

Dr. AMBEDKAR INSTITUTE OF TECHNOLOGY

(An Autonomous Institute, Affiliated to Visvesvaraya Technological University, Belagavi, Accredited by NAAC, with 'A' Grade)



Near Jnana Bharathi Campus, Bengaluru – 560056

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING INFORMATION SCIENCE & ENGINEERING PROGRAMME

BIG DATA ANALYTICS LAB

SUBCODE: 22ISU701

REPORT

SUBMITTED BY

Faculty-in-charge: MALATHI P

Dr. AMBEDKAR INSTITUTE OF TECHNOLOGY

(An Autonomous Institute, Affiliated to Visvesvaraya Technological University, Belagavi, Accredited by NAAC, with 'A' Grade)

Near Jnana Bharathi Campus, Bengaluru – 560056



SCHOOL OF COMPUTER SCIENCE AND ENGINEERING INFORMATION SCIENCE & ENGINEERING PROGRAMME

Certificate

This is to certify that the **Big Data Analytics Laboratory Work** has been successfully carried out by ____ bearing USN: ____, a bonafide student of **Dr. Ambedkar Institute of Technology**, in partial fulfillment of the requirements for the award of the **Bachelor of Engineering degree in Information Science and Engineering** of **Visvesvaraya Technological University, Belagavi**, during the academic year **2025–2026**.

It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. The **Big Data Analytics Lab Manual** has been approved as it satisfies the academic requirements prescribed by **Visvesvaraya Technological University** for the said degree.

Signature of the Faculty In-charge

Mrs. Malathi P

Associate Professor,

ISE Programme,

Dr. AIT, Bengaluru -56.

Signature of the HOD

Dr. Prabha. R

Professor and HEAD,

ISE Programme,

Dr. AIT, Bengaluru – 56.

INDEX

Sl. No.	Contents	Page No.	Marks
1	Introduction to Hadoop Environment – Installation of VirtualBox and Cloudera QuickStart VM, Basic Hadoop Commands	4-8	
2	Introduction to HDFS Commands – Copy, List, and Display Files in Hadoop Distributed File System	9-9	
3	Program 1: Managing Files and Directories in HDFS – Move, Create, Delete, and Verify Operations	10-12	
4	Program 2: Hive Implementation on Movie Ratings Dataset – Creating Databases, Tables, and Executing Queries.	13-19	
5	Program3: Matrix Multiplication using MapReduce (Java Implementation)	20-26	
6.	Program 4: Develop a Map Reduce program that mines weather data and displays appropriate messages indicating the weather conditions of the day.	27-29	
7.	Program 5: Write Pig Latin scripts to sort, group, join, project, and filter the data.	30-34	
8.	Program 6: Write Pig Latin scripts to sort, group, join, project, and filter the data. MovieLens Data Analysis using MapReduce – Computing Average Movie Ratings	35-41	
9.	Program 7: Implement Functions: Count – Sort – Limit – Skip – Aggregate using MongoDB	42-52	
10.	Program 8: Word Count Implementation using Hadoop and Spark	53-54	

INTRODUCTION TO HADOOP INSTALLATION

Step 1: Download VirtualBox

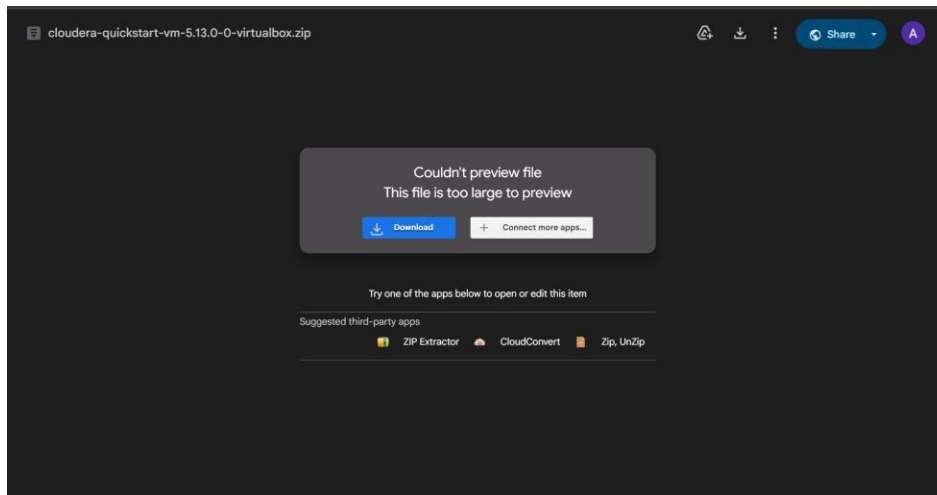
1. Go to the link: <https://www.virtualbox.org/wiki/Downloads>
2. Choose your Operating System (usually Windows) and click on it. The download will begin.
3. Once downloaded, run the installer. Click **Next** until the installation is complete, then click **Finish**.



Step 2: Download Cloudera QuickStart VM

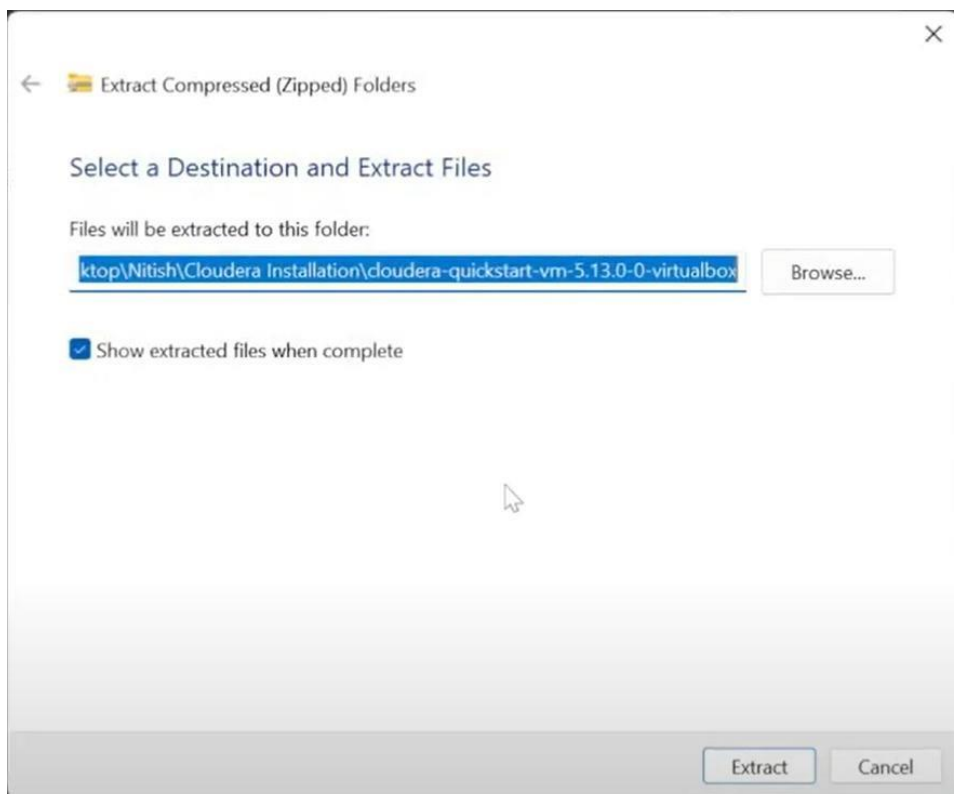
1. Go to the link: [Cloudera QuickStart VM Download](#).
2. Click on **Download** and wait until the process is completed.

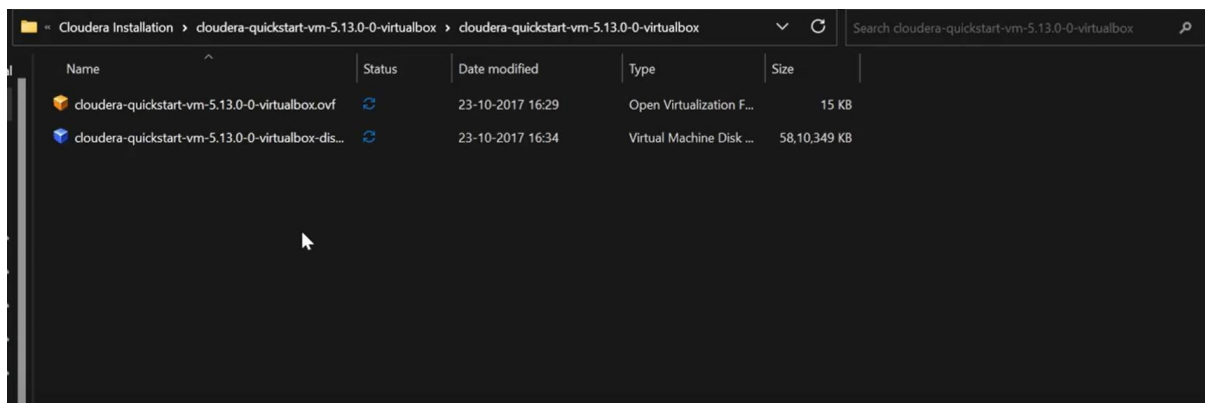
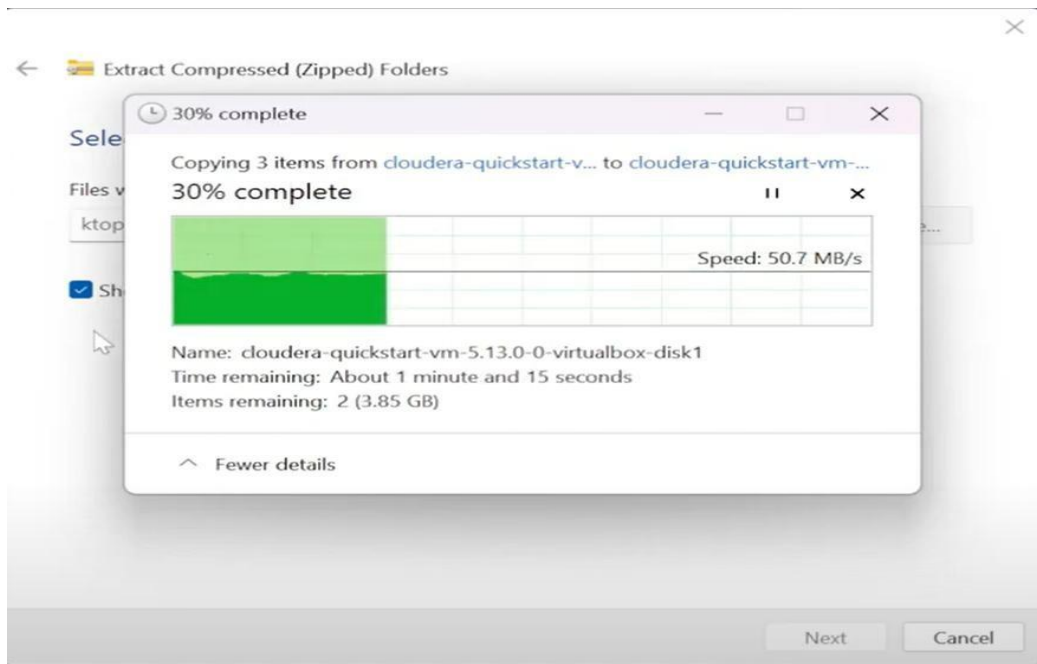
<https://drive.google.com/file/d/1yaoxB6bo8MbZfFkznKYfgy3Uza3lqI2G/view>



