CS 1555/2055 – Database Management Systems (Fall 2019)
Dept. of Computer Science, University of Pittsburgh

Project: PittSocial

Release Date: Oct. 21, 2019
Team Formation Due: 8:00 PM, Oct. 24, 2019
Phase 1 Due: 8:00 PM, Nov. 7, 2019
Phase 2 Due: 8:00 PM, Nov. 21, 2019
Phase 3 Due: 8:00 PM, Dec. 7, 2019
Project Demos: Dec. 11-13, 2019

Purpose of the project

The primary goal of this project is to implement a single Java application program that will operate **PittSocial**, a Social Networking System for the University of Pittsburgh. The core of such a system is a database system. The secondary goal is to learn how to work as a member of a team which designs and develops a relatively large, real database application.

You must implement your application program using Java, PostgreSQL, and JDBC. The assignment focuses on the database component and not on the user interface. Hence, NO HTML or other graphical user interface is required for this project and carry no bonus points.

Team Formation Deadline: 8:00 PM, Oct. 24, 2019

- You are asked to form teams of three. You need to notify the instructor and the TA by emailing team information to cs1555-staff@cs.pitt.edu (Remember that you need to send the email from your pitt email address, otherwise the email will not go through). The email should include the names of the team members and should be CC-ed to both team members (otherwise the team will not be accepted).
- The deadline to declare teams is **8:00 PM, Thursday, Oct. 24, 2019** . If no email was sent before this time, you will be assigned to a team by the instructor. Each team will be assigned a unique number which will be sent by email upon receiving the notification email.
- It is expected that all members of a team will be involved in all aspects of the project development and contribute equally. Division of labor to data engineering (db component) and software engineering (java component) is not acceptable since each member of the team will be evaluated on both components.

Phase 1: The PittSocial database schema and example data
Due: 8:00 PM, Nov. 7, 2019

Your PittSocial database includes the standard basic information found in a social networking system such as user profiles, friends, etc. It will have the following relations. You are required to define all of the structural and semantic integrity constraints and their modes of evaluation. For both structural and semantic integrity constraints, you must state your assumptions as comments in your database creation script. Semantic integrity constraints involving multiple relations should be specified using triggers. **Please do not change table and attribute names.**

- **profile** (userID, name, email, password, date_of_birth, lastlogin)
  Stores the profile and login information for each user registered in the system.
  Datatype

userID: integer

name: varchar(50)

email: varchar(50)

password: varchar(50)

date_of_birth: date

lastlogin: timestamp

- **friend** (userID1, userID2, JDate, message)
  Stores the friends lists for every user in the system. The JDate is when they became friends, and the message is the message of friend request.
  Datatype

  userID1: integer

  userID1: integer

  JDate: date

  message: varchar(200)

- **pendingFriend** (fromID, toID, message)
  Stores pending friends requests that have yet to be confirmed by the recipient of the request.
  Datatype

  fromID: integer

  toID: integer

  message: varchar(200)

- **message** (msgID, fromID, message, toUserID, toGroupID, timeSent)
  Stores every message sent by users in the system. Note that the default values of ToUserID and ToGroupID should be NULL.
  Datatype

  msgID: integer

  fromID: integer

  message: varchar(200)

  toUserID: integer

  toGroupID: integer

  timeSent: timestamp

- **messageRecipient** (msgID, userID)
  Stores the recipients of each message stored in the system.
  Datatype

  msgID: integer

  userID: integer

- **group** (gID, name, limit, description)
  Stores information for each group in the system.
  Datatype

  gID: integer

name: varchar(50)

limit: integer

description: varchar(200)

- **groupMember** (gID, userID, role)
  Stores the users who are members of each group in the system. The 'role' indicate whether a user is a manager of a group (who can accept joining group request) or a member.
  Datatype

  gID: integer

  userID: integer

  role: varchar(20)

- **pendingGroupMember** (gID, userID, message)
  Stores pending joining group requests that have yet to be accept/reject by the manager of the group.
  Datatype

  gID: integer

  userID: integer

  message: varchar(200)

Once you have created a schema and integrity constraints for storing all of this information, you should generate sample data to insert into your tables. Generate the data to represent at least 100 users, 200 friendships, 10 groups, and 300 messages.

