CSCU9YQ: MongoDB Queries on a Movies Database

Question 1.1

Compute the average duration of movies by genre and list the top 5 genres with the longest average duration. List the name of the genre and the average duration time sorted by time in decreasing order.

Query:

```
db.movies.aggregate([
      {$unwind: "$genres"},
      {$group: {_id: "$genres", avgDur:{$avg:"$runtime"}}},
      {$sort:{avgDur:-1}},{$limit:5}
])
```

Explanation:

Within the aggregation pipeline, we start by using the unwind function so we can split the genres array. If there is more than one genre, this will create a new record for each element in the genres array, where there was previously one array listing many genres. This allows us to then group by genre which means we can calculate an average run time called avgDur. Using the unwind operation makes it a lot easier to deal with different genres as the records are split, one for each genre. We then sorted the list of run times in descending order using the sort operation, where -1 represents descending order, and limited the number of documents displayed to five using the limit feature.

Results:

```
{ "_id" : "Musical", "avgDur" : 113.375 }
{ "_id" : "Romance", "avgDur" : 107.18571428571428 }
{ "_id" : "Crime", "avgDur" : 106.56571428571428 }
{ "_id" : "Action", "avgDur" : 105.208 }
{ "_id" : "Western", "avgDur" : 105.12121212121212 }
```

Question 1.2

List all the countries that have produced 10 or more movies with the UK. List the country names and number of movies produced in collaboration with the UK, in no particular order.

Query:

```
db.movies.aggregate([
      {$match:{countries:"UK"}},
      {$unwind: "$countries"},
      {$group: { _id: "$countries", numberOfMovies: { $sum: 1 } }},
      {$match: {_id: {$ne:"UK"}, numberOfMovies: {$gte:10}}},
      {$project: { _id: 0, numberOfMovies: 1, country: "$_id"}}
]).pretty()
```

Explanation:

At the first stage of the aggregation pipeline we are looking for all movies where “UK” appears in the list of countries. We then ‘unwind’ the countries field, creating an individual record for each country in the countries array, allowing us to count the number of times each country appears. The next stage of the pipeline ‘groups’ the new list of countries and counts the number of times each individual country appears in the data using the sum operator. With this count we can ‘match’ and ‘project’ each country where the count is more than 10 (not including the number of times ‘UK’ has appeared).

Results:

```
{ "numberOfMovies" : 62, "country" : "USA" }
{ "numberOfMovies" : 14, "country" : "Germany" }
{ "numberOfMovies" : 11, "country" : "Canada" }
{ "numberOfMovies" : 16, "country" : "France" }
```

Question 2.1

Report the total number of movies you can find in the collection which are related to any type of sport(s). Report only the number of movies, not their details.

Query: (Create the indexes first)

```
1. db.movies.createIndex({plot:"text", title:"text"})
2. db.movies.createIndex({genres:1})
3. db.movies.aggregate([
{$match:
{$or: [
     {genres: {$in: ["Sport"]}},
     {$text:{$search:"Acrobatics Aerobics Gymnastics Archery Badminton
          Baseball Basketball Bicycle Motocross BMX Billiards Bobsleigh
          Bodybuilding Bowling Boxing Canoeing racing Cheerleading Chess
          Cricket Croquet Curling Dance Sport Darts Diving Dodgeball Fencing
          Figure Skating Football Soccer Frisbee Golf Handball Hockey Skating
          Ski Judo Karate Kayaking Kendo KickBox Boxing Kite Surfing Lacrosse
          Luge Martial Motocross Paintball Parachuting Paragliding Parkour
          Polo Powerlifting Rafting Gymnastics Climbing Rowing Rugby Sailing
          Sandboarding Diving Shooting Skateboarding Skiing Snowboarding
          Softball Speed Skating Climbing Squash Sumo Wrestling Surfing
          Swimming Taekwondo Tennis Trampolining Triathlon Volleyball
          Windsurfing Wrestling"}} ]}},
{$count: "moviesRelatedToSport"}
]).pretty()
```

Explanation:

For this question we initially thought of simply searching the database to find all movies where ‘Sport’ appeared in the ‘Genres’ list, however it soon became apparent that this did not include all films that were related to sport. For example there were films that mentioned sports in their plot descriptions but didn't include it in the genres list. To make this search query more accurate we thought it best to search the title and plot of the movies as well as the genre that may be associated with it, cross referencing the contents of these fields to a list of key-words related to various sports.

