CSC 643 – Big Data & Web Intelligence

db.Team1.find({Ahmad : Sean})

Project 1

Table of Contents

1. Obtaining Data and Import into MongoDB

- Download data Last.FM dataset as .dat file type
- Two ways to import data .dat into MongoDB

I. import directly by using statement:

mongoimort -d FMDB -c Artists --file artists.dat --headerline --type tsv

II. convert data from .dat to .csv using Excel, import data .csv into MongoDB by using statement:

mongoimport -d FMDB -c Artists --file artists.csv --headerline --type csv

Note* FMDB is database_name, Artists is collection_name, artists.dat is file_name, type csv/ type tsv is file_type

***To check all imported data, do: **use FMDB → show collections**

2. Description of Data

```
-----------
Description
-----------
    This dataset contains social networking, tagging, and music artist listening information
    from a set of 2K users from Last.fm online music system.
    http://www.last.fm

    The dataset is released in the framework of the 2nd International Workshop on
    Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011)
    http://ir.ii.uam.es/hetrec2011
    at the 5th ACM Conference on Recommender Systems (RecSys 2011)
    http://recsys.acm.org/2011

---------------
Data statistics
---------------

    1892 users
   17632 artists

   12717 bi-directional user friend relations, i.e. 25434 (user_i, user_j) pairs
         avg. 13.443 friend relations per user

   92834 user-listened artist relations, i.e. tuples [user, artist, listeningCount]
         avg. 49.067 artists most listened by each user
         avg. 5.265 users who listened each artist

   11946 tags

  186479 tag assignments (tas), i.e. tuples [user, tag, artist]
         avg. 98.562 tas per user
         avg. 14.891 tas per artist
         avg. 18.930 distinct tags used by each user
avg. 8.764 distinct tags used for each artist
```

```
-----
Files
-----

   * artists.dat

        This file contains information about music artists listened and tagged by the users.

   * tags.dat

         This file contains the set of tags available in the dataset.

   * user_artists.dat

        This file contains the artists listened by each user.

        It also provides a listening count for each [user, artist] pair.

   * user_taggedartists.dat - user_taggedartists-timestamps.dat

        These files contain the tag assignments of artists provided by each particular user.

        They also contain the timestamps when the tag assignments were done.

   * user_friends.dat

         These files contain the friend relations between users in the database.


-----------
Data format
-----------

   The data is formatted one entry per line as follows (tab separated, "\t"):

   * artists.dat

        id \t name \t url \t pictureURL

        Example:
        707      Metallica        http://www.last.fm/music/Metallica        http://userserve-
ak.last.fm/serve/252/7560709.jpg

   * tags.dat

        tagID \t tagValue
        1        metal

   * user_artists.dat

        userID \t artistID \t weight
        2        51       13883

   * user_taggedartists.dat

        userID \t artistID \t tagID \t day \t month \t year
        2        52       13       1        4        2009

   * user_taggedartists-timestamps.dat

        userID \t artistID \t tagID \t timestamp
        2        52       13       1238536800000

   * user_friends.dat

        userID \t friendID
        2        275
```

3. Use Queries to Find the Data

a) find the total number of artists in database

using statement: **db.Artists.count( )** → output: **17632**

```
> db.Artists.count()
17632
>
```

b) find URLs of artists whose name contain the word ***punk***

using statement: **db.Artists.find({url:/punk/i})** / with count: **db.Artists.find({url:/punk/i}).count( )**

→ output: show all the Artists with the word ***punk*** case insensitive, and total number is ***7***

```
> db.Artists.find({url:/punk/i})
{ "_id" : ObjectId("5d81430592182faca3e2a946"), "id" : 56, "name" : "Daft Punk", "url" :
ictureURL" : "http://userserve-ak.last.fm/serve/252/10923145.png" }
{ "_id" : ObjectId("5d81430692182faca3e2c228"), "id" : 6561, "name" : "Jimmy And The Pun
mmy+And+The+Punks", "pictureURL" : "" }
{ "_id" : ObjectId("5d81430692182faca3e2c764"), "id" : 7942, "name" : "Punk Goes...", "u
...", "pictureURL" : "http://userserve-ak.last.fm/serve/252/26385001.jpg" }
{ "_id" : ObjectId("5d81430692182faca3e2de2c"), "id" : 14177, "name" : "Pink Punk", "url
 "pictureURL" : "http://userserve-ak.last.fm/serve/252/79726.jpg" }
{ "_id" : ObjectId("5d81430692182faca3e2e202"), "id" : 15313, "name" : "Daft Punk, Alive
m/music/Daft+Punk%2C+Alive+2007+%28live%29", "pictureURL" : "http://userserve-ak.last.fm
{ "_id" : ObjectId("5d81430692182faca3e2e5ed"), "id" : 16391, "name" : "Punk TV", "url"
ctureURL" : "http://userserve-ak.last.fm/serve/252/22811723.jpg" }
{ "_id" : ObjectId("5d81430692182faca3e2ed9e"), "id" : 18649, "name" : "Brownpunk", "url
 "pictureURL" : "http://userserve-ak.last.fm/serve/252/33706837.jpg" }
> db.Artists.find({url:/punk/i}).count()
7
>
```

c) find the most listened artist(s)

First, you should know which collection you should use to find the result sets.

