INTRODUCTION:

The aim of the project is to extract the well-known IMDB data base to any relational database management system; here I have used MySQL as target RDBMS to store the data. The data can be downloaded from the IMDB website and a small tool called JMDB (Java Movie Database) is used to import the data from the ftp servers that IMDB uses to MySQL.

Basic comparisons of MySQL and Mongo DB:

Schema vs. Schema less

As we all know MySQL as a relation data store requires a strict schema for its data model, all tables should be created with columns defined. Only then the data could be stored and queried using the SQL language. It somewhat complicates the development and deployment process as every time the modification of data model is required, the schema should be changed first and then data migrated.

Mongo DB does not impose any schema restriction on the documents being stored in the collection. It becomes the responsibility of application to deal with that, the only thing Mongo DB restricts is the supported data types. It significantly speeds up the development process as Mongo DB could be used right away to store JSON documents of any shape.

Advantages of MongoDB over MySQL:

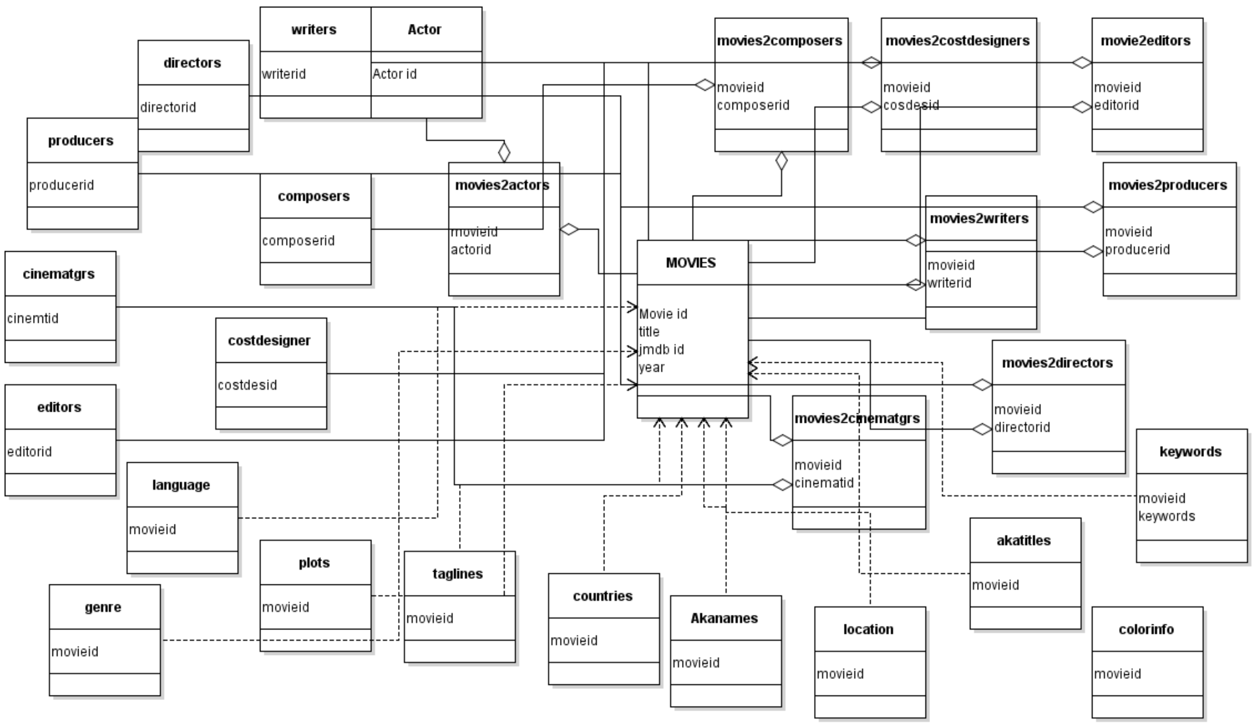
1. Fast and diverse data Types
2. Removes ORM layer
3. MongoDB is schemaless
4. Scalable

REQUIREMENT:

1. The requirement of this project is to find out the schema of the imdb database.
2. Download the database to any rdbms and then import the data from rdbms to a nosql database using a connector.
3. The data base schema must be built in the same was as how it is done in the rdbms database.
4. Then small unit test cases must be run on both the system using the same queries and a unit testing is done.

