NYU Tandon School of Engineering
Computer Science and Engineering
CS-UY 3083, Introduction to Database Systems, Spring 2019
Prof Frankl

COURSE PROJECT

Project Overview

The course project this semester is **Finstagram**, a web application for sharing photos. Finstagram gives users more privacy than many photo sharing sites by giving them more detailed control over who can see which photos they post. The focus of the project will be on storing data about who posted which photos and who has permission to view them, tag others in them, see who's tagged in what, etc.

Users will be able to log in, post photos, view (some of) the photos posted by others (public photos, photos posted by people they are `*following*`, and close friends, detailed below); `*tag*` photos items with handles of people, etc. In part 1 of the project, you will design an ER diagram for the database. In part 2, you will convert my E-R diagram, which I will post after Part 1 is due, to a relational schema and write table definitions in SQL. Part 3 is a milestone to guarantee that you're making progress: using my table definitions, which I will post after Part 2 is due, you will write and test some of your SQL queries and some application source code. Part 4 will be the most work: you will revise your queries from part 3, if necessary, and write and test the rest of the application code for the system. A detailed specification will be provided shortly after part 2 is due. Near the end of the semester you will schedule a demo/test session in which your Finstagram application will be tested.

Partners, etc

You may work alone or in a **team of up to 3 people**. For parts 3 and 4 of the project, there will be some extra requirements for teams, with the number of extra requirements increasing as team size increases. Most likely, there will be a core of features that everyone must implement (e.g. posting a photo, seeing all the photos that are visible to the user who's logged in, etc.) and an additional *2n* features required where *n* is the number of team members. You may work on part 1 alone then team up for later parts. There will be some deadline set for team formation, around when part 2 or 3 is due. Note that each team member is expected to contribute roughly equally and each team member is responsible for understanding the entire system. There may be some quiz or exam questions about the project.

The total project grade will be 25% of your course grade. Part 1 counts for about 20% of the project grade. Part 2 counts for about 15% of the project grade. Part 3 counts for about 15% of the project grade. There may also be a quiz or exam question(s) based on the project.

PART 1: Due mid February (see NYU Classes for exact date and time)

Finstagram allows users to have *followers* with whom they can share photos and also *close friends* who have special permissions to view certain posts. For example, if Brittany has a private Finstagram account and Kevin is following Brittany, he will be able to view some of her posts. However, Brittany can post a photo that only certain of her `*close* friends` can view. If Brittany did not add Kevin to this group of close friends, then Kevin will not be able to view that photo. People who have access to view a photo will also be able to like and comment on that photo. In addition, the person posting the photo can write a caption and tag the photo with people that may be in the photo.

For Part 1, you will draw an ER diagram modeling the data needed for the system. It should include entity sets (possibly including one or more weak entity sets) representing **Person, CloseFriendGroup,** and **Photo** as well as several relationship sets.

Details of the data model:

Each **User** has a unique *username*, a *password*, a *name*, which consists of a *first name* and *last name*, an *avatar filepath*, a *biography* message for their profile, and an *isPrivate* indicator, which will be used in determining which of their photos can be viewed by whom.

Each **Photo** has a unique *ID*, a *date*, a *filepath*, an *allFollowers* indicator (which will be used later to determine who can view the photo), and a *caption*. Each photo is **OwnedBy** one Person.

A Person can have zero or more **CloseFriendsGroup**s. Each group has a *group_name* and is *owned by* exactly one user. Different people can have groups with the same group_name, however, an individual user cannot have more than one group with the same group_name. Each CloseFriendsGroup has zero or more members, i.e. People who **BelongTo** the *CloseFriendsGroup*. Each Person can belong to any number of CloseFriendsGroups. For example, Brittany can add Kevin and John to her `DB classmates' CloseFriendsGroup and add John and Jane to her `WorkOut' CloseFriendsGroup. Sam could have a different CloseFriendsGroup, also called 'WorkOut'.

A photo can be **SharedWith** zero or more CloseFriendsGroups. Photos that are posted with allFollowers == True will be visible to all of the followers of the person who posted the photo, whereas photos that are posted with allFollowers == False will only be visible to people who belong to the CloseFriendsGroup(s) that the photo is SharedWith.

A Person can **Like** a Photo. When a person likes a photo, the database should also store a timestamp and a comment. (At this ER design stage, we will not worry about prohibiting a person from liking a photo that they cannot view; this constraint will be included later).

A Photo can be **Tagged** with people who are in the photo. For example, if Yanny posts a Photo of her and Laurel at the Empire State Building, the photo can be Tagged to indicate that Laurel is in the photo. A tag will not be active until the person who is tagged accepts the tag.

A Person can **Follow** another Person to view their photos (except those intended only for close friends). A Person is not officially Following another User until the Follow request gets accepted.

