Project Part 1

Done by: Group 18

- Pedro Santos 54129
- Rafael Oliveira 52848
- Rodrigo Serra 57414
- Rui Rocha 57588

Introduction:

From a series of csv files from dbpedia which contained basic information about various bands, albums, and members two databases were created in order to fulfil the two complex queries below using different methods.

Complex operation 1

Choose the genre less sold from the last decade and select the 10 best albums of that genre.

Steps:

1. Select the albums in the last decade.
2. Discover the worst selling genre in the discussed decade by grouping up the albuns of the last decade by genre and summing the sales of the albums of each genre.
3. Enumerate the top 10 best selling of the worst selling genre.

Complex operation 2:

Discover the worst genre in the last decade and up their sales 10 times the original value.

Steps:

1. Select the album in the last decade.
2. Discover the worst selling genre in the discussed decade by grouping up the albuns of the last decade by genre and summing the sales of the albums of each genre.
3. Up the sales of the albuns of that genre by 10 times.

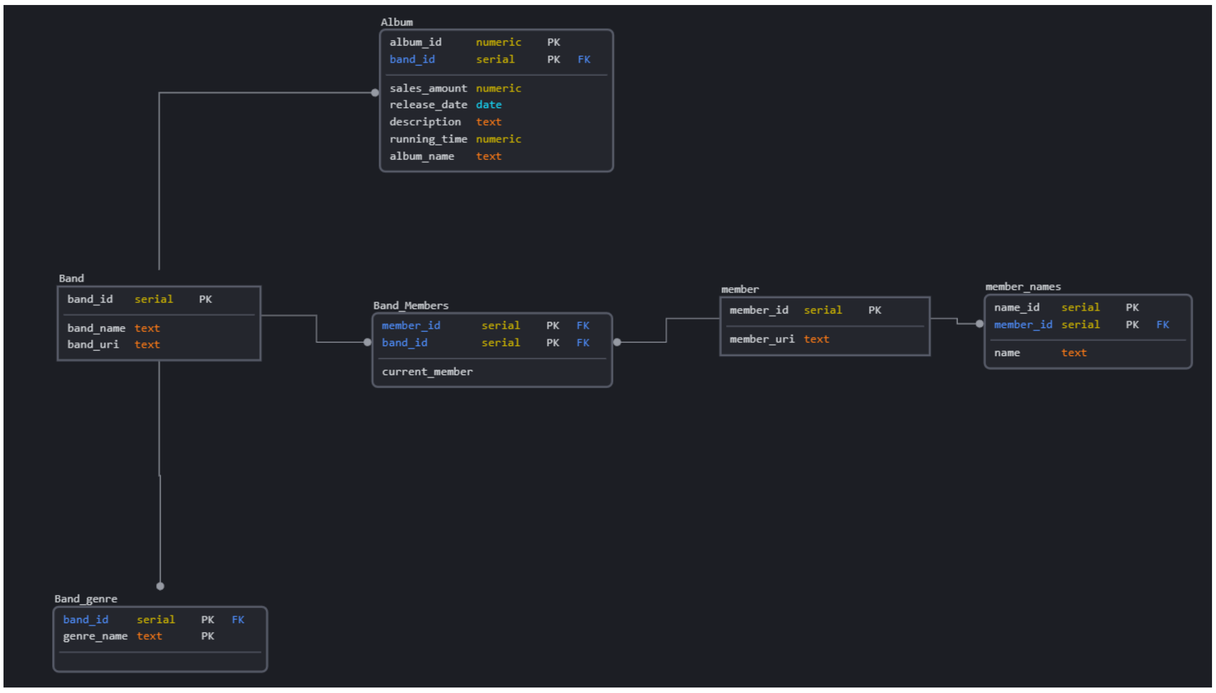
Conflict: Because the sales amount of the albums belonging to the genre found in the first query is changed by the second query both these complex operations can be conflictuous when run at the same time. The first query calculates the total sales of all genres in order to find the worst performing genre, and this amount is changed by the second query.

To study the difference between the relational model and a nosql model the data was modeled into two databases, one in SQL and the other one in Mongodb.

SQL:

The relational database was modeled based on the following schema. It contained six tables (Album, Band, Band_Members, member, member_names, and Band_genre). The tables Album, Band and member define the entities, while the table Band_genre associates a specific band with a string which contains a genre, where the same band can be associated with multiple genres; the Band_Members table associates a member with a band and stores the information of whether the member is currently in the band or is only a former member; and the member_names table associates a specific member with a string containing one of his names (the same member can have more than one string defining his name, for example, Charlie Parker can also be known as Bird, or even Charlie 'Bird' Parker).

