Mongo Report

1

WIT

1

Contents

Introduction…………………………………………………………………………………………………………………………………….2

Installation………………………………………………………………………………………………………………………………………2

Entity Relation Diagram…………………………………………………………………………………………………………………..2

Mongo Database……………………………………………………………………………………………………………………………..3

User…………………………………………………………………………………………………………………………………….3

Playlist………………………………………………………………………………………………………………………………..3

Songs………………………………………………………………………………………………………………………………….4

Genre………………………………………………………………………………………………………………………………….4

Artist…………………………………………………………………………………………………………………………………..5

Mlab………………………………………………………………………………………………………………………………………………..6

Create……………………………………………………………………………………………………………………………………………..7

User…………………………………………………………………………………………………………………………………….7

Playlist………………………………………………………………………………………………………………………………..8

Songs………………………………………………………………………………………………………………………………..10

Genre………………………………………………………………………………………………………………………………..17

Artist………………………………………………………………………………………………………………………………..18

Queries……………………………………………………………………………………………………………………………………….…21

Update………………………………………………………………………………………………………………………………………..…27

Remove…………………………………………………………………………………………………………………………………………28

Conclusion………………………………………………………………………………………………………………………………….…30

References………………………………………………………………………………………………………………………………….…30

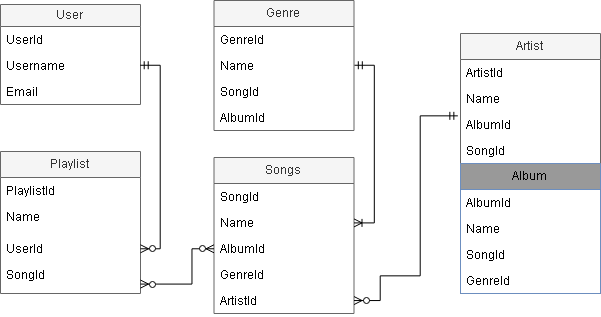
# Introduction

For this project, the goal was to create a NoSQL database based on any idea/project of our choosing. My choice was based off of Spotify/iTunes whereby a user could make playlists that contain music that other artists have uploaded. I felt this was an appropriate project as there was a vast amount of potential collections to pick and choose from. From here, I chose five collections of varying complexity, these being: User, Playlist, Artist, Songs, Genre. A sixth collection, “Album” was originally its own collection, but was later embedded into the artist collection. I created several CRUD queries for this database, in order to add to, delete, or make changes to certain pieces of information within. This database was deployed onto the cloud using mLab.

# Installation

The installation of mongodb was a fairly straightforward process. From [www.mongodb.com](http://www.mongodb.com), I clicked the “get mongodb” button located at the top of the webpage. From here, the installation package was saved and ran. The setup file was executed, installing the program in the desired path environment. From here, mongo was run from the terminal just as any other program would be, by navigating to the correct environment and running the mongo.exe file.

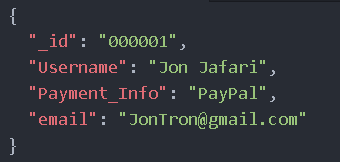
# Entity Relationship Diagram



# 

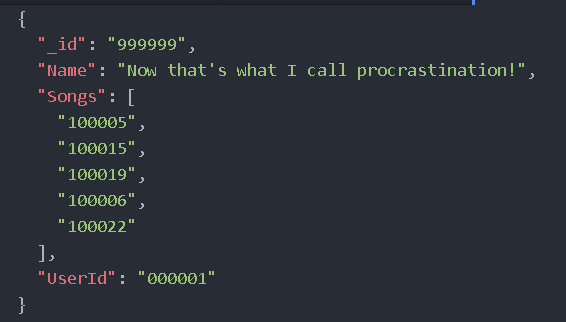
# Mongo Database

## Collection 1: User



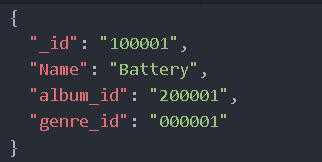
This collection is responsible for storing all the relevant user information. For user, I opted to store a username, email, and method of payment (i.e. PayPal). Each user also has their own unique ID. The ID allows users to be searched for based on the ID.

## Collection 2: Playlist



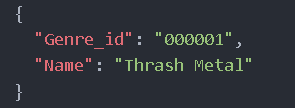
The playlist collection, like the user has its own unique ID. It also has a name, as well as the ID of the user that it belongs to. As we can see from the diagrams above, this playlist belongs to the user from the first image. The songs array stores the IDs of the songs in the playlist. These allows the user to find the playlists belonging to certain users and easily see which songs the playlist contains.

