MongoDB

Davide Antonino Giorgio
Politecnico di Torino
Student id: s291477
s291477@studenti.polito.it
http://giorgiodavide.it

***Abstract*—The project aims to apply the knowledge acquired on MongoDB. The dataset provided contains information on films with reviews from IMDB and Rotten Tomatoes.**

I. Project overview

The project is based on the processing and analysis of film information data. The use of MongoDB is required for the study. The dataset also provides information related to reviews on IMDB and Rotten Tomatoes. The goals of the project are: data ingestion and curation, standard query resolution, aggregation query resolution using MongoDB aggregation framework and aggregation query resolution using Map-reduce paradigm.

II. Data ingestion and curation

After looking at the collection provided I decided to proceed as follows:

- **year**: is a *string* but should be an *integer*

```
db.imdb.find({year: {$exists: true}}).forEach(
   function(doc) {
      var init_value = new NumberInt(doc.year);
      db.imdb.updateOne(
         {"_id": doc._id},
         {"$set": { "year": init_value }}
      );
   }
)
```

- **imdb.votes**: is a *string* but should be an *integer*

```
db.imdb.find({"imdb.votes": {$exists: true}}).forEach(
   function(doc) {
      var init_value = new NumberInt(doc.imdb.votes);
      db.imdb.updateOne(
         {"_id": doc._id},
         {"$set": { "imdb.votes": init_value }}
      );
   }
)
```

- **lastupdated**: is a *string* but should be a *date* (*ISODate*)

```
db.imdb.find({lastupdated: {$exists: true}}).forEach(
   function(doc) {
      var init_value = new Date(doc.lastupdated);
      db.imdb.updateOne(
         {"_id": doc._id},
         {"$set": { "lastupdated": init_value }}
      );
   }
)
```

These examples show the command to convert each individual field analysed. No other critical issues were found during this phase. The consequences of an incorrect data ingestion phase could lead to an inability to carry out certain comparison steps during future analyses. The nature of the attributes (nominal, ordinal, cardinal and so on) allows them to be treated and different operations to be applied between them. It is therefore important to recognise the correct type of each variable in this phase.

III. Standard query resolution

The following are the queries required. I thought it more appropriate to avoid including the screenshots of MongoDBCompass and preferred the queries as accepted by the MongoDB CLI.

1) Find all the movies which have been scored higher than 4.5 on Rotten Tomatoes. The reviews for Rotten Tomatoes are contained in the tomatoes nested document. Sort the results using the ascending order for the release date.

```
db.imdb.find({
   "tomatoes.viewer.rating": {$gt: 4.5}
}).sort({released: 1})
```

   In this query the first set of the result has the *released* field empty. And this is what we expected since there were no different specifications on the documents with empty releases. $gt is used to select only the values of "*tomatoes.viewer.rating*" that are greather than a threshold (4.5). Finally I ordered the result on the field "*released*".
2) Find the movies that have been written by 3 writers and directed by 2 directors.

```
db.imdb.find({
   directors: {$size: 2},
   writers: {$size: 3}
})
```

   In this case we are looking at the size of the two lists (*directors* and *writers*) using the operator $size.
3) For the movies that belong to the "Drama" genre and belong to the USA country, show their plot, duration (runtime), and title. Order the results according to the descending duration.

```
db.imdb.find({
   "genres": {$in: ['Drama']},
   "countries": {$in: ['USA']}
   },{_id: 0, title: 1, plot: 1, runtime: 1}
).sort({ runtime: -1 })
```

In this query I used the operator $in to check if the value is contained in the list observed. For example 'USA' in *countries*. Then I used the *projection* for select and display only the fields that are required. The result is ordered according to the descending duration.

4) Find the movies satisfying all the following conditions:
   - have been published between 1900 and 1910
   - have an imdb rating higher than 9.0
   - contain the fullplot attribute

```
db.imdb.find({
   released: {
      $gte: new Date(1900, 0, 1),
      $lte: new Date(1910, 12, 31)},
   "imdb.rating": {$gt: 9.0},
   fullplot: {$exists: true}
},{ _id: 0,
   year: 1,
   "fullplot_length": {$strLenCP: "$fullplot"}
}).sort({ "imdb.rating": 1 })
```

This query returns an empty set because there are no films in the date range considered that have an *imdb.rating* value greater than 9.0. The date range is selected using $gte and $lte to select the interval of the years that we want to select. For *fullplot*, I used $exists operator that select only the documents where the field exists. In the projection, instead, the length og the fullplot is retrieved using the operator $strLenCP.

IV. Aggregation framework and map-reduce

Below is reported the list of solutions proposed for this section.

