be able to mark the post (if it is an answer) as the accepted answer. If the question has already an accepted answer, the user should be prompted if s/he wants to change the accepted answer. The user can select to change the accepted answer or leave it unchanged.
2. *Post action-Give a badge.* The user can 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.
3. *Post action-Add a tag.* The user should be able to add tags to the post.
4. *Post Action-Edit.* The user should be able to edit the title and/or the body of the post. Other fields are not updated when a post is edited.

String matching. Except the password which is case-sensitive, all other string matches (include user id, name, etc.) are case-insensitive. This means the keyword "database" will match Database, DATABASE, DataBase and database, and you cannot make any assumption on the case of the strings in the database. The database can have strings in uppercase, lowercase or any mixed format.

Error checking. Every good programmer should do some basic error checking to make sure the data entered is correct. We cannot say how much error checking you should or should not do, or detail out all possible checkings. However, we can say that we won't be trying to break down your system but your system also should not break down when the user makes a mistake.

Groups of size 3 must counter SQL injection attacks and make the password non-visible at the time of typing.

Testing

At **development time**, you will be testing your programs with your own data sets but conforming to the project specification.

At demo time, you will be given a database file name that has our test data (e.g., prj-test.db), and you will be passing the file name to your application as a command line argument. Don't hard-code the database name in your application since the database name is not known in advance, and you don't want to change your code at demo time (see next). The database will include the tables given above (created using these SQL statements) and with our own test data. Your application will be tested under a TA account.

Every group will book a time slot convenient to all group members to demo their projects. **At demo time, all group members must be present.** Once we create our tables and populate them with our data, you will be asked to start your application and perform various tasks, showing how your application is handling each task. A mark will be assigned to your demo immediately after the testing.

Here are **some important details about our testing process and your choices**:

1. The demo will be run using the source code submitted and nothing else. It is essential to include every file that is needed to compile and run your code.
2. We must be able to compile and run your code under our account on undergrad machines and using our own database. You are not allowed to make any changes to the code without a hefty penalty.
3. Our test data and our test cases will be published after the project due date but before our demo times. This means, you have a chance to test your application and learn about possible issues (if any) before your demo time.
4. Your code cannot be demoed on a laptop (yours or ours) or any machine other than the lab machine with only one exception. The exception is if you are developing your application using a less traditional programming language or tool that is not available on lab machines, you MAY be allowed to demo your application on a laptop. Those cases should be discussed with the instructor well before the project due date and an approval must be obtained. Otherwise, you cannot demo your project on any machine other than the lab machines.

Instructions for Submissions

Your submission includes (1) the application source code, (2) README.txt, and (3) your design document *Report.pdf.*

- Create a single gzipped tar file with all your source code, additional files you may need for your demo, README.txt and Report.pdf. Name the file *prjcode.tgz*.
- Submit your project tarfile in the project submission site.
- All partners in a group must submit their own copies (even though the copies may be identical).

The file README.txt is a text file that lists the names and ccids of all group members. This file must also include the names of anyone you collaborated with (as much as it is allowed within the course policy) or a line saying that you did not collaborate with anyone else. This is also the place to acknowledge the use of any source of information besides the course textbook and/or class notes.

Your design document must be type-written and is saved as a PDF and be included in your submission. Your design document cannot exceed 4 pages.

The design document should include (a) a general overview of your system with a small user guide, (b) a detailed design of your software with a focus on the components required to deliver the major functions of your application, (c) your testing strategy, and (d) your group work break-down strategy. The general overview of the system gives a high level introduction and may include a diagram showing the flow of data between different components; this can be useful for both users and developers of your application. The detailed design of your software should describe the responsibility and interface of each primary function or class (not secondary utility functions/classes) and the structure and relationships among them. Depending on the programming language being used, you may have methods, functions or classes. The testing strategy discusses your general strategy for testing, with the scenarios being tested, the coverage of your test cases and (if applicable) some statistics on the number of bugs found and the nature of those bugs. The group work strategy must list the break-down of the work items among partners, both the time spent (an estimate) and the progress made by each partner, and your method of coordination to keep the project on track. The design document should also include a documentation of any decision you have made which is not in the project specification or any coding you have done beyond or different from what is required.

Your design document will not include the source code. However your source code will be inspected for source code quality (whether the code is easy to read and if data processing is all done in SQL and not in the application) and self-documentation (whether the code is properly commented).

Last modified: Thursday, 29 October 2020, 1:22 PM

◄ Mini-Project 1 groups

Jump to...

Midterm Part I (A2/EA2) ►