# Base de Dados Avançadas

Project - Group 7




André Mendes fc54453 | Filipa Almendra fc54396 | Rafaela Rodrigues fc54383 | Ricardo Pedro fc55309

# Introduction

This work has the purpose of understanding the differences between Relational Databases and Non-Relational Databases, in the context of the Advanced Databases Course. The differences can be reflected in the structure of the database, the way queries are made and their efficiency. Our group created two databases, one using Sqlite and the other using MongoDB, which are relational and non-relational, respectively.

# Description of the dataset

The dataset consists of 3 CSV's based on Disney movies:

**disney-characters:**

- movie_title - the title of the movie
- release_date - the release date of the movie
- hero - the hero of the movie
- villain - the villain of the movie
- song - the main song of the movie

**disney-voiceactors:**

- movie - the title of the movie
- voice-actor - the name of the voice actor(s) one specific character has
- character - the name of the character the voice actor acts

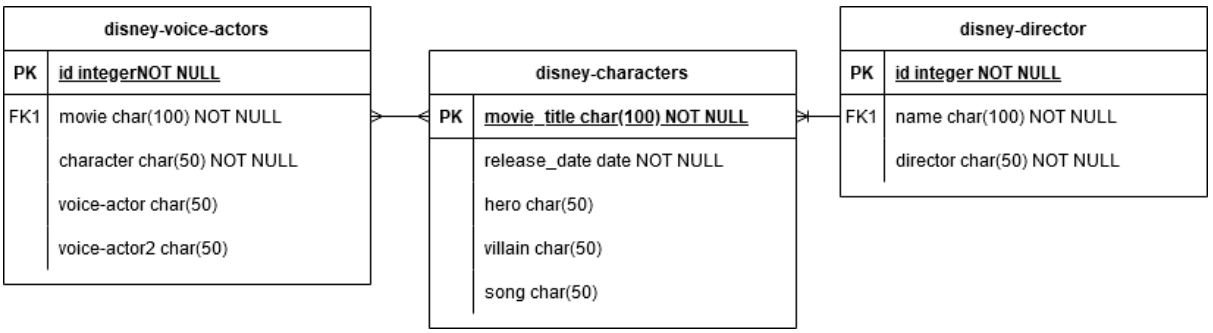
**disney-directors:**

- name - title of the movie
- director - the name of the director of the movie

# Database schemes

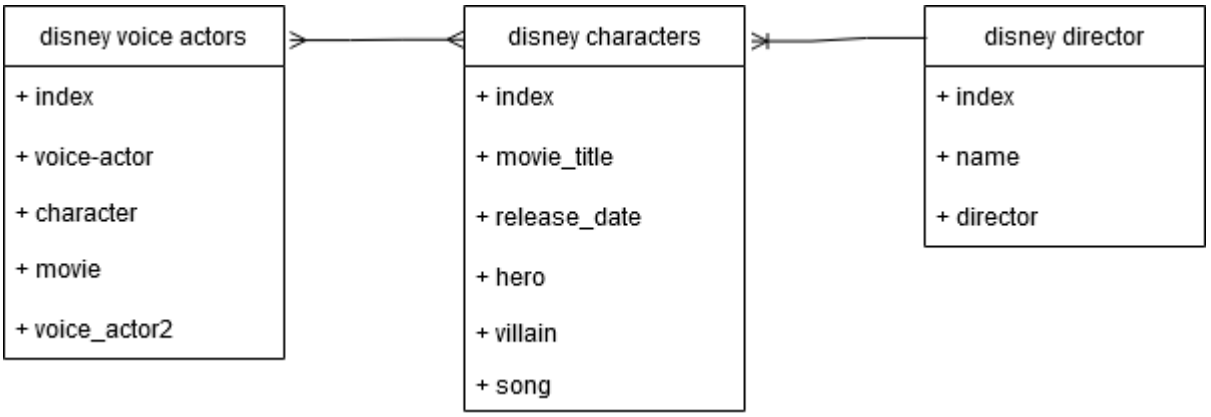
## Relational

In the first schema, we are representing our schema for the relational database, where we have three tables in which the name of the movie is the common column in every table. Each table has a primary key, relevant fields to that table and their types, and the director and voice-actors will have a foreign key, which is the name of the movie that comes from the characters table. The table of disney-characters represents a movie and its features. A voice-actor can impersonate more than one character, a character can have more than one voice-actor, and a director can direct more than one movie, but a movie has only one director.




## Non-Relational

In this case, we can see the exact same names and attributes but displayed in collections and the information we have to provide is reduced.




# Points done/not done in the project

We completed the 4 goals of the project, which means we created the databases, collections, tables, queries and indexing, in Sqlite and MongoDB, being the last goal could have been done better.

## Impacts on each query performance

### SQL

Regarding SQLite, we can only apply one type of indexing and we could apply it to one column or create one with more than one column. We observed our current queries and we identified that the performance could improve with the creation of the following indexes: in the table characters, create a multi-column index with movie_title, hero and villain - this way it will be more efficient to access the correspondent hero or villain from a certain movie; in the table voice_actors, create an index with the movie and the character. We tested both implementations with and without indexes and we can observe some improvements.

