Sri Raghavendra Educational Institutions Society(R)

Sri Krishna Institute of Technology

(Accredited by NAAC Approved by A.I.C.T.E. New Delhi, Recognized byGovt. of Karnataka
Affiliated to V.T U., Belagavi)
#29, Chimney Hills, Hesaraghatta Main Road, Chikkabanavara Post, Bangalore- 560090

LABORATORY MANUAL

Big Data Analytics Lab

[BCS701]

Compiled by

Ms. Kavana V,

Asst. Professor, ISE

Name of the Student:	
USN:	
Branch/Semester:	
Academic Year:	

Sri Raghavendra Educational Institutions Society(R)

Sri Krishna Institute of Technology

(Accredited by NAAC Approved by A.I.C.T.E. New Delhi, Recognized byGovt. of Karnataka
Affiliated to V.T U., Belagavi)
#29, Chimney Hills, Hesaraghatta Main Road, Chikkabanavara Post, Bangalore- 560090

GENERAL INSTRUCTIONS

The resource has been established as per requirement on need basis. In order to utilize the resource effectively and efficiently we need to follow few guidelines.

Do's

- Students **must** leave your footwear outside the lab at designated place.
- Students **must** keep your belongings in the designated place.
- Students **must** maintain discipline & follow ethics in the lab.
- Students **must** indicate the Login & Logout time on the record.
- Students **must** come prepare with Observation / Data sheet.
- Students **must** enter the lab and login to the allotted computer.
- Students **must** use the computer lab for Academic related activities.
- Students **must** use the computer lab for Academic related activities.
- Students **must** keep their working area orderly & neatly.
- Students **must** turn off the computer and keep the chairs properly before leaving the lab.

Don'ts

- Students **must not** eat food, chew gum in the lab.
- Students **must not** change the existing setup in the lab.
- Students **must not** disrupt, or trouble other students in the lab.
- Students **must not** interfere or tamper anyone else's work.
- Students **must not** access any computer or data without permission.
- Students **must not** connect the external storage to avoid transfer of virus.

Sri Raghavendra Educational Institutions Society(R)

Sri Krishna Institute of Technology

(Accredited by NAAC Approved by A.I.C.T.E. New Delhi, Recognized byGovt. of Karnataka
Affiliated to V.T U., Belagavi)
#29, Chimney Hills, Hesaraghatta Main Road, Chikkabanavara Post, Bangalore- 560090

Department of Information Science and Engineering

Vision

To nurture students to be globally competent professionals with a strong software development expertise, a passion for research, and a commitment to ethical values.

Mission

Our Mission is to:

1. Produce technically competent graduates through a well structured curriculum and interdisciplinary applications.

2. Create a competitive ambience for grooming young minds to meet the industry trends and become self-sustainable.

3. Impart knowledge of innovative technologies to encourage research and higher studies within the core areas of computation.

Sri Raghavendra Educational Institutions Society(R)

Sri Krishna Institute of Technology

(Accredited by NAAC Approved by A.I.C.T.E. New Delhi, Recognized byGovt. of Karnataka
Affiliated to V.T U., Belagavi)
#29, Chimney Hills, Hesaraghatta Main Road, Chikkabanavara Post, Bangalore- 560090

Vision of the Institute

To create a community of knowledgeable and competent engineers to embody global standards of excellence and drive innovation and progress in industries, businesses, and research organizations around the world.

Mission of the Institute

To facilitate an inclusive and supportive learning environment that fosters collaboration, creativity, and the pursuit of excellence in engineering.

Sri Raghavendra Educational Institutions Society(R)

Sri Krishna Institute of Technology

(Accredited by NAAC Approved by A.I.C.T.E. New Delhi, Recognized byGovt. of Karnataka
Affiliated to V.T U., Belagavi)
#29, Chimney Hills, Hesaraghatta Main Road, Chikkabanavara Post, Bangalore- 560090

Program Educational Objectives (PEOs)

PEO 1: Work effectively as an individual and in a team, exhibiting leadership qualities to meet the goals of the organization.

