INTRODUCTION

MySQL is a popular open-source relational database management system (RDBMS) that is developed, distributed and supported by Oracle Corporation. In MySQL, you pre-define your database schema based on your requirements and set up rules to govern the relationships between fields in your tables. In MySQL, related information may be stored in separate tables, but relationships among tables is made through the use of joins. Data duplication is highly likely to be minimized.

Problem in RDBMS

IMPEDANCE MISMATCH

- *Division of data in a tabular way: Not natural to languages and technologies object oriented*

Solution

1. *Object Oriented Database Systems i.e. NoSQL*
2. *ORM Techniques*
    - *Object – Relational Mapping*

NoSQL Databases types:

1. *Document Oriented*
    a. *MongoDB*
    b. *RAVENDB*
2. *Column – Oriented*
    a. *Cassandra*
    b. *APACHE HBASE*
3. *Graph*
    a. *Neo4J*
4. *Key – Value*
    a. *Redis*
    b. *Riak*
    c. *Amazon S3 (Dynamo)*

MongoDB is an open-source database developed by MongoDB, Inc. MongoDB stores data in JSON-like documents that can vary in structure. Related information is stored together in a document for fast query access. MongoDB uses dynamic schemas, i.e. you can create records without first defining the structure, such as the fields or the types of their values.

TERMINOLOGY AND CONCEPTS

Many concepts in MySQL have close analogs in MongoDB. This table outlines some of the common concepts in each system.

MySQL	MongoDB
Table	Collection
Row	Document
Column	Field
Joins	Embedded documents, linking

Advantages of using MongoDB instead of MySQL

1. MongoDB enables them to build applications faster, handle highly diverse data types, and manage applications more efficiently at scale.
2. MongoDB documents map naturally to modern, object-oriented programming languages.
3. MongoDB's performance is better than that of MySQL and other relational DBs. This is because MongoDB sacrifices JOINS and other things and has excellent performance analysis tools.
4. MongoDB has a Map Reduce feature that allows for easier scalability within and across multiple distributed data centers, providing new levels of availability and scalability previously unachievable with relational databases like MySQL.

REQUIREMENT

The system supposed to complete the following task to carry out "Performance Testing of MySQL and MongoDB database."

1. Import IMDB in to MySQL
2. Extraction of IMDB from MySQL into MongoDB
   2.1 Challenges and Solutions
3. Performance testing of both Database Systems

ARCHITECTURE / DESIGN SECTION

1. Importing IMDB in to MySQL

To import IMDB in MySQL, go to www.jmdb.de and download Java Movie Database V1-40pre2p 2014-02-12 gen.zip

Step 1: extract the zip file and run startlinux shell script on terminal. Java movie database window appears

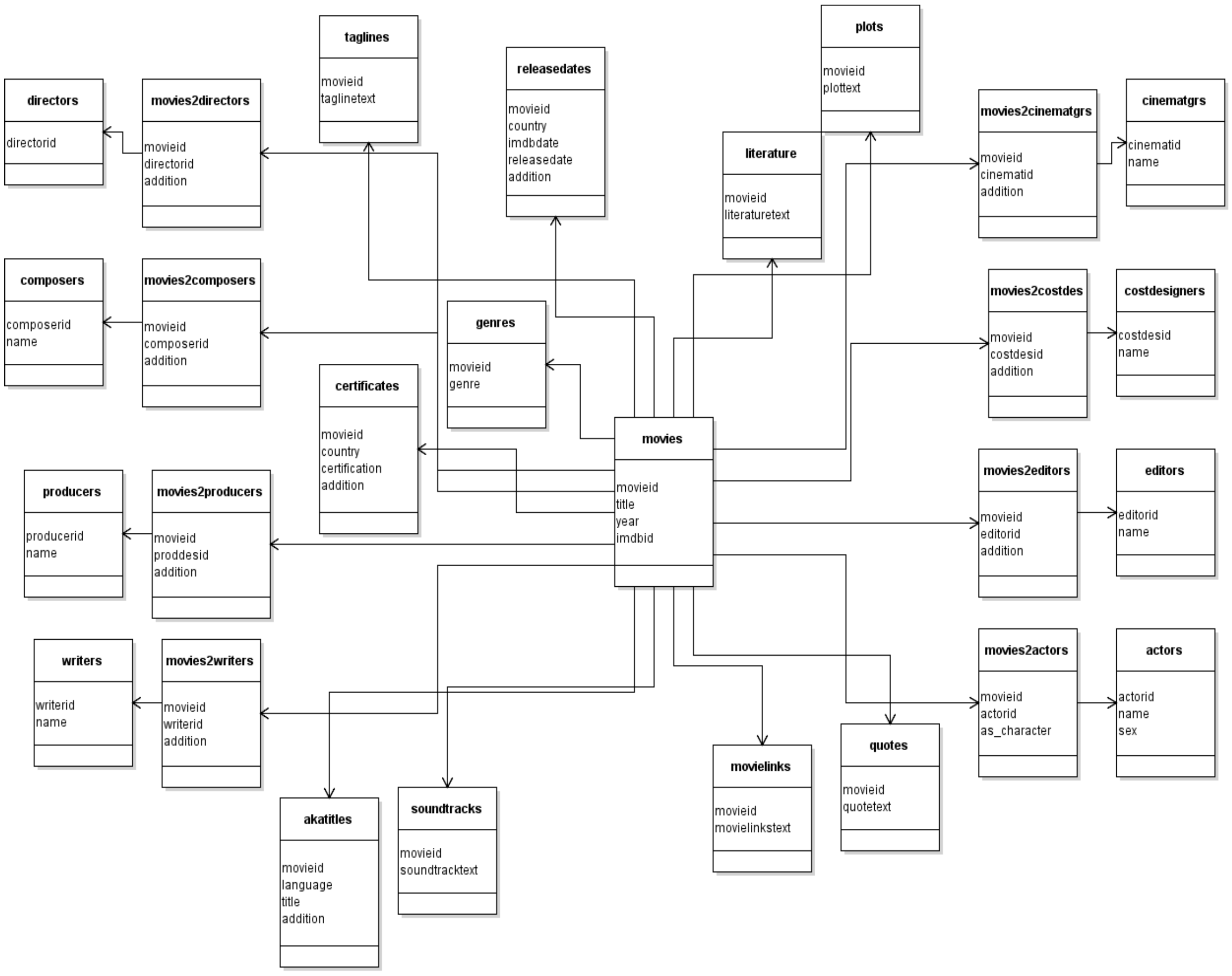
Step 2: set the correct paths and user information in the "file" -> "setup" window

Step 3: you can test if the connection works by selecting the “Test Connection” button.

