Evaluation code:

For 10 testcases
Number of operations: [49221, 190262, 366369, 572300, 707500, 928074, 1166732, 1326480, 1387200, 1545540]
Percentage of Products: [17,9,5,4,3,2,1,1,1,1]
As threads increase from 1 to 10.

- For the graph of ‘No. of threads’ vs ‘Number of Operations’: Number of Operations value consistently increases with number of threads. In previous assignment, Number of Operations increased at the beginning till some number of threads (4 in my case) then became constant. However, with mongoDB it continuously increases as threads increases.

- For the graph of ‘No. of threads’ vs ‘Percentage of products with a stock level less than zero’: Since I have kept initial stock to be in between 1 to 1000, Percentage value comes out to be very low. I think if we keep initial stock between 1 to 10 or 1 to 50, then percentage value would be slightly bigger. As the number of Threads increases the percentage of products(stock<0) decreases. The decrease is like linear. In previous assignment, percentage of products also decreased in random way.

MySQL has one thread per connection and MongoDB can have multiple threads from one host. Data is stored in different ways in both the databases. MySQL is optimised for join operations as they as needed for data across different tables. MongoDB doesn’t need many looksups as data can be stored at one place. In MySQL data has to be inserted row by row but in MongoDB we can do InsertMany. So, write operations are fast in MongoDB. MySQL is selecting records whereas MongoDB is quick in insert and update due to quick write operation. So, number of submitOrder, postReview, addUser, addProduct, such operations are done quickly and hence, number of operation increases with threads in MongoDB. Performance wise MongoDB is quicker as there are more inserting and update operations in our application. Also, we are storing data in such way that joins are not required, there are only 2 collections, data can be stored in such way that a document and relevant fields can be fetched just by its ‘_id’ (all related data stored at one place). Complete data scanning is avoided in MongoDB as compared to MYSQL application where joins are required, queries are also complex and sometimes need to scan entire data.. This increases the performance of this application in MongoDB.