Based on section ***2. Description of Data***:

* user_artists.dat

```
This file contains the artists listened by each user.

It also provides a listening count for each [user, artist] pair.
```

Therefore, **User_Artists** collection shall be used.

Now let's observe what's inside the **User_Artists** collection data, using: **db.User_Artists.find( )**

➔ output:

```
> db.User_Artists.find()
{ "_id" : ObjectId("5d814565ce77ccc0e6410cdf"), "userID" : 2, "artistID" : 52, "weight" : 11690 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410ce0"), "userID" : 2, "artistID" : 53, "weight" : 11351 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410ce1"), "userID" : 2, "artistID" : 54, "weight" : 10300 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410ce2"), "userID" : 2, "artistID" : 55, "weight" : 8983 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410ce3"), "userID" : 2, "artistID" : 51, "weight" : 13883 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410ce4"), "userID" : 2, "artistID" : 57, "weight" : 5955 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410ce5"), "userID" : 2, "artistID" : 58, "weight" : 4616 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410ce6"), "userID" : 2, "artistID" : 59, "weight" : 4337 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410ce7"), "userID" : 2, "artistID" : 60, "weight" : 4147 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410ce8"), "userID" : 2, "artistID" : 61, "weight" : 3923 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410ce9"), "userID" : 2, "artistID" : 62, "weight" : 3782 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410cea"), "userID" : 2, "artistID" : 63, "weight" : 3735 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410ceb"), "userID" : 2, "artistID" : 64, "weight" : 3644 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410cec"), "userID" : 2, "artistID" : 65, "weight" : 3579 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410ced"), "userID" : 2, "artistID" : 66, "weight" : 3312 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410cee"), "userID" : 2, "artistID" : 67, "weight" : 3301 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410cef"), "userID" : 2, "artistID" : 68, "weight" : 2927 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410cf0"), "userID" : 2, "artistID" : 70, "weight" : 2686 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410cf1"), "userID" : 2, "artistID" : 71, "weight" : 2654 }
{ "_id" : ObjectId("5d814565ce77ccc0e6410cf2"), "userID" : 2, "artistID" : 72, "weight" : 2619 }
Type "it" for more
```

We can see the attribute **artistID** projects each artist, **weight** is the number of listens, therefore, we use them in the statement, we do:

➔ output:

```
> var most = db.User_Artists.aggregate ([
... { $group: {_id:"$artistID", count:{$sum:"$weight"} } },
... { $sort: {count:-1} },
... { $limit:1 } ]).toArray()
>
> most
[ { "_id" : 289, "count" : 2393140 } ]
>
```

Ref:https://docs.mongodb.com/manual/reference/operator/aggregation/group/

This statement perform grouping and counting all the **weight** in the collection **User_Artists**, sorting the count descending and projecting the only first element, and stored in the Array of the variable ***most***, which represent to the most listened artist, obtaining value **2393140**.

To get the artist name, using: **db.Artists.find({id:289},{name:1})**

➔ output:

```
> db.Artists.find({id:289},{name:1})
{ "_id" : ObjectId("5d81430592182faca3e2aa30"), "name" : "Britney Spears" }
>
```

Hence, the most listened artist is "Britney Spears".

d) write a mapReduce to compute the top 10 popular artists

Again, we use same collection **User_Artists** with attribute **artistID** and **weight** to find the result sets.

we do:

```
> var top10 = db.User_Artists.mapReduce(
... function () {emit(this.artistID, this.weight);},
... function (key, values) {return Array.sum(values)},
... {out: "artist_total"}).find().sort({"value": -1}).limit(10).toArray()
>
```

Ref: https://www.tutorialspoint.com/mongodb/mongodb_map_reduce.htm

This statement perform map and reduce for all the **artistID** in the collection, sorting value descending and projecting the only first 10, and stored in the Array of the variable ***top10***, which represent to the most top 10 listened artists.

➔ ouput:

```
> top10
[
        {
                "_id" : 289,
                "value" : 2393140
        },
        {
                "_id" : 72,
                "value" : 1301308
        },
        {
                "_id" : 89,
                "value" : 1291387
        },
        {
                "_id" : 292,
                "value" : 1058405
        },
        {
                "_id" : 498,
                "value" : 963449
        },
        {
                "_id" : 67,
                "value" : 921198
        },
        {
                "_id" : 288,
                "value" : 905423
        },
        {
                "_id" : 701,
                "value" : 688529
        },
        {
                "_id" : 227,
                "value" : 662116
        },
        {
                "_id" : 300,
                "value" : 532545
        }
```

To get the top 10 artist names, we can do one by one using the same statement in 3-C. Alternatively, creating an Array that obtain all 10 **_id,** then project the 10 artist names once together.