DATABASE SCHEMA:

The schema that I have found from MYSQL after importing the data from imdb database.I have used a class diagram to represent my schema so that it can be easily understandable and fit within a page.



SCHEMA EXPLANATION AND IMPLEMENTATION:

The database has two main tables' movies and actors. All the other tables have movieid as the primary key. I consider this to be the key point when forming the schema in mongo DB. Since mongo DB is a schema less database we need to form the schema within the code.

Movie table as a collection:

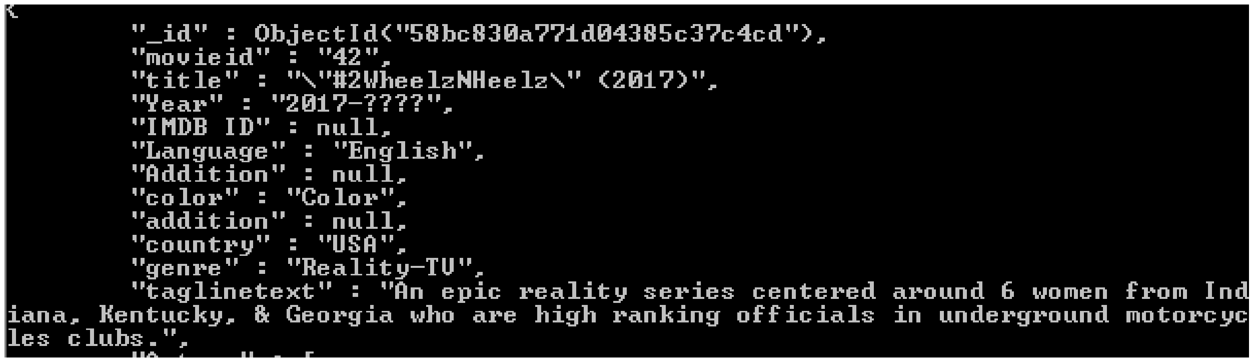
A database with the name "imdb" is created. Then a movie collection is created. The table "movies" from the mysql is then inserted into the collection using the insert command.

Tables with only movie id:

There is a list of tables that have movie id as the primary key. These tables are kept as an array.

```
String[] tableNames = {"colorinfo","countries","genres","plots","taglines"};
```

Then these tables are merged into the “movies” collection. The BasicDBObject is created each time and these tables are appended to it and then completely merged with the movies collection.



```
{
        "_id" : ObjectId("58bc830a771d04385c37c4cd"),
        "movieid" : "42",
        "title" : "\"#2WheelzNHeelz\" (2017)",
        "Year" : "2017-????",
        "IMDB ID" : null,
        "Language" : "English",
        "Addition" : null,
        "color" : "Color",
        "addition" : null,
        "country" : "USA",
        "genre" : "Reality-TV",
        "taglinetext" : "An epic reality series centered around 6 women from Ind
iana, Kentucky, & Georgia who are high ranking officials in underground motorcyc
les clubs.",
```

Tables without movie id:

There are a few tables without movie id as primary key, but have their own primary key , for example “actor id”, ”composer id” etc.

For these tables we have to look up the tables that have both movieid and other id together. These tables are:

- Movies2actors
- Movies2producers
- Movies2composers
- Movies2directors
- Movies2costdesigners
- Movies2editors
- Movies2writers
- Movies2cinematographers

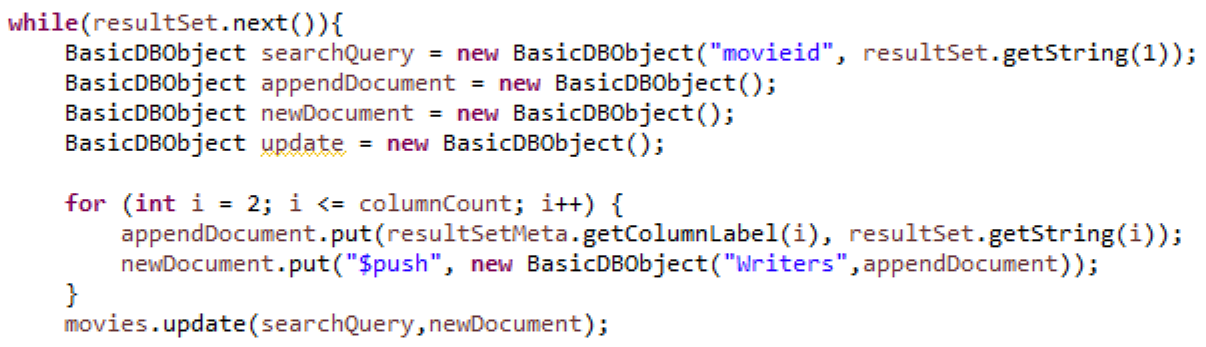
Process to embed documents into movies collection:

These tables have to inserted into the “movies” collection as an embedded document. Hence we need to use a “join” query and get a resultset from which we need to update the collection.

Sample query:

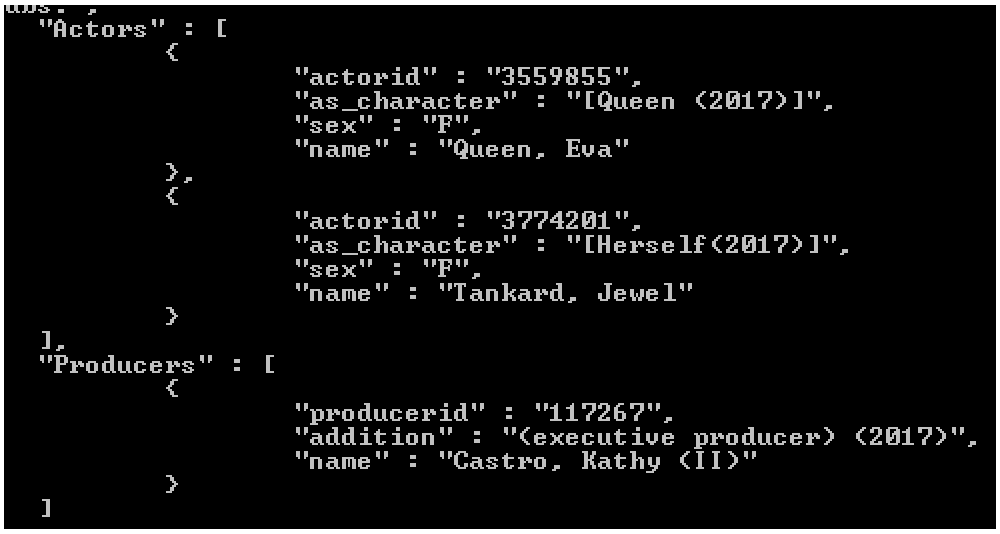
resultSet = stmt.executeQuery("select m.movieid,w.writerid,m.addition,w.name from writers as w inner join movies2writers as m on w.writerid = m.writerid order by m.movieid ");

Update query using BasicDBObject :



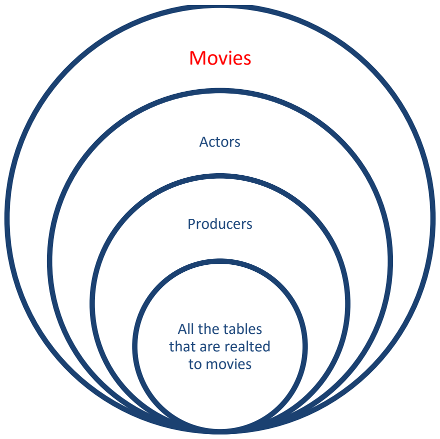
```
while(resultSet.next()){
    BasicDBObject searchQuery = new BasicDBObject("movieid", resultSet.getString(1));
    BasicDBObject appendDocument = new BasicDBObject();
    BasicDBObject newDocument = new BasicDBObject();
    BasicDBObject update = new BasicDBObject();

    for (int i = 2; i <= columnCount; i++) {
        appendDocument.put(resultSetMeta.getColumnLabel(i), resultSet.getString(i));
        newDocument.put("$push", new BasicDBObject("Writers",appendDocument));
    }
    movies.update(searchQuery,newDocument);
```

