Edit on Mar 21st at 9.20pm:
Some edits have been made which are highlighted in yellow below.

Assignment #4

Marking

- README.txt file: 0 marks if submitted as required, -10 otherwise.
- A4T1.py: 10 marks if it produces a correct A4dbNorm database, 0 if it does not. In the latter case the TAs will use a correct database in order to test the other tasks.
- A4T2.py: 10 marks if it produces a correct A4dbEmbed database, 0 if it does not. In the latter case the TAs will use a correct database in order to test the other tasks.
- A4Q1Norm.py, A4Q1Embed.py, … A4Q4Norm.py, A4Q4Embed.py: 10 marks each if they produce the correct answer, 0 otherwise.

Preamble

The main goal is to write a few queries using MongoDB as the underlying database system, so that you can appreciate the differences between using embedded and normalized document models. The queries will be embedded into separate Python applications that will make use of the PyMongo library (you can find a sample Python application using PyMongo to connect to a MongDB here). As done in previous assignments you must follow strictly the submission instructions provided below.

Tasks

You are provided with two JSON files, namely songwriters.json and recordings.json that contain information about songwriters and songs. Their fields have the following semantics:

songwriters.json documents:

- _id
  - ObjectId string, the id given to the document by mongodb
- songwriter_id
  - string (“Primary Key”)
  - Id of songwriter
- fans
  - int
  - number of fans for the songwriter
- name
  - string
  - Name of the songwriter
- reputation
  - int
  - reputation (popularity score) ranking from 0-100
- recordings
  - Array of strings
  - size: min = 0, average = 4.29, max = 83

    - These values are given from a sample of 1000 documents
    - Each entry in the array is a recording id that relates to recording_id in the recordings collection
    - “Foreign key” to recordings.recording_id

recordings.json documents:

- _id
    - ObjectId string, the id given to the document by mongodb
- recording_id
    - string (“Primary Key”)
    - Id of the recording
- songwriter_ids
    - Array of strings
    - size: min = 1 average = 1.05 max = 3
    - These values are given from a sample of 1000 documents
    - Each entry in the array is a songwriter id that relates to songwriter_id in the songwriter table
    - “Foreign key” to songwriters.songwriter_id
- rhythmicality
    - float
    - rating from 0 to 1
- length
    - int
    - length of the recording in milliseconds
- name
    - string
    - The name of the recording
- reputation
    - int
    - The reputation (popularity score) of the recording from 0-100
- issue_date
    - Date
    - date object in iso 8601 format, eg. 1949-01-01T00:00:00.000+00:00 or as a Unix timestamp string.

Hint for Task 1 and Task 2. You may want to use the ***bson.json_utils.loads*** function to automatically convert dates and ids into the proper format. Do not install the *bson* module, PyMongo comes with its own *bson* package.

Task 1

Write a Python application **(A4T1.py)** which will take as input the JSON files above and (1) create a MongoDB database called *A4dbNorm* containing two sets of documents, one for each JSON dataset above. That is, effectively a set of normalized MongoDB documents. For querying purposes you need to keep in mind the intrinsic PK/FK-like dependencies between such tables. The input (the JSON datasets) must be assumed to be in the same folder as the Python application and should be hard coded in the application, i.e., not requiring any user input. The output of this task is a MongoDB database to be named A4dbNorm, containing two collections “*songwriters”* and “*recordings”*, which will

reside in the local Mongo data repository but that will not be submitted. You will need to submit only the A4T1.py file.

Task 2

Write a Python application **(A4T2.py)** which will create another MongoDB database called *A4dbEmbed* containing a single collection called "*SongwritersRecordings"*. Recordings where all the recordings (in recordings.json) associated with a songwriter (in songwriter.json) will be embedded along with the corresponding songwriters' documents. Note that songs associated with multiple songwriters should be embedded in each of those songwriter's documents. As an example of an embedded collection, assume you have two collections, called Profs and Courses with the possible (partial) schemas:

Profs:

```
[
        {
                prof_id: "luigi1983",
                prof_email: "luigi@marioland.game",
                teaches: [ "CMPUT291", "CMPUT391" ]
        } ...
]
```

Courses:

```
[
        {
                course_id: "CMPUT291",
                course_name: "Intro to File and Database Management",
                level: "undergraduate"
        },
        {
                course_id: "CMPUT694",
                course_name: "Spatiotemporal Data Management",
                level: "graduate"
        }, ...
]
```

An embedded document ProfCourses could look like the following:

```
[
        {
        prof_id: "luigi1983",
                prof_email: "luigi@marioland.game",
                teaches: [{
                        course_id: "CMPUT291",
                        course_name: "Intro to File and Database Management",
                        level: "undergraduate"
                }, {
                        course_id: "CMPUT694",
                        course_name: "Spatiotemporal Data Management",
                        level: "graduate"
                }]
        }, ...
]
```

The output of this task is a database to be named A4dbEmbed which will reside in the local Mongo data repository but it will not be submitted either. You will need to submit only the A4T1.py/A4T2.py file. Your applications must accept the port number as a command-line argument. Please note that we will be running the applications using the following command:

python3 A4Tx.py <port number>

For example, to run the application on port 27017, you would use the following command:

python3 A4Tx.py 27017

Task 3

Next, you need to use MongoDB's Query Language (MQL) via a Python application to answer the following queries (note that, for sake of simplicity, none of them will require any user input). The query's answers should not be saved onto a file, rather they must be output to stdout (the screen).

