Lab-1 :-

① Creating Database & CRUD Operations :-

```
> use myDB;
> db;
myDB.
> db.createCollection ("Student");
{ ok: 1 }
> db.Student.insert ({ _id: 1, StudName: "Rev", Grade: "X",
    Hobbies: "Intent Surfing" });
{ acknowledged: true, insertedIds: { '0': 1 } }
> db.Student.update ({ _id: 3, StudName: "Aryan", Grade: "VII" },
                     { $set: { Hobbies: "Skating" } },
                     { upsert: true });
{ acknowledged: true }
> db.Student.find ({ StudName: "Aryan" });
[ { _id: 3, Grade: 'VII', StudName: "Aryan", Hobbies: 'Skating' } ]
> db.Student.find({ Grade: { $eq: 'VII' } }).pretty();
[ { _id: 1, StudName: 'Rev', Grade: 'X', Hobbies: 'Patent Surfing' } ]
> db.Student.count();
```

2.
```
> db.Student.find().sort ({ Stud.Name: -1 }).pretty();
[ { _id: 1, StudName: 'Rev', Grade: 'X', Hobbies: 'Intent Surfing' }
  { _id: 3, Grade: 'VII', StudName: 'Aryan', Hobbies: 'Skating' }
]
```

2) importing CSV file :-

> mongoimport --uri "mongodb+srv://vanath@cluster0.bda1ah.mongodb.net/myDB --collection Sample JSON --type CSV --headerline --file C:\Users\Admin\Desktop\bmsce\sample

3 documents imported Succesfully

3) Exporting CSV file :-

> mongo export mongodb+srv://

mongodb.net/myDB --collection=Student --out C:\Users\Admin\Desktop\bmsce\sample.txt.

exported 4 records.

→ Save.method >

db.Students.save({Stud name:"Vanusi", Grade:"VI"})

db.Students.update({_id:4}, {$set:{Location:"Network"}})

db.student.find({_id:1}, {Stud Name:1, Grade:1, _id:0});

db.student.find({Grade:{$ne:'5'}}).pretty();

db.Student.find({StudName:'5'}).pretty();

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

db.students.count()

db.Students.find.count({Grade:"VII"})

db.Students.find({Grade:"VII"}).limit(3).pretty()

db.Students.find().sort({StudName:1}).pretty();

db.Students.find().skip(2).pretty()

Skip

Lab-2

(1)

myDB> db.Customers.insertMany([

{ Cust_id:1, ACC_Bal:1500, ACC_Type='Z'},

{Cust_id:2, ACC_Bal:900, Acc_Type='X'},

{ Cust_id: 3, ACC_Bal: 2000, ACC_Type:'Z'},

{ cust_id:4, ACC_Bal: 1100, ACC_Type: 'Y'},

{ cust_id:5, ACC_Bal:1800, ACC_Type:'Z'}

])

(2) myDB> db.Customers.find({ACC_Bal:{$gt:1200}, ACC_Type:"Z"})

(3) myDB> db.Customers.aggregate([

{

$group: {

_id: "$Cust_id",

Min_Balance={$min: "$ACC_Bal"},

Max_Balance={$max:"$ACC_Bal"}

}

}

])

2/b

myDB> db.creatCollection("User")

myDB> db.creatCollection("Orders")

db.users.insertMany([

]))

myDB> db.Order.insertMons([

...

])

→ ②

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

② ③ myDB> db.Products.find({quantity: {$gt: 0}})

③ myDB> db.Products.find().sort({price: 1})

④ myDB> db.Users.findOne({_id: "789ghi"}, {cart: 1, _id: 0})

⑤ myDB> db.orders.find({user_id: "123abc"})

⑥ myDB> db.orders.aggregate([

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

{$group: {_id, "$user_id", total_spent: {$sum: "$total_price"}}})

Aggregat :-

① my DB> db.Products.aggregate([

{ $group: { _id: "$Category", total-products: { $sum: 1 } } }

])

②

db.Products.aggregate([

{ $group: { _id: "$category", total price: { $sum: "$price" } } }

])