Creating an Array of 10 **_id**, we do:

```
> var most10 = top10.map(function(x){return x._id})
> most10
[ 289, 72, 89, 292, 498, 67, 288, 701, 227, 300 ]
>
```

Projecting the top 10 artist names, using: **db.Artists.find({id:{$in:most10}},{name:1})**

➔ output:

```
> db.Artists.find({id: {$in: most10}}, {name:1})
{ "_id" : ObjectId("5d81430592182faca3e2a952"), "name" : "Madonna" }
{ "_id" : ObjectId("5d81430592182faca3e2a957"), "name" : "Depeche Mode" }
{ "_id" : ObjectId("5d81430592182faca3e2a969"), "name" : "Lady Gaga" }
{ "_id" : ObjectId("5d81430592182faca3e2a9f2"), "name" : "The Beatles" }
{ "_id" : ObjectId("5d81430592182faca3e2aa2f"), "name" : "Rihanna" }
{ "_id" : ObjectId("5d81430592182faca3e2aa30"), "name" : "Britney Spears" }
{ "_id" : ObjectId("5d81430592182faca3e2aa33"), "name" : "Christina Aguilera" }
{ "_id" : ObjectId("5d81430592182faca3e2aa3b"), "name" : "Katy Perry" }
{ "_id" : ObjectId("5d81430592182faca3e2aaff"), "name" : "Paramore" }
{ "_id" : ObjectId("5d81430592182faca3e2abcc"), "name" : "Shakira" }
```

e) write a mapReduce to find the least user tag

Following the same procedure as C & D, this time we use **tagID** to write mapReduce.

we do:

➔ output:

```
> var least_tag = db.User_TaggedArtists.mapReduce(
... function() { emit(this.tagID,1); },
... function(key, values) { return Array.sum(values) },
... { out:"total" }).find().sort({"value":1}).limit(1).toArray()
>
> least_tag
[ { "_id" : 11, "value" : 1 } ]
>
```

To get the artist name, using: **db.Artists.find({id:least_tag [0]._id},{name:1})**

➔ output:

```
> db.Artists.find({id:least_tag[0]._id},{name:1})
{ "_id" : ObjectId("5d81430592182faca3e2a91e"), "name" : "Agonoize" }
>
```

Hence, the least user tagged artist is "Agonoize".

f) write a mapReduce to compute the average numer of tags used by each user for all artists

Based on section ***2. Description of Data***:

```
* tags.dat

        This file contains the set of tags available in the dataset.

* user_artists.dat

        This file contains the artists listened by each user.

        It also provides a listening count for each [user, artist] pair.

* user_taggedartists.dat - user_taggedartists-timestamps.dat

        These files contain the tag assignments of artists provided by each particular user.

        They also contain the timestamps when the tag assignments were done.
```

Therefore, **Tags**, **User_Artists** , and **User_TaggedArtists** collection shall be used, as well their attributes, to write map reduce functions.

The average shall be expressed as : avg = tags_sum / tags_count

Similar to the above solution, we write the mapRedce function 2 layers as below:

First layer, we do:

```
> map1 = function(){
... emit( {userID:this.userID, artistID:this.artistID}, {count:1} );}
function(){
emit( {userID:this.userID, artistID:this.artistID}, {count:1} );}
>
> reduce1 = function (key, values) {
... var count = 0;
... values.forEach(function (v) {count += v['count']; });
... return {count: count}; }
function (key, values) {
var count = 0;
values.forEach(function (v) {count += v['count']; });
return {count: count}; }
>
> var tag_count_per_user_per_artist = db.User_TaggedArtists.mapReduce(
... map1, reduce1, {out: "tags_count_per_user_artist"} )
```

To show the result sets of each count, using: **db[tag_count_per_user_per_artist.result].find( )**

➔ output:

```
> db[tag_count_per_user_per_artist.result].find()
{ "_id" : { "userID" : 2, "artistID" : 52 }, "value" : { "count" : 5 } }
{ "_id" : { "userID" : 2, "artistID" : 63 }, "value" : { "count" : 4 } }
{ "_id" : { "userID" : 2, "artistID" : 73 }, "value" : { "count" : 8 } }
{ "_id" : { "userID" : 2, "artistID" : 94 }, "value" : { "count" : 8 } }
{ "_id" : { "userID" : 2, "artistID" : 96 }, "value" : { "count" : 2 } }
{ "_id" : { "userID" : 2, "artistID" : 995 }, "value" : { "count" : 6 } }
{ "_id" : { "userID" : 2, "artistID" : 3894 }, "value" : { "count" : 3 } }
{ "_id" : { "userID" : 2, "artistID" : 6160 }, "value" : { "count" : 2 } }
{ "_id" : { "userID" : 2, "artistID" : 6177 }, "value" : { "count" : 5 } }
{ "_id" : { "userID" : 2, "artistID" : 9322 }, "value" : { "count" : 2 } }
{ "_id" : { "userID" : 3, "artistID" : 101 }, "value" : { "count" : 23 } }
{ "_id" : { "userID" : 3, "artistID" : 102 }, "value" : { "count" : 8 } }
{ "_id" : { "userID" : 3, "artistID" : 106 }, "value" : { "count" : 5 } }
{ "_id" : { "userID" : 3, "artistID" : 108 }, "value" : { "count" : 6 } }
{ "_id" : { "userID" : 3, "artistID" : 122 }, "value" : { "count" : 3 } }
{ "_id" : { "userID" : 3, "artistID" : 128 }, "value" : { "count" : 2 } }
{ "_id" : { "userID" : 3, "artistID" : 130 }, "value" : { "count" : 4 } }
{ "_id" : { "userID" : 3, "artistID" : 134 }, "value" : { "count" : 4 } }
{ "_id" : { "userID" : 3, "artistID" : 137 }, "value" : { "count" : 1 } }
{ "_id" : { "userID" : 4, "artistID" : 51 }, "value" : { "count" : 2 } }
Type "it" for more
```