Step 3: Extract Cloudera QuickStart VM

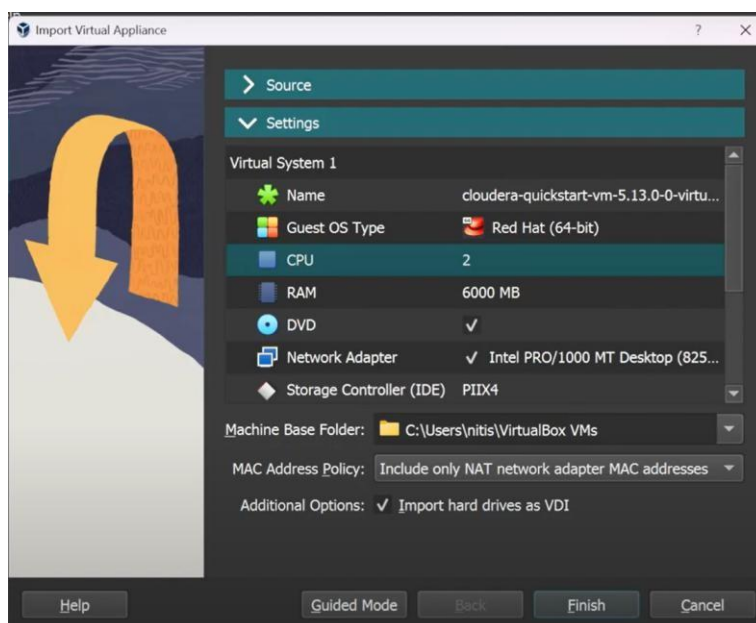
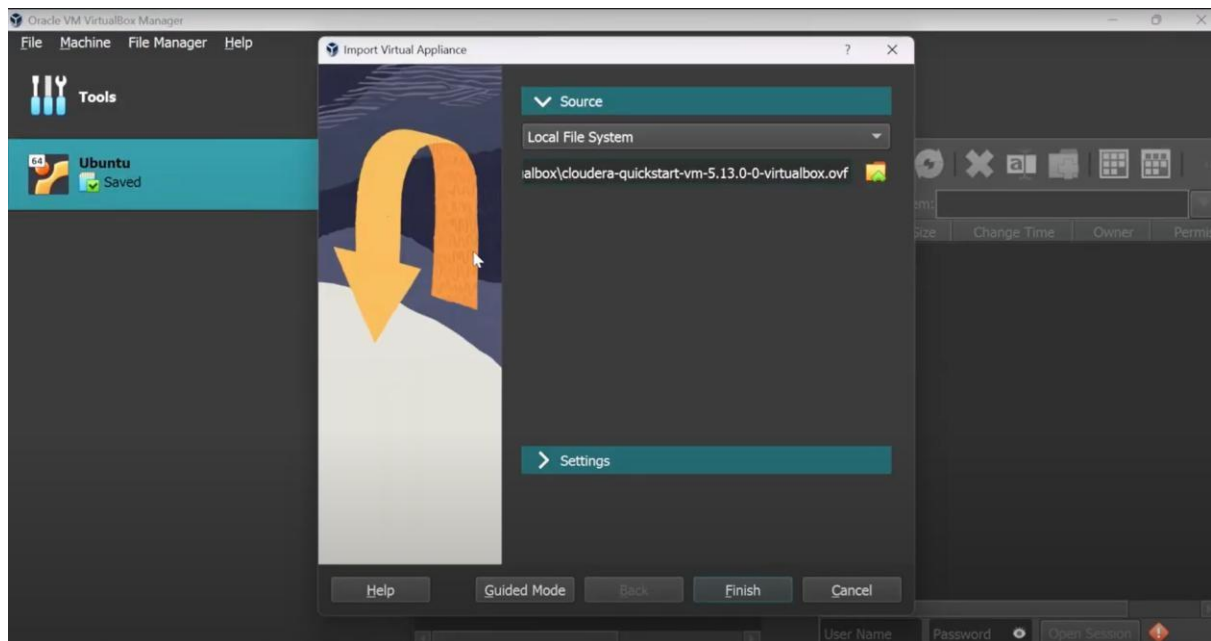
1. Go to your **Downloads** folder.
2. Right-click on the Cloudera zip file and choose **Extract All**.
3. After extraction, double-click on the first file to open it in VirtualBox.

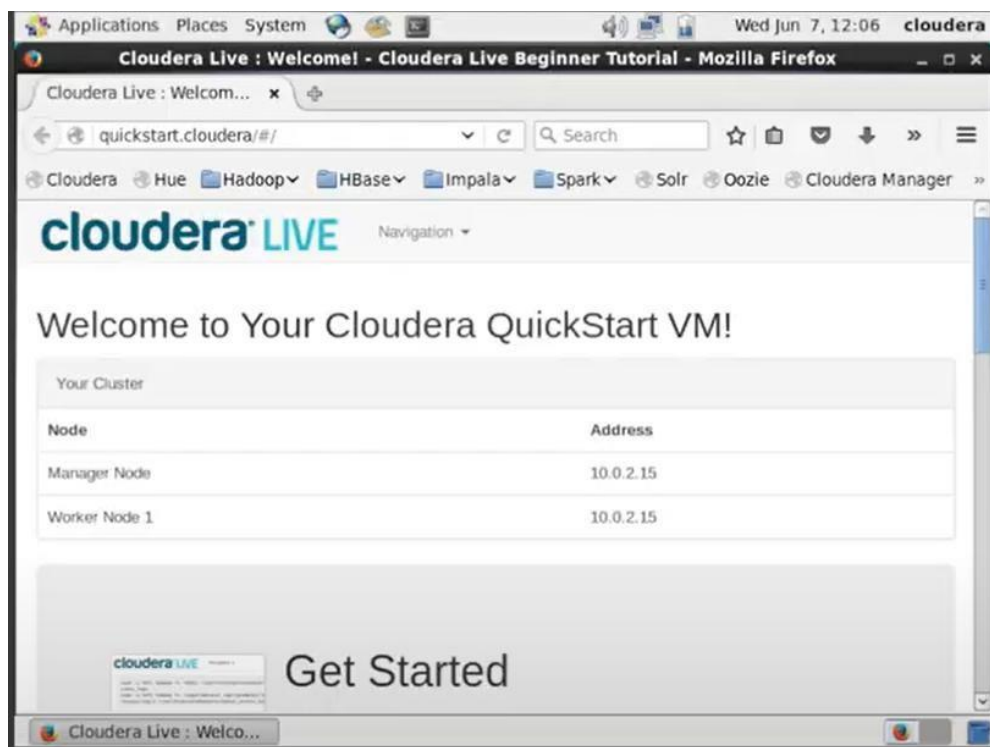
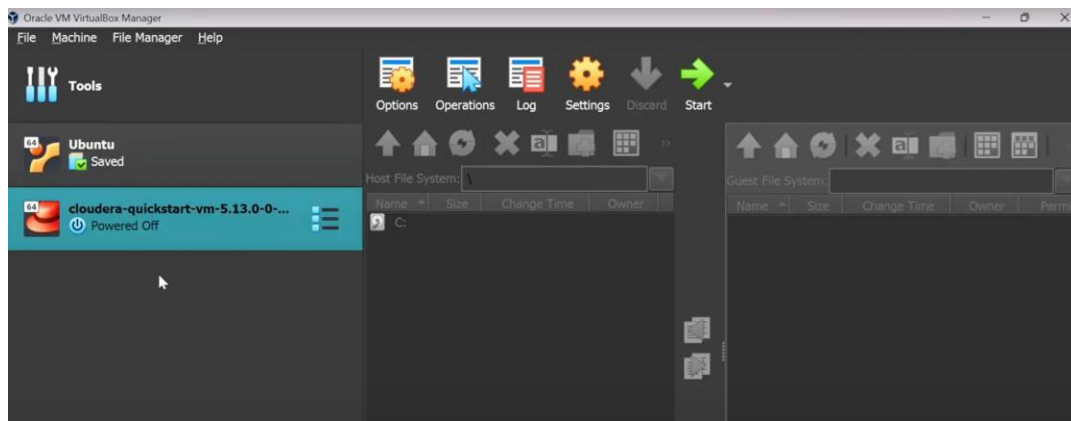


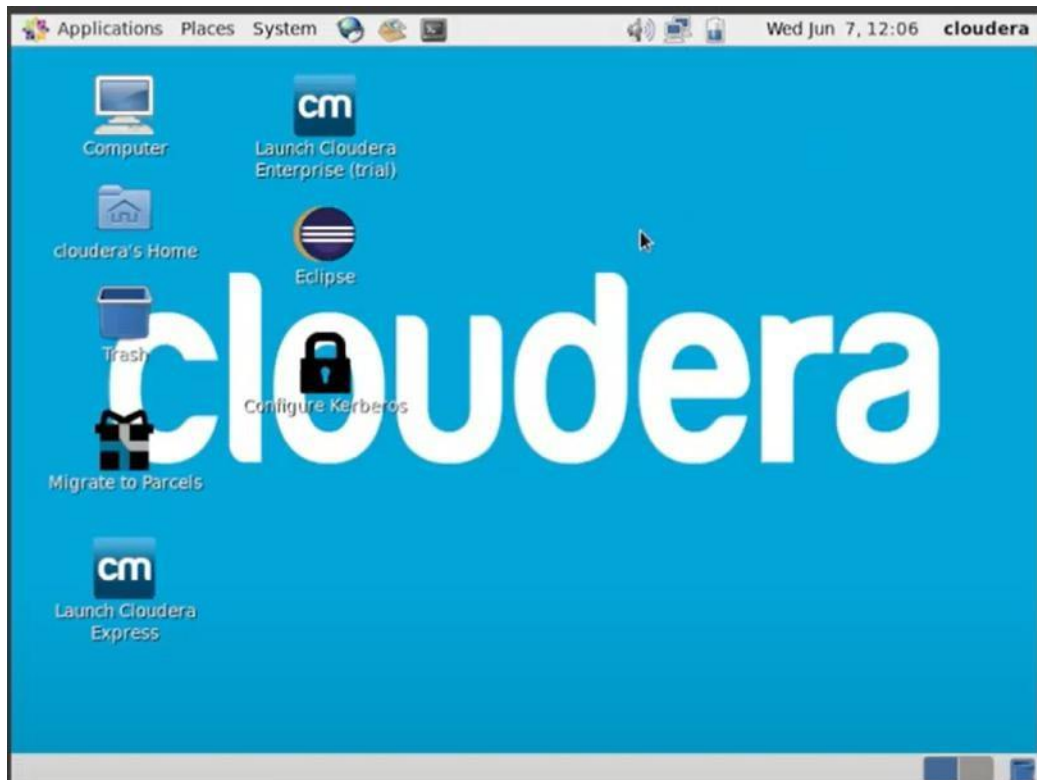


Step 4: Configure VM Settings

1. Click on **Settings** in VirtualBox.
2. Make the necessary changes as shown in the given images.
3. After clicking on **Finish**, it will take a while to import the file.
4. Once imported, double-click on the **Powered Off** VM to start it.







