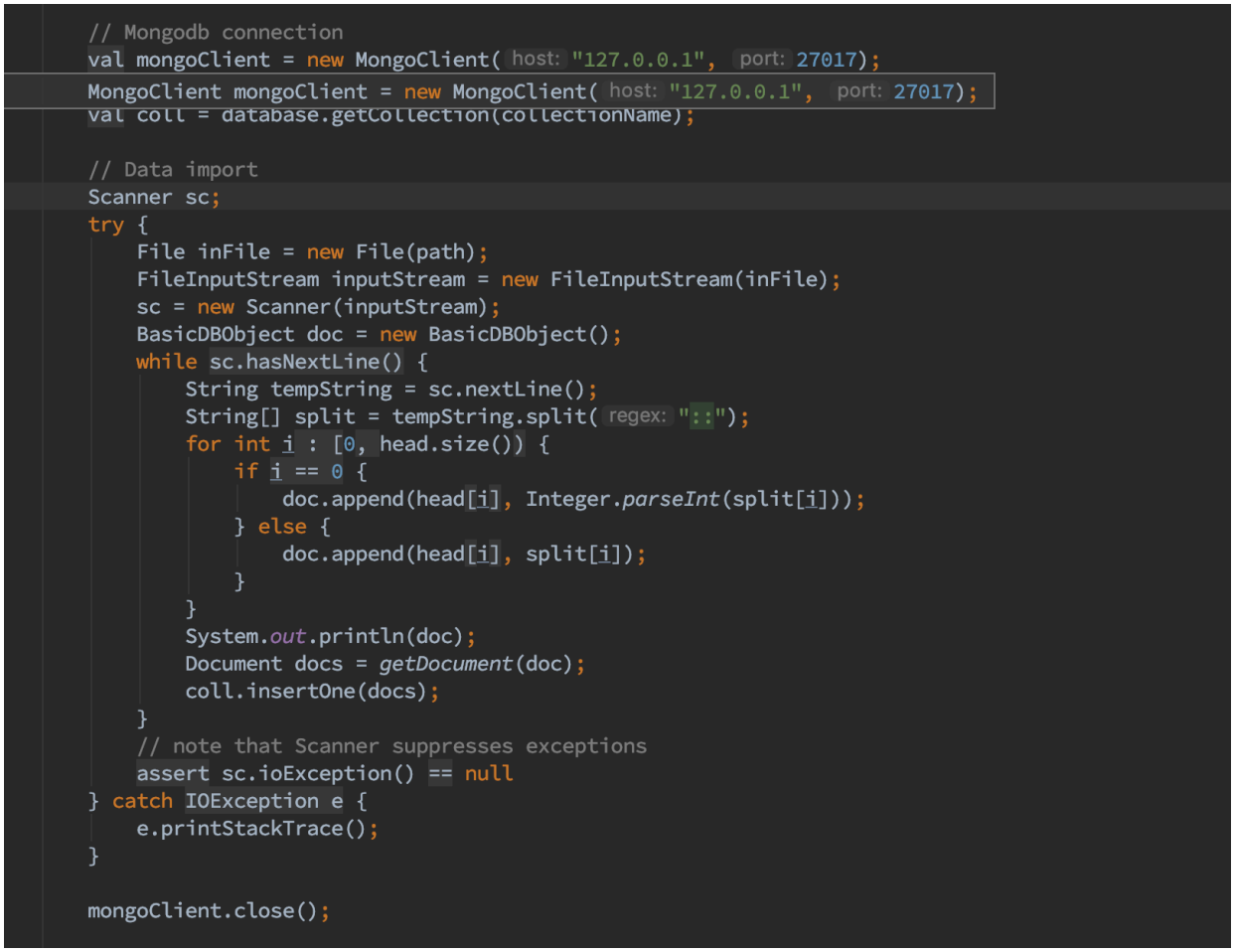
ASSIGNMENT 1 - Storage and processing of data

