Problem Set 5

All problems are due by 11:59 p.m. on Wednesday, May 1, 2019.

Preliminaries

In your work on this assignment, make sure to abide by the collaboration policies of the course. ***All of the problems in this assignment are individual-only problems that you must complete on your own.***

If you have questions, please come to office hours, post them on Piazza, or email `cs460-staff@cs.bu.edu`.

Make sure to submit your work on Gradescope, following the procedures found at the end of Part I and Part II.

Part I

50 points total

Creating the necessary file

This part of the assignment will all be completed in a single PDF file. To create it, you should do the following:

1. Open the template that we have created for Part I in Google Docs: ps5_partI
2. Select *File->Make a copy...*, and save the copy to your Google Drive using the name `ps5_partI`.
3. Add your work for all of the problems from Part I to this file. We have given you headings that you can use for each problem. For Problems 1 and 2, please number your answers to each part of the problem.
4. Once you have completed Part I, choose *File->Download as->PDF document*, and save the PDF file on your machine. The resulting PDF file (`ps5_partI.pdf`) is the one that you will submit. See the submission guidelines at the end of Part I.

Problem 1: Data models for a NoSQL document database

20 points total; 4 points each part

Consider the following `CREATE TABLE` commands for a portion of relational database maintained by an online music service:

```
CREATE TABLE Song(id CHAR(10) PRIMARY KEY, name VARCHAR(64),
    duration INTEGER, genre VARCHAR(10), best_chart_rank INTEGER,
    royalties_due REAL);

CREATE TABLE Artist(id CHAR(7) PRIMARY KEY, name VARCHAR(128),
    label VARCHAR(30), dob DATE, primary_genre VARCHAR(10));

CREATE TABLE Sings(songID CHAR(10), artistID CHAR(7),
    PRIMARY KEY(songID, artistID),
    FOREIGN KEY songID REFERENCES Song(id),
    FOREIGN KEY artistID REFERENCES Artist(id));

CREATE TABLE Played(date VARCHAR(10), time VARCHAR(5), song CHAR(10),
    PRIMARY KEY(date, time, song),
    FOREIGN KEY song REFERENCES Song(id));
```

The first two tables store information about songs and artists.

The third table (`Sings`) captures relationsips between songs and artists.

The last table (`Played`) allows the online music service to keep track of how often a song is played by its listeners, which is relevant for determining the royalties that the service will owe for that song (as recorded in the `royalties_due` attribute of `Song`).

You have been asked to create a database that stores this same information in MongoDB.

1. There are two types of relationships that the database needs to capture: those between songs and artists, and those between songs and their associated “played” data.

   One way to capture these relationships would be to take a purely reference-based approach that avoids embedding one type of document in another. Illustrate this approach by showing how the following tuples from the relational database would be represented as one or more documents in MongoDB:

   ```
     from Song: ('0123456789', 'Brave', 219, 'pop', 23, 1200.00)
   from Artist: ('9876543', 'Sara Bareilles', 'Epic', '1979-12-07, 'pop')
    from Sings: ('0123456789', '9876543')
   from Played: ('2015-04-22', '17:00', '0123456789')
   ```

   Guidelines:

   - You will need to determine how many documents are needed, how many references are needed, and where the references should go. Make sure to take into account the factors relevant to data modeling that we discussed in lecture.
   - Your answer should take into account the entire sets of relationships that the database will need to capture, but the only concrete relationship(s) that your answer needs to include are those described by the tuples above.

2. Briefly describe one advantage and one disadvantage of the approach to data modeling taken in part 1.

3. Another way to capture the relationships would be to take an approach that allows you to embed one type of document in another. Illustrate this approach by showing how the tuples from part 1 would be captured as one or more documents in MongoDB. The guidelines from part 1 also apply here.

4. Briefly describe one advantage and one disadvantage of the approach to data modeling taken in part 3.

5. Consider the updates that need to happen when a customer of the music service plays a song. Discuss the appropriateness of the data-modeling approaches from parts 1 and 3 in light of this particular use of the database.

