JAI SHRIRAM ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Affiliated to Anna University, Chennai
Recognized by UGC & Accredited by NAAC and NBA for CSE & ECE)
Dharapuram Road, Avinashipalayam, Tirupur – 638 660.

Academic Year 2023-2024 (Odd Semester)

LABORATORY RECORD

Certified that this is a bonafide record of work done by

Name	………………………………………………………………………..
Reg. No.	………………………………………………………………………..
Branch	………………………………………………………………………..
Year & Semester	………………………………………………………………………..
Course code & Name	………………………………………………………………………..

Course In-Charge **Head of the Department**

Submitted for the University Practical Examination held on ………………..

Internal Examiner **External Examiner**

INDEX

Ex.No : 1 **Date:**	DOWNLODING AND INSTALLING HADOOP: UNDERSTANDING DIFFERENT HADOOP MODES, STARTUP SCRIPTS, CONFIGURATION FILES.

<u>Aim:</u>

To write a downloding and installing hadoop: understanding different hadoop modes, startup scripts, configuration files.

<u>Program :</u>

INTRODUCTION

- LAB OBJECTIVES:
  - ❖ Understand the process of downloading and installing Hadoop
  - ❖ Familiarize yourself with different Hadoop modes.
  - ❖ Learn about startup scripts and configuration files in Hadoop
- BACKGROUND INFORMATION:

  Hadoop is an open-source framework that allows for the distributed processing of large data sets across clusters of computers, It is widely used in big data processing and analytics. Understanding the installation proce different modes, startup scripts, and configuration files is essential for working with Hadoop effectively.
- . Safety Guidelines:
  - ❖ Ensure that you have the necessary permissions and access rights to install software on your system.
  - ❖ Follow proper security protocols when handling sensitive data during the installation process.

1. DOWNLOADING AND INSTALLING HADOOP:

- Visit the official Apache Hadoop website at https://hadoop.apache.org/ (or the vendor's website if using a specific distribution).
- Navigate to the "Downloads" section and select the appropriate version of Hadoop for your operating system.

- .Download the distribution package (usually a tarball or zip file) to your local machine.

2. INSTALLING HADOOP:

- Extract the downloaded package to a directory of your choice. For example, you can extract it to /opt/hadoop on Linux.
- Set up the necessary environment variables
- Open a terminal or command prompt and navigate to the directory where Hadoop is installed.
- Edit the hadoop-envsh file, located in the echadoop directory
- Set the JAVA_HOME variable to the path where Java is installed on your system. For example: export JAVA HOME-uslim java-&-openj amd64
- Save the changes and exit the editor

3. UNDERSTANDING DIFFERENT HADOOP MODES

- Local (Standalone) mode:
- In this mode, Hadoop runs on a single machine without any distributel processing
- It is suitable for testing or debugging small datasets
- All components, such as HDFS and MapReduce, run as separate Java processes on the same machine.

Pseudo-Distributed mode:

- In this mode, Hadoop simulates a distributed environment on a single machine.
- It is suitable for development and testing purposes.
- Each Hadoop component runs as a separate lava process, but they communicate with each other as if they were running on different machines

Fully Distributed mode:

- In this mode, Hadoop runs across multiple machines in a cluster.
- Each machine in the cluster performs specific roles, such as NameNode. DataNode, and Task Tracker.

- This mode is suitable for large-scale production environments.

4. STARTUP SCRIPTS AND CONFIGURATION FILES:

- Hadoop provides startup scripts and configuration files to manage different aspects of the framework.
- Locate the etc/Hadoop directory in your Hadoop installation directory.
- Familiarize yourself with the following important files:
- hadoop-env.sh: Edit this file to set environment variables for Hadoop, such as the Java home directory and Hadoop home directory.
- . core-site.xml: Configure settings related to the Hadoop core, such as the default file system URI and block size.
- hdfs-site.xml: Configure settings specific to Hadoop Distributed File System (HDFS), such as the data directory, replication factor, and tuning parameters
- mapred-site.xml: Configure settings related to the MapReduce framework, including the job tracker URL, task tracker configuration, and resource allocation.

Result :

Hadoop was succesfully Downloaded and installed, Hadoop modes sturtup scripts configuration files are understanded briefly

Ex.No : 2 **Date:**	HADOOP IMPLEMENTATION OF FILE MANAGEMENT TASKS, SUCH AS ADDING FILES AND DIRECTORIES, RETRIEVING FILES AND DELETING FILES

Aim :

To write a hadoop implementation of file management tasks, such as adding files and directories, retrieving files and deleting files using queries.

