# Data Modelling and Querying

Advanced Databases - Report 1


**Group 1**

Daniela Vieira

João Raimundo

João Rato

Maria Vieira


**Professor:** Dra. Cátia Pesquita

18 October 2021

# 1. Introduction

The first part of the project involves Data Modelling and Querying which revolves around the concepts of Relational (SQL) and Non-Relational (NoSQL) databases. Thus, we explored the advantages and disadvantages of each implementation in order to better understand the logic of each decision, until deciding the best choice for each given case.

The data for this project is centered around the information about bands, their albums, their genres and their former and current artists, which was extracted from *dbpedia*. We started by analyzing the data to understand how the different components - soon to be declared as entities and attributes - interacted with each other. Then, we created the relational (including an entity-relationship model) and non-relational models to understand how the data should be structured in the databases.

We proceeded to create both databases, the first one with PostgreSQL and the second one with MongoDB, which required a bit more research. After the databases were built, there was a need to develop some Python scripts to curate the data so it would fit our needs for our database models and consequently, to insert the respective data in the databases. Furthermore, we created two queries to examine the data, in order to identify the main differences of both querying languages.

# 2. Implementation

In the following implementation steps, it will be described how SQL and NoSQL databases were modelled in order to understand the main differences between both implementations in the DBMS (Postgresql and MongoDB, respectively). Taking this into consideration, it will be explained how the data was transformed to fit on both databases.

## 2.1 - Relational Database

Relational databases are the most used model, because they have what is known as ACID properties, which ensures the Atomicity, Consistency, Isolation and Durability of the data. Considering this, an Entity-Relationship (ER) model was built to help us understand the structure of the given data, where the main entities and their attributes, as well the relationships between each entity were defined. After that, a relational model was created, based on the ER model, to represent our database structure in more detail.

### 2.1.1 - Entity-Relationship Model

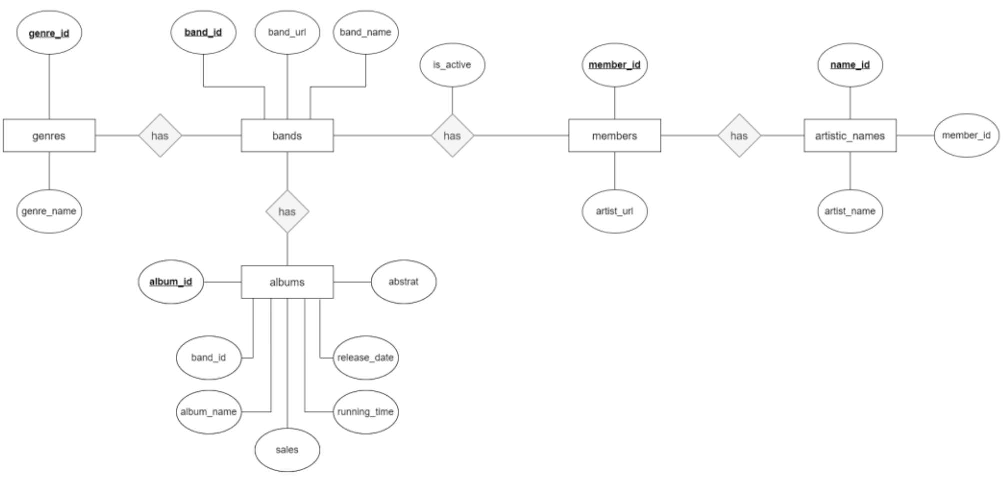


**Figure 1** - Entity-Relation Model

As you can see in Figure 1, we defined the following entities in our ER model: “bands”, “genres”, “albums”, “members” and “artistic_names”, as well as the relationships: “bands-genres”, “bands-albums”, “bands-members” and “members-artistic_names”.
The reason being for this structure is the fact that “bands”, “albums” and “members” have their own set of keys which makes them distinguishable from each other. Since both current and former members share the same keys, we decided to join both files and make it a single column. This brought a problem on how to differentiate between the current and former members, so to distinguish them, it was created the attribute “is_active” as *boolean* type, making it possible to distinguish the state of each band member. However, this brought us to another issue, which was that members can have different artistic names depending on which band they are/were in, so we decided to create an entity, with the name artistic_names, to identify each artistic name that each member can have.
Almost every entity has an unique identifier (url), but in most of the cases they are big strings, so we decided to create an unique identifier of the type “serial” - this way improving the performance of the queries since it makes the comparison of IDs easier as well as the joins.