Q1: Find the ids and names of each songwriter that has at least one recording and the number of recordings by each such songwriter.
Sample output:

```
{'_id': ObjectId('61002ea023062862f66c3af3'), 'songwriter_id':
'2mnD3lfc9infcjV6vHIV6b', 'name': 'La Roscada', 'num_recordings': 5}
{'_id': ObjectId('61002ea023062862f66c3af4'), 'songwriter_id':
'7y4tFWcESTjfDcPSzqjOzL', 'name': 'Lawson-Haggart Jazz Band', 'num_recordings': 4}
{'_id': ObjectId('61002ea023062862f66c3af5'), 'songwriter_id':
'3NfYRC8K0hTE1tWGu5AvMR', 'name': 'Sterling Brown', 'num_recordings': 3}
{'_id': ObjectId('61002ea023062862f66c3af6'), 'songwriter_id':
'1OCPhFtvkZDLUJJkrJfD2G', 'name': 'The De Castro Sisters', 'num_recordings': 10}
...
```

Q2: Write a query to get the average rhythmicality for all recordings that have a recording_id beginning with "70".
Sample output:

```
{'_id': '', 'avg_rhythmicality': 0.6285}
```

Q3: Find the sum of the length of all recordings for each songwriter along with the songwriter's id.
Sample output:

```
{'_id': '4ETXyV9H1p2P1XYgXXTjiO', 'total_length': 243067, 'songwriter_id':
'4ETXyV9H1p2P1XYgXXTjiO'}
{'_id': '7v8veUQH2fro2QjkOKS7vB', 'total_length': 253800, 'songwriter_id':
'7v8veUQH2fro2QjkOKS7vB'}
{'_id': '7EUjDdRgLHcgFIL4sfhvFo', 'total_length': 513787, 'songwriter_id':
'7EUjDdRgLHcgFIL4sfhvFo'}
{'_id': '4RWsSjLE6TMBwdYzHj2NiO', 'total_length': 781506, 'songwriter_id':
'4RWsSjLE6TMBwdYzHj2NiO'}
...
```

Q4: For each recording that was issued after 1950-01-01, find the recording name, songwriter name and recording issue date.
Sample output:

```
{'_id': ObjectId('61002ea023062862f66c3af3'), 'name': 'La Roscada', 'r_name':
'Preguntá vos chacarera (En Vivo)', 'r_issue_date': datetime.datetime(1950, 7, 27, 0,
0)}
{'_id': ObjectId('61002ea023062862f66c3af3'), 'name': 'La Roscada', 'r_name': 'Con tu
vida en pie (En Vivo)', 'r_issue_date': datetime.datetime(1950, 7, 27, 0, 0)}
{'_id': ObjectId('61002ea023062862f66c3af3'), 'name': 'La Roscada', 'r_name': 'La
arenosa (En Vivo)', 'r_issue_date': datetime.datetime(1950, 7, 27, 0, 0)}
{'_id': ObjectId('61002ea023062862f66c3af3'), 'name': 'La Roscada', 'r_name': 'Tanta
luz (En Vivo)', 'r_issue_date': datetime.datetime(1950, 7, 27, 0, 0)}
...
```

For each of the four queries above you will create two Python applications both connecting to a MongoDB server, but one using the **A4dbNorm** database and the other one using **A4dbEmbed** database, created in Tasks 1 and 2, respectively. Each application will process one query and output the query's result.

Your applications will be named **A4QxNorm.py** and **A4QxEmbed.py**, where x is the number of each query above. That is, your application A4Q1Norm.py will process query Q1 using A4dbNorm, and your application A4Q1Embed.py will do the same using A4dbEmbed, so on and so forth. Needless to say, both applications for the same query must yield the same result. Your applications must accept the port number as a command-line argument.

At the end you will produce and submit 10 Python applications: 2 to create the necessary MongoDB databases using the two given JSON files and 2 for each of the 4 queries Q1-Q4.

Submission

It is VERY important that all databases, collections, fields, and file names are EXACTLY as specified above. Failure to do so may yield loss of marks, as some of the tasks may be performed using databases other than the ones you produced/used.

There should be **only ONE submission per group** (i.e., only one student should submit on behalf of their group), and each student must be in ONE group. **If a student is listed in more than one group they will receive ZERO as their mark for the assignment.**

Note that eClass does not support versioning of submissions, and each new submission replaces your previous one. This makes last-minute submissions somewhat risky. **Avoid last-minute submissions, and check your submissions after an upload (ideally on the lab machines) to make sure the right content is uploaded.**

Your submission must include:

- All required 10 .py files, namely: A4T1.py, A4T2.py, A4Q1Norm.py, A4Q1Embed.py, …, A4Q4Norm.py and A4Q4Embed.py. The databases you created, namely A4dbNorm and A4dbEmbed must **NOT** be submitted, **NOR** should the JSON files provided to you, as different ones will be used at marking time.
- A README.txt file, which must contain at the top, the group number, the CCIDs and the names of all group members. It should also contain the list of resources used (other than regular course material) and/or people you collaborated with (as much as it is allowed as per our course's policy) or the single line "We declare that we did not collaborate with anyone outside our own group in this assignment". Please make sure that the README file also contains which version

of used libraries, e.g., pymongo, where used. If you have any comments you can add them on the file too.

All 11 files above must be embedded into a **single .tgz file** named **GroupXXA4.tgz** where XX is the group number.

Appeals

If you don’t agree with the marking and/or have questions about it please contact your TA (not the instructor), the instructor will be contacted by the TAs if there’s a need for a “second opinion.”