CS121 Final Exam Fall 2016

Due Wednesday, December 7 at 5:00PM

CS121 Final Exam Instructions

This is the cover sheet for the CS121 final exam. **The time limit on the final exam is six hours, in multiple sittings.** If you run over time then indicate where time ran out; you will receive 50% credit for the first hour over the time limit, and no credit thereafter.

The final exam has several problems focused on database schema design problems; these write-ups are a bit long because the problem specifications need to be complete. **When you start taking the exam, feel free to read through the entire exam once, before starting your timer.**

Feel free to consult any course material, including the MySQL online manual, the Database System Concepts textbook, any lecture slides, your graded homework assignments, or the solution sets for the assignments. Do not talk to anyone else about the final exam contents until both of you have actually completed it and turned it in. Do not use the Internet to find answers to the problems.

You should use your MySQL database for this final. Your database schemas and other SQL commands should run successfully against the database. Answers that fail to do so will most likely be penalized.

You may not use programs to solve functional dependency or multivalued dependency problems, or to check answers. You must solve these problems manually, and show your work.

If you run into any ambiguities in the final exam, please contact Donnie for clarification. I try to make things as clear as possible, but it doesn't always happen.

You can turn in a hard copy of your E-R diagrams at Donnie's office, but make sure to upload all other files to Moodle. Here are some additional rules to follow, to avoid point-deductions:

- **If you hand-write portions of your final exam, your writing must be legible!** Sloppy work will lose points.
- If you use software to create your E-R diagram, and it doesn't easily support double-lines for total participation constraints, you may indicate this with a very thick line instead. Make a note on your submission if you end up do this.
- If you submit your E-R diagram electronically, it must be as a PDF file formatted for 8½"x11" paper.
- **This year you will put your functional and multivalued dependency answers into plain-text files following a specific format.** These files can then be uploaded to Moodle with the rest of your submission. Details are given in the exam itself.

SQL Guidelines

The same general SQL guidelines apply as for the midterm, although all SQL must be valid MySQL syntax. As usual, keep lines to 80 chars or less.

Make sure your SQL DDL is well formatted and well commented. At the very least, you should have a comment on each table describing its purpose, and any columns that only allow specific values should have a comment explaining what values are valid. Poorly commented or uncommented DDL will be penalized.

WHEN YOU ARE READY TO START THE FINAL EXAM, GO ON TO THE NEXT PAGE!

CS121 Final Exam Fall 2016

Due Wednesday, December 7 at 5:00PM

Overview

The final exam includes three problems. Each problem starts on a new page, so make sure you read all the way through the entire exam. You wouldn't want to miss a part of the exam…

Problem 1: Photo and Video Website *(50 points)*

The database schema you will be designing for this problem is intended to be used in a photo and video sharing website. Users can create accounts on the website, and then upload photos or video files to the website to share with each other. Photos and videos can also be organized into albums. Finally, users can leave comments on photos and videos.

Initial Notes

None of the specifications below *require* you to create numeric IDs for any of these items, although you may find it useful to do so. Feel free to add numeric ID fields wherever you see fit.

In addition, you will likely want to add other attributes to some of these entities in order to implement all required functionality. Again, feel free to do so; just make sure to explain why you are adding such attributes.

Finally, you are only required to enforce the constraints specified in the problem. You might notice other constraints you'd like to enforce if this were going to be used for a real website, but don't worry about that; we want to keep the problem reasonably constrained.

User Accounts

The system needs to keep track of the following information for user accounts:

- A username that is no more than 20 characters. Every user must have a unique username.
- Every user must also specify an e-mail address. E-mail addresses don't need to be unique.
- Every user also has a password that is used to authenticate the user at login. Make sure your schema design properly supports user authentication, without opening up any security risks.

