The project is two parts, MapReduce and MongoDB

- For the two parts you need to discuss them, no submission on Moodle.

- For the MapReduce part you can write code on any IDE, and you must run it on the VM.

    - If it not on the VM you loose two marks, but still your code has to be in Java project that has all dependencies.

- Regarding the MongoDB, prepare the queries, and in the discussion show how to run them on MongoDB.
    - If not you loose 2 marks

- Last allowed date of submission is 18/5/2023,

Part 1: MapReduce

Create and and samples of the following relations, each relation represents a dataset of text files stores on HDFS.

Avg( )

1. ratings (UserID, MovieID, Rating) // where rating represent the rating between (from 1 to 5) given by the user to the corresponding movieID

2. users (UserID, Gender, Age)

3. movies (MovieID, Title, Genres) // where genres in the classification of the movie such as comedy, children, action, ….

Suppose you have been given a task to find the average rating for each movie in the form (movieID, Title, avg_rating). Computing the average rating must consider the following:

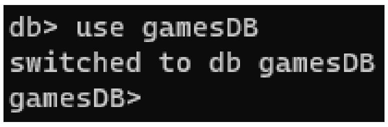
4. only children and comedy movies
5. consider rating values that are above 2
6. consider ratings from users who’s age is above 25

- Create a java project that contains MapReduce code to achieve the above described task, write as much as needed MapReduce jobs.

Part 2: MongoDB

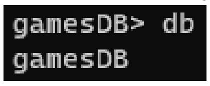
For each of the following, you need to write a query and also show the output (result returned from the shell)

1. Create a new database called gamesDB



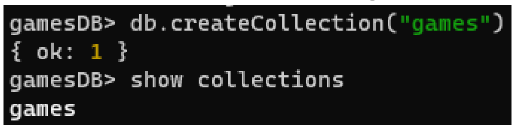
```
db> use gamesDB
switched to db gamesDB
gamesDB>
```

2. Write a query to make sure that you are using the gamesDB



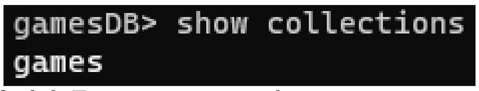
```
gamesDB> db
gamesDB
```

3. Create a new collection called games, make sure it has been created



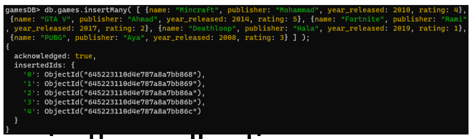
```
gamesDB> db.createCollection("games")
{ ok: 1 }
gamesDB> show collections
games
```

4. Write query to make sure that the collection was created



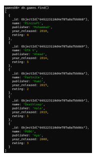
```
gamesDB> show collections
games
```

5. Add 5 games to the games collection; give each one of them has the following properties: name, publisher, year_released, and rating (value from 1 to 5)



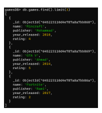
```
gamesDB> db.games.insertMany( [ {name: "Mincraft", publisher: "Mohammad", year_released: 2010, rating: 4},
 {name: "GTA V", publisher: "Ahmad", year_released: 2014, rating: 5}, {name: "Fortnite", publisher: "Rami"
, year_released: 2017, rating: 2}, {name: "Deathloop", publisher: "Hala", year_released: 2019, rating: 1},
 {name: "PUBG", publisher: "Aya", year_released: 2008, rating: 3} ] );
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("645223110d4e787a8a7bb868"),
    '1': ObjectId("645223110d4e787a8a7bb869"),
    '2': ObjectId("645223110d4e787a8a7bb86a"),
    '3': ObjectId("645223110d4e787a8a7bb86b"),
    '4': ObjectId("645223110d4e787a8a7bb86c")
  }
}
```

6. write a query to return all games in the collection



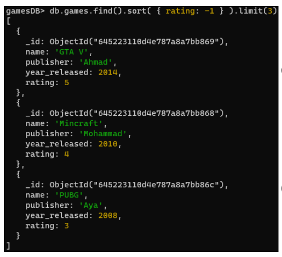
```
gamesDB> db.games.find()
[
  {
    _id: ObjectId("645223110d4e787a8a7bb868"),
    name: 'Mincraft',
    publisher: 'Mohammad',
    year_released: 2010,
    rating: 4
  },
  {
    _id: ObjectId("645223110d4e787a8a7bb869"),
    name: 'GTA V',
    publisher: 'Ahmad',
    year_released: 2014,
    rating: 5
  },
  {
    _id: ObjectId("645223110d4e787a8a7bb86a"),
    name: 'Fortnite',
    publisher: 'Rami',
    year_released: 2017,
    rating: 2
  },
  {
    _id: ObjectId("645223110d4e787a8a7bb86b"),
    name: 'Deathloop',
    publisher: 'Hala',
    year_released: 2019,
    rating: 1
  },
  {
    _id: ObjectId("645223110d4e787a8a7bb86c"),
    name: 'PUBG',
    publisher: 'Aya',
    year_released: 2008,
    rating: 3
  }
]
```

7. write a query that return only 3 games



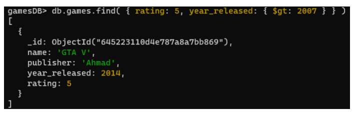
```
gamesDB> db.games.find().limit(3)
[
  {
    _id: ObjectId("645223110d4e787a8a7bb868"),
    name: 'Mincraft',
    publisher: 'Mohammad',
    year_released: 2010,
    rating: 4
  },
  {
    _id: ObjectId("645223110d4e787a8a7bb869"),
    name: 'GTA V',
    publisher: 'Ahmad',
    year_released: 2014,
    rating: 5
  },
  {
    _id: ObjectId("645223110d4e787a8a7bb86a"),
    name: 'Fortnite',
    publisher: 'Rami',
    year_released: 2017,
    rating: 2
  }
]
```

8. write a query to return the top 3 games based on rating value



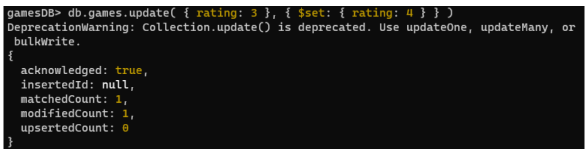
```
gamesDB> db.games.find().sort( { rating: -1 } ).limit(3)
[
  {
    _id: ObjectId("645223110d4e787a8a7bb869"),
    name: 'GTA V',
    publisher: 'Ahmad',
    year_released: 2014,
    rating: 5
  },
  {
    _id: ObjectId("645223110d4e787a8a7bb868"),
    name: 'Mincraft',
    publisher: 'Mohammad',
    year_released: 2010,
    rating: 4
  },
  {
    _id: ObjectId("645223110d4e787a8a7bb86c"),
    name: 'PUBG',
    publisher: 'Aya',
    year_released: 2008,
    rating: 3
  }
]
```

9. write a query that return games whose rating is 5 and released after 2007



```
gamesDB> db.games.find( { rating: 5, year_released: { $gt: 2007 } } )
[
  {
    _id: ObjectId("645223110d4e787a8a7bb869"),
    name: 'GTA V',
    publisher: 'Ahmad',
    year_released: 2014,
    rating: 5
  }
]
```

10. update the game whose rating is 3 to be 4



```
gamesDB> db.games.update( { rating: 3 }, { $set: { rating: 4 } } )
DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or
 bulkWrite.
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```