INTRODUCTION TO HDFS COMMANDS

Basic Hadoop Commands

1. Check Hadoop Version

hadoop version

Displays the installed Hadoop version.

2. List Files in HDFS Root Directory

hdfs dfs -ls /

3. Make a Directory in HDFS

hdfs dfs -mkdir /introduction

4. Upload a File to HDFS

hdfs dfs -put sample.txt /introduction

Uploads sample.txt from the local file system to HDFS /aishwarya.

4. View Contents of a File in HDFS

hdfs dfs -cat /introduction/sample.txt

6. Remove a File from HDFS

hdfs dfs -rm /introduction/sample.txt

Deletes the specified file from HDFS.

7. To create a file

Syntax: **vi<filename>**

Click I, you will be in insert mode then enter the contents/data to the file then

To save enter **Esc**, and exit by entering the command **:wq!**

PROGRAM 1

1.a) Start by reviewing HDFS. You will find that its composition is similar to your local Linux file system. You will use the `hadoop fs` command when interacting with HDFS.

a. Review the commands available for the Hadoop Distributed File System:

b. Copy file `foo.txt` from local disk to the user's directory in HDFS

c. Get a directory listing of the user's home directory in HDFS

d. Get a directory listing of the HDFS root directory

e. Display the contents of the HDFS file `user/fred/bar.txt`

a. Review the commands for HDFS

Command: **hadoop fs**

```
[cloudera@quickstart ~]$ hadoop fs
Usage: hadoop fs [generic options]
    [-appendToFile <localsrc> ... <dst>]
    [-cat [-ignoreCrc] <src> ...]
    [-checksum <src> ...]
    [-chgrp [-R] GROUP PATH...]
    [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
    [-chown [-R] [OWNER][:[GROUP]] PATH...]
    [-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
    [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
    [-count [-q] [-h] [-v] [-x] <path> ...]
    [-cp [-f] [-p] [-p[topax]] <src> ... <dst>]
    [-createSnapshot <snapshotDir> [<snapshotName>]]
    [-deleteSnapshot <snapshotDir> <snapshotName>]
    [-df [-h] <path> ...]
    [-du [-s] [-h] [-x] <path> ...]
    [-expunge]
    [-find <path> ... <expression> ...]
    [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
    [-getfacl [-R] <path>]
    [-getfattr [-R] {-n name | -d} [-e en] <path>]
    [-getmerge [-nl] <src> <localdst>]
    [-help [cmd ...]]
    [-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] <path> ...]
    [-mkdir [-p] <path> ...]
    [-moveFromLocal <localsrc> ... <dst>]
    [-moveToLocal <src> <localdst>]
    [-mv <src> ... <dst>]
    [-put [-f] [-p] [-l] <localsrc> ... <dst>]
    [-renameSnapshot <snapshotDir> <oldName> <newName>]
    [-rm [-f] [-r] [-R] [-skipTrash] <src> ...]
    [-rmdir [--ignore-fail-on-non-empty] <dir> ...]
    [-setfacl [-R] [{-b|-k} {-m|-x <acl_spec>} <path>][--set <acl_spec> <path>]]
    [-setfattr [-n name [-v value] | -x name] <path>]
    [-setrep [-R] [-w] <rep> <path> ...]
    [-stat [format] <path> ...]
    [-tail [-f] <file>]
    [-test [-defsz] <path>]
    [-text [-ignoreCrc] <src> ...]
    [-touchz <path> ...]
    [-usage [cmd ...]]

Generic options supported are
-conf <configuration file>      specify an application configuration file
-D <property=value>             use value for given property
-fs <local|namenode:port>       specify a namenode
-jt <local|resourcemanager:port> specify a ResourceManager
```

S

b. Copy file pgm.txt from local disk to user's directory in HDFS first create file pgm.txt

Command: **vi pgm.txt**

Type some contents in the file pgm1.txt, save and exit by pressing 'esc' button where you will come out of text box and then use the cmd ('**:wq!**' to save and exit).

In order to copy the file foo.txt from root directory to user directory in HDFS, use the

Command: **hadoop fs -put pgm.txt /user**

```
[cloudera@quickstart ~]$ vi pgm.txt
[cloudera@quickstart ~]$ hadoop fs -put pgm.txt /user
25/09/28 23:15:14 WARN hdfs.DFSClient: Caught exception
java.lang.InterruptedException
    at java.lang.Object.wait(Native Method)
    at java.lang.Thread.join(Thread.java:1281)
    at java.lang.Thread.join(Thread.java:1355)
    at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DFSOutputStream.java:967)
    at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.endBlock(DFSOutputStream.java:705)
    at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStream.java:894)
[cloudera@quickstart ~]$ hadoop fs -put pgm.txt /user
put: '/user/pgm.txt': File exists
```

c. Get a directory listing of the user's home directory in HDFS

Command: **hadoop fs -ls /user**

```
[cloudera@quickstart ~]$ hadoop fs -ls /user
Found 15 items
-rw-r--r-- 1 cloudera supergroup      11 2025-09-14 21:56 /user/aish
drwxr-xr-x - cloudera cloudera        0 2025-09-14 23:36 /user/cloudera
-rw-r--r-- 1 cloudera supergroup       6 2025-09-14 21:49 /user/foo
drwxr-xr-x - cloudera supergroup       0 2025-09-14 22:44 /user/fred
drwxr-xr-x - mapred  hadoop            0 2017-10-23 09:15 /user/history
drwxrwxrwx - hive    supergroup       0 2017-10-23 09:17 /user/hive
-rw-r--r-- 1 cloudera supergroup     25 2025-09-21 23:30 /user/hivedata
drwxrwxrwx - hue     supergroup       0 2017-10-23 09:16 /user/hue
drwxrwxrwx - jenkins supergroup       0 2017-10-23 09:15 /user/jenkins
drwxrwxrwx - oozie   supergroup       0 2017-10-23 09:16 /user/oozie
-rw-r--r-- 1 cloudera supergroup     19 2025-09-28 23:15 /user/pgm.txt
-rw-r--r-- 1 cloudera supergroup     20 2025-09-14 22:19 /user/pgm1.txt
-rw-r--r-- 1 cloudera supergroup     23 2025-09-14 22:10 /user/pgm2.txt
drwxrwxrwx - root    supergroup       0 2017-10-23 09:16 /user/root
drwxr-xr-x - hdfs    supergroup       0 2017-10-23 09:17 /user/spark
```

d. Get a directory listing of HDFS file

Command: **`hadoop fs -mkdir user/fred`**

`vi bar.txt`

`hadoop fs -put /root/bar.txt /user/fred`

```
Deleted user/fred
[cloudera@quickstart ~]$ hadoop fs -mkdir -p user/fred
[cloudera@quickstart ~]$ vi bar.txt
[cloudera@quickstart ~]$ hadoop fs -put bar.txt user/fred
25/09/28 23:30:34 WARN hdfs.DFSClient: Caught exception
```

e. Display the contents of the HDFS file user/fred/bar.txt

Command: **`hadoop fs -cat /user/fred`**

```
[cloudera@quickstart ~]$ hadoop fs -cat user/fred/bar.txt
this is text file owned by user fred
```

b) Start by reviewing HDFS. You will find that its composition is similar to your local Linux file system. You will use the `hadoop fs` command when interacting with HDFS.

a. Move that file to the local disk, named as `baz.txt`

b. Create a directory called `input` under the user's home directory

c. Delete the directory `input` old and all its contents

d. Verify the copy by listing the directory contents in HDFS.

a. Move the file to the local disk named as `baz.txt` (first create `baz.txt` using '`vi baz.txt`')
`vi baz.txt` (click I to enter insert mode, then enter the data then click ESC, type `:wq!` To save and exit)

```
[cloudera@quickstart ~]$ vi baz.txt
```

b. Create a directory called input under user's home directory

Command: **hadoop fs -mkdir /user/input**

c. Delete the directory input and all its contents

Command: **hadoop fs -rm -r /user/input**

d. Verify the copy by listing the directory contents in HDFS

Command: **hadoop fs -ls /user**

```
[cloudera@quickstart ~]$ vi baz.txt
[cloudera@quickstart ~]$ hadoop fs -mkdir /user/input
[cloudera@quickstart ~]$ hadoop dfs -rm -r /user/input
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

Deleted /user/input
[cloudera@quickstart ~]$
```

PROGRAM 2

PROBLEM STATEMENT:

Using movie ratings data, Develop the queries in Hive for the following.

- List all the Users who have rated the movies (Users who have rated at least one movie).
- List all the User with the max, min, average ratings they have given against any movie.
- List all the movies with the max, min, average ratings given by the user.

THEORY:

Hive is a data warehouse infrastructure tool to process structured data

in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying

and analysing easy. Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

PROCEDURE:

STEP 1: Create 3 text files movies, ratings and users and enter the data as shown in the screenshot.

Command: **vi user.txt;**

vi movie.txt;

vi rating.txt;

```
[cloudera@quickstart ~]$ vi user.txt
[cloudera@quickstart ~]$ vi movie.txt
[cloudera@quickstart ~]$ vi rating.txt
[cloudera@quickstart ~]$ hadoop fs -mkdir /user/cloudera
```

To save the file Press Esc → type :wq! → press Enter.

STEP 2: Copy files into HDFS

Create your HDFS directory (if not already):

Command: **hadoop fs -mkdir /user/cloudera**

upload the files into that directory

Command: **hadoop fs -put user.txt /user/cloudera**

hadoop fs -put movie.txt /user/cloudera

hadoop fs -put rating.txt /user/cloudera

```

[cloudera@quickstart ~]$ hadoop fs -put movie.txt /user/cloudera
[cloudera@quickstart ~]$ hadoop fs -put user.txt /user/cloudera
[cloudera@quickstart ~]$ hadoop fs -put rating.txt /user/cloudera
[cloudera@quickstart ~]$ hadoopfs -ls /user/cloudera
bash: hadoopfs: command not found
[cloudera@quickstart ~]$ hadoop fs -ls /user/cloudera
Found 3 items
-rw-r--r--    1 cloudera cloudera      55 2025-11-02 08:38 /user/cloudera/movi
e.txt
-rw-r--r--    1 cloudera cloudera      40 2025-11-02 08:38 /user/cloudera/rati
ng.txt
-rw-r--r--    1 cloudera cloudera      38 2025-11-02 08:38 /user/cloudera/user
.txt

```

To Confirm the files:

Command: **hadoop fs -ls /user/cloudera**

STEP 3: Open Hive

Command: **hive**

```

[cloudera@quickstart ~]$ hive

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
hive> create database movie_ratings;
OK
Time taken: 1.236 seconds
hive> use movie_ratings;
OK
Time taken: 0.068 seconds

```

Wait for the Hive shell (hive>) prompt.