To do this we had to create two indexes. Firstly, a text index on the title and plot of the movie allowing us to cross reference the contents and secondly an index on the genres array allowing to search for the key-word ‘sport’. Once these indexes were created we were able to run an aggregation, matching on either the ‘genre’ array containing ‘Sport’ or matching one of our many sports related key-words to the contents of our text indexes on the title or plot. The next part of the pipeline simply counts the number of times we successfully match the genre or key-word and prints the result to the terminal.

Results:

```
{ "moviesRelatedToSport" : 123 }
```

Question 2.2

From the sport-related movies you found in part 1, list the top 3 that have received the highest IMDb 'rating'. List the movie title, the year, and both the IMDb 'rating' and the votes', sorted by the 'rating'.

Query:

```
db.movies.aggregate([
{$match:
{$or: [
      {genres: {$in: ["Sport"]}},
                {$text:{$search:"Acrobatics Aerobics Gymnastics Archery
                Badminton Baseball Basketball Bicycle Motocross BMX Billiards
                Bobsleigh Bodybuilding Bowling Boxing Canoeing racing
                Cheerleading Chess Cricket Croquet Curling Dance Sport Darts
                Diving Dodgeball Fencing Figure Skating Football Soccer
                Frisbee Golf Handball Hockey Skating Ski Judo Karate Kayaking
                Kendo KickBox Boxing Kite Surfing Lacrosse Luge martial
                Motocross Paintball Parachuting Paragliding Parkour Polo
                Powerlifting Rafting Gymnastics Climbing Rowing Rugby Sailing
                Sandboarding Diving Shooting Skateboarding Skiing Snowboarding
                Softball Speed Skating Sport Climbing Squash Sumo Wrestling
                Surfing Swimming Taekwondo Tennis Trampolining Triathlon
                Volleyball Windsurfing Wrestling"}} ]}},
{$sort:{"imdb.rating":-1}}, {$limit:3},
{$project: { _id:0, title:1, year:1, "imdb.rating":1, "imdb.votes":1}}
]).pretty()
```

Explanation:

This works very similarly to question 2.1. However instead of counting the movies that have the genre sport or contain a sports word, we are projecting details about the movies. So we initially look for a movie in the sports genre or one that contains a sports word in it. We then sort these results by imdb rating in descending order (where -1 represents descending order). We then use the limit operation to limit the output to 3 documents. Using the project statement we are choosing what information we want to output for the sports movies. We have displayed the title, the year, the imdb rating and the imdb votes. We used '_id:0' so that the id of the result is not displayed, to make the results neater.

Results:

```
{
      "title" : "Liverpool FC: Champions of Europe 2005",
      "year" : 2005,
      "imdb" : {
                  "rating" : 9.5,
                  "votes" : 410
      }
}
{
      "title" : "Somay Ku: A Uganda Tennis Story",
      "year" : 2008,
      "imdb" : {
                  "rating" : 8.6,
                  "votes" : 19
      }
```

```
}
{
      "title" : "Fotbolls-VM krönikan 1994",
      "year" : 1994,
      "imdb" : {
                  "rating" : 8.6,
                  "votes" : 119
      }
}
```

Question 3.1

Create your own field named "myRating" and compute a movie score according to your design. The idea is to combine and aggregate the information about the quality of movies existing in the collection in a single number which will be your rating.

Queries:

1. `db.movies.update({}, {$set: { "myRating": null}}, false, true)`

2. `db.movies.updateMany({'tomato.meter':{$exists:true}}, [{$set:{'tomato.average':{$divide:[{$add:['$tomato.meter',{$multiply:['$tomato.rating',10]},{$divide:['$tomato.fresh',4]},'$tomato.userMeter',{$multiply:['$tomato.userRating',20]}]},5]}}}])`

3. `db.movies.updateMany({$and:[{'tomato.average':{$exists:true}}, {imdb:{$exists:true}}, {metacritic:{$exists:true}}]}, [{$set:{'myRating':{$round:[{$divide:[{$add:['$tomato.average',{$multiply:['$imdb.rating',10]},'$metacritic']},3]},0]}}}])`

4. `db.movies.updateMany({$and:[{'tomato.average':{$exists:false}}, {imdb:{$exists:true}}, {metacritic:{$exists:true}}]}, [{$set:{'myRating':{$round:[{$divide:[{$add:['$tomato.average',{$multiply:['$imdb.rating',10]},'$metacritic']},2]},0]}}}])`

5. `db.movies.updateMany({$and:[{'tomato.average':{$exists:true}}, {imdb:{$exists:false}}, {metacritic:{$exists:true}}]}, [{$set:{'myRating':{$round:[{$divide:[{$add:['$tomato.average',{$multiply:['$imdb.rating',10]},'$metacritic']},2]},0]}}}])`