## 2.1.2 - Relational Model

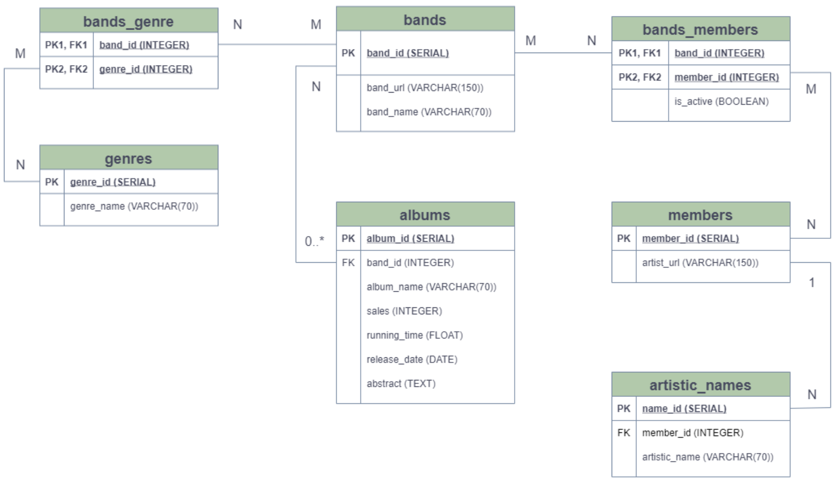


**Figure 2** - Relational Model

From the ER model and considering the data, as you can see in Figure 2, we determined all the data types and constraints, including unique, primary and foreign keys to help make the querying more efficient since that is one of the strongest points of relational databases.
We also defined the type of relationships between the entities where the relationship between “bands” and “members” is many-to-many, as each band can have multiple “members” and each member can belong to several “bands”.
As said before, “members” can have more than one “artistic_name” for each “artist_url”, depending on which band they are/were in, so the relation between these two is one-to-many. The relationship between “genres” and “bands” is also many-to-many, as each band can have multiple “genres”, and the same genre can relate to several different “bands”.
The relationship between the “bands” and the “albums” is a many-to-many relationship since in some cases for the same album, more than one band is featured in that same album.

### 2.1.3 - Data Processing for Relational Database

To fit the data in the database, we performed the data cleanup developing a Python script (Appendix 6.1) and defined each of the original files as a dataframe. After the curation of the data, we created a SQL script (Appendix 6.2) to create the tables and insert the files into the database.
Since we defined in our SQL model that all tables would have a serial key as their primary key, we decided to add a column to each of the following data frames with an index.
The “bands” data did not get any transformation; the “albums” data got merged both of the albums and “bands” dataframes using the “band_url”, keeping all of the columns from the “albums” and the column “band_id” from the “bands” dataframe. We also changed the “release_date” to “YYYY-MM-DD” format since PostgreSQL complained about the format. The “genres” dataframe was created by only using the unique values, and the “bands_genres” dataframe was built by merging the unique “genres” dataframe and the “bands” dataframe using the “band_url” as the index.
For the “members” table we used the same approach as the “genres” and we selected only the column of the unique values of the “artist_url”. In the “artistic_names” table we selected again the “members” dataframe but only the columns “artistic_name” and “artist_url” and then merged it with the “members” dataframe using the “artistic_url” as index for the merge dropping the column “artistic_url”. Given that, the same logic was applied to the “bands_members” so we selected the “bands” dataframe and the “members” dataframe, merging both of them with the “band_url” as the index and selected only the “members_id”, “band_id” and the “is_active” columns to create this table. Since there are cases where members joined and left bands more than one time, we decided to remove those repeated cases from the database, since it is not important for our analysis. We also found issues with some of the strings in the column “abstract” from the albums csv - this problem was related to the bad characters format (not UTF8) of some strings - so we decided to remove those characters from the “abstract” strings using the command *iconv* in the *Linux* commands line.

## 2.2 - NoSQL Database

NoSQL data models can handle large volumes of data at high speed, to store unstructured semi-structured or structured data. In this project we use MongoDB’s document data model that has many benefits such as: dynamic schema, excellent performance, adding new columns and fields without impacting the existing rows or the application’s performance.
In order to create our MongoDB database, we started by defining the aggregation schema of the database. Considering the data and the defined queries (described in the next section), we decided to do three aggregations with the entity “albums” (collection albums). The black-diamonds define the aggregation of the entities “bands”, “genres” and “artists” to the collection “albums”. Thus, Figure 3 represents the defined aggregation schema for our database, the collection “albums” it’s composed of with the attributes “album_name”, “sales”, “release_date”, “abstract” and three attributes aggregated by array: the entities “bands”, “genre” and “artists”. Thereby, the band list is composed with the attributes “band_url” and “band_name”; the artist list is composed with “artist_url”, “artist_name” and “is_active” (defines if the artist is a former or a current member); and the attribute “genre_name” defines the “genres” list. Taking the cardinality of the model in consideration, it’s the same described in section 2.1.2.

The MongoDB database was created using the DBMS MongoDBCompass, which was connected to a MongoDB Atlas cluster. Given that, a database named “bandsDB” and a collection named “albums” were created.

To structure and insert data into the MongoDB database (bandsDB.albums), a Python script was developed (insert_mongoDB.py - Appendix 6.3). Given the fact that in document models there are no empty attributes, we structure the date for the albums data and prepare a dictionary (with the attributes) for the collection. Thus, it was defined some independent functions to read and structure the entities data: “getBand(url)”, “getGenres(url)” and “getArtists(url)” In case of the non-existence of “bands”, “genres” and “artists” array/dictionary associated with the “band_url” for an album, the array/dictionary will not be appended to the album dictionary, aka document. Finally, we connected to the MongoDB Atlas cloud database and inserted all of the structured documents, in JSON format, into the database with the command “collection.insert_one(album)”.

