LAB 2 REPORT

Github Link - https://github.com/Parag000/Data-225-Lab-2

1. Problem Statement:

Application System:

Our team proposes the development of an analytics system using the MongoDB cluster for the TMDB dataset. The proposed system aims to provide users with a scalable platform for conducting analytics on movie data. The limitation of the system is that the analysis cannot be performed on real-time data or streaming data.

Justification for Database & Need For Migration:

With MongoDB's adaptable schema design, scalability, and resilience, it emerges as the ideal choice for migrating the TMDB analysis database to MongoDB Atlas on AWS. MongoDB's document-oriented architecture seamlessly handles the varied metadata of movies and user ratings, facilitating agile data modeling. Leveraging MongoDB on AWS simplifies the migration process, harnessing the cloud infrastructure of AWS and MongoDB expandable schema. Given the semi-structured nature of the dataset, MongoDB's architecture easily adjusts, guaranteeing scalable and resilient performance. By migrating to MongoDB Atlas on AWS, advanced analytics using AWS cloud applications become feasible, enhancing flexibility and scalability.

2. System Requirements:

Outline of requirements for the solution:

The system must ensure scalability to handle increasing data volumes and user loads without performance degradation, guarantee resilience through high availability and fault tolerance mechanisms, support flexible schema design and prioritize efficient query processing for real-time analytics.

Functionalities and Limitations:

- Data Preprocessing: Converting data to denormalized form using python to migrate it to MongoDB
- Data Ingestion Using Python: Support for importing data from CSV files into MongoDB Atlas.
- Querying and Aggregation: Enable users to perform queries and aggregations on TMDB dataset
- Access Control: Define roles and permissions to restrict access to sensitive data and functionalities based on user roles.
- Logging: Log database activities for auditing and compliance purposes.

User Interface:

Users can seamlessly interact with the system using MongoDB Atlas tools to perform analytics, leveraging the intuitive interface provided by MongoDB's cloud-based platform. Through the MongoDB Atlas dashboard and integrated tools, users can effortlessly access and analyze data stored in MongoDB collections. They can execute queries, apply filters, and visualize results using built-in features or third-party visualization tools supported by MongoDB Atlas.

3. Conceptual Database Design:

Database requirements for the application system & Document Structure:

The database will feature a single collection named "Movies" to serve as the central repository for all TMDA data.

To provide the data structure we have extracted a single entry from mongoDB and displayed it in a new sample collection.
The way we performed it is:
MongoDB Compass —>Aggregations Tab —> Add Stage —> $limit (Specified entry as 1) —> Added another Stage —>$out (Result exported to another collection names Document-Structure) —> Run

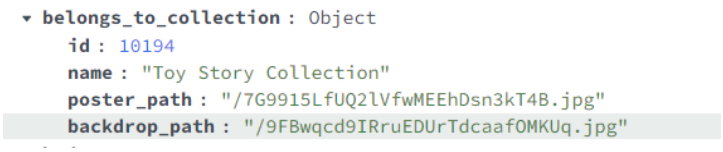
Click here to view images of our document structure & Schema.

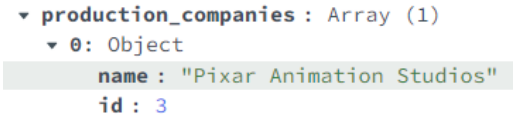
Denormalization Process:

Initially, essential libraries are imported, and CSV files containing data on movie credits, keywords, links, ratings, and movie metadata are read into pandas DataFrames. Following this, various cleaning and preparation steps are undertaken, including converting numeric columns to the appropriate data type, and merging datasets based on shared identifier “movie_id”. Subsequently, JSON string columns are converted into lists of dictionaries to ensure compatibility with MongoDB's document-oriented structure. Additionally, certain data transformation tasks are performed, such as calculating average ratings. The denormalization process involves using a left join operation when merging this denormalized data with other files. Finally, the denormalized data is inserted into a MongoDB collection named 'Movies', facilitating efficient storage, retrieval, and analysis of movie information. This denormalization process streamlines data management and enhances the usability of movie data within a MongoDB environment.
Link to view the denormalization process done using python
https://github.com/Parag000/Data-225-Lab-2/blob/main/Denormalization-and-Migration.ipynb

Primary Key & Embedding Representation:

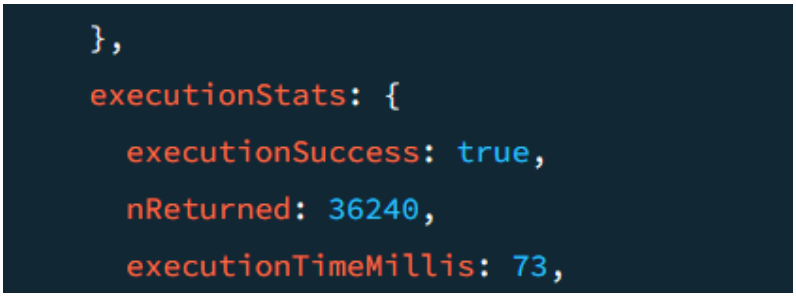
- In MongoDB, each document in a collection is uniquely identified by a "movie_id" field or primary key.
- Rather than using foreign keys to represent relationships, MongoDB uses the below method:
    a. Embedding: directly includes related information in the document. For instance, in a movie database, the "belongs_to_collection" field could embed information about a movie's collection directly within the movie data.





4. Functional Analysis:

- The proposed application system in which we have used python for the Denormalization process where the dataset is merged to a single file format with a list of Dictionaries. This Dataframe is imported directly to the MongoDb Atlas which is acting as a Database for our application. Queries and Aggregations are written in the MongoDb Compass which will give the desired results.
- The database interactions for functional components are provided in the NoSql Queries file which is attached to the GitHub link.
- Github Link - https://github.com/Parag000/Data-225-Lab-2
- Document structure
  Click the below link to view the JSON document format
  Doc-Structure
- Some of the concepts we have addressed are
    - Created 3 users and granted various types of access privileges to each of the users.
    - Generated 11 NoSQL queries that make use of various concepts such as Regex functions, expressions, sort functions and having clauses, and other basic operations.
    - Generated 6 Aggregation functions for the querying operations using most of the fields.

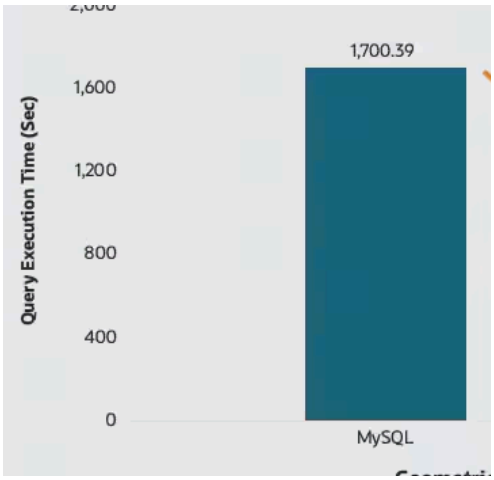
- Logging mechanisms are monitored in our local directory and the screenshot is attached to a NoSQL file.
- NoSQL performance Measurement and comparison with MySQL is given in the below graphs :
  - 

    In the below NoSql query, we have given explain() function to find the execution details for the find() function. This function took around 73 milliseconds for execution.

    Code :

```
db.Metadata.find({"vote_average": { $gte: 5, $lt: 10 }}, {"_id":   0,"id":
1,"original_title": 1}).explain("executionStats")
```

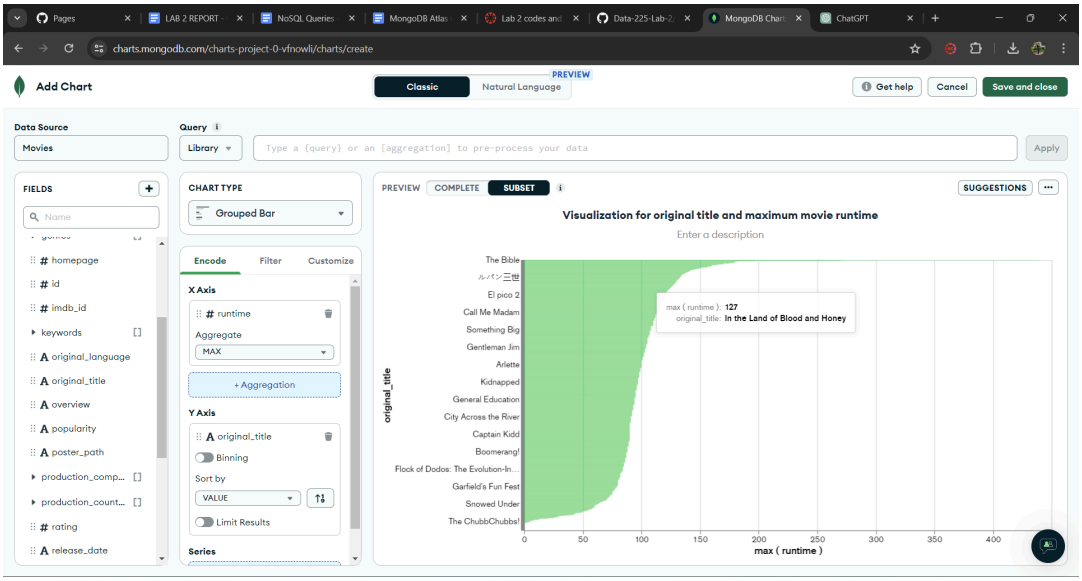
  - But when compared with the SQL SELECT statement, it took around 17 milliseconds for executing the query as shown in the figure.