Problem 2: Logging and recovery

20 points total

We will complete the material needed for this question during the week of April 22.

Consider the following sequence of log records written by a system that uses **undo-redo** logging:

LSN	record contents
0	txn: 1; BEGIN
10	txn: 1; item: A; old: 100; new: 130; olsn: 0
20	txn: 2; BEGIN
30	txn: 2; item: C; old: 400; new: 430; olsn: 0
40	txn: 1; item: A; old: 130; new: 150; olsn: 10
50	txn: 1; item: B; old: 500; new: 510; olsn: 0
60	txn: 1; COMMIT
70	txn: 2; item: D; old: 200; new: 210; olsn: 0
80	txn: 2; item: B; old: 510; new: 570; olsn: 50

1. (3 points) If a crash occurs and log record 80 is the last one to make it to disk, what are *all* possible on-disk values of each of the data items (A, B, C, and D) after the crash but before recovery?
2. (3 points) How would your answer to part 1 change if the system were using **redo-only** logging instead of undo-redo? Explain the reason for any changes briefly. (Note: You should assume that none of the data items (A, B, C, D) are on the same page.)
3. (3 points) How would your answer to part 1 change if the system were using **undo-only** logging? Explain the reason for any changes briefly. (Here again, you should assume that none of the data items are on the same page. In addition, you should assume that when the DBMS forces dirty database pages to disk, it forces only those pages that must go to disk in order for undo-only logging to work correctly.)
4. (4 points) If a crash occurs and log record 80 is the last one to make it to disk, what steps would be performed during recovery if the system is performing **undo-redo** logging and the on-disk datum LSNs are *not* consulted? (In other words, you should assume that the system is *not* performing logical logging, and thus you don't need to worry about redoing or undoing a change unnecessarily.) Complete the table provided in `ps5_partI` to show how each log record would be handled during both the backward and forward passes.
5. (4 points) If a crash occurs and log record 80 is the last one to make it to disk, what steps would be performed during recovery if the system is performing **undo-redo** logging and the on-disk datum LSNs *are* consulted (i.e., the system *is* performing logical logging)?

Complete the table provided in `ps5_partI` to show how each log record would be handled during both the backward and forward passes. You should assume that the datum LSNs at the start of recovery are the following:

- A: 0
- B: 50
- C: 0
- D: 70

In addition, you should assume that the recovery subsystem does not perform any actions that the LSNs indicate are unnecessary.

6. (3 points) If a dynamic checkpoint had occurred between log records 30 and 40, how would that change your answer to part 4? Explain any changes briefly.

Problem 3: Two-phase commit

10 points

We will cover the material needed for this question during the week of April 22.

When a site is ready to commit its subtransaction under two-phase commit, it must take whatever steps are needed to put itself into a *ready* or *precommitted* state before sending a ready message to the coordinator of the distributed transaction. What steps are needed if the site is using **undo-only** logging? Give the steps in the order in which they would need to occur, beginning with the receipt of the prepare message from the coordinator and ending with the sending of the ready message to the coordinator.

Submitting your work for Part I

1. Once you have completed Part I in Google Drive, choose *File->Download as->PDF document*, and save the resulting file (`ps5_partI.pdf`) on your machine.
2. Login to Gradescope by clicking the following link:

   Gradescope

3. Once you are in logged in, click on the box for ***CS 460***.
4. Click on the name ***PS 4, Part I*** in the list of assignments. You should see a pop-up window labeled ***Submit Assignment***. (If you don't see it, click the **Submit** or ***Resubmit*** button at the bottom of the page.)
5. Choose the ***Submit PDF*** option, and then click the *Select PDF* button and find the `ps4_partI.pdf` that you created in step 1. Then click the *Upload PDF* button.
6. You should see an outline of the problems along with thumbnails of the pages from your uploaded PDF. For each problem in the outline:

  - Click the title of the problem.
  - Click the page(s) on which your work for that problem can be found.