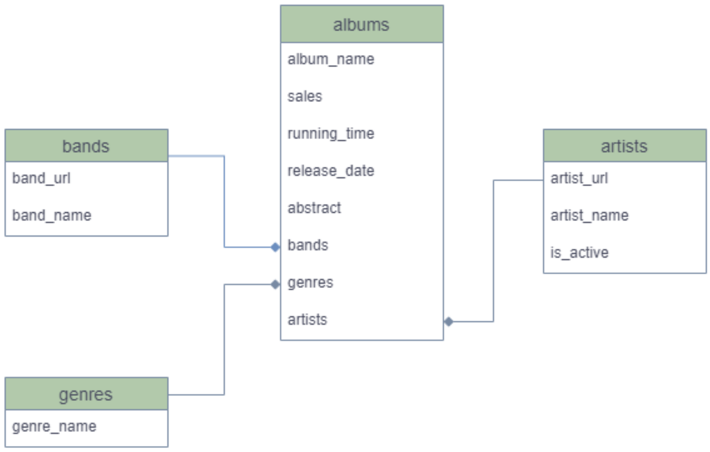


**Figure 3** - Database NoSQL aggregation schema.

## 3 - Queries and Results

In the following topics we will introduce you to our "questions" to the database and the queries that will be performed to answer those questions, both for SQL and NoSQL. This way the differences between both models are shown. In the next figures you can see the data that is going to be updated as well as the code necessary to obtain this result.

**Query 1 -** Update to 1980 the release date of all albums for the genre Math Rock, which were released in the 90's with an abstract over 200 characters, and that had the most sales.

**Query 2 -** Update to 0 the sales from the album with the most sales in the first decade of the year 2000, and which the running time is longer than 45 minutes.

## 3.1 - SQL Queries and Results

The Query 1 is made of two main parts, the UPDATE, where we select the "albums" table and then update the "release_date" of an album to "1980/01/01". The second part is the SELECT, where we will obtain the "release_date" by first making 3 INNER JOINs ("albums" with "bands", "bands" with "bands_genre" and "bands_genre" with "genres"). This way we can obtain all of the information about the "genres" of all the "bands". Thus, we are going to split the information to get only the "genre_name" that is "Math rock" and after that we are going to choose only the albums released between "1990/01/01" and "1999/12/31" (90's). The "abstracts" must have more than 200 characters and after that we group the results by the columns of our interest (the albums information plus the "genre_name") and organizing it by descending order it by the number of "sales" (since we want the album with the most sales). In the end we add a LIMIT 1 to only obtain the first row of our table.
The structure of Query 2 is the same as the first one - first we have the UPDATE, where we select the "albums" table and then update the "sales" of an album to 0. The second part is the SELECT, in which we are only going to choose the albums released between "2000/01/01" and "2010/12/31" (first decade of 2000). The running time of the album needs to be superior to 45 minutes and ordered by the number of sales (since we want the album with the most sales). In the end we add a LIMIT 1 to only obtain the first row of our table.
As you can see in Figure 4 and 5 we started by doing only the SELECT to show which data we are going to UPDATE and then the results can be seen in the Figures 7 and 8.

| band_id | release_date | sales | genres | abstract |
| --- | --- | --- | --- | --- |
| 6171 | 1994-10-24 | 9779 | Math rock | At Action Park is the first full-length record by Shellac, released in 1994. The title is unrelated to the infamous New Jersey theme park, Action Park, which closed in 1996 due to numerous fatalities. The drummer, Todd Trainer, came up with the title of the fictional park because it sounded cool. |

**Figure 4** - Results of the SQL Query 1 before the update.

| band_id | sales | release_date | running_time |
| --- | --- | --- | --- |
| 464 | 12410000 | 2000-11-13 | 76.933334 |

**Figure 5** - Results of the SQL Query 2 before the update.

**A**

```
UPDATE albums
SET release_date = '1980/01/01'
WHERE albums.release_date = (SELECT albums.release_date
FROM (((albums
INNER JOIN bands ON albums.band_id = bands.band_id)
INNER JOIN bands_genre ON bands.band_id = bands_genre.band_id)
INNER JOIN genres ON bands_genre.genre_id = genres.genre_id)
      WHERE genres.genre_name = 'Math rock'
      AND albums.release_date >= '1990/01/01'
      AND albums.release_date <= '1999/12/31'
      AND LENGTH(albums.abstract) > 200
      ORDER BY albums.sales
      DESC
      LIMIT 1
);
```

**B**

```
UPDATE albums
SET sales = 0
WHERE albums.sales = (SELECT A.sales
                      FROM albums AS A
                      WHERE A.running_time > '45'
                      AND A.release_date >= '2000/01/01'
                      AND A.release_date <= '2010/12/31'
                      ORDER BY A.sales
                      DESC
                      LIMIT 1
);
```

**Figure 6** - **(A)** SQL Query 1 Update to 1980 the release date of all albums for the genre Math Rock, which were released in the 90's with an abstract over 200 characters, and that had the most sales; (**B)** SQL Query 2 Update to 0 the sales from the album with the most sales in the first decade of the year 2000, and which the running time is longer than 45 minutes.

| band_id | release_date | sales | genre_name | abstract |
| --- | --- | --- | --- | --- |
| 6171 | 1980-01-01 | 9779 | Math rock | At Action Park is the first full-length record by Shellac, released in 1994. The title is unrelated to the infamous New Jersey theme park, Action Park, which closed in 1996 due to numerous fatalities. The drummer, Todd Trainer, came up with the title of the fictional park because it sounded cool. |