Step 4: when you have setup JMDB and MySQL server is running, convert IMDB files into MySQL database. Use “File” -> “create database”

Step 5: you can use database when conversion is finished.

2. Scheme of IMDB database



3. Extraction of IMDB from MySQL into MongoDB

The following are the step to populate IMDB database into MongoDB from MySQL.

Step 1: Create a java program and add MySQL-connector-java and mongo-java-driver jar files.

Step 2: Create a MySQL database connection

```
Class.forName("com.mysql.jdbc.Driver");
```

```
String connectionURL =
"jdbc:mysql://localhost:3306/jmdb?autoReconnect=true&useSSL=false";
Connection con= DriverManager.getConnection(connectionURL,"root","mysql");
Statement stmt=con.createStatement();
```

Step 3: Executing the SELECT query, we get a Java ResultSet from that query. By iterating each row through resultset, the column values (movieid, title, imdbid, year) are stored in respective array lists.

```
//executing the query, and getting java resultset
ResultSet moviesrs =stmt.executeQuery("select * from movies LIMIT 1000");

//creating arraylist to store information from Movies table
ArrayList movieid = new ArrayList();
ArrayList title = new ArrayList();
ArrayList imdbid = new ArrayList();
ArrayList year = new ArrayList();
ResultSet moviesrs =stmt.executeQuery("select * from movies LIMIT 1000");

//iterating through the java resultset
while(moviesrs.next())
{
movieid.add(moviesrs.getString(1));
title.add(moviesrs.getString(2));
year.add(moviesrs.getString(3));
imdbid.add(moviesrs.getString(4));
}
```

Step 4: Follow the step 3 to retrieve information from other tables that have movieid column. For example,

```
//retrieving information from language table
ArrayList langmovieid = new ArrayList();
ArrayList language = new ArrayList();
ArrayList addition = new ArrayList();
ResultSet languagers = stmt.executeQuery("select * from language LIMIT 1000");
while(languagers.next())
{
langmovieid.add(languagers.getString(1));
language.add(languagers.getString(2));
addition.add(languagers.getString(3));
}
```

Step 5: To populate a MongoDB database with document corresponding to movies, create a database **"jmdb"** and a collection **"movies"** on MongoDB. Now, establish a connection to MongoDB in java program.

```
//establishing connection
MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

//getDB() method gets the jmdb database
DB db = mongoClient.getDB("jmdb");

//getCollection retrieve movies collection
DBCollection movies = db.getCollection("movies");
```

Step 6: Initially, compare movie id stored on arraylist “movieid” against movie id stored on array list on “langmovieid”. If there is a match, retrieve the relevant information from arraylists of language table and insert into the collection “movies.”

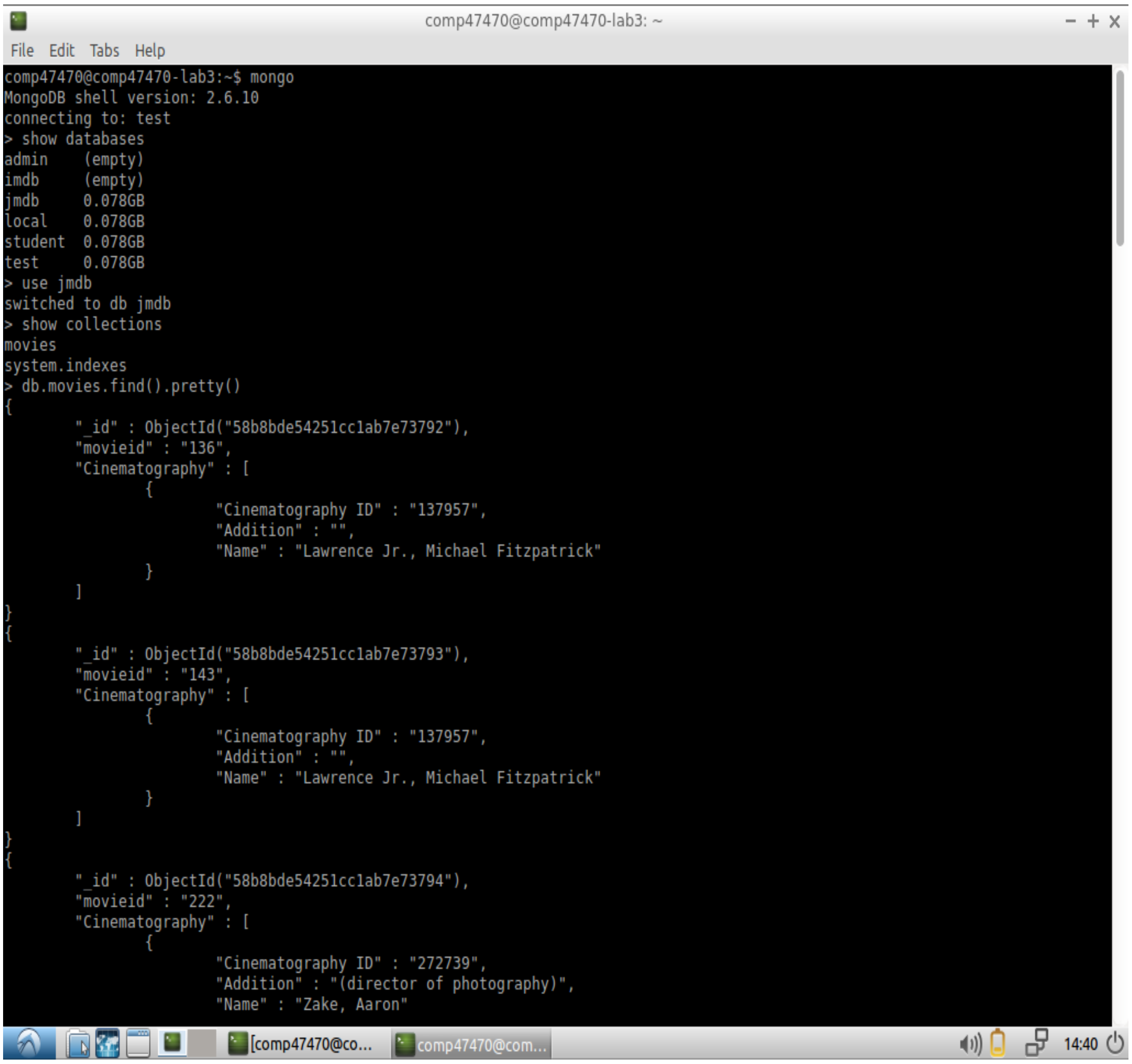
```
for(int i=0;i<langmovieid.size();i++)
{
for(int j=0;j<movieid.size();j++)
{
//comparing movie id
if(langmovieid.get(i).equals(movieid.get(j)))
{
//inserting language table information as document on collections “movies”
BasicDBObject newDocument = new
BasicDBObject("movieid",movieid.get(j)).append("title",title.get(j)).append("Year",ye
ar.get(j)).append("IMDBID",imdbid.get(j)).append("Language",language.get(i)).appe
nd("Addition", addition.get(i));
movies.insert(newDocument);
}}}
```

Step 7: Repeat the step 6 for other tables’ arraylist. This time instead of inserting the document, update the created collection if there is a match.

