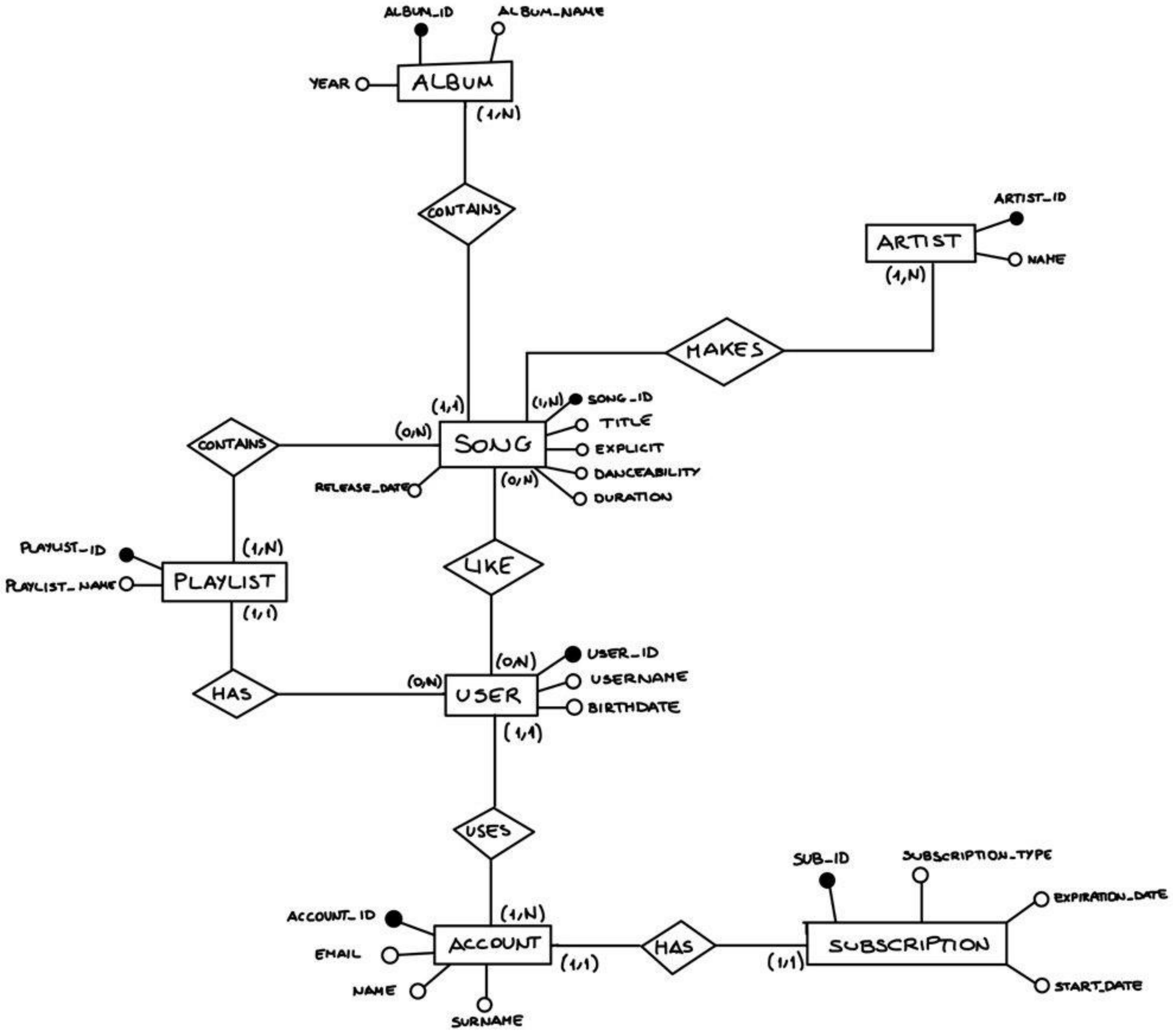
ADM PROJECT

Riccardo Gjini 4640527
Tommaso Parodi 4697321

1. Domain and application
   Our project it's a music streaming service where the user can choose the subscription type and listen to music, create playlists and like their favorite songs.
   Entities: Song, Album, Artist, Playlist, User, Account, Subscription
   Relations: Like, Dislike, User_has_Playlist, Artist_makes_song, User_has_Subscription, Album_contains_Song
   Typical workload:
   - Like or dislike a song
   - Create playlists and add songs into them
   - Check information of a song
   - Show the albums of an artist
   - Show the songs in an album
   - Show the playlist of a user
   - Show the songs in a playlist
   - Retrieve most like songs of an artist
2. The application we propose is read/write intensive and mainly uses points and range queries
3. Given the predominant nature of the queries, we would like to employ both hash-based and range-based partitioning. High availability is a priority, and strong consistency is not required, so we are looking to implement an asynchronous multi-leader replication scheme or a leaderless one.
4. We use a dataset containing song data, a dataset of users and a dataset containing account data.
5. Cassandra could be the best option since it provides high availability, guarantees eventual consistency, uses a leaderless replication protocol, and uses a hash-based partition scheme. It also supports indexes for efficient multi-point queries.

6. Conceptual schema



7. <u>Workload</u>

We define the workload as follows:

1. Given a user, find all the songs and the titles of the songs that the given user likes
2. Given a user find all his playlist and their name, and the number of songs in each one
3. Given an account find all user associated with it and their username
4. Given a playlist, find the name of the user who created it and the name of the songs contained
5. Given a song find all its information (album & artist)
6. Given a song find the average age of users who liked it
7. Given an artist find the title of his albums and the titles of the songs in each of them.
8. Given an artist find his 3 most liked songs by users
9. Given a playlist find the average danceability of its songs
10. Find the name and surname of the accounts whose subscription is expired
11. Given an artist find all his albums titles sorted by year
12. Given a playlist find the duration of it
13. Given an album find the artist, the title and the duration of each song in it
14. Given an account show email, subscription type and start date of the subscription
15. Given an artist and a user show the title of all the songs of this artist like by the user
16. Given a song and an account, show the name of all the playlist of this account that contain the song

