4/March/2025

Lab - 1

Working with MongoDB.

1) Creating DATABASE in MONGODB

Use myDB;

db;

show dbs;

II) CRUD (CREATE, READ, UPDATE, DELETE) Operations

→ db.createCollection("Student");

→ db.Student.drop()

→ db.Student.insert({_id:1, StudName: "Michellejoint ha", Grade: "VII", Hobbies: "internetsurfing"});

→ db.Student.update({_id:3, StudName: "Aryan", Grade: "VIII"}, {$set: {Hobbies: "Skating"}}, {Upset: true});

FIND METHOD

→ db.Student.find({StudName: "Aryan"});

→ db.Student.find({}, {StudName:1, Grade:1, _id:0});

→ db.Student.find({Grade: {$eq: 'VII'}}).Pretty();

→ db.Student.find({Hobbies: {$in: ['chess', 'Skating']}}).Pretty();

→ db.Student.find({StudName: /^M/}).Pretty();

db. Student. find ({ StudName: /e/ }).Pretty();

db. Student. count();

db. Student. find().sort({ StudName: -1 }).Pretty();

III import data from a csv file.

Csv file "Sample.txt"

- Import file into the MongoDB collection, "SampleJson".

The collection is in the dataset "test".

mongoimport --db Student --collection airlines --type csv --h
/home/hduser/Desktop/airline.csv

IV Export data to a csv file

The

mongoexport --host localhost --db Student --collection airlin
--csv --out /home/hduser/Desktop/output.txt --fields
"year", "Quarter"

V Save method.

db. Student. Save ({ StudName: "Vamsi", Grade:

add a new field

db. Students. Update ({_id: 4}, { $set: {location: "Netwo

remove a field

db. Students. Update ({_id: 4}, { $unset: {location:
"Network"}})

db, Student. find ({_id: 1}, { StudName: 1, Grade: 1, _id:

db. Student. find ({ Grade: { $ne: 'VII' } }). Pretty()

find doc Student name ends with s

db. Student. find ({ Stud name: /s$/ }). Pretty();

db. Students. update ({ Stud name: /s$/ }). Pretty();

→ set a Particular field value to NULL

db. Students. update ({_id: 3}, { $set: {location: null} })

→ Count the number of doc

db. students. count ({Grade: "VII"})

retrieve first 3 documents

db. Student. find ({Grade: "VII"}). limit (3). Pretty();

Lab-2

MongoDB Lab Exercises

1) Exercise, Collection Database.

① db.createCollection("Customers")

Output: {ok : 1}.

② db.Customers.insertMany([{Cust-id: 1, Acc-Bal: 1500, Acc-Type: "Z"},
{Cust-id: 2, Acc-Bal: 900, Acc-Type: "A"},
{Cust-id: 3, Acc-Bal: 1300, Acc-Type "Z"}
{Cust-id: 4, Acc-Bal: 700, Acc-Type: "B"},
{Cust-id: 5, Acc-Bal: 1600, Acc-Type: "Z"}])

Output

acknowledged: true,
insertIds: {
'0': ObjectId("67cffd77"),
'1': -11-
'2':
'3': -11-
'4':
'5':
}

③ db.Customers.find({Acc.Bal: {$gt: 1200},
Acc-Type: "Z"})

Output
{
-id: ObjectId("67cffd"),
Cust-id: 1,
Acc-Bal: 1500,
Acc-Type: 'Z'
}

① db.customers.aggregate([

{

$group: {

_id : "$Cust_id",

Min_Acc_Bal : { $min: "$Acc_Bal" },

Max_Acc_Bal : { $max: "$Acc_Bal" }

}}

])

Output:

[

{ _id : 5, Min_Acc_Bal : 1600, Max_Acc_Bal : 1600 },

{ _id : 4, Min_Acc_Bal : 700, Max_Acc_Bal : 700 }

]

② Exercise

Products database

db.createCollection("Products")

db.Products.insertMany([

{ _id: ObjectId("507f11"), name: "Laptop", Category: "Electronics", Price: 800, Quantity: 10 },

{ _id: ObjectID("507f12"), name: "Smartphone", Category: "Electronics", Price: 500, Quantity: 5 },

{ _id: ObjectId("507f13"), name: "Shoes", Category: "Fashion", Price: 50, Quantity: 20 },

{ _id: ObjectId("507f14"), name: "Tablet", Category: "Electronics", Price: 300, Quantity: 8 },