STEP 4: Create database

Command: **create database movie_ratings;**

use movie_ratings;

STEP5: Create tables

(a) users table

create table users (

u_id int,

name string

)

row format delimited

fields terminated by '\t'

stored as textfile;

```
hive> create table users(  
  > u_id int,  
  > name string  
  > )  
  > row format delimited  
  > fields terminated by '\t'  
  > stored as textfile;  
OK
```

```
hive> create table movies(  
  > movie_id int,  
  > movie_name string  
  > )  
  > row format delimited  
  > fields terminated by '\t'  
  > stored as textfile;  
OK
```

(b) movies table

create table movies (

movie_id int,

movie_name string

)

row format delimited

fields terminated by '\t'

stored as textfile;

(c) ratings table

create table ratings (

u_id int,

movie_id int,

rating int

)

row format delimited

fields terminated by '\t'

stored as textfile;

```
hive> create table ratings(  
  > u_id int,  
  > movie_id int,  
  > rating int  
  > )  
  > row format delimited  
  > fields terminated by '\t'  
  > stored as textfile;  
OK  
Time taken: 0.058 seconds
```

STEP 6: Load data into the tables

Command: **load data inpath '/user/cloudera/user.txt' into table users;**

load data inpath '/user/cloudera/movie.txt' into table movies;

load data inpath '/user/cloudera/rating.txt' into table ratings;

```
hive> load data inpath '/user/cloudera/user.txt' into table users;  
Loading data to table movie_ratings.users  
Table movie_ratings.users stats: [numFiles=1, totalSize=38]  
OK  
Time taken: 0.453 seconds  
hive> load data inpath '/user/cloudera/movie.txt' into table movies;  
Loading data to table movie_ratings.movies  
Table movie_ratings.movies stats: [numFiles=1, totalSize=55]  
OK  
Time taken: 0.184 seconds  
hive> load data inpath '/user/cloudera/rating.txt' into table ratings;  
Loading data to table movie_ratings.ratings  
Table movie_ratings.ratings stats: [numFiles=1, totalSize=40]  
OK  
Time taken: 0.201 seconds  
hive> █
```

Check Data

Command: **select * from users;**

select * from movies;

select * from ratings;

```
hive> select * from users;
OK
1      Ammy
2      Bob
3      Celine
4      David
Time taken: 0.376 seconds, Fetched: 4 row(s)
hive> select * from movies;
OK
1      Inception
2      Titanic
3      Avengers
4      Fantastic4
NULL    NULL
Time taken: 0.086 seconds, Fetched: 5 row(s)
hive> select * from ratings;
OK
1      1      4
2      2      5
3      3      4
4      4      3
Time taken: 0.06 seconds, Fetched: 4 row(s)
```

QUERIES

a. List all the Users who have rated the movies (Users who have rated at least one movie).

Command: **select distinct u.u_id, u.name**

from users u

join ratings r on u.u_id = r.u_id;

```
hive> select distinct u.u_id, u.name
> from users u
> join ratings r on u.u_id=r.u_id;
Query ID = cloudera_20251102090505_6e3debdd-29e5-4acb-a2df-3a7dc24c220b
Total jobs = 1
Execution log at: /tmp/cloudera/cloudera_20251102090505_6e3debdd-29e5-4acb-a2df-3a7dc24c220b.log
2025-11-02 09:05:10 Starting to launch local task to process map join; maximum memory = 1013645312
2025-11-02 09:05:10 Dump the side-table for tag: 0 with group count: 4 into file: file:/tmp/cloudera/13cc5996-a32b-4101-851e00-..hashtable
2025-11-02 09:05:10 Uploaded 1 File to: file:/tmp/cloudera/13cc5996-a32b-4101-85fd-496bcc5686d0/hive_2025-11-02_09-05-06_3
2025-11-02 09:05:10 End of local task; Time Taken: 0.935 sec.
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1762100069545_0001, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1762100069545_0001/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1762100069545_0001
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2025-11-02 09:05:20,102 Stage-2 map = 0%, reduce = 0%
2025-11-02 09:05:25,468 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 1.13 sec
2025-11-02 09:05:31,753 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 2.0 sec
MapReduce Total cumulative CPU time: 2 seconds 0 msec
Ended Job = job_1762100069545_0001
MapReduce Jobs Launched:
Stage-Stage-2: Map: 1 Reduce: 1 Cumulative CPU: 2.0 sec HDFS Read: 10734 HDFS Write: 30 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 0 msec
OK
1      Ammy
2      Bob
3      Celine
4      David
Time taken: 26.462 seconds, Fetched: 4 row(s)
hive>
```

b. List all the User with the max, min, average ratings they have given against any movie.

Command: **select**

u.u_id,

```

u.name,

max(r.rating) as max_rating,

min(r.rating) as min_rating,

avg(r.rating) as avg_rating

from users u

join ratings r on u.u_id = r.u_id

group by u.u_id, u.name;

```

```

hive> select
>   u.u_id,
>   u.name,
>   max(r.rating) as max_rating,
>   min(r.rating) as min_rating,
>   avg(r.rating) as avg_rating
> from users u
> join ratings r on u.u_id = r.u_id
> group by u.u_id, u.name;
Query ID = cloudera_20251102091616_21c2d1c8-4a71-4275-ad8f-cf8c50bca484
Total jobs = 1
Execution log at: /tmp/cloudera/cloudera_20251102091616_21c2d1c8-4a71-4275-ad8f-cf8c50bca484.log
2025-11-02 09:16:12   Starting to launch local task to process map join;      maximum memory = 1013645312
2025-11-02 09:16:12   Dump the side-table for tag: 0 with group count: 4 into file: file:/tmp/cloudera/13cc5996-a32b-4101-85
le10--.hashtable
2025-11-02 09:16:12   Uploaded 1 File to: file:/tmp/cloudera/13cc5996-a32b-4101-85fd-496bcc5686d0/hive_2025-11-02_09-16-08_e
2025-11-02 09:16:12   End of local task; Time Taken: 0.958 sec.
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1762100069545_0002, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1762100069545_0002/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1762100069545_0002
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2025-11-02 09:16:19,621 Stage-2 map = 0%,   reduce = 0%
2025-11-02 09:16:26,981 Stage-2 map = 100%,   reduce = 0%, Cumulative CPU 1.24 sec
2025-11-02 09:16:33,279 Stage-2 map = 100%,   reduce = 100%, Cumulative CPU 2.14 sec
MapReduce Total cumulative CPU time: 2 seconds 140 msec
Ended Job = job_1762100069545_0002
MapReduce Jobs Launched:
Stage-Stage-2: Map: 1   Reduce: 1   Cumulative CPU: 2.14 sec   HDFS Read: 13386 HDFS Write: 62 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 140 msec
OK
1      Ammy      4      4      4.0
2      Bob       5      5      5.0
3      Celine    4      4      4.0
4      David     3      3      3.0
Time taken: 25.525 seconds, Fetched: 4 row(s)

```

c. List all the movies with the max, min, average ratings given by the user.

Command: *select*

```

m.movie_id,

m.movie_name,

max(r.rating) as max_rating,

min(r.rating) as min_rating,

avg(r.rating) as avg_rating

```

from movies m

join ratings r on m.movie_id = r.movie_id

group by m.movie_id, m.movie_name;

```
hive> select
>   m.movie_id,
>   m.movie_name,
>   max(r.rating) as max_rating,
>   min(r.rating) as min_rating,
>   avg(r.rating) as avg_rating
> from movies m
> join ratings r on m.movie_id = r.movie_id
> group by m.movie_id, m.movie_name;
```

Query ID = cloudera_20251102091818_7ab04bb8-5195-4d88-8fa7-e7a79a6e4909
Total jobs = 1
Execution log at: /tmp/cloudera/cloudera_20251102091818_7ab04bb8-5195-4d88-8fa7-e7a79a6e4909.log
2025-11-02 09:18:08 Starting to launch local task to process map join; maximum memory = 1013645312
2025-11-02 09:18:09 Dump the side-table for tag: 1 with group count: 4 into file: file:/tmp/cloudera/13cc5996-a32b-4101-8le21--.hashtable
2025-11-02 09:18:09 Uploaded 1 File to: file:/tmp/cloudera/13cc5996-a32b-4101-85fd-496bcc5686d0/hive_2025-11-02_09-18-05_
2025-11-02 09:18:09 End of local task; Time Taken: 0.927 sec.
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Starting Job = job_1762100069545_0003, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1762100069545_0003/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1762100069545_0003
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2025-11-02 09:18:15,484 Stage-2 map = 0%, reduce = 0%
2025-11-02 09:18:21,762 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.99 sec
2025-11-02 09:18:29,104 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 1.91 sec
MapReduce Total cumulative CPU time: 1 seconds 910 msec
Ended Job = job_1762100069545_0003
MapReduce Jobs Launched:
Stage-Stage-2: Map: 1 Reduce: 1 Cumulative CPU: 1.91 sec HDFS Read: 13441 HDFS Write: 78 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 910 msec
OK
1 Inception 4 4 4.0
2 Titanic 5 5 5.0
3 Avengers 4 4 4.0
4 Fantastic4 3 3 3.0
Time taken: 24.998 seconds, Fetched: 4 row(s)

PROGRAM 3

Develop a MapReduce program to implement Matrix Multiplication

Step 1: Create and move into a new directory

Command: **cd /home/cloudera**

mkdir -p matrix_job

cd matrix_job

pwd # /home/cloudera/matrix_job

```
[cloudera@quickstart ~]$ cd /home/cloudera/  
[cloudera@quickstart ~]$ mkdir -p matrix_job  
[cloudera@quickstart ~]$ cd matrix_job  
[cloudera@quickstart matrix_job]$ pwd  
/home/cloudera/matrix_job
```

Step 2: Create the Java source files

Command: **vi Map.java**

vi Reduce.java

vi MatrixMultiply.java

```
[cloudera@quickstart operation]$ vi Map.java  
[cloudera@quickstart operation]$ vi Reduce.java  
[cloudera@quickstart operation]$ vi MatrixMultiply.java
```

```
[cloudera@quickstart matrix_job]$ ls  
Map.java MatrixMultiply.java Reduce.java
```

Code:

a) Map.java

```
import org.apache.hadoop.conf.*;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Mapper;  
import java.io.IOException;  
  
public class Map extends org.apache.hadoop.mapreduce.Mapper<LongWritable, Text, Text,  
Text> {
```

@Override

```
public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
    Configuration conf = context.getConfiguration();
    int m = Integer.parseInt(conf.get("m"));
    int p = Integer.parseInt(conf.get("p"));

    String line = value.toString();
    String[] indicesAndValue = line.split(",");
    Text outputKey = new Text();
    Text outputValue = new Text();

    if (indicesAndValue[0].equals("M")) {
        // input line: M,i,j,Mij
        for (int k = 0; k < p; k++) {
            outputKey.set(indicesAndValue[1] + "," + k); // key = i,k
            outputValue.set(indicesAndValue[0] + "," + indicesAndValue[2] + "," +
indicesAndValue[3]); // M,j,Mij
            context.write(outputKey, outputValue);
        }
    } else {
        // input line: N,j,k,Njk
        for (int i = 0; i < m; i++) {
            outputKey.set(i + "," + indicesAndValue[2]); // key = i,k
            outputValue.set("N," + indicesAndValue[1] + "," + indicesAndValue[3]); // N,j,Njk
            context.write(outputKey, outputValue);
        }
    }
}
```

```
}
```

b) Reduce.java

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
import java.util.HashMap;

public class Reduce extends org.apache.hadoop.mapreduce.Reducer<Text, Text, Text, Text> {
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context)
        throws IOException, InterruptedException {
        String[] value;
        HashMap<Integer, Float> hashA = new HashMap<Integer, Float>();
        HashMap<Integer, Float> hashB = new HashMap<Integer, Float>();

        for (Text val : values) {
            value = val.toString().split(",");
            if (value[0].equals("M")) {
                // value = ["M", j, Mij]
                hashA.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));
            } else {
                // value = ["N", j, Njk]
                hashB.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));
            }
        }

        int n = Integer.parseInt(context.getConfiguration().get("n"));
        float result = 0.0f;
        float m_ij;
        float n_jk;
```