**Figure 7** - Results of the SQL Query 1 after the update.

| band_id | sales | release_date | running_time |
| --- | --- | --- | --- |
| 1664 | 0 | 2002-06-25 | 46.1 |

**Figure 8** - Results of the SQL Query 2 after the update.

## 3.2 - NoSQL Queries and Results

For the querying in the MongoDB database it was used the "Mongosh" command line in the "MongoDB Compass" DBMS. Instead of using the "MongoDB's Aggregation Pipeline" ("db.collection.aggregate()") to filter the pretended data in the database, we used the MongoDB's function "db.collection.find()", this decision was made due to the fact that we had simple queries and the "db.collection.find()" could handle this task more easily than the "db.collection.aggregate()" function, however the results are presented in a JSON format, rather than a tabular format. Figures 9(A) and Figure 9(B) presents the MongoDB script used to perform the Queries 1 and 2, respectively and the results are presented in Figure 9(C) for Query 1 and in Figure 9(D) for Query 2. Thus, we proceed to update the data using the "ObjectId", obtained in the previous "db.collection.find()" query, and select the field that we want to update for Query 1 and 2. For Query 1 we update (Figure 10(A)) the album "release_date", and for Query 2 we update (Figure 10(B)) the album "sales", taking into account the criteria of our questions, like we have done for the SQL queries. After the update, we check again both album documents that have suffered the update with the function "db.collection.find()" using the "ObjectId" to infer the requested modification. Figure 11(A) shows the script and the result of the Query 1, in the other side Figure 11(B) represents the script used and the result of Query 2.

A

```
db.albums.find({
        $and :
             [
             {genres: "Math rock"},
             {release_date: {$gte: ISODate("1990-01-01T00:00:00.000+00:00")}},
             {release_date: {$lte: ISODate("1999-12-31T00:00:00.000+00:00")}},
             {abstract: /^[\s\S]{200,}$/ }
             ]
        }).sort({sales: -1}).limit(1)
```

B

```
db.albums.find({
          $and :
               [
               {running_time: {$gte: 45}},
               {release_date: {$gte: ISODate("2000-01-01T00:00:00.000+00:00")}},
               {release_date: {$lte: ISODate("2010-12-31T00:00:00.000+00:00")}}
               ]
          }).sort({sales: -1}).limit(1)
```

C

```
{ _id: ObjectId("6168bd6e416ed4db14e46417"),
  band_url: 'http://dbpedia.org/resource/Shellac_(band)',
  album_name: 'At Action Park',
  sales: 9779,
  running_time: 37.05,
  release_date: 1994-10-24T00:00:00.000Z,
  abstract: 'At Action Park is the first full-length record by Shellac, released in 1994.
           The title is unrelated to the infamous New Jersey theme park, Action Park,
           which closed in 1996 due to numerous fatalities. The drummer, Todd Trainer,
           came up with the title of the fictional park because it sounded cool.',
  bands: 'Shellac (band)',
  genres: [ 'Noise rock', 'Math rock', 'Post-hardcore' ],
  artists:
   [ { artist_url: 'http://dbpedia.org/resource/Todd_Trainer',
       artistic_name: [ 'Todd Trainer' ],
       is_active: 'True' },
     { artist_url: 'http://dbpedia.org/resource/Bob_Weston',
       artistic_name: [ 'Bob Weston' ],
       is_active: 'True' },
     { artist_url: 'http://dbpedia.org/resource/Steve_Albini',
       artistic_name: [ 'Steve Albini' ],
       is_active: 'True' } ] }
```

D

```
{ _id: ObjectId("6168b8f6416ed4db14e40d38"),
  band_url: 'http://dbpedia.org/resource/The_Beatles',
  album_name: '1 (Beatles album)',
  sales: 12410000,
  running_time: 79.13333333,
  release_date: 2000-11-13T00:00:00.000Z,
  abstract: '1 is a compilation album by English rock band the Beatles, originally
           released on 13 November 2000. The album features virtually every number-one
           single the band achieved in the United Kingdom and United States from 1962 to 1970.
           Issued on the 30th anniversary of the band\'s break-up, it was their first compilation
           available on only one compact disc. 1 was a commercial success and topped the charts worldwide.
           It has sold over 31 million copies. In addition, 1 is the fourth best-selling album in the US
           since Nielsen SoundScan began tracking US album sales in January 1991 and the best-selling album
           of the decade (2000 to 2009) in the US, as well as the best-selling album of the decade worldwide.
           1 was remastered and reissued in September 2011, and was remixed and reissued again in several
           different deluxe editions in November 2015, the most comprehensive of which is a three-disc set
           entitled 1+, which includes video discs of the band\'s music videos. As of June 2015, 1 is the
           sixth best-selling album of the 21st century in the UK, having sold over 3.1 million copies.',
  bands: 'The Beatles',
  genres: [ 'Pop music', 'Rock music' ],
  artists:
   [ { artist_url: 'http://dbpedia.org/resource/George_Harrison',
       artistic_name: [ 'George Harrison' ],
       is_active: 'False' },
     { artist_url: 'http://dbpedia.org/resource/John_Lennon',
       artistic_name: [ 'John Lennon' ],
       is_active: 'False' },
     { artist_url: 'http://dbpedia.org/resource/Ringo_Starr',
       artistic_name: [ 'Ringo Starr' ],
       is_active: 'False' },
     { artist_url: 'http://dbpedia.org/resource/Paul_McCartney',
       artistic_name: [ 'Paul McCartney' ],
       is_active: 'False' },
     { artist_url: 'http://dbpedia.org/resource/Paul_McCartney',
       artistic_name: [ 'Sir Paul McCartney' ],
       is_active: 'False' } ] }
```