{ _id: ObjectId("507f15"), name: "T-shirt", Category: "Fashion", Price: 20, Quantity: 15 }

])

Output:

acknowledged: true,

insertIds: [

'0': ObjectId ("507f111"),

'1': ObjectId ("507f12"),

'2': ObjectId ("507f13")

ii) db.createCollection("Orders")

{ok:1}

db.Orders.insertMany([

{

_id: ObjectId ("607f121"),

user_id: "123abc",

items: [{ Product_id: ObjectId("507f112"),

Quantity:1, Price:500 }],

total_Price: 500

},

{

_id: ObjectId("607f1022"),

user_id: "123abc",

items: [{Product_id: ObjectId("5071f13"),

Quantity: 3, Price: 450 }],

total_Price: 150

},

{

_id: ObjectId("607f123"),

user_id: "u456def",

items: [{ Product_id: ObjectId("507f11"),

Quantity: 2, Price: 1600 }],

total_Price: 1600

}

])

insertedIds: {

'0' : ObjectId ("607021")

'1' : ObjectId ("607f12")

'2' : ObjectId ("607f13")

}

}

db.createCollection ("carts")

{ ok : 1 }

db.carts.insertOne({

user-id : "78aghi",

items : [

{ Product_id: ObjectId ("507f11"), Quantity: 1 },

{ Produ_id: ObjectId ("507f2"), Quantity: 2 }] })

OR

acknowledged : true,

insertedId : ObjectId ("6rdoobs")

db.Products.find()

[

{ _id : ObjectId ("507f1"),

Name : 'laptop',

Category : 'Electronics',

Price : 800,

Quantity : 10 },

{ _id : ObjectId ("50f15"),

Name : 'T-Shirt',

Category : 'Fashion',

Price : 20,

Quantity : 45 }

Output: ObjectId(

db.Products.find({Category: "Electronics"})

[

{_id: ObjectId("507f1"),
name: 'Laptop',
Category: 'Electronics',
Price: 800,
Quantity: 10},

db.Products.find({Quantity: {$gt: 0}})

[

{

_id: ObjectId("507f1"),

name: 'Laptop',

Category: 'Electronics',

Price: 800,

Quantity: 10

},

db.Products.find().sort({Price: 1})

{

_id: ObjectId("507f1"),

name: 'Fashion';

Price: 20

Quantity: 15},

db.Products.find({Price: {$lte: 100}})

[{

_id: ObjId("507f1")

name: 'Shoes',

Category: 'Fashion',

Price: 20,

db.carts.aggregate([

{ $match: { user_id: "789hi" } },

{ $unwind: "$items" },

{ $lookup: {

from: "products",

as: "productsInfo"

} },

])

or.

{

_id: ObjId("6071

user_id: "123abc",

items: [

{ prod_id: ObjId("5071"))

price: 500

},

db.orders.aggregate([

{ $match: { user_id: "123abc" } },

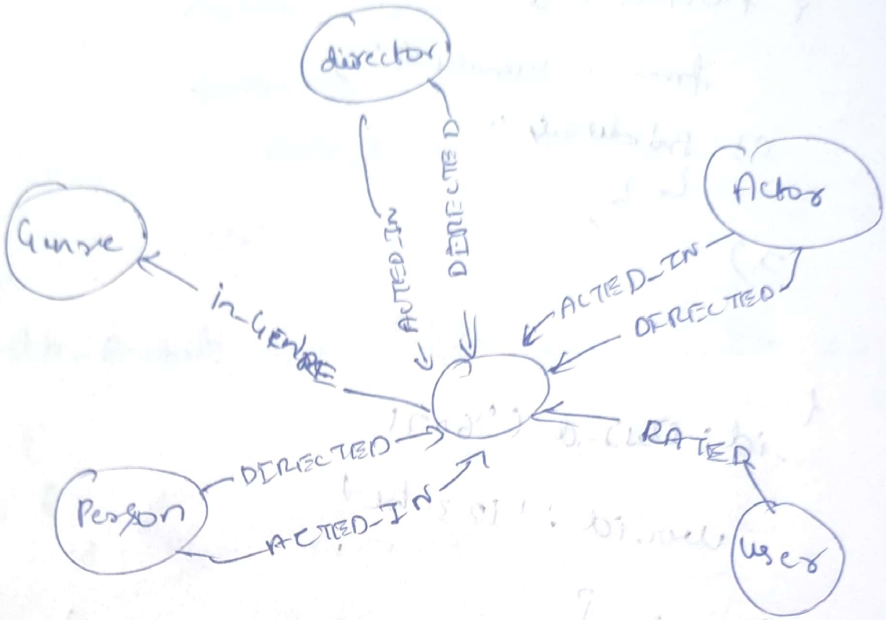
{ $group: { _id: $user_id, total_sum: { $sum: $total_price } } }])

