Politecnico di Torino

IMDB database exercises

DANIELE APILETTI

POLITECNICO DI TORINO

Main steps

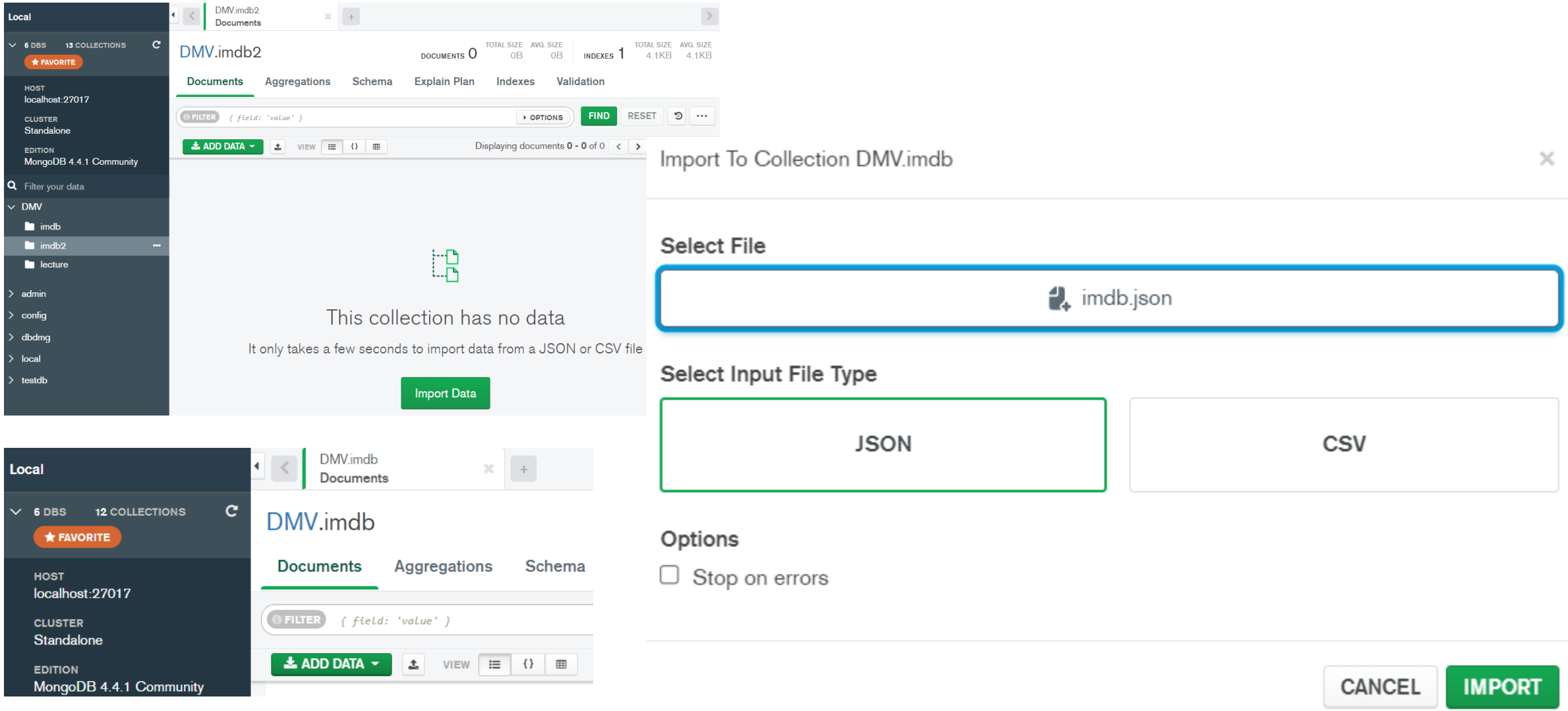
Create and query a MongoDB database containing IMDB Movie data:

1. Data ingestion and curation.
2. Standard query resolution.
3. Aggregation query resolution using MongoDB aggregation framework.
4. Aggregation query resolution using Map-Reduce paradigm.

The data consists of a collection of movies' information, it can be downloaded from the course web page (imdb.json ZIP archive).

This collection includes information about movies and their associated reviews on IMDB and Rotten Tomatoes.

Data ingestion



Data curation


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

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

Queries

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.

Queries

1. Solution

```
FILTER: {"tomatoes.viewer.rating": {$gt: 4.5}}
SORT:   {released: -1}
```

Queries

2.