Second layer,

we do:

```
> map2 = function(){
... emit( {userID:this._id.userID}, {tagsCount:1, tagsSum: this.value.count} );}
function(){
emit( {userID:this._id.userID}, {tagsCount:1, tagsSum: this.value.count} );}
>
> reduce2 = function(key, values){
... reduceVal = {tagsCount:0, tagsSum:0};
... for(var i=0; i<values.length; i++){
... reduceVal.tagsCount += values[i].tagsCount;
... reduceVal.tagsSum += values[i].tagsSum;
... } return reduceVal; }
function(key, values){
reduceVal = {tagsCount:0, tagsSum:0};
for(var i=0; i<values.length; i++){
reduceVal.tagsCount += values[i].tagsCount;
reduceVal.tagsSum += values[i].tagsSum;
} return reduceVal; }
```

Create a function to aggregate the average, we do:

```
> finalize = function(key, reduceVal) {
... reduceVal.avg = reduceVal.tagsSum / reduceVal.tagsCount;
... return reduceVal; }
function(key, reduceVal) {
reduceVal.avg = reduceVal.tagsSum / reduceVal.tagsCount;
return reduceVal; }
```

Applying all functions together to get the result of average, we do:

```
> var users_tags_avg = db[tag_count_per_user_per_artist.result].mapReduce(
... map2, reduce2, {out: "avg}", finalize: finalize});
>
> db[users_tags_avg.result].find()
{ "_id" : { "userID" : 2 }, "value" : { "tagsCount" : 10, "tagsSum" : 45, "avg" : 4.5 } }
{ "_id" : { "userID" : 3 }, "value" : { "tagsCount" : 9, "tagsSum" : 56, "avg" : 6.222222222222222 } }
{ "_id" : { "userID" : 4 }, "value" : { "tagsCount" : 57, "tagsSum" : 61, "avg" : 1.0701754385964912 } }
{ "_id" : { "userID" : 5 }, "value" : { "tagsCount" : 34, "tagsSum" : 43, "avg" : 1.2647058823529411 } }
{ "_id" : { "userID" : 6 }, "value" : { "tagsCount" : 4, "tagsSum" : 14, "avg" : 3.5 } }
{ "_id" : { "userID" : 7 }, "value" : { "tagsCount" : 2, "tagsSum" : 3, "avg" : 1.5 } }
{ "_id" : { "userID" : 8 }, "value" : { "tagsCount" : 7, "tagsSum" : 20, "avg" : 2.857142857142857 } }
{ "_id" : { "userID" : 9 }, "value" : { "tagsCount" : 101, "tagsSum" : 129, "avg" : 1.2772277227722773 } }
{ "_id" : { "userID" : 10 }, "value" : { "tagsCount" : 13, "tagsSum" : 24, "avg" : 1.8461538461538463 } }
{ "_id" : { "userID" : 11 }, "value" : { "tagsCount" : 20, "tagsSum" : 22, "avg" : 1.1 } }
{ "_id" : { "userID" : 12 }, "value" : { "tagsCount" : 257, "tagsSum" : 788, "avg" : 3.066147859922179 } }
{ "_id" : { "userID" : 13 }, "value" : { "tagsCount" : 8, "tagsSum" : 84, "avg" : 10.5 } }
{ "_id" : { "userID" : 14 }, "value" : { "tagsCount" : 110, "tagsSum" : 444, "avg" : 4.036363636363636 } }
{ "_id" : { "userID" : 15 }, "value" : { "tagsCount" : 1, "tagsSum" : 14, "avg" : 14 } }
{ "_id" : { "userID" : 16 }, "value" : { "tagsCount" : 56, "tagsSum" : 188, "avg" : 3.357142857142857 } }
{ "_id" : { "userID" : 17 }, "value" : { "tagsCount" : 1, "tagsSum" : 5, "avg" : 5 } }
{ "_id" : { "userID" : 18 }, "value" : { "tagsCount" : 2, "tagsSum" : 3, "avg" : 1.5 } }
{ "_id" : { "userID" : 20 }, "value" : { "tagsCount" : 6, "tagsSum" : 9, "avg" : 1.5 } }
{ "_id" : { "userID" : 21 }, "value" : { "tagsCount" : 23, "tagsSum" : 284, "avg" : 12.347826086956522 } }
{ "_id" : { "userID" : 22 }, "value" : { "tagsCount" : 3, "tagsSum" : 4, "avg" : 1.3333333333333333 } }
Type "it" for more
```

Ref: https://docs.mongodb.com/manual/reference/method/db.collection.mapReduce/

4. Connect MongoDB using Java, write and execute all above queries

Here, we won't describe the procedure how to connect MongoDB with Java, for more information and instruction, please go to this link http://people.sju.edu/~bforoura/tutorials/nosql/index.htm#interface

