**CSCI 620/Section 03/Mior, Introduction to Big Data, Spring 2215 Assignment 4 – Document Databases**

**Submitted by – Prakhar Gupta(pg9349)**

**Ans1**

To load the data sets from sql to mongodb run the code the following codes in the below order-

* Sql2mongo.sql- This file contains the code to clean the data of extra double quotes which may be present in the names of various entities like moviestitle, character name etc also joins the title table with actors, genres, writers, producers, directors and covert the values into JSON array format to be readily be imported by mongodb
* CD into the location where the JSON files are created by the above program

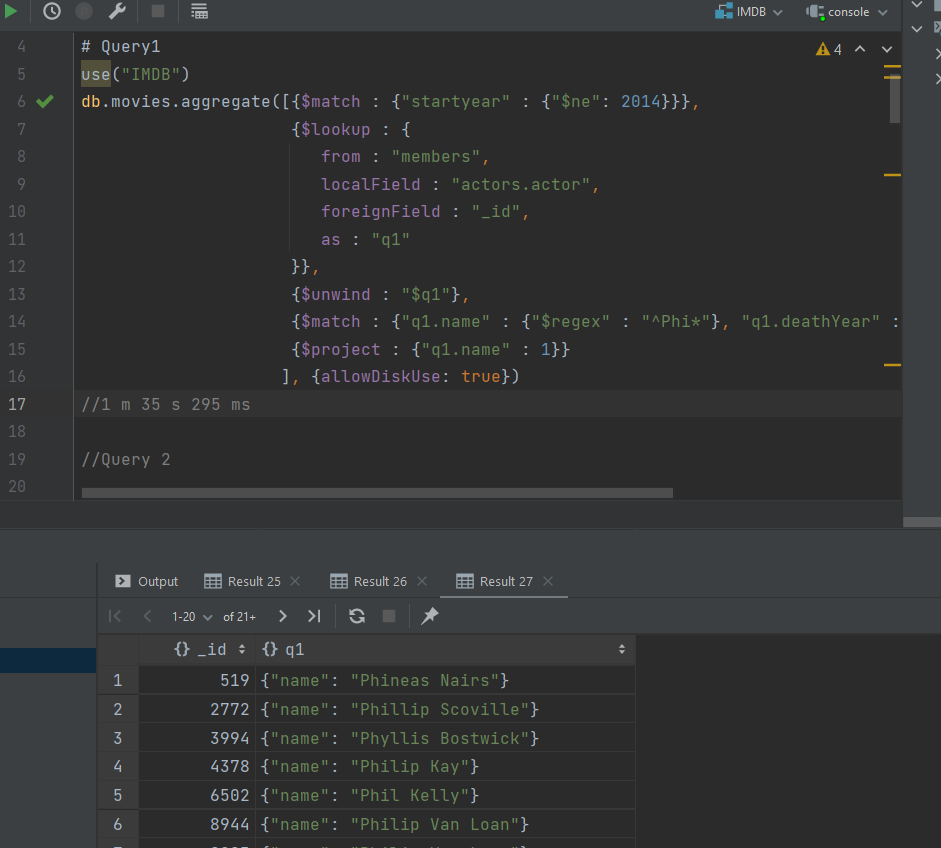
Run-

* + mongoimport --db IMDB --collection members --jsonArray --file "members.json" –drop
  + mongoimport --db IMDB --collection movies --file "movies.json" --drop