Photos and Videos

Both photos and videos must be supported within the database. The main difference between the two is that each photo's data will be stored within the database (as blobs), but the data for each video will be stored on the file-system, and only a path to the video file is stored within the database. Both photos and video files must have these details:

- Every photo or video must have a one-line title. This title doesn't have to be unique, but some value must be specified. Make sure to support reasonably verbose titles, e.g. up to 200 chars.
- Every photo or video may also optionally have a text description. This description is expected to be no larger than a single paragraph.
- Each photo or video has an "upload time" specifying the date and time when it was added to the website. This value is required.
- Every photo or video will also have a total number of views, which is the number of times it has been downloaded from the website. This value should default to 0 when the item is first uploaded to the website, and should be present on all items.
- Of course, every photo or video must be associated with the user who uploaded the item. This association is required.
- Each photo or video may be put into at most one album. (Albums are described later.) Items are not *required* to appear within an album, but if they are, they may only be stored within one album.

In addition to the above properties, photos must also specify these additional values:

- The filename of the photo. This value excludes any path information.
- The data for the photo itself. You can use a **`BLOB`** column to store the photo data.

Videos, on the other hand, have these properties:

- The total length of the video in seconds. This value is a whole number; fractional seconds will be rounded up to the next second when the video is stored. The value will be formatted into a human-readable format in the presentation layer. This value must be specified for each video.
- A path and filename for where the video file is stored on the server. This path could be up to 4K characters, since some file-systems support paths of this length. Again, this value is required.

Albums

Each user can create any number of albums, to organize their photos. There is no requirement to store media items in albums, but they are available if a user wants to use them. Each album has these characteristics:

- An album name must be specified. A given user's albums must all have unique names, but albums owned by different users may have the same name.
- An optional album description, which may be up to a paragraph in length.

Each album must be associated with the user who created it. Also, as mentioned before, each photo or video may be stored in at most one album; an item is not allowed to appear in multiple albums.

Each album may also optionally specify a "summary photo" for the album. This is a photo that will be displayed for the album when a user's albums are listed on their webpage. Videos may not be used for this, only photos.

NOTE: Don't worry about enforcing that an album can only contain items uploaded by the album's owner. Also, don't worry about enforcing that an album's summary photo was uploaded by the album's owner. These constraints definitely make the database more complex.

Comments

Users are allowed to leave comments on the photos or videos on the website. The database should store:

- The user who left the comment.
- The date and time the comment was created.
- The item (photo or video) that the comment is for.
- The text of the comment itself, which should not be more than a few sentences.

A user should be able to leave multiple comments on the same item; all comments for an item will be listed in time order. Each comment should be associated with exactly one photo or video.

Your Tasks

1. Create an E-R diagram for a database schema that satisfies the above specification. Make sure to indicate mapping cardinalities, participation constraints, and any other E-R modeling constraints required by your design. Missing constraints will be penalized. Also, make sure your diagram is neat and readable; sloppiness will also be penalized. *(16 points)*

CS121 Final Exam Fall 2016

Due Wednesday, December 7 at 5:00PM

2. In a file **`website/webschema.txt`**, map the E-R diagram to a relational model schema. Make sure to indicate all primary keys, other candidate keys, and foreign keys for each one of your schemas. If you perform any schema combinations, etc., make sure to explain briefly why it is desirable to do so. *(10 points)*

3. In a file **`website/make-website.sql`**, create the sequence of SQL DDL commands necessary for dropping (if tables exist) and then creating the database schema. Make sure to specify reasonable types, sizes, and nullity constraints on each column. Also, specify all primary keys, foreign keys, and other candidate keys in your SQL DDL. Make sure to comment your DDL as well. Your DDL must load into MySQL. *(12 points)*

4. Finally, in a file **`website/queries.sql`**, write SQL queries against your schema that produce the following results:

   a) Report the title, time uploaded, and the number of views for the top 20 photos and videos uploaded by the user "bopeep", in descending order of number of views.

   b) For every album created by the user "garth", report the album's name, its description, the number of photos and videos in the group, and the filename of the group's summary photo, if any. (If the album has no summary photo, include NULL for the filename.)

   c) Write a query that reports the top comment-posters across the entire website. The result should contain each user's username, and the total number of comments that user has posted. List only the top 10 users, in descending order of comments posted.

