Carnegie Mellon University
Department of Computer Science
15-415/615 - Database Applications
C. Faloutsos & A. Pavlo, Fall 2016
Prepared by Jiexi Lin
DUE DATES: Ph1: 11/9, Ph2: 11/28, both at 3:00 PM

Homework 7

IMPORTANT - what to hand in: For each of the two phases, please deliver **all** the elements below (*penalties* for omissions).

- Phase 1: Due at **11/9, 3:00 PM**:
  - *hard copy*: All your documentation for Phase 1.
  - *Blackboard submission*: a file `[andrew_id_hw7_phase1].pdf` with your documentation.
- Phase 2: Due at **11/28, 3:00 PM**
  - *hard copy*: A printed version of your `./cmupaper/paper/functions.py` , **rememeber to include your name and andrew id in the header comment of the source file.**
  - *Blackboard submission*: a tar file named`[andrew_id_hw7_phase2].tar`, with your source file `./cmupaper/paper/functions.py` , all your test scripts and a brief ReadMe explaining how to use these test scripts.

Important for Phase 2: This url (`http://15415.courses.cs.cmu.edu/fall2016/hws/HW7/cmupaper_vm.tar`) contains everything you need for Phase 2. See detailed instructions on this webpage (`http://15415.courses.cs.cmu.edu/fall2016/hws/HW7/index.html`).

Reminders:

- *Plagiarism:* Homework may be discussed with other students, but all homework is to be completed **individually**.
- *Late Homeworks*: The usual rules: (a) hard copy to Mrs. Marilyn Walgora, and (b) email to all TAs, with the subject line `15-415 Homework Submission (HW 7) Phase`[phase#], and the count of slip-days used (and remaining).

For your information:

- Graded out of **100** points;
- **2** questions total
- 5-10 hours for phase 1; 10-20 hours for phase 2.

Question	Points	Score
Deliverables - ph1	35	
Deliverables - ph2	65	
Total:	100	

Homework 7 continues…

1 Introduction

The goal is to design and implement CMUPaper, a very simple application that lets you upload a paper, search papers with different criteria, vote for other one's papers and see some statistics.

Your work is divided into two phases. In the first phase you will come up with some design concepts according to the application requirement.

The second phase consists of *implementation and testing*: You need to implement the APIs defined by us and get your website running.

2 Requirements

2.1 Data requirements

The followinhs are entities that we need to record in the database. In Phase 1, it's up to you to decide how many tables we need to store the data. After Phase 1 is due, we will release our version of schema and every one must use that to implement the project.

- **User**: Each user has a unique username (up to 50 characters) and a password (up to 32 characters). A user can upload zero or more papers and *like* zero or more papers.
- **Papers**: Each uploaded paper has a title (up to 50 characters), a timestamp it was uploaded, 0 or more tags and the text content of this paper extracted from an uploaded pdf file. Note that the text content can be arbitrally long. Tags should have only alphabetic characters and should be less than 50 characters. Optionally a paper can have a description (up to 500 characters). The system should also keep track of which user uploaded the paper and which user *likes* the paper at what time. Papers may have the same title and the same author. The system will need to identify a paper with a unique id internally.

Example: The user *"Jiexi"* uploaded a new paper with the title "Scalability of Random String Generators" and the description as *"Random generators are a common component in all kinds of benchmark tools. However simple they are, sometimes they can become a severe scalability bottleneck."*. He also uploaded a pdf file named "jiexil.pdf" and then added three tags "*tag1*", *"tag2"*, *"tag3"* to the paper. This publication should be stored in the database as described above.

Another user *"Huanchen"* logined to the website, searched with the keyword *"scalability"* and found the paper uploaded by *"Jiexi"*. He downloaded and read the paper and *liked* it by clicking the buttons on the webpage. As a result, the paper *"Scalability of Random String Generators"* becomes more popular and the system may recommend it to those users who have tastes similar to *Huanchen's*.

2.2 Functionality requirements

In CMUPaper we have users who upload papers, *like* papers uploaded by other users, read papers, etc, as described below. For more implementation details please check comments in the `./cmupaper/paper/functions.py` . You will implement the following Tasks:

T.1 **Reset database**: Keep the tables, but delete all their records.

T.2 **Create user account (Register)**: We need the username of the user and the password. Prompt for a new username, if the proposed one is taken.

T.3 **Login**: Your system would ask for a username and password, and authenticate the user or deny further access.