db.o

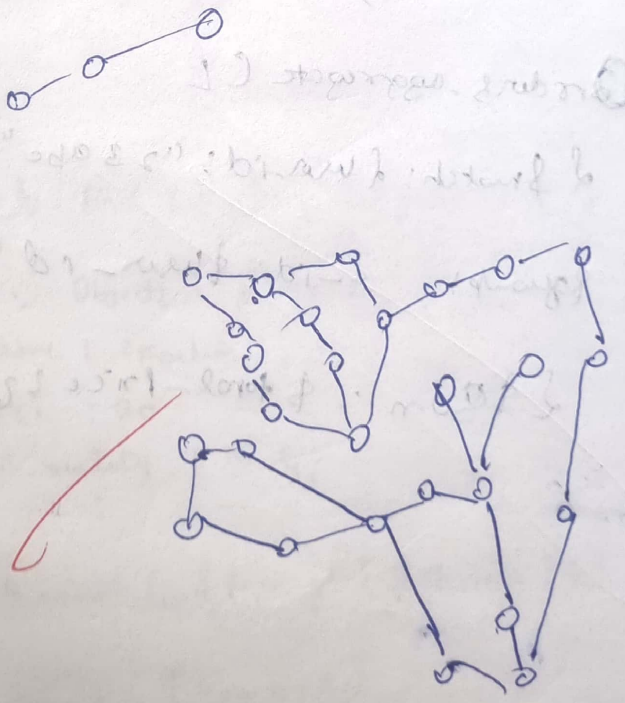
11/3/25

Lab 3 (NEO 4J)

1) call . db . schema . visualization



2) MATCH (n) return n limit 100



1/4/2025

Lab-4

Exploring operations on NOSQL - Cassandra.

Create keyspace:

→ CREATE KEYSPACE Students WITH REPLICATION =
{'class': SimpleStrategy', 'replication_factor': 1};

KeySpace Created Successfully

→ DESCRIBE KEYSPACES;

Students System_auth System_schema System_views
System System_distributed System_traces System_virtual_schema

→ SELECT * FROM System_schema.keyspaces;

keyspace_name	durable_writes	replication
system_auth	True	{'class': 'org.apache.cassandra.locator.S
imblestrategy', 'replication_factor': '1'}		

(6 rows)

→ USE Students;

CREATE TABLE Student_Info (ROLL_NO int PRIMARY KEY, StudName text, DateOfJoining timestamp, last_exam_Percent double);

→ DESCRIBE TABLES;

Student_Info user_login

→ DESCRIBE TABLE Students_Info;

/* Description of Table */

BEGIN BATCH
INSERT INTO Students_info (Roll_NO, Stud Name,
Date of Joining, last_exam_Percent
Values (1, 'Asha', 2012-03-12', 79.9)
;

INSERT INTO Students_info (Roll_NO; Stud Name,
Date of Joining, last_exam_Percent)
VALUES (1, 'Rohan', '2012-03-12', 56.9)

→ SELECT * FROM Students_Info;

→ SELECT * FROM Students_Info WHERE Roll_NO
IN (1, 2, 3);

→ SELECT * FROM Students_Info WHERE
STUD Name = 'Asha';

→ CREATE INDEX ON STUDENTS_Info (Stud Name

→ select * from Students_info where Stuname = 'Asha';

→ Select Roll_NO As "USN" from Students_Info;

→ UPdate Students_Info Set Studname = 'David Stein'
where roll NO = 2;

→ DELETE FROM Student_info WHERE Roll_No = 2;

ALTER TABLE Student_info ADD hobbies Set <text>;

UPDATE Student_info
SET hobbies = hobbies + {'chess', 'Table Tennis'}
WHERE Roll_No = 1;

List collection :

ALTER TABLE STUDENTS_Info Add long list <text>;

Update Stud_info
Set long = [Hindi, English]
where roll no = 1;

Counter Table

create table library_book (
book_name varchar,
Stud_name varchar
);

Time To Live

create Table user_login (
user_id
Password text
);

12/4/25

Lab – 5

Cassandra Exercise.