By Jing Pang, Tian Xue, Chuqian Ma, Jiaxiang Leng

To fulfill the request, we choose to use MongoDB as database storage for the dataset which consists of 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. We construct a local environment for a MongoDB database to store our data. We set up a new database named mydb as our initial database. Then we use Java language as a pipeline to load the primitive data from three separate files into the MongoDB database. The whole process could review from java file named as DBMoviesBuilder, DBTagsBuilder and DBRatingsBuilder. We answered four questions as requested. We firstly solve each question answer by using MongoDB's shell language. Then we have translated this method into Java language. The DBQueryOne to DBQueryFour will point to each problem separately. The following screenshots show that how the data load into the MongoDB database by using Java code.



```
// Mongodb connection
val mongoClient = new MongoClient( host: "127.0.0.1",  port: 27017);
MongoClient mongoClient = new MongoClient( host: "127.0.0.1",  port: 27017);
val coll = database.getCollection(collectionName);

// Data import
Scanner sc;
try {
    File inFile = new File(path);
    FileInputStream inputStream = new FileInputStream(inFile);
    sc = new Scanner(inputStream);
    BasicDBObject doc = new BasicDBObject();
    while sc.hasNextLine() {
        String tempString = sc.nextLine();
        String[] split = tempString.split( regex: "::");
        for int i : [0, head.size()) {
            if i == 0 {
                doc.append(head[i], Integer.parseInt(split[i]));
            } else {
                doc.append(head[i], split[i]);
            }
        }
        System.out.println(doc);
        Document docs = getDocument(doc);
        coll.insertOne(docs);
    }
    // note that Scanner suppresses exceptions
    assert sc.ioException() == null
} catch IOException e {
    e.printStackTrace();
}

mongoClient.close();
```

```
// Mongodb connection
val mongoClient = new MongoClient( host: "127.0.0.1",  port: 27017);
val database = mongoClient.getDatabase(dbName);
val coll = database.getCollection(collectionName);

// Data import
Scanner sc;
try {
    val inFile = new File(path);
    val inputStream = new FileInputStream(inFile);
    sc = new Scanner(inputStream);
    val doc = new BasicDBObject();
    while sc.hasNextLine() {
        val tempString = sc.nextLine();
        val split = tempString.split( regex: "::");
        for val i : [0, head.size()) {
            if i == 2 {
                doc.append(head[i], split[i]);
            } else if i == 3 {
                doc.append(head[i], new Timestamp(Long.valueOf(split[i]) * 1000));
            } else {
                doc.append(head[i], Integer.parseInt(split[i]));
            }
        }
        System.out.println(doc);
        val docs = getDocument(doc);
        coll.insertOne(docs);
    }
    // note that Scanner suppresses exceptions
    assert sc.ioException() == null
} catch IOException e {
    e.printStackTrace();
}

mongoClient.close();
```

```
// Data import
Scanner sc;
try {
    val inFile = new File(path);
    val inputStream = new FileInputStream(inFile);
    sc = new Scanner(inputStream);
    val doc = new BasicDBObject();
    while sc.hasNextLine() {
        val tempString = sc.nextLine();
        val split = tempString.split( regex: "::");
        for val i : [0, head.size()) {
            if i == 2 {
                doc.append(head[i], Double.parseDouble(split[i]));
            } else if i == 3 {
                doc.append(head[i], new Timestamp(Long.valueOf(split[i]) * 1000));
            } else {
                doc.append(head[i], Integer.parseInt(split[i]));
            }
        }
        System.out.println(doc);
        val docs = getDocument(doc);
        coll.insertOne(docs);
    }
    // note that Scanner suppresses exceptions
    assert sc.ioException() == null
} catch IOException e {
    e.printStackTrace();
}

mongoClient.close();
```

/Users/pangjing/Desktop/Screen Shot 2018-10-06 at 22.23.06.png

1. Write a query that finds average rating of each movie.

   To answer this question, we write a simple MongoDB shell query to help us figure out the correct solution. Then we translate this theory into a Java language to fit with this shell query.

```
db.ratings.aggregate( [
    { $group: { _id: "$MovieID", avgRating: { $avg: "$Rating" } } }, {
$sort : { _id : 1 } }
]).pretty()
```

```
public static void main(String[] args) {

    // Prepare documents
    val dbName = "mydb";

    [< 1.8 >] java.util.logging
    public class Logger extends Object
    odb server
    val mongoLogger = Logger.getLogger("org.mongodb.driver");
    mongoLogger.level = Level.SEVERE;

    // Mongodb connection
    val mongoClient = new MongoClient( host: "127.0.0.1",  port: 27017);
    val database = mongoClient.getDatabase(dbName);

    // 1) Write a query that finds average rating of each movie.
    //db.ratings.aggregate( [
    //    { $group: { _id: "$MovieID", avgRating: { $avg: "$Rating" } } }, { $sort : { _id : 1 } }
    //]).pretty()

    // This is how I get question 1 result in java

    val ratings = database.getCollection( collectionName: "ratings");

    val queryOne = ratings.aggregate(Arrays.asList(
            Aggregates.group( id: "$MovieID", avg( fieldName: "AverageRating",  expression: "$Rating")),
            Aggregates.sort(orderBy(ascending( ...fieldNames: "_id")))
    ));

    printResult(queryOne);
  }
}
```

```
DBQueryThree › main()
Run:  DBQueryOne
Document{{_id=65006, AverageRating=4.0}}
Document{{_id=65011, AverageRating=4.0}}
Document{{_id=65025, AverageRating=2.0}}
Document{{_id=65027, AverageRating=2.5}}
Document{{_id=65037, AverageRating=3.4166666666666665}}
Document{{_id=65088, AverageRating=3.0625}}
Document{{_id=65091, AverageRating=3.6666666666666665}}
Document{{_id=65126, AverageRating=3.6666666666666665}}
Document{{_id=65130, AverageRating=2.25}}
Document{{_id=65133, AverageRating=3.357142857142857}}
```

2. Write a query that finds users who are similar to a given user (target user), the id of the target user is an input parameter. Users are similar to the target user if they rate the same movies.