Program :

OBJECTIVE:

- Gain hands-on experience with file management tasks in Hadoop using the Hadoop Distributed File System (HDFS).
- . Learn how to add files and directories, retrieve files, and delete files in HDFS.

Background Information:

Hadoop is a powerful framework for distributed processing of large datasets. The Hadoop Distributed File System (HDFS) is a key component of Hadoop that provides reliable and scalable storage for big data. Understanding file management in HDFS is crucial for effective data processing and analysis in Hadoop.

Equipment and Materials:

- Personal computer or virtual machine with Hadoop installed and configured.
- Access to HDFS through Hadoop Command-Line Interface (CLI) or programming environment (e.g., Java).

Experiment Setup:

- cc

Task 1: Adding Files and Directories to HDFS

- Use the HDFS Command-Line Interface (CLI) or Hadoop APIs to add files and directories to HDFS.
- Follow the provided instructions to:
    - ❖ Create directories in HDFS.

- ❖ Copy files from the local filesystem to HDFS.

Using HDFS CLI:

- Open a terminal or command prompt.
- Use the hdfs dfs command to interact with HDFS. For example:
  - ❖ To create a directory: hdfs dfs -mkdir /path/to/directory
  - ❖ To copy a file from the local filesystem to HDFS: hdfs dfs-put /path/to/ local/file/path/in/hdfs
  - ❖ To copy a directory from the local filesystem to HDFS: hdfs dfs -put/ path/to/local/directory /path/in/hdfs

Using Hadoop APIs:

- Write a Java program using Hadoop APIs to interact with HDFS.
- Use the FileSystem class to create directories or copy files.

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

public class HadoopFileManagement [

public static void main(String[] args) {

try {

Configuration conf- new Configuration();

FileSystem fs-FileSystem.get(conf);

fs.copyFromLocalFile(new Path("/path/to/local/file"), new Path("/path/in hdfs"));

fs.close():

} catch (Exception e) {

e.printStackTrace();

}}}
```

Task 2: Retrieving Files from HDFS

- Use the HDFS CLI or Hadoop APIs to retrieve files from HDFS.
- Follow the provided instructions to:
- Copy files from HDFS to the local file system.

Using HDFS CLI:

Use the hdfs dfs-get command to copy files from HDFS to the local filesystem.

For example:

To copy a file from HDFS to the local filesystem: hdfs dfs-get /path/in/hdfs /path/to/local/file

Using Hadoop APIs:

Write a Java program using Hadoop APIs to retrieve files from HDFS.

Use the FileSystem class to open an input stream to the desired file and read its contents.

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

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

public class HadoopFileManagement (

public static void main(String[] args) {

try {

Configuration conf= new Configuration();

FileSystem fs = FileSystem.get(conf);

fs.copy ToLocalFile(new Path("/path/in/hdfs"), new Path("/path/to/local/file"));

fs.close():
} catch (Exception e) {

e.printStackTrace();

}}}
```

Task 3: Deleting Files from HDFS

- Use the HDFS CLI or Hadoop APIs to delete files and directories from HDFS
- Follow the provided instructions to:
- Delete files and empty directories from HDFS.
- Delete non-empty directories and their contents from HDFS,

Using HDFS CLI:

- Use the hdfs dfs -rm command to delete files or directories in HDFS, For example:
    - ❖ To delete a file: hdfs dfs -rm /path/to/file
    - ❖ To delete an empty directory: hdfs dfs -rmdir /path/to/empty/directory
    - ❖ To delete a non-empty directory and its contents: hdfs dfs -rm -r/path/ to/non-empty/directory

Using Hadoop APIs:

- Write a Java program using Hadoop APIs to delete files or directories in HDFS.
- Use the FileSystem class to delete files or directories.