Phase 2: A JDBC application to manage PittSocial
Due: 8:00 PM, Nov. 21, 2019

You are expected to do your project in Java interfacing PostgreSQL server using JDBC. You can develop your project on your local environments.

For all tasks, you are expected to check for any errors reported by the DBMS (PostgreSQL), and provide appropriate success or failure feedback to user. Further, be sure that your application carefully checks the input data from the user and avoids SQL injection.

Attention must be paid in defining transactions appropriately. Specifically, you need to design the **SQL transactions** appropriately and when necessary, use the concurrency control mechanism supported by PostgreSQL (e.g., isolation level, locking models) to make sure that inconsistent states will not occur. Assume that multiple requests for changes of PittSocial can be made on behalf of multiple different users concurrently. For example, it could happen when a group manager A is trying to add a new group member B while another group manager C is trying to add a new group member D at the same time (i.e., concurrently).

The objective of this project is to familiarize yourself with all the powerful features of SQL/PL which include functions, procedures and triggers. Recall that triggers and stored procedures can be used to make your code more efficient besides enforcing integrity constraints.

You application should implement the following functions for managing PittSocial. The user has three options at the top level of the UI: createUser, login, and exit.

1. **createUser**
   Given a name, email address, and date of birth, add a new user to the system by inserting a new entry into the *profile* relation. userIDs should be auto-generated.

2. **login**
   Given email and password, login as the user in the system when an appropriate match is found.

3. **initiateFriendship**
   Create a pending friendship from the logged-in user to another user based on userID. The application should display the name of the person that will be sent a friends request and the user should be prompted to enter a message to be sent along with the request. A last confirmation should be requested of the user before an entry is inserted into the *pendingFriend* relation, and success or failure feedback is displayed for the user.

4. **createGroup**
   Given a name, description, and membership limit, add a new group to the system, add the user as its first member with the role manager. gIDs should be auto-generated.

5. **initiateAddingGroup**
   Given a group ID and a message, create a pending request of adding the logged-in user to the group by inserting a new entry into the *pendingGroupMember* relation.

6. **confirmRequests**
   This task should first display a formatted, numbered list of all outstanding friend requests (as well as group requests for the groups where the user is a group manager) with associated messages. Then, the user should be prompted for a number of the request he or she would like to confirm or given the option to confirm them all. The application should move the request from the appropriate *pendingFriend* or *pendingGroupMember* relation to the *friend* or *groupMember* relation (adding group members should not violate the group's membership limit). The remaining requests which were not selected are declined and removed from *pendingFriend* and *pendingGroupMember* relations.

7. **sendMessageToUser**
   With this the user can send a message to one friend given the user's userID. The application should display the name of the recipient and the user should be prompted to enter the text of the message, which could be multi-lined. Once entered, the application should "send" the message to the user by adding an appropriate entry into the *message* relation (msgIDs should be auto-generated) and **use a trigger** to add a corresponding entry into the *messageRecipient* relation. The user should lastly be shown success or failure feedback.

8. **sendMessageToGroup**
   With this the user can send a message to a recipient group given the group ID, if the user is within the group. Every member of this group should receive the message. The user should be prompted to enter the text of the message, which could be multi-lined. Then the application should "send" the message to the group by adding an appropriate entry into the *message* relation (msgIDs should be auto-generated) and **use a trigger** to add corresponding entries into the *messageRecipient* relation. The user should lastly be shown success or failure feedback. Note that if the user sends a message to one friend, you only need to put the friend's userID to ToUserID in the table of *message*. If the user wants to send a message to a group, you need to put the group ID to ToGroupID in the table of *message* and **use a trigger** to populate the *messageRecipient* table with proper user ID information as defined by the *groupMember* relation.

9. **displayMessages**
   When the user selects this option, the entire contents of every message sent to the user should be displayed in a nicely formatted way.

10. **displayNewMessages**
    This should display messages in the same fashion as the previous task except that only those messages sent since the last time the user logged into the system should be displayed.

11. **displayFriends**
    This task supports the browsing of the logged-in user's friends' profiles. It first displays each of the user's friends' names and userIDs. Then it allows the user to either retrieve a friend's entire profile by entering the appropriate userID or exit browsing and return to the main menu by entering 0 as a userID. When selected, a friend's profile should be displayed in a nicely formatted way, after which the user should be prompted to either select to retrieve another friend's profile or return to the main menu. The matching is case sensitive.