What You Should Do

Design an ER diagram for Finstagram. When you do this, think about: which information should be represented as attributes, which as entity sets or relationship sets? Are any of the entity sets weak entity sets? If so, what is the identifying strong entity set? What is the primary key (or discriminator) of each entity set? What additional attributes are needed to keep track of data about the status of requests to follow someone, tag someone in a photo, etc? What are the cardinality constraints on the relationship sets? Draw the ER diagram neatly. You may draw it by hand or use a drawing tool.

Your diagram should fit neatly onto one page. Most, if not all, of the relevant entity sets and relationship sets are highlighted in **bold** and most of the attributes are in *italics* in the project description, above.

Note: At this point you should NOT worry about enforcing rules about who can do what with which photos, etc. For example, you do not need to assure in the ER diagram that a Person can only comment on photos that are shared with a CloseFriendGroup that the person belongs to; we'll add that later with additional constraints on the relational schema in part 2 and/or with queries and /or application code in parts 3 and 4.

See next page for part 2

PART 2

Use the ER diagram posted (my solution to Part 1, with the changes in cardinality constraints that are noted in red above):

1. Make one small change to the ER: Instead of including comment and timestamp as attributes of Like, create a different relationship set allowing People to make comments about Photos; include the comment text and a time stamp as attributes.

2. Using the rules we studied for converting an ER diagram to a relational database schema, draw a relational schema diagram. Remember to underline primary keys and use arrows from referencing to referenced attributes to denote foreign keys. Note that you will need to rename a few attributes when the basic rules would result in two attributes with the same name in the same table. In particular, *username* will end up occurring in several different contexts. Choose meaningful names.

3. Write SQL CREATE TABLE statements for each table. Use INT for IDs and Boolean flags, DATETIME or something similar for dates, timestamps, etc (see https://dev.mysql.com/doc/refman/8.0/en/date-and-time-types.html), Boolean or INT, and VARCHAR for the other attributes. Remember to include PRIMARY KEY and FOREIGN KEY constraints.

4. Write a query to find the IDs of Photos that are SharedWith any CloseFriendGroup to which the user with username “Ann” belongs.

What / How to hand in:
1. Via NYU Classes: A plain text file that includes all of your CREATE TABLE statements (3) and the query (4), AND
2. Via GradeScope: A pdf file that includes
   a. The schema diagrame (2)
   b. The CREATE TABLE statement (3)
   c. The query (4)

NOTE: Mark which is which part is where when you hand it in via GradeScope.
Each file should have the names of all team members. Use the Group Handin option on GradeScope. On NYU Classes only one team member should hand the work in.

PART 3 and 4

In parts 3 and 4 of the project, you will use the table definitions I will post (solution to part 2) along with any additional table definitions you need, to implement application code for Finstagram as a web-based application. You may use Python, PHP, Java, node.js, Go, or C#. If you'd like to use some other language, check with me by 3/15. You must use prepared statements if your application language supports them.

Part 3 (Due 4/5/19) is a milestone in which you must show that you've written and tested code for at least *n* of the "use cases" shown below (*other than the login, which we'll provide for you*), where *n* is the number of people in your group. Each person on your team should write the code for one of these use cases and the whole team should review it and understand it.

Part 4 is the completed project, due on **4/24 (small penalties for a few days after that; increasing penalties thereafter)**. You will hand in your code, a short write up (details coming soon), proposals for any extra features that are not on the list below (. You will also schedule a demo session in which one of the TAs will run through a series of tests with you.

Your Finstagram implementation should allow a user to log in; view photos that she has access to, i.e. photos that were were posted by people she's following or that are shared with CloseFriendGroups to which she belongs; post photos and designate who can view them; request to follow others; accept or decline requests to follow her; propose to tag others in photos that she has access to; accept or decline proposed tags. In addition, you will propose and add some other features, as described below.
Assume each user has already registered and created a password, which is stored as an SHA-2 hash. (We will provide Python/Flask code to manage user registration and login).

A photo is *visible to* a user U if either

- U has been accepted as a follower by the owner of the photo, or
- The photo is shared with a CloseFriendGroup to which U belongs

Remember that a CloseFriendGroup is identified its groupName *along with* the groupOwner (the username of the owner of the group). We will assume that in the initial database, the owner of each CloseFriendGroup is a member of that CloseFriendGroup. When modifying the database, Finstagram should maintain that, by adding the user as a member of each CloseFriendGroup she creates.

Finstagram should support the following use cases:

Login: The user enters her username and password. Finstagram will add "salt" to the password, hash it, and check whether the hash of the password matches the stored password for that e-mail. If so, it initiates a session, storing the username and any other relevant data in session variables, then goes to the home page (or provides some mechanism for the user to select her next action.) If the password does not match the stored password for that username (or no such user exists) Finstagram informs the user that the the login failed and does not initiate the session. **We will supply Python/Flask**

code for this. If you're using a different implementation language, you'll need to write this yourself.