**Figure 9** - NoSQL Queries and results before the update; **(A)** Query 1 before the update **(B)** Query 2 before the update **( C)** Results of the Query 1 before the update **(D)** Results of the Query 2 before the update.

A

```
> db.albums.updateOne({_id: ObjectId("6168bd6e416ed4db14e46417")},
                              {$set: {release_date: ISODate("1980-01-01T00:00:00.000+00:00")}}
                              )
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
```

B

```
> db.albums.updateOne({_id: ObjectId("6168b8f6416ed4db14e40d38")},
                              {$set: {sales: 0 }}
                              )
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
```

**Figure 10** - NoSQL Update Queries; **(A)** Query Update 1 **(B)** Query Update 2.

A

```
db.albums.find({_id: ObjectId("6168bd6e416ed4db14e46417")})


{ _id: ObjectId("6168bd6e416ed4db14e46417"),
  band_url: 'http://dbpedia.org/resource/Shellac_(band)',
  album_name: 'At Action Park',
  sales: 9779,
  running_time: 37.05,
  release_date: 1980-01-01T00:00:00.000Z,
  abstract: 'At Action Park is the first full-length record by Shellac,
            released in 1994. The title is unrelated to the infamous
            New Jersey theme park, Action Park, which closed in 1996 due
            to numerous fatalities. The drummer, Todd Trainer, came up with
            the title of the fictional park because it sounded cool.',
  bands: 'Shellac (band)',
  genres: [ 'Noise rock', 'Math rock', 'Post-hardcore' ],
  artists:
   [ { artist_url: 'http://dbpedia.org/resource/Todd_Trainer',
       artistic_name: [ 'Todd Trainer' ],
       is_active: 'True' },
     { artist_url: 'http://dbpedia.org/resource/Bob_Weston',
       artistic_name: [ 'Bob Weston' ],
       is_active: 'True' },
     { artist_url: 'http://dbpedia.org/resource/Steve_Albini',
       artistic_name: [ 'Steve Albini' ],
       is_active: 'True' } ] }
```

B

```
db.albums.find({_id: ObjectId("6168b8f6416ed4db14e40d38")})

{ _id: ObjectId("6168b8f6416ed4db14e40d38"),
  band_url: 'http://dbpedia.org/resource/The_Beatles',
  album_name: '1 (Beatles album)',
  sales: 0,
  running_time: 79.13333333,
  release_date: 2000-11-13T00:00:00.000Z,
  abstract: '1 is a compilation album by English rock band the Beatles,
                       originally released on 13 November 2000. The album features
                       virtually every number-one single the band achieved in the
                       United Kingdom and United States from 1962 to 1970.
                       Issued on the 30th anniversary of the band\'s break-up, it was
                       their first compilation available on only one compact disc.
                       1 was a commercial success and topped the charts worldwide.
                       It has sold over 31 million copies. In addition, 1 is the fourth
                       best-selling album in the US since Nielsen SoundScan began tracking
                       US album sales in January 1991 and the best-selling album of the
                       decade (2000 to 2009) in the US, as well as the best-selling album
                       of the decade worldwide. 1 was remastered and reissued in September 2011,
                       and was remixed and reissued again in several different deluxe editions
                       in November 2015, the most comprehensive of which is a three-disc set
                       entitled 1+, which includes video discs of the band\'s music videos.
                       As of June 2015, 1 is the sixth best-selling album of the 21st century
                       in the UK, having sold over 3.1 million copies.',
  bands: 'The Beatles',
  genres: [ 'Pop music', 'Rock music' ],
  artists:
   [ { artist_url: 'http://dbpedia.org/resource/George_Harrison',
       artistic_name: [ 'George Harrison' ],
       is_active: 'False' },
     { artist_url: 'http://dbpedia.org/resource/John_Lennon',
       artistic_name: [ 'John Lennon' ],
       is_active: 'False' },
     { artist_url: 'http://dbpedia.org/resource/Ringo_Starr',
       artistic_name: [ 'Ringo Starr' ],
       is_active: 'False' },
     { artist_url: 'http://dbpedia.org/resource/Paul_McCartney',
       artistic_name: [ 'Paul McCartney' ],
       is_active: 'False' },
     { artist_url: 'http://dbpedia.org/resource/Paul_McCartney',
       artistic_name: [ 'Sir Paul McCartney' ],
       is_active: 'False' } ] }
```

Figure 11 - NoSQL Queries and results after the update; **(A)** Query 1 and results after the update **(B)** Query 2 and results after the update.