  As you do so, click on the magnifying glass icon for each page and doublecheck that the pages that you see contain the work that you want us to grade.

7. Once you have assigned pages to all of the problems in the question outline, click the *Submit* button in the lower-right corner of the window.

8. You should see a box saying that your submission was successful. Click the `(x)` button to close that box.

9. You can use the ***Resubmit*** button at the bottom of the page to resubmit your work as many times as needed before the final deadline.

Important

- It is your responsibility to ensure that the correct version of a file is on Gradescope before the final deadline. ***We will not accept any file after the submission window for a given assignment has closed, so please check your submission carefully using the steps outlined above.***

- If you are unable to access Gradescope and there is enough time to do so, wait an hour or two and then try again. If you are unable to submit and it is close to the deadline, email your homework ***before the deadline*** to `cs460-staff@cs.bu.edu`

Part II

50 points total

Overview

In this part of the assignment, you will write queries for a MongoDB version of our movie database – one that uses the data model outlined in lecture.

Using MongoDB

MongoDB is already installed on the virtual machine that you used in PS 2-4. If for some reason you need to redownload the VM, you should do so as soon as possible.

Downloading the necessary files

Once you have started the VM, you should take the following steps to obtain the files that you will need for this assignment.

Important

To simplify matters, you should ***store all of these files in*** `mongodb/bin` – i.e., the `bin` subfolder of the `mongodb` folder that we have placed in the `Home` directory on the VM.

1. Run Firefox on the VM, and navigate to the page for this assignment (this page!).
2. ***Right-click*** on each of the following links, choose *Save Link As*, and save the file in `mongodb/bin`. (You may need to click on the `Home` icon in the file-save window to locate the `mongodb` folder.)
   - movies.json
   - people.json
   - oscars.json
   - ps5_queries.js

   Important: Make sure to ***right-click*** on each link and choose *Save Link As*. This will preserve the correct file extension for each file.

Starting the MongoDB server

MongoDB uses a client/server architecture, so we will need to run two different programs: the server (a program called `mongod`) and a client (which in our case will be the MongoDB shell). Once you have completed the steps outlined above, you can start the `mongod` server by doing the following:

1. Open a terminal window by double-clicking on the Terminal Emulator icon on the Desktop of the virtual machine.

2. `cd` into the `mongodb/bin` directory:

   ```
   cd mongodb/bin
   ```

3. Enter the following command:

   ```
   ./mongod --dbpath .
   ```

   Important: There is a dot (`.`) at the end of the command!

   You should see a series of status messages from `mongod`, ending with one about the admin web console waiting for connections.

Important

You should leave this terminal window open while working with the database so that the server will continue to run and be accessible.

Before closing down the virtual machine, you should return to the terminal window in which the `mongod` server is running and type CTRL-C to shut down the server.

Creating the database

1. Leaving the `mongod` server running in the initial terminal window, open a second, separate terminal window.

2. `cd` into the `mongodb/bin` directory:

   ```
   cd mongodb/bin
   ```

3. To create the `movies` collection (as well as the `imdb` database in which it will reside), enter the following command from within the `mongodb/bin` directory:

   ```
   ./mongoimport --db imdb --collection movies --file movies.json --jsonArra
   ```

   You should see a message saying that 703 objects were imported.

4. To create the `people` collection, enter the following command:

   ```
   ./mongoimport --db imdb --collection people --file people.json --jsonArra
   ```

You should see a message saying that 2561 objects were imported.

5. To create the `oscars` collection, enter the following command:

   ```
   ./mongoimport --db imdb --collection oscars --file oscars.json --jsonArra
   ```

   You should see a message saying that 522 objects were imported.

Performing queries in MongoDB

Once you have completed the steps outlined above, you can perform queries by taking the following steps:

1. Make sure that the `mongod` server is running in one terminal window (see above).

2. If you don't already have a second terminal window open, open one now. (If you just created the database, you can continue to use the same terminal window in which you created it.)

