MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DSE-3264 - Big Data Analytics
Laboratory Manual

Department	**:**	Data Science Engineering And Computer Applications
Course Name & code	**:**	DSE-3264 & Big Data Analytics Laboratory
Semester & branch	**:**	VI Sem & BTech Data Science & Engineering
Name of the faculty	**:**	Dr. Saraswati Koppad, Dr. Shavantrevva Sangappa Bilakeri

No of contact hours/week:	**L**	**T**	**P**	**C**
	0	0	3	1

CONTENTS

Course Objectives ·

1. Gain practical experience using big data tools and platforms like Hadoop, Spark, Hive, Pig, and HBase to store, process, and analyze large datasets.
2. Implement Hadoop MapReduce for processing Big Data
3. Apply data analytics techniques to solve problems and extract meaningful insights using big data frameworks.

Course Outcomes:

At the end of this course, students will have the ability to

1. Demonstrate the ability to use big data frameworks such as Hadoop, Spark
2. Apply MapReduce techniques to process large data sets.
3. Demonstrate the ability to use big data tools such as Pig, Hive, and HBase to store and process Big Data
4. Apply analytical methods and techniques to solve data-driven problems.

Evaluation plan : (Tentative)

- Internal Assessment Marks: 60%
- End semester assessment of 2-hour duration: 40 %

Evaluation pattern

Internal Marks – 60 + End Sem - 40	
Internal Marks	Internal Assessment – 40 + Mid-sem - 20
Internal Assessment	Lab observations – 2*11(22) + Viva (18)
Lab Observation	Record (1*11) + Execution (1*11)
Viva	Quiz and mini-project (18)

INSTRUCTIONS TO THE STUDENTS

Pre-Lab Session Instructions

- Be on time, adhere to the institution's rules, and maintain decorum.
- Leave your mobile phones, pen drives, and other electronic devices in your bag and keep the bag in the designated place in the lab.
- Must Sign in to the log register provided.
- Make sure to occupy the allotted system and answer the attendance.

In-Lab Session Instructions

- Follow the instructions on the allotted exercises.
- Show the program and results to the instructors on completion of experiments.
- Copy the program and results for the lab record.
- Prescribed textbooks and class notes can be kept ready for reference if required.

General Instructions for the Exercises in Lab

- Academic honesty is required in all your work. You must solve all programming assignments independently, except where group work is authorized. This means you must not take, show, give, or otherwise allow others to take your program code, problem solutions, or other work.
- The programs should meet the following criteria:
    - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
    - Programs should perform input validation (Data type, range error, etc.), give appropriate error messages, and suggest corrective actions.
    - Comments should be used to give the statement of the problem, and every function should indicate the purpose of the function, inputs, and outputs.
    - Statements within the program should be properly indented.
    - Use meaningful names for variables and functions.
    - Make use of constants and type definitions wherever needed.
    - The exercises for each week are divided into three sets:
        - Solved solutions
        - Lab exercises - to be completed during lab hours
        - Additional Exercises - to be completed outside the lab or in the lab to enhance the skill

Questions for lab tests and examinations are not necessarily limited to the questions in the manual but may involve some variations and/or combinations of the questions.

THE STUDENTS SHOULD NOT

- Possess mobile phones or any other electronic gadgets during lab hours.
- Go out of the lab without permission.
- **Change/update any configuration in your allotted system. If so, the student will lose internal assessment marks**

Week 1 - Understanding Hadoop and HDFS Basic Commands

Introduction to the Hadoop Ecosystem and HDFS

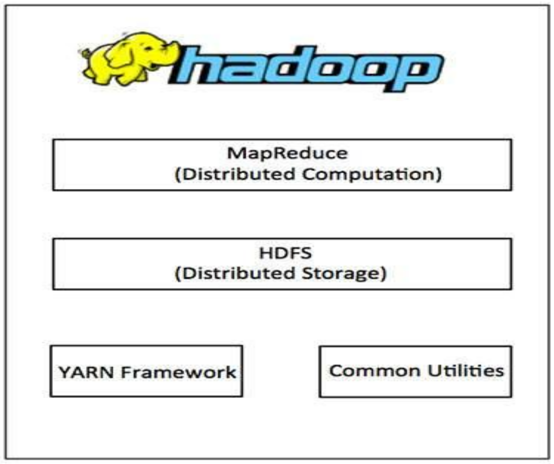
The Hadoop Ecosystem is a collection of tools and frameworks that work together to manage and process big data. HDFS (Hadoop Distributed File System) is the backbone of the Hadoop ecosystem, providing distributed storage and fault tolerance.