T.4 **upload a paper with tags**: Once authenticated, a use rcan post a paper. Note that the creating a new paper and add tags should be atomic.

T.5 **Delete a paper**: Given a paper, delete it and any related information from the database.

T.6 **User timeline**: After the user logs in, the main page would display the user's timeline. That is, your system should show the K most recent papers, with their ids, titles, authors, descriptions, uploaded time, tags and numbers of *likes*. The results should be sorted first by uploaded time (newest first) and then break ties by the paper's id. Treat K as a parameter.

T.7 **Global timeline**: Similar to the user timeline, we need to display the K most recent papers' information (id, title, authors, descriptions, uploaded time) uploaded in the system.

T.8 **Search for papers**: For the input query, search for the papers that match a single keyword query in either its title, description or content. The systen should return at most K paper with the following infomation: id, title, authors, descriptions, uploaded time. The results should be sorted first by uploaded time (newest first) and then break ties by the paper's id. Treat K as a parameter.

T.9 **Search by a tag**: Given a tag, search for the papers that has the tag. The systen should return at most K paper with the following information: id, title, authors, descriptions, uploaded time. The result should be ordered first by the paper's uploaded time (newest first) and then break ties by the paper's id. Treat K as a parameter.

T.10 **Get tags of a paper**: Given an id of a paper, return all tha tags used in this paper. Tags returned should be ordered by lexical ascending order.

T.11 **Like/Unlike a paper**: Your system should allow a user to *like* another paper uploaded by other users once and only once. A user is not allowed to "*like*" one's own paper. Once a user *liked* a paper they are allowed to *unlike* it. After that, they can thumbsup for it again. Associated with each *like*, the system should record the timestamp that the user made the *like*.

T.12 **Count likes**: Count the number of *likes* for a particular paper.

Homework 7 continues…

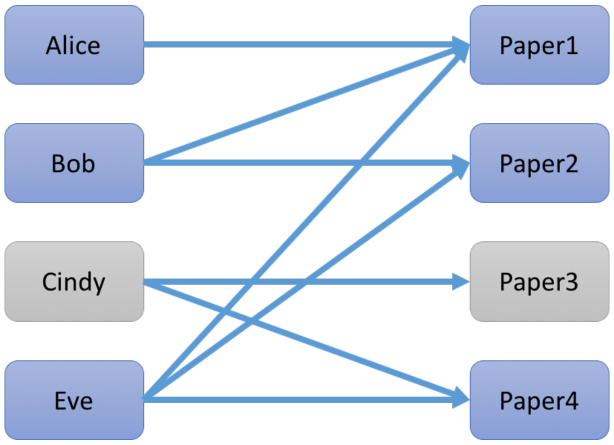

Figure 1: Example of cohorts

T.13 **List favourite papers of a user**: List all the paper *liked* by a given user. The results should be sorted first by **the timestamp of the *like* (newest first)** and then break ties by the paper's id.

T.14 **List most popular papers**: Generate a list of most popular paper uploaded after a given timestamp T. The popularity of a paper is based on the total number of *likes* it has received from when it's uploaded. The systen should return at most K paper with the following information: id, title, authors, descriptions, uploaded time. The results should be sorted first by **number of *likes* (most popular first)** and then break ties by the paper's id. Treat K as a parameter.

T.15 **Recommend papers based on likes**: For a given user, your system should recommend papers based on "cohort". For a given user U, the cohorts of U are users that *like* the sampe paper *liked* by U. We recommend a paper to U if at leasts of $U's$ cohort *likes* the paper and at the same time this paper is not *liked* by U. The more cohorts of U *likes* the paper, the higher ranks the paper gets. The systen should return at most K paper with the following information: id, title, authors, descriptions, uploaded time. The results should be sorted first by **the number of *likes* from cohorts (descending)** and then break ties by the paper's id.

Example: In Figure T.15. The system will first recommend *Paper2* and then *Paper3* to *User1*.

T.16 **User statistics**: For a given user, display the following counts, or print a warning if the user does not exist.

(a) The number of papers uploaded by that user
(b) The number of *likes* made by that user
(c) The number of unique tags posted for that user's papers
Example: If a user has uploaded 2 paper, each of them has only the tag "CMU". We'd say that the number of unique tags used by this user is 1.

T.17 **Global statistics**: The system should be able to generate statistics for all of the users and papers.

(a) **Most active K users**: List of K users with the most papers uploaded as well as their numbers of paper uploaded. Return a list of usernames sorted in descending order of number of uploaded papers. Break ties by username in lexical ascending order.

