Tufts University
CS 115: Database Systems
Problem Set 4
Spring 2022

Complete the following exercises to the best of your ability and submit your answers on Gradescope by 11:59 PM on Wednesday, April 13, 2022. There will be a 10% deduction for submissions made by Thursday, April 14 at 11:59 PM, and a 20% deduction for submissions made by Friday, April 15 at 11:59 PM.

You can access Gradescope here: https://www.gradescope.com/. Click Log In and then School Credentials and scroll down to Tufts to log in using your Tufts credentials.

Format your answers as a PDF file. I recommend making a copy of this file as a Google Doc (File > Make a copy), filling in your answers, and then saving it as a PDF (File > Download > PDF document). You can also download this Google Doc as a Word document and save that as a PDF, or use other software entirely.

If you choose to, you can work with 1-2 partners on this assignment. If you worked with partners, mark in Gradescope the partners that you worked with. Make only one submission per group.

Direct any questions about the assignment to Piazza.

Problem 1 (10 points)

Consider the following `CREATE TABLE` commands for a relational database that captures information about songs and the artists who sing them:

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

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));
```

Let's assume that we want to develop an XML version of this database. In addition to translating the relational schema above, we would ideally like to capture the following details:

- The database does not include a `best_chart_rank` value for all songs.
- The database does not include a `primary_genre` value for all artists. If the primary genre of an artist is not specified, we can assume that that the artist's primary genre is `'pop'`.
- Aside from a song's `best_chart_rank` and an artist's `primary_genre`, all of the other song and artist attributes are always included.
- When stored as a string, an artist's date of birth should be of the form `'yyyy-mm-dd'`.
- Every song is sung by at least one artist, but possibly more than one.
- Every artist has sung at least one song, but possibly more than one.

Write an XML DTD for the information in this database that retains as much of the structure and constraints of the relational version of the database as possible. Assume that the entire database will be stored in a single document, and ***use attributes (not elements) to capture relationships between entities***.

Notes:

- Any attributes that represent relationships should use the `ID`, `IDREF`, and/or `IDREFS` constraints as appropriate.
- It may not be possible to capture all of the details mentioned above in your DTD. Do your best to capture as many of them as possible.

```
<!DOCTYPE sings [
      <!ELEMENT sings ((song+ | artist+)*)>


      <!ELEMENT song (name, duration, genre, best_chart_rank?)>
      <!ATTLIST song
            sid ID #REQUIRED
            artists IDREFS #REQUIRED>
      <!ELEMENT name (#PCDATA)>
      <!ELEMENT duration (#PCDATA)>
      <!ELEMENT genre (#PCDATA)>
      <!ELEMENT best_chart_rank (#PCDATA)>


      <!ELEMENT artist (name, label, dob)>
      <!ATTLIST artist
            aid ID #REQUIRED
            primary_genre CDATA “pop”
            songs IDREFS #REQUIRED >
      <!ELEMENT name (#PCDATA)>
      <!ELEMENT label (#PCDATA)>
      <!ELEMENT dob (#PCDATA)>
]>
```

Problem 2 (15 points)

Suppose you need to create a database to track teams in a soccer league. Each team has a team name, a team location, and information about the people on the team. Each person needs to have a jersey number, first and last name, email address, and position. Each team has at least 11 players on it. Email addresses use this pattern: *Firstname*.*Lastname*@myleague.com, where the first and last names are capitalized.

a. Give an example JSON record that shows the information for a single team that reflects the description above. The information for players should be embedded inside of the team document. You can make up values for the record as needed.