12. **searchForUser**
    Given a string on which to match any user in the system, any item in this string must be matched against the "name" and "email" fields of a user's profile. That is if the user searches for "xyz abc", the results should be the set of all profiles that match "xyz" union the set of all profiles that matches "abc".

13. **threeDegress**
    Given a userID, find a path, if one exists, between the logged-in user and that user with at most 3 hop between them. A hop is defined as a friendship between any two users.

14. **topMessages**
    Display the top k users with respect to the number of messages sent to the logged-in user plus the number of messages received from the logged-in user in the past x months. x and k are input parameters to this function.

15. **logout**
    The function should return the user to the top level of the UI after marking the time of the user's logout in the user's "lastlogin" field of the *profile* relation.

16. **dropUser**
    This function should logout the user and remove the user together with all of his/her information from the system. When a user is removed, the system should **use a trigger** to delete the user from the groups he or she is a member of. The system should also **use a trigger** to delete any message whose sender and all receivers are deleted. Attention should be paid to handling integrity constraints.

17. **exit**
    This option should cleanly shut down and exit the program.

Since this is a database course, if a function or a task can be implemented using the database approach, you should use that approach. You might lose some points if you use the Java approach. Nontrivial tasks (e.g., "Three Degrees of Separation") can all be implemented in the database approach using stored procedures or functions.

Phase 3: Bringing it all together
Due: 8:00 PM, Dec. 7, 2019

The primary task for this phase is to create a Java driver program to demonstrate the correctness of your social network backend. The driver program needs to call all of the above functions and display the content of the affected tables afterwards. It may prove quite handy to write this driver

as you develop the functions as a way to test them. You may also wish to reuse your data generation code to dynamically generate a large number of function calls within your driver.
Now this may not seem like a lot for a third phase (especially since it is stated that you may want to do this work alongside Phase 2), the reasoning for this is to allow you to focus on incorporating feedback from the TA regarding Phases 1 and 2 into your project as part of Phase 3. This will be the primary focus of Phase 3.

Project Submission

Phase 1 Phase 1 should produce the following files:
schema.sql the script to create the tables.
trigger.sql the script to create other database objects, e.g., trigger, functions, procedures, etc.
insert.sql the script to insert sample records into the database, that test triggers and IC etc.

Note that after the first phase submission, you should continue working on your project without waiting for our feedback. Furthermore, you should feel free to correct and enhance your SQL part with new views, functions, procedures etc.

Phase 2 Phase 2 should produce, in addition to Phase 1, the following files:
PittSocial.java the file containing your main class and all the functions.
Readme.txt the file with instructions of how to compile and run your system.

Phase 3 Phase 3 should produce, in addition to Phase 2, the following file:
Driver.java the driver file to show correctness of your functions.
Readme.txt the file with instructions of how to compile and run your driver program.

The project will be collected by the TA via the private GitHub repository that is shared with only your instructor (GitHub username: panos4pittcs), TA (GitHub username: xzhangedu) and your team members. To turn in your code, you must do three things by each deadline:

1. Make a commit to your project repository that represents what should be graded as your team's submission for that phase. The message for this commit should be "Phase X submission" where X is 1,2 or 3.

2. Push that commit to the GitHub repository that you have shared with the instructor and TA.

3. Send an email to cs1555-staff@cs.pitt.edu with the title "[CS1555] Project Phase X submission" that includes a link to your GitHub repository and Git commit ID for the commit that should be graded. Commit ID can be found on GitHub or as part of output of "gitlog" command.

Each team may submit multiple times for each phase. The feedback/grading for each phase will be based on the last commit before the corresponding deadline. *NO late submission is allowed.*

Grading

The project will be graded on correctness (including transaction usage), robustness (error-checking, i.e., it produces user-friendly and meaningful error messages) and readability. You will not be graded on efficient code with respect to speed although bad programming will certainly lead to incorrect programs. For Phases 1 and 2, feedback will be given on how to improve your program based on the project requirement. Your team's project grade is based on the final program (i.e., Phase 3 submission) and the project demo. Programs that fail to compile or run or connect to the database server earn zero and *no partial points.*

Academic Honesty

The work in this assignment is to be done *independently* by each team. Discussions with other students or teams on the project should be limited to understanding the statement of the problem. Cheating in any way, including giving your work to someone else will result in an F for the course and a report to the appropriate University authority.

Enjoy your class project!