## 4 - Conclusion

In general, SQL and NoSQL databases have advantages and disadvantages depending on the purpose of our application. For our database design, both models performed well. It is common to use NoSQL databases when there is a need to speed up the pace of the application development, to querying faster and to have structures based on our needs and not on relationships (schemaless), making it easier to access a single piece of data. Thus, if we have to perform a query using SQL that needs various inner joins to implement, that could compromise the query performance due to the complexity in terms of relationships between entities. MongoDB querying language makes the process of accessing a data chunk easier because it is a document database.

In our case, the use of a NoSQL database is not justified, since we have a small set of entities and relationships, and regarding our queries we have a unique query that has the need to apply INNER JOINS (Query 1) in SQL - which was easy; the Query 2 was quite simple because there was no need to use INNER JOIN. Querying within MongoDB was quite difficult, because it was not a declarative syntax and is more targeted to developers who already have the expertise. Debugging turns into a difficult task, due to the excessive use of parentheses and brackets.

In conclusion, considering the data curation and the database population processes, SQL proved to be more efficient, intuitive and easier than MongoDB.

# 5 - References

[1] Raghu Ramakrishnan e Johannes Gehrke, Database Management Systems , McGraw Hill, 3ª edição, 2003, ISBN 0072465638.

[2] Sadalage , P. J., Fowler, M. ( NoSQL distilled: a brief guide to the emerging world of polyglot persistence . Pearson.

[3] MongoDB. *MongoDB Documentation.* https://docs.mongodb.com/

[4] PostgreSQL.*Documentation.* https://www.postgresql.org/docs/

[5] PyMongo.*Documentation.* https://pymongo.readthedocs.io/en/stable/

[6] MongoDB Compass. https://www.mongodb.com/products/compass

# 6- Appendices

## 6.1 - Data curation script

```
import pandas
import numpy as np

# import import files

bands = pandas.read_csv("band-band_name_processed.csv", sep=',',
                        header=0, engine='python', encoding='utf8')

bands_genre = pandas.read_csv("band-genre_name_processed.csv", sep=',',
                              header=None, engine='python', encoding='utf8')

albums = pandas.read_csv("band-album_data_processed.csv",
                         sep=',', header=None, engine='python')

former_member = pandas.read_csv("band-former_member-member_name_processed.csv", sep=',',
                                header=None, engine='python', encoding='utf8')

member = pandas.read_csv("band-member-member_name_processed.csv", sep=',',
                         header=None, engine='python', encoding='utf8')


# adding index(id) to each dataframe
albums_index = albums.index.values
albums.insert(0, column="band_id", value=albums_index)

bands_index = bands.index.values
bands.insert(0, column="album_id", value=bands_index)


# adding column names
bands.columns = ["band_id", "band_url", "band_name"]
bands_genre.columns = ["band_url", "genre"]
albums.columns = ["album_id", "band_url", "album_name",
                  "release_date", "abstract", "running_time", "sales"]
former_member.columns = ["band_url", "artist_url", "artistic_name"]
member.columns = ["band_url", "artist_url", "artistic_name"]

# adding isActive column to members related table
former_member['is_active'] = False
member['is_active'] = True
```

```
# creating the necessary members dataframes
all_members = [former_member, member]
all_members = pandas.concat(all_members)
all_members_band = all_members.reset_index(drop=True)


all_members_url_only = all_members_band[["artist_url"]]
all_members_url_only = all_members_url_only["artist_url"].unique()
all_members_url_only = pandas.DataFrame(all_members_url_only)
all_members_url_only_index = all_members_url_only.index.values
all_members_url_only.insert(0, column="member_id",
                            value=all_members_url_only_index)
all_members_url_only.columns = ["member_id", "artist_url"]

all_members_name = all_members_band[["artistic_name", "artist_url"]]
all_members_name = pandas.DataFrame(all_members_name)
all_members_name_index = all_members_name.index.values
all_members_name.insert(0, column="name_id", value=all_members_name_index)
all_members_name.columns = ["name_id", "artistic_name", "artist_url"]


# creating genre dataframe
genre = bands_genre["genre"].unique()
genre_dataframe = pandas.DataFrame(genre)
genre_index = genre_dataframe.index.values
genre_dataframe.insert(0, column="genre_id", value=genre_index)
genre_dataframe.columns = ["genre_id", "genre_name"]


# merging dataframes
genre_band_merge_genre = pandas.merge(
    bands_genre, genre_dataframe, left_on='genre', right_on='genre_name', how='left')

genre_merged_merge_bands = pandas.merge(
    bands, genre_band_merge_genre, left_on='band_url', right_on='band_url', how='left')

genre_band_dataframe = genre_merged_merge_bands[[
    'band_id', 'genre_id']].astype('Int64').dropna()


albums_merge_band = pandas.merge(
    bands, albums, left_on='band_url', right_on='band_url', how='left').dropna()


albums_merge_band_mong = albums_merge_band[[
    "album_id", "band_url", "album_name", "sales", "running_time", "release_date", "abstract"]]

albums_merge_band = albums_merge_band[[
    "album_id", "band_id", "album_name", "sales", "running_time", "release_date", "abstract"]]


all_members_url_only_merge_all_members_name = pandas.merge(
    all_members_url_only, all_members_name, left_on='artist_url', right_on='artist_url', how='left')
all_members_url_only_merge_all_members_name = all_members_url_only_merge_all_members_name[[
    "name_id", "member_id", "artistic_name"]]


all_members_band_merge_band = pandas.merge(
    all_members_band, all_members_url_only, left_on='artist_url', right_on='artist_url', how='left')
all_members_band_merge_band_merge_bands = pandas.merge(
    all_members_band_merge_band, bands, left_on='band_url', right_on='band_url', how='left')

all_members_band_merge_band_merge_bands = all_members_band_merge_band_merge_bands[[
    "band_id", "member_id", "is_active"]]
all_members_band_merge_band_merge_bands = all_members_band_merge_band_merge_bands.drop_duplicates(
    ["band_id", "member_id", "is_active"])[["band_id", "member_id", "is_active"]]

# fix date time values
#albums_merge_band['sales'].str.replace(['.',','], '').astype(int)
albums_merge_band['release_date'] = pandas.to_datetime(
    albums_merge_band.release_date)
```

```
albums_merge_band['release_date'] = albums_merge_band['release_date'].dt.strftime(
    '%Y/%m/%d')


albums_merge_band_mong['release_date'] = pandas.to_datetime(
    albums_merge_band_mong.release_date)
albums_merge_band_mong['release_date'] = albums_merge_band_mong['release_date'].dt.strftime(
    '%Y/%m/%d')
print(albums_merge_band_mong)
# export data to files
bands.to_csv("D:/BDA project/export_files/bands.csv", sep="\t", index=False)
genre_dataframe.to_csv(
    "D:/BDA project/export_files/genre.csv", sep="\t", index=False)
genre_band_dataframe.to_csv(
    "D:/BDA project/export_files/bands_genre.csv", sep="\t", index=False)
albums_merge_band.to_csv(
    "D:/BDA project/export_files/albums.csv", sep="\t", index=False)
all_members_url_only.to_csv(
    "D:/BDA project/export_files/members.csv", sep="\t", index=False)
all_members_url_only_merge_all_members_name.to_csv(
    "D:/BDA project/export_files/artistic_names.csv", sep="\t", index=False)
all_members_band_merge_band_merge_bands.to_csv(
    "D:/BDA project/export_files/bands_members.csv", sep="\t", index=False)

all_members_band_mongo = all_members_band[["band_url", "artist_url", "artistic_name",
                                           "is_active"]]
all_members_band_mongo.to_csv(
    "D:/BDA project/export_files/mongodb_members.csv", sep="\t", index=False)

albums_merge_band_mong.to_csv(
    "D:/BDA project/export_files/albums.csv", sep=",", index=False)
```