③ db.Products.aggregate([

{ $group: { _id: null, avg-price: { $avg: "$price" } } }

])

④ db.product.find({ quantity: { $lt: 10 } })

⑤ db.Products.find().sort({ price: -1 })

S.Price

⑥ db.orders.aggregate([

{ $group: { _id: "$user_id", total_spent: { $sum: "total-price" } } }

])

⑦ db.orders.aggregate([

{ $group: { _id: "$user_id", totalspent: { $sum: "totalprice" } } }

{ limit: 1 }

])

⑧ db.orders.aggregate([

{ $group: { _id: null, avg-order-value: { avg: totalprice } } }

])

Lab-3

Working with Cassandra

① Create keyspace:

Create keyspace Students With Replication = {'class': 'SimpleStrategy', 'replication_factor':1};

② Describe keyspaces;

③ Select * From 'system_schema'.keyspaces;

④ Use Students;

⑤ Create table Students_Info (Roll_NO int PRIMARY KEY, StudName text, Date Of Joining timestap, last_exam_percnt double);

⑥ Desribe Tables;

⑦ Describe Table <Table_Name>;

⑧ BEGIN BATCH

Insert INTO students_Info (Rodl_no, Stud Name, Date Of Joining, last_exam_percnt)

value (1, 'Asha', '2012-03-12', 79.9)

Insert INTO Students_Info (Rodl_..Data of Joining, last_exam_percnt)

Values (2, 'Kiran', '2012-03-12', 89.9)

⋮

⑨ Select * from Students_info :

roll_no	date of joining	last-exa
1	2012-03-12 00.000 +00	1
2	"	1
3	"	1
4	"	1

⑩

⑩ Select * from Students_Info where Roll_N

⑪ Select * from students_info where Studn

⑫ Create Index ON Students_Info (S

⑬ Select * from students_info where $f

⑭ Select Roll_NO, StudName from Stud

⑮ Selat Roll_No as "USN" fro

⑯ Update Student_info SET StudN

Roll No = 2;

⑰ Update students_info SET roll no

⑱ Delete Last_Exam_Percent FR

⑲ Delete from student_info where

⑳ Alter Table Students_Info Ad

㉑ Alter Table students_info ADD

'Simple Strategy', 'replication_factor':1};

KEY, StudName text,
double);

ud Name, Date Of Joining,
erand)

Joining, last_exam_percent)

⑨ Select * from Students_info :

roll_no	date of joining	last_exam_percent	Studname
1	2012-03-11 00.00.000 +00	79.9	Asha
2	"	"	"
3	"	"	"
4	"	"	"

⑩

⑩ Select * from Students_Info where Roll_No IN (1,2,3);

⑪ Select * from students_info where Studname = 'Asha';

⑫ Create Index ON Students_Info (StudName);

⑬ Select * from students_info where Studname = 'Asha';

⑭ Select Roll_NO, StudName from Students_info Limit 2;

⑮ Selat Roll_No as "USN" from Students_info;

⑯ Update Students_info SET StudName = 'David Sheen' where Roll No = 2;

⑰ Update students_info SET roll no = 3 where roll no = 2

⑱ Delete Last_Exam_Percent FROM students_info where RollNo = 2;

⑲ Delete from student_info where RollNo = 2;

⑳ Alter Table Students_info ADD hobbies set <text>;

㉑ Alter Table students_info ADD language list <text>;

UPdate Students_info

SET hobbies = hobbies + {'Chess, Tabb, Tennis'}

where Roll No = 1;

Update

UPdate library_book SET counter value = counter_value + 1

where book_name varchar, Stud_name varchar,

Primary key (book_name, Stud_name);

UPdate library_book SET Counter value = counter_value + 1

where book_name = 'BDA' AND Stud_name = "jeet";

Time to Learn :

Create Tabb userlogin (userid int PRIMARY KEY, password text);

Insert Into userlogin (userid, password) Values (1, 'info')

using TTL 30;

Select TTL (password) FROM userlogin where userid = 1;