To call the database and collections in Java,

```
// connect to mongodb with default address and port
MongoClient mongoClient = MongoClients.create();
// Connect to FMDB database
MongoDatabase fmdb = mongoClient.getDatabase("FMDB");
// Get the Artists collection
MongoCollection<Document> artistsCollection = fmdb.getCollection("Artists");
```

a) find the total number of artists in database

```
/** 3 - a Find the total number of artists in the database */
System.out.println(
        "3 - a\n" + "Total number of artists in FMDB: " + artistsCollection.countDocuments() + "\n");
```

b) find URLs of artists whose name contain the word ***punk***

```
System.out.println("3-b find URLs of artists whose name contain the word \"punk\"");
System.out.println();
// use the artists collection to do the filtration
// first do create the filter items
Document whereQ = new Document();
// here to identify the regex and ignore case
whereQ.put("url", new Document("$regex", "punk").append("$options", "i"));
// Apply the filter
FindIterable<Document> punkResult = Artists.find(whereQ);
Iterator<Document> itr = punkResult.iterator();
while (itr.hasNext()) {
    System.out.println(itr.next().toJson());
}
System.out.println();
System.out.println("Total "+Artists.countDocuments(whereQ));
```

c) find the most listened artist

```
/**
 * 3 - c Find the most listened artist(s).
 * https://mongodb.github.io/mongo-java-driver/3.11/driver/tutorials/aggregation/
 */
// Use user_artists collection to do the aggregation
MongoCollection<Document> user_artists_collection = fmdb.getCollection("User_Artists");

// apply an aggregation to do grouping by artist id and sort list as descending.
// then limit the number of row to 1
AggregateIterable<Document> user_artists_aggregation = user_artists_collection
        .aggregate(Arrays.asList(Aggregates.group("$artistID", Accumulators.sum("count", "$weight")),
                Aggregates.sort(new BsonDocument("count", new BsonInt32(-1))), Aggregates.limit(1)));
// Display the result
System.out.println("3 - c\n" + "Most Listened Artist");

Iterator<Document> itrUAA = user_artists_aggregation.iterator();
while (itrUAA.hasNext()) {
    System.out.println(itrUAA.next().toJson());
}

// Get the corresponding name for the most listened it
// iterate on the user_artists_aggregation result to get the ids one buy on
Iterator<Document> itr = user_artists_aggregation.iterator();
while (itr.hasNext()) {
    // get the id
    int lookFor_id = itr.next().getInteger("_id");
    // using find command and projection to display name only
    FindIterable mostListenedName = artistsCollection
            .find(new BasicDBObject("id", new BsonInt32(lookFor_id)))
            .projection(new BasicDBObject("name", new BsonInt32(1)));
    // print the result
    Iterator<Document> itrListenedName = mostListenedName.iterator();
    while (itrListenedName.hasNext()) {
        System.out.println(itrListenedName.next().toJson());
    }
    System.out.println();

}
```

d) find the top 10 artists

```
/** 3 - d Write a mapReducer to compute the top 10 popular artists */
// use User_Artists Collection
System.out.println("3 - d");
String map = "function () {emit({artistID:this.artistID}, {count:this.weight});}";
String reduce = "function (key, values) {var count = 0; " + "values.forEach(function (v) { "
        + " count += v['count']; });" + " return {count: count};}";

MapReduceIterable<Document> mapReduceMost10 = user_artists_collection.mapReduce(map, reduce);
// Create a collection for the mapReduce with mapReduceMost10Result
mapReduceMost10.collectionName("mapReduceMost10Result");
// Get the created collection
MongoCollection<Document> mapReduceMost10Collection = fmdb.getCollection("mapReduceMost10Result");
// Apply find, sort descendingly and limit to ten commands.
FindIterable<Document> mapReduceMost10Collection_find = mapReduceMost10Collection.find()
        .sort(new BasicDBObject("value.count", new BsonInt32(-1))).limit(10);
// Print the result
Iterator<Document> itrMost10 = mapReduceMost10Collection_find.iterator();
while (itrMost10.hasNext()) {
    System.out.println(itrMost10.next().toJson());
}
```

e) write a mapReduce to find the least user tag

```
/** 3 - e Write a mapReducer to find the least used tag */
System.out.println("3 - e");
// User User_TaggedArtists collection
MongoCollection<Document> user_TaggedArtists_collection = fmdb.getCollection("User_TaggedArtists");
String mapLeastTag = "function () { emit(this.tagID,1);}";
String reduceLeastTag = "function(key,values){return Array.sum(values);}";
// Apply the mapReduce on the User_TaggedArtists collection
MapReduceIterable<Document> mapReduceLeastTag = user_TaggedArtists_collection.mapReduce(mapLeastTag,
        reduceLeastTag);
// Create a collection for the mapReduce with mapReduceLeastTagResult
mapReduceLeastTag.collectionName("mapReduceLeastTagResult");
// Get the created collection
MongoCollection<Document> mapReduceLeastTagCollection = fmdb.getCollection("mapReduceLeastTagResult");
// Apply find, sort ascendancy and limit to one commands
FindIterable<Document> mapReduceLeastTagCollection_find = mapReduceLeastTagCollection.find()
        .sort(new BsonDocument("value", new BsonInt32(1))).limit(1);
// iterate to print data
Iterator<Document> itrLeastTag = mapReduceLeastTagCollection_find.iterator();
while (itrLeastTag.hasNext()) {
    System.out.println(itrLeastTag.next().toJson());
}
```