## 6.2 - Code for the creation and insertion of tables in PostgreSQL

```
CREATE TABLE bands (
               band_id SERIAL,
               band_url VARCHAR(150) NOT NULL,
               band_name VARCHAR(70) NOT NULL,
        PRIMARY KEY(band_id),
        UNIQUE(band_url)
);

CREATE TABLE albums (
               album_id SERIAL,
               band_id INTEGER,
               album_name VARCHAR NOT NULL,
               sales INTEGER,
               running_time FLOAT(8) NOT NULL,
               release_date DATE NOT NULL,
               abstract TEXT,
        PRIMARY KEY(album_id),
        FOREIGN KEY(band_id) REFERENCES bands
);

CREATE TABLE members (
               member_id SERIAL,
               artist_url VARCHAR NOT NULL,
        PRIMARY KEY(member_id),
        UNIQUE (artist_url)
);


CREATE TABLE bands_members (
               band_id INTEGER,
```

```
                member_id INTEGER,
                is_active BOOLEAN NOT NULL,
        PRIMARY KEY(band_id,member_id,is_active),
        FOREIGN KEY(band_id) REFERENCES bands,
        FOREIGN KEY(member_id) REFERENCES members
);


CREATE TABLE bands_members (
                band_id INTEGER,
                member_id INTEGER,
                is_active BOOLEAN NOT NULL,
        PRIMARY KEY(band_id,member_id,is_active),
        FOREIGN KEY(band_id) REFERENCES bands,
        FOREIGN KEY(member_id) REFERENCES members
);


CREATE TABLE artistic_names (
                name_id SERIAL,
                member_id INTEGER,
                artistic_name VARCHAR(70) NOT NULL,
        PRIMARY KEY(name_id),
        FOREIGN KEY(member_id) REFERENCES members
);

CREATE TABLE genres (
                genre_id SERIAL,
                genre_name VARCHAR(70) NOT NULL,
        PRIMARY KEY(genre_id)
);

CREATE TABLE bands_genre (
                band_id INTEGER,
                genre_id INTEGER,
        PRIMARY KEY(band_id,genre_id),
        FOREIGN KEY (band_id) REFERENCES bands,
        FOREIGN KEY (genre_id) REFERENCES genres
);


-- Populate bands table
COPY "bands" FROM '~/bands_data/bands.csv' DELIMITER E'\t' CSV HEADER;

-- Populate genres table
COPY "genres" FROM '~/bands_data/genres.csv' DELIMITER E'\t' CSV HEADER;

-- Populate bands_genre table
COPY "bands_genre" FROM '~/bands_data/bands_genre.csv' DELIMITER E'\t' CSV HEADER;

-- Populate bands_genre table
COPY "bands_genre" FROM '~/bands_data/bands_genre.csv' DELIMITER E'\t' CSV HEADER;

-- Populate albums table
COPY "albums" FROM '~/bands_data/albums.csv' DELIMITER E'\t' CSV HEADER;

-- Populate members table
COPY "members" FROM '~/bands_data/members.csv' DELIMITER E'\t' CSV HEADER;

-- Populate bands_members table
COPY "bands_members" FROM '~/bands_data/bands_members.csv' DELIMITER E'\t' CSV HEADER;

-- Populate artistic_names table
COPY "artistic_names" FROM '~/bands_data/artistic_names.csv' DELIMITER E'\t' CSV HEADER;
```