Code:

```
AtomicInteger atomicint=new AtomicInteger(0);
    //Stores number of operations for a thread
    static List<AtomicInteger> operations =new ArrayList<AtomicInteger>();
    //Stores percentage of products whose stock is less than 0
    static List<Integer> stocklt0 =new ArrayList<Integer>();
    //Stores number of operations in each test
    static List<Integer> operationpertest =new ArrayList<Integer>();

public static void main(String[] args) throws Exception {

        //test1
        Runnable runnable1 = new NosqlMongo();
        Thread t1 = new Thread(runnable1);
        t1.start();
        t1.join();
        Bson query= lte("initialStock", 0);
            long estimatedCount = col2.estimatedDocumentCount();
            long matchingCount = col2.countDocuments(query);
        stocklt0.add((int) ((matchingCount*100)/estimatedCount));
        int sum=0;
        for(AtomicInteger ati:operations){
            sum =sum+ati.intValue();
        }
        operationpertest.add(sum);
        operations.clear();

        //test2
        Runnable runnable2 = new NosqlMongo();
        Thread t2 = new Thread(runnable2);
        Thread t3 = new Thread(runnable2);
        t2.start();
        t3.start();
        t2.join();
        t3.join();
        estimatedCount = col2.estimatedDocumentCount();
        matchingCount = col2.countDocuments(query);
        stocklt0.add((int) ((matchingCount*100)/estimatedCount));
        sum=0;
        for(AtomicInteger ati:operations){
            sum =sum+ati.intValue();
        }
        operationpertest.add(sum);
```

```
operations.clear();

//test3
Runnable runnable3 = new NosqlMongo();
Thread t4 = new Thread(runnable3);
Thread t5 = new Thread(runnable3);
Thread t6 = new Thread(runnable3);
t4.start();
t5.start();
t6.start();
t4.join();
t5.join();
t6.join();
estimatedCount = col2.estimatedDocumentCount();
matchingCount = col2.countDocuments(query);
stocklt0.add((int) ((matchingCount*100)/estimatedCount));
sum=0;
for(AtomicInteger ati:operations){
   sum =sum+ati.intValue();
}
operationpertest.add(sum);
operations.clear();

//test4
Runnable runnable4 = new NosqlMongo();
Thread t7 = new Thread(runnable4);
Thread t8 = new Thread(runnable4);
Thread t9 = new Thread(runnable4);
Thread t10 = new Thread(runnable4);
t7.start();
t8.start();
t9.start();
t10.start();
t7.join();
t8.join();
t9.join();
t10.join();
estimatedCount = col2.estimatedDocumentCount();
matchingCount = col2.countDocuments(query);
stocklt0.add((int) ((matchingCount*100)/estimatedCount));
sum=0;
for(AtomicInteger ati:operations){
   sum =sum+ati.intValue();
}
```

```
operationpertest.add(sum);
operations.clear();

//test5
Runnable runnable5 = new NosqlMongo();
Thread t11 = new Thread(runnable5);
Thread t12 = new Thread(runnable5);
Thread t13 = new Thread(runnable5);
Thread t14= new Thread(runnable5);
Thread t15 = new Thread(runnable5);
t11.start();
t12.start();
t13.start();
t14.start();
t15.start();
t11.join();
t12.join();
t13.join();
t14.join();
t15.join();
estimatedCount = col2.estimatedDocumentCount();
matchingCount = col2.countDocuments(query);
stocklt0.add((int) ((matchingCount*100)/estimatedCount));
sum=0;
for(AtomicInteger ati:operations){
   sum =sum+ati.intValue();
}
operationpertest.add(sum);
operations.clear();

//test6
Runnable runnable6 = new NosqlMongo();
Thread t16 = new Thread(runnable6);
Thread t17 = new Thread(runnable6);
Thread t18 = new Thread(runnable6);
Thread t19 = new Thread(runnable6);
Thread t20 = new Thread(runnable6);
Thread t21 = new Thread(runnable6);
t16.start();
t17.start();
t18.start();
t19.start();
t20.start();
t21.start();
```

```
t16.join();
t17.join();
t18.join();
t19.join();
t20.join();
t21.join();
estimatedCount = col2.estimatedDocumentCount();
matchingCount = col2.countDocuments(query);
stocklt0.add((int) ((matchingCount*100)/estimatedCount));
sum=0;
for(AtomicInteger ati:operations){
   sum =sum+ati.intValue();
}
operationpertest.add(sum);
operations.clear();

//test7
Runnable runnable7 = new NosqlMongo();
Thread t22 = new Thread(runnable7);
Thread t23 = new Thread(runnable7);
Thread t24 = new Thread(runnable7);
Thread t25 = new Thread(runnable7);
Thread t26 = new Thread(runnable7);
Thread t27 = new Thread(runnable7);
Thread t28 = new Thread(runnable7);
t22.start();
t23.start();
t24.start();
t25.start();
t26.start();
t27.start();
t28.start();
t22.join();
t23.join();
t24.join();
t25.join();
t26.join();
t27.join();
t28.join();
estimatedCount = col2.estimatedDocumentCount();
matchingCount = col2.countDocuments(query);
stocklt0.add((int) ((matchingCount*100)/estimatedCount));
sum=0;
for(AtomicInteger ati:operations){
```

```
    sum =sum+ati.intValue();
}
operationpertest.add(sum);
operations.clear();

//test8
Runnable runnable8 = new NosqlMongo();
Thread t29 = new Thread(runnable8);
Thread t30 = new Thread(runnable8);
Thread t31 = new Thread(runnable8);
Thread t32 = new Thread(runnable8);
Thread t33 = new Thread(runnable8);
Thread t34 = new Thread(runnable8);
Thread t35 = new Thread(runnable8);
Thread t36 = new Thread(runnable8);
t29.start();
t30.start();
t31.start();
t32.start();
t33.start();
t34.start();
t35.start();
t36.start();
t29.join();
t30.join();
t31.join();
t32.join();
t33.join();
t34.join();
t35.join();
t36.join();
estimatedCount = col2.estimatedDocumentCount();
matchingCount = col2.countDocuments(query);
stocklt0.add((int) ((matchingCount*100)/estimatedCount));
sum=0;
for(AtomicInteger ati:operations){
    sum =sum+ati.intValue();
}
operationpertest.add(sum);
operations.clear();

//test9
Runnable runnable9 = new NosqlMongo();