The end result after embedding the document is :



```
  "Actors" : [
          {
                  "actorid" : "3559855",
                  "as_character" : "[Queen (2017)]",
                  "sex" : "F",
                  "name" : "Queen, Eva"
          },
          {
                  "actorid" : "3774201",
                  "as_character" : "[Herself(2017)]",
                  "sex" : "F",
                  "name" : "Tankard, Jewel"
          }
  ],
  "Producers" : [
          {
                  "producerid" : "117267",
                  "addition" : "(executive producer) (2017)",
                  "name" : "Castro, Kathy (II)"
          }
  ]
```

MOVIES-A SINGLE COLLECTION



Challenges faced during the execution:

1) There were two types of coding methods in java for using the mongo driver. One of the codes was using the mongo Collection object and another was BasicDBObject.
   Eg: void com.mongodb.client.MongoCollection.insertOne (Document arg0).

Certain commands were working with BasicDBObject and a few were working with mongoCollection object. So I had to change everything to BasicDBObject. The code was working but the only downfall of the code was that it was deprecated.

2) Embedding the actors, producers, composers, writers and other tables into the movies table took a long time.

3) JMeter testing for Mongo DB was a bit tricky as the version to be used was only 2.13 and not the latest as the mongodb script is deprecated in the new version

4) The tables took a very long time time load from imdb database to MySql and again from MySql to mongo db. So only first 1000 rows are kept in the table and then the performance testing was done.

Performance Testing:

3 queries with differed level of difficulties were taken and performed on both MySql and Mongo DB

Query 1) Find movies with genre "Comedy"

Query 2) Find movies with female actors

Query 3) Find movies with genre "Comedy" and with female actors

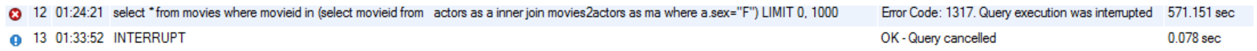
MySQL vs MongoDB

MySql Query :

1) select * from movies where movieid in (select movieid from genres where genre="Comedy") ;

2) select * from movies where movieid in (select movieid from actors as a inner join movies2actors as ma where a.sex="F");

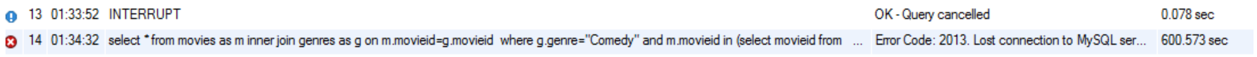
Output:



```
12  01:24:21  select * from movies where movieid in (select movieid from   actors as a inner join movies2actors as ma where a.sex="F") LIMIT 0, 1000    Error Code: 1317. Query execution was interrupted   571.151 sec
13  01:33:52  INTERRUPT                                                                                                                      OK - Query cancelled                                 0.078 sec
```

It takes a very long time to find the output even if this is the right method and MySQL informs me that LIMIT and in function is not yet implemented.

3) select * from movies as m inner join genres as g on m.movieid=g.movieid where g.genre="Comedy" and m.movieid in (select movieid from actors as a inner join movies2actors as ma where a.sex="F");



```
13  01:33:52  INTERRUPT                                                                                                                   OK - Query cancelled                           0.078 sec
14  01:34:32  select * from movies as m inner join genres as g on m.movieid=g.movieid  where g.genre="Comedy" and m.movieid in (select movieid from  ...   Error Code: 2013. Lost connection to MySQL ser...   600.573 sec
```

The same results for this query also as it has more complex join than the previous one.

Disadvantages:

The Queries here needs to be very complex similar to the 3rd query where we have to have two joins and then sub query each into another.

This is just for a simple case in MySQL as the tables here are not connected via foreign keys , instead they are given a separate table in "movies2xyz…" fashion. So we need to join each time and then sub query them.

Mongo DB Query :

1) db.movies.find({"genre":"Comedy"}).pretty();

2) db.movies.find({"Actors.sex":"F"}).pretty();

3) db.movies.find({$and:[{"genre":"Comedy"},{"Actor.sex":"F"}]}).pretty();

The queries are very easily built in mongo DB as all the information is in a single collection.

Disadvantages:

When we query by giving the condition for female actors we get the movies with female actors , but the names of male actors also get displayed , this is due to the fact that movies is a single collection and all the values will be mandatorily displayed.

