Lab 10 – MongoDB

CC5212-1 – May 22, 2024

Today we will learn about MongoDB. You will need to submit all commands you enter into the MongoDB console as a text file. You do not need to submit data returned.

- Let's get into the MongoDB shell and load the database
  - Log into the cluster as `uhadoop` and run `mongo`. Ignore the warnings.
  - Run `show dbs` to see the databases available.
  - Run `use tvdb`. This is the database we'll use. Afterwards run `db` to see the current database. Run `show collections` to see the collections in the database.
- Let's run some high-level queries to see what's in the `series` collection.
  - Query for all documents in the collection `series`.
  - Repeat that last step but make the output look pretty with proper indentation.
  - Get a count of the number of documents in the collection.
  - Get an array of all the unique values for the key `"type"` in the collection.
- Okay, now let's add some new data in there. We're going to grab some data from the TVMaze API, which you can access at: `http://www.tvmaze.com/api`.
  - Use the API to find a TV series *not already in the collection `series`*. (To help you choose a unique series, you can get an array of all the unique values for the key `"name"` in the collection.)
    - For example, `http://api.tvmaze.com/singlesearch/shows?q=the-wire` will return data about the first series returned for the keyword match "`the wire`". If you like, you can just change the `the+wire` part of the URL directly, making sure to replace spaces with '`+`'.
    - Your document should not start with a field "`score`"; if it does, copy the sub-document that is the value for show. (The document you enter should start with a field "`id`".)
    - You can use `http://jsonprettyprint.com/` to make the JSON look nice if you want, but it will not matter once loaded into MongoDB.
    - Note that the `id` key in the data is not going to conflict (or have anything to do with) the `_id` key that MongoDB uses. The `id` key in the data is like any other field. However, we will use it later to get more data from the API, so you can take a note of it if you like.
  - Load the data as a document into the `series` collection in MongoDB.
- Time for some queries with selection on the `series` collection. For the moment, you do not need to use projection: return the entire documents that match. As a hint, keys and values are case sensitive.
  - Find all series of status Ended.
  - Find all series with a runtime less than 45.
  - Find all series with a rating average less than 9.0.
  - Find all series in the genre Comedy (i.e., where at least one genre value is Comedy).
  - Find all series with genre Science-Fiction *and* Adventure.
  - Find all series with a rating average greater than 8.5 and genre Crime or Drama.
  - Find all series with (exactly) two genres.
- We don't always want to see the full documents in the results; let's try some projection.
  - Return only the name and _id of all series of status Ended.
  - Return only the name of all series of status Ended.
  - Return the full documents of all series of the network HBO omitting only the summary and _id field.
  - Return the name of each series and the last genre mentioned in its array.
- Now we will try out a couple of aggregation examples.
  - Count how many series there are shown on Adult Swim (return the count).
  - Take the maximum average rating for each genre (return all genres and the maximum of the average rating of their series).
  - Sort all English-language series by average rating (top-rated first) and print only their names.
- Let's try the text search feature.
  - Use `$text` to search for documents in `series` mentioning `alcoholic grandfather`.

    - Actually this will not work for whoever is first. They will need to create an index with:
        * `db.series.createIndex( { summary: "text" } )`

      That person can check out `https://docs.mongodb.com/manual/core/index-text/`. Thanks that person! Or if they don't much like the idea of helping others, they can also drop the index when they're done with the keyword search:
        * `db.series.getIndexes()`
        * `db.series.dropIndex("` NAMEOFTEXTINDEX `")`
    - Try another keyword query search.
- Last part: let's load some crew data for your series from TVMaze.
    - Grab the data from `http://api.tvmaze.com/shows/` ID `/crew` where ID is the TVMaze (not MongoDB) id of the series you loaded earlier. Have a look! Notice it doesn't mention the series. How are we gonna link the series to the crew? ...
        * Create a collection *just* for the crew of that series. Load it into that collection (again, not the `crew` collection yet, for example load it into `crew` SERIESNAME; as a tip, life will be easier if you avoid using collection names with punctuation like "-").[1] You can insert an array directly in the console using `db.crew` SERIESNAME `.insert(` DATA `)`. However, there is a limit in the number of characters (4096) that you can have for a command in the console. If your data exceed that size, you may encounter a parse warning. If so, you should create a JSON file on the server and use the following command to load it:
            · `mongoimport --db tvdb --collection crew` SERIESNAME `--file` FILENAME `.json --jsonArray`

          This is the standard way to load larger documents[2]
        * Find and copy the `ObjectId` of your series in the `series` collection. Say this value is `ObjectId("5555")`.
        * Now we need to embed a reference to the series into each document in `crew` SERIESNAME. To do this, you need to use an update to set the following field in each document in `crew` SERIESNAME,[3] where you should replace the ObjectId `"55555"` with that of your series.
            · `"tvseries": new ObjectId("55555"))`
        * Now we have the series info in each crew document, we can copy the documents from `crew` SERIESNAME to `crew` and drop the `crew` SERIESNAME collection. To do this, you can run:
            · `db.crew` SERIESNAME `.aggregate([ { $match: {} }, { $out: "crew" } ])`
            · `db.crew` SERIESNAME `.drop()`
        * Okay, now use `db.series.aggregate(...)` to do a left-outer-join to embed the crew data into your series. As a hint, here's the start to match your series. You need to change the ObjectID and fill in the rest of the `$lookup` stage (you can name the `as` field `"crew"`):
            ```
            · db.series.aggregate( [ { $match: { _id: new ObjectId("55555") } },
                                     { $lookup: {  TO-BE-COMPLETED  } } ] )
            ```
- To exit the `mongodb` console, type `exit`.
- SUBMIT to u-cursos a text file with all commands entered into the `mongodb` console.

[1] If we loaded crew for several series directly into the `crew` collection we would not know which crew worked on which series.

[2] Note that the limit of document size for MongoDB is 16MB. For documents larger than that limit, one could, e.g., unnest some inner documents and store them in a separate collection with a foreign key. We will see something similar in a moment.

[3] Another option would be to use `DBRef` as mentioned briefly in the class. The benefit of a DBRef is that we could add not only the ObjectId, but also the collection (and optionally database) containing that object. However the documentation `https://docs.mongodb.com/manual/reference/database-references/` actually states: *Unless you have a compelling reason to use DBRefs, use manual references instead.* Presumably this is because DBRefs are more complicated. We don't have a compelling reason. A compelling reason (perhaps) might be if we wanted to add foreign keys pointing to different collections and/or databases from the same collection and with the same local key.