CSCU9YQ

MongoDB Queries on a Movies Database

2636515

March 2020

General Queries

1) **Description**

To compute the average duration of movies by genre the $unwind stage is utilised first in the aggregation pipeline. $unwind deconstructs genres, an array field, from the input document to output a document for each element. This is helpful because movies can be grouped into more than one genre and the movie duration impacts all genres it is classified under. Grouping the documents by the 'genres' field, the $avg accumulator is then used to compute the average runtime for each genre grouping. The new field 'avgDuration' equals the average runtime for each genre. $sort takes the grouped documents by genres and sorts by their average duration in descending order. $limit is the last stage in the pipeline used to list only the top 5 genres with the longest average duration.

Query

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

Result

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

2) **Description**

$match is the first stage in the aggregation pipeline. $match filters the documents to pass only those movies that have "UK" listed in their countries field to the next pipeline stage. The $filter pipeline stage selects a subset of the countries array to return based on the conditions specified. It is used to remove the UK from any solution as it wouldn't make sense to say the UK produced movies with itself. It does this by filtering the array as an 'item' which is a variable name that represents each element of the countries array. A condition is then specified to determine if an element should be in the output array. In this case, the condition selects the array elements where the value is not equal to the 'UK'. The $unwind aggregation is the next stage of the pipeline. It is used to deconstruct the countries field and is after the $filter stage as input to $filter must be an array and not a string. The documents are grouped by the countries field with the $sum accumulator used to compute the count for each group of countries. This number represents the number of movies each country has produced with the UK. The last stage uses $match to list only those countries that have produced 10 or more movies in collaboration with the UK.

Query

```
db.movies.aggregate([
    {$match:{countries:"UK"}},
    {$project: {countries: {$filter: {input:"$countries",as: "item",
                cond: {$ne: ["$$item", "UK"]}}}}},
    {$unwind:"$countries"},
    {$group:{_id:"$countries",numMovies:{$sum:1}}},
    {$match:{numMovies:{$gte:10}}}
  ])
```

Result

```
{ "_id" : "Canada", "numMovies" : 11 }
{ "_id" : "Germany", "numMovies" : 14 }
{ "_id" : "France", "numMovies" : 16 }
{ "_id" : "USA", "numMovies" : 62 }
```

Movies Related to Sports

1) **Description**
Finding movies related to any type of sport requires creating a text index before producing any type of query. Any indicators that a movie is related to sports could arise from not just one field but multiple, such as the genre, title, plot, etc. To index a field that may contain sport-related information, the field is included with the string literal "text" specified in the index document as in the query below. The query below indexes multiple fields for the text index that may contain sport-related information. With a text index on the movies collection, a text search query can be performed. The $text query operator tokenizes the search string using whitespace as a delimiter to perform an OR operation of all keywords in the search string. The query will $match all movies containing any terms from the list "sport", "football", etc. The last stage of the pipeline is to $group documents by the new field 'sportMovies' returning the count of sport-related movies using the $sum operator.

Query

```
db.movies.createIndex({"title":"text","genres":"text","plot":"text","tomato.consensus":"text
"})

db.movies.aggregate([
   {$match:{$text:{$search:"sport football hockey NFL basketball soccer tennis cricket
     olympics rugby"}}},
   {$group:{_id:"sportMovies",count:{$sum:1}}}
])
```

Result

```
{ "_id" : "sportMovies", "count" : 50 }
```

2) **Description**
The first stage of this aggregation pipeline is to $match the sport-related movies using the same $text search in the above query. It will use the $search operator to find all movies containing any keywords from the search list. From the movie's matched $sort is utilised to return the documents to the pipeline in descending order sorted by the IMDb rating. $project takes the sorted movies specifying which fields to include (title, year, IMDb rating and votes) and which fields to suppress (_id). The $limit operator passes the top 3 sport-related movies with the highest IMDb 'rating'.

Query

```
db.movies.aggregate([
    {$match:{$text:{$search:"sport football hockey NFL basketball soccer tennis cricket
    olympics rugby"}}},
    {$sort:{"imdb.rating":-1}},
    {$project:{title:1,year:1,rating:"$imdb.rating",votes:"$imdb.votes",_id:0}},
    {$limit:3}
  ])
```

Result

```
{ "title" : "Liverpool FC: Champions of Europe 2005", "year" : 2005, "rating" : 9.5,
  "votes" : 410 }
{ "title" : "Fotbolls-VM krönikan 1994", "year" : 1994, "rating" : 8.6, "votes" : 119 }
{ "title" : "Somay Ku: A Uganda Tennis Story", "year" : 2008, "rating" : 8.6, "votes" : 19 }
```

Rating and Recommending Movies

1) **Description**
This section details the process of designing and adding the field 'myRating' to the movies collection. The result sub-section provides an example of the 'myRating' value which is highlighted in yellow. Each document in the collection has a 'myRating' value associated with it. The rating is computed by calculating a score out of 5 for each of the three rating systems: Metacritic, Rotten Tomatoes and IMDb. These scores are averaged to give the final 'myRating' score out of 5. If a movie document doesn't contain a rating for all three methods this doesn't affect the final 'myRating' score. The score is computed by averaging the available scores.