1) Create a keyspace by name Library

```
Library WITH REPLICATION = {'class': 'SimpleStrategy',
'replication_factor': 1};
```

2) USE Library ;

```
CREATE TABLE Library_Info (
    Stud_Id int PRIMARY KEY,
    Counter_value counter,
    Stud_Name text,
    Book_Name text,
    Book_Id int,
    Date_of_issue timestamp );
```

3) BEGIN BATCH

```
INSERT INTO Library_Info (Stud_Id, Counter_value,
Stud_Name, Book_Name, Book_Id, Date_of_issue)
VALUES (111, 1, 'Alice', 'Introduction to Cassandra',
101, '2025-04-10 10:00:00+0000');

INSERT INTO Library_Info (Stud_Id, Counter_value,
Stud_Name, Book_Name, Book_Id, Date_of_issue)
VALUES (112, 1, 'Bob', 'DBA', 205, '2025-04-11
14:30:00+0000');

INSERT INTO Library_Info (Stud_Id, Counter_value,
Stud_Name, Book_Name, Book_Id, Date_of_issue)
VALUES (113, 4, 'David', 'NOSQL', 206,
'2025-06-11', 14:30+007 0000');
```

4) DESCRIBE TABLE Library_Info

output:

```
CREATE TABLE Library_Info (
    Stud_Info int PRIMARY KEY,
    Book_Id int,
    Book_Name text,
    Counter_value Counter,
    Date_of_issue timestamp,
    Stud_Name text );
```

5) Increase counter val for Student with Stud_id = 112;

```
UPDATE Library_Info SET Counter_value = Counter_value
    WHERE Stud_Id = 112;
```

```
SELECT Stud_Id, Counter_value, Stud_Name,
Book_Name, Book_Name FROM Library_Info
WHERE Stud_Id = 112;
```

Output:

Stud_id	Counter_value	book_name	Stud_name
112	2	BDA	Bob

) CREATE TABLE Library_Book_Tracking (

```
    Stud_Id int,
    Book_name text,
    Issue_number, int,
    Date_of_issue timestamp,
PRIMARY KEY ((Stud_Id, Book_Name),
```

SELECT * FROM Library_info WHERE Stud_Id =
112 AND Book_Name = 'BDA' ;

Stud_id	book_id	book_Name	Counter value	date_issued
112	205	BDA	2	2025-04-1
112	205	BAA	2	2025-04-1

(2 rows)

b) Export

COPY Library.Library_Info TO '/Path/to/your/annual_subscription.csv'
With Header = TRUE ;

"SELECT Stud_Id, Counter_value, Stud_Name,
Book_Name, Book_Id FROM Library.Library_info

1) Stud_Id, Counter_value, Stud_Name, Book_Name,
Book_Id, Date_of_issue

201, 1, "Eve", "DS", 305, "2025-04-15"
202, 2, "Frank", "Algorithms", 308, "2025-03,4"

COPY Library.Library_Info (Stud_Id, Counter_val,
Stud_Name, Book_name, Book_Id,
From '/Path/to/your/import_data.csv',
With Header = TRUE;

output Processed 2 rows in 0.xxx Seconds

Lab-6
Hadoop

→ $ start-all.sh
→ $ hdfs dfs -mkdir /bda_hadoop
→ hdfs dfs -ls /

found 1 items
drwxr-xr-x - hduser supergroup

→ hdfs dfs -ls /bda_hadoop/

found 1 items
-rw-r--r-- 1 hduser supergroup

→ hdfs - dfs - cat /bda_hadoop/file.txt

→ hdfs -get /abc/WC.txt /home/hduser/Downloads/annual income.csv.

ls /home/hduser/Downloads/

6) → hdfs -dfs -copyToLocal /abc/annual.csv /home/hduser/Desktop/

ls /home/hduser/Desktop/

hdfs dfs -cat /abc/annual.csv /

→ OP: drwxr-xr-x - hduser supergroup

→ hadoop fs -cp / cpe / lll.

hadoop fs -ls / LLL

Op. bound : items

drwxr-xr-x -hdusers Subgroups

0 YYYY-MM-DD HH:M/LLL

15/4/25

6/may/25

Scala

Scala

scala> 8 * 5 + 2
res0: Int = 42

Scala > 0.5 * res0
res1 : double = 21.0

Scala > "Hello," + res0
res2 : String = Hello, 42

Scala > val answer = 8 * 5 + 2
answer : Int = 42

Scala > ~~val answer~~ 0.5 * answer
res3 : Double = 21.0

Scala > var Counter = 0
Counter : Int = 0

Scala > counter = 1
Counter : Int = 1

Scala > val greeting : String = null
greeting : String = null

Scala > val greeting : Any = "Hello"
greeting : Any = Hello

Scala > val xmax, ymax = 100
xmax : Int = 100
ymax : Int = 100

Scala > var greeting, message : String = null
greeting : String = null
message : String = null

Scala > toString()

res4 : String = 29c24ce

Scala > toString(10)

Scala > "Hello" .intersect ("world")

res7 : String = lo

Scala > 1 to 10

res8 : Scala.collection.immutable.Range.