PEO2: To be able to ad pt to the evolving technical challenges and changing career opportunities and to pursue higher education.

PEO3: Learn to apply modern skills, techniques, and engineering tools to create computational systems.

Program Specific Objectives (PSOs)

PSO1: To understand and process the principles of mathematics in the field of information Science by applying different design principles.

PSO2: To impart the knowledge by experimental methods in multidisciplinary domains.

PSO3: To inculcate communication skills and team work in developing sustainable software's by imparting professional and ethical values.

Sri Raghavendra Educational Institutions Society(R)

Sri Krishna Institute of Technology

(Accredited by NAAC Approved by A.I.C.T.E. New Delhi, Recognized byGovt. of Karnataka
Affiliated to V.T U., Belagavi)
#29, Chimney Hills, Hesaraghatta Main Road, Chikkabanavara Post, Bangalore- 560090

Program Outcomes (POs)

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Sri Raghavendra Educational Institutions Society(R)

Sri Krishna Institute of Technology

(Accredited by NAAC Approved by A.I.C.T.E. New Delhi, Recognized byGovt. of Karnataka
Affiliated to V.T U., Belagavi)
#29, Chimney Hills, Hesaraghatta Main Road, Chikkabanavara Post, Bangalore- 560090

PRACTICAL COMPONENT

SI.NO	Experiments (Java/Python/R)
1	Install Hadoop and Implement the following file management tasks in Hadoop: Adding files and directories Retrieving files Deleting files and directories. **Hint:** A typical Hadoop workflow creates data files (such as log files) elsewhere and copies them into HDFS using one of the above command line utilities.
2	Develop a MapReduce program to implement Matrix Multiplication
3	Develop a Map Reduce program that mines weather data and displays appropriate messages indicating the weather conditions of the day.
4	Develop a MapReduce program to find the tags associated with each movie by analyzing movie lens data.
5	Implement Functions: Count - Sort - Limit - Skip - Aggregate using MongoDB
6	Write Pig Latin scripts to sort, group, join, project, and filter the data.
7	Use Hive to create, alter, and drop databases, tables, views, functions, and indexes.
8	Implement a word count program in Hadoop and Spark.

Experiment 1:

Install Hadoop and Implement the following file management tasks in Hadoop:

Adding files and directories

Retrieving files

Deleting files and directories.

Hint: A typical Hadoop workflow creates data files (such as log files) elsewhere and copies them into HDFS using one of the above command line utilities.

- **Install Hadoop**

Download and Install Java

Hadoop requires Java to run. Install Java if you haven't already.

Check Java Installation

Open Command Prompt (cmd) and type:

```
java -version
```

If Java is not installed, download and install Java JDK (8 or 11) from:

Official Oracle Java Download

OpenJDK Download

Set Java Environment Variables:

Go to Control Panel > System > Advanced System Settings > Environment Variables.

Under System Variables, click New and set:

Variable name: JAVA_HOME

Variable value: C:\Program Files\Java\jdk-<version> (replace <version> with your installed JDK version).

Edit the Path variable and add:

```
%JAVA_HOME%\bin
```

Download and Configure Hadoop

Download Hadoop binary for Windows from:

Apache Hadoop Releases

Extract the zip file to C:\hadoop or any directory.

Set Hadoop Environment Variables:

Add a new System Variable:

Variable name: HADOOP_HOME

Variable value: C:\hadoop

Edit the Path variable and add:

%HADOOP_HOME%\bin

Configure Hadoop for Windows:

Inside the Hadoop folder (C:\hadoop\etc\hadoop), edit these configuration files:

Edit core-site.xml:

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

Edit hdfs-site.xml:

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
```

```
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:///C:/hadoop/data/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:///C:/hadoop/data/datanode</value>
  </property>
</configuration>
```

Edit mapred-site.xml:

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

Edit yarn-site.xml:

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

Format the Namenode (Run in Command Prompt):

