**Team 5:**

Alex Howell

Ulrich Wartes

**T593 – MongoDB Query Assignment**

**Question 1**

Scenario

RancidPineapple (RP) is a company that wants to launch a new web site focusing on movie reviews. They plan to use MongoDB to store and serve data about movies. In particular, they are focusing on a single document containing all the movie information, plus reviews as linked documents. These documents should have the following format:

**Movie data** – \_id, Name of movie, PineappleMetric (a number between 0 and 100), release year, rating, runtime, director [*assume, for sake of simplicity, that each movie has a single director*], a list of writers, production company (embedded, containing the name of the production company and its location), budget, a list of weekly box offices (most recent last)

**Review data** – Review author, date of review, review text, link to movie being reviewed

Assignment

Part 1

Use the sample documents available at the end of this document to create movies and reviews collections.

Part 2

Answer the following queries (that is, the queries you provide below should work on the sample collections in the appendix) and post a screen shot.

**Note 1: Use MongoDB Atlas and MongoDB Compass (or another MongoDB interface) to implement and test your responses in this assignment.**

* 1. Retrieve all documents in the movies collection.

Answer:

db.movies.find()

Graphical user interface, text, application, email

Description automatically generated

* 1. Retrieve all movies released in 2018.

Answer:

db.movies.find({releaseyear:'2018'})

Graphical user interface, text, application, email

Description automatically generated

* 1. Retrieve all movies with a Pineapple Metric lower than 50.

Answer:

db.movies.find({pineappleMetric: { $lt: 50 } } )

Graphical user interface, text, application

Description automatically generated

* 1. Retrieve all movies whose production company is located in the “UK”.

Answer:

db.movies.find({"productionCompany.prodCompanyLocation":"UK"})

Graphical user interface, text, application, email

Description automatically generated

* 1. Retrieve the name and opening week box office (ONLY these “columns”) from all movies.

Answer:

db.movies.find({},{name:1,boxOffice:{$slice:1}, \_id:0})

Graphical user interface, text, application, email

Description automatically generated

* 1. Retrieve all movies produced by “Rinse Wash Repeat”

Answer:

db.movies.find({"productionCompany.prodCompanyName":"Rinse Wash Repeat"})

Graphical user interface, text, application

Description automatically generated

* 1. Retrieve all reviews written in 2018 (notice the format of that field as you develop your answer).

Answer:

db.reviews.find({reviewDate:{$regex:/2018/}})

Graphical user interface, text, application, email

Description automatically generated

Part 3

1. Answer the following question: With a single document for each movie that embeds information such as production company and a list of writers, it seems that there will be redundant data since, for example, a production company can produce multiple movies. What would be a justification for embedding this data inside a movie document?

Perhaps embedding the production company fits more naturally in your application. For instance, if you will only every view a production company as an attribute of a movie (rather than the other way around), this would be a more natural way of modelling this relationship. And any errors that could result from the duplication wouldn’t really impact the success of the application.

1. After procuring and creating all the data for movies and starting to have reviews added by users, RancidPineapple noticed that it was missing data about actors in its database. It procured data from the (fictitious) International Union for Actors and Part-Time Waiters containing a list of current and future actors. Indicate three ways in which this data could be incorporated into the current data structure that RancidPineapple created and list the advantages (if any) and the drawbacks (if any) for each of these approaches.
2. Actor data could be embedded into movie documents as an array (all at once). The benefits of this would be that it would parallel how director, writer, and production company are currently modeled (which presumably works well for the RancidPinnapple application). Drawbacks would be that the Actor data would probably be more difficult to maintain overtime. For instance, if an actor changes their name (for whatever reason), this update would now have to be made in a bunch of places. Also, you wouldn’t be able to use the future actors, as they wouldn’t have appeared in any movies yet….though, you also wouldn’t have to maintain those records.
3. Actor data could be loaded as a new collection. This would be easier to maintain, but it would require more computing power to “join” data in transactions (i.e., expensive joins) and more coding. This could get expensive depending on the sizes of the datasets.
4. Actor data could be migrated on demand (kinda splitting the difference between the two). If you were to load all the data into the movie documents at once, this might be an incredibly huge job. But if you were to update the data as the movie records are being accessed in real-time, then you could spread out that work over time. This would still have some of the drawbacks of embedding the data mentioned in 1, and it may also cause problems if you wanted to summarize data about actors across movies.