Hadoop Ecosystem Overview

The Hadoop Ecosystem is built around the core Hadoop components and includes tools for data storage, processing, querying, and analytics. Here's a breakdown:

Core Components:

1. **HDFS (Hadoop Distributed File System):**
    - Provides distributed storage.
    - Handles large files across multiple machines.
2. **YARN (Yet Another Resource Negotiator):**
    - Manages resources and job scheduling in the cluster.
3. **MapReduce:**
    - Programming model for data processing.



Supporting Tools:

1. **Apache Hive:**
    - Data warehouse tool for querying data in HDFS using SQL-like language (HiveQL).
2. **Apache HBase:**
    - NoSQL database for real-time read/write access to large datasets.
3. **Apache Spark:**
    - Fast, in-memory data processing engine.
4. **Apache Pig:**
    - High-level platform for creating MapReduce programs using a scripting language (Pig Latin).
5. **Apache Sqoop:**
    - Tool for transferring data between Hadoop and relational databases.
6. **Apache Flume:**
    - Tool for collecting and transferring large amounts of log data into HDFS.
7. **ZooKeeper:**
    - Centralized service for managing distributed systems.
8. **Oozie:**
    - Workflow scheduler for Hadoop jobs.

HDFS Basics

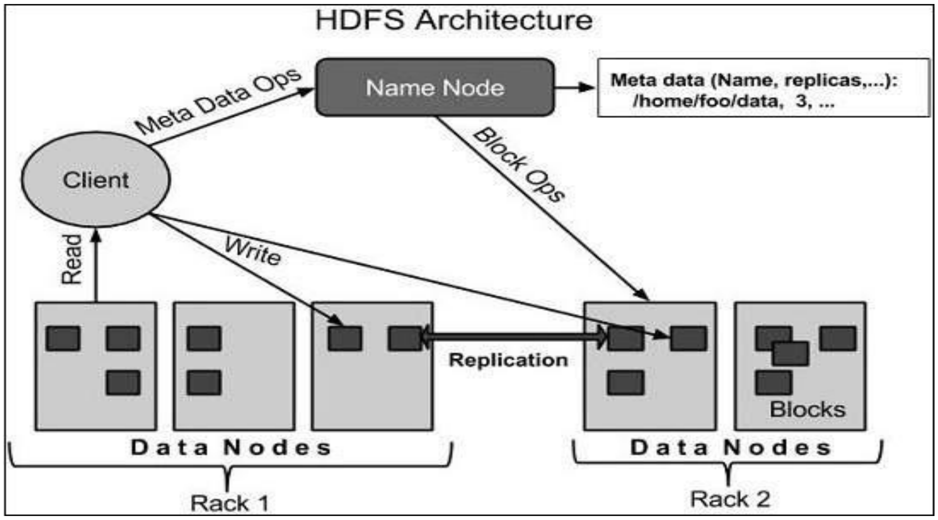
HDFS is the storage layer of Hadoop, designed for scalability and fault tolerance. It handles massive data volumes by breaking files into smaller chunks and distributing them across a cluster.

HDFS Features:

- Replication: Ensures fault tolerance by replicating data blocks (default: 3 copies).
- Write-Once, Read-Many: Optimized for batch processing.
- Scalability: Handles petabytes of data across thousands of nodes.