Without indexes:
sel_a_1 time:  0.00015592575073242188
sel_a_2 time:  0.00011086463928222656
sel_b_1 time:  0.0009520053863525391
sel_b_2 time:  0.0004241466522216797
ins_c_1 time:  0.0002231597900390625
up_c_2 time:  4.9114227294921875e-05

With indexes:
sel_a_1 time:  0.00014495849609375
sel_a_2 time:  0.00011873245239257812
sel_b_1 time:  0.0003581047058105469
sel_b_2 time:  0.0003693103790283203
ins_c_1 time:  0.0003032684326171875
up_c_2 time:  0.00015997886657714844

Where the time is given in seconds.

## MONGODB

In mongoDB there are more types of indexing and the two we think were the most beneficial are the compound index and the text index. Firstly, we used the compound index in order to combine documents. We used Text indexes to remove certain words of movie names from every collection, and we added the used keys in the queries, having almost no difference to if they're ascending or descending.

```
select1query
Time:  3.0517578125e-05
Time:  8.106231689453125e-06
select2query
Time:  6.67572021484375e-06
Time:  5.9604644775390625e-06
sel_comp1
Time:  0.02227473258972168
Time:  0.0015513896942138672
sel_comp2
Time:  0.021343708038330078
Time:  0.0017695426940917969
```

The times shown are referring to the simple and complex select queries and the times are shown again in seconds.

# How to replicate the project: the creation of the databases, and running the queries

Regarding the relational database, we start by using the module sqlite3 and connect our “open_disney.db” database. With the connection object we just created, we will apply a cursor so that we are able to perform queries. After that, we executed the queries to create the tables, insert queries to add the data to the tables and requested other queries.
And now regarding the non-relational database, we used MongoDB, and we used pandas alongside pymongo. We start by making the connection to the database, and creating a client. After making the connection to the client we made collections for every CSV file. We decided to drop the collections if they are done and then insert the data into every collection. Then to perform the queries we use find(). After that, we just do what we did in SQLite.

# Queries

1. Select voice actors from the movie “The Little Mermaid”.

```
SELECT voice_actor1 FROM voice_actors WHERE movie = 'The Little
Mermaid';
```

```
disneyVA.find({'movie':"The Little Mermaid"}, { 'voice-actor': 1})
```

2. Select characters who have more than one voice_actor.

```
SELECT character FROM voice_actors WHERE voice_actor2 is not
null;
```

```
disneyVA.find({"voice_actor2" : { "$ne" : None}}, {"character" : 1})
```

3. Select the heroes from the movie(s) in which the name of the director starts with a “B” and that movie has more than 12 voice_actors.

```
SELECT hero FROM characters
INNER JOIN directors ON directors.name = characters.movie_title AND
directors.director like 'B%'
INNER JOIN voice_actors
ON voice_actors.movie = characters.movie_title
GROUP BY voice_actors.movie HAVING count(voice_actors.movie) > 12;
```

```
sel_comp1 = disneyC.aggregate([
    {"$lookup": {# Join with director table
        "from": "disneyD",       # other table name
        "localField": "movie_title",   # name of disneyD table field
        "foreignField": "name", # name of userinfo table field
        "as": "disney_director",}},# alias for userinfo table
    {"$unwind" : "$disney_director"},
    {"$match": {"disney_director.director": { "$regex": "^B" }  }},
    {"$lookup":{
        "from": "disneyVA",
        "localField": "movie_title",
        "foreignField": "movie",
        "as": "disney_voiceactor"},},
    {"$unwind" : "$disney_voiceactor"},
    {"$group":{"_id":"$disney_voiceactor.movie", "count":{"$sum":1},"hero" : {"$first" : "$hero"}}},
    {   "$match":{"count":{"$gt":12}}},{"$project" : {"hero" : 1}}
])
```

4. Select the villains from the movie, where the voice actor(s) of the villain did not work with the Director “Ron Clements”.

```
SELECT villain FROM characters
INNER JOIN voice_actors ON characters.movie_title = voice_actors.movie
AND characters.villain = voice_actors.character
LEFT JOIN directors ON characters.movie_title = directors.name AND
directors.director == "Ron Clements" WHERE directors.director IS NULL;
```

```
sel_comp2 = disneyC.aggregate([
    {"$lookup": {
        "from": "disneyVA",
        "localField": "movie_title",
        "foreignField": "movie",
        "as": "disney_voiceactor"},},
    {"$unwind" : "$disney_voiceactor"},
    {"$match" : { "$expr" : {"$eq" : ["$disney_voiceactor.character","$villain"]}}},
    {"$lookup":{
        "from": "disneyD",
        "localField": "movie_title",
        "foreignField": "name",
        "as": "disney_director"},},
    {"$unwind" : "$disney_director"},
    {"$match": {"disney_director.director": {"$ne": "Ron Clements"}}},
    {"$project" :{"villain" : 1}},
])
```

5. Insert of Director “Stephen Hillenburg” with the movie name “Spongebob Squarepants” value on the directors table.

```
INSERT INTO directors (director, name) values ("Stephen
Hillenburg","Spongebob Squarepants");

disneyD.insert_one({'director':'Stephen Hillenburg','movie':'Spongebob
Squarepants'})
```

6. Update the value of the villain from the movie “The Jungle Book” to “Baloo Bear” on the character table.

```
UPDATE characters
SET villain = "Baloo Bear"
WHERE movie_title = "The Jungle Book";

disneyC.update_one({"movie_title": "The Jungle
Book"},{"$set":{"villain":"Baloo Bear"}})
```