```
{
        "name" : "Inter Milan",
        "location" : "Milan, Italy",
        "players" :
         [
                {
                        "firstname" : "Samir", "lastname" : "Handanović", "email" :
"Samir.Handanović@myleague.com", "jersey" : "1", "position" : "GK"},
                {
                        "firstname" : "Milan", "lastname" : "Škriniar", "email" :
"Milan.Škriniar@myleague.com", "jersey" : "37", "position" : "CB"},
                {
                        "firstname" : "Stefan", "lastname" : "de Vrij", "email" :
"Stefan.DeVrij@myleague.com", "jersey" : "6", "position" : "CB"},
                {
                        "firstname" : "Alessandro", "lastname" : "Bastoni", "email" :
"Alessandro.Bastoni@myleague.com", "jersey" : "95", "position" : "CB"},
                {
                        "firstname" : "Ivan", "lastname" : "Perišić", "email" :
"Ivan.Perišić@myleague.com", "jersey" : "14", "position" : "LWB"},
                {
                        "firstname" : "Denzel", "lastname" : "Dumfries", "email" :
"Denzel.Dumfries@myleague.com", "jersey" : "2", "position" : "RWB"},
                {
                        "firstname" : "Hakan", "lastname" : "Çalhanoğlu", "email" :
"Hakan.Çalhanoğlu@myleague.com", "jersey" : "20", "position" : "CM"},
                {
                        "firstname" : "Marcelo", "lastname" : "Brozović", "email" :
"Marcelo.Brozović@myleague.com", "jersey" : "77", "position" : "CM"},
                {
                    "firstname" : "Nicolo", "lastname" : "Barella", "email" :
"Nicolo.Barella@myleague.com", "jersey" : "23", "position" : "CM"},
            {
                    "firstname" : "Lautaro", "lastname" : "Martínez", "email" :
"Lautaro.Martínez@myleague.com", "jersey" : "10", "position" : "ST"},
            {
                    "firstname" : "Edin", "lastname" : "Džeko", "email" :
"Edin.Džeko@myleague.com", "jersey" : "9", "position" : "ST"}
        ]
}
```

b. Define a JSON schema that matches the description above (and the example JSON record that you devised). Disallow all records that do not conform to the given format.

(Added 4/6) Note: you can capture the pattern for email addresses using regular expressions in JSON. We covered one example of this in lecture, but you can find more information here: https://json-schema.org/understanding-json-schema/reference/regular_expressions.html

You can try validating the schema you wrote in part (a) against the schema you write in part (b) here: https://www.jsonschemavalidator.net/

```
{
        "$schema": "https://json-schema.org/draft/2020-12/schema",
        "$id": "https://example.com/soccerteams.schema.json",
        "title" : "Soccer Teams",
        "description" : "A soccer team",
        "type" : "object",
        "properties" : {
                "name" : {"type" : "string"},
                "location" : {"type" : "string"},
                "players" : {
                        "type" : "array",
                        "items" : {
                                "firstname" : {"type" : "string"},
                                "lastname" : {"type" : "string"},
                                "email" : {
                                        "type" : "string",
                                        "pattern" : "^[A-Z][a-z]+[.]{1}[A-Z][a-z]+@myleauge.com$"
                                },
                                "jersey" : {
                                        "type" : "integer",
                                        "minimum" : 0
                                },
                    "position" : {"type" : "string"}
                },
                "minItems" : 11,
                "uniqueItems" : true,
                "required" : ["jersey", "firstname", "lastname", "email", "position"]
            }
        },
        "required" : ["name", "location", "players"]
}
```

Problem 3 (20 points)

In lecture, we saw how the movie/Oscar relational database could be converted to MongoDB collections. Now, consider the following `CREATE TABLE` commands for a relational database maintained by a bookseller:

```
CREATE TABLE Author(id CHAR(5) PRIMARY KEY, name VARCHAR(64), dob DATE);

CREATE TABLE Book(isbn CHAR(13) PRIMARY KEY, title VARCHAR(128),
    publisher VARCHAR(64), num_pages INTEGER, genre VARCHAR(10),
    numInStock INTEGER);

CREATE TABLE Wrote(authorID CHAR(5), book CHAR(13),
    PRIMARY KEY(authorID, book),
    FOREIGN KEY authorID REFERENCES Author(id),
    FOREIGN KEY book REFERENCES Book(isbn));