   First of all, we use a query to find out the total number of users, which is 71509.

```
db.ratings.aggregate([
      {
        $group:
          {
            _id: "$MovieID",
           max: { $max: "$UserID" }
          }
      }
])
```

Secondly, we have to create a temporary table that contains the information between users and the statistical result of rating movies. The MongoDB's bucket function is using in this process, and we used preious find of total user number as a bucket parameter setting.

```
db.ratings.aggregate([
    { $group: {
        _id: "$UserID",
        count: { $sum: 1 },
        Movies: {$push: "$MovieID"}
    }}
])
```

In the end, we find this question's answer by using MongoDB lookup function through the temporary table. The solution to how to successfully getting the correct result is shown in the following query. The Java file will show that how do we obtain this one by converting the idea from query to Java language.

```
db.tempuser.aggregate([
   {
      $lookup:
         {
           from: "tempuser",
           let: { numberOfMovies: "$numberOfMovies", movies: "$Movies" },
           pipeline: [
              { $match:
                 { $expr:
                    { $and:
                       [
                         { $eq: [ "$Movies",  "$$movies" ] },
                         { $eq: [ "$numberOfMovies",  "$$numberOfMovies" ]
}
                       ]
                    }
                 }
              }
           ],
           as: "moviematch"
         }
    },
    { $match: { "_id": { $eq: 1} }},
    { $project: { moviematch: 1, _id: 0}}
]).pretty()
```

Because the dataset is extremely large, we only test our proccess in our test database. The screenshots of whole process results are shown in the following.

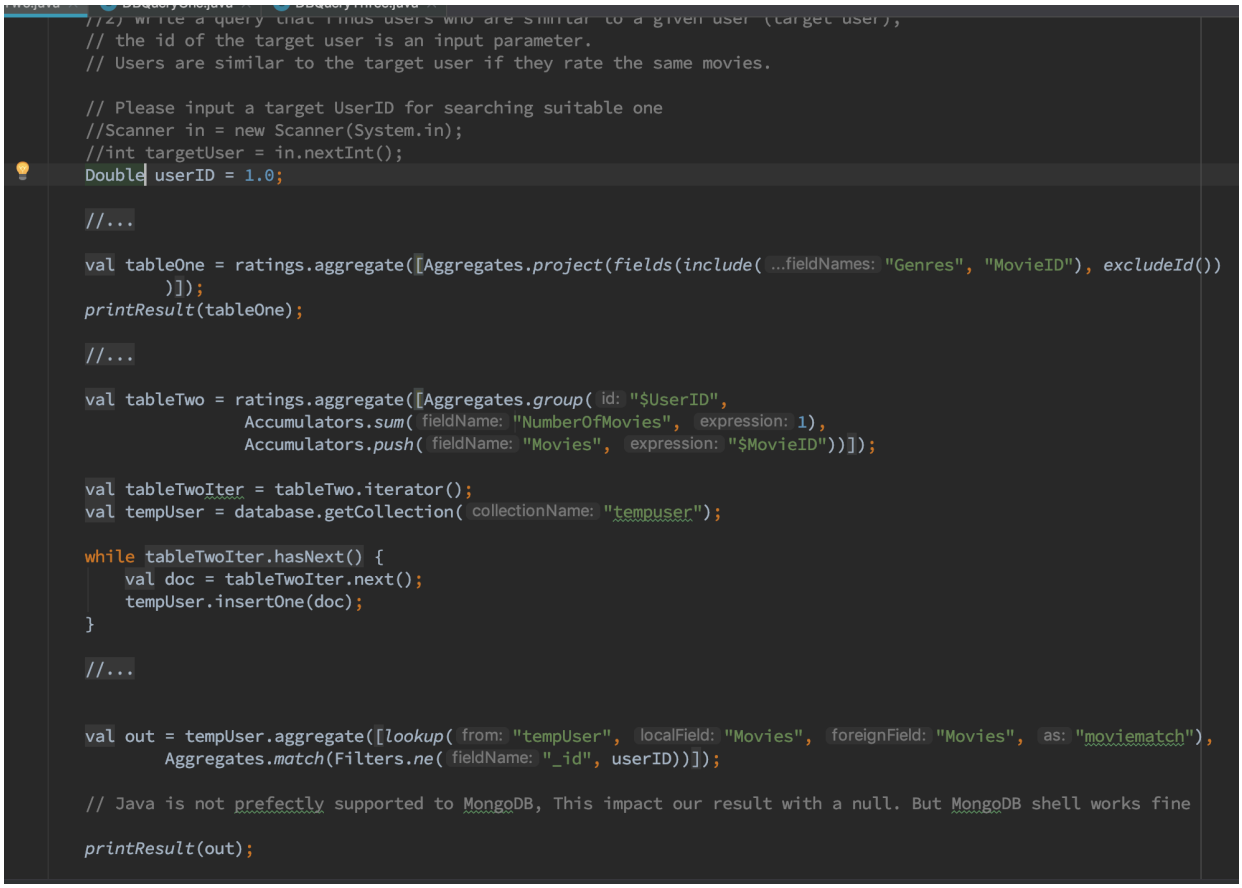
We build a simple database to test our query

```
db.test.insert([
   { "_id" : 1, "MovieID" : 1, "UserID" : 1},
   { "_id" : 2, "MovieID" : 1, "UserID" : 2},
   { "_id" : 3, "MovieID" : 1, "UserID" : 3},
```

```
    { "_id" : 4, "MovieID" : 1, "UserID" : 4},
    { "_id" : 5, "MovieID" : 1, "UserID" : 5},
    { "_id" : 6, "MovieID" : 1, "UserID" : 6},
    { "_id" : 7, "MovieID" : 1, "UserID" : 7},
    { "_id" : 8, "MovieID" : 1, "UserID" : 8},
    { "_id" : 9, "MovieID" : 1, "UserID" : 9},
    { "_id" : 10, "MovieID" : 2, "UserID" : 1},
    { "_id" : 11, "MovieID" : 2, "UserID" : 2},
    { "_id" : 12, "MovieID" : 1, "UserID" : 4},
    { "_id" : 13, "MovieID" : 1, "UserID" : 11},
    { "_id" : 14, "MovieID" : 1, "UserID" : 12},
    { "_id" : 15, "MovieID" : 1, "UserID" : 13},
    { "_id" : 16, "MovieID" : 1, "UserID" : 10},
    { "_id" : 17, "MovieID" : 1, "UserID" : 14},
    { "_id" : 18, "MovieID" : 1, "UserID" : 15},
    { "_id" : 19, "MovieID" : 1, "UserID" : 16},
    { "_id" : 20, "MovieID" : 1, "UserID" : 17},
])
```