f) write a mapReduce to compute the average numer of tags used by each user for all artists

```
 * 3 - f Write a mapReducer to compute the average number of tags used by each
 * user for all artists
 */
System.out.println("3 - f");
// use User_TaggedArtists collection
String mapTagAvg1 = "function () {\n"
        + "    emit({userID: this.userID, artistID: this.artistID}, {count: 1});\n" + "}";
String reduceTagAvg1 = "function (key, values) {\n" + "    var count = 0;\n"
        + "    values.forEach(function (v) {\n" + "        count += v['count'];\n" + "\n" + "    });\n"
        + "    return {count: count};\n" + "}";

MapReduceIterable<Document> mapReduceTagAvg1 = user_TaggedArtists_collection.mapReduce(mapTagAvg1,
        reduceTagAvg1);
// Create a collection for the mapReduce with tags_count_per_user_artist
// collection
mapReduceTagAvg1.collectionName("tags_count_per_user_artist");
// Get the created collection
MongoCollection<Document> mapReduceTagAvgOutput = fmdb.getCollection("tags_count_per_user_artist");
String mapTagAvg2 = "function () {\n"
        + "  emit({userID:this._id.userID},{tagsCount : 1, tagsSum: this.value.count});\n" + "}";
String reduceTagAvg2 = "function (key, values) {\n" + "    reducedVal = {tagsCount: 0, tagsSum: 0};\n"
        + "    for(var i = 0; i < values.length; i++){\n"
        + "        reducedVal.tagsCount += values[i].tagsCount;\n"
        + "        reducedVal.tagsSum += values[i].tagsSum;\n" + "    }\n" + "    return reducedVal;\n"
        + "}";
String finalizeTagAvg = "function (key, reducedVal) {\n"
        + "    reducedVal.avg = reducedVal.tagsSum / reducedVal.tagsCount;\n" + "\n"
        + "    return reducedVal;\n" + "}";
// Apply the 2nd mapReduce on the collection of 1st mapReduce
MapReduceIterable<Document> mapReduceTagAvg2 = mapReduceTagAvgOutput.mapReduce(mapTagAvg2, reduceTagAvg2)
        .finalizeFunction(finalizeTagAvg);
// Create a collection the mapReduce to collection with mapReduceTagAvg2Result
// name
mapReduceTagAvg2.collectionName("mapReduceTagAvg2Result");
// Get the created collection mapReduceTagAvg2Result
MongoCollection<Document> mapReduceTagAvg2Collection = fmdb.getCollection("mapReduceTagAvg2Result");
// Do Find command on the collection
FindIterable<Document> mapReduceTagAvg2Collection_find = mapReduceTagAvg2Collection.find().limit(10);
// iterate the collection to print data
Iterator<Document> itrTagAvg = mapReduceTagAvg2Collection_find.iterator();
while (itrTagAvg.hasNext()) {
    System.out.println(itrTagAvg.next().toJson());
}
```

Appendix:

1- Java Source:

```
import com.mongodb.*;
import com.mongodb.client.*;
import com.mongodb.client.MongoClient;
import com.mongodb.client.model.*;
import org.bson.*;
import java.util.Arrays;
import java.util.Iterator;

public class JavaToMongoDriver {
	public static void main(String[] args) {

		try {
			// connect to mongodb with default address and port
			MongoClient mongoClient = MongoClients.create();
			// Connect to FMDB database
			MongoDatabase fmdb = mongoClient.getDatabase("FMDB");
			// FGet the Artists collection
			MongoCollection<Document> artistsCollection = fmdb.getCollection("Artists");

			/** 3 - a Find the total number of artists in the database */
			System.out.println(
					"3 - a\n" + "Total number of artists in FMDB: " +
artistsCollection.countDocuments() + "\n");

			/** 3 - b Find URLs of artists whose names contain the word punk. */

			// use the artists collection to do the filtration
```

```
// first do create the filter items
Document whereQ = new Document();
// here to identify the regex and ignore case
whereQ.put("url", new Document("$regex", "punk").append("$options", "i"));
// Apply the filter
FindIterable<Document> punkResult = artistsCollection.find(whereQ);

System.out.println("3 - b\n db.Artists.find({url:/punk/i}");
Iterator<Document> itrPunkResult = punkResult.iterator();
while (itrPunkResult.hasNext()) {
        System.out.println(itrPunkResult.next().toJson());
}
// Empty Line
System.out.println();
// print the count of the filter
System.out.println("db.Artists.find({url:/punk/i}).count( )");
System.out.println(artistsCollection.countDocuments(whereQ));
// Empty Line
System.out.println();

/**
 * 3 - c Find the most listened artist(s).
 * https://mongodb.github.io/mongo-java-driver/3.11/driver/tutorials/aggregation/
 */
// Use user_artists collection to do the aggregation
MongoCollection<Document> user_artists_collection =
fmdb.getCollection("User_Artists");

// apply an aggregation to do grouping by artist id and sort list as descending.
// then limit the number of row to 1
AggregateIterable<Document> user_artists_aggregation = user_artists_collection
```

```
                .aggregate(Arrays.asList(Aggregates.group("$artistID",
Accumulators.sum("count", "$weight")),
                        Aggregates.sort(new BsonDocument("count", new
BsonInt32(-1))), Aggregates.limit(1)));
        // Display the result
        System.out.println("3 - c\n" + "Most Listened Artist");

        Iterator<Document> itrUAA = user_artists_aggregation.iterator();
        while (itrUAA.hasNext()) {
            System.out.println(itrUAA.next().toJson());
        }

        // Get the corresponding name for the most listened it
        // iterate on the user_artists_aggregation result to get the ids one buy on
        Iterator<Document> itr = user_artists_aggregation.iterator();
        while (itr.hasNext()) {
            // get the id
            int lookFor_id = itr.next().getInteger("_id");
            // using find command and projection to display name only
            FindIterable mostListenedName = artistsCollection
                    .find(new BasicDBObject("id", new BsonInt32(lookFor_id)))
                    .projection(new BasicDBObject("name", new BsonInt32(1)));
            // print the result
            Iterator<Document> itrListenedName = mostListenedName.iterator();
            while (itrListenedName.hasNext()) {
                System.out.println(itrListenedName.next().toJson());
            }
            System.out.println();

        }
```

```
/** 3 - d Write a mapReducer to compute the top 10 popular artists */
// use User_Artists Collection
System.out.println("3 - d");
String map = "function () {emit({artistID:this.artistID}, {count:this.weight});}";
String reduce = "function (key, values) {var count = 0; " + "values.forEach(function (v) { "
        + " count += v['count']; });" + " return {count: count};}";

MapReduceIterable<Document> mapReduceMost10 =
user_artists_collection.mapReduce(map, reduce);
// Create a collection for the mapReduce with mapReduceMost10Result
mapReduceMost10.collectionName("mapReduceMost10Result");
// Get the created collection
MongoCollection<Document> mapReduceMost10Collection =
fmdb.getCollection("mapReduceMost10Result");
// Apply find, sort descendingly and limit to ten commands.
FindIterable<Document> mapReduceMost10Collection_find =
mapReduceMost10Collection.find()
        .sort(new BasicDBObject("value.count", new BsonInt32(-1))).limit(10);
// Print the result
Iterator<Document> itrMost10 = mapReduceMost10Collection_find.iterator();
while (itrMost10.hasNext()) {
    System.out.println(itrMost10.next().toJson());
}

/** 3 - e Write a mapReducer to find the least used tag */
System.out.println("3 - e");
// User User_TaggedArtists collection
MongoCollection<Document> user_TaggedArtists_collection =
fmdb.getCollection("User_TaggedArtists");
String mapLeastTag = "function () { emit(this.tagID,1);}";
String reduceLeastTag = "function(key,values){return Array.sum(values);}";
// Apply the mapReduce on the User_TaggedArtists collection
```

```
MapReduceIterable<Document> mapReduceLeastTag =
user_TaggedArtists_collection.mapReduce(mapLeastTag,
                reduceLeastTag);
// Create a collection for the mapReduce with mapReduceLeastTagResult
mapReduceLeastTag.collectionName("mapReduceLeastTagResult");
// Get the created collection
MongoCollection<Document> mapReduceLeastTagCollection =
fmdb.getCollection("mapReduceLeastTagResult");
// Apply find, sort ascendancy and limit to one commands
FindIterable<Document> mapReduceLeastTagCollection_find =
mapReduceLeastTagCollection.find()
                .sort(new BsonDocument("value", new BsonInt32(1))).limit(1);
// iterate to print data
Iterator<Document> itrLeastTag = mapReduceLeastTagCollection_find.iterator();
while (itrLeastTag.hasNext()) {
        System.out.println(itrLeastTag.next().toJson());
}

/** Part 3 - f is Not completed yet */
/**
 * 3 - f Write a mapReducer to compute the average number of tags used by each
 * user for all artists
 */
System.out.println("3 - f");
// use User_TaggedArtists collection
String mapTagAvg1 = "function () {\n"
                + "    emit({userID: this.userID, artistID: this.artistID}, {count: 1});\n" + "}";
String reduceTagAvg1 = "function (key, values) {\n" + "    var count = 0;\n"
                + "    values.forEach(function (v) {\n" + "       count += v['count'];\n" + "\n"
+ "    });\n"
                + "    return {count: count};\n" + "}";
```

```
MapReduceIterable<Document> mapReduceTagAvg1 =
user_TaggedArtists_collection.mapReduce(mapTagAvg1,
                reduceTagAvg1);
// Create a collection for the mapReduce with tags_count_per_user_artist
// collection
mapReduceTagAvg1.collectionName("tags_count_per_user_artist");
// Get the created collection
MongoCollection<Document> mapReduceTagAvgOutput =
fmdb.getCollection("tags_count_per_user_artist");
String mapTagAvg2 = "function () {\n"
                + "  emit({userID:this._id.userID},{tagsCount : 1, tagsSum:
this.value.count});\n" + "}";
String reduceTagAvg2 = "function (key, values) {\n" + "    reducedVal = {tagsCount: 0,
tagsSum: 0};\n"
                + "    for(var i = 0; i < values.length; i++){\n"
                + "       reducedVal.tagsCount += values[i].tagsCount;\n"
                + "       reducedVal.tagsSum += values[i].tagsSum;\n" + "    }\n" + "
return reducedVal;\n"
                + "}";
String finalizeTagAvg = "function (key, reducedVal) {\n"
                + "    reducedVal.avg = reducedVal.tagsSum / reducedVal.tagsCount;\n" +
"\n"
                + "    return reducedVal;\n" + "}";
// Apply the 2nd mapReduce on the collection of 1st mapReduce
MapReduceIterable<Document> mapReduceTagAvg2 =
mapReduceTagAvgOutput.mapReduce(mapTagAvg2, reduceTagAvg2)
                .finalizeFunction(finalizeTagAvg);
// Create a collection the mapReduce to collection with mapReduceTagAvg2Result
// name
mapReduceTagAvg2.collectionName("mapReduceTagAvg2Result");
// Get the created collection mapReduceTagAvg2Result
MongoCollection<Document> mapReduceTagAvg2Collection =
fmdb.getCollection("mapReduceTagAvg2Result");
```

```
                        // Do Find command on the collection
                        FindIterable<Document> mapReduceTagAvg2Collection_find =
mapReduceTagAvg2Collection.find().limit(10);
                        // iterate the collection to print data
                        Iterator<Document> itrTagAvg = mapReduceTagAvg2Collection_find.iterator();
                        while (itrTagAvg.hasNext()) {
                                System.out.println(itrTagAvg.next().toJson());
                        }

                } catch (MongoException e) {
                        e.printStackTrace();
                }
        }
}
```