```
hdfs namenode -format
```

Start Hadoop Services

Open Command Prompt (cmd) as Administrator.

Navigate to Hadoop directory:

cd C:\hadoop\sbin

Start Hadoop services:

start-dfs.cmd

start-yarn.cmd

jps

Verify HDFS is running:

Open a browser and go to:

http://localhost:9870

http://localhost:8088/

http://localhost:9870/explorer.html

You should see the Hadoop Web UI.

Task 1: Adding Files and Directories to HDFS

Create a directory in HDFS:

hdfs dfs -mkdir /mydata

Copy a local file to HDFS:

hdfs dfs -put C:/Users/YourUser/Desktop/sample.txt /mydata/

Explanation:

The -mkdir command creates a directory inside HDFS (/mydata).

The -put command uploads a local file (sample.txt) to HDFS (/mydata).

Steps:

You first create a directory in HDFS where your data will be stored.

Then, you upload the data (for example, log files or text files) into that directory.

Task 2: Retrieving Files from HDFS

Copy a file from HDFS to the local filesystem:

hdfs dfs -get /mydata/sample.txt C:/Users/YourUser/Desktop/

Display the content of a file in HDFS:

hdfs dfs -cat /mydata/sample.txt

Explanation:

The -get command downloads a file from HDFS to your local directory.

Here, the file sample.txt from /mydata is downloaded to the local directory C:\Users\YourUser\Downloads.

Steps:

You can use this command to retrieve files after processing or analysis on Hadoop.

Task 3: Deleting Files and Directories in HDFS

Delete a file in HDFS:

hdfs dfs -rm /mydata/sample.txt

Delete a directory in HDFS:

hdfs dfs -rm -r /mydata

Explanation:

The -rm command deletes a specific file (sample.txt) from HDFS.

The -rm -r command recursively deletes a directory (/mydata) and all of its contents.

Steps:

Use this to clean up data after processing or if the directory is no longer needed.

Experiment 2:

Develop a MapReduce program to implement Matrix Multiplication

Open Notepad and write

```
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.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


import java.io.IOException;

import java.util.HashMap;


public class MatrixMultiplication {


   // Mapper Class

   public static class MatrixMapper extends Mapper<Object, Text, Text, Text> {

      public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {

         String[] tokens = value.toString().split(",");

         String matrix = tokens[0];  // "A" or "B"

         int row = Integer.parseInt(tokens[1]);

         int col = Integer.parseInt(tokens[2]);
```

```
        int val = Integer.parseInt(tokens[3]);

        if (matrix.equals("A")) {
            for (int k = 0; k < 2; k++) { // Assuming B has 2 columns
                context.write(new Text(row + "," + k), new Text("A," + col + "," + val));
            }
        } else {
            for (int i = 0; i < 2; i++) { // Assuming A has 2 rows
                context.write(new Text(i + "," + col), new Text("B," + row + "," + val));
            }
        }
    }

    // Reducer Class
    public static class MatrixReducer extends Reducer<Text, Text, Text, IntWritable> {
        public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
            HashMap<Integer, Integer> A = new HashMap<>();
            HashMap<Integer, Integer> B = new HashMap<>();

            for (Text val : values) {
                String[] parts = val.toString().split(",");
                int index = Integer.parseInt(parts[1]);
                int value = Integer.parseInt(parts[2]);
```

```
            if (parts[0].equals("A")) {
                A.put(index, value);
            } else {
                B.put(index, value);
            }
        }

        int sum = 0;
        for (int j : A.keySet()) {
            if (B.containsKey(j)) {
                sum += A.get(j) * B.get(j);
            }
        }

        context.write(key, new IntWritable(sum));
    }
}

// Driver Class
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Matrix Multiplication");

    job.setJarByClass(MatrixMultiplication.class);
    job.setMapperClass(MatrixMapper.class);
    job.setReducerClass(MatrixReducer.class);
```

```
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

Save the Program:

Click of Save and save the program with the name MatrixMultiplication.java

Make two text file and save as matrixA.txt, matrixB.txt:

A,0,0,1

A,0,1,2

A,1,0,3

A,1,1,4

B,0,0,5

B,0,1,6

B,1,0,7

B,1,1,8

Change the directory from cmd:

Go to the directory where the code is saved using cd

Get the Hadoop class path:

hadoop classpath

Compile the Java File

Now, compile your Java program using the copied classpath:

javac -classpath "PASTE_HADOOP_CLASSPATH_HERE" -d . MatrixMultiplication.java

Create a JAR File

Once compilation is successful, create the JAR file:

jar cf matrixmultiplication.jar *.class

Upload Input Files to HDFS

start-dfs.cmd

start-yarn.cmd

hdfs dfs -mkdir /matrix

hdfs dfs -put matrixA.txt /matrix/

hdfs dfs -put matrixB.txt /matrix/

Run the Hadoop Job

hadoop jar matrixmultiplication.jar MatrixMultiplication /matrix /output

View the Output

hdfs dfs -cat /output/part-r-00000

Experiment 3:

Develop a Map Reduce program that mines weather data and displays appropriate messages indicating the weather conditions of the day.

Open Notepad and write

```
import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


import java.io.IOException;


public class WeatherAnalysis {


   // Mapper Class

   public static class WeatherMapper extends Mapper<Object, Text, Text, Text> {

      public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {

         String line = value.toString();

         // Skip header line

         if (line.startsWith("Date")) {

            return;

         }
```

```
        String[] parts = line.split(",");
        if (parts.length < 3) {
            return; // Ignore malformed lines
        }

        String date = parts[0];  // Date column
        String maxTempStr = parts[2];  // Max temperature column

        try {
            context.write(new Text(date), new Text(maxTempStr));
        } catch (NumberFormatException e) {
            // Skip invalid temperature values
        }
    }
}

// Reducer Class
public static class WeatherReducer extends Reducer<Text, Text, Text, Text> {
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
        for (Text val : values) {
                float maxTemp = Float.parseFloat(val);
        String condition;
            // Determine weather condition
            if (maxTemp > 30) {
                condition = "Hot Day";
```

```
            } else if (maxTemp < 10) {
                condition = "Cold Day";
            } else {
                condition = "Normal Day";
            }
            context.write(key, condition);
        }
    }
}

// Driver Class (Main Method)
public static void main(String[] args) throws Exception {
    if (args.length != 2) {
        System.err.println("Usage: WeatherAnalysis <input path> <output path>");
        System.exit(-1);
    }

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Weather Condition Analysis");

    job.setJarByClass(WeatherAnalysis.class);
    job.setMapperClass(WeatherMapper.class);
    job.setReducerClass(WeatherReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
```

```
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

Save the Program:

Click of Save and save the program with the name WeatherMapper.java

Make a CSV file and save as weather_data.csv:

Date,Location,MaxTemperature,MinTemperature

2025-03-01,NewYork,32,20

2025-03-02,NewYork,8,-2

2025-03-03,NewYork,25,15

Change the directory from cmd:

Go to the directory where the code is saved using cd

Get the Hadoop class path:

hadoop classpath

Compile the Java File

Now, compile your Java program using the copied classpath:

javac -classpath "PASTE_HADOOP_CLASSPATH_HERE" -d . WeatherMapper.java

Create a JAR File

Once compilation is successful, create the JAR file:

jar cf weather.jar WeatherAnalysis*.class

Upload Input Files to HDFS

start-dfs.cmd

start-yarn.cmd

hdfs dfs -mkdir -p /user/hadoop/weather

hdfs dfs -put weather_data.csv /user/hadoop/weather/

Run the Hadoop Job

hadoop jar weather.jar WeatherAnalysis /user/hadoop/weather/weather_data.csv /user/hadoop/weather_output

View the Output

hdfs dfs -cat /user/hadoop/weather_output/part-*

Experiment 4:

Develop a MapReduce program to find the tags associated with each movie by analyzing movie lens data.

Open Notepad and write