HDFS Architecture:

1. NameNode:
    - Maintains metadata (e.g., file structure, permissions).
    - Coordinates DataNodes but does not store data.
2. DataNodes:
    - Store actual data blocks.
    - Perform read/write operations as instructed by NameNode.
3. Secondary NameNode:
    - Periodically saves snapshots of NameNode metadata (not a backup).



Working with Hadoop:

- Open a new terminal and start the Hadoop service by following commands

```
start-dfs.sh
start-yarn.sh
```

This command initializes all the required Hadoop daemons for the cluster to become operational.

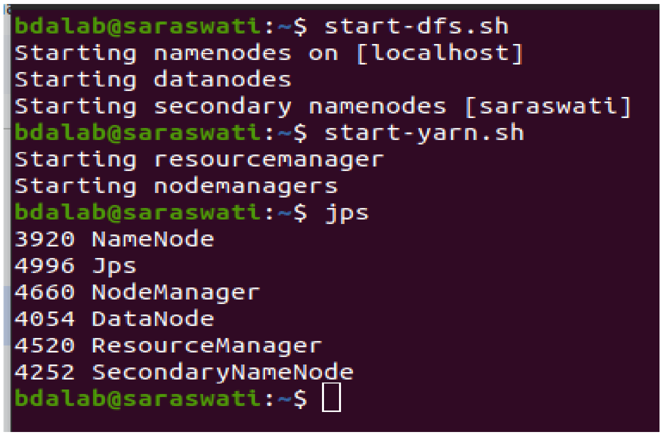
When you want to stop the services use the command:

```
stop-dfs.sh
stop-yarn.sh
```

- Check the Status of Hadoop Services (To confirm that all necessary Hadoop services are up and running, you can check their status using the ‘jps’ command)

```
jps
```

The ‘jps’ command displays the Java Virtual Machine (JVM) processes and should show a list of Hadoop services running, such as the NameNode, DataNode, ResourceManager, and NodeManager.



```
bdalab@saraswati:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [saraswati]
bdalab@saraswati:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
bdalab@saraswati:~$ jps
3920 NameNode
4996 Jps
4660 NodeManager
4054 DataNode
4520 ResourceManager
4252 SecondaryNameNode
bdalab@saraswati:~$
```

- **Accessing Hadoop Namenode and Resource Manager:**

  To access the Hadoop Namenode, open a web browser and enter the following URL:

  http://localhost:9870

  Similarly, to access the Hadoop Resource Manager, open a web browser and enter the following URL:

  http://localhost:8088

- **Verifying the Hadoop Cluster:**

  You can check the Hadoop version by following command

  ```
  hadoop version
  ```

  Create Directories in HDFS:

  ```
  hdfs dfs -mkdir /user/dir_name
  ```

  List Directories in HDFS:

  ```
  hdfs dfs -ls /
  hdfs dfs -ls /user/
  ```

  Transfer Files to the Hadoop File System:

  ```
  hdfs dfs -put source-filename /destination_folder/
  ```

Week-1 Exercise:

1. Practice Linux commands (No need to write in the record book)
    a. What command would you use to display the current directory?
    b. How do you list all files, including hidden ones, in a directory?
    c. Which command is used to create a new directory?
    d. How can you check the currently logged-in user?
    e. What does the pwd command do?
    f. How do you copy a file from one location to another?
    g. What is the command to move a file?
    h. How can you delete a directory and all its contents?
    i. How do you rename a file in Linux?
    j. What is the difference between `rm` and `rmdir`?
    k. What command is used to change the permissions of a file?
    l. How can you view the permissions of a file?
    m. Which command changes the owner of a file?
    n. How do you add execute permission for the owner of a file?
    o. What is the command to display all currently running processes?
    p. How can you check the IP address of your system?
    q. How do you display the contents of a file?
    r. How can you display the current date and time in Linux?

2. Explore Basic HDFS Commands
    a. Remove directory
    b. View/Read File Contents
    c. Download/copy a file from HDFS to the local system
    d. Copy/move files within HDFS
    e. Remove file from HDFS
    f. View file permission

Week 2: Advanced HDFS Commands and MapReduce

Week 2 Exercises (Advanced HDFS Commands):

Create a directory in Hadoop (/user/your_reg_no/lab2). Use the same directory to keep all your files.

hdfs dfs -mkdir /user/your_reg_no/lab2

(a).