Inclusive = Range (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

Scala > Scala.math.sqrt(2)

res10 : Double = 1.4142

Scala > import Scala.math

import Scala.math_

Lab-7

TOPN

Create a Mapreduce Program to find average temp for each year from NCDC data set Package Sample.txt :

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable
import org.apache.hadoop.io.Text;
import org.apache.mapreduce.Mapper;
import org.apache.mapreduce.lib.input.FileInput
import org.apache.mapreduce.lib.output.FileOutput
import org.apache.mapreduce.util.GenericOptionsParser;
Public class TopN {
  Public static void main (String[] args) throws
Exception {
  Configuration conf = new Configuration();
  String [] otherArgs = (new GenericOptionsParser
    (conf,args)).getRemainingArgs();
    if (otherArgs length != 2) {
      System.err.println("Usage: TopN <in>
      <out>");
      System.exit(2);
  }
  Job job = Job.getInstance(conf);
  job.setJobName("TopN");
```

```
job. Set JarByClass (TopN.class)
job. Set MapperClass (TopN MapperClass);
job. Set ReducerClass (TopN Reducer.class);
job. Set Output KeyClass (Text.class);
job. Set output value class (IntWritable.class);
File input format.add input path (job,
new Path (other Args [1]));
System.exit (job.wait for completion (true) ? 0 : 1);
}

Public Static class TopN Mapper Extends Mapper
<Object, Text, Text, intwritable> {
Private static final intwritable one = new intwritable (1);
Private Text word = new Text();
Private String tokens = "[ | $# <>\\^=\\[\\ ].\\*\\ \\;\\-( )?| \\*]";

Public void map (Object key, Text value, Mapper
<Object, Text, Text, intwritable> Context)
throws IOException, interrupted Exception {
String CleanLine = value.to String ( ). toLower
Case ( ) replace All (this. token " ");
}
}
}
```

Top N Reduce: Using.

```
Package Sambler.TopNN;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.Reduce;
import Util.MapUtils;

Public class TopNReducer extends Reducer
<Text, IntWritable, Text, IntWritable>
Private Map<Text, IntWritable> CountMap=
    new HashMap<>();

Public void reduce (Text key, IterableWritable
<IntWritable> values, Reducer<Text, IntWritable,
Text, IntWritable> context) throws
IOException, InterruptedException {
    int sum=0;
for (IntWritable val; values)
sum+= val.get();
this.CountMap.put(new Text(key),
new IntWritable(sum));
}
```

b) Mean max temperature for every month

```
Package meanmax;
Public class MeanMaxDriver {
Public static void main (String [] args) throws
Exception {
    if (args.length != 2) {
        System.exit(-1);
    }
    Job job = new Job();
    Job.setJarByClass (MeanMaxDriver.class);
file input format.addInputPath (job, new Path (args[0]);
System.exit (job.waitForCompletion (true) ? 0 :1);
}
}
```

```
Mean Max Mapper.class.
Package meanmax;
Public class MeanMaxMapper extends
Mapper <LongWritable, Text, Text, IntWritable> {
Private Map <Text, IntWritable> countMap = new
    HashMap <>();
Public void reduce (Text key, Iterable <IntWritable>
values, Reducer (Text, IntWritable, Text,
IntWritable > Context Context)
Throws IOException, InterruptedException {
    int sum = 0;
for (IntWritable Val : values)
    sum += val.get();
this.countMap.put (new Text (key),
    new IntWritable (sum));
}
```

```
Protected void cleanup (Reducer <Text, IntWritable
Text, IntWritable > context Context)
throws Exception {
    map < Text, IntWritable > SortedMap =
    MapUtil . SortByValues (this.countMap);

    int Counter = 0;
    for (Text key: SortedMap . keySet()){
    if (Counter++ == 20)
      break;
    context . write (key, SortedMap.get(key))
    }
}
```

Lab-8

① Write a Scala Program to Print number from 1 to 100 using for loop

> sudo apt install scala

> nano PrintNumbers.scala