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

import org.apache.hadoop.fs.Path;

public class HadoopFileManagement {

public static void main(String[] args) {

try { Configuration conf= new Configuration();

FileSystem fs-FileSystem.get(conf);
```

```
fs delete(new Path("/path/to/file_or_directory"), true); // Set recursive to true

for directories

fs.close();

catch (Exception e) {

} e.printStackTrace();
```

Remember to update the paths in the code snippets with your specific file and directory locations in HDFS or the local filesystem.

Result:

Thus the implementation of file management tasks,such as adding file and directories,retrieveing files and deleting file was implemented successfully.

Ex.No : 3 **Date:**	HADOOP IMPLEMENT OF MATRIX MULTIPLICATION WITH HADOOP MAP

Aim :

To write a program for matrix multiplication with hadoop map.

Programe :

Objective:

Implement matrix multiplication using Hadoop MapReduce to understand the distributed processing capabilities of Hadoop.

Equipment and Materials:

- Personal computer or virtual machine with Hadoop installed and configured.
- Access to HDFS through Hadoop Command-Line Interface (CLI) or programming environment (e.g., Java).
- Text editor or Integrated Development Environment (IDE) for coding.

Experiment Steps:

Dataset Preparation:

- Generate or obtain two input matrices, Matrix A and Matrix B, in a suitable format (e.g., CSV or text file).
- Divide the matrices into blocks to enable parallel processing. Each block represents a subset of rows and columns from the original matrices.

Mapper Implementation:

- Create a new Java class, MatrixMultiplicationMapper, to implement the map function for matrix multiplication.
- Use Hadoop's Mapper class as the base class for your implementation.
- Override the map() method to read input matrix blocks, perform partial multiplications, and emit intermediate key-value pairs.

- The key should represent the resulting cell position in the output matrix, and the value should contain the partial multiplication result.

Reducer Implementation:

- Create a new Java class, MatrixMultiplicationReducer, to implement the reduce function for matrix multiplication.
- Use Hadoop's Reducer class as the base class for your implementation.
- Override the reduce() method to receive the shuffled intermediate key-value. pairs and perform the final summation.
- Iterate through the values associated with a key and calculate the final cell value of the output matrix.
- Emit the final key-value pairs as the output of the reduce function..

Hadoop Job Configuration:

- Create a new Java class, MatrixMultiplicationJob, to configure and submit the Hadoop job.
- Use Hadoop's Job class to set up the MapReduce job configuration.
- Specify the input and output paths for the matrices.
- Set the mapper and reducer classes to be used in the job.
- Set any additional job-specific configurations, such as the number of reducers.

Execute the Job:

- In the main method of the MatrixMultiplicationJob class, instantiate the Job object and set the jar file for the job.
- Submit the job for execution using the waitForCompletion() method of the Job object.

Output Verification:

- After the job completes, verify the output file in HDFS to ensure that the matrix multiplication has been performed correctly.
- Compare the output with a known result or perform further analysis on the output as required.

EperimentExtension:

- Explore and experiment with different matrix sizes, block sizes, and optimization techniques to observe the impact on performance and scalability.

- Consider using tools like Apache Mahout or optimizing the algorithm further to enhance the efficiency of matrix multiplication.

In this lab experiment, you have implemented matrix multiplication using HadoopMapReduce, You have learned how to divide the input matrices into blocks, performpartial multiplications in the map phase, and combine the results in the reduce phaseto obtain the final output matrix.

MatrisMultiplication Mapper.java:

```
import java.io.IOException:

import org.apache.hadoop.io.

import org.apache.hadoop.mapreduce.

public class Matrix MultiplicationMapper extends Mapper<Long Writable, Text, Text,

Text1

@Override

public void map(Long Writable key, Text value, Context context) throwsIOException,
InterruptedException

// Parse the input value to extract matrix information

String row value.toString.split( x [l ^ m]

String matrixName- row[0];

int rowIndex Integer.parseInt(row[1]):

int collndex-Integer.parseInt(row[21:

int cellValue-Integer.parseInt(row[3]

if (matrixName.equals("A")) {

// Emit intermediate key-value pairs for Matrix A
for (int k + 0 k < MatrixMultiplicationJob.numColsB; k++) {context.write(new
Text(rowindex +^ prime prime ,^ prime prime + k ) , new Text(matrixName+","

colindex + "," + cellValue));

