Chapter 1

Document database

In this chapter we will discuss of the organization of the document database and how we handled the replicas. We decided to use MongoDB as DBMS for the document database for the purpose of storing the main information for movies, users, reviews and personnel. In MongoDB we created the following three collection:

- movie
- user

Inside the movie collection there are embedded the documents for the reviews and personnel

1 Collection composition

1 .1 Movie collection

The following is the composition of the Movie collection.

```
{
    "_id" : ObjectId(<<id_field>>),
    "primaryTitle": "The first ever movie",
    "year": 1989,
    "runtimeMinutes": 70,
    "genres": ["Crime", "Drama", "Romantic"],
    "productionCompany": "Dingo Picture Production" ,
    "personnel": [
        {
            "name": "John Doe",
            "category": "producer",
            "job": "writer"
        },
        {
            "name": "Christopher Lee",
            "category": "actor",
            "character": ["The one"]
        },
        ...
    ],
    "top_critic_fresh_count": "4",
    "top_critic_rotten_count": 0,
    "user_fresh_count": 14,
```

```
    "user_rotten_count": 1,
    "top_critic_rating": 100,
    "top_critic_status": "Fresh",
    "user_rating": 93,
    "user_status": "Fresh",
    "review":
    [
        {
            "critic_name":    "AntonyE",
            "review_date": "2018-12-07",
            "review_type": "Rotten",
            "top_critic":  false,
            "review_content":"I really didn't liked it!",
            "review_score":   "1/10"
        },
        {
            "critic_name":    "AntonyE",
            "review_date": "2015-12-07",
            "review_type": "Fresh",
            "top_critic":  true,
            "review_content":"I really liked it!",
            "review_score":   "A-"
        },
        ...
    ]
}
```

Listing 1.1: Test

As previously stated the field personnel and review are arrays of embedded documents.

1 .2 User collection

The following is the composition of the User collection.

```
{
    "_id"               : ObjectId(<<id_field>>),
    "username"          :"AntonyE",
    "password"          :"hashed_password",
    "firstName"         :"Anton",
    "lastName  "        :"Ego",
    "registrationreview_date"  :"2019-06-29",
    "review_date of birth"     :"2002-07-16",
    "last3Reviews":
        [
            {
                "_id"               : ObjectId(<<id_field>>),
                "primaryTitle"    : "Star Wars: A new Hope",
                "review_date"          : "2018-12-07",
                "review_type"     : "fresh",
                "top_critic"   : false,
                "review_content"       : "I really liked it!",
                "vote"          : "8/10"
```

```
            },
            {
                "_id"               : ObjectId(<<id_field>>),
                "primaryTitle"    : "Ratatouille",
                "review_date"           : "2019-02-17",
                "review_type"     : "rotten",
                "top_critic"   : false,
                "review_content"        : "The director was also
    ↪ controlled by a rat",
                "vote"          : "D+"
            },
            {
                "_id"               : ObjectId(<<id_field>>),
                "primaryTitle"    : "300",
                "review_date"           : "2020-02-15",
                "review_type"     : "fresh",
                "top_critic"   : false,
                "review_content"        : "Too much slow motion",
                "vote"          : "3.5/5"
            }
        ],
     "reviews":[
        {
            "movie_id" : ObjectId(<<id_field>>),
            "primaryTitle" : "Evidence",
            "review_index": 11
        },
        ...
     ]

}
```

Listing 1.2: Test

Is important to notice that in this collection we have two fields for the reviews, *last3Reviews* and *reviews* which present different ways of storing data for the same underling entity. As suggested by the name itself *last3Reviews* contains the last three review in the form of an embedded document meanwhile *reviews* contains all the reviews made by a user in the form of document linking towards the movie for which they were written; in particular, the link is formed by the id of the movie, the title and the position in which the review can be found in the array *review* of the movie document. This structure was chosen so that we could avoid a full replica of the reviews in both collections and, at the same time, avoiding to perform join operation for those reviews that are more frequently checked, which are the most recent ones. The idea of creating a separated collection for reviews was immediately discarded because it would have resorted in a design for the document database that resemble a third normal form of relational database.