## 6.3 - Script used to insert into MongoDB

```
# Import Packages
from pymongo import MongoClient
import certifi
from datetime import datetime


#___________________Def Functions__________________________________//

# get bands

def getBand(url):
    # add bands
    name = None
    with open("mongo_csv/bands_mongoDB.txt","r",encoding="utf-8") as file:
        for line in file:
            line = line.split("\t")
            band_url = line[0]
            band_name = line[1].strip('\n')
            if url == band_url:
                name = band_name

    # verify is the band associated with the band_url exist for the album's band,
    # otherwise the key for the bands list/array in the albums dictionary will not
    # be created, and the band list/array will not be appended to the albums dictionary.
    # return name if name else None



# get genres________________________________________________________________________________/

def getGenre(url):
    # add genres
    genres = []
    with open("mongo_csv/genres_mongoDB.csv","r",encoding="utf-8") as file:
        # add genres
        for line in file:
            line = line.split(",")
            band_url = line[0].strip('"')
            genre_name = line[1].strip('\"\n')
            if url == band_url:
                genres.append(genre_name)

    # verify is the genres associated with the band_url exist for the album's band,
    # otherwise the key for the genres list/array in the albums dictionary will not
    # be created, and the genres list/array will not be appended to the albums dictionary.


    return genres if genres else None



# get artists________________________________________________________________________________/

def getArtists(url):

    # add artists
    artists = []

    with open("mongo_csv/artists_mongoDB.csv","r",encoding="utf-8") as file:
        for line in file:

            dict_artist = {
                "artist_url": None,
                "artistic_name": list(),
                "is_active": None
                }
```

```
            line = line.split('\t')
            band_url = line[0]
            artist_url = line[1]
            artistic_name = line[2]
            is_active = line[3].strip('\n')

            if url == band_url:
                dict_artist["artist_url"] = artist_url
                dict_artist["is_active"] = is_active
                if artist_url == dict_artist["artist_url"]:
                    dict_artist["artistic_name"].append(artistic_name)

                else:
                    dict_artist["artistic_name"] = [artistic_name]
                    dict_artist["is_active"] = is_active
                artists.append(dict_artist)

  # verify is the artists associated with the band_url exist for the album's band,
  # otherwise the key for the artists dictionary in the albums dictionary will not be created,
  # and the artist dictionary will not be appended to the albums dictionary.

    return artists if artists else None





#___________________Albums Dictionary______________________________________//


# import albums data
albums_albums = open("mongo_csv/albums_mongoDB.txt","r",encoding="utf-8")

# open a list that will contain one dictionary for each band
albums = []

# open a set of unique band URLS
album_set = set()


# add albums. Create a dictionary for the albums colection
#i=0
for line in albums_albums:
    #if i<5  :
        line = line.split("\t")

        # transform realease date str into datetime
        year = int(line[4].split("-")[0])
        month = int(line[4].split("-")[1])
        day = int(line[4].split("-")[2])
        date_str = line[4]
        date = datetime.strptime(date_str,'%Y-%m-%d')

        band_url = line[0]
        album_name = line[1].strip('\"')
        sales = int(line[2])
        running_time = float(line[3])
        release_date = date
        abstract = line[5].strip('\"').strip('\"\n')


        # add new band URL to albums set and create a dictionary entry
        dict_album = {
            "band_url": band_url,
            "album_name": album_name,
            "sales": sales,
            "running_time": running_time,
            "release_date": release_date,
            "abstract": abstract,
            }
```

```
# in case of the inexistance of the bands, genres and artists list/array/dictionary associated # with
the band_url for an album,the list/array/dictionary will not be appended to the album  # dictionary,
aka # document,and the a key will not be created.

        # get bands
        bands = getBand(band_url)
        if bands:
            dict_album["bands"] = bands

        # get genre
        genres = getGenre(band_url)
        if genres:
            dict_album["genres"] = genres

        # get artists
        artists = getArtists(band_url)
        if artists:
            dict_album["artists"] = artists

        # add an album dictionary to albums list, for each band
        albums.append(dict_album)

        #i+=1

    #else:
        #break



#______________Connect to MongoDB_______________________________________________________//

# connect to Mongo Atlas DB (replace <password>)
ca = certifi.where()
cluster=
MongoClient("mongodb+srv://bandsDB:<password>@bandscluster.1eit9.mongodb.net/myFirstDatabase?retryWri
tes=true&w=majority",tlsCAFile=ca)

db = cluster["bandsDB"]
collection = db["albums"]


#_____Populate MongoDB with albums dictionaries___________________________//

for album in albums:
    collection.insert_one(album)
```