Cassandra Loh (5653663108)

DSCI 551: Midterm Progress Report

Recap of Project:

The project aims to design and implement a database system that supports and manages movie data from IMDB obtained from Kaggle with its own query language. It has data such as the movie title, rating, actors, genre, certificate, and directors.

Data Cleaning:

After cleaning the original dataset, I found a lot of missing values and non-English characters especially in the Movie Title and Rating columns. To avoid storing any empty strings or non-English characters which would be impossible to query without copying and pasting the character, I have employed a list-wise decision to remove all these unwanted values and characters. My final IMDB.csv dataset was only 3MB which was smaller than I initially planned for.

- **Challenge #1**
  To overcome this, I have decided to change the method of storage from a relational structure like that of MySQL to JSON, with a structure like the ones we constructed in Homework 1 using Firebase. This brought my JSON file size to about 8MB so my new main memory will be set to 2MB which satisfies the ¼ requirement.
    - Also, since we can't use the pandas library extensively as a part of the requirements, using JSON would be an added advantage as we are able to read and write data without the help of pandas.

Storage System:

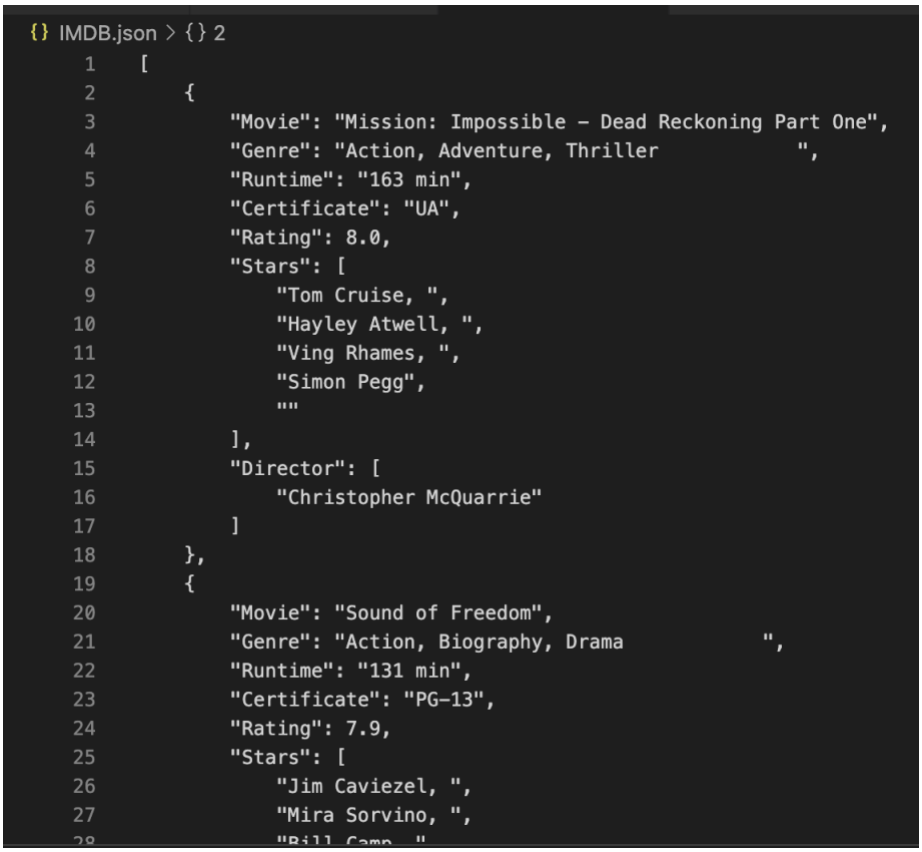
I have decided to store each table under a JSON file locally. For example, my main dataset would be the IMDB dataset with columns such as Actors, Movie, Rating, Certificate. I can also store additional information about the actors such as their age and hometown by creating another .JSON file (e.g., Actors.JSON) and storing all the information. I can then join both tables when needed to get their combined values. Using the load function in the JSON package for Python, the script can read the file line by line instead of loading the whole file into main memory.

Query Language Syntax:

As we are required to have our own query language syntax for our database, I'm running into some problems with the grammar and parsing to make it sound more like natural language.

- **Challenge #2**
  For example, "find movies which stars Tom Cruise", "find movies which genre is Action", "find movies which rating is 8.5", only the first query is grammatically correct. The other 2 sound wrong so I will need to spend more time on parsing or finding a more general sentence structure which fits more of the queries that I will be running.

Screenshots of Project Code & Files:



```
{} IMDB.json > {} 2
[
    {
        "Movie": "Mission: Impossible – Dead Reckoning Part One",
        "Genre": "Action, Adventure, Thriller           ",
        "Runtime": "163 min",
        "Certificate": "UA",
        "Rating": 8.0,
        "Stars": [
            "Tom Cruise, ",
            "Hayley Atwell, ",
            "Ving Rhames, ",
            "Simon Pegg",
            ""
        ],
        "Director": [
            "Christopher McQuarrie"
        ]
    },
    {
        "Movie": "Sound of Freedom",
        "Genre": "Action, Biography, Drama           ",
        "Runtime": "131 min",
        "Certificate": "PG-13",
        "Rating": 7.9,
        "Stars": [
            "Jim Caviezel, ",
            "Mira Sorvino, ",
```

Storage structure of IMDB data in a JSON file.

```
def do_create(self, line):
    try:
        self.table = line.strip()
        if os.path.exists(f'{self.table}.json'):
            print("Table already exists.")
            return

        with open(f'{self.table}.json', 'w') as f:
            json.dump([], f)
        print(f"Table {self.table} created.")
    except Exception as e:
        print(f"Error creating table: {e}")
```

Create table function for CLI.

```
def do_load(self, table):
    try:
        with open(f'{table.strip()}.json', 'r') as f:
            self.data = json.load(f)
            self.table = table.strip()
        print(f"Data loaded from {self.table}.json.")
    except Exception as e:
        print(f"Error loading data: {e}")

def do_find(self, line):
    if self.table is None:
        print("No table is loaded/selected.")
        return

    try:
        # Corrected regular expression pattern
        match = re.search(r"\w+ whose (\w+) is (.+)", line)
        if match:
            field, value = match.groups()
            value = value.strip()

            # Check if the first record's field value is a list
            results = []
            if isinstance(self.data[0][field], list):
                try:
                    for record in self.data:
                        for item in record[field]:
```

Load tables function for CLI.