CREATE TABLE Sales(date VARCHAR(10), time VARCHAR(5), book CHAR(13),
    numSold INTEGER, PRIMARY KEY(date, time, book),
    FOREIGN KEY book REFERENCES Book(isbn));
```

Here are some example tuples from these tables:

```
from Author: ('12345', 'David Sedaris', '1956-12-26')
from Book: ('9780316154697', 'Let's Explore Diabetes With Owls',
            'Little, Brown', 288, 'humor', 20)
from Wrote: ('12345', '9780316154697')
from Sales: ('2021-04-21', '12:00', '9780316154697', 2)
```

a. Say that you want to convert these tables to a set of collections for a MongoDB database. One way to capture the relationships (`Wrote` and `Sales`) 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 above tuples from the relational database would be represented as one or more documents in MongoDB.

Notes:

- 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. For example, the book in the tuples above only has one author, but more generally books can have multiple authors, so your document design should account for that.

```
Author Document
{
      '_id' : '12345',
      'name': 'David Sedaris',
      'dob': '1956-12-26',
      'books' : ['9780316154697']
}

Book Document
{
      '_id' : '9780316154697',
      'title' : 'Let's Explore Diabetes With Owls',
      'publishers' : ['Little, Brown'],
      'num_pages' : 288,
      'genres' : ['humor'],
      'numInStock' : 20,
}

Sales Document
{
      'date' : '2021-04-21',
      'time' : '12:00',
      'book' : '9780316154697',
      'num_sold' : '2'
}
```

}

b. 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 example tuples would be captured as one or more documents in MongoDB.

Notes:

- The same guidelines as those from part (a) also apply here.
- Assume that we are expecting queries that relate books and their authors to mostly be concerned with book names and author names, but not other attributes related to books and authors. For example, a common query would be something like "What are the names of books written by JK Rowling?" An uncommon query would be something like "What are the dates of birth for authors who have written science fiction books?"
- If you want to make any additional assumptions, state them in your answer.