1. Upload a new file (Ex. File1.txt) into Hadoop. Demonstrate the hdfs commands for the following:
   a. Check the permission of the file
   b. Change the permission of the file.
   c. Check ownership of the file
2. Upload a large file (Ex: largefile.txt) into Hadoop. Demonstrate the hdfs commands for following.
   a. Check block details of the file
   b. View the current replication factor for file
   c. Modify the replication factor of the file
3. Demonstrate hdfs commands for the following tasks
   a. Check directory size for the directory you have created (Disk usage analysis)
   b. Find the total capacity and usage of HDFS
4. Upload a text file with sample data to the Hadoop (Ex: analytics.txt). Using HDFS commands, find
   a. Number of lines in the file
   b. Number of occurrences of a specific word (Ex. Hadoop) in the file.

(b). Executing simple MapReduce jobs:

What is Hadoop MapReduce?

Hadoop MapReduce is a programming model and processing framework for processing large datasets in a distributed environment. It divides the job into smaller tasks, processes them in parallel, and consolidates the results.

Key Components

1. **Map Phase:**
   - Processes input data and generates intermediate key-value pairs.
   - Example: Counting words in a document; the mapper emits each word as a key and the count 1 as a value.
2. **Shuffle and Sort Phase:**
   - Intermediate key-value pairs are grouped and sorted by key.
   - Ensures all values for a specific key are brought together.
3. **Reduce Phase:**
   - Processes grouped data from the shuffle phase to generate final output.
   - Example: Aggregating word counts.

MapReduce Workflow

1. **Input Data:**
   - Stored in HDFS and divided into splits.
2. **Mapper:**
   - Processes each split and outputs key-value pairs.
3. **Shuffle and Sort:**
   - Intermediate outputs are shuffled and sorted by the framework.
4. **Reducer:**
   - Processes sorted key-value pairs and generates the final result.
5. **Output:**
   - Written back to HDFS.

(b). Executing sample MapReduce jobs:

What is Hadoop MapReduce?

Hadoop MapReduce is a programming model and processing framework for processing large datasets in a distributed environment. It divides the job into smaller tasks, processes them in parallel, and consolidates the results.

Key Components

4. **Map Phase:**
   - Processes input data and generates intermediate key-value pairs.

- Example: Counting words in a document; the mapper emits each word as a key and the count 1 as a value.

5. **Shuffle and Sort Phase:**
   - Intermediate key-value pairs are grouped and sorted by key.
   - Ensures all values for a specific key are brought together.
6. **Reduce Phase:**
   - Processes grouped data from the shuffle phase to generate final output.
   - Example: Aggregating word counts.

MapReduce Workflow

6. **Input Data:**
   - Stored in HDFS and divided into splits.
7. **Mapper:**
   - Processes each split and outputs key-value pairs.
8. **Shuffle and Sort:**
   - Intermediate outputs are shuffled and sorted by the framework.
9. **Reducer:**
   - Processes sorted key-value pairs and generates the final result.
10. **Output:**
    - Written back to HDFS.

Part 1: Run bellow commands to install python (Locally)

```
wget https://www.python.org/ftp/python/3.6.5/Python-3.6.5.tgz
tar -xvf Python-3.6.5.tgz
```

Step-1. Write a Mapper

Mapper.py: Initially the partition of content takes place based on line.split() function, number of partitions made = number of mapper class gets created. Mapper overrides the ―map‖ function which provides <key, value> pairs as the input. Even the key is repeated in same or different mapper class it doesnot matter as the default value for every key is assigned to be as "1". A Mapper implementation may output<key,value> pairs using the provided Context .

Input value of the WordCount Map task will be a line of text from the input data file and the key would be the line number <line_number, line_of_text> . Map task outputs <word, one> for each word in the line of text.