## Collection 3: Songs



The song collection stores, as usual, its own unique ID as well as the name of the song itself. The IDs for the album that it appears on as well as the genre of the song are also stored here. This allows the user to search for a song based on the album or genre ID.

## Collection 4: Genre



Genre is a fairly simple collection, which only stores the name of the genre as well as the unique ID for the genre. As seen above, the genre ID in both the songs sample image and genre sample image match. This shows that the song “Battery” is of the “Thrash Metal” genre.

## Collection 5: Artist



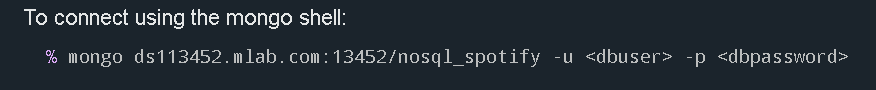
As you can see, artist is the most complex of the collections, as it stores the “Album” object. The usual applies for artist, whereby it stores its own unique ID as well as name. The album object once again has its own unique ID, name, and year of release, while also containing an array of all the song IDs from that album. This allows the user to easily find which album, a particular song features on.

## MLab

The implementation of mLab was quite simple. First an foremost, an account must be created (this was already completed prior, due to the usage of mLab in other modules). From here the user must create a “new deployment”. Choose your desired options (I went with, AWS, Sandbox, and Ireland) and in a few moments, your cloud database will be created. From here, just create the necessary collections. Adding data to these collections is described in the “Create” section below.

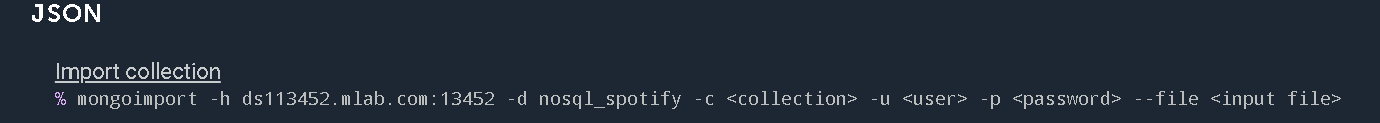
One other thing that is important, is the creation of a user, such that you can access mLab. From within the database, navigate to the “users” tab and create a user. Be sure to remember the username and password, as these will allow us to connect to mLab.

In order to connect to mLab such that you can input the queries listed below (note any queries in the “create” section, must be ran before entering mLab from the terminal), just input the following command which should be displayed at the top of your database page on mLab:



Note replace the <dbuser> and <dbpassword> with the username and password mentioned above.

## Create



For the population of these collections, I used the “mongoimport” command for json files found in the “tools” tab on mLab. This enabled me to simply load in a json file that contained all of the content that I wished to populate the database with.

## Users:

{

"\_id": "000001",

"Username": "Jon Jafari",

"Payment\_Info": "PayPal",

"email": "JonTron@gmail.com"

}

{

"\_id": "000002",

"Username": "John Cena",

"Payment\_Info": "Visa",

"email": "JohnCena@gmail.com"

}

{

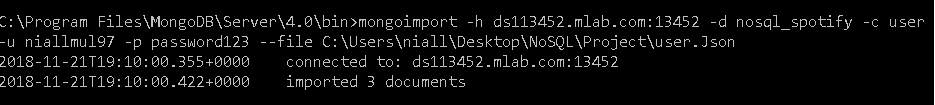
"\_id": "000003",

"Username": "Dwayne 'The Rock' Johnson",

"Payment\_Info": "MasterCard",

"email": "TheRock@gmail.com"

}



## Playlists:

{

"\_id": "999999",

"Name": "Now that's what I call procrastination!",

"Songs": [

"100005",

"100015",

"100019",

"100006",

"100022"

],

"UserId": "000001"

}

{

"\_id": "999998",

"Name": "You can't see me.",

"Songs": [

"100002",

"100018",

"100008",

"100011",

"100029"

],

"UserId": "000002"

}

{

"\_id": "999997",

"Name": "The Rock and Roll",

"Songs": [

"100023",

"100024",

"100014",

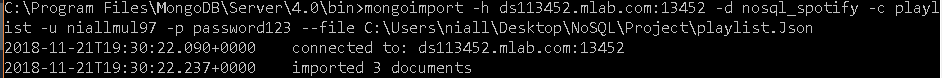
"100013",

"100020"