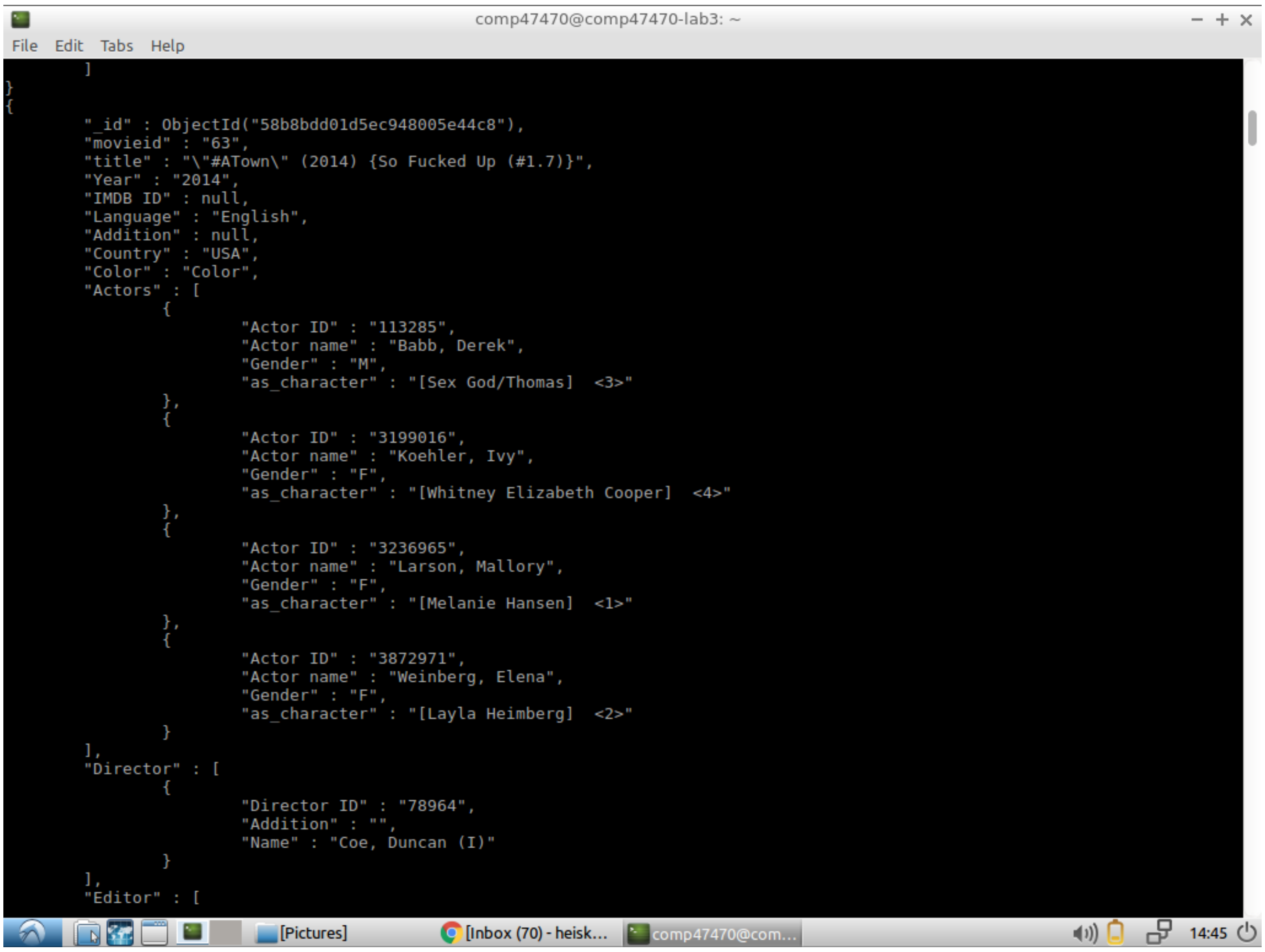
```
//retrieving information from genres table
ArrayList genresMovieID = new ArrayList();
ArrayList genres = new ArrayList();
ResultSet genresRS = stmt.executeQuery("select * from genres LIMIT 1000");

while(genresRS.next())
{
genresMovieID.add(genresRS.getString(1));
genres.add(genresRS.getString(2));
}

//updating the collection “movies” with genres information
for(int i=0;i<genresMovieID.size();i++)
{
for(int j=0;j<movieid.size();j++)
{
```

```
if(genresMovieID.get(i).equals(movieid.get(j)))
{
BasicDBObject newDocument = new BasicDBObject();
newDocument.append("$set", new BasicDBObject().append("Genres",
genres.get(i)));
BasicDBObject searchQuery = new BasicDBObject().append("movieid",
movieid.get(j));
movies.update(searchQuery, newDocument);
}
}
}
```