By using this database, it should result a return tuple matching with input target UserID 1.

```
> db.tempuser.aggregate([
...     {
...        $lookup:
...           {
...             from: "tempuser",
...             let: { numberOfMovies: "$numberOfMovies", movies: "$Movies" },
...             pipeline: [
...                { $match:
...                   { $expr:
...                      { $and:
...                         [
...                           { $eq: [ "$Movies",  "$$movies" ] },
...                           { $eq: [ "$numberOfMovies",  "$$numberOfMovies" ] }
...                         ]
...                      }
...                   }
...                }
...             ],
...             as: "moviematch"
...           }
...      },
...      { $match: { "moviematch._id": { $eq: 1} }},
...      { $match: { "_id": { $ne: 1} }},
...      { $project: { moviematch: 0}}
... ]).pretty()
{ "_id" : 2, "NumberOfMovies" : 2, "Movies" : [ 1, 2 ] }
>
```



```
//2) Write a query that finds users who are similar to a given user (target user);
// the id of the target user is an input parameter.
// Users are similar to the target user if they rate the same movies.

// Please input a target UserID for searching suitable one
//Scanner in = new Scanner(System.in);
//int targetUser = in.nextInt();
Double userID = 1.0;

//...

val tableOne = ratings.aggregate([Aggregates.project(fields(include( ...fieldNames: "Genres", "MovieID"), excludeId())
        )]);
printResult(tableOne);

//...

val tableTwo = ratings.aggregate([Aggregates.group( id: "$UserID",
                Accumulators.sum( fieldName: "NumberOfMovies", expression: 1),
                Accumulators.push( fieldName: "Movies", expression: "$MovieID"))]);

val tableTwoIter = tableTwo.iterator();
val tempUser = database.getCollection( collectionName: "tempuser");

while tableTwoIter.hasNext() {
    val doc = tableTwoIter.next();
    tempUser.insertOne(doc);
}

//...


val out = tempUser.aggregate([lookup( from: "tempUser", localField: "Movies", foreignField: "Movies", as: "moviematch"),
        Aggregates.match(Filters.ne( fieldName: "_id", userID))]);

// Java is not prefectly supported to MongoDB, This impact our result with a null. But MongoDB shell works fine

printResult(out);
```

3. Write a query that finds to number of movies in each genre.

To answer this question, we write a simple MongoDB shell query to help us figure out the correct solution. Then we translate this theory into a Java language to fit with this shell query.

```
db.movies.aggregate([
    { $project : { genre : { $split: ["$Genres", "|"] }, MovieID: 1} },
    { $unwind : "$genre" },
    { $group: { _id: "$genre", numberOfMovies: { $sum: 1 } } }
]).pretty()
```

```
val movies = database.getCollection( collectionName: "movies");

val queryThreeExtract = movies.aggregate(Arrays.asList(
        Aggregates.project(fields(include( ...fieldNames: "Genres", "MovieID"), excludeId())
        )));

val tempDrop = database.getCollection( collectionName: "tempGenre");
tempDrop.drop();
val tempGenre = database.getCollection( collectionName: "tempGenre");

val doc = new BasicDBObject();
val cursor = queryThreeExtract.iterator();

while cursor.hasNext() {
    val item_doc = cursor.next();

    val genreBeforeSep = item_doc.getString( key: "Genres");
    val movieID = item_doc.getInteger( key: "MovieID");

    val arrayToSplit = genreBeforeSep.split("\|");

    for val i : [0, arrayToSplit.length) {
        System.out.println(arrayToSplit[i]);
        doc.append( key: "MovieID", movieID);
        doc.append( key: "Genres", arrayToSplit[i]);
    }
    val genreNew = getDocument(doc);
    tempGenre.insertOne(genreNew);
}


val QueryThreeFinal = tempGenre.aggregate([sortByCount( filter: "$Genres")]);

printResult(QueryThreeFinal);
```

```
Run:  DBQueryThree
Document{{_id=Musical, count=289}}
Document{{_id=Western, count=275}}
Document{{_id=Fantasy, count=236}}
Document{{_id=Crime, count=180}}
Document{{_id=Mystery, count=175}}
Document{{_id=Children, count=92}}
Document{{_id=Action, count=82}}
Document{{_id=Adventure, count=77}}
Document{{_id=Film-Noir, count=69}}
Document{{_id=IMAX, count=24}}
Document{{_id=Animation, count=5}}
Document{{_id=(no genres listed), count=1}}
```