N/A

Lab-03

Cassandra

① -- Cqlsh> Create keyspace Libra

'Cla

};

cqlsh > use library ;

② Create a column family :-

cqlsh: library > Create ta

Cqlsh: library >

Create table

counter

~~Cqlsh: library > begin~~

③ Insert the values :-

Cqlsh : library > begin

... insert into Li

values (112,

... insert "

... apply batch

Lab-03

Cassandra

① -- cqlsh> Create keyspace Library with replication = {'class': 'SimpleStrategy', 'replication_factor': 1};

cqlsh > use library ;

② Create a column family :-

cqlsh: library > Create table Library-Info (Stud-Id int,
Stud-Name text,
Book name text, Dat_of_Issue date,
Primary Key (Stud_Id, Book_Id));

Cqlsh: library >

Create table Book_counter (Stud-Id int, Stud-Name text,
counter_value counter, primary key (Stud-Name,
Book_name));

~~cqlsh: library >~~ begin

③ Insert the values :-

cqlsh : library > begin batch

... insert into Library_Info (Stud_Id, Stud_Name,
Book name, Book_Id, DOI)
values (112, 'alia', 'bda', 'b101', '08-07-2025');

... insert ''

... apply batch

Cqlsh : library > Select * from Library_Info:

→

Stud_Id	book_id	book_name	date_of_issue	Stud_name
113	b102	ads	2025-04-07	bob
112	b101	bda	2025-04-07	alice

④ cqlsh : library > update Book_Counter Set Counter_value =

Counter_value + 1 where Stud_Id = 112

and Book_name = 'boa';

Cqlsh : library > update Book_Counter Set Counter_value = Counter_value+1

where Stud_Id = 112 and Book_name = 'bda';

Cqlsh :

" "

Stud_Id = 112 and Book_name = 'bdo';

S. Pradeep

Exporting to CSV :

→ cqlsh -e "copy Library.Library_Info to 'library.info.csv'

with header = True:"

Importing from CSV :

→ cqlsh -e " copy Library.Library_Info from 'newfile.csv' with

header = True",

Hadoop

① Start-all.sh (start all the hadoop files)

② Creating a directory inside hadoop - mkdir

hdfs dfs - mkdir /bda-hadoop

③ Listing all content inside hadoop - ls

hadoop $ s - ls/

drwxr-xr-x - hadoop supergroup 0 2025-04-15 12:23 /bda-hadoop

④ Copying files from desktop to hdfs using put command

hdfs - dfs - put /home/hadoop/Desktop/bda.local.txt/

bda-hadoop/file.txt.

⑤ Copying files using copy from local cmd

hdfs dfs - copyfromlocal /home/hadoop/Desktop/bdd-local/

bad.hadoop /file2.txt.

⑥ Cat cmd (-cat)

hdfs-dfs - cat / bda-hadoop/file.txt.

⑦ get cmd

hdfs dfs - get /bda-hadoop/file.txt /home/hadoop/Downloads/

downloaded.txt.

⑧ get merge cmd

~~hdfs dfs -get /bda_hadoop /file.txt /home/hadoop/~~

hdfs dfs -getmerge /bda/file.txt /bda hadoop /file2.txt
home/hadoop/Desktop/bda_local.txt.

⑨ hadoop fs -get fact /bda_local)

file: /bda hadoop
owner : hduser
group : supergroup
user :: rwx
group :: rx
Other :: r-x

⑩ Copy to local

hdfs -dfs - copyToLocal /bda_hadoop/file.txt /home/hadoop/Desktop

⑪ move cmd

hadoop fs -mv /bda_hadoop/abc

hadoop fs -ls /abc

S. Pradeep 2024/25

⑫ Copy cmd

hadoop fs -cp /hello /# hadoop_lab

Lab-5

Word Count Program :-

→ Create JAR file & input text file.

