Carnegie Mellon University
Department of Computer Science
15-415/615 - Database Applications
C. Faloutsos & A. Pavlo, Spring 2014
Prepared by Alex Beutel and Vagelis Papalexakis
DUE DATES: Ph1: 4/1, Ph2: 4/15, both at 1:30pm

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 **4/1, 1:30pm**:
  - *hard copy*: All your documentation for Phase 1.
  - *Blackboard submission*: a file `[andrew_id_hw7_phase1].pdf` with your documentation.
- Phase 2: Due at **4/15, 1:30pm**
  - *Website*: A working web site - include its URL in both hard and e-copies below.
  - *hard copy*: the code you wrote for Phase 2.
  - *Blackboard submission*: a zip file named`[andrew_id_hw7_phase2].zip`, with all the necessary files for us to re-create your web site.

Important for Phase 2: The zip-file `http://www.cs.cmu.edu/~christos/courses/dbms.S14/hws/HW7/hw7.zip`. contains everything you need for Phase 2.

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	

1 Introduction

The goal is to design and implement Flitter (CMU Fictitious light-weight Twitter), a very simple version of Twitter. Your work is divided into two phases, as we discuss in the lecture foils, following the `http://www.cs.cmu.edu/~christos/courses/dbms.S14/slides/CMU_ONLY/Roussopoulos-Yeh.pdf` The first phase consists of the *requirement analysis*, *system analysis*, *conceptual modeling* and *task emulation*. The second phase consists of *implementation and testing*: Using the API we provide you in the *hw7.zip* file above, you will have to implement your database design, as well as the functionality that the API supports, and deploy your solution on a webserver.

2 Requirements

2.1 Data requirements

- **User**: For each user, the system needs to track the unique username, ~~(between 4 and 50 characters)~~ (between 2 and 50 characters), the password, the users he/she is following, and the users that follow him/her.
- **Tweet**: The system needs to store the body of the tweet (at most 140 printable characters), the username of the author, and the timestamp it was created.

2.2 Functionality requirements

In Flitter we have users who post tweets, follow other users, read tweets, etc, as described below. You have to implement the following Tasks:

T.1 **Create user account**: We need the user-name of the user and the password. Prompt for a new user-name, if the proposed one is taken.

T.2 **Reset database**: Keep the tables, but delete all their records.

T.3 **Login**: Obvious - your system would ask for a user-name and password, and authenticate the user or deny further access.

T.4 **Timeline**: After log-in, the main page should display the user's timeline. That is, in chronological order (newest first), your system should show 1) all the posts of the followees of the current user and 2) all the posts of the current user.

T.5 **Post a tweet**: Once authenticated, a user should be able to post a tweet. Make sure the body obeys the specifications above (at most 140, printable characters).

T.6 **Search for a user**: Given query sub-string (say *mic*) print all the user-names that contain this sub-string (e.g., *michael*, *mickey*, *karmic*)

T.7 **Check if a given user follows another user**: Obvious.

T.8 **Follow a user**: Your system should allow a user to follow an unlimited number of users, but not himself.

T.9 **Unfollow a user**: The reverse of the previous task. Print a warning, when user 'U' tries to unfollow someone that 'U' is not following.

T.10 **Recommend users to follow**: For a given user 'U', your system should recommend users to follow. In short, we want to recommend the *two-step-away* followees, that is, the followees of the followees, in some *priority* order, and, of course, excluding users that 'U' is following already. 'Priority' is the count of paths connecting user 'U' to the recommended user 'V'.

For example, consider the who-follows-whom graph of Figure 1. For user 'U1', your system should recommend 'U3', 'U4', in this order. The reason is that

a) the followees of the followees of 'U1' are ('U3', 'U6', 'U3', 'U4', 'U6');
b) 'U3' appears twice (paths through 'U2' and 'U5'), and gets priority over 'U4' (who has a single path, through 'U5' only) and
c) 'U6' disqualifies since 'U1' already follows him.

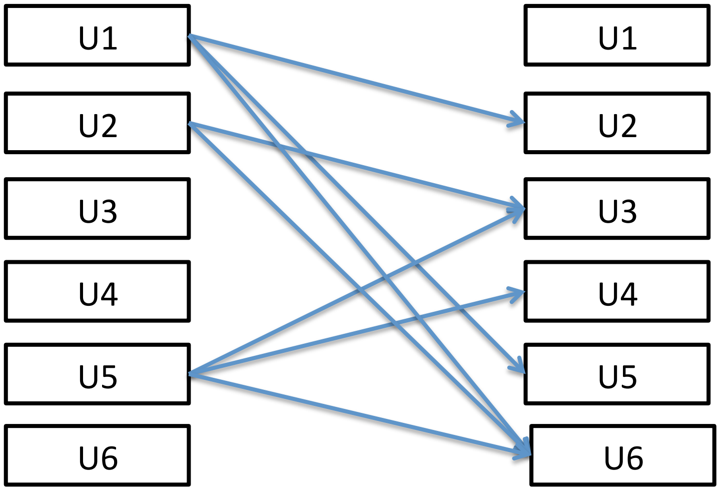

Figure 1: Sample who-follows-whom graph. Followers (on the left) and the same users as followees, on the right.

T.11 **List the followers & followees**: For a given user 'U', list all his followers and followees.

T.12 **List all tweets of a user**: For a given username 'X', retrieve all his tweets, newest tweet first.

T.13 **String search**: Given a string (say, "*quake*"), retrieve all tweets that contain it in their body (eg. "*OMG earthquake in oregon!*"). String search should be case sensitive.

T.14 **User statistics**: For a given user-name ’U’, display the following counts, or print a warning if ’U’ does not exist.