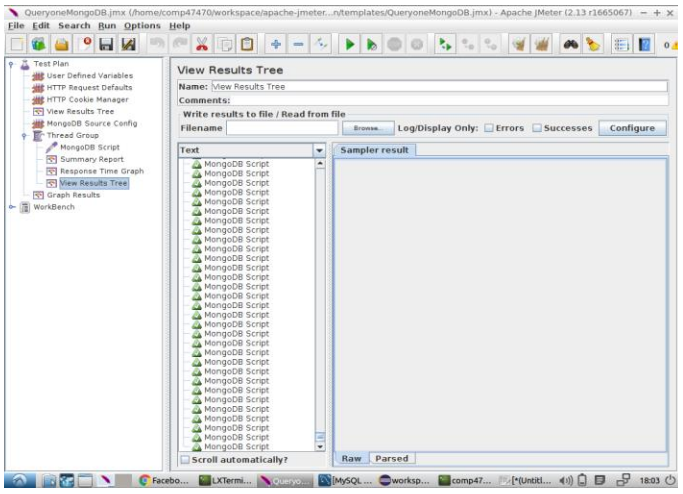
JMeter testing for the same Queries:

Mongo DB:

The following query is tested:

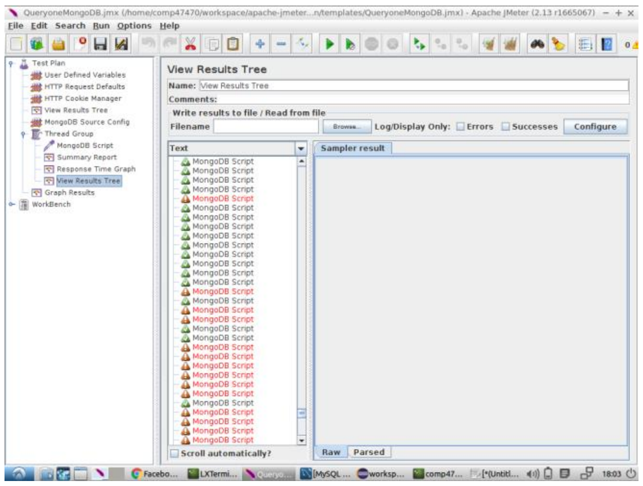
db.movies.find({$and:[{"genre":"Comedy"},{"Actor.sex":"F"}]}).pretty();

Output listeners: The mongodb graph tree results were green and running successfully initially But there were many errors in between the results


QueryoneMongoDB.jmx (/home/comp47470/workspace/apache-jmeter...n/templates/QueryoneMongoDB.jmx) - Apache JMeter (2.13 r1665067)

View Results Tree

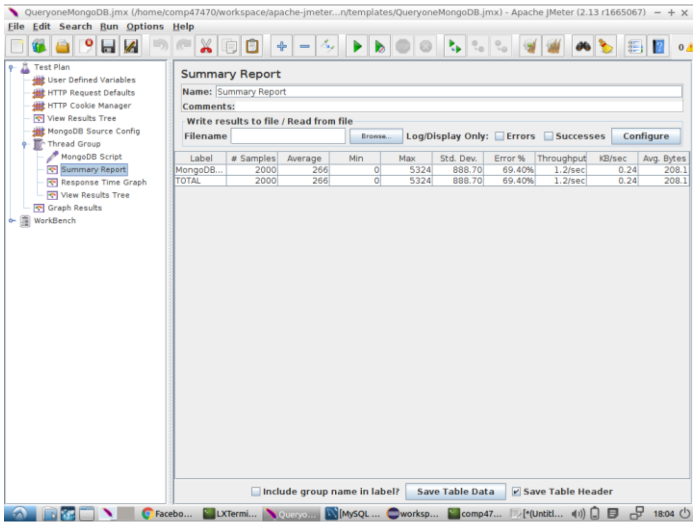
Name: View Results Tree


QueryoneMongoDB.jmx (/home/comp47470/workspace/apache-jmeter...n/templates/QueryoneMongoDB.jmx) - Apache JMeter (2.13 r1665067)

View Results Tree

Name: View Results Tree

This is also proven from the Summary report where we can see the error rate.