→ Create Three Java classes into project Named WC Driver, WC Mapper, WC R

→ import java.io.IOException
import org.apache.hadoop.io.int
import org.apache.hadoop.io.
import org.apache.hadoop.io.Te

"
Public class WCMapper extends Ma
Mapper < LongWritable, Text, Tex
public void map (Long wri
Collector< Text,
Int writable > output, Repla
String line = value.toString
for (String word : line.Sp
if (word.length()
output.collect(
}
}
}

→ import java.IO.exception
import java.IO.exception
import org.apache.io.intwri
import org.apache.hadoop.io

.txt /home /hadoop)

txt /hdo hadoop /file2.txt
hdo_local.txt.

)

op / file.txt /home /

S. Pradeep
2024/25

lab

Lab-5

Word Count Program :-

→ Create JAR file & input text file

→ Create Three Java classes into project
Named WC Driver, WC Mapper, WC Reducer

→ import java.io.IOException
import org.apache.hadoop.io.intwritable
import org.apache.hadoop.io.longwritable
import org.apache.hadoop.io.Text

"
Public class WCMapper extends MapReduce Base implements
Mapper < Longwritable, Text, Text, Intwritable > {
public void map (Longwritable Key, Text value, Output
collector < Text,
Intwritable > output, Replar rep) throws IOException {

String line = value.toString();
for (String word : line.split(" ")) {
if (word.length() > 0) {

output.collect (new Text (word), new Intwritable (1));

}
}
}

→ import java.IO.exception
import java.IO.exception
import org.apache.io.intwritable
import org.apache.hadoop.io.Text

```
public class WC Reducer extends MapReduceBase implements Reducer <
IntWritable, Text, IntWritable, IntWritable > {

  Public void reduce ( Text key, Iterator < IntWritable > value,
        OutputCollector < Text, IntWritable >, output ) throws
                                        IOException {

    int count = 0 ;
    while (value.hasNext()) {
    count += i.get() ;
    }
  output.collect ( key, new IntWritable (count)) ;
}
```



```
Public class WCDriver extends Configured implements Tool {

  Public int run (String args[]) throws IOException {

      if (args.length < 2) {

        System.out.println ("Please give valid inputs");
        return -1;

      }

  }

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

    int exitCode = ToolRunner.run (new WC Driver(), args)
      System.exit (exitCode);

  }
}
```

⑤ Mean Max temp for every month

Mapper :

```
import sys
import csv
from datetime import datetime

reader = csv.reader (sys.stdin, delimiter)
next (reader)

for row in reader:
  try:
  date_str = row[0]
  temp = float (row[1])
  month = datetime.strptime (da
  print ("{}\t{}".format (month, temp
except Exception:
    continue
```

Reducer :

```
#! /usr/bin/env python3
import sys

current_month = None
total_temp = 0.0
count = 0

for line in sys.stdin:
    line = line.strip()
  if not line:
      continue
```

⑤ Mean Max temp for any month :

Mapper :

```
import sys
import csv
from datetime import datetime
reader = csv.reader(sys.stdin, delimiter=',')
next(reader)

for row in reader:
    try:
        date_str = row[0]
        temp = float(row[1])
        month = datetime.strptime(date_str, "%Y-%m-%d").month
        print(f"{month}\t{temp}")
except Exception:
    continue
```

Reducer :

```
#!/usr/bin/env python3
import sys

current_month = None
total_temp = 0.0
cont = 0

for line in sys.stdin:
    line = line.strip()
    if not line:
        continue
```

month, temp = line.split('\t')

if current_month == month:
    total_temp += temp
    count += 1
else:
    if current_month is not None:
        mean_temp = total_temp / count
        print(f"{current_month}\t{mean_temp:.2f}")
    current_month = month
    total_temp = temp
    count = 1

if current_month is not None:
    mean_temp = total_temp / count
    print(f"{current_month}\t{mean_temp:.2f}")

Lab 6

Hadoop → map reduce to list top 10 max

start-all.sh

nano mapperword.py

```
#!/bin/env python 3
import sys
import re
for line in sys.stdin
    words = re.findall(r'\w+', line.lo
    for word in words:
        print(f"{word}\t1")
```

nano reducer word.py

```
#!/usr/bin/env python3
import sys
from collection import defaultdict
count_map = defaultdict(int)
for line in sys.stdin:
        word, count = line.strip().split(
    count_map[word] += int(count)
sorted_words = sorted(count_map.items(),
for word, count in sorted_words[:10]:
    print(f"{word}\t{count}")
```