```

    for (int j = 0; j < n; j++) {
        m_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;
        n_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;
        result += m_ij * n_jk;
    }

    if (result != 0.0f) {
        context.write(null, new Text(key.toString() + "," + Float.toString(result)));
    }
}
}

```

c) **MatrixMultiply.java**

```

import org.apache.hadoop.conf.*;
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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class MatrixMultiply {

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

        if (args.length != 2) {

            System.err.println("Usage: MatrixMultiply <in_dir> <out_dir>");

            System.exit(2);

        }

        Configuration conf = new Configuration();

```

```

conf.set("m", "2"); // example: M has 2 rows
conf.set("n", "2"); // inner dimension
conf.set("p", "2"); // N has 2 columns

```

```

@SuppressWarnings("deprecation")

Job job = new Job(conf, "MatrixMultiply");

job.setJarByClass(MatrixMultiply.class);

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

job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);

job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);

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

job.waitForCompletion(true);
}
}

```

Step 3: Compile Java files

Command: **rm -rf matrix_classes MatrixMultiplyMatrix.jar**

mkdir -p matrix_classes

javac -classpath `hadoop classpath` -d matrix_classes Map.java Reduce.java MatrixMultiply.java

ls -R matrix_classes

```

[cloudera@quickstart matrix_job]$ rm -rf matrix_classes MatrixMultiplyMatrix.jar
[cloudera@quickstart matrix_job]$ mkdir -p matrix_classes
[cloudera@quickstart matrix_job]$ javac -classpath `hadoop classpath` -d matrix_
classes Map.java Reduce.java MatrixMultiply.java
[cloudera@quickstart matrix_job]$ ls -R matrix_classes
matrix_classes:
Map.class  MatrixMultiply.class  Reduce.class

```

Step 4: Create a jar

Command: **jar -cvf MatrixMultiplyMatrix.jar -C matrix_classes .**

jar tf MatrixMultiplyMatrix.jar | head

```
[cloudera@quickstart matrix_job]$ jar -cvf MatrixMultiplyMatrix.jar -C matrix_classes .
added manifest
adding: Map.class(in = 2333) (out= 986)(deflated 57%)
adding: MatrixMultiply.class(in = 1726) (out= 900)(deflated 47%)
adding: Reduce.class(in = 2899) (out= 1253)(deflated 56%)
```

```
[cloudera@quickstart matrix_job]$ jar tf MatrixMultiplyMatrix.jar | head
META-INF/
META-INF/MANIFEST.MF
Map.class
MatrixMultiply.class
Reduce.class
```

Step 5: Create & upload inputs(in HDFS directory or local)

Command: **cat > M <<'EOF'**

M,0,0,1

M,0,1,2

M,1,0,3

M,1,1,4

EOF

cat > N <<'EOF'

N,0,0,5

N,0,1,6

N,1,0,7

N,1,1,8

EOF

hdfs dfs -mkdir -p /user/cloudera/matrix_input

hdfs dfs -put -f M /user/cloudera/matrix_input/

hdfs dfs -put -f N /user/cloudera/matrix_input/

hdfs dfs -ls /user/cloudera/matrix_input # verify

```
[cloudera@quickstart matrix_job]$ cd ..
[cloudera@quickstart ~]$ pwd
/home/cloudera
[cloudera@quickstart ~]$ cat > M <<'EOF'
> M,0,0,1
> M,0,1,2
> M,1,0,3
> M,1,1,4
> EOF
```

```
[cloudera@quickstart ~]$ cat > N <<'EOF'
> N,0,0,5
> N,0,1,6
> N,1,0,7
> N,1,1,8
> EOF
```

```
[cloudera@quickstart ~]$ cat N
N,0,0,5
N,0,1,6
N,1,0,7
N,1,1,8
[cloudera@quickstart ~]$ cat M
M,0,0,1
M,0,1,2
M,1,0,3
M,1,1,4
[cloudera@quickstart ~]$ hdfs dfs -mkdir -p /
```

```
[cloudera@quickstart ~]$ hdfs dfs -mkdir -p /user/cloudera/matrix_input
[cloudera@quickstart ~]$ hdfs dfs -put -f M /user/cloudera/matrix_input/
```

```
[cloudera@quickstart ~]$ hdfs dfs -put -f N /user/cloudera/matrix_input/
[cloudera@quickstart ~]$ hdfs dfs -ls /user/cloudera/matrix_input
Found 2 items
-rw-r--r-- 1 cloudera cloudera      32 2025-11-05 22:26 /user/cloudera/matrix_input/M
-rw-r--r-- 1 cloudera cloudera      32 2025-11-05 22:27 /user/cloudera/matrix_input/N
```

Step 6: Run the MapReduce job

Command: **hdfs dfs -rm -r -f /user/cloudera/matrix_output**

hadoop jar MatrixMultiplyMatrix.jar MatrixMultiply /user/cloudera/matrix_input /user/cloudera/matrix_output

```
[cloudera@quickstart matrix_job]$ hdfs dfs -rm -r -f /user/cloudera/matrix_output
```

```
[cloudera@quickstart matrix_job]$ hadoop jar MatrixMultiplyMatrix.jar MatrixMultiply /user/cloudera/matrix_input /user/cloudera/matrix_output
25/11/05 22:36:51 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
25/11/05 22:36:52 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
25/11/05 22:36:52 INFO input.FileInputFormat: Total input paths to process : 2
25/11/05 22:36:52 INFO mapreduce.JobSubmitter: number of splits:2
25/11/05 22:36:53 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1762362989980_0002
25/11/05 22:36:53 INFO impl.YarnClientImpl: Submitted application application_1762362989980_0002
25/11/05 22:36:53 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1762362989980_0002/
25/11/05 22:36:53 INFO mapreduce.Job: Running job: job_1762362989980_0002
25/11/05 22:37:03 INFO mapreduce.Job: Job job_1762362989980_0002 running in uber mode : false
25/11/05 22:37:03 INFO mapreduce.Job: map 0% reduce 0%
25/11/05 22:37:25 INFO mapreduce.Job: map 100% reduce 0%
25/11/05 22:37:33 INFO mapreduce.Job: map 100% reduce 100%
25/11/05 22:37:33 INFO mapreduce.Job: Job job_1762362989980_0002 completed successfully
25/11/05 22:37:33 INFO mapreduce.Job: Counters: 49
```

Step 7: Check results

Command: **hdfs dfs -ls /user/cloudera/matrix_output**

hdfs dfs -cat /user/cloudera/matrix_output/part-r-00000

```
bytes written=0
[cloudera@quickstart matrix_job]$ hdfs dfs -ls /user/cloudera/matrix_output
Found 2 items
-rw-r--r-- 1 cloudera cloudera      0 2025-11-05 22:37 /user/cloudera/matrix_output/_SUCCESS
-rw-r--r-- 1 cloudera cloudera     36 2025-11-05 22:37 /user/cloudera/matrix_output/part-r-00000
```

```
[cloudera@quickstart matrix_job]$ hdfs dfs -cat /user/cloudera/matrix_output/part-r-00000
0,0,19.0
0,1,22.0
1,0,43.0
1,1,50.0
[cloudera@quickstart matrix_job]$
```

PROGRAM 4

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

Step 1: Create the input CSV.

Command: **cat > weatherdata.txt <<'EOF'**

Date,MaxTemp,MinTemp

2025-11-01,35,22

2025-11-02,40,25

2025-11-03,30,20

2025-11-04,18,10

2025-11-05,25,17

2025-11-06,42,28

EOF

head -n 20 weatherdata.txt

Step 2: Create project directory

Command: **cd /home/cloudera**

mkdir -p weather_job

cd weather_job

```
[cloudera@quickstart ~]$ cd /home/cloudera
[cloudera@quickstart ~]$ mkdir -p weather_job
[cloudera@quickstart ~]$ cd weather_job
[cloudera@quickstart weather_job]$ pwd
/home/cloudera/weather_job
```

```
[cloudera@quickstart ~]$ cat > weatherdata.txt <<'EOF'
> Date,MaxTemp,MinTemp
> 2025-11-01,35,22
> 2025-11-02,40,25
> 2025-11-03,30,20
> 2025-11-04,18,10
> 2025-11-05,25,17
> 2025-11-06,42,28
> EOF
[cloudera@quickstart ~]$ head -n 20 weatherdata.txt
Date,MaxTemp,MinTemp
2025-11-01,35,22
2025-11-02,40,25
2025-11-03,30,20
2025-11-04,18,10
2025-11-05,25,17
2025-11-06,42,28
```

Step 3: Create the mapper and reducer Python scripts

Command: **vi weather_mapper.py**

chmod +x weather_mapper.py #Make it executable:

vi weather_reducer.py

chmod +x weather_reducer.py #Make it executable

```
[cloudera@quickstart weather_job]$ vi weather_mapper.py
[cloudera@quickstart weather_job]$ chmod +x weather_mapper.py
[cloudera@quickstart weather_job]$ vi weather_reducer.py
[cloudera@quickstart weather_job]$ chmod +x weather_reducer.py
```

Code:

```
File Edit View Search Terminal Help
#!/usr/bin/env python
import sys

for line in sys.stdin:
    line = line.strip()
    if not line or line.startswith("Date"):
        continue
    parts = line.split(",")
    if len(parts) < 2:
        continue
    date = parts[0]
    max_temp = parts[1]
    print("%s\t%s" % (date, max_temp))
```

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

for line in sys.stdin:
    line = line.strip()
    if not line:
        continue
    parts = line.split('\t')
    if len(parts) != 2:
        continue
    date, max_temp_str = parts
    try:
        max_temp = float(max_temp_str)
    except:
        continue

    if max_temp > 35:
        condition = "Hot Day"
    elif max_temp < 20:
        condition = "Cold Day"
    else:
        condition = "Normal Day"
```

"weather reducer.py" 25L 486C

Step 4: Upload weatherdata.txt to HDFS

Command: **hdfs dfs -mkdir -p /user/cloudera/weather_input**

hdfs dfs -put -f /home/cloudera/weatherdata.txt /user/cloudera/weather_input/

hdfs dfs -ls /user/cloudera/weather_input # verify on HDFS

hdfs dfs -cat /user/cloudera/weather_input/weatherdata.txt | head -n 20

```
[cloudera@quickstart ~]$ hdfs dfs -mkdir -p /user/cloudera/weather_input
[cloudera@quickstart ~]$ hdfs dfs -put -f /home/cloudera/weatherdata.txt /user/c
loudera/weather_input/
[cloudera@quickstart ~]$ hdfs dfs -ls /user/cloudera/weather_input
Found 1 items
-rw-r--r-- 1 cloudera cloudera 123 2025-11-07 20:27 /user/cloudera/weat
her_input/weatherdata.txt
[cloudera@quickstart ~]$ hdfs dfs -cat /user/cloudera/weather_input/weatherdata.
txt | head -n 20
Date,MaxTemp,MinTemp
2025-11-01,35,22
2025-11-02,40,25
2025-11-03,30,20
2025-11-04,18,10
2025-11-05,25,17
2025-11-06,42,28
```

Step 5: Run the Hadoop Streaming job.(Change to project dir and run)

hdfs dfs -rm -r -f /user/cloudera/weather_output

```
STREAMING=$(find / -name "*hadoop-streaming*.jar" 2>/dev/null | head -n 1)
```

```
hadoop jar "$STREAMING" \
```

```
-files weather_mapper.py,weather_reducer.py \
```

```
-mapper "python weather_mapper.py" \
```

```
-reducer "python weather_reducer.py" \
```

```
-input /user/cloudera/weather_input/weatherdata.txt \
```

```
-output /user/cloudera/weather_output
```

```
[cloudera@quickstart weather_job]$ hdfs dfs -rm -r -f /user/cloudera/weather_output
Deleted /user/cloudera/weather_output
[cloudera@quickstart weather_job]$ STREAMING=$(find / -name "*hadoop-streaming*.jar"
2>/dev/null | head -n 1)
[cloudera@quickstart weather_job]$ hadoop jar "$STREAMING" \
> -files weather_mapper.py,weather_reducer.py \
> -mapper "python weather_mapper.py" \
> -reducer "python weather_reducer.py" \
> -input /user/cloudera/weather_input/weatherdata.txt \
> -output /user/cloudera/weather_output
```

Step 5: Check output.