3. If you're not already in the `mongodb/bin` directory, enter it now:

   ```
   cd mongodb/bin
   ```

4. Start up the MongoDB shell from within that directory, specifying that you want to use the `imdb` database. (**Note:** The name of the shell program is `mongo`, ***without*** a `d` at the end.)

   ```
   ./mongo imdb
   ```

   You should see a message telling you that MongoDB is connecting to `imdb`, followed by the shell's command prompt:

   ```
   >
   ```

5. Enter the appropriate method calls for your queries. If you simply enter the method call (and don't assign its return value to a variable), you should automatically see up to the first 20 results.

6. If there are more than 20 results, you can enter the following command to continue iterating through them:

   ```
   it
   ```

7. You can leave the MongoDB shell by entering the following command:

   ```
   exit
   ```

8. ***Before closing down the virtual machine, you should return to the terminal window in which the `mongod` server is running and type CTRL-C to shut down the server.***

Important guidelines

1. If you are using a Mac to edit `ps5_queries.js`, you should disable smart quotes, because they will lead to errors in MongoDB and in our testing. There are instructions for doing so here.

2. Construct the queries needed to solve the problems given below, and put your answers into the `ps5_queries.js` file that you downloaded above. Use a text editor to edit the file, and make sure that you keep the file as a plain-text file.

3. Each of the problems must be solved by means of ***a single query*** (i.e., a single method call). The results of the query should include only the requested information, with no extraneous fields. In particular, you should exclude the `_id` field from the results of your queries whenever it is possible to do so. You do ***not*** need to worry about the order of the fields in the results.

4. For each problem, put your MongoDB method call in the space provided for that problem in the file, assigning it to the results variable that we have provided. We have included a sample query that you can use as a model. ***Do not make any other additions or modifications to this file.***

5. You can see the results of the queries that you have added to the file by entering the following command from your terminal program's command line (assuming that you are in the `mongodb/bin` directory):

   ```
   ./mongo ps5_queries.js
   ```

6. Once you are satisfied with your queries, you should create a file containing the output of your queries by entering the following command from your terminal program's command line:

   ```
   ./mongo ps5_queries.js > output.txt
   ```

   Make sure that the output file looks reasonable, and modify your queries as needed until you get the desired output.

The query problems

5 points each problem

Make sure to follow the guidelines above – in particular, guideline 3.

1. Write a query to find the names and runtimes of all movies in the database that were released in the year 2000 and that have a PG-13 rating.

2. Regina King and Mahershala Ali won Oscars at this year's Academy Awards. Write a query to find out when and where each of them was born. The results of your query should include the name, place of birth, and date of birth of these two actors.

3. Write a query to find all films that won Best Picture in the 1990s (i.e., from 1990-1999). The result documents should include both the name of the film and the year in which the award was given.

   Hints:

   - Because the name of the movie is part of a subdocument in the relevant documents, you will need to use dot notation for the name of the field used to obtain the name, and you will need to surround the field name with quotes.

   - The name of the movie will also end up in a subdocument in the results of the query. For example, here is what one of the result documents should look like:

     ```
     { "year" : 1995, "movie" : { "name" : "Forrest Gump" } }
     ```

4. Write a query to find the names and places of birth of all **directors** in the database who were born in France.

5. Write a query to compute, for each movie rating, the average runtime of movies with that rating.

   Hints/guidelines:

   - You will need to use an aggregation pipeline (i.e., the `db.collection.aggregate` method) with a `$group` pipeline operator as the only stage of the pipeline.
   - Use the field name `average_runtime` for the average values that the query computes.
   - Because the `_id` field is used to specify the field that we are grouping by, you should *not* remove it from the results of this problem.
   - Because you are using an aggregation pipeline, the results of the query will be an *array* of documents.