Chmod +x mapp word.py

Chmod +x reducerword.py

hadoop fs -mkdir /word

hdfs dfs -put /home/hadoop/Downloads/

hdfs dfs -ls /word

Lab 6

Hadoop → map reduce to list top 10 max words

start-all.sh

nano mapperwad.py

```
#!/bin/env python 3
import sys
import re
for line in sys.stdin
  words = re.findall('\w+', line.lower())
  for word in words:
      print(f"{word}\t1")
```

nano reducer word.py

```
#!/usr/bin/env python3
 import sys
 from collection import defaultdict
count_map = defaultdict(int)
for line in sys.stdin:
    word, count = line.strip().split('\t')
    count_map[word] += int(count)
sorted_words: sorted(count_map.items(), key= lambda x: (-x[1], x[0])
for word, count in sorted_words[:10]:
    print(f"{word}\t{count}")
```

chmod +x mapp word.py

Chmod +x reducerwad.py

hadoop fs -mkdir /word

hdfs dfs -put /home/hadoop/Downloads/txt, txt /word

hdfs dfs -ls /word

hadoop - strap - 3 -> jar

2ab -

Scala :

write a program to print from 1 to 100

→

```
nano Print Numbers . Scala

    object Print Number {

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

```
scalac Print Number.scala

Scala Print Numbers
```

Output : 1 2 3 4 ... 100

Spark :-

· List of word in strictly greater than using Spark.

nano wordcount.py

from pyspark import Sparkcontext

Sc = Sparkcontext ("local", "Word Count App")

text.add = sc.textfile ("input.txt")

words =add = text-add. flat Map (lamda line: line.split())

pair-add = words-add.map (landa word: (word,1)

words_count add = pair-add reduce Byte by(lamda a, b: a+b)

filtered add = words_counts-add.filter (lambda x: x[1] >2)

for word.count in filtered_add.collect (),
    print (f" {word}:{count}")

filter add .Saveas Txt File ("Output.dir")

nano input.txt

Spark-submit wordcount.py nano input.txt

Output:

apple : 6

red : 10

Spark - Open Ended :-

TextClean . Scala

import org.apache.spark.SparkConf

import org.apache.spark.Streaming

object TextClean {

def main (args: Array [String

val conf = new SparkCon

Set Master =("local [*]

val ssc = new StreamingC

val stopwords = Set ("a", "an", "t

"with", ".

val lines = ssc .Socket Text St

val cleaned = lines. flatMap (_

map (_ toLower(as. trim). filt

empty && ! Stop word

Cleaned .for each RDD (rdd

println ("Clea

rdd. collect (

})

ssc. start()

ssc. awaitTermination

})

Spark – Open Ended :-

TextClean . Scala

```
import org. apache. spark . Spark Conf
import org. apache. Spark . Streaming

object TextClean {
    def main ( args : Array [String] ) : unit = {
        val conf = new Spark Conf ( ). Set App name ("Text Clean")
        Set Maste = ( " local [*] " )
        val ssc = new Streaming Context ( conf, Seconds (5))
    val stopwords = Set ( "a", "an", "the", "is", "are", "on", "in",
                         "with", "to")
    val lines = ssc . Socket Text Stream ( "Local host", 9979)
  val cleaned = lines. flatMap ( _ .split ( "\\w+" ))
  map ( _ to Lower (as. trim ). filter (w => w.non
     Empty && ! Stop words . Contains (w))

Cleaned . for each RDD ( rdd => {
        println ( "Cleaned Output : " )
       rdd . collect ( ) . for each ( println )
   })

SSC . start ( )
SSC . awaitTermination ( )

})
```

Scalac -classpath "$SparkHome/jars/*" Test clean.scala

jar cf Test clean.jar Test Clean*.class

Spark-submit --class Test Clean --master local[*]

Test Clean.jar

nc -lk 9999

This is a example

Output:

This example.