Step 8: Close the connection and view the collection **"movies"** created on MongoDB database **"jmdb"**



```
comp47470@comp47470-lab3:~$ mongo
MongoDB shell version: 2.6.10
connecting to: test
> show databases
admin    (empty)
imdb     (empty)
jmdb     0.078GB
local    0.078GB
student  0.078GB
test     0.078GB
> use jmdb
switched to db jmdb
> show collections
movies
system.indexes
> db.movies.find().pretty()
{
        "_id" : ObjectId("58b8bde54251cc1ab7e73792"),
        "movieid" : "136",
        "Cinematography" : [
                {
                        "Cinematography ID" : "137957",
                        "Addition" : "",
                        "Name" : "Lawrence Jr., Michael Fitzpatrick"
                }
        ]
}
{
        "_id" : ObjectId("58b8bde54251cc1ab7e73793"),
        "movieid" : "143",
        "Cinematography" : [
                {
                        "Cinematography ID" : "137957",
                        "Addition" : "",
                        "Name" : "Lawrence Jr., Michael Fitzpatrick"
                }
        ]
}
{
        "_id" : ObjectId("58b8bde54251cc1ab7e73794"),
        "movieid" : "222",
        "Cinematography" : [
                {
                        "Cinematography ID" : "272739",
                        "Addition" : "(director of photography)",
                        "Name" : "Zake, Aaron"
```



```
        ]
}
{
        "_id" : ObjectId("58b8bdd01d5ec948005e44c8"),
        "movieid" : "63",
        "title" : "\"#ATown\" (2014) {So Fucked Up (#1.7)}",
        "Year" : "2014",
        "IMDB ID" : null,
        "Language" : "English",
        "Addition" : null,
        "Country" : "USA",
        "Color" : "Color",
        "Actors" : [
                {
                        "Actor ID" : "113285",
                        "Actor name" : "Babb, Derek",
                        "Gender" : "M",
                        "as_character" : "[Sex God/Thomas]  <3>"
                },
                {
                        "Actor ID" : "3199016",
                        "Actor name" : "Koehler, Ivy",
                        "Gender" : "F",
                        "as_character" : "[Whitney Elizabeth Cooper]  <4>"
                },
                {
                        "Actor ID" : "3236965",
                        "Actor name" : "Larson, Mallory",
                        "Gender" : "F",
                        "as_character" : "[Melanie Hansen]  <1>"
                },
                {
                        "Actor ID" : "3872971",
                        "Actor name" : "Weinberg, Elena",
                        "Gender" : "F",
                        "as_character" : "[Layla Heimberg]  <2>"
                }
        ],
        "Director" : [
                {
                        "Director ID" : "78964",
                        "Addition" : "",
                        "Name" : "Coe, Duncan (I)"
                }
        ],
        "Editor" : [
```

3.1 Challenges

Challenge that I faced was to embed the information of actor table and tables similar to actor from MySQL to already created MongoDB document. In conventional database, it is a straight forward query whereas I found it difficult in MongoDB.

The solution that I prepared for this challenge was to retrieve information of actor table by comparing actorid on both “movies2actors” table and “actors” table via inner join.

> *ResultSet m2a=stmt.executeQuery(“select m.movieid,m.actorid,m.as_character,a.sex,a.name from actors as a inner join movies2actors as m on a.actorid=m.actorid order by m.movieid asc limit 1000”);*

The result of this query was stored in ArrayLists. For example,

> *ArrayList m2aMovieID = new ArrayList();*
> *ArrayList actorid = new ArrayList();*
> *ArrayList actorname = new ArrayList();*
> *ArrayList actorascharacter = new ArrayList();*
> *ArrayList actorgender = new ArrayList();*

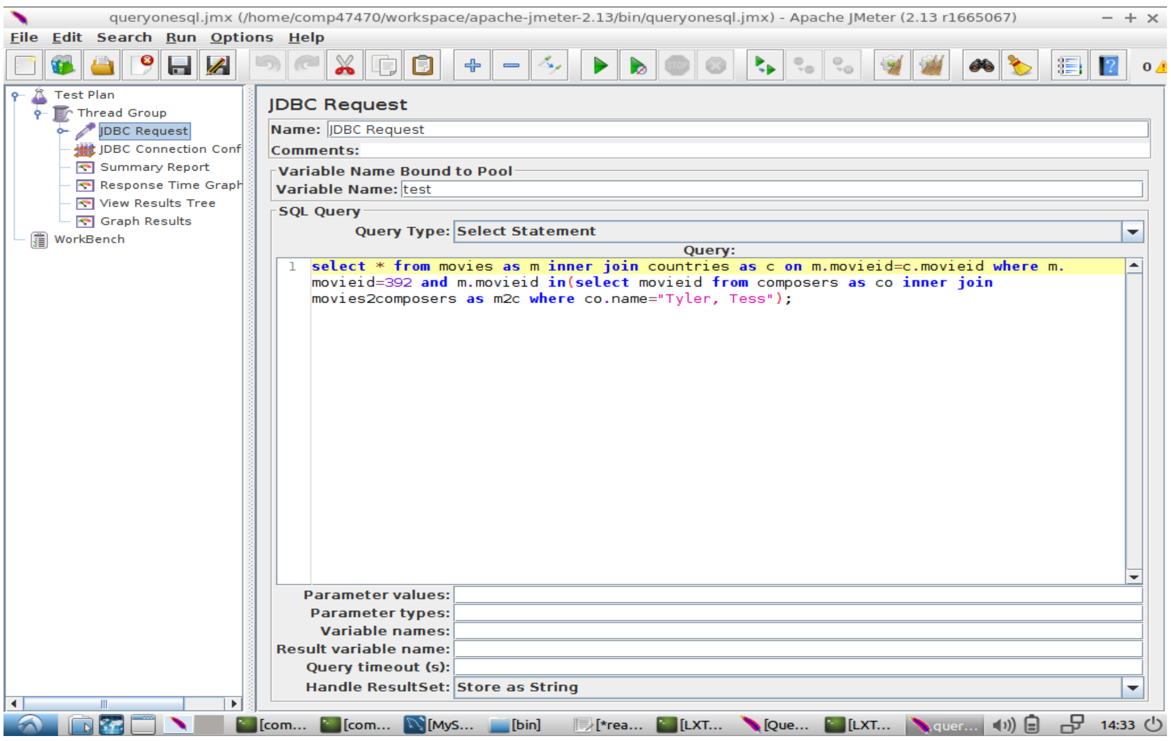
```
while(m2a.next())
{
m2aMovieID.add(m2a.getString(1));
actorid.add(m2a.getString(2));
actorascharacter.add(m2a.getString(3));
actorgender.add(m2a.getString(4));
actorname.add(m2a.getString(5));
}
```

Then, I created the document containing actor details by comparing arraylist “movieid” which is result of step 3 (refer “Extraction of IMDB from MySQL into MongoDB” above) against the arraylist “m2aMovieID”. This document was pushed to existing corresponding MongoDB collection “movies” with respect to movie id by using update( ) function.

```
//updating actor information on MongoDB document with respect to movie id
for(int i=0;i<m2aMovieID.size();i++)
{
for(int j=0;j<movieid.size();j++)
{
if(m2aMovieID.get(i).equals(movieid.get(j)))
{
BasicDBObject newDocument = new BasicDBObject();
BasicDBObject update = new BasicDBObject();
BasicDBObject query = new BasicDBObject();
query.put( "movieid",movieid.get(j) );

newDocument.put("Actor ID", actorid.get(i));
newDocument.put("Actor name",actorname.get(i));
newDocument.put("Gender", actorgender.get(i));
newDocument.put("as_character", actorascharacter.get(i));
update.put("$push", new BasicDBObject("Actors",newDocument));

/*updating movies collection with the actor data, “query” parameter in update gives
us the target collection and “update” parameter embed the actor data.*/
movies.update(query, update,true,true);
}
}
}
```