6. `db.movies.updateMany({$and:[{'tomato.average':{$exists:true}}, {imdb:{$exists:true}}, {metacritic:{$exists:false}}]}, [{$set:{'myRating':{$round:[{$divide:[{$add:['$tomato.average',{$multiply:['$imdb.rating',10]},'$metacritic']},2]},0]}}}])`

7. `db.movies.updateMany({$and:[{'tomato.average':{$exists:true}}, {imdb:{$exists:false}}, {metacritic:{$exists:false}},{"tomato.userReviews":{$gte: 10000}}]}, [{$set:{'myRating':{$round:['$tomato.average',0]}}}])`

8. `db.movies.updateMany({$and:[{'tomato.average':{$exists:false}}, {imdb:{$exists:true}}, {metacritic:{$exists:false}},{"imdb.votes":{$gte: 10000}}]}, [{$set:{'myRating':{$round:[{$multiply:['$imdb.rating',10]},0]}}}])`

9. `db.movies.updateMany({myRating:null}, [{$set:{'myRating':0}}])`

Explanation:

Query 1 : Creates the myRating field

Query 2: Creates the ‘tomato.average’ field and stores the average of tomato ratings.

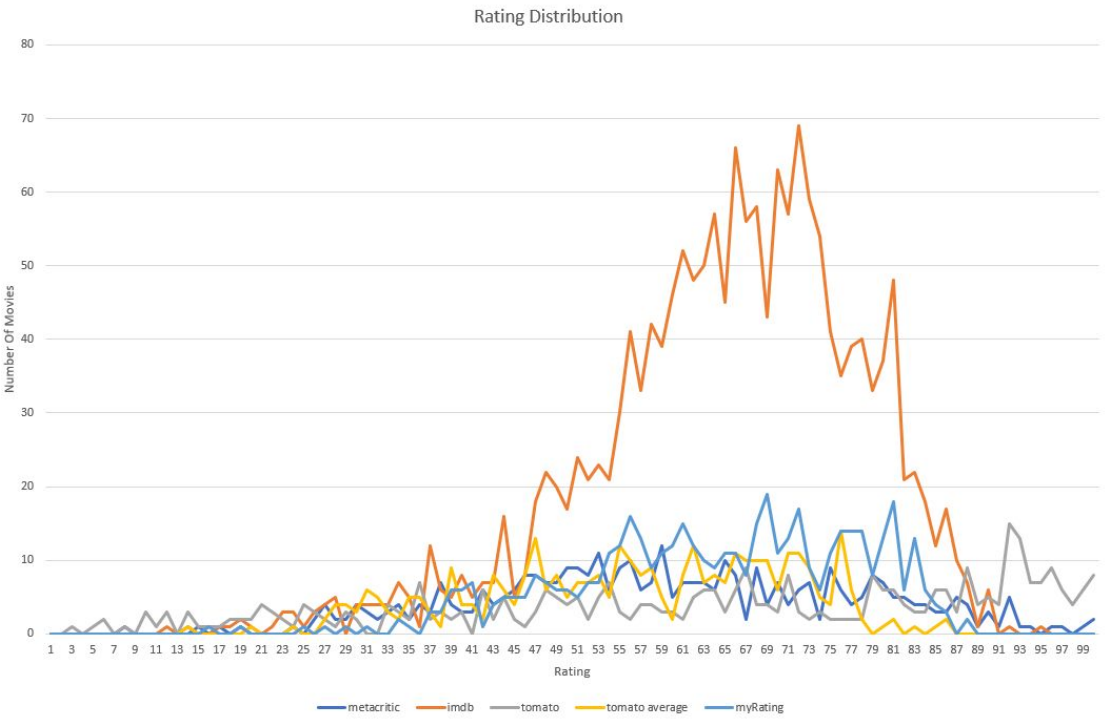
Queries 3 - 9: calculates and updates ‘myRating’

The 'myRating' field is calculated using the ratings provided by imdb, rotten tomato and metacritic. To prepare the collection before assigning a rating, we created a new field within the 'tomato' embedded document called 'tomato.average' using the update method. This field stores an average of 'tomato.meter', 'tomato.rating', 'tomato.fresh', 'tomato.userMeter' and 'tomato.userRating'. These fields were scaled to be out of 100, multiplying the fields that rate 1-10 like 'tomato.rating' and 'tomato.userRating' by 10, and dividing the 'tomato.fresh' field by 4. As the largest value for this field was 371, we took 4 as a reasonable number to ensure it fit the average. After scaling, these values are summed and divided by 5 to give the 'tomato.average'. We decided it was important to capture both the opinions of the critics and audience, as well as the 'fresh' field which represents a consistent high rating and popularity.