8. <u>Aggregate Schema</u>

1. E: User
   LS: [User(user_id)_!]
   LP: [Song(song_id,title)_L]

2. E: User
   LS: [User(User_id)_!]
   LP: [Playlist(playlist_id, playlist_name)_H, Song(song_id)_CH]

3. E: Account
   LS: [Account(account_id)_!]
   LP: [User(user_id, username)_U]

4. E: Playlist
   LS: [Playlist(playlist_id)_!]
   LP: [User(user_id,username)_H, Song(song_id,title)_C]

5. E: Song
   LS: [Song(song_id)_!]
   LP: [Song_!, Album_C, Artist_M]

6. E: Song
   LS: [Song(song_id)_!]
   LP: [User(user_id,birthdate)_L]

7. E: Artist
   LS: [Artist(artist_id)_!]
   LP: [Album(album_id, album_name)_CM, Song(song_id, title)_M]

8. E: Artist
   LS: [Artist(artist_id)_!]
   LP: [Song(song_id)_M, User(user_id)_LM]

9. E: Playlist
   LS: [Playlist(Playlist_id)_!]
   LP: [Song(song_id, danceability)_C]

10. E: Subscription
    LS: [Subscription(expiration_date)_!]
    LP: [Account(account_id, name, surname)_H]

11. E: Artist
    LS: [Artist(artist_id)_!]
    LP: [Album(album_id, album_name, year)_CM]

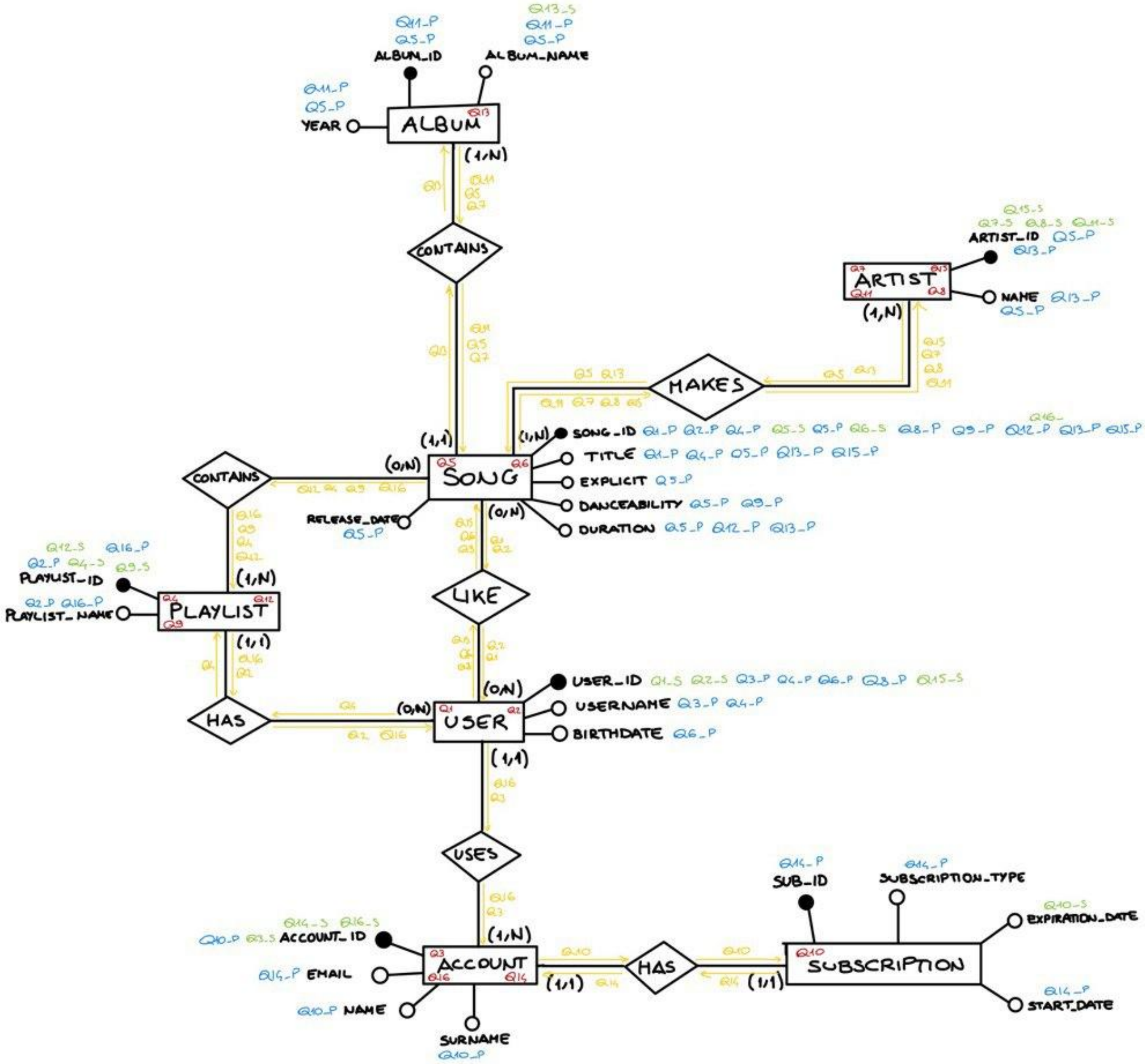
12. E: Playlist
    LS: [Playlist(playlist_id)_!]
    LP: [Song(song_id, duration)_C]

13. E: Album
    LS: [Album(album_id)_!]
    LP: [Artist(artist_id, name)_MC, Song(song_id, title, duration)_C]

14. E: Account
    LS: [Account(account_id)_!]
    LP: [Account(email)_!, Subscription(sub_id, subscription_type, start_date)_H]