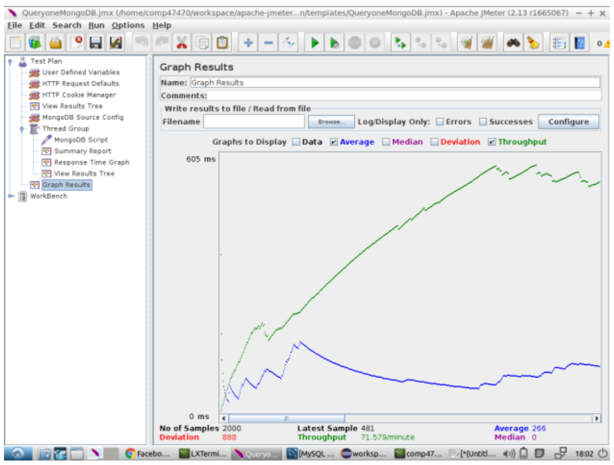


Summary Report

Name: Summary Report

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
MongoDB...	2000	266	0	5324	888.70	69.40%	1.2/sec	0.24	208.1
TOTAL	2000	266	0	5324	888.70	69.40%	1.2/sec	0.24	208.1

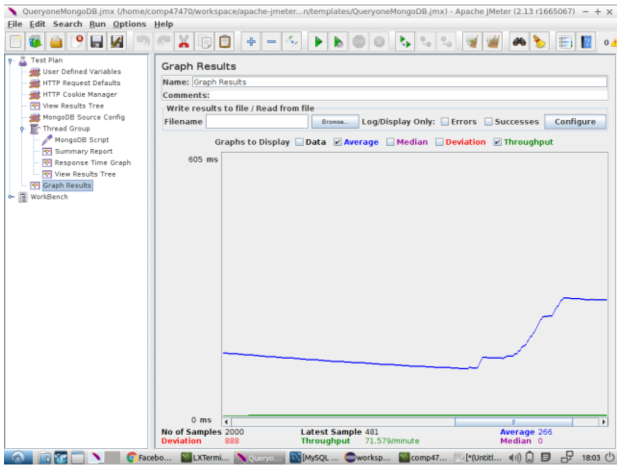
From the graph results also we can interpret that the response time initially was less but once the errors started appearing the response time went down and throughput was totally nil.



Graph Results

Name: Graph Results

No of Samples 2000 Latest Sample 481 Average 266
Deviation 888 Throughput 71.579/minute Median 0



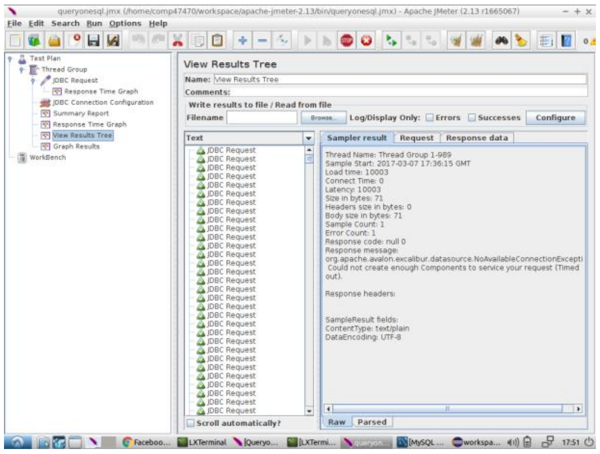
Graph Results

Name: Graph Results

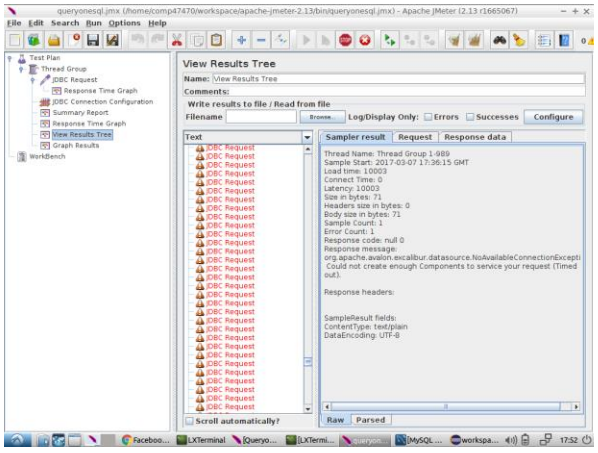
No of Samples 2000 Latest Sample 481 Average 266
Deviation 888 Throughput 71.579/minute Median 0