6. Write a query that finds the same results as the previous problem, but that renames the `_id` field in the results to `rating`. *Hint*: You will need to add a stage to the aggregation pipeline that (1) creates a new field called `rating` that has the same values as the `_id` field, and (2) excludes the `_id` field from the results. Here again, the results of the query will be an array of documents.

7. Write a query to find all movies that have won exactly 4 of the Oscars in the database – no more and no less. You may assume that all films in the database have unique names.

Because the `_id` field is used to specify the field that we are grouping by, you should *not* remove it from the results of this problem.

The results of the query should be an array containing a document for each movie. These result documents should include the name of the movie, and it is also acceptable for them to include the count of the number of Oscars won by the movie. No other fields should be included in the results. Because all of the movies in the results should have a count of 4, you may optionally take steps to remove the count from the final results. You may also rename the `_id` field to make it clearer what it represents, but doing so is not required.

8. Write a query to find the names of all actors or actresses in the database who have acted in a movie in the database that was directed by Steven Spielberg. A given person's name should appear at most once in the result of the query. You should use one of the single-purpose aggregation methods, *not* an aggregation pipeline. The results of the query will be an array of string values.

9. Write a query to find the name of the longest **animated** movie in the database, along with its runtime. Recall that an animated movie has the letter N as part of its genre.

   Hints:

   - One way to solve this problem is to use an aggregation pipeline that includes a `$sort` pipeline stage followed by a `$limit` pipeline stage (and possibly other pipeline stages as well). You can find more information about the `$sort` and `$limit` pipeline operators here.

   - If you use an aggregation pipeline, the result of the query will be an array containing a single document.

10. Write a query that lists the winners of the "big six" Oscars (i.e., the ones in our database) from the **2003** Academy Awards. The result documents should include the award type, the name of the person (if any), and the name of the movie associated with the award. In the case of the Best Picture winner, there is no person, so that field should be omitted. For full credit, you should reformat the results so that there are no subdocuments within the results document for a given award. Use the field name `person` for the name of the person and the field name `movie` for the name of the movie.

Submitting your work for Part II

You should submit only your `ps5_queries.js` file.

Important: You *cannot* drag files stored inside the virtual machine to an application running outside of the virtual machine. Instead, you should submit everything from *within* the virtual machine by using the copy of Firefox that is installed on the VM.

Here are the steps:

1. Click on the icon for Firefox that is found on the desktop of the virtual machine.

2. Login to Gradescope by going to `www.gradescope.com` on Firefox.

3. Click on the box for ***CS 460***.
4. Click on the name of the assignment in the list of assignments. You should see a pop-up window with a box labeled ***DRAG & DROP***. (If you don't see it, click the ***Submit*** or ***Resubmit*** button at the bottom of the page.)
5. Add your file to the box labeled ***DRAG & DROP***. You can either drag and drop the file from its folder into the box, or you can click on the box itself and browse for the file.
6. Click the ***Upload*** button.
7. You should see a box saying that your submission was successful. Click the `(x)` button to close that box.
8. The autograder will perform some tests on your files. **Once it is done, check the results to ensure that the tests were passed.** If one or more of the tests did *not* pass, the name of that test will be in red, and there should be a message describing the failure. Based on those messages, make any necessary changes. Feel free to ask a staff member for help.

   Note: For now, you will only see the results of a test that checks that the required file was submitted, and that you didn't accidentally submit the original, unchanged starter file.
9. If needed, use the ***Resubmit*** button at the bottom of the page to resubmit your work.
10. Near the top of the page, click on the box labeled ***Code***. Then click on the name of the file to view its contents. Check to make sure that the file contains the work that you want us to grade.

Important

- It is your responsibility to ensure that the correct version of every file is on Gradescope before the final deadline. ***We will not accept any file after the submission window for a given assignment has closed, so please check your submissions carefully using the steps outlined above.***
- If you are unable to access Gradescope and there is enough time to do so, wait an hour or two and then try again. If you are unable to submit and it is close to the deadline, email your homework ***before the deadline*** to `cs460-staff@cs.bu.edu`