```
import java.io.IOException;

import java.util.StringJoiner;

import java.util.HashMap;

import java.util.Map;


import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


public class MovieTags {

   // Mapper Class

   public static class TagMapper extends Mapper<LongWritable, Text, Text, Text> {

      public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {

         String line = value.toString();
```

```
        String[] fields = line.split(",");

        if (fields.length >= 4 && !fields[0].equals("userId")) {  // Skip header
            String movieTitle = fields[2];  // Extract movie title
            String tag = fields[3];         // Extract tag

            context.write(new Text(movieTitle), new Text(tag)); // Emit movieTitle -> tag
        }
    }
}

// Reducer Class
public static class TagReducer extends Reducer<Text, Text, Text, Text> {
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
        StringJoiner tags = new StringJoiner(", ");
        for (Text val : values) {
            tags.add(val.toString());
        }
        context.write(key, new Text(tags.toString())); // Emit movieTitle -> tags
    }
}

// Driver Code
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
```

```
        Job job = Job.getInstance(conf, "Movie Tags");

        job.setJarByClass(MovieTags.class);
        job.setMapperClass(TagMapper.class);
        job.setReducerClass(TagReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

Save the Program:

Click of Save and save the program with the name MovieTags.java

Make a CSV file and save as tags.csv:

```
userId,movieId,movieTitle,tag,timestamp
1,296,Terminator 2: Judgment Day (1991),funny,1139045764
2,306,Pulp Fiction (1994),dark humor,1255630284
3,307,The Matrix (1999),sci-fi,1343294286
4,307,The Matrix (1999),futuristic,1343294287
```

5,308,Se7en (1995),thriller,1455639852

6,308,Se7en (1995),suspense,1467890234

7,309,Inception (2010),mind-bending,1501234567

8,309,Inception (2010),dreams,1501237890

9,310,The Dark Knight (2008),gritty,1522345678

10,310,The Dark Knight (2008),masterpiece,1522348999

Change the directory from cmd:

Go to the directory where the code is saved using cd

Get the Hadoop class path:

hadoop classpath

Compile the Java File

Now, compile your Java program using the copied classpath:

javac -classpath "PASTE_HADOOP_CLASSPATH_HERE" -d . MovieTags.java

Create a JAR File

Once compilation is successful, create the JAR file:

jar cf movietags.jar MovieTags*.class

Upload Input Files to HDFS

start-dfs.cmd

start-yarn.cmd

hdfs dfs -mkdir -p /user/hadoop/movie

hdfs dfs -put tags.csv /user/hadoop/movie/

Run the Hadoop Job

hadoop jar movietags.jar MovieTags /user/hadoop/movie/tags.csv /output/movietags

View the Output

hdfs dfs -cat /output/movietags/part-r-00000

Experiment 5:

Implement Functions: Count – Sort – Limit – Skip – Aggregate using MongoDB

Open Command Prompt as Administrator and type:

mongod

mongosh

1. Create a Database

In MongoDB, a database is created automatically when you insert a document into a collection.

use university

This switches to (or creates if not existing) a database named university.

2. Create a Collection and Insert Documents

Create a students collection and insert multiple documents.

```
db.students.insertMany([
  { name: "Alice", age: 22, department: "CS", marks: 85 },
  { name: "Bob", age: 24, department: "IT", marks: 78 },
  { name: "Charlie", age: 21, department: "CS", marks: 90 },
  { name: "David", age: 23, department: "IT", marks: 88 },
  { name: "Eve", age: 20, department: "ECE", marks: 75 },
  { name: "Frank", age: 25, department: "CS", marks: 92 },
  { name: "Grace", age: 22, department: "ECE", marks: 80 }
])
```

3. Count Documents

The countDocuments() method counts documents that match a query.

```
db.students.countDocuments({ age: { $gte: 22 } })
```

- Counts the number of students with age greater than or equal to 22.

4. Sort Documents

Sorting documents in ascending (1) or descending (-1) order.

```
db.students.find().sort({ marks: -1 })
```

- Sorts students in descending order of marks.

5. Limit the Number of Results

The limit() method restricts the number of documents retrieved.

```
db.students.find().limit(3)
```

- Retrieves only the first 3 student documents.