15. E: Artist
    LS: [Artist(artist_id)_!, User(user_id)_LM]
    LP: [Song(song_id,title)_M]

16. E: Account
    LS: [Account(account_id)_!, Song(song_id)_CHU]
    LP: [Playlist(playlist_id, playlist_name)_HU]


Q11_P
Q5_P
ALBUM_ID
Q13_S
Q11_P
Q5_P
ALBUM_NAME
Q11_P
Q5_P
YEAR
ALBUM
Q13
(1/N)
Q11
Q11
Q5
Q7
CONTAINS
Q11
Q5
Q7
Q13
Q15_S
Q7_S Q8_S Q11_S
ARTIST_ID Q5_P
Q13_P
ARTIST
Q7
Q13
Q11
Q8
NAME Q13_P
Q5_P
(1,N)
Q15
Q7
Q8
Q11
Q5 Q13
MAKES
Q5 Q13
Q11 Q7 Q8 Q5
(1,1)
(1,N)
SONG_ID Q1_P Q2_P Q4_P Q5_S Q5_P Q6_S Q8_P Q9_P Q12_P Q13_P Q15_P
Q16_
TITLE Q1_P Q4_P Q5_P Q13_P Q15_P
EXPLICIT Q5_P
DANCEABILITY Q5_P Q9_P
DURATION Q5_P Q12_P Q13_P
CONTAINS
(0/N)
SONG
Q5
Q6
Q12 Q4 Q9 Q16
RELEASE_DATE
Q5_P
(0/N)
Q15
Q6
Q3
Q1
Q2
Q16
Q9
Q4
Q12
Q12_S Q16_P
Q2_P Q4_S Q9_S
PLAYLIST_ID
(1/N)
Q2_P Q16_P
PLAYLIST_NAME
PLAYLIST
Q4
Q12
Q9
(1/1)
Q4
Q16
Q2
LIKE
Q15
Q6
Q3
Q2
Q1
(0/N)
USER_ID Q1_S Q2_S Q3_P Q4_P Q6_P Q8_P Q15_S
USERNAME Q3_P Q4_P
BIRTHDATE Q6_P
HAS
Q4
(0/N)
USER
Q1
Q2
Q2 Q16
(1/1)
Q16
Q3
USES
Q16
Q3
Q14_S Q16_S
Q10_P Q3_S ACCOUNT_ID
(1/N)
Q14_P EMAIL
ACCOUNT
Q3
Q16
Q14
Q10_P NAME
SURNAME
Q10_P
Q10
(1/1)
Q14
HAS
Q10
Q14
(1/1)
Q14_P
SUB_ID
Q14_P
SUBSCRIPTION_TYPE
Q10_S
EXPIRATION_DATE
SUBSCRIPTION
Q10
Q14_P
START_DATE

Song: { song_id, title, explicit, danceability, duration, release_date, album_id, album_name, year, madeByArtists:[{ artist_id, artist_name }], users:[{ user_id, birthdate }] }

User: {user_id, playlists: [{ playlist_id, playlist_name, songs: [{ song_id, title }]}], likedSongs: [{ song_id, title }] }

Account: {account_id, email, sub_id, start_date, subscription_type, users: [{ user_id, username, playlists: [{ playlist_id, playlist_name, songs: [{ song_id, title }] }] }]}

Playlist: {playlist_id, user_id, username, songs:[{song_id, title, danceability, duration}]}

Album: {album_id, songs: [{ song_id, title, duration, createdByArtist: [{ artist_id, artist_name }] }] }

Artist: {artist_id, songs:[{song_id, title, album_id, album_name, year, users:[{user_id, birthdate}]}]}

Subscription: {account_id, expiration_date, name, surname}

9. Logical Schema

In the first part of our project, we initially considered using Cassandra for our application because of its high availability features.

However, we encountered challenges when dealing with our workload, which involves a large number of range queries and complex attribute. Unfortunately, performing these queries in Cassandra was either difficult or sometimes not even possible.

Due to these limitations, we made the decision to change the choice and work with MongoDB. This transition allows us to use a smaller number of aggregates, reducing redundancy and enabling easier execution of certain queries.

Although MongoDB is primarily designed to be CP by default, we have the flexibility to adjust its availability at the expense of consistency by modifying certain system parameters according to our specific requirements, which is something that we will show next on the report.

To ensure data integrity, we have assigned a specific data type to each attribute and made every attribute of every collection mandatory.

Here below we provide the logical schema used in MongoDB:

Account:

```
db.createCollection("accountss",{
   validator: {
```

```
$jsonSchema:{
   bsonType: "object",
   required:
   ["account_id", "email", "sub_id", "start_date", "subscription_type", "users"],
   properties: {
      account_id: {
         bsonType: "int"
      },
      email: {
         bsonType: "string"
      },
      sub_id: {
         bsonType: "int"
      },
      start_date: {
         bsonType: "date"
      },
      subscription_type: {
         bsonType: "string"
      },
      users:{
         bsonType: [ "array" ],
         items: {
            bsonType: "object",
            required:
            ["user_id", "username","playlists"],
            properties: {
               user_id: {
                  bsonType: "int"
               },
               username: {
                  bsonType: "string"
               },
               playlists: {
                  bsonType: [ "array" ],
                  items: {
                     bsonType: "object",
                     required: ["playlist_id", "playlist_name", "songs"],
                     properties: {
                        playlist_id: {
                           bsonType: "int"
                        },
                        playlist_name: {
                           bsonType: "string"
                        },
                        songs: {
                           bsonType: [ "array" ],
                           items: {
                              bsonType: "object",
                              required: ["song_id", "title"],
                              properties: {
                                 song_id: {
```

```
                                                        bsonType: "int"
                                                    },
                                                    title: {
                                                        bsonType: "string"
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
})
```