(b) **Most popular tags**: List of K tags sorted in descending order of popularity, i.e. count of usage. Also include their occurencies in the result. Break ties by tagname in lexical ascending order. tags that used most. Return a list of

(c) **Most popular tag pairs**: List of K tag pairs that appear together most frequently as well as their occurencies. Return a list of tag pairs where tags has lower lexical order comes first in the pair. The list should be sort first by the frequency of co-occurence of the tag pair, and then break ties by lexical ascending order (apply to first tag and then the second one).

In all the above cases, treat K as a parameter.

Homework 7 continues. . .

Phase 1 - Deliverables - ph1 . . . . [35pts]

No need to separate your answers

We have the following deliverables:

Specifically, for Phase 1, the point distribution is as follows:

(a) [**3 points**] The document forms, including the assumptions and design decisions you made.

(b) [**3 points**] The ER diagram. Make sure you specify the cardinalities of the relationships, as in HW1.

(c) [**6 points**] The relational schema.

(d) [**8 points**] SQL DDL statements that create the above schema - make sure you include all constraints (primary key, foreign key, etc)

(e) [**15 points**] SQL DML statements for tasks T.1-T.17.

Homework 7 continues. . .

Phase 2 - Deliverables - ph2 . . . . [65pts]

No need to separate your answers

The deliverables and point distribution of Phase 2 are as follows:

(a) [**0 points**] **Checkout the instruction for phase to in the project webpage (`http://15415.courses.cs.cmu.edu/fall2016/hws/HW7/index.html`).**

(b) [**0 points**] Checkout the schema uploaded by us on the project web page (`http://15415.courses.cs.cmu.edu/fall2016/hws/HW7/index.html`). **In phase 2, you are required to use the schema provided by us, otherwise the auto grader can not grade your submission.**

(c) [**10 points**] Listing of your testing efforts - for each task, please write down which error cases you tested for (ie., non-existing user, illegal timestamp, etc)

(d) [**50 points**] The actual implementation: In the source code provided by us, we have already implemented the web frontend for you. We also defined a set of APIs in `./cmupaper/paper/functions.py` . The web frontend will communicate to the backend database via these APIs. Your job is to implement these APIs, fully test them and make the website running. T.1-T.17.

Table 1 shows the breakdown of the points. For each tasks, half of the assigned points go to the basic implementation (i.e. having a working implementation of the task) and the second half goes to *testing* and *error checking*.

Task	**Points**
T.1 Reset database	1
T.2 Create user account (Register)	1
T.3 Login	1
T.4 upload a paper with tags	3
T.5 Delete a paper	3
T.6 User timeline	3
T.7 Global timeline	3
T.8 Search for papers	3
T.9 Search by a tag	2
T.10 Get tags of a paper	1
T.11 Like/Unlike a paper	4
T.12 Count likes	1
T.13 List favourite papers of a user	3
T.14 List most popular papers	3
T.15 Recommend papers based on likes	6
T.16 User statistics	6
T.17 Global statistics	6

Table 1: Point breakdown

Question 2 continues. . .

(e) [**5 points**] *SQL injection*: In short, make sure you strip-off all the "escape" characters from your SQL strings. The reason is that, apart from the above functions, you want to prevent unauthorized access to your system. More specifically, you want to protect your SQL queries from an attack called *SQL injection* (see `http://en.wikipedia.org/wiki/SQL_injection`). For instance, code oblivious to SQL injection, may allow an intruder to log-in without a password. In python it's relatively easy to rule out SQL injection (see `http://initd.org/psycopg/docs/usage.html`).

Question 2 continues…

3 What to handin

3.1 Phase 1

- *Hard copy*: All your documentation for Phase 1.
- *Blackboard submission*: a file `[andrew_id_hw7_phase1].pdf` with your documentation.

Don't forget to include your name and andrew ID in both the hard copy and the blackboard submission.

3.2 Phase 2

- *Hard copy*: A printed version of your `./cmupaper/paper/functions.py` .
- *Blackboard submission*: a tar file named`[andrew_id_hw7_phase2].tar`, it should contain the following things:
  - `./functions.py` – Your implementation of all the database APIs.
  - `./customize_checker.py` – Your customized checker to test your implementation.
  - `./ReadMe` – A brief introduction on how to run your checker.

Don't forget to include your name and andrew ID in both the hard copy and the blackboard submission. **Rememeber to include your name and andrew id in the header comment of the source file.**