PERFORMANCE TESTING FOR MYSQL FOR THE SAME QUERY USED IN MySQL:

As we obtained the same result while executing the query in MySQL the same effect can be seen here.Initially the query seems to be executing, but then since the query is very complex with 2 joins each as subquery there is **Time-out issue and we get errors continuosly.**

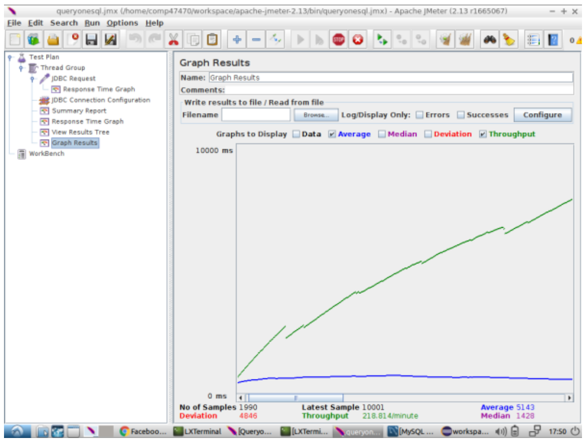


```
Thread Name: Thread Group 1-989
Sample Start: 2017-03-07 17:36:15 GMT
Load time: 10003
Connect Time: 0
Latency: 10003
Size in bytes: 71
Headers size in bytes: 0
Body size in bytes: 71
Sample Count: 1
Error Count: 1
Response code: null 0
Response message:
org.apache.avalon.excalibur.datasource.NoAvailableConnectionExcepti
 Could not create enough Components to service your request (Timed
out).

Response headers:


SampleResult fields:
ContentType: text/plain
DataEncoding: UTF-8
```

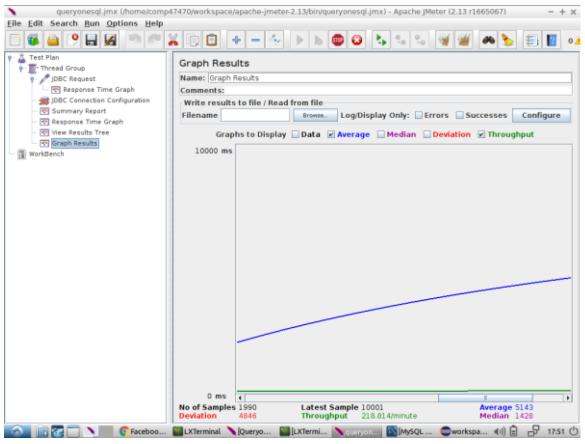


```
Thread Name: Thread Group 1-989
Sample Start: 2017-03-07 17:36:15 GMT
Load time: 10003
Connect Time: 0
Latency: 10003
Size in bytes: 71
Headers size in bytes: 0
Body size in bytes: 71
Sample Count: 1
Error Count: 1
Response code: null 0
Response message:
org.apache.avalon.excalibur.datasource.NoAvailableConnectionExcepti
 Could not create enough Components to service your request (Timed
out).

Response headers:


SampleResult fields:
ContentType: text/plain
DataEncoding: UTF-8
```

We can see the same from the graph results where initially the through put is high and response time is less but then respose time increase and we can see **nil throughput due to errors.**



```
No of Samples 1990      Latest Sample 10001                 Average 5143
Deviation     4846      Throughput    218.814/minute        Median  1428
```



```
No of Samples 1990      Latest Sample 10001                 Average 5143
Deviation     4846      Throughput    218.814/minute        Median  1428
```

Conclusion:

We can conclude from our results that for this imdb database it is easier to use mongo DB using a single collection rather than MySQL because of the complex joins and sub queries needed in MySQL which can be easily dealt in mongo DB. When we try to execute simpler queries both give us same response. But the Sql is more accurate than mongoDB because mongoDB returns all the results of that particular documents if just one of the document matches, unlike MySQL which return the exact data that we mention in the select query.

In a nutshell:

- Large and complex queries required to join data – Use MongoDB (Single collection)
- Simple queries – Depends on the storage space and the need for schema.
- Accuracy – MySQl or Mongo DB without all the data as a single collection