Album:

```
db.createCollection("albums",{
    validator: {
        $jsonSchema: {
            bsonType: "object",
            required:
            ["album_id", "songs"],
            properties: {
                album_id: {
                    bsonType: "int"
                },
                songs: {
                    bsonType: [ "array" ],
                    items: {
                        bsonType: "object",
                        required: ["song_id", "title", "duration", "createdByArtists"],
                        properties: {
                            song_id: {
                                bsonType: "int"
                            },
                            title: {
                                bsonType: "string"
                            },
                            duration: {
                                bsonType: "int"
                            },
                            createdByArtists: {
                                bsonType: [ "array" ],
                                items: {
                                    bsonType: "object",
                                    required: ["artist_id", "artist_name"],
                                    properties: {
                                        artist_id: {
                                            bsonType: "int"
                                        },
                                        artist_name: {
                                            bsonType: "string"
                                        }
                                    }
                                }

                            }
                        }
                    }
                }
            }
        }
    }
})
```

Artist:

```
db.createCollection("artists",{
    validator: {
        $jsonSchema: {
            bsonType: "object",
            required:
            ["artist_id", "songs"],
            properties: {
                album_id: {
                    bsonType: "int"
                },
                songs: {
                    bsonType: [ "array" ],
                    items: {
                        bsonType: "object",
                        required: ["song_id", "title", "album_id", "album_name", "year", "users"],
                        properties: {
                            song_id: {
                                bsonType: "int"
                            },
                            title: {
                                bsonType: "string"
                            },
                            album_id: {
                                bsonType: "int"
                            },
                            album_name: {
                                bsonType: "string"
                            },
                            year: {
                                bsonType: "int"
                            },
                            users: {
                                bsonType: [ "array" ],
                                items: {
                                    bsonType: "object",
                                    required: ["user_id", "birthdate"],
                                    properties: {
                                        user_id: {
                                            bsonType: "int"
                                        },
                                        birthdate: {
                                            bsonType: "date"
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
})
```

Playlist:

```
db.createCollection("playlists",{
    validator: {
        $jsonSchema: {
            bsonType: "object",
            required:
            ["playlist_id", "user_id", "username", "songs"],
            properties: {
                playlist_id: {
                    bsonType: "int"
                },
                user_id: {
                    bsonType: "int"
                },
                username: {
                    bsonType: "string"
                },
                songs: {
                    bsonType: [ "array" ],
                    items: {
                        bsonType: "object",
                        required: ["song_id", "title", "danceability", "duration"],
                        properties: {
                            song_id: {
                                bsonType: "int"
                            },
                            title: {
                                bsonType: "string"
                            },
                            danceability: {
                                bsonType: "float"
                            },
                            duration: {
                                bsonType: "int"
                            }
                        }
                    }
                }
            }
        }
    }
})
```

User:

```
db.createCollection("users",{
    validator: {
        $jsonSchema: {
            bsonType: "object",
            required:
            ["user_id", "playlists", "likedSongs"],
            properties: {
                user_id: {
                    bsonType: "int"
                },
                playlists: {
                    bsonType: [ "array" ],
                    items: {
                        bsonType: "object",
                        required: ["playlist_id", "playlist_name", "songs"],
                        properties: {
                            playlist_id: {
                                bsonType: "int"
                            },
                            playlist_name: {
                                bsonType: "string"
                            },
                            songs: {
                                bsonType: [ "array" ],
                                items: {
                                    bsonType: "object",
                                    required: ["song_id", "title"],
                                    properties: {
                                        song_id: {
                                            bsonType: "int"
                                        },
                                        title: {
                                            bsonType: "string"
                                        }
                                    }
                                }
                            }
                        }
                    }
                },
                likedSongs: {
                    bsonType: [ "array" ],
                    items: {
                        bsonType: "object",
                        required: ["song_id", "title"],
                        properties: {
                            song_id: {
                                bsonType: "int"
                            },
                            title: {
                                bsonType: "string"
                            }
                        }
                    }
                }
            }
        }
    }
})
```