**Question 2**

You will now practice writing some aggregate queries with MongoDB

Visit the following site:

<https://docs.mongodb.com/manual/tutorial/aggregation-zip-code-data-set/>

Part 1

You will simply recreate the following queries from the website. Provide a screenshot as evidence that you were able to run the queries,

* [Return States with Populations above 10 Million](https://docs.mongodb.com/manual/tutorial/aggregation-zip-code-data-set/#return-states-with-populations-above-10-million)

Graphical user interface, application

Description automatically generated

* [Return Average City Population by State](https://docs.mongodb.com/manual/tutorial/aggregation-zip-code-data-set/#return-average-city-population-by-state)

Graphical user interface, application

Description automatically generated

* [Return Largest and Smallest Cities by State](https://docs.mongodb.com/manual/tutorial/aggregation-zip-code-data-set/#return-largest-and-smallest-cities-by-state)

Graphical user interface

Description automatically generated

Part 2

Provide queries and a screenshot for the following questions.

1. Provide a list of cities in each state that have more than 4 zip codes?

QUERY:

db.zips.aggregate[

{$group: {

\_id: {state: "$state",city: "$city"},

ZipCount: {

$sum: 1

}

}}, {$match: {

ZipCount: {$gt: 4}

}}, {$sort: {

ZipCount: -1

}}]

Screen Shot:

Graphical user interface, application

Description automatically generated

1. Provide a list of cities and their states whose population is greater than 500,000

QUERY:

db.zips.aggregate[

{$group: {

\_id: {state: "$state",city: "$city"},

Total\_pop: {

$sum: "$pop"

}

}}, {$match: {

Total\_pop: {$gt: 500000}

}}, {$sort: {

Total\_pop: -1

}}]

Screenshot:

Graphical user interface, application

Description automatically generated

**Question 3**

You will explore movies data from the MovieLens database (<http://grouplens.org/datasets/movielens/>). Use the small dataset (same as what you used in the Hive assignment) from the category MovieLens Latest Datasets section. You will only need to create a collection for the movies and ratings data to answer the below questions.

**Note:** Make sure you import the ratings value as a number and not as a String.

**Note 2:** The import process will lead to some errors due to the presence of “,” in some titles. You will find some genres values are in error. You can ignore this.

Please answer the following questions and provide a screenshot:

1. Generate a list of movies that have “Adventure” as a genre

QUERY:

db.movies.aggregate[

{$project: {

\_id:0,

title:1,

genres:{$split: ["$genres","|"]}

}}, {$match: {

genres: {$elemMatch:{$eq: "Adventure"}}

}}, {$project: {

title:1

}}]

SCREENSHOT:

Graphical user interface, application

Description automatically generated

1. Generate the movies that have been categorized into more than 2 genres

QUERY:

db.movies.aggregate[

{$project: {

\_id:0,

title:1,

genres:{$split: ["$genres","|"]}

}}, {$project: {

title:1,

genrecount:{ $size: "$genres"}

}}, {$match: {

genrecount:{$gt:2}

}}, {$sort: {

genrecount: -1

}}, {$project: {

title:1

}

}]

SCREENSHOT:

Graphical user interface

Description automatically generated

1. Generate a list of movies that have the “Children” genre and a user rating > 3

QUERY:

db.movies.aggregate[

{$project: {\_id:0,

movieId: 1,

title: 1,

genres: {$split: ["$genres","|"]}

}}, {$match: {

genres:"Children"

}}, {$lookup: {

from: 'ratings',

localField: 'movieId',

foreignField: 'movieId',

as: 'ratings'

}}, {$unwind: {

path: "$ratings"

}}, {$group: {

\_id: '$title',

averageRating: {

$avg: "$ratings.rating"

}

}}, {$match: {

averageRating:{$gt:3}

}}]

SCREENSHOT:

Graphical user interface, application

Description automatically generated

1. Generate all users whose average rating across the movies they have rated is greater than 3

QUERY:

db.ratings.find[

{

$group: {

\_id: "$userId",

averageRating: {

$avg: "$rating"