Find the movies that have been written by 3 writers and directed by 2 directors.

Queries

2. Solution

```
FILTER: {"writers": {$size: 3}, "directors": {$size: 2}}
```

Queries

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.

Queries

3. Solution

```
FILTER: {"genres": 'Drama', "countries": 'USA'}
PROJECT:{runtime: 1, plot: 1}
SORT:   {runtime: -1}
```

Queries

3b.

For the movies that belong to **both** the “Drama” and “Fantasy” genres and belong to the USA country, show their plot, duration (runtime), and title.

Order the results according to the descending duration.

Queries

3b. Solution

```
FILTER: {"genres": {$all: ['Drama', 'Fantasy']}, "countries": 'USA'}}
PROJECT:{runtime: 1, plot: 1}
SORT:   {runtime: -1}
```

Queries

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

In the results, show the following values:

- the publication year,
- the length of the full plot in terms of number of characters.

Sort the results according to the ascending order of the IMDB rating.

Queries

4. Solution

```
FILTER: {"year": {$gte: 1900}, "year": {$lte: 1910},
         "fullplot": {$exists: true}, "imdb.rating": {$gte: 7.0}}
PROJECT:{"length" : { $strLenCP: "$fullplot" }, year: 1,
         "imdb.rating":1}
SORT:   {"imdb.rating": -1}
```

Aggregation

1.

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

Aggregation

1. Solution

$group

```
{
  _id: "$year",
  average_tomatoes_score: {
       $avg: "$tomatoes.viewer.rating",
  }
}
```

Aggregation

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.

Aggregation

2. Solution

$match

```
{
  countries: {$in : ["Italy"]}
}
```

$group

```
{
  _id: "italian_movies",
  avg_num_directors: {
       $avg: { $size: "$directors" },
  }
}
```

Aggregation

2. Solution

$match

```
{
  countries: {$in : ["Italy"]},
  directors: {$exists: true}
}
```

$group

```
{
  _id: "italian_movies",
  avg_num_directors: {
       $avg: { $size: "$directors" },
  }
}
```

Aggregation

3.

Considering only movies that:

a. contain information about IMDB score ratings

b. 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, and

- the maximum score on IMDB.

Aggregation

3. Solution

$match	$unwind	$group
{ "imdb.rating": {$exists: true}, "imdb.rating": {$type: "number"}, }	{ path: "$genres", }	{ _id: "$genres", avg_year: { $avg: "$year", }, avg_score_imdb: { $max: "$imdb.rating", } }

Aggregation

4.

Count the number of movies directed by each director.

Sort the results according to the descending order of the number of directed movies.

Aggregation

4. Solution

$match	$unwind	$group
{ path: "$directors", }	{ _id: "$directors", num_movies: { $sum: 1 } }	{ num_movies: -1 }

Map Reduce

1.

Find the number of movies published for each year.

Map Reduce

1. Solution

```
var mapFunction1 = function() {
   emit(this.year, 1);
};


var reduceFunction1 = function(id, values) {
   return Array.sum(values);
};

db.movies.mapReduce(
   mapFunction1,
   reduceFunction1,
   { out: "num_movies_per_year" }
)

db.num_movies_per_year.find()
```

Map Reduce

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).

Map Reduce

2. Solution

```
var mapFunction2 = function() {
   if (this.writers !== undefined){
      var l = String(this.title).split(" ").length
      emit(this.writers.length, l);
   }
};

var reduceFunction2 = function(id, values) {
   return Array.sum(values)/values.length;
};

db.movies.mapReduce(
   mapFunction2,
   reduceFunction2,
   { out: "average_title_perwriters" }
)

db.average_title_perwriters.find()
```

Map Reduce

3.

Count the number of movies available for each language (attribute languages).

NB1: 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.

Map Reduce

3. Solution

```
var mapFunction3 = function() {

    if (this.languages !== undefined){
        for (var i = 0; i < this.languages.length; i++) {
                emit(this.languages[i], 1);
        }
    }
};

var reduceFunction3 = function(id, values) {
    return Array.sum(values);
};

db.movies.mapReduce(
    mapFunction3,
    reduceFunction3,
    { out: "num_movies_perlanguage" }
)

db.num_movies_perlanguage.find()
```

«Your query» challenge

1. Write and solve one additional novel query, defined by you: when and how it could be helpful in a real use-case?

2. Post it on the "challenge" folder on Piazza
   - publish the text only, first

3. Read the proposed queries, learn by finding a collective solution.

4. Vote the best queries!