Song:

```
db.createCollection("songs",{
    validator: {
        $jsonSchema: {
            bsonType: "object",
            required: ["song_id", "title", "explicit", "danceability", "duration", "release_date",
"album_id", "album_name", "year", "madeByArtists", "users"],
            properties: {
                song_id: {
                    bsonType: "int"
                },
                title: {
                    bsonType: "string"
                },
                explicit: {
                    bsonType: "bool"
                },
                danceability: {
                    bsonType: "float"
                },
                duration: {
                    bsonType: "int"
                },
                release_date: {
                    bsonType: "date"
                },
                album_id: {
                    bsonType: "int"
                },
                album_name: {
                    bsonType: "string"
                },
                year: {
                    bsonType: "int"
                },
                madeByArtists: {
                    bsonType: [ "array" ],
                    items: {
                        bsonType: "object",
                        required: ["artist_id", "artist_name"],
                        properties: {
                            artist_id: {
                                bsonType: "int"
                            },
                            artist_name: {
                                bsonType: "string"
                            }
                        }
                    }
                },
                users: {
                    bsonType: [ "array" ],
                    items: {
                        bsonType: "object",
                        required: ["user_id", "birthdate"],
                        properties: {
                            user_id: {
                                bsonType: "int"
                            },
                            birthdate: {
                                bsonType: "date"
                            }
                        }
                    }
                }
            }
        }
    }
})
```

Subscription:

```
db.createCollection("subscriptions", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         required:
         ["expiration_date", "account_id", "name", "surname"],
         properties: {
            expiration_date: {
               bsonType: "date"
            },
            account_id: {
               bsonType: "int"
            },
            name: {
               bsonType: "string"
            },
            surname: {
               bsonType: "string"
            }
         }
      }
   }
})
```

10. Implementation of the workload in MongoDB

1. Given a user, find all the songs and the titles of the songs that the given user likes

   ```
   db.users.aggregate([
     { $match: { user_id: 42 } },
     { $unwind: "$likedSongs" },
     { $project: { _id: 0, song_id: "$likedSongs.song_id", title: "$likedSongs.title"}}
   ])
   ```

2. Given a user find all his playlist and their name, and the number of songs in each one

   ```
   db.users.aggregate([
      { $match: { user_id: 5 } },
      { $unwind: "$playlists" },
      { $project: { _id: 0, playlist_id: "$playlists.playlist_id", playlist_name:
   "$playlists.playlist_name", num_songs: { $size: "$playlists.songs" } } }
   ])
   ```

3. Given an account find all user associated with it and their username

   ```
   db.accountss.aggregate([
     { $match: { "account_id": 10 } },
     { $unwind: "$users" },
     { $project: { "_id": 0,"user_id": "$users.user_id", "username":
   "$users.username" } }
   ])
   ```

4. Given a playlist, find the name of the user who created it and the name of the songs contained

   ```
   db.playlists.aggregate([
     { $match: { "playlist_id": 19 } },
     { $project: { "_id": 0, "username": "$username", "song_names":
   "$songs.title" }}
   ])
   ```

5. Given a song find all its information (album & artist)

   ```
   db.songs.find({ "song_id": 1 })
   ```

6. Given a song find the average age of users who liked it

   ```
   db.songs.aggregate([
     { $match: { "song_id": 8 } },
     { $unwind: "$users" },
     {
       $group: {
         "_id": 0,
         "average_age": {
           $avg: {
             $divide: [
               {
                 $subtract: [new Date(), "$users.birthdate"]
               },
               1000 * 60 * 60 * 24 * 365.25
             ]
           }
         }
       }
     },
     {
       $project: {
         "_id": 0,
         "average_age": { $floor: "$average_age" }
       }
   ```

```
  }])
```

7. Given an artist find the title of his albums and the titles of the songs in each of them.

```
db.artists.aggregate([
  { $match: { "artist_id": 50} },
  { $unwind: "$songs" },
  { $project: { "_id": 0, "album_title": "$songs.album_name", "song_titles":
"$songs.title"   }}
])
```

8. Given an artist find his 3 most liked songs by users

```
db.artists.aggregate([
  { $match: { artist_id: 50 } },
  { $unwind: "$songs" },
  { $addFields: {
    likedCount: { $size: "$songs.users" }
  } },
  { $sort: { likedCount: -1 } },
  { $limit: 3 },
  { $project: {
    _id: 0,
    song_id: "$songs.song_id",
    title: "$songs.title",
    likedCount: 1
  } }
])
```

9. Given a playlist find the average danceability of its songs

```
db.playlists.aggregate([
 { $match: { "playlist_id": 71 } },
 { $unwind: "$songs" },
 { $group: {
     _id: null,
     average_danceability: { $avg: "$songs.danceability" }
   }
 },
 { $project: {
     _id: 0,
     average_danceability: { $round: ["$average_danceability", 2] }
   }
 }
])
```