```
Author Document
{
      '_id' : '12345',
      'name': 'David Sedaris',
      'dob': '1956-12-26'
      'books' :
      [
            {
                  'isbn' : '9780316154697',
                  'title' : 'Let's Explore Diabetes With Owls'
            }
      ]
}
Book Document
{
      '_id' : '9780316154697',
      'title' : 'Let's Explore Diabetes With Owls',
      'publishers' : ['Little, Brown'],
      'num_pages' : 288,
      'genres' : ['humor'],
      'numInStock' : 20,
      'sales' :
      [
            {
                  'date' : '2021-04-21',
                  'time' : '12:00',
                  'num_sold' : 2
            }
      ]
}
```

Problem 4 (55 points)

Setup

1. Install MongoDB

Follow the appropriate set of instructions (macOS or Windows) on Canvas to get MongoDB installed and running. Note that if you already set up MongoDB for our in-class exercises, you don't need to repeat these steps.

2. Download Necessary Files

Similar to how we created the pokemon collection in the MongoDB setup guide, we will now create collections for the assignment. Download the .json files below, which contain the data for the collections, as well as the .js file that we will use for writing queries.

Important

To simplify matters, you should store **all** of these files in `Desktop/mongodb/bin` – i.e., the `bin` subfolder of the `mongodb` folder that should now be on your Desktop.

- movies.json
- people.json
- oscars.json
- ps4_queries.js

3. Start MongoDB Server

Using the instructions in the setup guide, start the MongoDB server in a Command Prompt (Windows) or Terminal (macOS). From within the `Desktop/mongodb/bin` directory, this command should be:

For Windows:
```
mongod --dbpath .
```

For macOS:
```
./mongod --dbpath .
```

4. Create the Collections

1. Leaving the `mongod` server running in the initial Command Prompt/Terminal window, open a second, separate Command Prompt (Windows) or Terminal (macOS) window.

2. `cd` into the `Desktop/mongodb/bin` directory as you did before.

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

   For Windows:
   ```
   mongoimport --db imdb --collection movies --file movies.json --jsonArray --drop --legacy
   ```

   For macOS:
   ```
   ./mongoimport --db imdb --collection movies --file movies.json --jsonArray --drop --legacy
   ```

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

   For Windows:
   ```
   mongoimport --db imdb --collection people --file people.json --jsonArray --drop --legacy
   ```

   For macOS:
   ```
   ./mongoimport --db imdb --collection people --file people.json --jsonArray --drop --legacy
   ```

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

   For Windows:
   ```
   mongoimport --db imdb --collection oscars --file oscars.json --jsonArray --drop --legacy
   ```

   For Windows:
   ```
   ./mongoimport --db imdb --collection oscars --file oscars.json --jsonArray --drop --legacy
   ```

Performing queries in MongoDB

Once you have everything configured, you can perform queries by taking the following steps:

1. Make sure that the `mongod` server is running in one Terminal/Command Prompt window (see the earlier instructions).

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

3. Make sure you are in the `Desktop/mongodb/bin` directory as you did earlier.

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

   On Windows:
   ```
   mongo imdb
   ```

   On macOS:
   ```
   ./mongo imdb
   ```

   You should see some messages, followed by the shell's command prompt:
   ```
   >
   ```

5. Enter the appropriate method calls for your queries. If you simply enter the method call, you should automatically see up to the first 20 results.

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

6. You can leave the MongoDB shell by entering the following command:
   ```
   exit
   ```

7. At the end of your MongoDB session, don't forget to return to the terminal window in which the `mongod` server is running and type CTRL-C to shut down the server.

Important guidelines

1. Construct the queries needed to solve the problems given below, and put your answers into the ps4_queries.js file that you downloaded above. Use a text editor to edit the file.

2. If you are using a Mac to edit ps4_queries.js, you should disable smart quotes, because they will lead to errors in MongoDB. There are instructions for doing so here.

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 the Terminal/Command Prompt (assuming that you are in the `Desktop/mongodb/bin` directory):

   On Windows:
   ```
   mongo ps4_queries.js
   ```

   On macOS:
   ```
   ./mongo ps4_queries.js
   ```

6. To make it easier to see all of the results, you can store them in a file by entering the following command:

   On Windows:
   ```
   mongo ps4_queries.js > output.txt
   ```

   On macOS:
   ```
   ./mongo ps4_queries.js > output.txt
   ```

7. Make sure that the output file looks reasonable, and modify your queries as needed until you get the desired output. (Added 4/7) When the queries look correct, copy them over into your PDF for submission (you don't need to submit ps4_queries.js).

Query problems

Make sure to follow the guidelines above.

Query 1 (5 points)

Write a query to find the names and runtimes of all movies in the database that have an PG rating and a runtime greater than 120 minutes.

```
      db.movies.find( { rating: "PG", runtime: {$gt: 120} },
                      { name: 1, runtime: 1, _id: 0 } )
```

Query 2 (5 points)

Write a query to find the places of birth of Will Smith and Chris Rock. The result documents should contain the name of each person and their place of birth.

```
      db.people.find( { $or:[{name: "Will Smith"}, {name:"Chris Rock"}]},
                      {name: 1, pob: 1, _id:0} )
```

Query 3 (5 points)

Write a query to find the names of all people who have won a supporting actor/actress Oscar from 2015 to the present. The result documents should include the name of the person and the year in which the award was given.

Hints:

- Because the name of the person is part of a subdocument in the relevant documents, you will need to use dot notation.
- The name of the person 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" : 2019, "person" : { "name" : "Mahershala Ali" } }`

```
      db.oscars.find( { $or:[{type: "BEST-SUPPORTING-ACTOR"},
                             {type: "BEST-SUPPORTING-ACTRESS"}],
                        year: {$gte: 2015}},
                      {year: 1, _id: 0, "person.name": 1} )
```

Query 4 (5 points)

Write a query to find the names and dates of birth of all actors in the database who were born 40 years ago (i.e., in 1982).

```
      db.people.find( { dob:/^1982/ },
                      {name: 1, dob: 1, _id:0} )
```

Query 5 (5 points)

Write a query to find the names of directors who have directed a movie in which Daniel Radcliffe acted. 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.