6. Skip Documents

The skip() method allows skipping a specified number of documents.

```
db.students.find().skip(2)
```

- Skips the first 2 student records and returns the rest.

7. Aggregate Function

Aggregation allows data transformation and complex calculations.

```
db.students.aggregate([
  { $match: { marks: { $gte: 80 } } },  // Filter students with marks >= 80
  { $group: { _id: "$department", avgMarks: { $avg: "$marks" } } }, // Group by department, calculate avg marks
  { $sort: { avgMarks: -1 } }, // Sort by average marks in descending order
  { $limit: 2 } // Return top 2 departments with highest average marks
])
```

- Finds the average marks for each department, sorts them, and returns the top 2.

Experiment 6:

Use Hive to create, alter, and drop databases, tables, views, functions, and indexes.

Start Hadoop

Open Command Prompt as Administrator.

Go to the Hadoop sbin folder:

cd C:\hadoop-3.3.6\sbin and type

start-all.cmd

Confirm Namenode and Datanode started by checking:

http://localhost:9870/ (Namenode UI)

Also in the Command Prompt type

jps

it should show: NameNode, DataNode, SecondaryNameNode, ResourceManager, NodeManager

Initialize Hive Metastore (First Time Only)

If not done already, initialize the schema for Hive using Derby (default):

Open Command Prompt

Go to the Hive bin folder:

cd C:\hive\apache-hive-3.1.2-bin\bin and type

hive --service schematool -dbType derby -initSchema

You should see output like: Initialization script completed

Start Hive CLI

hive

Wait for the Hive prompt:

hive>

The Main program:

Create Hive Script File

Open **Notepad** or any text editor.

Copy and paste the following Hive commands into it:

```
-- DATABASE OPERATIONS

-- Create a database

CREATE DATABASE IF NOT EXISTS college;

-- Alter a database by setting properties

ALTER DATABASE college SET DBPROPERTIES ('creator' = 'Surbhi', 'created_on' = '2025-05-02');

-- Describe the database

DESCRIBE DATABASE EXTENDED college;

-- TABLE OPERATIONS

-- Create a table

CREATE TABLE IF NOT EXISTS students (
  id INT,
  name STRING,
  marks FLOAT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

-- Alter the table

ALTER TABLE students ADD COLUMNS (gender STRING);

ALTER TABLE students CHANGE name full_name STRING;

ALTER TABLE students REPLACE COLUMNS (id INT, full_name STRING, marks FLOAT, gender STRING, grade STRING);
```

```
-- Describe the table

DESCRIBE students;

-- VIEW OPERATIONS

-- Insert data into table

INSERT INTO TABLE students VALUES (1, 'John Doe', 92.5, 'M', 'A'), (2, 'Jane Smith', 75.0,
'F', 'B');

-- Create a view

CREATE VIEW IF NOT EXISTS top_students AS

SELECT id, full_name, marks FROM students WHERE marks >= 80;

-- Alter a view (by dropping and recreating)

DROP VIEW IF EXISTS top_students;

CREATE VIEW top_students AS

SELECT full_name, marks FROM students WHERE marks >= 80;

-- Drop the view

DROP VIEW IF EXISTS top_students;

-- FUNCTION OPERATIONS

-- Create a temporary function (UDF)

CREATE TEMPORARY FUNCTION reverse_udf AS
'org.apache.hadoop.hive.ql.udf.UDFReverse';

-- Use the function

SELECT reverse_udf('HiveExample') AS reversed;

-- Drop the function

DROP TEMPORARY FUNCTION reverse_udf;

-- INDEX OPERATIONS

-- Create a new table

CREATE TABLE IF NOT EXISTS marks (
```

```
  id INT,
  name STRING,
  marks INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
-- Create an index
CREATE INDEX idx_marks ON TABLE marks (marks)
AS 'COMPACT'
WITH DEFERRED REBUILD;
-- Rebuild the index
ALTER INDEX idx_marks ON marks REBUILD;
-- Drop the index
DROP INDEX idx_marks ON marks;
-- DROP TABLE students;
-- DROP TABLE marks;
-- DROP DATABASE college CASCADE;
```

Save the file as:

hive_script.sql

Save it inside: C:\hadoop-3.3.6\apache-hive-3.1.2-bin\bin (or any location you can navigate to from CMD).