**Ans2**

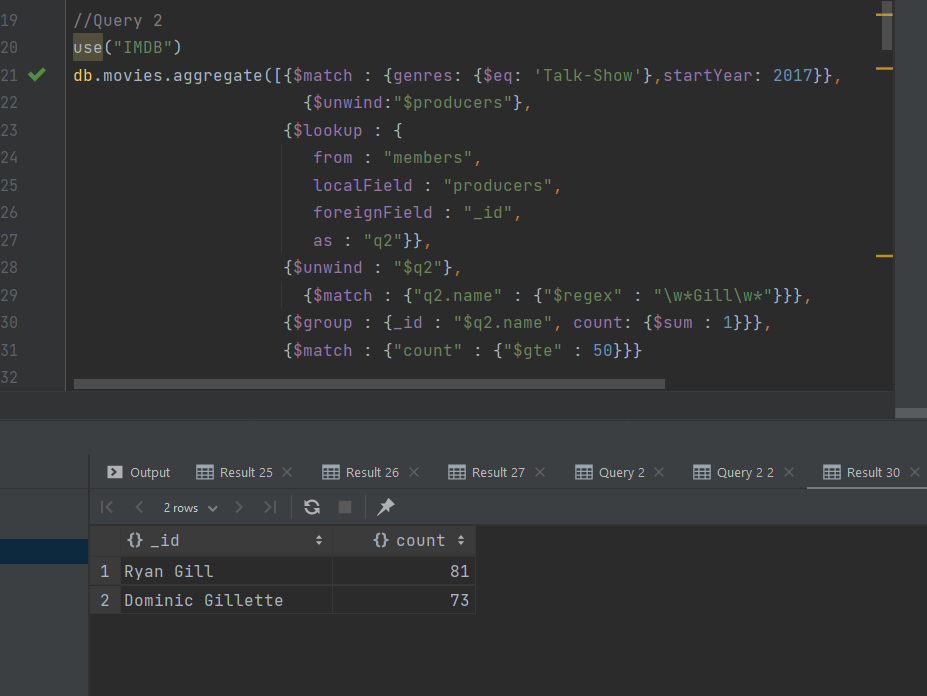
Refer to queries.txt for queries

Query1



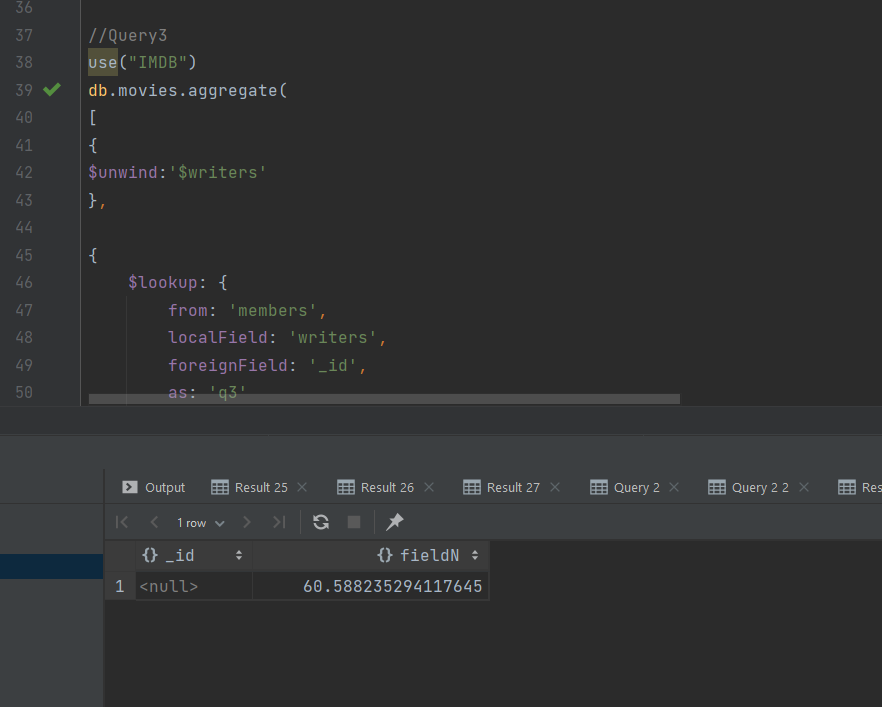
Runtime 1m 35secs

Query2



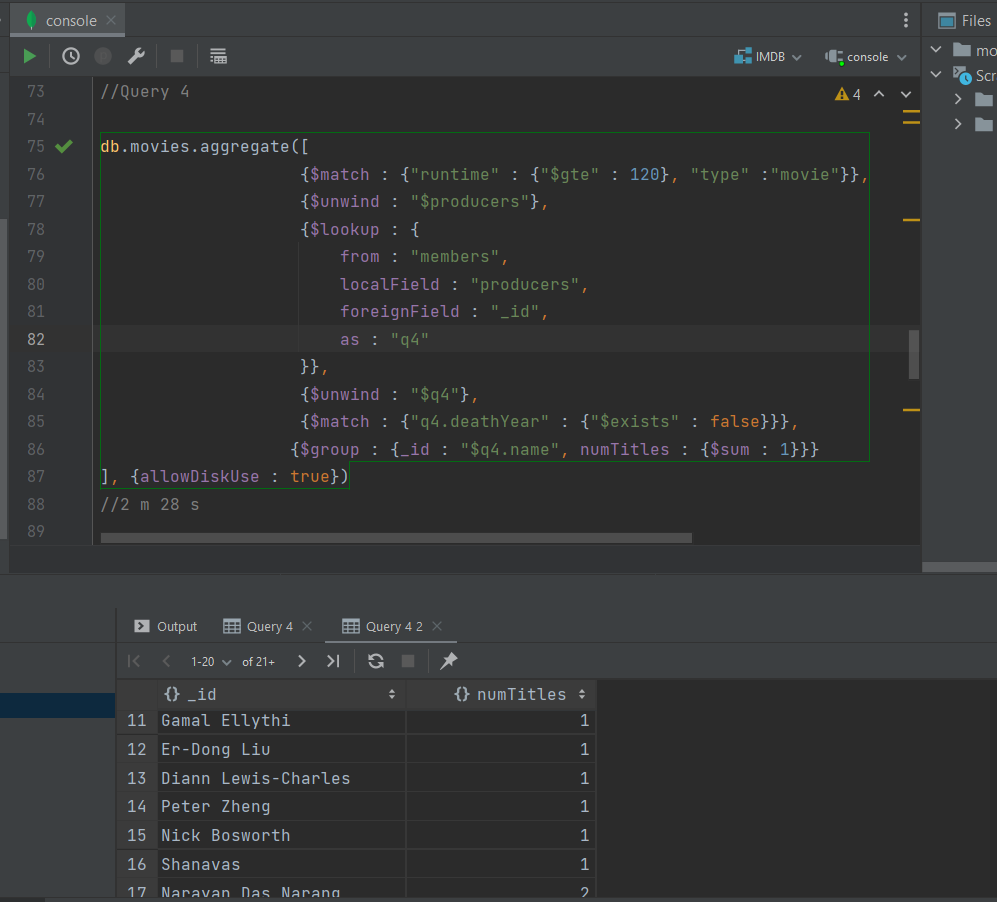
Runtime- 6mins 58 sec

Query3



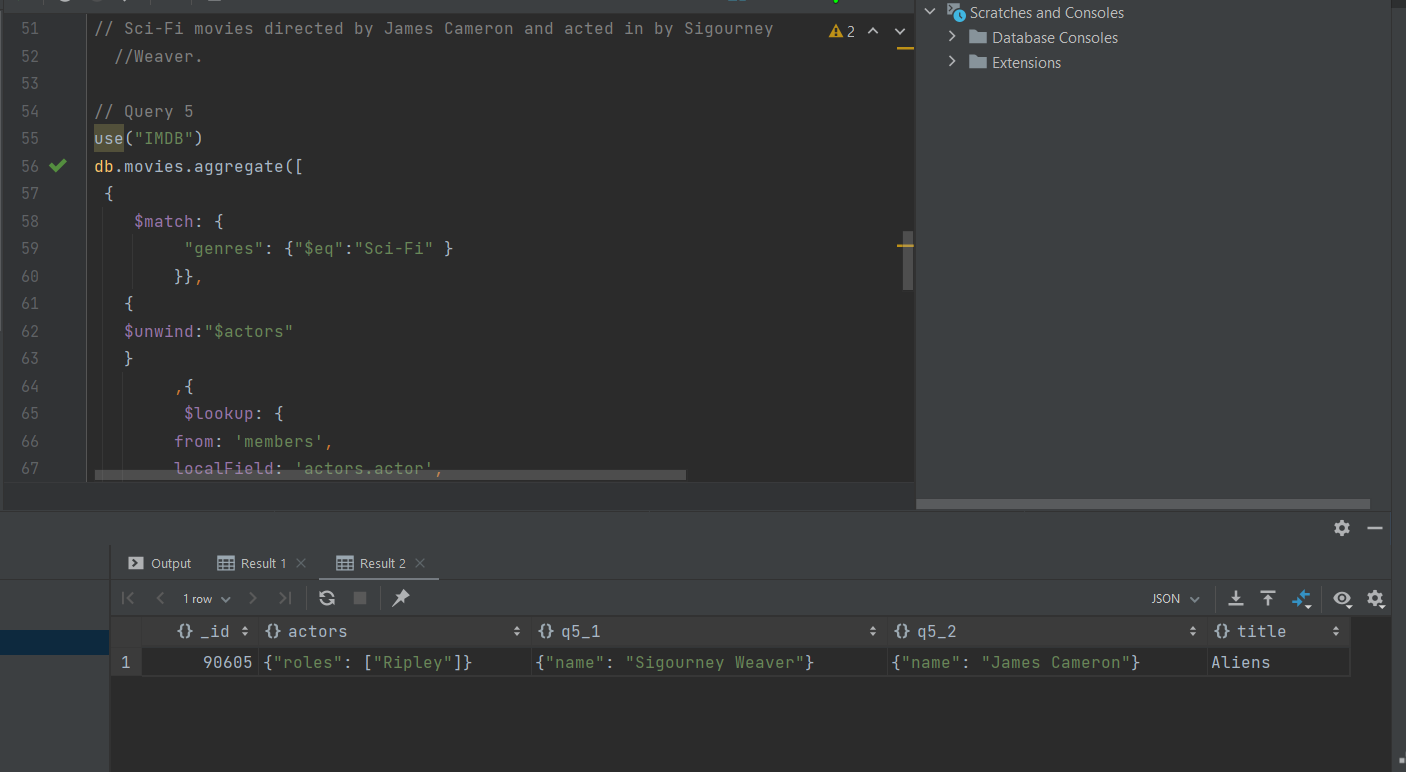
Runtime – 20 mins 28 secs

Query 4



Runtime – 2 min 28 secs

Query 5



Runtime – 1 min 48 secs

**Ans 3**

(Refer to explain.txt for queries)

Also in the submission folders are the JSON responses to various explain queries. Along with details of queries execution we can see the memory usage of the DB engine in the server parameters of the json.

Major point to note in each queries in the stages attributes and the winning plan which shows the plan selected for execution.

* Query1: q1.json – In stage field pipeline executes a simple filter to identify start year not equal to 2014 which is done using default projection and collection scan (Collscan) to filter the records the results are merged using a lookup with members collection which has its own pipeline to filter on name using regex and unwinding the results to remove null records. The final stage includes projecting the results with title id and name as the result of the queries show.
