NYU Tandon School of Engineering
Computer Science and Engineering
CS6083, Spring 2024

April 9, 2024

Project #1 (due April 22)

In this project, you have to design a relational backend for an online service that allows people to communicate with others in their neighborhood. That is, people should be able to sign up for the service and specify where they live; they can then send and receive messages to other users living close by, and participate in discussions with those users. There are existing services such as `nextdoor` that you could look at to understand the general concept. Note that in this first project, you only have to design the relational schema for the site; the web interface will be designed in the second project. You may work alone or in groups of two, but you will need to register your group members with the GAs some days before the deadline (no last-minute switches) – wait for announcement from the TAs on this issue.

More precisely, at a minimum your system will have to support the following functionality. Users should be able to register for the service and specify where they live; this could be either done by providing an address, or by allowing them to click on their location on a map, but you can decide how to exactly do this in the second part of the project. Users should also be able to post a short profile where they introduce themselves and maybe their family, including optionally a photo. In the website, there are two levels of locality, *hoods* (neighborhoods, such as Bay Ridge or Park Slope) and *blocks* (a part of a neighborhood but not necessarily one block, e.g., "7th Avenue between 3rd and 6th Street in Park Slope"). Each block belongs to exactly one neighborhood. Users can apply to *join* a block; they are accepted as a new member if at least three existing members (or all members if there are less than three) approve. A user can only be member of one block, and is also automatically a member of the neighborhood in which the block is located. For simplicity, we assume that the names and definitions of blocks and neighborhoods are predefined by the company, so that users cannot create new ones, and that the location of a block is approximated by a center point (latitude and longitude) and a radius (though the description of a block may give a more precise location, such as "7th Avenue between 3rd and 6th Street"). Users can also choose to *follow* other blocks, which means they can read messages sent to that block or to the hood in which that block is, but they cannot post and they are not members.

Members can specify two types of relationships with other members. They can *friend* other members, including members not in the same hood or block, and they can specify (direct) *neighbors*, i.e., members of the same block that live next door, in the same building, or very close by. Friendship is symmetric and requires both sides to accept, while neighbors can be chosen unilaterally. Also, people should be able to post, read, and reply to messages.

To start a new topic, a user chooses a subject, and also chooses who can read the message and reply to it. A user can direct a message to a particular person who is a friend or a neighbor, or all of their friends or all of their neighbors, or to the entire block or the entire hood they are a member of. When others reply to a message, their reply can be read and replied to by anyone who received the earlier message. Thus, messages are organized into threads, where each thread is started by an initial message and is visible by the group of people specified in the initial message. A message consists of a title and a set of recipients (specified in the inital message), an author, a timestamp, a text body, and optionally the coordinates of a location the message refers to; thus, a message about a stoop sale or a traffic accident can potentially be placed on a map in the second part of the project. For simplicity, you do not have to worry about how to include image files and hyperlinks in a message.

It is important to think a little about how users will interact with threads and messages. Most likely, when they visit the website, they will first be directed to a main page listing all recent threads they can read, maybe separated into neighbor feeds, friend feeds, block feeds, and hood feeds. Your system should also store information about past accesses to the site by each user, so that the system can optionally show only threads with new messages posted since the last time the user visited the site, or profiles of new members, or threads with messages that are still unread. ~~Also, a real system would probably have settings where a user can choose to be notified by email of any, or just certain types of, new messages, but you do not have to implement this in the project.~~

Users who move may leave one block and/or hood and apply for membership in another block. For this case, you need to decide what content the user can access. The simplest approach might be to treat messages like email, where the user would still have access to her old messages from the previous block and hood, and only see future messages from the new block. Or should the user lose access to old threads (except maybe those she posted to), and gain access to old messages in the new block? The same problem arises for new friends and neighbors. Make a reasonable choice and justify it.

In this first part of the project, you are asked to design a database backend for such a system, that is, a suitable relational schema with appropriate keys and other constraints, plus a set of useful queries that could be created as stored procedures. You should use PostgreSQL to define the schema and test the queries. You should of course not use database permissions

to implement individual content access restrictions - there will not be a separate database account for each user, but the web interface and application itself will log into the database. Thus, the system you implement can see all the content, but has to make sure at the application level that each user only sees what she is allowed to see.

Note that in the second part, to be handed out in about two weeks and due at the end of the semester, you will have to extend this part to provide a suitable web-based interface. Thus, you cannot skip this project. Following are more details about the problem domain you are dealing with. Obviously, we will have to make some simplifying assumptions to the scenario to keep the project reasonable.

Project Steps and Deliverables: In the following, we describe the suggested steps you should take for this project, and the associated deliverables. You should approach this project like one of the design problems in the homeworks, except that the schema may end up being a bit more complicated. You should spend some time carefully designing sample data that allows you to test some of the functionality. Note again that in this first problem, you will only deal with the database side of this project - a suitable web interface will be designed in the second part. However, you should already envision, plan, and maybe describe the interface that you plan to implement. The suggested steps are as follows:

(a) Design, justify, and create an appropriate relational database schema for the above scenario. Make sure your schema is space efficient, and suitably normalized. Show an ER diagram of your design, and a translation into relational format. Identify keys and foreign key constraints. Provide a short discussion of any assumptions that you made in your design, and how they impact the model. Note that you may have to revisit your design if it turns out later that the design is not suitable.

(b) Use a database system to create the database schema, together with key, foreign key, and other constraints.

(c) Write SQL queries (or sequences of SQL queries or scripting language statements) for the following tasks.

(1) **Joining:** Write a few (3-4) queries that allow a user to sign up, to apply to become members of a block, and to create or edit their profiles.

(2) **Content Posting:** Write queries that implement what happens when a user starts a new thread by posting an initial message, and replies to a message.

(3) **Friendship:** Write queries that allow users to add or accept someone as their friend or neighbor, and to list all their current friends and neighbors.

(4) **Browse and Search Messages:** Write a few (say 3-4) different queries that might be useful when a user accesses content. For example, list all threads in a user's block feed that have new messages since the last time the user accessed the system, or all threads in her friend feed that have unread messages, or all messages containing the words "bicycle accident" across all feeds that the user can access.

(5) **Geographic Search:** Write one interesting query that uses coordinates to find messages about a particular location, say any accessible messages posted in blocks within a mile of a particular location, or messages referencing a location within 200 feet of a user's home.

(d) Populate your database with some sample data, and test the queries you have written in part (c). Make sure to input interesting and meaningful data and to test a number of cases. Limit yourself to a few users and a few messages and threads each, but make sure there is enough data to generate interesting test cases. It is suggested that you design your test data very carefully. Draw and submit a little chart of your tables that fits on one or two pages and that illustrates your test data! Print out and submit your testing.

(e) Document and log your design and testing appropriately. Submit a well-written description and justification of your entire design, including ER diagrams, tables, constraints, queries, procedures (if any), and tests on sample data. Your documentation should be a comprehensive paper, including introduction, explanations, ER and other diagrams, and more, of about 10-15 pages. This paper will be expanded in the second part of the project.