4. Write 3 different queries of your choice to demonstrate that your data storage is working.

   4.1. Show each movie's information and the tags.

```
1>. Show each movie's information and the tags.
db.movies.aggregate([
   {
      $lookup:
         {
           from: "tags",
           let: { movieid: "$MovieID" },
           pipeline: [
              { $match:
                 { $expr:
                    { $and:
                       [
                         { $eq: [ "$MovieID",  "$$movieid" ] }
                       ]
                    }
```

```
                }
            },
            { $project: { _id:0,Tag:1 } }
        ],
        as: "movietags"
      }
  }
]).pretty()
```

```
// 4-1) Show each movie's information and the tags.
val variables = [new Variable<>( name: "movieid",  value: "$MovieID")];

val pipeline = [Aggregates.match(expr(new Document("$and",
                [new Document("$eq", ["$MovieID", "$$movieid"])]))),
        project(fields(include( ...fieldNames: "Tag"), excludeId()))];

val a = movies.aggregate([Aggregates.lookup( from: "tags", variables, pipeline,  as: "movietags")]);

printResult(a);
```

```
Document{{_id=5bb122b24bf37a16f4c61b47, Title=Toy Story (1995), MovieID=1, Genres=Adventure|Animation|Children|Comedy|Fantasy, movietags=[Document{{Tag=
Document{{_id=5bb122b24bf37a16f4c61b48, Title=Jumanji (1995), MovieID=2, Genres=Adventure|Children|Fantasy, movietags=[Document{{Tag=For children}}, Doc
Document{{_id=5bb122b24bf37a16f4c61b49, Title=Grumpier Old Men (1995), MovieID=3, Genres=Comedy|Romance, movietags=[Document{{Tag=Funniest Movies}}, Doc
Document{{_id=5bb122b24bf37a16f4c61b4a, Title=Waiting to Exhale (1995), MovieID=4, Genres=Comedy|Drama|Romance, movietags=[Document{{Tag=girl movie}}]}}
Document{{_id=5bb122b24bf37a16f4c61b4b, Title=Father of the Bride Part II (1995), MovieID=5, Genres=Comedy, movietags=[Document{{Tag=steve martin}}, Doc
Document{{_id=5bb122b24bf37a16f4c61b4c, Title=Heat (1995), MovieID=6, Genres=Action|Crime|Thriller, movietags=[Document{{Tag=Can't remember}}, Document{
Document{{_id=5bb122b24bf37a16f4c61b4d, Title=Sabrina (1995), MovieID=7, Genres=Comedy|Romance, movietags=[Document{{Tag=based on a play}}, Document{{Ta
Document{{_id=5bb122b24bf37a16f4c61b4e, Title=Tom and Huck (1995), MovieID=8, Genres=Adventure|Children, movietags=[Document{{Tag=adapted from:book}}, D
Document{{_id=5bb122b24bf37a16f4c61b4f, Title=Sudden Death (1995), MovieID=9, Genres=Action, movietags=[Document{{Tag=Can't remember}}, Document{{Tag=Je
Document{{_id=5bb122b24bf37a16f4c61b50, Title=GoldenEye (1995), MovieID=10, Genres=Action|Adventure|Thriller, movietags=[Document{{Tag=franchise}}, Docu
Document{{_id=5bb122b24bf37a16f4c61b51, Title=American President, The (1995), MovieID=11, Genres=Comedy|Drama|Romance, movietags=[Document{{Tag=girlie m
Document{{_id=5bb122b24bf37a16f4c61b52, Title=Dracula: Dead and Loving It (1995), MovieID=12, Genres=Comedy|Horror, movietags=[Document{{Tag=Mel Brooks}
Document{{_id=5bb122b24bf37a16f4c61b53, Title=Balto (1995), MovieID=13, Genres=Animation|Children, movietags=[Document{{Tag=Ei muista}}, Document{{Tag=d
Document{{_id=5bb122b24bf37a16f4c61b54, Title=Nixon (1995), MovieID=14, Genres=Drama, movietags=[Document{{Tag=5}}, Document{{Tag=biopic}}, Document{{Ta
Document{{_id=5bb122b24bf37a16f4c61b55, Title=Cutthroat Island (1995), MovieID=15, Genres=Action|Adventure|Romance, movietags=[Document{{Tag=pirates}},
Document{{_id=5bb122b24bf37a16f4c61b56, Title=Casino (1995), MovieID=16, Genres=Crime|Drama, movietags=[Document{{Tag=based on a book}}, Document{{Tag=i
Document{{_id=5bb122b24bf37a16f4c61b57, Title=Sense and Sensibility (1995), MovieID=17, Genres=Comedy|Drama|Romance, movietags=[Document{{Tag=Ang Lee}},
Document{{_id=5bb122b24bf37a16f4c61b58, Title=Four Rooms (1995), MovieID=18, Genres=Comedy|Drama|Thriller, movietags=[Document{{Tag=Tarantino}}, Documen
Document{{_id=5bb122b24bf37a16f4c61b59, Title=Ace Ventura: When Nature Calls (1995), MovieID=19, Genres=Comedy, movietags=[Document{{Tag=Not my kind of
Document{{_id=5bb122b24bf37a16f4c61b5a, Title=Money Train (1995), MovieID=20, Genres=Action|Comedy|Crime|Drama|Thriller, movietags=[Document{{Tag=intens
Document{{_id=5bb122b24bf37a16f4c61b5b, Title=Get Shorty (1995), MovieID=21, Genres=Action|Comedy|Drama, movietags=[Document{{Tag=Hollywood}}, Document{
Document{{_id=5bb122b24bf37a16f4c61b5c, Title=Copycat (1995), MovieID=22, Genres=Crime|Drama|Horror|Mystery|Thriller, movietags=[Document{{Tag=serial ki
Document{{_id=5bb122b24bf37a16f4c61b5d, Title=Assassins (1995), MovieID=23, Genres=Action|Crime|Thriller, movietags=[Document{{Tag=afternoon section}},
Document{{_id=5bb122b24bf37a16f4c61b5e, Title=Powder (1995), MovieID=24, Genres=Drama|Sci-Fi, movietags=[Document{{Tag=albino}}, Document{{Tag=albino}},
Document{{_id=5bb122b24bf37a16f4c61b5f, Title=Leaving Las Vegas (1995), MovieID=25, Genres=Drama|Romance, movietags=[Document{{Tag=alcoholism}}, Documen
Document{{_id=5bb122b24bf37a16f4c61b60, Title=Othello (1995), MovieID=26, Genres=Drama, movietags=[Document{{Tag=based on a play}}, Document{{Tag=Shakes
Document{{_id=5bb122b24bf37a16f4c61b61, Title=Now and Then (1995), MovieID=27, Genres=Drama, movietags=[]}}
Document{{_id=5bb122b24bf37a16f4c61b62, Title=Persuasion (1995), MovieID=28, Genres=Drama|Romance, movietags=[Document{{Tag=based on a book}}, Document{
Document{{_id=5bb122b24bf37a16f4c61b63, Title=City of Lost Children, The (Cit茅des enfants perdus, La) (1995), MovieID=29, Genres=Adventure|Drama|Fantasy
Document{{_id=5bb122b24bf37a16f4c61b64, Title=Shanghai Triad (Yao a yao yao dao waipo qiao) (1995), MovieID=30, Genres=Crime|Drama, movietags=[]}}
Document{{_id=5bb122b24bf37a16f4c61b65, Title=Dangerous Minds (1995), MovieID=31, Genres=Drama, movietags=[Document{{Tag=Dangerous Minds}}, Document{{Ta
```