- Visualization of MongoDB data.

  This graph compares movie titles to their maximum runtime. Hover over each

point to see the specific runtime for each movie. It highlights the diversity in movie titles and their corresponding durations, offering insights into the relationship between title length and runtime.



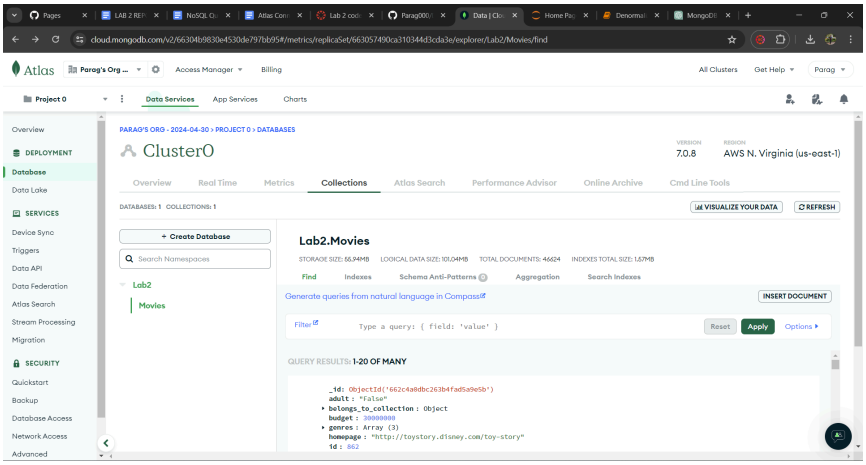
5. Database Migration to Atlas:

Step-by-step process on Connecting to MongoDB Cluster:

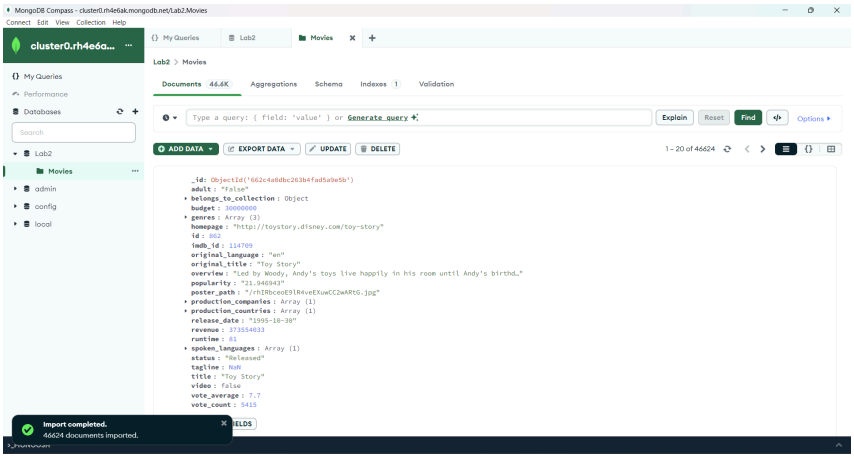
- Logged in to MongoDB Atlas
- Created new cluster and named it as “Cluster0”
- Created Database with name “Lab2” and schema name “Movies”
- Migrated data by connecting to server which is running on atlas using python

Demonstrate the connectivity to Mongo cluster using Python.

Screenshot of cluster after migration on Atlas:



Screenshot of MongoDB Compass which is connected to Atlas:



Include screenshots and code snippets to validate the successful migration.

- Python code for migration and importing TMDB data

Connecting to MongoDB

In [11]:
```
# Connect to MongoDB
connection_string = 'mongodb+srv://itsmeparag14:sarsamba99@cluster0.rh4e6ak.mongodb.net/'
client = MongoClient()
db = client['Lab2']
collection = db['Movies']
```

In [12]:
```
collection.insert_many(denormalized_data)
```

```
# Connect to MongoDB Atlas Cluster
connection_string =
'mongodb+srv://itsmeparag14:sarsamba99@cluster0.rh4e6ak.mongodb.net/'
client = MongoClient()
db = client['Lab2']
collection = db['Movies']
collection.insert_many(denormalized_data)
```