],

"UserId": "000003"

}



## Songs:

{

"\_id": "100001",

"Name": "Battery",

"album\_id": "200001",

"genre\_id": "000001"

}

{

"\_id": "100002",

"Name": "Master of Puppets",

"album\_id": "200002",

"genre\_id": "000001"

}

{

"\_id": "100003",

"Name": "The Thing That Should Not Be",

"album\_id": "200002",

"genre\_id": "000001"

}

{

"\_id": "100004",

"Name": "Welcome Home (Sanitarium)",

"album\_id": "200002",

"genre\_id": "000001"

}

{

"\_id": "100005",

"Name": "Disposable Heroes",

"album\_id": "200002",

"genre\_id": "000001"

}

{

"\_id": "100006",

"Name": "Leper Messiah",

"album\_id": "200002",

"genre\_id": "000001"

}

{

"\_id": "100007",

"Name": "Orion",

"album\_id": "200002",

"genre\_id": "000001"

}

{

"\_id": "100008",

"Name": "Damage, Inc.",

"album\_id": "200002",

"genre\_id": "000001"

}

{

"\_id": "100009",

"Name": "Ashes",

"album\_id": "200003",

"genre\_id": "000002"

}

{

"\_id": "100010",

"Name": "Rats",

"album\_id": "200003",

"genre\_id": "000002"

}

{

"\_id": "100011",

"Name": "Faith",

"album\_id": "200003",

"genre\_id": "000002"

}

{

"\_id": "100012",

"Name": "See the Light",

"album\_id": "200003",

"genre\_id": "000002"

}

{

"\_id": "100013",

"Name": "Miasma",

"album\_id": "200003",

"genre\_id": "000002"

}

{

"\_id": "100014",

"Name": "Dance Macabre",

"album\_id": "200003",

"genre\_id": "000002"

}

{

"\_id": "100015",

"Name": "Pro Memiora",

"album\_id": "200003",

"genre\_id": "000002"

}

{

"\_id": "100016",

"Name": "Witch Image",

"album\_id": "200003",

"genre\_id": "000002"

}

{

"\_id": "100017",

"Name": "Helvetesfonster",

"album\_id": "200003",

"genre\_id": "000002"

}

{

"\_id": "100018",

"Name": "Life Eternal",

"album\_id": "200003",

"genre\_id": "000002"

}

{

"\_id": "100019",

"Name": "Ocean Planet",

"album\_id": "200002",

"genre\_id": "000003"

}

{

"\_id": "100020",

"Name": "Backbone",

"album\_id": "200002",

"genre\_id": "000003"

}

{

"\_id": "100021",

"Name": "From The Sky",

"album\_id": "200002",

"genre\_id": "000003"

}

{

"\_id": "100022",

"Name": "Unicorn",

"album\_id": "200002",

"genre\_id": "000003"

}

{

"\_id": "100023",

"Name": "Where Dragons Dwell",

"album\_id": "200002",

"genre\_id": "000003"

}

{

"\_id": "100024",

"Name": "Heaviest Matter of the Universe",

"album\_id": "200002",

"genre\_id": "000003"

}

{

"\_id": "100025",

"Name": "Flying Whales",

"album\_id": "200002",

"genre\_id": "000003"

}

{

"\_id": "100026",

"Name": "In The Wilderness",

"album\_id": "200002",

"genre\_id": "000003"

}

{

"\_id": "100027",

"Name": "World To Come",

"album\_id": "200002",

"genre\_id": "000003"

}

{

"\_id": "100028",

"Name": "From Mars",

"album\_id": "200002",

"genre\_id": "000003"

}

{

"\_id": "100029",

"Name": "To Sirus",

"album\_id": "200002",

"genre\_id": "000003"

}

{

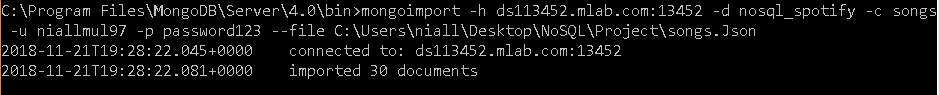
"\_id": "100030",

"Name": "Global Warming",

"album\_id": "200002",

"genre\_id": "000003"

}



## Genre:

{

"\_id": "000001",

"Name": "Thrash Metal"

}

{

"\_id": "000002",

"Name": "Heavy Metal"