2- Java Output:

```
3 - a
Total number of artists in FMDB: 17633

3 - b
 db.Artists.find({url:/punk/i}
{"_id": {"$oid": "5d81525a08975ef3ffc51696"}, "id": 56, "name": "Daft Punk",
"url": "http://www.last.fm/music/Daft+Punk", "pictureURL": "http://userserve-
ak.last.fm/serve/252/10923145.png"}
{"_id": {"$oid": "5d81525b08975ef3ffc52f77"}, "id": 6561, "name": "Jimmy And The
Punks", "url": "http://www.last.fm/music/Jimmy+And+The+Punks", "pictureURL": ""}
{"_id": {"$oid": "5d81525b08975ef3ffc534b9"}, "id": 7942, "name": "Punk Goes...",
"url": "http://www.last.fm/music/Punk+Goes...", "pictureURL": "http://userserve-
ak.last.fm/serve/252/26385001.jpg"}
{"_id": {"$oid": "5d81525b08975ef3ffc54b80"}, "id": 14177, "name": "Pink Punk",
"url": "http://www.last.fm/music/Pink+Punk", "pictureURL": "http://userserve-
ak.last.fm/serve/252/79726.jpg"}
{"_id": {"$oid": "5d81525b08975ef3ffc54f57"}, "id": 15313, "name": "Daft Punk,
Alive 2007 (live)", "url":
"http://www.last.fm/music/Daft+Punk%2C+Alive+2007+%28live%29", "pictureURL":
"http://userserve-ak.last.fm/serve/252/7048893.jpg"}
{"_id": {"$oid": "5d81525b08975ef3ffc5534a"}, "id": 16391, "name": "Punk TV",
"url": "http://www.last.fm/music/Punk+TV", "pictureURL": "http://userserve-
ak.last.fm/serve/252/22811723.jpg"}
{"_id": {"$oid": "5d81525b08975ef3ffc55aed"}, "id": 18649, "name": "Brownpunk",
"url": "http://www.last.fm/music/Brownpunk", "pictureURL": "http://userserve-
ak.last.fm/serve/252/33706837.jpg"}
```

```
db.Artists.find({url:/punk/i}).count( )
7

3 - c
Most Listened Artist
{"_id": 289, "count": 2393140}
{"_id": {"$oid": "5d81525a08975ef3ffc51788"}, "name": "Britney Spears"}

3 - d
{"_id": {"artistID": 289.0}, "value": {"count": 2393140.0}}
{"_id": {"artistID": 72.0}, "value": {"count": 1301308.0}}
{"_id": {"artistID": 89.0}, "value": {"count": 1291387.0}}
{"_id": {"artistID": 292.0}, "value": {"count": 1058405.0}}
{"_id": {"artistID": 498.0}, "value": {"count": 963449.0}}
{"_id": {"artistID": 67.0}, "value": {"count": 921198.0}}
{"_id": {"artistID": 288.0}, "value": {"count": 905423.0}}
{"_id": {"artistID": 701.0}, "value": {"count": 688529.0}}
{"_id": {"artistID": 227.0}, "value": {"count": 662116.0}}
{"_id": {"artistID": 300.0}, "value": {"count": 532545.0}}


3 - e
{"_id": 11.0, "value": 1.0}




3 - f
{"_id": {"userID": 2.0}, "value": {"tagsCount": 10.0, "tagsSum": 45.0, "avg":
4.5}}
{"_id": {"userID": 3.0}, "value": {"tagsCount": 9.0, "tagsSum": 56.0, "avg":
6.222222222222222}}
{"_id": {"userID": 4.0}, "value": {"tagsCount": 57.0, "tagsSum": 61.0, "avg":
1.0701754385964912}}
{"_id": {"userID": 5.0}, "value": {"tagsCount": 34.0, "tagsSum": 43.0, "avg":
1.2647058823529411}}
{"_id": {"userID": 6.0}, "value": {"tagsCount": 4.0, "tagsSum": 14.0, "avg":
3.5}}
{"_id": {"userID": 7.0}, "value": {"tagsCount": 2.0, "tagsSum": 3.0, "avg": 1.5}}
{"_id": {"userID": 8.0}, "value": {"tagsCount": 7.0, "tagsSum": 20.0, "avg":
2.857142857142857}}
{"_id": {"userID": 9.0}, "value": {"tagsCount": 101.0, "tagsSum": 129.0, "avg":
1.2772277227722773}}
{"_id": {"userID": 10.0}, "value": {"tagsCount": 13.0, "tagsSum": 24.0, "avg":
1.8461538461538463}}
{"_id": {"userID": 11.0}, "value": {"tagsCount": 20.0, "tagsSum": 22.0, "avg":
1.1}}
```