Regarding the MongoDB aggregation pipeline stages, the rating is computed by first appending three new fields using the $addFields operator. The 'tomatoRating' field computes the $sum of the tomato meter and userMeter. This sum is the dividend and is passed into the $divide operator as the first argument. The second argument is the divisor which is 200 (the tomato meter and userMeter each give movies a score out of 100). This number is passed into the $divide operator and is multiplied by 5 to return the final 'tomatoRating' score out of 5. The 'metaRating' and 'imdbRating' are computed similarly. Both, however, don't make use of the $sum operator in calculating the score out of 5 as they only have one rating associated with a movie. The divisor in the second argument of the $divide operator is also different, 100 for the Metacritic score and 10 for the IMDb score. This is because of the different number system each method uses to rate movies. The $multiply operator is used the same way for both to give the ‘metaRating’ and ‘imdbRating’ fields a score out of 5. As some of the movie documents in the collection don’t have ratings for all three metrics, scores of 0 of out 5 are sometimes passed into the fields. The next stage of the pipeline deals with this issue. $cond is utilised within $addFields to set the ratings to null if they are equal to 0 as null is not considered when using the $avg operator. This prevents movies from obtaining a low ‘myRating’ score if they don’t have ratings for all three systems. The ‘tomatoRating’, ‘metaRating’, and ‘imdbRating’ scores are averaged to give the final ‘myRating’ score out of 5. The $round operator rounds the ‘myRating’ score to 1 decimal place. $unset is the last stage of the aggregation pipeline used to delete the ‘tomatoRating’, ‘metaRating’ and ‘imdbRating’ fields before updating the movie collection as only the final ‘myRating’ score is needed.

Query

```
db.movies.updateMany({},
    [{$addFields: {
      tomatoRating:  {$multiply:  [{$divide:[{$sum:["$tomato.meter","$tomato.userMeter"]},
200]}, 5]},
      metaRating: {$multiply: [{$divide:["$metacritic",100]}, 5]},
      imdbRating: {$multiply:[{$divide:["$imdb.rating",10]}, 5]}}}},
    {$addFields:{
      tomatoRating:{$cond:
             {if: {$eq: ["$tomatoRating", 0]},
          then: null,
          else: "$tomatoRating"}},
     metaRating:{$cond:
            {if: {$eq: ["$metaRating", 0]},
         then: null,
         else: "$metaRating"}},
     imdbRating:{$cond:
           {if: {$eq: ["$imdbRating", 0]},
        then: null,
        else: "$imdbRating"}} }},
    {$addFields:{myRating:{$round:[{$avg:["$tomatoRating","$metaRating",
"$imdbRating"]}, 1]}}}},
    {$unset:["tomatoRating","metaRating","imdbRating"]}])
```

Result

```
{ "_id" : ObjectId("6040bb9d879638fe95593335"), "title" : "Star Trek II: The Wrath of Khan", "year" : 1982, "rated" : "PG", "runtime" : 113, "countries" : [ "USA" ], "genres" : [ "Action", "Adventure", "Drama" ], "director" : "Nicholas Meyer", "writers" : [ "Gene Roddenberry", "Harve Bennett", "Jack B. Sowards", "Jack B. Sowards" ], "actors" : [ "William Shatner", "Leonard Nimoy", "DeForest Kelley", "James Doohan" ], "plot" : "With the assistance of the Enterprise crew, Admiral Kirk must stop an old nemesis, Khan Noonien Singh, from using the life-generating Genesis Device as the ultimate weapon.", "poster" : "http://ia.media-imdb.com/images/M/MV5BMTcwNjc5NjA4N15BMl5BanBnXkFtZTcwNDk5MzI4OA@@._V1_SX300.jpg", "imdb" : { "id" : "tt0084726", "rating" : 7.7, "votes" : 86687 }, "tomato" : { "meter" : 88, "image" : "certified", "rating" : 8, "reviews" : 49, "fresh" : 43, "consensus" : "Considered by many fans to be the best of the Star Trek movies, Khan features a strong plot, increased tension, and a sharp supporting performance from Ricardo Montalban.", "userMeter" : 90, "userRating" : 3.8, "userReviews" : 84279 }, "metacritic" : 71, "awards" : { "wins" : 2, "nominations" : 9, "text" : "2 wins & 9 nominations." }, "type" : "movie", "myRating" : 3.9 }
```

2) **Description**
The $match operator is used to recommend movies according to the type of movies I enjoy. It matches drama, western or crime movies from the ’60s or ’80s. Movies must also have more than 2 award wins, a ‘myRating’ score of 4.0 or greater and be produced in the UK or USA. Matching these preferences will pass the selected movies to the next stage of the pipeline. Movie recommendations are specified with only the title and ‘myRating’ score. The last pipeline stage uses $limit to reduce the number of movie recommendations based on my preferences to 3.

Query

```
db.movies.aggregate([
     {$match: {
        $and:[
         {$or:[
             { year: {$gte: 1960, $lt: 1970} },
             { year: {$gte: 1980, $lt:1990} }
          ]},
          {genres:{$in: ["Drama", "Western", "Crime"]}},
          {"awards.wins":{$gt:2}},
          {myRating:{$gte:4}},
          {countries:{$in: ["USA","UK"]}},
         ]
        }
      },
      {$project:{title:1,myRating:1,_id:0}},
      {$limit:3}
      ])
```

Result

```
{ "title" : "Once Upon a Time in the West", "myRating" : 4.3 }
{ "title" : "Dead Poets Society", "myRating" : 4.1 }
{ "title" : "Drugstore Cowboy", "myRating" : 4.1 }
```