2 Indexes

In order to provide the best execution speed in search queries we use indexes. In particular we focus on the application part that is available also without registering to the site, like Hall of Fame and search movies functionalities. Without indexes we need a collection scan in user and

movie collections to find the right document.

2 .1 Movie collection

- primaryTitle
- year
- genres
- top_critic _rating
- user_rating

```
// Dimension is expressed in Kb
{
  _id_: 228,
  primaryTitle_1: 292,
  year_1: 92,
  genres_1: 180,
  'personnel.primaryName_1': 1680,
  top_critic_rating_1: 80,
  user_rating_1: 80
}
```

Listing 1.3: Test

2 .2 User collection

- username
- date_of _birth

```
// Dimension is expressed in Kb

{ _id_: 180, username_1: 168, date_of_birth_1: 100 }
```

Listing 1.4: Test

We decided to use these indexes because they provide a jump in term of speed in search queries without occupying much space. We accept the fact that writes are slower because our application is read-intesive. We consider the idea of using an index on personnel.primaryName but the cost in term of space was higher than potential benefit. For a full analityc report for the indexes you can see the appendix. (Appendix **??**)

3 Partition and replicas

To ensure high-availability we deploy a cluster of replica (3). We install and configure MongoDB in all machines, with these priorities

1. 172.16.5.26 (primary)
2. 172.16.5.27 (secondary)
3. 172.16.5.28 (secondary)

We decided to use **nearest read preference** and **W2 write preference**. In fact we can tolerate that users see temporarily an old version of data with a 33% chanche.

4 Aggregations

In this section we shall discuss on the different aggregation that the application will implement, the values between «» represent a values passed by an above level.

4 .1 Return the best years by top critic and user ratings

```
db.movie.aggregate([
    {$group:
        {
        _id: "$year",
        topCritic:{$avg:"$top_critic_rating"},
        rate:{$avg:"$user_rating"},
        count:{$sum:1}
        }
    },
    {$match:{count:{$gte:<<i>>}}},
    {$sort:{topCritic:-1, rate:-1}},
    {$limit: <<j>>}
])
```

Listing 1.5: Test

4 .2 Return the best genres by top critic and user ratings

```
db.movie.aggregate([
    {
        $unwind:"$genres"
    },
    {$group:
        {
            _id: "$genres",
            topCritic:{$avg:"$top_critic_rating"},
            rate:{$avg:"$user_rating"},
            count:{$sum:1}
        }
    },
    {$match:{count:{$gte:<<i>>}}},
    {$sort:{topCritic:-1, rate:-1}},
    {$limit:<<j>>}
])
```

Listing 1.6: Test

4 .3 Return the best production houses by top critic and user ratings

```
db.movie.aggregate([
    {$group:
        {
            _id: "$production_company",
            topCritic:{$avg:"$top_critic_rating"},
            rate:{$avg:"$user_rating"},
            count:{$sum:1}
        }
    },
```

```
    {$match:{count:{$gte:<<i>>}}},
    {$sort:{topCritic:-1, rate:-1}},
    {$limit:<<j>>}
])
```

Listing 1.7: Test

4 .4 Given a movie count the review it has received by each month

```
db.movie.aggregate([
    {
        $match:{id:<<"id">>}
    },
    {$unwind:"$review"},
    {
        $group:
        {
            _id:{year:{$year:"$review.review_date"}, month:{$month:"$review.review_date"}},
            count:{$sum:1}
        }
    },
    {$sort:{_id:1}}

])
```

Listing 1.8: Test

4 .5 Given a user count the review he/she has made divided by genres

```
db.movie.aggregate([
    {$match:
        {"review.critic_name":{$eq:<<"name">>}}
    },
    {$unwind:"$genres"},
    {$group:{_id:"$genres", count:{$sum:1}}},
    {$sort:{count:-1}},
    {$limit: <<n>>}
])
```

Listing 1.9: Test

4 .6 Return the number of user divided by an age bucket

```
db.user.aggregate([
    {
        $match:{"date_of_birth":{$exists:true}}
    },

    {
        $bucket:
        {
            groupBy: {$year:"$date_of_birth"},
            boundaries: [<<values>>],
            output:
            {
```

```
                    "population": {$sum:1}
                }
            }
        }
])
```

Listing 1.10: Test