   (4 points per part; 12 points total)

(Exam continues next page.)

CS121 Final Exam Fall 2016

Due Wednesday, December 7 at 5:00PM

Problem 2: Breed Registry *(20 points)*

Maybe over the Thanksgiving weekend you saw the National Dog Show on television. Animal breeding is a big business, and animals with specific characteristics are highly desirable to use for breeding offspring. Breeders keep very careful records, so that a given animal's pedigree can easily be determined. A number of calculations can also be performed on this data to determine whether it would be a good or bad idea to breed particular animals together.

In the file **`pedigrees/registry.sql`** you will find a simple table used for keeping a breed registry in a database. A particular registry will only record pedigree data for one kind of animal; e.g. horses, dogs, rabbits or turkeys. However, the terms in this SQL file are generalized so that it could be used for any given kind of animal.

Note that the relationships between animals in a breed registry form a graph, not a tree. For example, a particular animal could be both a parent and a great-grandparent of another animal. This practice is called *line breeding*, and it is essential for retaining and enhancing specific breed-line traits. Of course, there are many nuances to how a breed-line is developed; these are irrelevant to the problem, but the consequence for us is that our data does not form a simple tree.

Part A: Generating Pedigrees *(15 points)*

An animal's *pedigree* is simply a record of all known ancestors of the animal, including some specific number of generations. In the file **`pedigrees/pedigree.sql`**, write a stored procedure to generate an animal's pedigree, which operates as follows.

The stored procedure's signature is:

```
sp_make_pedigree(animal_id INTEGER, generations INTEGER)
```

The procedure may assume that all inputs and breed-registry data are valid.

- The registry will not contain paradoxes such as an animal being its own parent.
- The specified *animal_id* will appear in the registry.
- The specified *generations* value will be ≥ 0.

A **`generations`** value of 1 should generate a pedigree that only includes the animal and its parents (a maximum number of 3 rows); a value of 2 would also include the animal's grandparents (a maximum number of 7 rows); and so forth. Note that the registry may not actually contain all of the animal's ancestors back through the specified number of generations; the pedigree should contain as much information as is known about the animal.

The result should be written into the **`pedigree`** table (also declared in **`registry.sql`**). This table should be cleared at the start of your stored procedure, and then the procedure can write its results into the table. The pedigree data should be generated as follows:

- The pedigree data should include one row for the animal being pedigreed, plus a row for each known ancestor of the animal that falls within the specified number of generations. (As stated earlier, an ancestor may appear multiple times in an animal's pedigree; in real breed registries this is rather common.)
- Each row has the schema (*animal_id*, *relationship*). The *relationship* attribute is a candidate key on the schema. The *animal_id* attribute is <u>not</u> a candidate key on the schema!
- The *animal_id* is either the animal itself, or one of the animal's ancestors.
- The *relationship* value specifies the relationship of the animal in the row, to the animal being pedigreed. It follows the path enumeration model for representing hierarchies:

- The original animal being pedigreed has the *relationship* value "A" (for "animal").
- That animal's sire (father) will have the value "A.S" (animal's sire); the animal's dam (mother) will have the value "A.D".
- The animal's sire's sire (paternal grandfather) will have the value "A.S.S"; the animal's dam's sire (maternal grandfather) will have the value "A.D.S".
- And so forth.

A few final requirements and suggestions:

- You may find the MySQL **`CONCAT()`** string-concatenation function useful in your implementation. (There are several ways to do this part, so don't feel obliged to use it.)
- Make sure to write a comment before your stored procedure, describing what it does.
- Make sure to write comments in the body of your procedure explaining how it works.
- You may not change the schema of the **`pedigree`** table.