Thread t37 = new Thread(runnable9);
```

```
Thread t38 = new Thread(runnable9);
Thread t39 = new Thread(runnable9);
Thread t40 = new Thread(runnable9);
Thread t41 = new Thread(runnable9);
Thread t42 = new Thread(runnable9);
Thread t43 = new Thread(runnable9);
Thread t44 = new Thread(runnable9);
Thread t45 = new Thread(runnable9);
t37.start();
t38.start();
t39.start();
t40.start();
t41.start();
t42.start();
t43.start();
t44.start();
t45.start();
t37.join();
t38.join();
t39.join();
t40.join();
t41.join();
t42.join();
t43.join();
t44.join();
t45.join();
estimatedCount = col2.estimatedDocumentCount();
matchingCount = col2.countDocuments(query);
stocklt0.add((int) ((matchingCount*100)/estimatedCount));
sum=0;
for(AtomicInteger ati:operations){
   sum =sum+ati.intValue();
}
operationpertest.add(sum);
operations.clear();

//test10
Runnable runnable10 = new NosqlMongo();
Thread t46 = new Thread(runnable10);
Thread t47 = new Thread(runnable10);
Thread t48 = new Thread(runnable10);
Thread t49 = new Thread(runnable10);
Thread t50 = new Thread(runnable10);
Thread t51 = new Thread(runnable10);
```

```
        Thread t52 = new Thread(runnable10);
        Thread t53 = new Thread(runnable10);
        Thread t54 = new Thread(runnable10);
        Thread t55 = new Thread(runnable10);
        t46.start();
        t47.start();
        t48.start();
        t49.start();
        t50.start();
        t51.start();
        t52.start();
        t53.start();
        t54.start();
        t55.start();
        t46.join();
        t47.join();
        t48.join();
        t49.join();
        t50.join();
        t51.join();
        t52.join();
        t53.join();
        t54.join();
        t55.join();
        estimatedCount = col2.estimatedDocumentCount();
        matchingCount = col2.countDocuments(query);
        stocklt0.add((int) ((matchingCount*100)/estimatedCount));
        sum=0;
        for(AtomicInteger ati:operations){
            sum =sum+ati.intValue();
        }
        operationpertest.add(sum);
        operations.clear();

        //System.out.println(stocklt0);
        //System.out.println(operationpertest);


    }


@Override
    public void run() {
        Random r = new Random();
```

```
    int low = 0;
    int high = 101;
    long before = System.nanoTime();

    while(((System.nanoTime()-before)/ (1e9 * 60))<=5){
        int result = r.nextInt(high-low) + low;
        boolean success = selectmethod(result);
        if(success)
            atomicint.getAndIncrement();
    }
    operations.add(atomicint);
}


/**CreateAccount 0-3
 AddProduct 4-5
 UpdateStockLevel 6-15
 GetProductAndReviews 16-80
 GetAverageUserRating 81-85
 SubmitOrder 86-95
 PostReview 96-100
 **/
public boolean selectmethod(int prob){
    Fairy fairy = Fairy.create();
    Person person =fairy.person();
    BaseProducer base = fairy.baseProducer();
    TextProducer text = fairy.textProducer();
    boolean retval=true;
    if(prob<=3){
        String username = "user"+userid;
        String password = "password"+userid;
        String firstname =person.getFirstName();
        String lastname = person.getLastName();
        retval = createAccount(username,password,firstname,lastname);
        if(retval)
            userid++;
    }else if(prob>3 && prob<=5){
        String name = "prod"+productid;
        String description = text.sentence();
        int price = base.randomBetween(1,100);
        int initialStock = base.randomBetween(1,1000);
        retval = addProduct(productid, name, description,price,initialStock);
        if(retval)
            productid++;
```

```
        }else if(prob>5 && prob<=15){
            int itemCountToAdd = base.randomBetween(1,100);
            int productID = base.randomBetween(1,1000);
            updateStockLevel(productID, itemCountToAdd);
        }else if(prob>15 && prob<=80){
            int productID = base.randomBetween(1,1000);
            getProductAndReviews(productID);
        }else if(prob>80 && prob<=85){
            int i = base.randomBetween(1,1000);
            String usernameav = "user"+i;
            getAverageUserRating(usernameav);
        }else if(prob>85 && prob<=95){
            boolean val=false;
            int c=10;
            while(!val && c>0) {
                c--;
                int i = base.randomBetween(1, 1000);
                String username = "user" + i;
                String password = "password" + i;
                Map<Integer, Integer> listOfProductsAndQuantities = new HashMap<>();
                while (listOfProductsAndQuantities.size() < 10) {
                    int prodid = base.randomBetween(1, 1000);
                    int quant = base.randomBetween(1, 50);
                    listOfProductsAndQuantities.put(prodid, quant);
                }
                DateProducer d = fairy.dateProducer();
                LocalDateTime date = d.randomDateInThePast(50);
                val = submitOrder(orderid, false, date.toLocalDate(), username, password,
listOfProductsAndQuantities);
            }
            if(c!=0){
                orderid++;
                retval=true;
            }else{
                retval=false;
            }

        }else{
            int id = base.randomBetween(1,1000);
            String username = "user"+id;
            String password = "password"+id;
            int prodid = base.randomBetween(1,1000);
            float rating = base.randomBetween(0,10);
            String reviewText = text.sentence();
```

```
        retval = postReview(username,password, prodid, rating, reviewText);
    }
    return retval;
}
```