Homework Assignment #3

(MongoDB)

After completing the previous assignment successfully, your reputation as a ZotMusic data engineer has been skyrocketing! Based on your fast learning skills you have been approached for a part-time job at a startup that wants to perform some queries on top of a ZotMusic cluster. You have been asked to set up a cluster for them in the cloud, to get it running and ready to go, to load the pre-acquired data, and to run some initial analytical queries to give them some insights into the data that they have.

Get Ready...

Start by reading through the setup instructions **carefully** in order to create an Atlas account with MongoDB and set up a cluster using the provided steps. Afterward, you should installMongoDB Compass in order to load data successfully. You will also make use of a Jupyter notebook for this assignment.

Get (Data) Set...

Your next step is to grab yourself a copy of the new json-based ZotMusic data. **See the setup instructions document for details.** Since MongoDB is a document database, the schema is self-describing (i.e., within the data itself); you will be unveiling some information about the data's schema as the assignment progresses. A bit of other sample data has also been provided, along with a few queries, to help jumpstart your interaction with the MongoDB query language (MQL).

Go...!

Now that you have the data, it's time to get to work. You will be using Compass to answer the schema analysis questions and then the rest of the MongoDB queries will be performed using a Jupyter Notebook where you will write your queries using Python (and Pymongo). You may also want to use Compass to "unit test" your query filters first; that's up to you.

1. You will start by exploring the schema for several of the ZotMusic collections here. In order to do that you will need to connect to your cluster using Compass and click on the ZotMusic database on the left panel. After that, choose the desired collection. Once a collection is chosen, you can then navigate to the **Schema** tab to analyze a sample of the stored data. This will help you answer the following questions:
    A. In the **records** collection, what is the data type for the field *released_by*?
    B. In the **records** collection, do you see any field that is not present in the original data file (records.json)? If yes,
        i. What is the name of the newly added field?
        ii. What is its data type?

iii. How can we interpret the new field? (**Hint**: Check the MongoDB documentation on how it is constructed and what its components are.)
iv. Is it unique for all the documents in the collection?

C. In the **reviews** collection, which field or nested field can you use to identify the name of a review's creator by linking it with the **users** collection?
D. From the **users** collection, what is the data type for the field named *subscription* and what are its values that are displayed in the analysis?
E. In the **users** collection, what is the type of field *genres,* and on average, how many genres does a user maintain? (Note: You don't have to write a query to compute the average. Simply use the value provided by the Compass schema analysis feature and round it to the nearest integer.)

2. Now we will begin exploring the ZotMusic data with the help of the **find** or **count_documents** API of MQL. Do not process/filter the data in Python outside of the **find** or **count_documents** methods. The goal of this assignment is for you to get familiar with "pymongo". We know that you are likely already a Python expert!
Note: If you want, you can export queries that you first create and run on Compass as language-specific (Python) code that can then be run in Jupyter if you want. (This is optional.).
   A. There is a single with the title "Big worry" in ZotMusic. Print this document.
   B. Print the titles and release dates of the 3 most recently released albums on the platform.
   C. Notice that in the original ER model in HW 1, listeners and artists are not disjoint, i.e., a user can be both a listener and an artist. Find the number of users who are both listeners and artists. **Only print the count**.
   D. Count the total number of records on ZotMusic that are singles in the "Pop" genre and released after 2023-06-30 or are albums in the "Jazz" genre and released before 2023-01-01. **Only print the count**.
   E. Find the email address and full_name defined as "last_name, first_name" of all listeners who are interested in at least 9 genres and have a complete mailing address associated with their account. A complete mailing address means all the fields (Street, City, State, ZIP) are filled. Order the results by email in ascending order. [Hint: Look for the "**$exists**" operator in MQL when trying to check if a field is missing.]

   We will now be shifting API gears and will make use of the **aggregation pipeline** feature i.e., (the **aggregate** API) that MongoDB provides for the rest of the queries. Do not process/filter the data in Python outside of the **aggregate** method.

   F. Using the MQL aggregation pipeline API, determine and print the total number of listeners who joined in 2022 and have a subscription type of "yearly".
   G. Find all the users with the top 4 "total number of posted reviews". For example, if there are 7 users (u1, u2, u3, u4, u5, u6, u7) who have posted 20, 15, 13, 11, 11, 8, 5 total reviews respectively, you should return users u1, u2, u3, u4, and u5.

Print only the user user_id, total reviews posted, and their dense rank. [**Hint**: look for the **$denseRank** aggregation operator in the MQL documentation.]

H. The creator of the record with id **"`record_qGHboIDL`"** is trying to send gifts to the fans of this record but notice that not all the fans have entered their complete mailing address on ZotMusic. Find all the listeners that have given this record a rating of at least 4 and have complete address info listed on their account. Print the user_id, nickname, and address of these users.

I. Your boss would like you to compile a playlist of "Chill Music Essentials" by picking songs on the platform based on their BPMs and the average ratings of the record that the song comes from. Find all the songs at ZotMusic with a BPM less than 63 and belonging to records with an average rating of 4.0 according to the reviews. (Note: unlike HW1, there is no need to do a weighted average here.) Print the record's id, record's title, song's track_number, and song's title and include a random sample of 5 result documents. [Hint: Look for the **$sample** pipeline stage in the MQL documentation.]

J. In order to demonstrate your impressive mastery of MQL aggregation pipelines' stages, your boss would like for you to convert the following SQL statement into an equivalent MQL pipeline and execute it. [Hint: Look for the **$addToSet** aggregation operator in the MQL documentation.]

```
SELECT u.nickname, u.email
FROM users u
JOIN sessions s ON u.user_id = s.user_id
JOIN records r ON r.record_id = s.record_id
WHERE r.released_date > '2023-01-01'
  AND s.replay_count > 3
  AND s.device = 'mobile-app'
GROUP BY u.user_id, u.nickname, u.email, u.joined_date
HAVING COUNT(DISTINCT r.genre) > 6
ORDER BY u.joined_date DESC
LIMIT 2;
```

What To Turn In

When you have finished your assignment you should use Gradescope to turn in a **PDF** file that lists all of the answers to the questions together with the **results** of running each cell. Please follow the following steps in order to generate the file for submission.

1. Develop and keep your answers in a .ipynb file using Jupyter. You can make use of the template file on Canvas.
2. For the first (non-MQL) question, put your answers in one or more cells as comments or text. (Recall that comments in Python begin with a hashtag for each line.)

3. Put each of your queries for the second question in a separate appropriately commented cell and make sure that they actually run and print out their results after every query. To produce your to-be-turned-in output, restart the Jupyter kernel and run all your queries one last time in your notebook; they should all be run together and in sequence from top-to-bottom right before turning it in.
4. Remove (or just obfuscate) your Atlas login information from the notebook.
5. Make sure all of your code can be clearly seen without having to scroll to the right in any of the cells. (Break your code into multiple lines if necessary.)
6. Save your .ipynb file, with all of the cells and their results, as a PDF. To do that nicely in Jupyter, if you have LaTex available, you should use File -> Download as -> PDF via LaTex (.pdf). If not, you can use File -> Print Preview and then PDF-print the result. Once you have done this, be sure to double-check to see that all of your code is visible in the resulting PDF file.
7. Turn in the PDF!