* Query2: q2.json – default projection and collection scan is done to filter on genre and the startyear Followed by an unwind process on producers. Then a lookup operation with the members collection which includes filtering by name and unwinding the result. At last group by stage to count and a match stage to filter.
* Query 3 : q3.json- Winning plan consist of simple projection followed by collection scan.

Then we have the unwind step on writers followed by lookup on members collection which has its own pipeline to match by name using regex and checking whether death year doesn’t exists. Following that we have an unwind step post that we groupby all ids to find the avg runtime.

* Query 4: q4.json- The stages start with collection scan to filter out type movie and runtime greater than 120 followed by unwinding step for producers. Then we do a lookup with members which has internal pipeline to check that deathyear doesn’t exists. Unwinding and grouping by name for count of titles produced
* Query 5 :q5.json- Winning plan includes simple projection followed by collection scan to find genres which have sci-fi unwinding actor to do a lookup with members table twice for actors and directors. Only after the lookup unwinding and matching on name takes place which is again followed by projection stage.

**Ans4**

Refer to index.txt for reference

* Query1: Indexed on movies(startyear,name) members(deathyear)

Idea was that since collection scan is running if we index it would speed up the process.

Runtime after index: 30 secs

* Query2: Indexed on movies(genre)

Idea was since we are searching in genre value if would be helpful if we index it. This had massive impact on runtime reducing it to just 20 secs

Runtime after index: 20 secs

* Query3: Indexed on movies(writers,runtime)

The approach was to index on the runtime which is used to filter also on the lookup local field value writer. This didn’t help much in the speed.

Runtime after index: 17 mins 4 secs

* Query4: Indexed on movies(type,producer)

The approach was to index on the type which is used to filter also on the lookup local field value producer. Runtime did reduce

Runtime after index: 1 mins 26 secs

* Query5: Indexed on movies(actors) and members(name)

Idea was since we are searching across member name and unwinding actors for lookup this would help.

Minor Runtime improvements

Runtime after index: 51 secs