Testing

A data file **`pedigrees/dogs.sql`** is provided, which contains a fabricated breed registry for basset hounds. This data is not particularly accurate (e.g. it doesn't take gestation times or litters into account, and it doesn't follow the rules for how dogs pass along coat coloration), but it attempts to mimic the structure that a real breed registry would have. You can generate and view pedigrees as follows:

```
-- Generate pedigree for Apollo, back to great grand-parents.
call sp_make_pedigree(72, 3);

-- Now view the pedigree.
select animal_id, pet_name, relationship
from pedigree natural join breed_registry;

+-----------+----------+--------------+
| animal_id | pet_name | relationship |
+-----------+----------+--------------+
|        72 | Apollo   | A            |
|        57 | Mickey   | A.S          |
|        31 | Abby     | A.S.D        |
|        18 | Willow   | A.S.D.D      |
+-----------+----------+--------------+
```

You should verify that your procedure works properly with various values for the *generations* argument; getting this to work correctly can be a bit tricky.

Here are some dogs with pedigrees that you can look at:

- Hunter (ID = 6) doesn't have any known ancestors
- Zoe (ID = 22) only has parents in the registry, but no further ancestors
- Hannah (ID = 107) has all parents and grandparents in the registry, but no further
- Rocco (ID = 166) has a deeper pedigree on the mother's side than on the father's side
- Dixie (ID = 175) has a very deep pedigree, going back 8 generations

Part B: Test Query *(5 points)*

Finally, once you have completed the stored procedure, write a query against the **`pedigree`** and **`breed_registry`** tables that reports each animal that appears in the current pedigree more than once. The schema of the result should be (*animal_id*, *registered_name*, *num_times*). Order your results by *num_times* in descending order.

You should put your query into the **`pedigrees/pedigree.sql`** file, after the stored procedure definition.

In a comment, include the output of your query for the pedigree of Dixie (ID = 175). (Note that this is an example of how the breed registry contains completely fabricated data. An animal with a pedigree like this would likely have many severe congenital issues.)

(Exam continues next page.)

CS121 Final Exam Fall 2016

Due Wednesday, December 7 at 5:00PM

Problem 3: Functional and Multivalued Dependencies *(30 points)*

(Put your answers to this problem in the file **`problem3.txt`** *in your submission. This file also describes the formatting you should follow for your answers.)*

1. Given the relation-schema $R(A, B, C, G, H, I)$, and the set of functional dependencies $F = \{ A \rightarrow CG, B \rightarrow A, AB \rightarrow C, B \rightarrow G, GH \rightarrow I \}$.

   a) Find a candidate key for R. Demonstrate that it is a candidate key, not just a superkey. *(4 points)*

   b) Compute a canonical cover F_c from F. Make sure to show all steps in your computation. *(8 points)*

   c) Create a 3NF decomposition of R. Make sure to indicate all candidate keys on your resulting schemas. *(6 points)*

2. Given the following database and set of functional dependencies:

   pizza_orders(*order_id*, *order_time*, *cust_id*, *cust_address*, *cust_phone*, *pizza_id*, *pizza_size*, *topping*)

   order_id → *cust_id*, *order_time*
   cust_id → *cust_address*, *cust_phone*
   pizza_id → *order_id*, *pizza_size*
   pizza_id →→ *topping* (a multivalued dependency)

   Answer the following questions:

   a) Create a set of 4NF relation schemas for the above database. Make sure to indicate all primary/candidate/foreign keys in your resulting schemas. (You do not need to write SQL DDL; just write relation schemas.) *(7 points)*

   b) Is it possible to create a version of the above database schema that is in BCNF but is <u>not</u> also in 4NF? (Note: Every 4NF schema is also in BCNF, but the inverse is not necessarily true.) If so, give an example, and explain what the difference(s) would be. If not, explain why it's not possible. *(5 points)*

(End of exam! Hooray!)