Command: **hdfs dfs -cat /user/cloudera/weather_output/part-00000**

```
Bytes Written=124
25/11/08 00:21:37 INFO streaming.StreamJob: Output directory: /user/cloudera/weather_output
[cloudera@quickstart weather_job]$ hdfs dfs -cat /user/cloudera/weather_output/part-00000
2025-11-01      Normal Day
2025-11-02      Hot Day
2025-11-03      Normal Day
2025-11-04      Cold Day
2025-11-05      Normal Day
2025-11-06      Hot Day
```

PROGRAM 5

PROBLEM STATEMENT

1. Start the Grunt shell and execute the following statements to set up a dataflow- Create and insert into MovieRating table. Check the schema of table created.
2. Sort the records of MovieRating table into a separate table.
3. GROUP the records of MovieRating table into a separate table.
4. Export records of the table into a data file.

Apache Pig is an abstraction over MapReduce. It is a tool/platform which is used to analyze larger sets of data representing them as data flows. Pig is generally used with Hadoop; we can perform all the data manipulation operations in Hadoop using Pig.

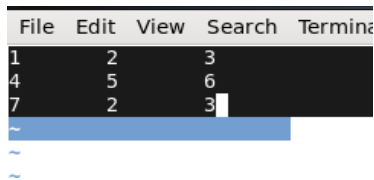
- Using Pig Latin, programmers can perform MapReduce tasks easily without having to type complex codes in Java.
- Apache Pig uses multi-query approach, thereby reducing the length of codes. For example, an operation that would require you to type 200 lines of code (LoC) in Java can be easily done by typing as less as just 10 LoC in Apache Pig. Ultimately Apache Pig reduces the development time by almost 16 times.
- Pig Latin is SQL-like language and it is easy to learn Apache Pig when you are familiar with SQL.
- Apache Pig provides many built-in operators to support data operations like joins, filters, ordering, etc. In addition, it also provides nested data types like tuples, bags, and maps that are missing from MapReduce.

1. Start the Grunt shell and execute the following statements to set up a dataflow- Create and insert into MovieRating table. Check the schema of table created

Command : ***vi sample***

```
[cloudera@quickstart ~]$ vi sample
[cloudera@quickstart ~]$
```

2. Enter the data by using tab as a delimiter.



3. Start a Grunt shell using a command shown below which will take you in local mode

Command: **pig -x local**

```
[cloudera@quickstart ~]$ pig -x local
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
2025-10-12 21:43:03.666 [main] INFO org.apache.pig.Main - Apache Pig version 0.12.0-cdh5.13.0 (reexported) compiled Oct 04 2017, 11:09:03
2025-10-12 21:43:03.666 [main] INFO org.apache.pig.Main - Logging error messages to: /home/cloudera/pig_1760330583656.log
2025-10-12 21:43:03.682 [main] INFO org.apache.pig.impl.util.Utils - Default bootstrap file /home/cloudera/pigbootstrap not found
2025-10-12 21:43:03.851 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-10-12 21:43:03.860 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2025-10-12 21:43:03.867 [main] INFO org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: file:///
2025-10-12 21:43:04.197 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2025-10-12 21:43:04.329 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-10-12 21:43:04.330 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2025-10-12 21:43:04.407 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2025-10-12 21:43:04.407 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-10-12 21:43:04.408 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2025-10-12 21:43:04.408 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2025-10-12 21:43:04.537 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-10-12 21:43:04.537 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2025-10-12 21:43:04.538 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2025-10-12 21:43:04.619 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-10-12 21:43:04.639 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2025-10-12 21:43:04.639 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2025-10-12 21:43:04.713 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-10-12 21:43:04.713 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2025-10-12 21:43:04.714 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2025-10-12 21:43:04.795 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-10-12 21:43:04.796 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2025-10-12 21:43:04.803 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2025-10-12 21:43:04.850 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-10-12 21:43:04.851 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2025-10-12 21:43:04.851 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2025-10-12 21:43:04.919 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-10-12 21:43:04.920 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2025-10-12 21:43:04.920 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
grunt>
```



The Grunt shell of Apache Pig is mainly used to write Pig Latin scripts. Prior to that, we can invoke any shell commands using sh.

We can invoke the ls command of Linux shell from the Grunt shell using the sh option as shown below.

Command: **sh ls**



```

grunt> sh ls
aish
bar.txt
baz.txt
cloudera-manager
cm_api.py
Desktop
Documents
Downloads
eclipse
enterprise-deployment.json
express-deployment.json
kerberos
lab1.txt
lib
movies.csv
movie.txt
Music
parcels
pgm.txt
Pictures
pig_1758866421787.log
Public
ratings.csv
rating.txt
sample
samp.sql
Templates
testfile.txt
user.txt
Videos
workspace
YOUTPUT
grunt>

```

5. Load the sample file created into a pig storage by executing below command and describe the same.

Command: ***movierating = LOAD `sample` Using PigStorage('\t') as (cid :int, mid:int,rating:int);***

Describe moviearting;

```

grunt> movierating = LOAD 'sample' Using PigStorage('\t') as (cid:int,mid:int,rating:int);
grunt>

```

```

grunt> Describe movierating;
movierating: {cid: int,mid: int,rating: int}
grunt>

```

6. Dump the file content to movie rating by executing dump movierating .

Command: ***dump movierating;***

```
grunt> dump movierating;
2025-10-12 21:49:23,301 [main] INFO org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: UNKNOWN
2025-10-12 21:49:23,320 [main] INFO org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach, ColumnMapKeyPrune, DuplicateFilter, LoadTypeCastInserter, MergeFilter, MergeForEach, NewPartitionFilterOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFilter, StreamTypeCastInserter]}
```

```
(1,2,3)
(4,5,6)
(7,2,3)
grunt>
```

7. Sort the records of MovieRating table into a separate table.

The **ORDER BY** operator is used to display the contents of a relation in a sorted order based on one or more fields.

To order the movie ID execute the bellow command and once you describe it will show the sorted elements as shown below.

Command: ***moviegroup = Order movierating by mid desc;***

Describe moviegroup;

Dump moviegroup;

```
details at logfile: /home/cloudera/pig 1/003300383030.log
grunt> moviegroup = Order movierating by mid desc;
grunt> Describe moviegroup;
moviegroup: {cid: int,mid: int,rating: int}
grunt>
```

```
grunt> dump moviegroup;
2025-10-12 21:55:33,456 [main] INFO org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: ORDER BY
2025-10-12 21:55:33,456 [main] INFO org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach, ColumnMapKeyPrune, DuplicateFilter, LoadTypeCastInserter, MergeFilter, MergeForEach, NewPartitionFilterOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFilter, StreamTypeCastInserter]}
```

```
2025-10-12 21:55:33,456 [main] INFO org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach, ColumnMapKeyPrune, DuplicateFilter, LoadTypeCastInserter, MergeFilter, MergeForEach, NewPartitionFilterOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFilter, StreamTypeCastInserter]}
(4,5,6)
(7,2,3)
(1,2,3)
grunt>
```

8. GROUP the records of MovieRating table into a separate table.

The GROUP operator is used to group the data in one or more relations. It collects the data having the same key.

Group is done by giving the following command

Command: ***movierating2 = group movierating by rating;***

Describe moviearting2;

```
grunt> movierating2 = group movierating by rating;
grunt> describe movierating2;
movierating2: {group: int,movierating: {(cid: int,mid: int,rating: int)}}
grunt>
```

Dump movierating2;

```
grunt> dump movierating2;
2025-10-12 22:01:44,519 [main] INFO org.apache.pig.tools.pigstats.ScriptState - P
2025-10-12 22:01:44,520 [main] INFO org.apache.pig.newplan.logical.optimizer.Logi
timizer, LoadTypeCastInserter, MergeFilter, MergeForEach, NewPartitionFilterOptimi
}

2025-10-12 22:01:57,112 [main] INFO org.apa
(3,{(7,2,3),(1,2,3)})
(6,{(4,5,6)})
grunt>
```

9. Export records of the table into a data file.

We can store the loaded data in the file system using the **store** operator.

Command : **STORE moviegroup INTO `YOUTPUT` using PigStorage(`\t`);**

```
grunt> STORE moviegroup INTO 'YOUTPUT' using PigStorage('\t');
```

After executing the **store** statement, you will get the following output.

```
grunt> ls
file:/home/cloudera/.bash_logout<r 1> 18
file:/home/cloudera/.gconfd <dir>
file:/home/cloudera/.mozilla <dir>
file:/home/cloudera/parcels<r 1> 4228
file:/home/cloudera/user.txt<r 1> 28
file:/home/cloudera/Music <dir>
file:/home/cloudera/.esd_auth<r 1> 16
file:/home/cloudera/bar.txt<r 1> 38
file:/home/cloudera/.ICEauthority<r 1> 310
file:/home/cloudera/.hivehistory<r 1> 475
file:/home/cloudera/pig_1760330583656.log<r 1> 15478
file:/home/cloudera/Downloads <dir>
file:/home/cloudera/.vboxclient-draganddrop.pid<r 1> 5
file:/home/cloudera/movie.txt<r 1> 35
file:/home/cloudera/YOUTPUT <dir>
file:/home/cloudera/lib <dir>
file:/home/cloudera/sample<r 1> 18
file:/home/cloudera/Documents <dir>
file:/home/cloudera/testfile.txt<r 1> 0
file:/home/cloudera/lab1.txt<r 1> 11
file:/home/cloudera/.vboxclient-clipboard.pid<r 1> 5
file:/home/cloudera/Pictures <dir>
file:/home/cloudera/.gstreamer-0.10 <dir>
file:/home/cloudera/samp.sql<r 1> 50
file:/home/cloudera/.nautilus <dir>
file:/home/cloudera/.bash_profile<r 1> 176
file:/home/cloudera/.pulse-cookie<r 1> 256
file:/home/cloudera/.cache <dir>
file:/home/cloudera/baz.txt<r 1> 47
file:/home/cloudera/aish<r 1> 11
file:/home/cloudera/express-deployment.json<r 1> 50515
file:/home/cloudera/workspace <dir>
```

