CS 1555/2055 – Database Management Systems (Fall 2017)
Dept. of Computer Science, University of Pittsburgh

# Project: Social@Panther

Release Date: Oct. 27, 2017 — Phase 1 Due: 11:59 PM, Nov. 16, 2017
Team Formation Due: 11:59 PM, Nov. 2, 2017 — Phase 2 Due: 11:59 PM, Dec. 1, 2017
Release Date: Oct. 30, 2017 — Phase 3 Due: 11:59 PM, Dec. 10, 2017

---

**Purpose of the project**

The primary goal of this project is to implement a single Java application program that will operate **Social@Panther**, 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, Oracle, 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.

**Team Formation Deadline: 11:59 PM, Nov. 2, 2017**

**Phase 1: The Social@Panther database schema and example data**
**Due: 11:59 PM, Nov. 16, 2017**

Your Social@Panther 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.

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

  userID: varchar2(20)

  name: varchar2(50)

  password: varchar2(50)

  date_of_birth: date

  lastlogin: timestamp

- **friends** (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: varchar2(20)

  userID1: varchar2(20)

  JDate: date

  message: varchar2(200)

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

  fromID: varchar2(20)

  toID: varchar2(20)

  message: varchar2(200)

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

  msgID: varchar2(20)

  fromID: varchar2(20)

  message: varchar2(200)

  toUserID: varchar2(20)

  toGroupID: varchar2(20)

  dateSent: date

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

  msgID: varchar2(20)

  userID: varchar2(20)

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

  gID: varchar2(20)

  name: varchar2(50)

  description: varchar2(200)

- **groupMembership** (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 not.
  Datatype

  gID: varchar2(20)

  userID: varchar2(20)

  role: varchar2(20)

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

  gID: varchar2(20)

  userID: varchar2(20)

  message: varchar2(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 FaceSpace**
**Due: 11:59 PM, Dec. 1, 2017**

You are expected to do your project in Java interfacing Oracle server using JDBC. You can develop your project on either unixs, class3 or Windows environments, but we will only test your code in unixs or class3 machine. So even though your code works in a windows environment, you should make sure it works on unixs or class3 before you submit your project.

For all tasks, you are expected to check for an properly react to any errors reported by the DBMS (Oracle), 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 Oracle (e.g., isolation level, locking models) to make sure that inconsistent states will not occur. Assume that multiple request for changes to Social@Panther can be made on behalf of multiple different users concurrently. For example,

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 Social@Panther.

1. **createUser**
   Given a name, email address, and date of birth, add a new user to the system by inserting as new entry in the *profile* relation.

2. **Login**
   Given userID 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 *pendingFriends* relation, and success or failure feedback is displayed for the user.

4. **confirmFriendship**
   This task should first display a formatted, numbered list of all outstanding friends and group requests with an 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 *pendingFriends* or *pendingGroupmembers* relation to the *friends* or *groupMembership* relation. The remaining requests which were not selected are declined and removed from *pendingFriends* and *pendingGroupmembers* relations.

5. **displayFriends**
   This task supports the browsing of the user's friends and of their friends' profiles. It first displays each of the user's friends' names and userIDs and those of any friend of those friends. 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.

6. **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.

7. **initiateAddingGroup**
   Given a user and a group, create a pending request of adding to group (if not violate the group's membership limit). The user should be prompted to enter a message to be sent along with the request and inserted in the *pendingGroupmembers* relation.

8. **sendMessageToUser**
   With this the user can send a message to one friend given his 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 message should be "sent" to the user by adding appropriate entries into the *messages* and *messageRecipients* relations **by creating a trigger**. The user should lastly be shown success or failure feedback.

9. **sendMessageToGroup**
   With this the user can send a message to a recipient group, 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 created new message should be "sent" to the user by adding appropriate entries into the *messages* and *messageRecipients* relations **by creating a trigger**. 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 *messages*. If the user wants to send a message to a group, you need to put the group ID to ToGroupID in the table of *messages* and **use a trigger** to populate the *messageRecipient* table with proper user ID information as defined by the *groupMembership* relation.

10. **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.

11. **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.

12. **searchForUser**
    Given a string on which to match any user in the system, any item in this string must be matched against any significant field 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 two users (A and B), find a path, if one exists, between A and B with at most 3 hop between them. A hop is defined as a friendship between any two users.

14. **topMessages**
    Display top K who have sent to received the highest number of messages during for the past x months. x and K are input parameters to this function.

15. **dropUser**
    Remove a user and all of their information from the system. When a user is removed, the system should delete the user from the groups he or she was a member of **using a trigger**. Note: messages require special handling because they are owned by both sender and receiver. Therefore, a message is deleted only when both he sender and all receivers are deleted. Attention should be paid handling integrity constraints.

16. **LogOut**
    This option should cleanly shut down and exit the program after marking the time of the user's logout in the *profile* relation,

## Phase 3: Bringing it all together
## Due: 11:59 PM, Dec. 10, 2017

The primary task for this phase is to create a Java driven program to demonstrate the correctness of your social network backend by calling all of the above functions.

## Project submission

Milestone 1 The first milestone should contain only the SQL part of the project. Specifically, the scripts to create the database schema along with the definition of the triggers, and any stored procedures or functions that you designed. If you wish to receive more feedback, feel free to include any or all of your SQL queries in your repository.

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.

Milestone 2 The second milestone should contain, in addition to the SQL part, the Java code.

Milestone 3 The third milestone should contain, in addition to the second milestone, the driver and benchmark. *NO late submission is allowed.*

The project will be collected by the TA via the GitHub repository that you share with only your instructor, TA and your team member. 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 group's submission for that milestone. 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.

For the project, each group can **only submit once per milestone per group**.

## Group Rules

- You are asked to form groups of two. You need to notify the instructor and the TA by emailing group 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 **Thursday, 11:59 PM, Nov. 2, 2017** . If no email was sent before this time, you will be assigned a team member by the instructor. Each group will be assigned a unique number which will be sent by email upon receiving the notification email.
- It is expected that both members of a group 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 group will be evaluated on both components.

## 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. 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!*