}

}

}, {

$match: {

averageRating: {

$gt: 3

}

}

}, {

$sort: {

averageRating: -1

}

}]

SCREENSHOT:

Graphical user interface, application

Description automatically generated

1. Which Star Wars movie has the highest average user rating and what is the average rating?

QUERY:

db.movies.aggregate[

{$match: {

title:{$regex:/^Star Wars/}

}}, {$lookup: {

from: 'ratings',

localField: 'movieId',

foreignField: 'movieId',

as: 'ratings'

}}, {$unwind: {

path: "$ratings"

}}, {$group: {

\_id: '$title',

averageRating: {

$avg: "$ratings.rating"

}

}}, {$sort: {

averageRating: -1

}}, {$limit: 1}]

SCREENSHOT:

Graphical user interface, application

Description automatically generated

1. What is the average rating of all movies with a date of 2008 in the title.

QUERY:

db.movies.aggregate[

{$match: {

title:{$regex:/(2008)/}

}}, {$lookup: {

from: 'ratings',

localField: 'movieId',

foreignField: 'movieId',

as: 'ratings'

}}, {$unwind: {

path: "$ratings"

}}, {$addFields: {

2008: "2008"

}}, {$group: {

\_id: '$2008',

averageRating2008: {

$avg: "$ratings.rating"

}

}}, {$project: {

averageRating2008:1,

\_id:0

}}]

SCREENSHOT:

Graphical user interface, application

Description automatically generated

**APPENDIX - COLLECTIONS**

\*\*\* MOVIE DOCUMENTS - “MOVIE” COLLECTION \*\*\*

{

"\_id": self generate,

"movieId": "1",

"name": "Mission Impossible 14 - The AARP Threat",

"pineappleMetric": "88",

"releaseYear": "2018",

"rating": "PG-13",

"runtime": 136,

"director": "Robert Silva",

"writer": [

"Mary Silva",

"Mario Silva"

],

"productionCompany": {

"prodCompanyName": "Rinse Wash Repeat",

"prodCompanyLocation": "USA"

},

"budget": 127000000,

"boxOffice": [

32000000,

15000000

]

}

{

"\_id": self generate,

"movieId": "2",

"name": "Die Hard - Resurrection",

"pineappleMetric": "58",

"releaseYear": "2019",

"rating": "R",

"runtime": 117,

"director": "Joan Smith",

"writer": [

"Terry Smith",

"Francine Smith"

],

"productionCompany": {

"prodCompanyName": "Yippee-Ki-Yay",

"prodCompanyLocation": "Australia"

},

"budget": 63000000,

"boxOffice": [

19000000,

7000000,

5300000

]

}

{

"\_id": self generate,

"movieId": "3",

"name": "Fifty Shades Greener",

"pineappleMetric": "35",

"releaseYear": "2018",

"rating": "PG",

"runtime": 98,

"director": "Cookie Muster",

"writer": [

"Elmo Jones",

"Sue Bird"

],

"productionCompany": {

"prodCompanyName": "Sesame Muppets",

"prodCompanyLocation": "UK"

},

"budget": 17000000,

"boxOffice": [

58000000,

43000000,

27000000,

13000000

]

}

\*\*\* REVIEW DOCUMENTS - “REVIEW” COLLECTION \*\*\*

{

"\_id": self generate,

"reviewId": "1",

"reviewAuthor": "Timmy Twopence",

"reviewDate": "2019-07-23",

"reviewText": "It is an unexpected pleasure to have John McClane back as a zombie pursuing the nefarious masterminds behind a plan to dominate the world by transforming most of the population into undead slaves",

"movieID": "2"

}

{

"\_id": self generate,

"reviewId": "2",

"reviewAuthor": "Sylvia Sterling",

"reviewDate": "2018-06-12",

"reviewText": "In this exciting movie, Ethan Hunt battles his worst enemy ever: a nefarious organization that advocates the retirement of actors when they are too old to play leads in action movies",

"movieID": "1"

}

{

"\_id": self generate,

"reviewId": "3",

"reviewAuthor": "Francis Pounder",

"reviewDate": "2018-12-04",

"reviewText": "Kermit the Frog against a nefarious industrialist trying to pollute the world is a fun premise but the use of powerpoint slides during the movie is very distracting",

"movieID": "3"

}