PROGRAM 6

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

Step 1: Create a working directory on the VM

Command: **cd /home/cloudera**

mkdir -p movielens

cd movielens

pwd # /home/cloudera/movielens

```
[cloudera@quickstart ~]$ cd /home/cloudera
[cloudera@quickstart ~]$ cd movielens
[cloudera@quickstart movielens]$ pwd
/home/cloudera/movielens
```

Step 2: Create the Java source files in the working directory

Command: **nano MovieRatingMapper.java**

nano MovieRatingReducer.java

nano MovieRatings.java

```
[cloudera@quickstart movielens]$ nano MovieRatingMapper.java
[cloudera@quickstart movielens]$ nano MovieRatingReducer.java
[cloudera@quickstart movielens]$ nano MovieRatings.java
[cloudera@quickstart movielens]$ ls
```

List the files under the working directory

```
[cloudera@quickstart movielens]$ ls
MovieRatingMapper.java MovieRatingReducer.java MovieRatings.java
```

Code

1. MovieRatingMapper.java

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;
```

```

public class MovieRatingMapper extends Mapper<LongWritable, Text, IntWritable,
DoubleWritable> {

    @Override

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

        // Skip header (if present)
        if (key.get() == 0 && value.toString().contains("userId")) {
            return;
        }

        String line = value.toString().trim();
        if (line.length() == 0) return;
        String[] ratingPieces = line.split(",");
        if (ratingPieces.length >= 3) {
            try {
                int movieId = Integer.parseInt(ratingPieces[1].trim());
                double rating = Double.parseDouble(ratingPieces[2].trim());
                context.write(new IntWritable(movieId), new DoubleWritable(rating));
            } catch (NumberFormatException e) {
                // skip malformed lines
            }
        }
    }
}

```

2. MovieRatingReducer.java

```

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.mapreduce.Reducer;

public class MovieRatingReducer extends Reducer<IntWritable, DoubleWritable,
IntWritable, DoubleWritable> {

    @Override

    public void reduce(IntWritable key, Iterable<DoubleWritable> values, Context context)

```

```

        throws IOException, InterruptedException {
    double sum = 0.0;
    int count = 0;
    for (DoubleWritable v : values) {
        sum += v.get();
        count++;
    }
    if (count > 0) {
        double avg = sum / count;
        context.write(key, new DoubleWritable(avg));
    }
}
}

```

3. **MovieRatings.java**

```

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.conf.Configuration;

public class MovieRatings {
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Syntax: MovieRatings <input path> <output path>");
            System.exit(-1);
        }
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Average movie rating");
        job.setJarByClass(MovieRatings.class);
    }
}

```

```

    job.setMapperClass(MovieRatingMapper.class);
    job.setReducerClass(MovieRatingReducer.class);

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

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

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

```

Step 3: Create CSV files

Command: **vi ratings.csv**

vi movies.txt

```

[cloudera@quickstart movielens]$ vi ratings.csv
[cloudera@quickstart movielens]$ vi movies.txt

```

Step 4: Upload files to HDFS

First create a directory called data

Command: **hdfs dfs -mkdir -p /data**

hdfs dfs -put -f ratings.csv /data/ratings.csv

hdfs dfs -put -f movies.txt /data/movies.txt

```

[cloudera@quickstart movielens]$ hdfs dfs -mkdir -p /data
[cloudera@quickstart movielens]$ hdfs dfs -put -f ratings.csv /data/ratings.csv
[cloudera@quickstart movielens]$ hdfs dfs -put -f movies.txt /data/movies.txt

```

To check if the file exists in the data directory

Command: **hdfs dfs -ls /data**

```

[cloudera@quickstart movielens]$ hdfs dfs -ls /data
Found 2 items
-rw-r--r-- 1 cloudera supergroup 1688 2025-11-05 09:41 /data/movies.txt
-rw-r--r-- 1 cloudera supergroup 810 2025-11-05 09:41 /data/ratings.csv

```

To check the contents of the file

Command: **hdfs dfs -cat /data/ratings.csv | head -n 5**

hdfs dfs -cat /data/movies.txt | head -n 5

```
[cloudera@quickstart movielens]$ hdfs dfs -cat /data/ratings.csv | head -n 5
userId,movieId,rating,timestamp
1,2,3.5,1112486027
1,29,3.5,1112484676
1,32,3.5,1112484819
1,47,3.5,1112484727
```

```
[cloudera@quickstart movielens]$ hdfs dfs -cat /data/movies.txt | head -n 5
movieId,title,genres
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Romance
4,Waiting to Exhale (1995),Comedy|Drama|Romance
[cloudera@quickstart movielens]$ pwd
```

Step 5: Compile the Java files

Command: **cd /home/cloudera/movielens**

rm -rf classes *.class MovieLensMapReduce.jar

mkdir -p classes

javac -classpath `hadoop classpath` -d classes MovieRatingMapper.java

MovieRatingReducer.java MovieRatings.java

ls -R classes # confirm classes created

```
[cloudera@quickstart movielens]$ rm -rf classes *.class MovieLensMapReduce.jar
[cloudera@quickstart movielens]$ mkdir -p classes
```

```
[cloudera@quickstart movielens]$ javac -classpath `hadoop classpath` -d classes
MovieRatingMapper.java MovieRatingReducer.java MovieRatings.java
[cloudera@quickstart movielens]$ ls -R classes
classes:
MovieRatingMapper.class MovieRatingReducer.class MovieRatings.class
```

Step 6: Create the jar

Command:

jar -cvf MovieLensMapReduce.jar -C classes # from/home/cloudera/movielens

jar tf MovieLensMapReduce.jar | head # inspect

```
[cloudera@quickstart movielens]$ jar -cvf MovieLensMapReduce.jar -C classes .
added manifest
adding: MovieRatingMapper.class(in = 2107) (out= 914)(deflated 56%)
adding: MovieRatings.class(in = 1566) (out= 863)(deflated 44%)
adding: MovieRatingReducer.class(in = 1710) (out= 706)(deflated 58%)
```

```
[cloudera@quickstart movielens]$ jar tf MovieLensMapReduce.jar | head
META-INF/
META-INF/MANIFEST.MF
MovieRatingMapper.class
MovieRatings.class
MovieRatingReducer.class
```

Step 7: Run the MapReduce job to compute average rating per movie

Command

```
export HADOOP_CLASSPATH=/home/cloudera/movielens/MovieLensMapReduce.jar
```

```
echo $HADOOP_CLASSPATH
```

```
hdfs dfs -rm -r -f /output # remove old HDFS output (if exists)
```

```
hadoop jar /home/cloudera/movielens/MovieLensMapReduce.jar MovieRatings
```

```
/data/ratings.csv /output # run the job
```

```
[cloudera@quickstart movielens]$ export HADOOP_CLASSPATH=/home/cloudera/movielens/MovieLensMapReduce.jar
[cloudera@quickstart movielens]$ echo $HADOOP_CLASSPATH
/home/cloudera/movielens/MovieLensMapReduce.jar
```

```
[cloudera@quickstart movielens]$ hdfs dfs -rm -r -f /output
[cloudera@quickstart movielens]$ hadoop jar /home/cloudera/movielens/MovieLensMapReduce.jar MovieRatings /data/ratings.csv /output
25/11/05 09:47:53 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
25/11/05 09:47:53 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
25/11/05 09:47:54 INFO input.FileInputFormat: Total input paths to process : 1
25/11/05 09:47:54 INFO mapreduce.JobSubmitter: number of splits:1
25/11/05 09:47:54 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1762362989980_0001
25/11/05 09:47:54 INFO impl.YarnClientImpl: Submitted application application_1762362989980_0001
25/11/05 09:47:54 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1762362989980_0001/
25/11/05 09:47:54 INFO mapreduce.Job: Running job: job_1762362989980_0001
25/11/05 09:48:04 INFO mapreduce.Job: Job job_1762362989980_0001 running in uber mode : false
```

Step 8: View Job Results

Command: **hdfs dfs -ls /output**

```
hdfs dfs -cat /output/part-r-00000 | head -n 50
```

```
[cloudera@quickstart movielens]$ hdfs dfs -ls /output
Found 2 items
-rw-r--r-- 1 cloudera supergroup 0 2025-11-05 09:48 /output/SUCCESS
-rw-r--r-- 1 cloudera supergroup 288 2025-11-05 09:48 /output/part-r-00000
```

```
---
[cloudera@quickstart movielens]$ hdfs dfs -cat /output/part-r-00000 | head -n 50
2      3.5
29     3.5
32     3.5
47     3.5
50     3.5
112    3.5
151    4.0
223    4.0
253    4.0
260    4.0
293    4.0
296    4.0
318    4.0
337    3.5
367    3.5
541    4.0
589    3.5
593    3.5
653    3.0
919    3.5
924    3.5
1009   3.5
1036   4.0
1079   4.0
1080   3.5
1089   3.5
1090   4.0
1097   4.0
1136   3.5
1193   3.5
1196   4.5
1198   4.5
1200   4.0
1201   3.0
1208   3.5
[cloudera@quickstart movielens]$
```

PROGRAM 7

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

```
test> use ProgBooksDB
```

```
ProgBooksDB> db.createCollection("ProgrammingBooks")
```

Insert a Single Document into **ProgrammingBooks**:

Use the `insertOne()` method to insert a new document into the **ProgrammingBooks** collection:

```
ProgBooksDB> db.ProgrammingBooks.insertOne({
  title: "The Pragmatic Programmer: Your Journey to Mastery",
  author: "David Thomas, Andrew Hunt",
  category: "Software Development",
  year: 1999
})
```

Insert multiple Documents into the **ProgrammingBooks** Collection :

Now, insert 5 documents representing programming books into the **ProgrammingBooks** collection using the `insertMany()` method:

```
ProgBooksDB> db.ProgrammingBooks.insertMany([
  {
    title: "Clean Code: A Handbook of Agile Software Craftsmanship",
    author: "Robert C. Martin",
    category: "Software Development",
    year: 2008
  },
  {
    title: "JavaScript: The Good Parts",
    author: "Douglas Crockford",
    category: "JavaScript",
    year: 2008
  },
  {
    title: "Eloquent JavaScript",
    author: "Mosh Bar",
    category: "JavaScript",
    year: 2010
  },
  {
    title: "The JavaScript Book",
    author: "David Flanagan",
    category: "JavaScript",
    year: 2014
  },
  {
    title: "Functional JavaScript",
    author: "Ben Lesh",
    category: "JavaScript",
    year: 2014
  }
])
```

```
[
  {
    title: "Design Patterns: Elements of Reusable Object-Oriented
Software",
    author: "Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides",
    category: "Software Design",
    year: 1994
  },
  {
    title: "Introduction to Algorithms",
    author: "Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest,
Clifford Stein",
    category: "Algorithms",
    year: 1990
  },
  {
    title: "Python Crash Course: A Hands-On, Project-Based Introduction to
Programming",
    author: "Eric Matthes",
    category: "Python",
    year: 2015
  }
])
```

Query operations

Find All Documents

To retrieve all documents from the **ProgrammingBooks** collection:

```
ProgBooksDB> db.ProgrammingBooks.find().pretty()
[
  {
    _id: ObjectId('664ee3b1924a8039f62202d8'),
    title: 'The Pragmatic Programmer: Your Journey to Mastery',
    author: 'David Thomas, Andrew Hunt',
    category: 'Software Development',
    year: 1999
  },
  {
```

```

    _id: ObjectId('664ee452924a8039f62202d9'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: ObjectId('664ee452924a8039f62202da'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  },
  {
    _id: ObjectId('664ee452924a8039f62202db'),
    title: 'Design Patterns: Elements of Reusable Object-Oriented
Software',
    author: 'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides',
    category: 'Software Design',
    year: 1994
  },
  {
    _id: ObjectId('664ee452924a8039f62202dc'),
    title: 'Introduction to Algorithms',
    author: 'Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest,
Clifford Stein',
    category: 'Algorithms',
    year: 1990
  },
  {
    _id: ObjectId('664ee452924a8039f62202dd'),
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to
Programming',
    author: 'Eric Matthes',
    category: 'Python',
    year: 2015
  }
]

```

Find Documents Matching a Condition

To find books published after the year 2000:

```
ProgBooksDB> db.ProgrammingBooks.find({ year: { $gt: 2000 } }).pretty()
[
  {
    id: ObjectId('664ee452924a8039f62202d9'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  },
  {
    id: ObjectId('664ee452924a8039f62202da'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  },
  {
    id: ObjectId('664ee452924a8039f62202dd'),
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to Programming',
    author: 'Eric Matthes',
    category: 'Python',
    year: 2015
  }
]
```

Update Operations

a. Update a Single Document

To update a specific book (e.g., change the author of a book):

```
ProgBooksDB> db.ProgrammingBooks.updateOne(
  { title: "Clean Code: A Handbook of Agile Software Craftsmanship" },
  { $set: { author: "Robert C. Martin (Uncle Bob)" } }
)
```

```
//verify by displaying books published in year 2008
ProgBooksDB> db.ProgrammingBooks.find({ year: { $eq: 2008 } }).pretty()
[
  {
    _id: ObjectId('663eaaebae582498972202df'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin (Uncle Bob)',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: ObjectId('663eaaebae582498972202e0'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  }
]

//another way to verify
ProgBooksDB> db.ProgrammingBooks.find({ author: { $regex: "Robert*" }
}).pretty()
[
  {
    _id: ObjectId('664ee452924a8039f62202d9'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin (Uncle Bob)',
    category: 'Software Development',
    year: 2008
  }
]
```

b. Update Multiple Documents

To update multiple books (e.g., update the category of books published before 2010):

```
ProgBooksDB> db.ProgrammingBooks.updateMany(
  { year: { $lt: 2010 } },
  { $set: { category: "Classic Programming Books" } }
)
```

```
//verify update operation by displaying books published before year 2010
ProgBooksDB> db.ProgrammingBooks.find({ year: { $lt: 2010 } }).pretty()
[
  {
    _id: ObjectId('663eaaebae582498972202df'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin (Uncle Bob)',
    category: 'Classic Programming Books',
    year: 2008
  },
  {
    _id: ObjectId('663eaaebae582498972202e0'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'Classic Programming Books',
    year: 2008
  },
  {
    _id: ObjectId('663eaaebae582498972202e1'),
    title: 'Design Patterns: Elements of Reusable Object-Oriented
Software',
    author: 'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides',
    category: 'Classic Programming Books',
    year: 1994
  },
  {
    _id: ObjectId('663eaaebae582498972202e2'),
    title: 'Introduction to Algorithms',
    author: 'Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest,
Clifford Stein',
    category: 'Classic Programming Books',
    year: 1990
  },
  {
    _id: ObjectId('663eab05ae582498972202e4'),
    title: 'The Pragmatic Programmer: Your Journey to Mastery',
    author: 'David Thomas, Andrew Hunt',
    category: 'Classic Programming Books',
    year: 1999
  }
]
```

```
}  
]  
]
```

Delete Operations

Delete a Single Document To delete a specific book from the collection (e.g., delete a book by title):

```
ProgBooksDB> db.ProgrammingBooks.deleteOne({ title: "JavaScript: The Good  
Parts" })  
  
{ acknowledged: true, deletedCount: 1 }  
  
//Verify to see document is deleted  
ProgBooksDB> db.ProgrammingBooks.find({ title: "JavaScript: The Good Parts"  
}).pretty()
```

Delete Multiple Documents

To delete multiple books based on a condition (e.g., delete all books published before 1995):

```
ProgBooksDB> db.ProgrammingBooks.deleteMany({ year: { $lt: 1995 } })  
  
{ acknowledged: true, deletedCount: 2 }
```

You can check whether the specified documents were deleted by displaying the contents of the collection.

Delete All Documents in the Collection: To delete all documents in a collection (e.g., **ProgrammingBooks**), use the **deleteMany()** method with an empty filter **{}**:

```
//delete all documents in a collection  
ProgBooksDB> db.ProgrammingBooks.deleteMany({})  
  
{ acknowledged: true, deletedCount: 3 }  
  
//verify by displaying the collection  
ProgBooksDB> db.ProgrammingBooks.find().pretty()
```

Projection Operations

In MongoDB, a projection refers to the mechanism of specifying which fields (or columns) should be returned from a query result. When querying a collection, you can use projection to control the shape of the returned documents by specifying which fields to include or exclude.

In MongoDB, projection is typically specified as the second parameter to the `find()` method. The projection parameter takes an object where keys represent the fields to include or exclude, with values of `1` (include) or `0` (exclude).

Include Specific Fields: Use `1` to include a field in the result:

```
ProgBooksDB> db.ProgrammingBooks.find({}, { title: 1, author: 1 } )
[
  {
    id: ObjectId('665163289edbf91e12202dd'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin'
  },
  {
    id: ObjectId('665163289edbf91e12202e1'),
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to
Programming',
    author: 'Eric Matthes'
  }
]
```

Exclude Specific Fields: Use `0` to exclude a field from the result:

```
ProgBooksDB> db.ProgrammingBooks.find({}, {year: 0})
[
  {
    id: ObjectId('665163289edbf91e12202dd'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin',
  }
]
```

```

    category: 'Software Development'
  },
  {
    _id: ObjectId('665163289edbd91e12202e1'),
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to
Programming',
    author: 'Eric Matthes',
    category: 'Python'
  }
]

```

Aggregation Pipeline and its operations

Execute Aggregation Pipeline and its operations (pipeline must contain group, project, \$skip etc.)

Let's consider a scenario involving a **restaurantDB** database with a **restaurants** collection. Each document in the **restaurants** collection contains details about a restaurant, including its name, cuisine, location, and an array of reviews. Each review includes a rating and a comment. After creating the **restaurantDB** database and insert sample documents into the **restaurants** collection we will create an aggregation pipeline as shown below.

```

// Switch to the restaurantDB database
use restaurantDB

// Insert sample documents into the restaurants collection
db.restaurants.insertMany([
  {
    name: "Biryani House",
    cuisine: "Indian",
    location: "Jayanagar",
    reviews: [
      { user: "Aarav", rating: 5, comment: "Amazing biryani!" },
      { user: "Bhavana", rating: 4, comment: "Great place!" }
    ]
  }
]

```

```

    },
    {
      name: "Burger Joint",
      cuisine: "American",
      location: "Koramangala",
      reviews: [
        { user: "Chirag", rating: 3, comment: "Average burger" },
        { user: "Devika", rating: 4, comment: "Good value" }
      ]
    },
    {
      name: "Pasta House",
      cuisine: "Italian",
      location: "Rajajinagar",
      reviews: [
        { user: "Esha", rating: 5, comment: "Delicious pasta!" },
        { user: "Farhan", rating: 4, comment: "Nice ambiance" }
      ]
    },
    {
      name: "Curry Palace",
      cuisine: "Indian",
      location: "Jayanagar",
      reviews: [
        { user: "Gaurav", rating: 4, comment: "Spicy and tasty!" },
        { user: "Harini", rating: 5, comment: "Best curry in town!" }
      ]
    },
    {
      name: "Taco Stand",
      cuisine: "Mexican",
      location: "Jayanagar",
      reviews: [
        { user: "Ishaan", rating: 5, comment: "Fantastic tacos!" },
        { user: "Jaya", rating: 4, comment: "Very authentic" }
      ]
    }
  ])

// Run the aggregation pipeline query to display reviews summary

```

```

db.restaurants.aggregate([
  {
    $match: {
      location: "Jayanagar"
    }
  },
  {
    $unwind: "$reviews"
  },
  {
    $group: {
      _id: "$name",
      averageRating: { $avg: "$reviews.rating" },
      totalReviews: { $sum: 1 }
    }
  },
  {
    $sort: {
      averageRating: -1
    }
  },
  {
    $project: {
      _id: 0,
      restaurant: "$_id",
      averageRating: 1,
      totalReviews: 1
    }
  },
  {
    $skip: 1
  }
]).pretty()

```

PROGRAM 8

Implement a word count program in Hadoop and Spark.

Step 1: Create input file and show contents

Create HDFS input directory and put the s1.txt into that directory

```
cloudera@quickstart ~]$ echo "hello spark hello world hello lsp" > s1.txt
cloudera@quickstart ~]$ hdfs dfs -mkdir /user/spark/input
cloudera@quickstart ~]$ hdfs dfs -put s1.txt /user/spark/input
25/10/28 23:08:42 WARN hdfs.DFSClient: Caught exception
java.lang.InterruptedException
  at java.lang.Object.wait(Native Method)
  at java.lang.Thread.join(Thread.java:1281)
  at java.lang.Thread.join(Thread.java:1355)
  at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DFSOutputStream.java:967)
  at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeInternal(DFSOutputStream.java:935)
  at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStream.java:931)
```

Step 2: Verify file and its contents on HDFS using the following and Start Spark

Command: **hdfs dfs -cat /user/spark/input/s1.txt**

```
at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStr
[cloudera@quickstart ~]$ hdfs dfs -cat /user/spark/input/s1.txt
hello spark hello world hello lsp
[cloudera@quickstart ~]$ spark
```

Step 3: Open and edit the Scala script

Command: **vi wordcount.scala**

```
scala> [cloudera@quickstart ~]$ vi wordcount.scala
```

Step 4: Code block

Command: **val textFile = sc.textFile("hdfs:///user/spark/input/s1.txt")**

val counts = textFile.flatMap(_.split(" ")).reduceByKey(_ + _)

counts.saveAsTextFile("hdfs:///user/spark/output_wc")

```
val textFile = sc.textFile("hdfs:///user/spark/input/s1.txt")
val counts = textFile.flatMap(_.split(" ")).map((_,1)).reduceByKey(_ + _)
counts.saveAsTextFile("hdfs:///user/spark/output_wc")
```

Step 5: Run the script in spark-shell

Command: **spark-shell -i wordcount.scala**

```
[cloudera@quickstart ~]$ spark-shell -i wordcount.scala
```

```
scala> [cloudera@quickstart ~]$ hdfs dfs -cat /user/spark/output_wc/part-*  
(spark,1)  
(lsp,1)  
(hello,3)  
(world,1)
```