} else {

// Emit intermediate key-value pairs for Matrix Bfor (int i=0;
i<MatrixMultiplicationJob.numRowsA; i++) {

context.write(new Text(i + "," + collndex), new Text(matrixName-+rowindex +^ n + +
cellValue));
}
}}
```

MatrixMultiplicationReducer.java:

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

public class MatrixMultiplicationReducer extends Reducer<Text, Text, Text
IntWritable> {

@Override

public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {

int[] rowA -new int[MatrixMultiplicationJob.numColsA];

int[] colB= new int[MatrixMultiplicationJob.numRowsB];

// Initialize arrays

for (int i=0; i<MatrixMultiplicationJob.numColsA; i++) { rowA[i] = 0;

for (int i=0; i< MatrixMultiplicationJob.numRowsB; i++) { }

// Populate rowA and colB arrays for (Text value: values) { String[] row
value.toString().split("."); String matrixName = row[0]; int index =
Integer.parseInt(row[1]);

int cellValue Integer.parseInt(row[2]);

if (matrixName.equals("A")) { rowA[index] cellValue; }
else{
```

```
          colB[index] cell Value;
}}
// Perform matrix multiplication and emit the result int result = 0; for (int i=0; i<
MatrixMultiplicationJob.numColsA; i++){ result + rowA[i] colB[i];

1

context.write(key, new IntWritable(result));
```

MatrixMultiplicationJob.java:

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

import org.apache.hadoop.io.*;

import org.apache.hadoop.mapreduce.*;

public class MatrixMultiplicationJob {

public static int numRowsA;

public static int numColsA;

public static int numRowsB;

public static int numColsB;

public static void main(String[] args) throws Exception { if (args.length != 4) {

System.err.println("Usage: MatrixMultiplicationJob <inputPathA> <inputPathB>
<outputPath> <numReducers>");

System.exit(1);

String inputPathA = args[0];

String inputPathB = args[1];

String outputPath = args[2];

int numReducers = Integer.parseInt(args[3]);

// Set the matric dimensions based on the input matrices Romitows A-getMatric
Dimension(inputPathA, 0): numColsA-getMatricDimension(inputPathA., 1); Rowall-
getMatrix Dimension(inputPathB, O monColch getMatrix Dimension(inputPath, 1)
Configuration conf-new Configuration():

Job job Job petlostance(conf, "Matrix Multiplication"

Job artarily Class(Matrix MaltiplicationJob.class); Job.setMapperClass(Matrix
MultiplicationMapper.class); job seifteducerClass(MatrixMultiplicationReducer.class);

job setMapOutputKeyClass(Text.class); job.setMapOutput ValueClass(Text.class);
job.setOutputKeyClass(Text.class); job setOutput ValueClass(IntWritable.class);

job setNumiteduce Tasks(mumReducers)

FiletoputFormat.addinputPath(job, new Path(inputPathA));
FilehoputFormat.addInputPath(job, new Path(inputPath());
FileOutputFormat.setOutputPath(job, new Path(outputPath));

System.exit(job.waitForCompletion(true) 70:1):

private static int getMatrix Dimension(String inputPath, int dimensionIndex) throws
10Exception(

Add code to read the input file and retrieve the matrix dimension based on the
dimensionIndex (0 for rows, 1 for columns)

// You can use Hadoop's FileSystem and BufferedReader to read the file and extract the
dimension information
}
}
```

Result :

Thus the implementation of matrix multiplication with hadoop MapReduce was implemented successfully.

Ex.No : 4 **Date:**	RUN A BASIC WORD COUNT MAP REDUCE PROGRAM TO UNDERSTAND MAP REDUCE PARADIGM.

Aim :

To write a run a basic word count map reduce program to understand map reduce paradigm.

Program :

To run this Word Count program, you'll need to follow these steps:

- Compile the Java code to generate the corresponding class files.
- Package the compiled class files into a JAR file.
- Transfer the JAR file to your Hadoop cluster or the machine where Hadoop is installed.
- Ensure that you have input data available in a text file.
- Open the terminal or command prompt and navigate to the Hadoop installation directory.

Run the following command to execute the Word Count program:

WordCountMapper.java:

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

public class WordCountMapper extends Mapper Long Writable, Text, Text, IntWritable> {

private final static IntWritable one- new IntWritable(1);

private Text word = new Text();

@Override

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

// Split the input line into words
```

String[] words= value.toString().split("\\s+");

// Emit intermediate key-value pairs (word, 1)
 for (String w: words) {
word.set(w);
west write(word, nee),

WordCountRedecer.java:

import java in JOException import org.apache.hadoop.io.. import org.apache.hadoop mapreduce.

public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable

private IntWritable result-new IntWritable();

@Override

public void reduce(Text key, Iterable<IntWritable values, Context context) throws IOException, InterruptedException [

int sum-0.