4. JMeter Performance testing of both Database Systems

JDBC Request:


queryonesql.jmx (/home/comp47470/workspace/apache-jmeter-2.13/bin/queryonesql.jmx) - Apache JMeter (2.13 r1665067)

JDBC Request

Name: JDBC Request

Comments:

Variable Name Bound to Pool

Variable Name: test

SQL Query

Query Type: Select Statement

Query:

```
select * from movies as m inner join countries as c on m.movieid=c.movieid where m.
movieid=392 and m.movieid in(select movieid from composers as co inner join
movies2composers as m2c where co.name="Tyler, Tess");
```

Parameter values:

Parameter types:

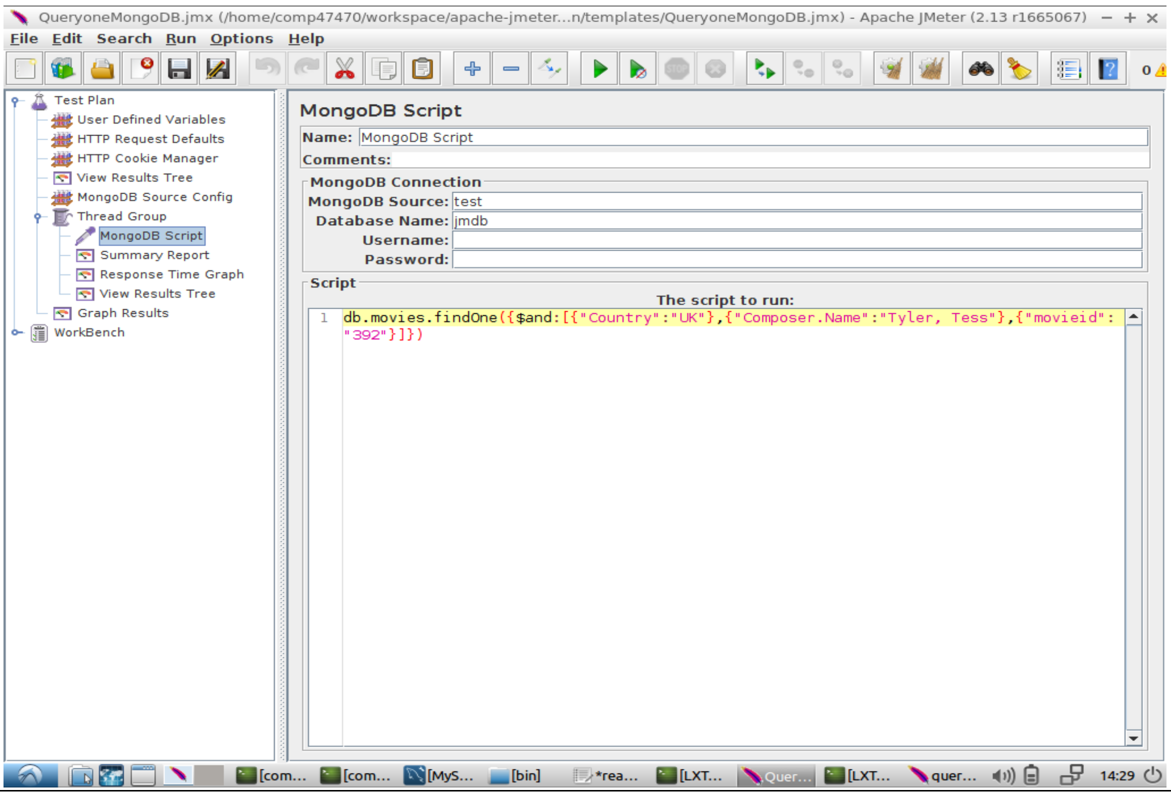
Variable names:

Result variable name:

Query timeout (s):

Handle ResultSet: Store as String

MongoDB Script:


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

MongoDB Script

Name: MongoDB Script

Comments:

MongoDB Connection

MongoDB Source: test

Database Name: jmdb

Username:

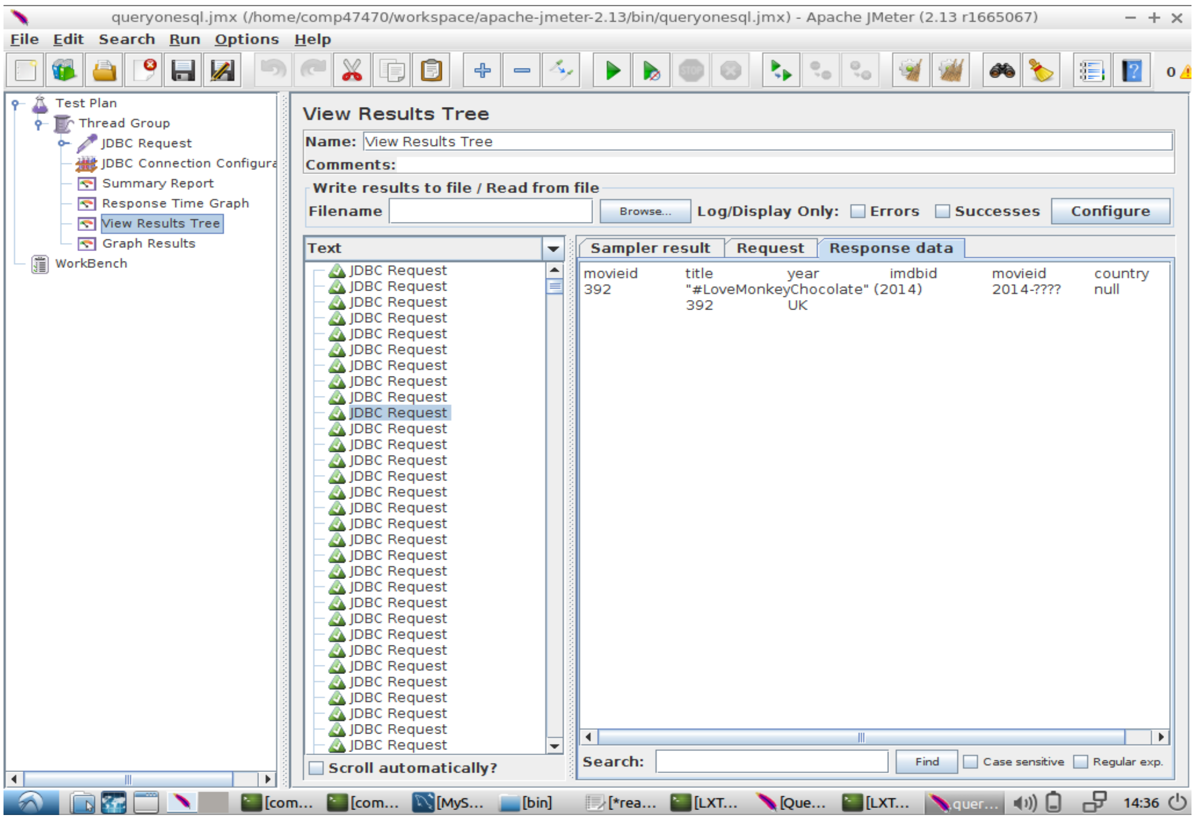
Password:

Script

The script to run:

```
db.movies.findOne({$and:[{"Country":"UK"},{"Composer.Name":"Tyler, Tess"},{"movieid":
"392"}]})
```

MySQL: “View Result Tree” screen



queryonesql.jmx (/home/comp47470/workspace/apache-jmeter-2.13/bin/queryonesql.jmx) - Apache JMeter (2.13 r1665067)

```
Test Plan
  Thread Group
    JDBC Request
    JDBC Connection Configura
    Summary Report
    Response Time Graph
    View Results Tree
    Graph Results
WorkBench
```

View Results Tree

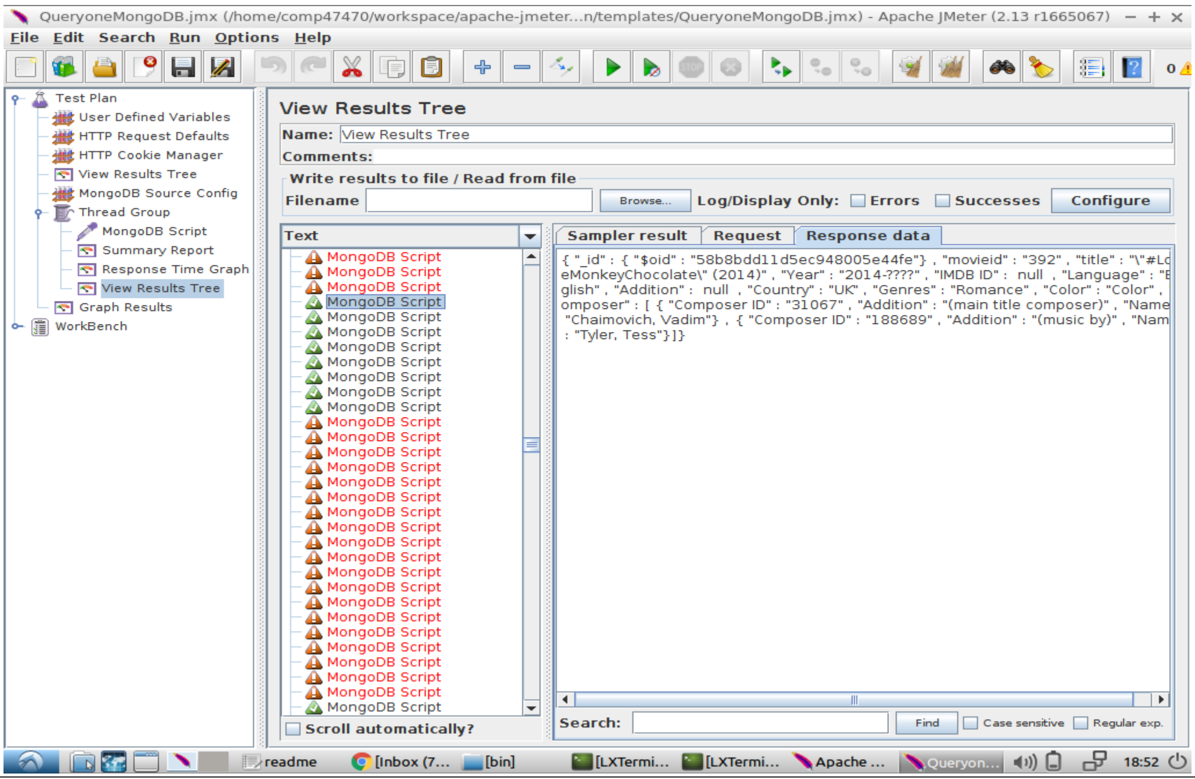
Name: View Results Tree

```
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
JDBC Request
```

Response data:

```
movieid    title                     year        imdbid      movieid     country
392        "#LoveMonkeyChocolate" (2014)                     2014-????   null
           392                       UK
```

MongoDB: “View Result Tree” screen



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

```
Test Plan
  User Defined Variables
  HTTP Request Defaults
  HTTP Cookie Manager
  View Results Tree
  MongoDB Source Config
  Thread Group
    MongoDB Script
    Summary Report
    Response Time Graph
    View Results Tree
  Graph Results
WorkBench
```

View Results Tree

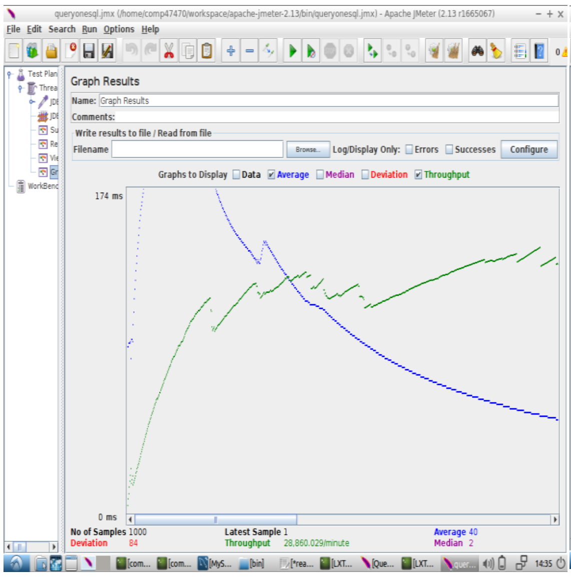
Name: View Results Tree

```
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
MongoDB Script
```

Response data:

```
{ "_id" : { "$oid" : "58b8bdd11d5ec948005e44fe"} , "movieid" : "392" , "title" : "\"#Lo
eMonkeyChocolate\" (2014)" , "Year" : "2014-????" , "IMDB ID" :  null  , "Language" : "E
glish" , "Addition" :  null  , "Country" : "UK" , "Genres" : "Romance" , "Color" : "Color" ,
omposer" : [ { "Composer ID" : "31067" , "Addition" : "(main title composer)" , "Name
 "Chaimovich, Vadim"} , { "Composer ID" : "188689" , "Addition" : "(music by)" , "Nam
 : "Tyler, Tess"}]}
```

MySQL Graph Result:

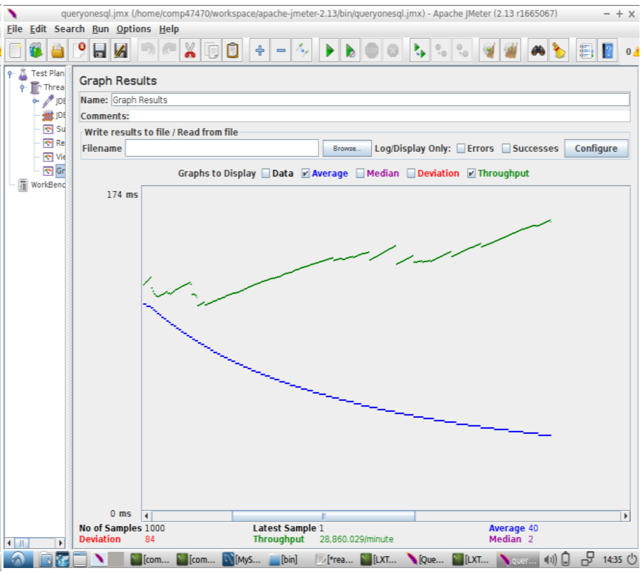

queryonesql.jmx (/home/comp47470/workspace/apache-jmeter-2.13/bin/queryonesql.jmx) - Apache JMeter (2.13 r1665067)

Graph Results

Name: Graph Results

Graphs to Display ☐ Data ☑ Average ☐ Median ☐ Deviation ☑ Throughput

No of Samples 1000 Latest Sample 1 Average 40
Deviation 84 Throughput 28,860.029/minute Median 2


queryonesql.jmx (/home/comp47470/workspace/apache-jmeter-2.13/bin/queryonesql.jmx) - Apache JMeter (2.13 r1665067)

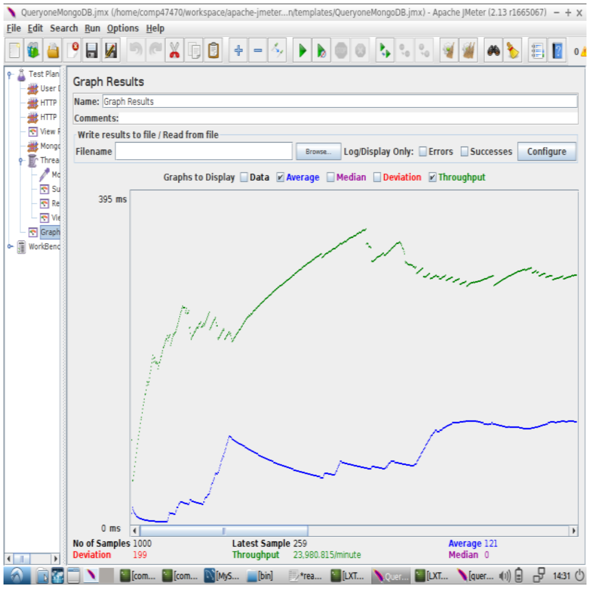
Graph Results

Name: Graph Results

Graphs to Display ☐ Data ☑ Average ☐ Median ☐ Deviation ☑ Throughput

No of Samples 1000 Latest Sample 1 Average 40
Deviation 84 Throughput 28,860.029/minute Median 2

MongoDB Graph Result:

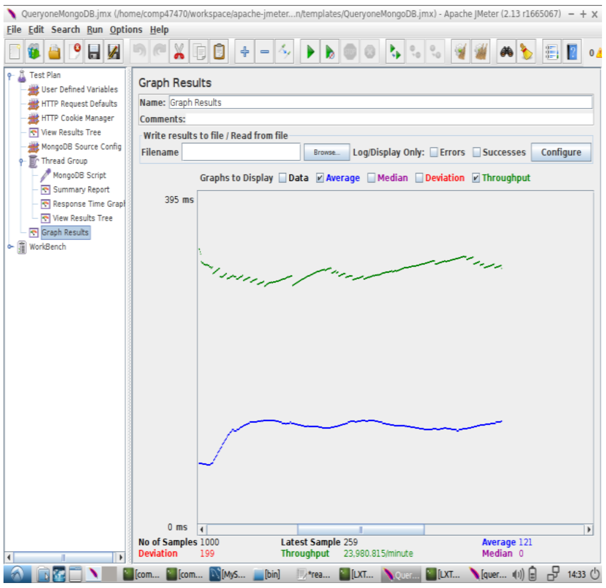

QueryoneMongoDB.jmx (/home/comp47470/workspace/apache-jmeter...n/templates/QueryoneMongoDB.jmx) - Apache JMeter (2.13 r1665067)

Graph Results

Name: Graph Results

Graphs to Display ☐ Data ☑ Average ☐ Median ☐ Deviation ☑ Throughput

No of Samples 1000 Latest Sample 259 Average 121
Deviation 199 Throughput 23,980.815/minute Median 0


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

Test Plan
User Defined Variables
HTTP Request Defaults
HTTP Cookie Manager
View Results Tree
MongoDB Source Config
Thread Group
MongoDB Script
Summary Report
Response Time Graph
View Results Tree
Graph Results
WorkBench