1) Find the average rating score on Rotten Tomatoes for each publication year.

```
db.imdb.aggregate([{
   $group: {
      _id: "$year",
      avg_rating: {$avg: "$tomatoes.viewer.rating"} }
}])
```

2) For movies that include Italy as a country, get the average number of directors. Be sure to consider only the movies that contain the list of directors.

```
db.imdb.aggregate([
   {$match: {
      countries: {$in: ["Italy"]},
      directors: {$exists: true}
   }},
   {$group: {
      _id: null,
      avg_num_directors: {$avg: {
         $size: "$directors"
      }}
   }}
])
```

3) Considering only movies that:
   - contain information about IMDB score ratings
   - contain a number for IMDB score ratings (you can check it by using `$type`)

   compute, separately for each movie's genre:
   - the average published year
   - the maximum score on IMDB

```
db.imdb.aggregate([
   {$match: {
      "imdb.rating": {$exists: true, $type: "number"}
   }},
   {$unwind: "$genres"},
   {$group: {
      _id: "$genres",
      avg_published_year: {$avg: "$year"},
      max_score_IMDB: {$max: "$imdb.rating"}
   }}
])
```

4) Count the number of movies directed by each director. Sort the results according to the descending order of the number of directed movies.

```
db.imdb.aggregate([
   {$unwind: "$directors"},
   {$group: {
      _id: "$directors",
      number_of_movies: {$sum: 1}
   }},
   {$sort: {
      number_of_movies: -1
   }}
])
```

This query can be useful in a real scenario if we are interested to the number of movies that are directed by each director. For example, if we wanted to give an award to the director who has made the most films ever. It would also be interesting to produce a similar query but using more interesting time slots. For example, look for the best directors in the last 10 years who have made the most films.

Regarding the Map-reduce framework, instead, the following are the solutions proposed for the problems.

1) Find the number of movies published for each year.

```
var map_f = function() {
   emit(this.year, 1);
}

var red_f = function(key, values) {
   return Array.sum(values);
}

db.imdb.mapReduce(
   map_f,
   red_f,
   {out: {inline: 1}}
)
```

The idea is to emit from the map phase a set of counters for the same year in order to then sum them in the reduce function and count the number of movies published for each year.

2) Group movies according to their number of writers. For each group, find the average number of words in the title. NB: Check in the map function if the writers attribute is defined (i.e., if it exists).

```
var map_f = function() {
   if (this.writers) {
      emit(this.writers.length,
      String(this.title).split(" ").length);
   }
}
```

```
var red_f = function(key, values) {
   return Array.sum(values)/values.length;
}

db.imdb.mapReduce(
   map_f,
   red_f,
   {out: {inline: 1}}
)
```

In this case I decided to emit a pair where the key is the length of the writers array (that is equal to the number of writers) and for the value I used an array of words of the title. Finally in the reduce function I obtained the average number of words in each title computing the total sum of the words divided by the number of the words.

3) Count the number of movies available for each language (attribute languages).
   NB: Check in the map function if the languages attribute is defined (i.e., if it exists).
   NB2: It is possible to emit multiple pairs for each document using iterators over an array.

```
var map_f = function() {
   if (this.languages) {
      for (const language in this.languages) {
         emit(String(language), 1)
      }
   }
}

var red_f = function(key, values) {
   return Array.sum(values);
}

db.imdb.mapReduce(
   map_f,
   red_f,
   {out: {inline: 1}}
)
```

In this solution I emitted a new pair for each language in the considered movie. The value is just a number used in the reduce function to compute the total number of movies for the specified language.

Two functions have been defined for all solutions: map_f and red_f, which are the mapper and reducer of the map-reduce framework respectively. The final result of the map-reduce framework is directly printed inline.

V. Interest Queries

For each year and for each genre, show the average number of votes and the average rating on IMDB and Rotten Tomatoes for all films that have been reviewed on both IMDB and Rotten Tomatoes. Sort the result in descending order by year and alphabetically by genre.

```
db.imdb.aggregate([
  {$match: {
      "imdb.votes": {
         $exists: true,
         $type: "number",
         $gt: 0},
      "tomatoes.viewer.rating": {
         $exists: true,
         $type: "number",
         $gt: 0}
   }},
   {$unwind: "$genres"},
   {$group: {
      _id: {
         "year": "$year",
         "genre": "$genres"
      },
      avg_num_votes_imdb: {
         $avg: "$imdb.votes"},
      avg_imdb_rating: {
         $avg: "$imdb.rating"},
      avg_num_votes_tomatoes: {
         $avg: "$tomatoes.viewer.numReviews"},
      avg_tomatoes_rating: {
         $avg: "$tomatoes.viewer.rating"}
   }},
   {$sort: {
      "_id.year": -1,
      "_id.genre": 1
   }}
])
```

Find all titles and comments of Italian films in the Horror genre that were released before 2000. For each comment it shows the email of the writer and the text of the comment.

```
db.imdb.find({
   "genres": {$in: ['Horror']},
   "countries": {$in: ['Italy']},
   "released": {$lt: new Date(2000, 1, 1)},
   title: {$exists: true},
   "comments.email": {$exists: true},
   "comments.text": {$exists: true}},
   {_id: 0, title: 1, "comments.email": 1, "comments.text":
       1}
)
```

a useful query to understand what people thought before the 2000s about Italian horror films.

Find all films in the Sicilian language. Show title and country of production.

```
db.imdb.find({
   "languages": {$in: ['Sicilian']}},
   {_id: 0, title: 1, "countries": 1}
)
```

Happy to find a dataset that also contains information on films in the language of my native land. That is really interesting for me.