Sum up the counts for the same word for (IntWritable val: values) (sum+val get();

}

result.set(sum);

Emit the final key-value pair (word, count) context.write(key, result);

WordCountJob.java:

import org.apache.hadoop.conf.Configuration; import org.apache.hadoop.fs Path;

import org.apache.hadoop.io.*; import org.apache.hadoop.mapreduce.";

public class WordCountJob [

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

if (args.length 1-2) (

System.err.println("Usage: WordCountJob <inputPath outputPath" System.exit(1);

1

```
String inputPath= args[0];

String outputPath = args[1];

Configuration conf-new Configuration(): Job job Job.getInstance(conf, "Word Count");

job.setJarByClass(WordCountJob.class); job.setMapperClass(WordCountMapper.class);
job.setCombinerClass(WordCountReducer.class);
job.setReducerClass(WordCountReducer.class);

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

FileInputFormat.addInputPath(job, new Path(inputPath)); FileOutputFormat.setOutputPath(job,
new Path(outputPath));

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

Result :

Thus the basic word count mapreduce program to understand mapreduce paradigm completed successfully**.**

Ex.No : 5 **Date:**	INSTALLATION OF HIVE ALONG WITH PRACTICE EXAMPLES.

<u>Aim :</u>

To write installation of hive along with practice examples.

<u>Program :</u>

1. Installation of Hive :

   - Download the latest stable version of Apache Hive fires for official Apache Hive website (https://his.apache.org/)
   - Extract the downloaded file to a suitable location on your system.
   - Set the HIVE HOME environment variable to the Hive installation directory
   - Update the PATH environment variable to include the Hive binary directory

2. Configure Hive:

   - Navigate to the Hive configuration directory (SHIVE_HOME/com)
   - . Copy the hive default.xml.template file and rename it as hive-site.xml.
   - Open the hive-site.xml file and make necessary configuration changes if required, such as specifying the metastore database connection details.

3. Start Hadoop:

   - Ensure that Hadoop is running before starting Hive, as Hive relies on Hadoop for underlying storage and processing
   - Start Hadoop by executing the appropriate start commands for Hadoop's daemons (eg, NameNode, DataNode, ResourceManager, NodeManager) based on your Hadoop distribution.

4. Start Hive:

- Open a terminal or command prompt.
- Navigate to the Hive installation directory (SHIVE_HOME)
- Execute the following command to start the Hive shell:

```
$ bin/hive
```

- Hive shell will be launched, and you can start executing HiveQL queries.

Practice Examples

Once you have Hive up and running, you can start practicing with HiveQL queries and examples.

,Here are a few simple examples to get you started:

Create a table:

```
CREATE TABLE employees (

id INT,

name STRING

age INT,

department STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY

STORED AS TEXTFILE;
```

Load data into the table:

```
LOAD DATA LOCAL INPATH /path/to/input/file.csv' INTO TABLE employees;
```

Query the table:

```
SELECT FROM employees;
```

Perform aggregations:

```
SELECT department, COUNT(*) as count

FROM employees

GROUP BY department;
```

Create a new table from existing data:

```
CREATE TABLE department_count AS

SELECT department, COUNT(*) as count

FROM employees

GROUP BY department;
```

Join two tables:

```
SELECT ename, d.department

FROM employees e

JOIN departments d ON e.department-d.department_id
```

Result:

Hive along with practice examples wasinstalled and implemented successfully.

Ex.No : 6 **Date:**	INSTALLATION OF HBASE, INSTALLING THRIFT ALONG WITH PRACTICE EXAMPLES

Aim :

To write a installation of hbase, installing thrift along with practice examples

Program :

1. Install Apache HBase:

- Download the latest stable version of Apache HBase from the official Apache HBase website (https://hbase.apache.org/).
- Extract the downloaded file to a suitable location on your system.
- Set the HBASE_HOME environment variable to the HBase installation directory.
- Update the PATH environment variable to include the HBase binary directory.

2. Configure HBase:

- Navigate to the HBase configuration directory (SHBASE_HOME/conf).
- Open the hbase-site.xml file and make necessary configuration changes if required, such as specifying the ZooKeeper quorum.

3. Start Hadoop:

- Ensure that Hadoop is running before starting HBase, as HBase relies on Hadoop for underlying storage and processing.
- Start Hadoop by executing the appropriate start commands for Hadoop's daemons (e.g., NameNode, DataNode, ResourceManager, NodeManager) based on your Hadoop distribution.

4. Start HBase:

- Open a terminal or command prompt.
- Navigate to the HBase installation directory (SHBASE_HOME).
- Execute the following command to start HBase:

```
$ bin/start-hbase.sh
```

- HBase will start, and you can access the HBase shell and other HBase services.

5. Install Apache Thrift:

- Download the latest stable version of Apache Thrift from the official Apache Thrift website (https://thrift.apache.org/).
- Extract the downloaded file to a suitable location on your system.
- Set the THRIFT HOME environment variable to the Thrift installation directory
- Update the PATH environment variable to include the Thrift binary directory

6. Generate HBase Thrift bindings

- Navigate to the HBase installation directory (SHBASE HOME)
- Execute the following command to generate the HBase Thrift bindings

```
$ bin/hbase-thriftsh build java
```

- This command generates the necessary Thrift bindings for HBase.

7. Generate HBase Thrift bindings:

- Navigate to the HBase installation directory (SHBASE HOME)
- Execute the following command to generate the HBase Thrift bindings:
- 5 bin/base-daemon.sh start thrift
- The HBase Thrift server will start, and you can interact with HBase using Thrift

Practice Examples:

- Once you have Hase and Thrift up and running, you can start practicing with Hilase commands and Thrift APIs
- Here are a few simple examples to get you started:

Create a table:

```
create "my table', 'efl', 'eft"
```

Put data into the table:

```
put my table', 'row!', 'efl coll", "value"

put 'my table', 'row2", "cfl:col2', 'value2
```

Get data from the table:

```
get 'my table", "row"
```

scan the table:

```
scan 'my table"
```

Delete data from the table:

```
delete "my_table', 'rowl', 'efl:coll"
```

Result :

Thus the installation of Hbase and thrift along with practice examples was successfully implemented.

Ex.No : 7 **Date:**	PRACTICE IMPORTING AND EXPORTING DATA FROM VARIOUS DATABASES.

Aim :

To write a practice importing and exporting data from various databases.

Program :

1. Importing Data:

a. Hadoop Distributed File System (HDFS):

- Use the Hadoop hdfs dfs command-line tool or Hadoop File System API to copy data from a local file system or another location to HDFS. For example:

```
Shdfs dfs -put local file.txt /hdfs/path
```

- This command uploads the local file.txt from the local file system to the HDFS path /hdfs/path.

b. Apache Hive:

- Hive supports data import from various sources, including local files, HDFS, and databases. You can use the LOAD DATA statement to import data into Hive tables. For example:

- LOAD DATA INPATH /hdfs/path/data.txt' INTO TABLE my_table;

- This statement loads data from the HDFS path /hdfs/path/data.txt into the Hive table my table.

c. Apache Spark:

- Spark provides rich APIs for data ingestion. You can use the DataFrameReader or SparkSession APIs to read data from different sources such as CSV files, databases, or streaming systems. For example:

```
val df spark.read.format("esv").load("/path/to/data.csv")
```

- This code reads data from the CSV file located at /path/to/data.csv into a DataFrame in Spark.

2. Exporting Data:

a. Hadoop Distributed File System (HDFS):

- Use the Hadoop hdfs dfs command-line tool or Hadoop File System API to copy data from HDFS to a local file system or another location. For example:

```
$ hdfs dfs-get/hdfs/path/file.txt local_file.txt
```

- This command downloads the file /hdfs/path/file.txt from HDFS and saves it as local file.txt in the local file system.

b. Apache Hive:

- Exporting data from Hive can be done in various ways, depending on the desired output format. You can use the INSERT OVERWRITE statement to export data from Hive tables to files or other Hive tables. For example:

```
INSERT OVERWRITE LOCAL DIRECTORY /path/to/output SELECT
FROM my_table;
```

- This statement exports the data from the my_table Hive table to the local directory /path/to/output.

c.Apache Spark:

- Spark provides flexible options for data export. You can use the DataFrame Writer or Dataset Writer APIs to write data to different file formats, databases, or streaming systems. For example:

```
df.write.format("parquet").save("/path/to/output")
```

- This code saves the DataFrame df in Parquet format to the specified output directory.

Result :

Thus importing and exporting data from various databases was practiced successfully.