}

{

"\_id": "000003",

"Name": "Death Metal"

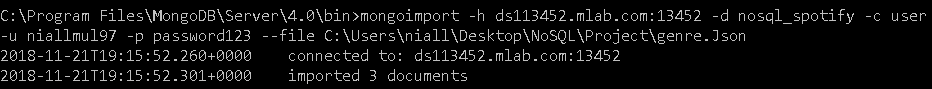
}

{

"\_id": "000004",

"Name": "Doom Metal"

}



## Artists:

{

"\_id": "000001",

"Name": "Metallica",

"Album": [

{

"\_id": "200001",

"Name": "Master of Puppets",

"Songs": [

"000001",

"000002",

"000003",

"000004",

"000005",

"000006",

"000007",

"000008"

],

"Release": "1986"

}

]

}

{

"\_id": "000002",

"Name": "Gojira",

"Album":[

{

"\_id": "200002",

"Name": "From Mars to Sirus",

"Songs": [

"000019",

"000020",

"000021",

"000022",

"000023",

"000024",

"000025",

"000026",

"000027",

"000028",

"000029",

"000030"

],

"Release": "2005"

}

]

}

{

"\_id": "000003",

"Name": "Ghost",

"Album": [

{

"\_id": "200003",

"Name": "Prequelle",

"Songs": [

"000009",

"000010",

"000011",

"000012",

"000013",

"000014",

"000015",

"000016",

"000017",

"000018"

],

"Release": "2018"

}

]

}



## Queries

db.songs.aggregate([

{

$lookup:

{

from: "genre",

localField: "genre\_id",

foreignField: "\_id",

as: "Genre"

}

}

]).pretty()

This command creates a “Genre” object inside the song object, that contains the genre id and the name.



db.artist.aggregate([

{

$lookup:

{

from: "songs",

localField: "Album.\_id",

foreignField: "album\_id",

as: "Album"

}

}

]).pretty()

This command creates an object called “Album” inside the Song collection which contains the song object of every song from a particular album.



db.playlist.aggregate([

{

$lookup:

{

from: "user",

localField: "UserId",

foreignField: "\_id",

as: "User\_Info"

}

}

]).pretty()

This command creates a “User info” field in the playlist collection that contains the user info of the user who made the playlist.



db.playlist.aggregate([

{

$lookup:

{

from: "songs",

localField: "Songs",

foreignField: "\_id",

as: "Featured"

}

}

]).pretty()

This command will create a “Featured” object in the playlist collection, containing the song objects for any songs featured in the playlist.



db.artist.distinct("Album")

This command will return all the album objects from the artist collection.



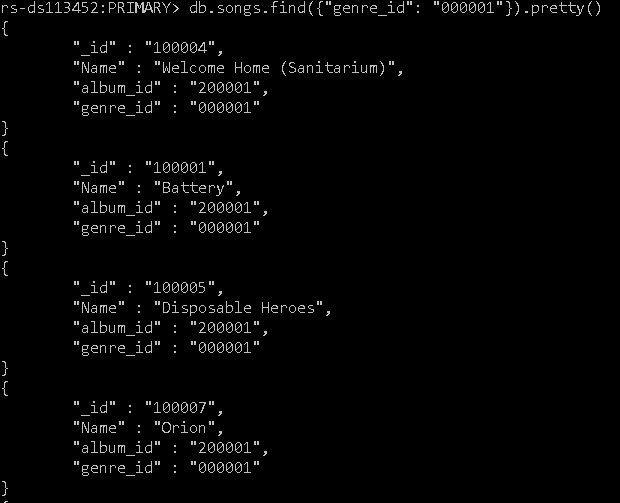
db.artist.distinct("Album.Names")

This command makes an array containing all the album names from the artist collection.



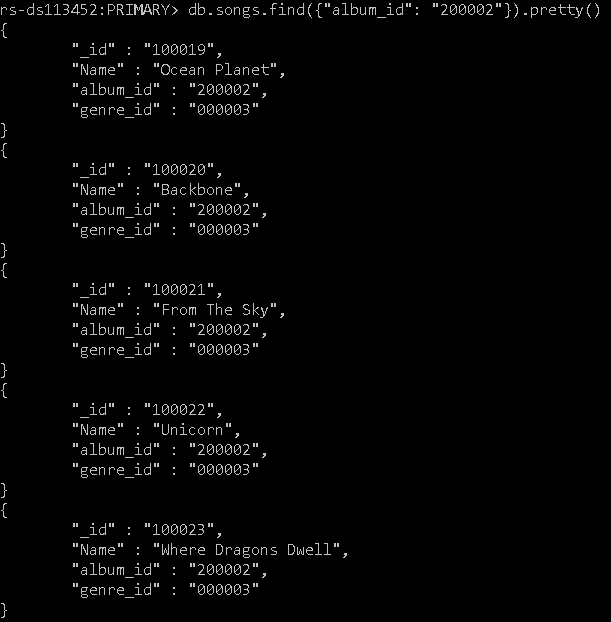
db.songs.find({"genre\_id": "000001"}).pretty()