At this point we were able to calculate a rating for movies that fit our criteria. The ‘myRating’ field value is a 1-100 rating. Using several queries, we were able to undertake multiple updateMany operations to set a rating for each movie. Ideally queries 3 to 9 would be combined into a single bulkWrite() update but sadly mongodb does not allow such an operation to take place, forcing us to run the queries separately.

The rating is calculated by taking an average of ‘imdb.rating’, ‘tomato.average’ and ‘metacritic’ ratings. To avoid bias from reviews by just one website, movies must have been rated by at least 2 websites OR rated by one website with at least 10,000 reviews in order to receive our rating. For example, a movie with an imdb rating of 7 and a metacritic rating of 90 would receive a ‘myRating’ of 80. If a movie has only been reviewed by a single website, but it has over 10,000 reviews, we figured that this is a reliable rating due to 'wisdom of the crowd'. Movies that fail to meet this criteria fail to receive a rating / get a 0 as they don’t provide enough data in order for us to make an informed decision.

Using Excel, we plotted the number of movies at each rating for all websites and compared our rating to see if it fairly captured movies in the correct category. As can be seen in the graph below, myRating appears to represent the data accurately, matching the distribution of the other ratings.



Query to see distribution of myRating:

```
db.movies.aggregate([{$match:{title:{$exists:true}}},{$group:{_id:"$myRating"
, noOfMoviesWithThisRating:{$sum:1}}},{$sort:{"_id":-1}}])
```

Query to see results:

```
db.movies.aggregate([{$match:{myRating:{$exists:true}}},{$sort:{myRating:-1}}
,{$project:{_id:0, title:1, myRating:1}}])
```

Results:

```
{ "title" : "The Wizard of Oz", "myRating" : 86 }

{ "title" : "Star Wars: Episode IV - A New Hope", "myRating" : 85 }

{ "title" : "Like Stars on Earth", "myRating" : 85 }
```

Question 3.2

Using your newly created rating to rank movies, your task is to recommend 3 good movies according to the type of movies you enjoy watching. Your preferences could be based on any movie characteristics such as genres, actors, topics, directors, years etc.

Query:

```
db.movies.aggregate([
{$match:
      {$and:[
            {$or:[
                  {$and: [
                        {year:{$gte: 1980, $lt: 2000}},
                        {$or:[
                              {director:"Steven Spielberg"},
                              {director:"Francis Ford Coppola"}
                        ]}
                  ]},
                  {actors:"Christopher Lloyd"}
            ]},
            {myRating: {$gte: 60}},
            {rated:"PG"},
            {'awards.wins':{$gt:5}}
            ]
      }
},
{$project:{title:1, year:1, director:1, actors:1, "awards.wins":1,
myRating:1,_id:0}}
]).pretty()
```

Explanation:

In this aggregate we are using a complex ordering of the $and and $or operators to look for movies that match the following criteria:

(((Have a release date from the year 1980 to 1999) AND (were directed by Steven Spielberg OR Francis Ford Coppola))

OR (have Christopher Lloyd listed as an actor))

All results must (have a myRating of 60 or above (determined using greater than or equal to), AND be rated ‘PG’ AND have more than 5 award wins).

We used the project statement to specify that we wanted to display the title, year, director, actors and the rating that we created for the film. We chose to hide the id as it is not relevant. Displaying this information allowed us to see that the results matched what we were looking for.

Results:


```
{
        "title" : "Raiders of the Lost Ark",
        "year" : 1981,
        "director" : "Steven Spielberg",
        "actors" : [
                "Harrison Ford",
                "Karen Allen",
                "Paul Freeman",
                "Ronald Lacey"
        ],
        "awards" : {
                "wins" : 28
        },
        "myRating" : 84
}
{
        "title" : "Back to the Future Part II",
        "year" : 1989,
        "director" : "Robert Zemeckis",
        "actors" : [
                "Michael J. Fox",
                "Christopher Lloyd",
                "Lea Thompson",
                "Thomas F. Wilson"
        ],
        "awards" : {
                "wins" : 9
        },
        "myRating" : 64
}
{
        "title" : "E.T. the Extra-Terrestrial",
        "year" : 1982,
        "director" : "Steven Spielberg",
        "actors" : [
                "Dee Wallace",
                "Henry Thomas",
                "Peter Coyote",
                "Robert MacNaughton"
        ],
        "awards" : {
                "wins" : 46
        },
        "myRating" : 82
}
```