The remaining use cases require the user to be logged in.

1. **View visible photos and info about them:** Finstagram shows the user the photoID, photoOwner's name, and caption of photos that are visible to the her, arranged in reverse chronological order. Display the photo or include a link to display it.
   Along with each photo the following further information should be displayed or there should be an option to display it: timestamp, the first names and last names of people who have been tagged in the photo (taggees), provided that they have accepted the tags (Tag.acceptedTag == true)

2. **Post a photo:** User enters the relevant data and a link to a real photo and a designation of whether the photo is visible to all followers (allFollowers == true) or only to members of designated CloseFriendGroups (allFollowers == false). Finstagram inserts data about the photo (including current time, and current user as owner) into the Photo table. If the photo is not visible to all followers, Finstagram gives the user a way to designate CloseFriendGroups that the user belongs to with which the Photo is shared.

3. **Manage Follows:**
   a. User enters the username of someone they want to follow. Finstagram adds an appropriate tuple to Follow, with acceptedFollow == False.
   b. User sees list of requests others has made to follow them and has opportunity to accept, by setting acceptFollow to True or to decline by removing the request from the Follow table.

4. **Manage tag requests:**
   a. Finstagram shows the user relevant data about photos that have proposed tags of this user (i.e. user is the taggee and acceptedTag is false.) User can choose to accept a tag (change acceptedTag to true), decline a tag (remove the tag from Tag table), or not make a decision (leave the proposed tag in the table with acceptedTag == false.)

5. **Tag a photo:** Current user, whom we'll call x, selects a photo that is visible to her and proposes to tag it with username y
   i. If the user is self-tagging (y == x), Finstagram adds a row to the Tag table:
      (x, photoID, true)
   ii. else if the photo is visible to y, Finstagram adds a row to the Tag table:
      (y, photoID, false)
   iii. else if photo is not visible to y, Finstagram doesn't change the tag table and prints some message saying that she cannot propose this tag.

6. **Add friend:** User selects an existing CloseFriendGroup that they own and provides username of someone she'd like to add to the group. Finstagram checks whether there is exactly one person with that name and updates the Belong table to indicate that the selected person is now in the CloseFriendGroup. Unusual situation such as the person already being in the group should be handled gracefully.

Extra Features:

Your choice: If you are working with others, you must add **two extra features per team member:**

- Working alone: No extra features required
- Two person team: 4 extra features
- Three person team: 6 extra features

These individual features should be designed and implemented by an individual team member who will get an individual grade for this work. You may help your teammates understand what they should do and you may review and test their code, but they should have primary responsibility for planning and implementing their features. Each additional feature must involve interactions with the database and must be relevant to this particular project. *Generic features like a registration page are not acceptable*. For each feature you must write a description including:

a. The name of the team member who is primarily in charge of implementing this feature
b. The queries (and any other SQL statements) used in your implementation of the feature
c. A clear indication of where to find the application source code for the feature. Include plenty of comments and/or a few sentences explaining the code.
d. One or more screenshots demonstrating the feature, showing how it appears in the browser; Also show the relevant data that's in the database when you execute this demonstration (before and after if the feature changes the data), either as screenshots or as text.
e. For features other than those I suggested (7 -- 11), also include:
    i. What the feature is (in the style of the 6 requirements above)
    ii. A sentence or two on why this is a good feature to add to Finstagram
    iii. A clear explanation of any changes to the database schema that are needed (including additional tables, additional attributes, or additional constraints)

Note: *If your extra features require extensive modification to the database schema, please develop and test them as a separate branch of your project, so that you'll still be able to demonstrate the basic features with data we will supply.*

Here are a few ideas for extra use features:

7. **Optional (this can be one of your extra features:) Add comments:** Users can add comments about content that is visible to them. Decide what data about the comments should be stored and displayed and how you want to restrict visibility of comments.
8. **Optional (this can be one of your extra features:) Like Photo** (restricted to liking Photos that are visible to the user)
9. **Optional (this can be one of your extra features:) Unfollow:** Think about what should be done when someone is unfollowed, including some reasonable approach to tags that they posted by virtue of being a follower. Write a short summary of how you're handling this situation. Implement it and test it.
10. **Optional (this can be one of your extra features:) Search by tag:**Search for photos that are visible to the user and that tag some particular person (the user or someone else)
11. **Optional (this can be one of your extra features:) Search by poster:**Search for photos that are visible to the user and that were posted by some particular person (the user or someone else)
12. Be creative. Propose something in part 3, check that it's approved, then implement it.