4.2. Show all movie information evaluated by person with userid of 1024.

```
2>. Show all movie information evaluated by person with userid of 1024.
db.ratings.aggregate([
   {
      $lookup:
         {
           from: "ratings",
           let: { movieid: "$MovieID"},
           pipeline: [
              { $match:
                    { $and:
                       [
                         { $eq: [ "$MovieID",  "$$movieid" ] },
                         { $eq: [ "$UserID", 1024 ] }
                       ]
                    }
              },
              { $project: { MovieID: 0, _id: 0 } }
```

```
            ],
            as: "moviematch"
          }
    },
    { $group : { _id : "$MovieID" } }
]).pretty()
```

```
// 4-2) Show all movie information evaluated by person with Userid of 1024
[mongo-java-driver-3.8.2] org.bson.conversions
public interface Bson
        //Use lookup to combine "tags" collection and "movies" collection
        Aggregates.lookup( from: "movies", localField: "MovieID", foreignField: "MovieID", as: "movieInfo"),
        //Use match to select person with Userid of 1024
        Aggregates.match(Filters.eq( fieldName: "UserID", value: 1024)),
        //Use project to filter useful information and output.
        //In this selection, only reserve the attributes "UserID", "MovieID" in "tags", and attributes "Title",
        Aggregates.project(Projections.fields(
                Projections.include( ...fieldNames: "UserID", "MovieID", "movieInfo.Title", "movieInfo.Genres"),
                Projections.excludeId()))
));

printResult(queryOne);
```

```
Document{{MovieID=3450, UserID=1024, movieInfo=[Document{{Title=Grumpy Old Men (1993), Genres=Comedy}}]}}
Document{{MovieID=3755, UserID=1024, movieInfo=[Document{{Title=Perfect Storm, The (2000), Genres=Drama|Thriller}}]}}
Document{{MovieID=3825, UserID=1024, movieInfo=[Document{{Title=Coyote Ugly (2000), Genres=Drama}}]}}
Document{{MovieID=3916, UserID=1024, movieInfo=[Document{{Title=Remember the Titans (2000), Genres=Drama}}]}}
Document{{MovieID=3916, UserID=1024, movieInfo=[Document{{Title=Remember the Titans (2000), Genres=Drama}}]}}
Document{{MovieID=3967, UserID=1024, movieInfo=[Document{{Title=Billy Elliot (2000), Genres=Drama}}]}}
Document{{MovieID=3967, UserID=1024, movieInfo=[Document{{Title=Billy Elliot (2000), Genres=Drama}}]}}
Document{{MovieID=4015, UserID=1024, movieInfo=[Document{{Title=Dude, Where's My Car? (2000), Genres=Comedy|Sci-Fi}}]}}
Document{{MovieID=4270, UserID=1024, movieInfo=[Document{{Title=Mummy Returns, The (2001), Genres=Action|Adventure|Horror|Thriller}}]}}
Document{{MovieID=4367, UserID=1024, movieInfo=[Document{{Title=Lara Croft: Tomb Raider (2001), Genres=Action|Adventure}}]}}
Document{{MovieID=4367, UserID=1024, movieInfo=[Document{{Title=Lara Croft: Tomb Raider (2001), Genres=Action|Adventure}}]}}
Document{{MovieID=4447, UserID=1024, movieInfo=[Document{{Title=Legally Blonde (2001), Genres=Comedy}}]}}
Document{{MovieID=4621, UserID=1024, movieInfo=[Document{{Title=Look Who's Talking (1989), Genres=Comedy}}]}}
Document{{MovieID=4621, UserID=1024, movieInfo=[Document{{Title=Look Who's Talking (1989), Genres=Comedy}}]}}
Document{{MovieID=5064, UserID=1024, movieInfo=[Document{{Title=Count of Monte Cristo, The (2002), Genres=Drama|Thriller}}]}}
Document{{MovieID=5152, UserID=1024, movieInfo=[Document{{Title=We Were Soldiers (2002), Genres=Action|Drama|War}}]}}
Document{{MovieID=5152, UserID=1024, movieInfo=[Document{{Title=We Were Soldiers (2002), Genres=Action|Drama|War}}]}}
Document{{MovieID=5152, UserID=1024, movieInfo=[Document{{Title=We Were Soldiers (2002), Genres=Action|Drama|War}}]}}
Document{{MovieID=5308, UserID=1024, movieInfo=[Document{{Title=Three Men and a Baby (1987), Genres=Comedy}}]}}
Document{{MovieID=5377, UserID=1024, movieInfo=[Document{{Title=About a Boy (2002), Genres=Comedy|Drama}}]}}
Document{{MovieID=5528, UserID=1024, movieInfo=[Document{{Title=One Hour Photo (2002), Genres=Drama|Thriller}}]}}
Document{{MovieID=6373, UserID=1024, movieInfo=[Document{{Title=Bruce Almighty (2003), Genres=Comedy}}]}}
Document{{MovieID=6378, UserID=1024, movieInfo=[Document{{Title=Italian Job, The (2003), Genres=Action|Adventure|Crime|Drama}}]}}
Document{{MovieID=6385, UserID=1024, movieInfo=[Document{{Title=Whale Rider (2002), Genres=Drama}}]}}
Document{{MovieID=6787, UserID=1024, movieInfo=[Document{{Title=All the President's Men (1976), Genres=Drama|Thriller}}]}}
Document{{MovieID=8361, UserID=1024, movieInfo=[Document{{Title=Day After Tomorrow, The (2004), Genres=Action|Adventure|Drama|Sci-Fi|Thriller}}]}}
Document{{MovieID=45722, UserID=1024, movieInfo=[Document{{Title=Pirates of the Caribbean: Dead Man's Chest (2006), Genres=Action|Adventure|Comedy|Fantasy}}]}}
Document{{MovieID=46578, UserID=1024, movieInfo=[Document{{Title=Little Miss Sunshine (2006), Genres=Comedy|Drama}}]}}
Document{{MovieID=46578, UserID=1024, movieInfo=[Document{{Title=Little Miss Sunshine (2006), Genres=Comedy|Drama}}]}}
```