(a) The number of followers
(b) The number of followees
(c) ~~The number of tweets~~ (You don’t need to implement this)

T.15 **Global statistics** The system should be able to list heavy users, and specifically:

(a) **Most popular**: List of K users with the most followers
(b) **Most active**: List of K users with the most posts
(c) **Most connected (’hubs’)**: List of K users with the highest degree (sum of followers plus followees).

In all the above cases, treat K as a parameter. In case of a tie in K-th place, report only K users, favoring the alphabetically first. I.e., if K=1, and ’bob’ and ’alice’ tie in first place, report only ’alice’.

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

No need to separate your answers

We follow the design methodology of Roussopoulos and Yeh, summarized in the lecture foils: `http://www.cs.cmu.edu/~christos/courses/dbms.S14/slides/19methodology.pdf`. Thus, we have the following deliverables:

- Phase 1: Environment and Requirement Analysis, System Analysis and Specication, Conceptual Modeling, and Task Emulation.
- Phase 2: Implementation and Testing.

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

(a) [**2 points**] The top-level information flow diagram, along with the system boundary.

(b) [**1 point**] The list of documents.

(c) [**2 points**] The document forms, including the assumptions and design decisions you made.

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

(e) [**2 points**] The relational schema.

(f) [**10 points**] SQL DDL statements that create the above schema - make sure you include all constraints (primary key, foreign key, etc)

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

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**] Download `http://www.cs.cmu.edu/~christos/courses/dbms.S14/hws/HW7/hw7.zip`.

(b) [**0 points**] List of changes, if **any**, to your Phase 1 schema.

(c) [**5 points**] Listing of your code, below.

(d) [**5 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)

(e) [**50 points**] The actual implementation: In `hw7.zip` above, we are providing you with a basic implementation of the web application, which includes calls to the database that you have designed but does not implement the required functionality. Your job will be to connect the provided code with the database that you have designed, in Phase 1 so that you have a working website that performs Tasks T.1-T.15.

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 Account creation	3
T.2 Reset database	4
T.3 Login	3
T.4 Timeline	4
T.5 Tweet	3
T.6 Search for user	3
T.7 Check if follows	1
T.8 Follow	3
T.9 Unfollow	3
T.10 Recommend	8
T.11 Get followers & followees	3
T.12 Get all tweets of a user	3
T.13 String search	3
T.14 User statistics	3
T.15 Global statistics	3

Table 1: Point breakdown

(f) [**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. Thus, your code should sanitize the arguments to the SQL queries from all escape characters. *HINT*: check the PHP function `pg_escape_string()`.

3 Setup and Hints for Phase 2

3.1 VERY IMPORTANT: Set up

As in the previous homeworks, you will use Postgres; the full documentation for Postgres is at http://www.postgresql.org/docs/.

For Phase 2, you need access to a webserver. You will use the webserver that is provided by the CMU Computer Club, which allows web content to be served directly from your Andrew s home directory in AFS without manual publishing. More details are in here: http://www.club.cc.cmu.edu/doc/contribweb.php.

To get started, please follow the following steps:

1. Log-in here http://my.contrib.andrew.cmu.edu using your Andrew ID, in order to set up your account for the Contributed server.
2. Read carefully the information provided by the Computer Club on how to set up your Postgres account and access the database on their server:
   http://www.club.cc.cmu.edu/doc/contribweb/sql.php.
3. Unzip `hw7.zip` and copy its contents on a folder called `flitter_s14` inside the `www` directory on your andrew AFS account.
4. Edit `config.php` to have the appropriate parameters for your site (the web address and your Postgres database information). Also, make sure that your files are set to be both read and executed by anyone (`chmod +rx`).
5. Open your favorite web browser and go to `http://www.contrib.andrew.cmu.edu/~andrew_id/flitter_s14` replacing `andrew_id` with your own Andrew ID. Make sure that you can see the Login screen of Flitter.
6. You are ready to start implementing `functions.php`!

3.2 IMPORTANT: Customization

- `functions.php`: The provided `hw7.zip` includes a source file `functions.php` which contains the definitions of the API functions that you will have to implement. This is the main file that you need to edit.
- `config.php`: Moreover, you will have to edit `config.php` and fill in your own log-in information for the database, as well as your host information.

3.3 Strong hints

Running prototype : See `http://gs11696.sp.cs.cmu.edu/~abeutel/flitter/`. for a prototype with all required functions. You are strongly encouraged to use this website in order to guide your own implementation, especially in cases when you are not sure if your design/implementation is working correctly.

End of Homework 7

Some sample unit tests : Additionally, we provide you with a set of unit tests, similar to the ones that we will use for grading Phase 2. These unit tests load data to the database and then evaluate functions of the API that you have implemented. You are welcome to develop your own unit tests based on the provided ones. To run the tests, follow these steps:

1. Go to your Flitter website (any page) on your browser.
2. Open the javascript console on your browser. Here is how to do it in different browsers:

   http://webmasters.stackexchange.com/questions/8525/how-to-open-the-javascript-console-in-different-browsers
3. Running `generate_data()` resets the database and then loads the test data.
4. Running `test_basic_functionality()` runs the unit tests and checks the correct operation of the implemented functions.

3.4 *Optional* documentation

For this phase, you will have to write some PHP, and in case you want to develop your own unit tests, some javascript and JSON object manipulation in PHP. Here we point you to some useful links:

1. PHP manual http://www.php.net/
2. Arrays in PHP http://www.php.net/manual/en/language.types.array.php
3. PHP and Postgres http://www.php.net/manual/en/book.pgsql.php.
4. Javascript tutorial http://www.w3schools.com/js/DEFAULT.asp
5. JSON in PHP http://www.php.net/manual/en/ref.json.php