10. Find the name and surname of the accounts whose subscription is expired

```
db.subscriptions.aggregate([
  {
    $match: {
      expiration_date: { $lt: new Date() }
    }
  },
  {
    $project: {
      _id: 0,
      name: "$name",
      surname: "$surname"
    }
  }
])
```

11. Given an artist find all his albums titles sorted by year

```
db.artists.aggregate([
  { $match: { artist_id: 33 } },
  { $unwind: "$songs" },
  { $group: {
      _id: "$songs.album_id",
      album_title: { $first: "$songs.album_name" },
      year: { $first: "$songs.year" } } },
  { $sort: { year: 1 } },
  { $project: {
      _id: 0,
      album_title: 1
    }
  }
 ])
```

12. Given a playlist find the duration of it

```
db.playlists.aggregate([
  { $match: { "playlist_id": 30 } },
  { $unwind: "$songs" },
  { $group: {
      _id: "$_id",
      duration: { $sum: "$songs.duration" }
    }
  },
  { $project: {
      _id: 0,
      durationInMinutes: { $divide: ["$duration", 1000 * 60] }
  }}])
```

13. Given an album find the artist, the title and the duration of each song in it

```
db.albums.aggregate([
  { $match: { "album_id": 47 } },
  { $unwind: "$songs" },
  {
    $project: {
      _id: 0,
      artist: "$songs.createdByArtists.artist_name",
      song: "$songs.title",
      duration: {
        $round: [{ $divide: ["$songs.duration", 60000] },2]
      }
    }
  }
])
```

14. Given an account show email, subscription type and start date of the subscription

```
db.accountss.aggregate([
 { $match: { "account_id": 24 } },
 { $project: {
   "_id": 0,
   "email": "$email",
   "subscription_type": "$subscription_type",
   "start_date": "$start_date"
 }}
])
```

15. Given an artist and a user show the title of all the songs of this artist like by the user

```
db.artists.aggregate([
  { $match: { "artist_id": 1 } },
  { $unwind: "$songs" },
  { $match: { "songs.users.user_id": 3756 } },
  { $project: {
    "_id": 0,
    "song_title": "$songs.title"
  }}
 ])
```

16. Given a song and an account, show the name of all the playlist of this account that contain the song

```
db.accountss.aggregate([
 { $match: {
    "users.playlists.songs.song_id": 357,
    account_id: 1
   }},
 { $unwind: "$users"},
 { $unwind: "$users.playlists" },
 { $match: {
    "users.playlists.songs.song_id": 357}},
 { $project: {
    _id: 0,
    playlist_name: "$users.playlists.playlist_name"
   }}
])
```

11. System configuration - Concerning Indexing and Sharding

We sharded collections as follows:

1. User

   ```
   db.users.createIndex( {user_id: 1}, {unique: true} )
   ```
   This index improves the performance of queries 1, 2. It also ensures the uniqueness of the user_id attribute.

   ```
   sh.shardCollection("user4_db.users", {user_id: 1}
   ```

2. Songs

   ```
   db.songs.createIndex( {song_id: 1}, {unique: true} )
   ```
   This index improves the performance of queries 5, 6. It also ensures the uniqueness of the song_id attribute.

   ```
   sh.shardCollection("user4_db.songs", {song_id: 1 })
   ```

3. Account

   ```
   db.accountss.createIndex( {account_id: 1}, {unique: true} )
   ```
   This index improves the performance of queries 3, 14, 16. It also ensures the uniqueness of the account_id attribute.

   ```
   sh.shardCollection("user4_db.accountss", {account_id: 1 })
   ```

4. Album

   ```
   db.albums.createIndex( {album_id: 1}, {unique: true} )
   ```

   This index improves the performance of query 13. It also ensures the uniqueness of the album_id attribute.

   ```
   sh.shardCollection("user4_db.albums", {album_id: 1 })
   ```

5. Artist

   ```
   db.artists.createIndex( {artist_id: 1}, {unique: true} )
   ```

   This index improves the performance of queries 7, 8, 11, 15. It also ensures the uniqueness of the artist_id attribute.

   ```
   sh.shardCollection("user4_db.artists", {artist_id: 1 })
   ```

6. Subscription

   ```
   db.subscriptions.createIndex( {expiration_date: 1} )
   ```

   This index improves the performance of query 10.

   ```
   db.subscriptions.createIndex( {account_id: 1}, {unique: true} )
   ```

   This index ensures the uniqueness of the account_id attribute.