```
object PrintNumbers {
  def main(args: Array[String]): Unit = {
    for (i <- 1 to 100) {
      Print(i)
    }
  }
}
```

> scala PrintNumbers.scala

> scala PrintNumbers

Output: 1
:
100

(Numbers Printed from 1 to 100)

e) Using RDD and Flatmap Count how many times each word appears in a file & write out a list of words whose count is strictly greater than 4 using spark.

> echo -e "hello world \n hello spark \n hello scala \n spark is fast \n hello spark spark spark" > input.txt

> spark-shell

> val rdd = sc.textFile("input.txt")
val words = rdd.flatMap(line => line.split("\W+"))
val pairs = words.map(word => (word.toLowerCase(),1))
val counts = pairs.reduceByKey(_+_)
val filtered = counts.filter{case(word, count) => count > 4}
filtered.collect().foreach{case(word,count) => println(s"$word: $count")}

output:
spark: 6

~~3) Write a Simple Streaming Program in~~

3) Map reduce Program to sort the Content in a alphabetic order listing only top 10 max occurences of words (Hadoop Prg).

WordCount Mapper.java

```
import java.io.IOException;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Mapper;

Public class WordCount Mapper extends Mapper
<Long Writable, Text, Text, IntWritable> {
Private final Static IntWritable One = new IntWritable(1);
    Private Text word = new Text();
Public void  = new Text();

Public void map (Long Writable key, Text value
    Context, Context)
throws IOException, InterruptedException {
String line = Value.toString().toLowerCase(),
replaceAll("[^a-zA-Z]", " ");
    String [] words = line.split("\\s+");
    for (String w: words) {
        if (!w.isEmpty()) {
            word.set(w);
            Context.write(word, One);
        }
    }
}
```

```
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Reducer;
Public class WordCountReducer extends Reducer<Text,
IntWritable, Text, IntWritable> {
    Private Map<Text, IntWritable> CountMap = new
        HashMap<>();
Public void reduce(Text key, Iterable<IntWritable>
    values, Context context)
    throws IOException, InterruptedException {
        int sum = 0;
    for (IntWritable val: values) {
        sum += val.get();
    }
CountMap.put(new Text(key), new IntWritable
        (sum));
    }
Protected void cleanup(Context context) throws
    IOException, InterruptedException {
    CountMap.entrySet().stream()
    .sorted((e1, e2) -> {
    int cmp = e2.getValue().compareTo(e1.getValue
        ());
If (cmp == 0) return e1.getKey().toString().
    compareTo(e2.getKey().toString());
        return cmp;
    })
    .limit(10)
    .forEach(e -> {
```

```
Context.write(e.getKey(), e.getValue());
} catch (IOException | InterruptedException ex) {
    throw new RuntimeException(ex); }
});
```

WordCountDriver.java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;

public class WordCountDriver {
public static void main(String[] args)
throws Exception {
    Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "job 10
    WordCount");

    job.setJarByClass(WordCountDriver.class);
    job.setMapperClass(WordCountMapper.class);
    job.setReduceClass(WordCountReducer.class);
    job.setOutputKeyClass(Text.class);

FileInputFormat.addInputPath(job, new Path
    (args[0]));

System.exit(job.waitForCompletion(true)
    ? 0 : 1);
}
}
```

OP: hadoop

and	1
an	1
are	1
big	1
data	2
framework	1
hadoop	3
is	1
large	1
amount	1

4) (Open Ended)

Write a Simple Streaming Prog in Spark to receive text data Streams on a particular Space removal Stop words removal, lemmatization, etc) and Print cleaned text on screen

Terminal A:

> Spark-shell

> import org.apache.spark.streaming._
import org.apache.spark.streaming.Context._

val ssc = new StreamingContext(sc, second(5))

val stopwords = Set("a", "an", "the", "is", "are", "and", "or", "to", "in")

val lines = ssc.socketTextStream("localhost", 9999)

```
val cleaned = lines.flatmap ("localhost, 9999)
val cleaned = lines.flatmap (_split (" ")
.map (_.to lower case.trim)
.filter (word => word.nonEmpty && !stopwords
.contains (word))
cleaned.print()
ssc.start()
ssc.await Termination();
```

Terminal B

→ nc -lk 9999

Terminal A: (run the Py Prg)

$ spark-submit wordcount.Py

Terminal B

The quick brown fox jumps over the lazy dog

Terminal A:

```
the
quick
brown
fox
jumps
over
the
lazy
dog
```