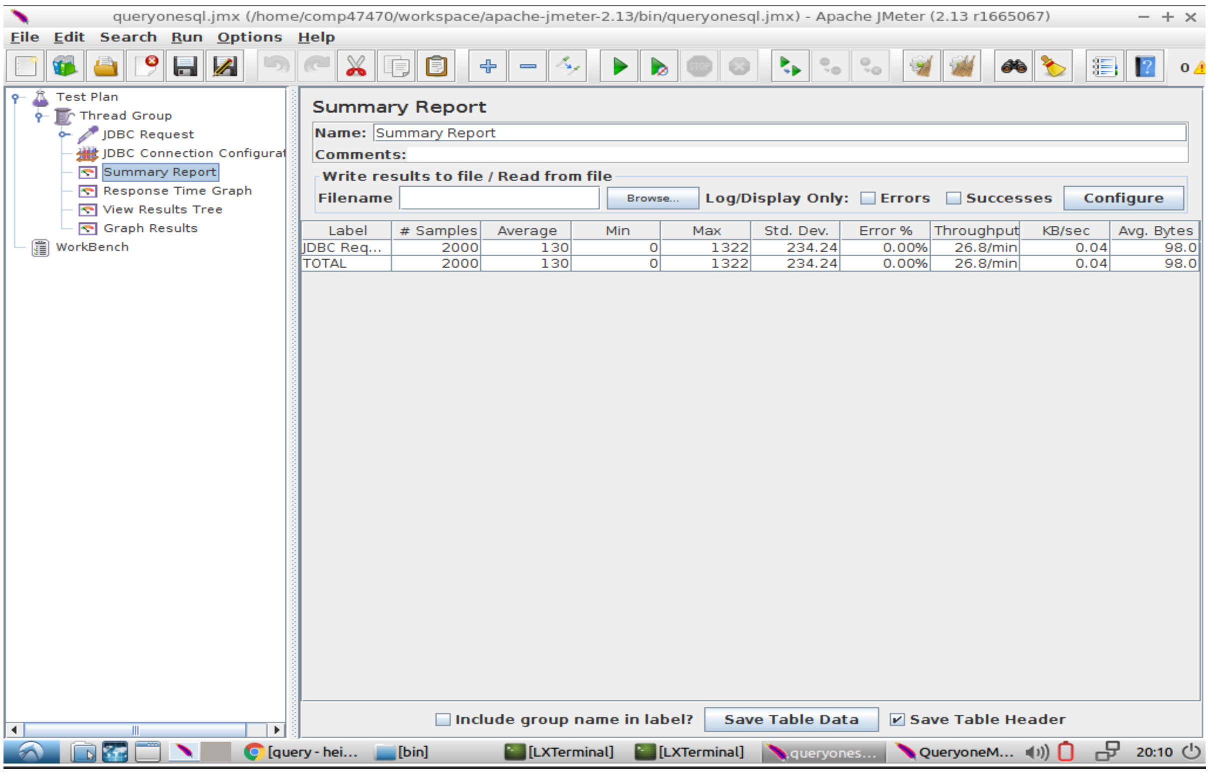
Graph Results

Name: Graph Results

Graphs to Display ☐ Data ☑ Average ☐ Median ☐ Deviation ☑ Throughput

No of Samples 1000 Latest Sample 259 Average 121
Deviation 199 Throughput 23,980.815/minute Median 0

MySQL Summary Report:



queryonesql.jmx (/home/comp47470/workspace/apache-jmeter-2.13/bin/queryonesql.jmx) - Apache JMeter (2.13 r1665067)

- Test Plan
  - Thread Group
    - JDBC Request
    - JDBC Connection Configurat
    - Summary Report
    - Response Time Graph
    - View Results Tree
    - Graph Results
- WorkBench

Summary Report

Name: Summary Report

Comments:

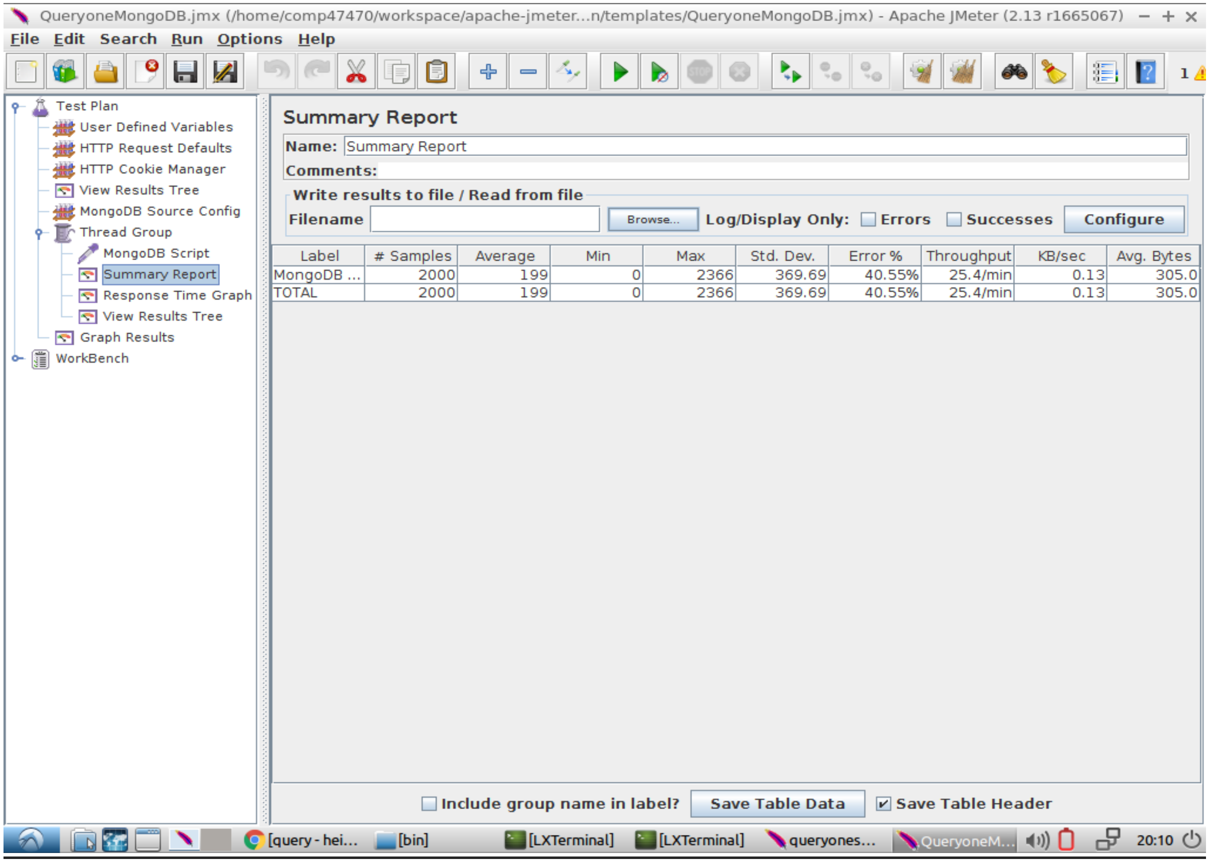
Write results to file / Read from file

Filename

Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
JDBC Req...	2000	130	0	1322	234.24	0.00%	26.8/min	0.04	98.0
TOTAL	2000	130	0	1322	234.24	0.00%	26.8/min	0.04	98.0

MongoDB Summary Report:



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

- Test Plan
  - User Defined Variables
  - HTTP Request Defaults
  - HTTP Cookie Manager
  - View Results Tree
  - MongoDB Source Config
  - Thread Group
    - MongoDB Script
    - Summary Report
    - Response Time Graph
    - View Results Tree
  - Graph Results
- WorkBench

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename

Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
MongoDB ...	2000	199	0	2366	369.69	40.55%	25.4/min	0.13	305.0
TOTAL	2000	199	0	2366	369.69	40.55%	25.4/min	0.13	305.0

CONCLUSION

After analysing the results from JMeter testing, the Error percentage is nil on MySQL compared to MongoDB whose Error percentage is around 41. In addition to this, the Throughput of MySQL is higher than the Throughput of MongoDB. However, MongoDB performed well in term of other aspects.

In conclusion, it is clear that the MySQL has the edge over MongoDB in terms of many aspects.