This command finds all the songs of a particular genre (Thrash Metal in this case).



db.songs.find({"album\_id": "200002"}).pretty()

This command finds all the songs from a particular album (in this case “From Mars to Sirus”).



## Update

db.playlist.update({"\_id": "999999"}, {$set: {"Name": "JohnTron'S playlist"}});

This command updates the name of the playlist belonging to the user with the id “999999”



db.playlist.update({ \_id: "999999" }, { $addToSet: {"Songs":"100001"} })

This command adds the song with the id “100001” to the playlist with the id “999999”.



db.artist.update({

\_id: "000001"

},

{

$addToSet:

{

"Album":{

"\_id": "200004",

"Name": "Ride The Lightning",

"Songs": [],

"Release": "1984"

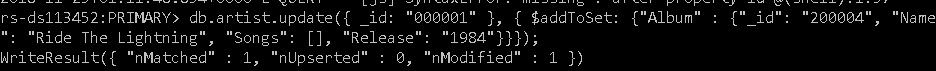
}

}

}

);

This command adds a new album “Ride the Lightning” to the artist of ID “000001” (Metallica).



## Remove

db.playlist.update({ \_id: "999999" }, { $pull: {"Songs":"100022"} })

This command removes the song of ID “100022” from the playlist of ID “999999”



db.artist.update({

\_id: "000001"

},

{

$pull:

{

"Album":{

"\_id": "200004",

"Name": "Ride The Lightning",

"Songs": [],

"Release": "1984"

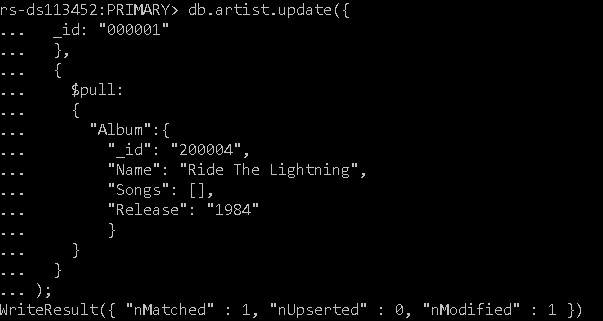
}

}

}

);

This command removes the album “Ride the Lightning” (one we created earlier) from the artist of ID “100001” (Metallica).



db.genre.remove({"\_id" : "000004"})

This command removes the genre of ID “000004” (Doom Metal).



# 

# Conclusion

Following my experiences with SQL Relational Databases last year, I had several doubts about this module at the start of the semester. But from the work throughout this module and this project in particular, I found the fluidity and dynamic nature of noSQL to be quite forgiving in comparison to SQL. In fact, I would go so far as to say that I enjoyed using mLab so much, that it is probably one of the best resources we have ever been shown in this course. The fact that I have personally gone on to use mLabs in four out of my six modules this year (even with that, there was potential to use it in a fifth, but unfortunately due to time constraints, the scope of that project had to be cut back) stands testament to this point.

I found that the creation of ER diagrams fairly straightforward as it was basically the exact same format from last year’s Relational Databases module and the online tool draw.io, made the construction of the diagram itself, very easy.

The constant use of the terminal was immensely frustrating. This was combatted by writing most of my queries in Atom, allowing me to copy them with correct formatting straight from Atom and into the terminal. Atom also helped in the population of the database, by allowing me to write the contents in Json format and use the mongoimport command provided by mLabs to load in the Json file contents to the desired collection.

# References

For the creation of the ER diagram, I used the online tool Draw.io:

<https://www.draw.io/>

For the creation of my Json files, I used the text editor Atom:

<https://atom.io/>

For cloud storage, I used mLab:

<https://mlab.com/>

For Mongo itself, and any MongoDB syntax, I used the MongoDB:

<https://www.mongodb.com/>