Schema:



Inserts:

To introduce the data correctly in the database, the csv files were edited in order to clean some of the dirty data and format it correctly, in the cases where it applied. Even so, the information couldn't be directly inserted to the database, some attributes needed to be associated to their respective primary keys and then to their specific line on the csv. So temporary tables were created which contained the information more or less how it was presented in the source and using some queries to those temporary tables the database tables were created.

Queries:

The first query picks the 10 best selling albums of the worst genre of the 2010's. It does this by summing up the sales_amount of the albums that were released in the 2010's, grouping them up by the genres of their respective bands, and ordering the genres, after the calculations, by ascending order of sales amount of each respective genre to get the worst genre of the 2010's; From that genre that was the least sold we pick the 10 best selling albums, if there are that many.

The second query also calculated which was the worst genre like the first query, using that result to get all the albums associated with it in the 2010's and change their amount of sales done, multiplying it by ten.

MongoDB:

Aggregation Choice:

Our choice for the aggregation was based on our complex operations. Operation one calculates the sales of the genres by using the sales_amount of the albums and also restricts them to the decades of the albums. Operation two also updates the sales of each genre by updating the sales amount of each album in that decade and genre. With that said, since both operations mess with values of the albuns, although for example the genres of the albuns had to be linked with the album we decided that our main aggregation would be the albums.

Aggregation Modeling:

As mentioned before, both our operations messed with attributes of the album. With that said, we also had an issue. The album genre was linked to the bands. And so with that said we decided to also include the band name and band genre in the album aggregation so it was faster to fetch the data with the queries, instead of having to go to two separate aggregations, the band and the album. We had then to use the SQL tables and do a csv with all that information from the aforementioned to be then converted into documents. Another problem also arose that we needed to deal with as we were modeling the aggregate: the bands had multiple genres and that would be the “album genres”, since we didn’t have the genres of the albums themselves. To solve that issue we modeled the albums document structure to have all the genres of the band in a single array, since the band genres will never change and with that type of modeling we could save space by not having repeats of the same album unnecessarily. The document structure for the albuns was the following (with the id of the document being created as we inserted the documents):

```
{
   "album_id": "1",
   "album_name": "Glass Floor",
   "release_date": "2004-06-01",
   "sales_amount": "1133",
   "running_time": "45.05",
   "band": {
     "band_name": "Maritime (band)",
     "band_genre": ["Indie pop","Indietronica","Indie rock","Emo"]
   }
 }
```

We also modeled the band and the member aggregations. As we modeled both of them we had the same issue described earlier for the albums. For the band we had more than one member that we also decided to model in an array, and in the member we also had the issue of the members being known by various names, which we also modeled in an array for the same reasons that guided our choice in using an array for the band genres. Both of the aggregations were modeled as follows:

Band Aggregation:

```
{
   "band_id": "1",
   "band_uri": "http://dbpedia.org/resource/Maritime_(band)",
   "band_name": "Maritime (band)",
   "band_genre": ["Indietronica","Indie pop", "Emo", "Indie rock" ],
   "members": {
    "members_id": [ "2528","10607"]
   }
 }
```

Member Aggregation:

```
 {
   "member_id": "4412",
   "member_uri": "  http://dbpedia.org/resource/Jack_Yarber  ",
   "current_member":"True" ,
   "band_id": "6",
   "name": ["Jack Yarber"," Jack Oblivian"]
 }
```

Queries:

Operation 1:

For the first complex operation we used an aggregation command to first choose the decade we are choosing then we unwind the band_genre array so it's easier to group each genre. After that we do the sum of each genre's value, sort it by total sales and then we use the worst genre. We put the value of that aggregation into a variable so that we can use the genre inside of that aggregation as a variable to be compared on our find. Before the find we need to use a forEach cycle because the variable is a cursor (although the aggregation is limited to only one genre) and then we use a function so we can do the find that we then print to the user using a print json function that gives us the values of the find to the console.

Operation 2:

In operation 2 we first use the first aggregate done in operation 1 to calculate what is the worst genre and after that we put it inside a variable to use it later. Then we use a forEach with that variable, that is a cursor (hence why the forEach cycle) to use the genre inside of a find command so we can get the albuns. After that, with that information we update the sales amount by 10 times of the albuns whose album id is the same as the ones in the find, making them the worst genre a better genre.

SQL vs MongoDB :

During the development of this project we used both SQL and MongoDB and although the data used wasn't of a large enough scale to experience a performance difference, there was a development experience.

Besides that differences we also noticed:

The modeling of the database in MongoDB was simpler, because we could more easily adjust our aggregations to the data given to us in the first stages of creating the database. In contrast, SQL was easier to query because it is a more intuitive language, therefore being easier for us to turn our complex operations into queries compared to MongoDB. If there were scalability issues and we needed to choose only one database the group would choose MongoDB because of its more elastic properties, there are no empty attributes; if an attribute is not found, it's assumed that it was not set or relevant, and if needed we have the ability to create new attributes without the need to define them or change the documents.