```
      db.movies.distinct( "directors.name",
                          { "actors.name": "Daniel Radcliffe"} )
```

Query 6 (5 points)

Write a query to find all people who have won 3 or more of the Oscars in the database. You may assume that all people in the database have unique names.

Notes:

- You will need to use an aggregation pipeline.
- 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. Here is what one of the results documents should look like: `{ "_id" : "Meryl Streep", "oscar_count" : 3 }`
- Best Picture Oscars do not have a person associated with them. As a result, you will see a subgroup for those Oscars in which the `_id` value is `null`. You should not remove that subgroup from the results of this query.

```
      db.oscars.aggregate( {$group: {
                                 _id: "$person.name",
                                 oscar_count: {$sum: 1}}
                              },
                             {$match: {oscar_count: {$gte: 3}}} )
```

Query 7 (5 points)

Write a query that finds the same results as the previous problem, but that also takes the steps needed to:

- Exclude the results for `null` (note: `null` should not be surrounded by quotes)
- Rename the `_id` field in the result documents to person by (1) creating a new field called `person` that has the same value as the `_id` field and (2) excluding the `_id` field from the results. Here is what one of the result documents should look like:
  `{ "oscar_count" : 3, "person" : "Meryl Streep" }`

```
db.oscars.aggregate( {$group: {
                        _id: "$person.name",
                        oscar_count: {$sum: 1}}
                      },
                     {$match: {oscar_count: {$gte: 3}}},
                     {$match: {_id: {$ne: null}}},
                     {$addFields: {person: "$_id"}},
                     {$project: {_id: 0}} )
```

Query 8 (5 points)

Write a query to compute the average runtime of the top-10 grossing movies (i.e., the movies with an earnings rank between 1 and 10). The result of the query should be a single document that looks like this: `{ "avg_runtime" : n }`, where n is the computed average runtime.

Hint: to compute an average over the set of all documents, you can use a `$group` stage in an aggregation where the `_id` field has a constant value, such as `null`.

```
db.movies.aggregate( {$match: {earnings_rank: {$lte: 10}}},
                     {$group: {
                        _id: null,
                        avg_runtime: {$avg: "$runtime"}}
                      },
                     {$project: {_id: 0}} )
```

Query 9 (5 points)

Write a query to find the name and runtime of the shortest top-grossing movie in the database.

Hints:

- For the purposes of this query, a top-grossing movie is any movie that has an earnings_rank field. Use the $exists operator for this purpose.
- Your query can assume that there is exactly one top-grossing movie that is the shortest (i.e., there are not multiple movies which tie as the shortest).

```
db.movies.aggregate( {$match: {earnings_rank: {$exists: true}}},
                     {$sort: {runtime: 1}},
                     {$limit: 1},
                     {$project: {_id: 0, name: 1, runtime: 1}} )
```

Query 10 (10 points)

Write a query that determines how many actors or actresses have won **both** a supporting acting award and a non-supporting acting award (i.e, won Best Supporting Actor and Best Actor, or won Best Supporting Actress and Best Actress). The result of your query should be a single number. More precisely, it should be a single name:value pair in which the name is “count” and the value is the single number that you need to compute.

Hint: You may want to consider using the $cond operator. If you choose to use the `$cond` operator, make sure you are using the MongoDB 4.4 version of its syntax (the documentation of which was just linked in the previous sentence).

```
db.oscars.aggregate(
            {$match:
                  {$or: [
                        {type: "BEST-ACTOR"},
                        {type: "BEST-ACTRESS"},
                        {type: "BEST-SUPPORTING-ACTOR"},
                        {type: "BEST-SUPPORTING-ACTRESS"}]}},
            {$group: {_id: {name: "$person.name", type: "$type"}}},
            {$group: {_id: "$_id.name", count: {$sum: 1}}},
            {$match: {count: {$gte: 2}}},
            {$group: {_id: "total", count: {$sum: 1}}},
            {$project: {_id: 0, count: 1}} )
```