NYU School of Engineering
Computer Science and Engineering
CS 6083, Fall 2014

November 4, 2014

Project #1 (due November 21)

A startup company is working on a new online community for fans of live music. The goal is to build an online service that offers information about bands and concerts. Fans who are interested in going to concerts can then sign up, become fans of bands, and also follow other users who might post information about new bands and upcoming concerts. The goal is for users to be able to find out about upcoming concerts by their favorite bands, and also about concerts by other bands that they may not know yet, but that they might also be interested in based on their music taste.

Many of the features that this site has to offer might be familiar to you from sites like Facebook and Twitter, or music streaming services such as Spotify. (Also, some of you have encountered a very simplified version of this database in your midterm exam.) At a minimum, your design should be able to support the following four aspects: First, the system should store information about bands (artists), concerts by these bands, and the places (venues, such as bars, concert halls, cafes, etc.) where the concerts take place. Second, users should be able to sign up and specify their musical tastes and which bands they like, and the system will then be able to inform them of relevant upcoming concerts. The users should then be able to specify that they plan to go to the concert, and to give it a rating afterwards. Third, users should also be able to post additional content to help other users find good concerts; in particular, users should be able to make lists of recommended upcoming concerts, and users may also post information about other concerts that are missing from the system. Fourth, the system should help users to find upcoming concerts that might be of interest to them, even if the user is not (yet) a fan of the band or type of music, by incorporating some (naive) recommender system technology and search mechanisms into the system.

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, constraints (and maybe views, or maybe not), plus a set of useful queries that should be created as stored procedures. You may choose your own database installation on your laptop (based on Windows, Linux or Mac), but not MS Access. Make sure to use a database system that supports text operators, such as "like" and "contains", since you should allow users to search the site and textual content by keywords.

Note that in the second part, to be handed out in about two weeks and due in early December, you will have to extend this part to provide a suitable web-based interface. Thus, you cannot skip this part of the 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.

Problem Domain: We assume that in order to post or view content, a user has to sign up with the site, select a unique user name, and provide some information such as name, year of birth, email address, and city of residence. Once she has signed up, a user can post a profile, can *like* certain types of music (such as Jazz, or Hip Hop, and Americana), and can become a *fan* of certain artists (bands).

Similarly, artists can also sign up, and post profiles and hyperlinks to their website, and list the type(s) of music they are playing. Artists can also post information about their upcoming concerts, including time and date, venue (location), and ticket prices and availabilities. You should assume that the company may have verified the artist's identity (so users cannot just create an account for a famous artist), and in some cases the company may even post concert information for famous artists with their permission, while less well known bands may have to post their own information. Assume that the company has already created tables with information for all the venues, so a new concert only has to specify the name of the venue, and maybe provide a hyperlink to the place selling the tickets. Also, assume there is a hierarchical system of categories for different types of music that needs to be modeled in the schema; thus, there may be categories such as Jazz, Blues, Hip Hop, etc., and subcategories such as Free Jazz, Bebob, or Cool Jazz within a category such as Jazz.

Users can also contribute information to the system in three different ways. First, users can state that they plan to attend or have attended a concert. They can also post ratings and write reviews for conferences, where a review consists of some text. (There is of course no way we can check if the user actually went to the concert.) Second, users who are experts for certain types of music can also post lists of recommended upcoming concerts. For example, a user named Jake who likes Jazz might create and maintain a list of recommended Jazz concerts called "Jake's Jazz Picks". Other users interested in his recommendations can decide to *follow* Jake and then will get notified about new concerts recommended by Jake.

Third, your design should have a *reputation system* that allows users to post information about concerts missing from the system. The problem here is that we do not want random users to post incorrect concert information. Thus, imagine that each user has a trust score that reflects how long they have been using the system and how many contributions they have

made in the past (in terms of concert reviews and ratings, concert recommendations they created, or past data about concerts they posted). For example, you could allow concert data to be added to the system by anyone with a trust score of at least, say, 8 out of 10, or by anyone as long as at least two users of trust score 7 or higher support the submission. The details are up to you.

Note that in the final system, users may want to browse or search the content of the system. They may decide to visit another user's profile page, or the page of a band, or a list of recommended concerts, or they may search based on keywords, categories, locations, or time (e.g., to find Jazz concerts in Chicago in December 2014). Each of these pages should probably be accessible via some URL derived from the user name or location name, e.g., `http://livejive/jake/` for a user with user name Jake if the site has the name LiveJive. From such a page, it should be possible to follow links, for example to the bands that a user is a fan of. Additionally, when a user logs in, she might see a *message feed* consisting of upcoming concerts recommended by users she follows, and maybe also concerts recommended by the system based on the users preferences and past ratings etc. (Note that these items only have to be implemented in the second project, but you should design a database in this first project that can support these types of operations.)

Two more remarks. First, it is recommended to add various time stamps indicating when content (reviews, ratings, recommendations, concert data) was posted by a user, when she started following another user or being a fan of a band, or even when she last accessed the system. This will allow the application display new content of interest has become available since the last time the user logged in. Second, you should not use database permissions or views to implement 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. So, the system you implement can see all the content, but has to make sure at the application level that each user only sees and does what she is allowed to see/do.

Project Steps: Note that the following list of suggested steps is intended to help you attack the problem. You do not need to follow them in this order, as long as you come up with a good overall design that achieves the requested functionality. Note that in this firstthey part of the project, you will only deal with the database side of this project - a suitable web interface will be designed in the second part of the project. However, you should already envision, plan, and describe the interface that you plan to implement.

(a) Design, justify, and create an appropriate relational schema for the above situation. Make sure your schema is space efficient. Show an ER diagram of your design, and a translation into relational format. Identify keys and foreign key constraints. 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 stored procedures) for the following tasks:

(1) **User Data:** Write queries that users need to sign up, to create or edit their profiles, to follow another user, to become fan of a band, and to post a review and rating.

(2) **Band and Concert Data:** Write queries that bands can use to post new concerts, and queries that users can use to post user data (with a check on the user's trust level), to create a list of recommended concerts, and to add a new concert to an existing list.

(3) **Browse/Search Queries:** Write three queries that a user could use when accessing content in your system. For example, a user might want to see all Jazz concerts in a certain city during the next week, or see all concerts recommended by people they follow during the next month, or see all newly posted concerts since the last time they logged in.

(4) **System Recommendations:** Write two or three queries that the system could use to recommend to a particular user bands and concerts that the user might be interested in, given past behavior by the user. For example, the system could recommend to the user those concerts in the categories the user likes that were recommended in many lists by other users. Or the system could suggest bands that were liked or who concerts were highly rated by other users that had similar tastes to this user in the past (in terms of being fans and rating concerts). Note there is an entire area called Recommender Systems in Computer Science that studies such problems – you may want to look up some very basic techniques in this area to get some ideas on what to do here.

(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 fan and follow relationships, 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 "map" that illustrates your test data! Print out and submit your testing.

(e) Consider defining appropriate stored procedures for various common tasks. Each stored procedure should have a few specified input parameters (such as the user name, or a set of keywords on which a search is performed, or the data for a new entry that should be created, etc.), and should return a defined result. Note that in the second project, you have to call these

procedures from a web-based interface outside your database, so find out how to define stored procedures, how they can be called, what restrictions there are, and what sort of technologies (like Java+JDBC, PHP, CGI) there are for interfacing from a web server.

(f) Document and log your design and testing appropriately. Submit a properly documented description and justification of your entire design, including ER diagrams, tables, constraints, queries, procedures, and tests on sample data, and a few pages of description. Submit in hardcopy only!