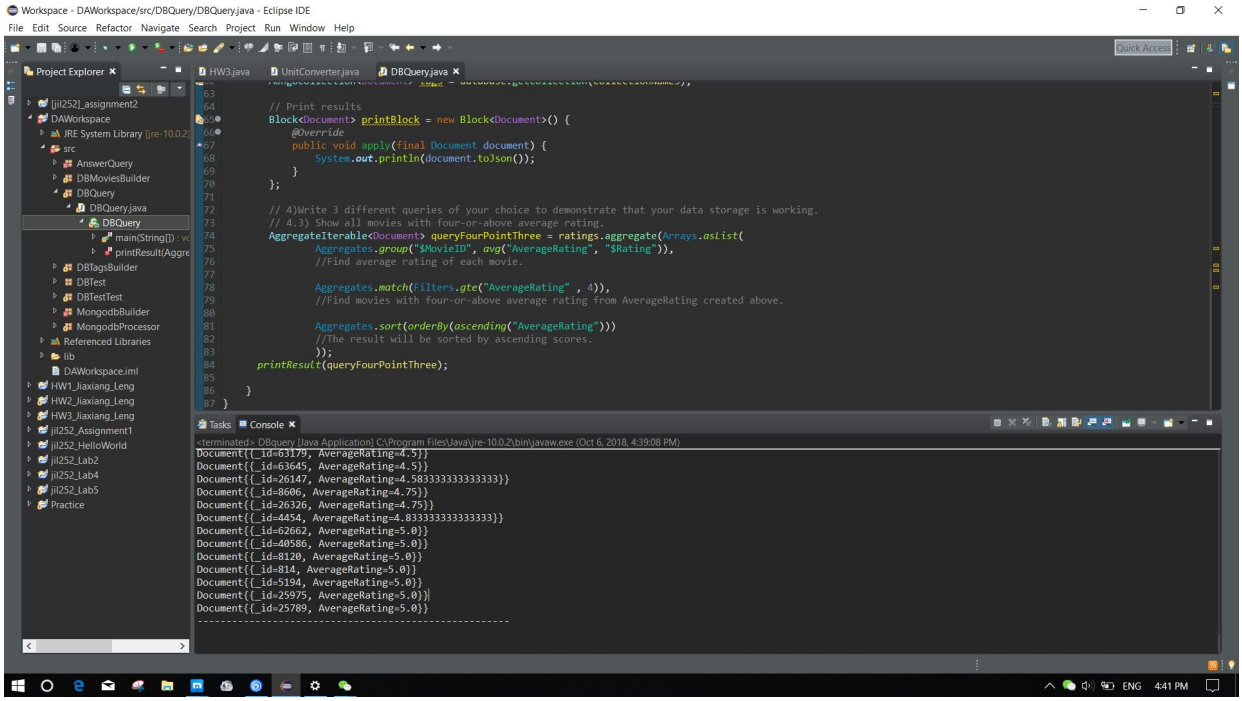
4.3. Show all movies with larger-than-four average rating.

```
3>. Show all movies with larger-than-four average rating.
db.ratings.aggregate([
{$group:{ _id:"$MovieID",AverageRating:{$avg:"$Rating"}}},
{$match:{AverageRating:{$gte:4}}},
{$sort:{AverageRating:1}}
]).pretty()
```

```
// 4.3) Show all movies with four-or-above average rating.
val queryFourPointThree = ratings.aggregate(Arrays.asList(
        Aggregates.group( id: "$MovieID", avg( fieldName: "AverageRating", expression: "$Rating")),
        //Find average rating of each movie.

        Aggregates.match(Filters.gte( fieldName: "AverageRating" , value: 4)),
        //Find movies with four-or-above average rating from AverageRating created above.

        Aggregates.sort(orderBy(ascending( ...fieldNames: "AverageRating")))
        //The result will be sorted by ascending scores.
));
printResult(queryFourPointThree);
```