   ```
   sh.shardCollection("user4_db.subscriptions", {account_id: 1 })
   ```

7. Playlist

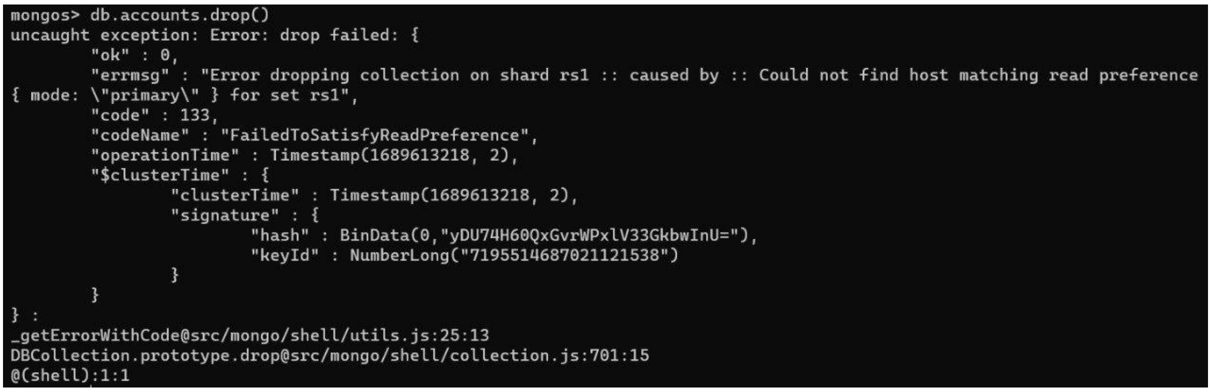
   ```
   db.playlists.createIndex( {playlist_id: 1}, {unique: true} )
   ```

   This index improves the performance of queries 4, 9, 12. It also ensures the uniqueness of the playlist_id attribute.

   ```
   sh.shardCollection("user4_db.playlists", {playlist_id: 1 })
   ```

12. Issues with dropping collections

We found some issues in dropping the collections created: user, song, subscription, album, artist, account, playlist.
This was the issue:



```
mongos> db.accounts.drop()
uncaught exception: Error: drop failed: {
        "ok" : 0,
        "errmsg" : "Error dropping collection on shard rs1 :: caused by :: Could not find host matching read preference
{ mode: \"primary\" } for set rs1",
        "code" : 133,
        "codeName" : "FailedToSatisfyReadPreference",
        "operationTime" : Timestamp(1689613218, 2),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1689613218, 2),
                "signature" : {
                        "hash" : BinData(0,"yDU74H60QxGvrWPxlV33GkbwInU="),
                        "keyId" : NumberLong("7195514687021121538")
                }
        }
} :
_getErrorWithCode@src/mongo/shell/utils.js:25:13
DBCollection.prototype.drop@src/mongo/shell/collection.js:701:15
@(shell):1:1
```

Since we didn’t manage to make it work, we created new collections and named them like the previous but making them plural.
So now we are using this collections: users, songs, subscriptions, albums, artists, accountss, playlists. Notice that account was created two times because we had a typo with the logical schema.

13. Tuning on MongoDB

As we said before, we want to tune our mongoDB and in order to do this we have to put this preferences:
We set “read preference” to “nearest” with the command:

```
db.getMongo().setReadPref(“nearest”)
```

This means that the read requests are completed fast by checking in a replica which is less than a specified latency threshold, not taking into account if it is primary or secondary.
We set “defaultReadConcern” assigned to “available” and “defaultWriteConcern” assigned to “2” with the command:

```
db.adminCommand({
        setDefaultRWConcern : 1, defaultReadConcern: { level:
        “available” }, defaultWriteConcern: { w : 2 },
})
```

This configuration allows read operations to retrieve data without a guarantee that it has been written to a majority of the replica set members. For write operations to succeed, they require acknowledgement from at least two replica set members.

By implementing these changes, we have prioritized availability by allowing read operations to be serviced by the closest replica, and relaxed the requirements for write operations to proceed after receiving confirmation from a subset of the replica set.

14. Dataset choice

We choose a dataset with plenty of data token from kaggle:

Spotify 1.2M+ Songs | Kaggle

Many of the columns were dropped because we found them not useful for the aim of the project, the manipulation of the dataset and the adaptation took us a lot of time. In fact, we didn't also have the users and the accounts, in order to retrieve them we used two scripts in python using the library faker which helped us in generating fake users.