ECE 656 Winter 2020: Project

The course project is a database-design and implementation exercise. You are required to create an appropriate database design for the problem space and implement a prototype of the design. There are two possible problem spaces you may select from:

1. a un*x file system
2. a social network
3. with permission of the instructor by February 14th, an alternate space from Kaggle or other suitable sizable data-space (*e.g.* Yelp! (`https://www.yelp.com/dataset`))

Additional details for the specific problem space will be presented below.

There are five steps required in the design and prototype process:

1. Create an Entity-Relationship Model of the problem space
2. Translate the ER model to a relational model, writing the necessary SQL to create the tables, primary keys, foreign keys, and any additional indexes and/or other constraints
3. Populate your database with real-world sample data, as appropriate to your problem space
4. Create the prototype client (in the programming language of your choice), as required by the specific problem space
5. Data mine some aspect of your problem space, presenting some interesting finding

You will submit a deliverable for each of the five items. In the case of the first, a PDF with the ER model diagram plus description as necessary (should not be more than a couple of pages). In the case of the second and third, the `.sql` code and source of the sample data. In the case of the fourth, whatever code is necessary, together with a README so that the evaluator knows what is necessary for the purpose of executing your code. Since a demo would be desirable, but we have only one TA and so scheduling in-person demos is infeasible, make a short (˜5 minutes) video demo. Finally, for the fifth item, write a short (˜2 pages, not including illustrations and/or graphs) report describing what mining technique you used, how you applied it to your specific problem space, and what you discovered.

Option (i): The Un*x File System

The standard un*x file system is very inefficient for a number of reasons. However, the basic directory structure is well understood by users and developers alike, as are the various file-system utilities (cd, ls, find, grep, *etc.*). Using a relational database as a file system should be much more efficient, though only if we do so in a non-standard way. The standard method is to implement the various file-system system calls (open, close, read, write, lseek, *etc.*) for the new file system and then nothing else needs to change for the user (existing utilities and shell remain unaltered). The problem with this is that doing so would essentially negate any advantage that the use of a relational database would bring. Instead of doing this, it is necessary to rewrite the standard utilities so as to provide the standard "look-and-feel" of that utility, but doing so on top of a relational database.

For this option, you are required to do the following:

1. Create an entity-relationship model for this un*x file-system. Your diagram may follow any of the notational conventions described in the lecture notes, in class, or the course textbook, but should be consistent in its usage. Include all relevant entities, attributes and relationships, cardinality constraints, and participation constraints. Indicate primary keys of entities by underlining them. Files have all of the standard un*x file-system properties: name, permission bits, size, type, *etc.*, as well as the file contents (you may choose to add additional properties if you wish (*e.g.,* a user-level type (C source, C++ source, text, executable, *etc.*), applicable programs for the given user-level type, *etc.*) though it is not required). In addition, directories have to be maintained in the standard un*x manner with sub-directories as created, and types per the standard un*x permissions. Also, links, both hard and symbolic, should be supported by your design.
2. Create the necessary relational database form of your ER model. It should be appropriately normalized. For file contents, you should use the type BLOB (Binary Large OBject) or TEXT types, as appropriate. SQL to create all necessary tables, keys, *etc.* is the required deliverable for this part.
3. Populate your database with appropriate sample data; a good starting point would be to write a small utility that would copy the file system from a un*x box into your system.
4. Create the following client utilities to run on top of your database:
   (a) rdbsh: a shell program that can, at a minimum, keep track of the current working directory, change the working directory (cd), maintain a PATH variable, and execute any executable program that is in the PATH.
   (b) ls: you should be able to accept the "-l" argument or just plain usage; you may choose to implement "ls" as a shell command in your shell.
   (c) find: accept the directory and (partial) name of the file being found; output the "ls -l" results for all matches; again, you may choose to implement "find" as a shell command.
   (d) grep: accept the (partial) name of the file and seek the relevant pattern in the matching file(s). Output the line number and line for the matching lines.
5. Data mine as appropriate ...

Option (ii): A Simple Social Network

Broadly speaking, all social networks are centered around "people" posting "things" about "topics" and commenting on those posts. People have various attributes that may be genetic (*e.g.,* birthdate, sex, *etc.*) or chosen (*e.g.,* vocation, religion, *etc.*). People are part of one or more groups (reciprocal) or follow one or more other people (non-reciprocal).

Topics are ideally self-organized by the people in the social network, but you can think of things like: politics, sports, business, finance, news, ... and then sub-topics of these (Canadian politics, oil business, *etc.*), which in turn can be composed of sub-sub-topics (Toronto politics, Alberta oil business, ...). The hierarchy may be explicit or implicit in the nature of the topic names.

Things posted are either some initial post, comprising text, image(s), and link(s) by a person on a topic/topics. Those who follow that topic or posting person will be notified of the posting, and may choose to respond to the post, either simply with a thumbs up/down or by posting their own text, image(s) and link(s) in response, which in turn may generate responses. Likewise, a person or topic may be searched, and the person receiving the results may choose to respond to those results. If a person has read a post, the system should know this, although "read" may simply mean "the client has presented it to the user."

For this option, you are required to do the following:

1. Create an entity-relationship model for this social network. Your diagram may follow any of the notational conventions described in the lecture notes, in class, or the course textbook, but should be consistent in its usage. Include all relevant entities, attributes and relationships, cardinality constraints, and participation constraints. Indicate primary keys of entities by underlining them. People should have appropriate attributes (name, birthdate, others of your choosing), and connections with other people, again of your own choosing (*e.g.,* you might want a social network that limited the number of "friends" you had, so that it was plausible that it actually represented friends). Postings will have to be defined in terms of text, image(s), link(s), as you see fit, and the connection between the post and (a) other posts (is it a response?) (b) topics (is there a limit on the number of topics?). A simple thumbs up/thumbs down needs to be supported for posts, in addition to responses. You may choose to have more such simple responses (*e.g.,* Yelp! has "useful" and "funny"). Topics are, in some ways, the most flexible, but you need to create a design for that and explain your choices.
2. Create the necessary relational database form of your ER model. It should be appropriately normalized. SQL to create all necessary tables, keys, *etc.* is the required deliverable for this part.
3. Populate your database with a non-trivial sample of data (possibly from Kaggle; Yelp!, ...).
4. Create the social-network client to run on top of your database:
   (a) A person should be able to initial a post on a topic
   (b) A person should be able to follow/join a group with another person
   (c) A person should be able to follow a topic
   (d) A person should be able to determine what posts have been added by people and/or topics that they are following since they last read from those people/topics
   (e) A person should be able to respond to a post with thumbs up/down and/or a response post.
   (f) Additional as appropriate ...
5. Data mine as appropriate ....

You are *NOT* required to have a GUI client; a simple command-line interface is sufficient; what is preferred above either a CLI or a GUI client, is a set of well-defined APIs for your client, and then a simple client that used said APIs.

Option (iii): An Alternate Problem Space

Make a proposal before February 14th that clearly delineates the problem space and the five required aspects of the project.