Run Hive Script File

Run your script using the command below in the same folder:

```
hive -f hive_script.sql
```

This will execute all the Hive commands in the file one by one.

Experiment 7:

Develop Pig Latin scripts to sort, group, join, project, and filter the data.

Start Hadoop Services

start-dfs.sh

start-yarn.sh

Start Pig in Local or MapReduce Mode

- **Local Mode (for testing):** pig -x local
- **MapReduce Mode:** pig

Assume we have a CSV file: students.csv

101,John,CS,80

102,Alice,EC,90

103,Bob,CS,70

104,David,ME,85

Suppose we have another file: dept_info.csv

CS,Computer Science

EC,Electronics

ME,Mechanical

Open Notepad.

Save the following script as student_analysis.pig

```
-- Load student data
        students = LOAD 'students.csv' USING PigStorage(',')
                AS (id:int, name:chararray, dept:chararray, marks:int);
-- Load department data
        departments = LOAD 'departments.csv' USING PigStorage(',')
                AS (code:chararray, dept_name:chararray);
```

-- 1. Projection: Select name and marks

projected = FOREACH students GENERATE name, marks;

-- 2. Filtering: Students with marks > 80

filtered = FILTER students BY marks > 80;

-- 3. Sorting: Students sorted by marks descending

sorted = ORDER students BY marks DESC;

-- 4. Grouping: Group by department

grouped = GROUP students BY dept;

-- Calculate average marks per department

avg_marks = FOREACH grouped GENERATE group AS dept, AVG(students.marks) AS avg_marks;

-- 5. Joining: Join student with department info

joined = JOIN students BY dept, departments BY code;

-- Show outputs

DUMP projected;

DUMP filtered;

DUMP sorted;

DUMP avg_marks;

DUMP joined;

How to Execute the Script

From the terminal, navigate to the directory where the script is saved and run:

In local mode (no need for Hadoop):

pig -x local student_analysis.pig

In MapReduce mode (uses Hadoop cluster):

pig student_analysis.pig

Experiment 8:

Implement a word count program in Hadoop and Spark.

Word Count in Hadoop MapReduce

Write the WordCount Program in Java

Create a file WordCount.java:

```
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.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
 public static class TokenizerMapper
      extends Mapper<Object, Text, Text, IntWritable>{
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();
   public void map(Object key, Text value, Context context)
       throws IOException, InterruptedException {
     StringTokenizer itr = new StringTokenizer(value.toString());
     while (itr.hasMoreTokens()) {
```

```
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
}
public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
```

```
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Compile and Create JAR

```
javac -classpath $(hadoop classpath) -d wordcount_classes WordCount.java
jar -cvf wordcount.jar -C wordcount_classes/ .
```

Input File and HDFS Setup

```
hdfs dfs -mkdir /input
hdfs dfs -put input.txt /input
```

Run the Job

```
hadoop jar wordcount.jar WordCount /input /output
```

View Output

```
hdfs dfs -cat /output/part-r-00000
```

Word Count in Apache Spark (PySpark)

Prerequisites

- **Apache Spark installed.**
- **Python 3 installed.**
- **$SPARK_HOME and $PATH configured.**

- **Step 2: Create wordcount.py Script**

```
from pyspark import SparkContext
sc = SparkContext("local", "WordCountApp")
# Read input file
text_file = sc.textFile("input.txt")
# Word Count Logic
counts = (text_file.flatMap(lambda line: line.split())
            .map(lambda word: (word, 1))
            .reduceByKey(lambda a, b: a + b))
# Save Output
        counts.saveAsTextFile("output_spark")
```

- **Step 3: Run the Spark Job**

```
spark-submit wordcount.py
```

- **Step 4: View Output**

```
cat output_spark/part-*
```