Workspace - DAWorkspace/src/DBQuery/DBQuery.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
HW3.java
UnitConverter.java
DBQuery.java
// Print results
Block<Document> printBlock = new Block<Document>() {
@Override
public void apply(final Document document) {
System.out.println(document.toJson());
}
};
// 4)Write 3 different queries of your choice to demonstrate that your data storage is working.
// 4.3) Show all movies with four-or-above average rating.
AggregateIterable<Document> queryFourPointThree = ratings.aggregate(Arrays.asList(
Aggregates.group("$MovieID", avg("AverageRating", "$Rating")),
//Find average rating of each movie.
Aggregates.match(Filters.gte("AverageRating" , 4)),
//Find movies with four-or-above average rating from AverageRating created above.
Aggregates.sort(orderBy(ascending("AverageRating")))
//The result will be sorted by ascending scores.
));
printResult(queryFourPointThree);
}
}
Tasks
Console
<terminated> DBQuery [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (Oct 6, 2018, 4:39:08 PM)
Document{{_id=63179, AverageRating=4.5}}
Document{{_id=63645, AverageRating=4.5}}
Document{{_id=26147, AverageRating=4.583333333333333}}
Document{{_id=8606, AverageRating=4.75}}
Document{{_id=26326, AverageRating=4.75}}
Document{{_id=4454, AverageRating=4.833333333333333}}
Document{{_id=62662, AverageRating=5.0}}
Document{{_id=40586, AverageRating=5.0}}
Document{{_id=8120, AverageRating=5.0}}
Document{{_id=814, AverageRating=5.0}}
Document{{_id=5194, AverageRating=5.0}}
Document{{_id=25975, AverageRating=5.0}}
Document{{_id=25789, AverageRating=5.0}}
ENG
4:41 PM