Pseudo-code : mapper.py

```
#!/usr/bin/python3
"mapper.py"
import sys
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        print ('%s\t%s' % (word, 1))
```

Step-2. Write a Reducer

A Reducer collects the intermediate <key,value> output from multiple map tasks and assemble a single result. Here, the WordCount program will sum up the occurrence of each word to pairs as <word, occurrence>.

Pseudo-code: reducer.py

```
#!/usr/bin/python3
"reducer.py"
import sys
current_word = None
current_count = 0
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t')
    count = int(count)
    if current_word == word:
        current_count += count
    else:
        if current_word:
            print ('%s\t%s' % (current_word, current_count))
        current_word = word
        current_count = 1
```

TO run word count program locally use following commands

Command: `cat input.txt  |pyhton3 mapper.py`

Output:

```
hi      1
how     1
are     1
you     1
i       1
am      1
good    1
hope    1
you     1
doing   1
good    1
too     1
how     1
about   1
you.    1
i       1
am      1
in      1
manipal         1
studying        1
Btech   1
in      1
Data    1
science.        1
```

Command: `cat input.txt |python3 mapper.py|sort|python3 reducer.py`

Output:

```
about   1
am      2
are     1
```

```
Btech   1
Data    1
doing   1
good    2
hi      1
hope    1
how     2
i       2
in      2
manipal         1
science.        1
studying        1
too     1
you     2
you.    1
```

Part 2. TO run word count program on Hadoop framework use following command:

(i)Prepare input data (inputwc.txt)

Load input.txt into Hadoop:

```
hdfs dfs -put inputwc.txt /user/studentregno/lab2
```

(ii) Run the Example MapReduce Job: (Available with Hadoop Installation)

```
hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.4.1.jar
wordcount /user/studentregno/lab2/inputwc.txt/user/studentregno/lab2/output
```

(iii) View the Results

```
hdfs dfs -cat /user/studentregno/lab2/output/part-r-00000
```

Week 2: Exercise: MapReduce with Python

1. Consider the text file (consider larger file size) of your choice and perform word count using MapReduce technique.
2. Perform Matrix operations using MapReduce by considering 3 * 3 matrix and perform following operations:
   i. Matrix addition and subtraction
   ii. Matrix Multiplication
   iii. Matrix transpose

Note: Consider 3*3 matrix content as shown below

a,0,0,10

a,0,1,20

a,0,2,30

a,1,0,40

a,1,1,50

a,1,2,60

a,2,0,70

a,2,1,80

a,2,2,90

b,0,0,1

b,0,1,2

b,0,2,3

b,1,0,4

b,1,1,5

b,1,2,6

b,2,0,7

b,2,1,8

b,2,2,9

3. Create a text file containing the 20 student details such as registration number, name and marks (ex: 1001, john,45) .Write a MapReduce program to sort data by student name

Week 3: MapReduce Programs

Example Program: (calculate word counts in a text file)

(i) Create a mapper script

The Mapper processes the input line by line, splitting it into words and emitting each word with a count of 1.

```
#!/usr/bin/env python3
# mapper.py
import sys

# Read lines from standard input
for line in sys.stdin:
    # Split the line into words
    words = line.strip().split()
    for word in words:
        # Emit each word with a count of 1
```

(ii) Create a Reducer script

The Reducer receives sorted input from the Mapper, groups values by key, and computes the sum of counts for each word.

```
#!/usr/bin/env python3
# reducer.py
import sys

current_word = None
current_count = 0
word = None

# Read lines from standard input
for line in sys.stdin:
    # Parse the input we got from mapper
    word, count = line.strip().split("\t")
    count = int(count)

    # If this is the first word or a new word
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # Emit the word and its count
            print(f"{current_word}\t{current_count}")
        current_word = word
        current_count = count

# Emit the last word
if current_word == word:
    print(f"{current_word}\t{current_count}")
```

(iii) Prepare input data (inputwc.txt)

```
Manipal Institute of Technology Manipal
Manipal Academy of Higher Education
Department of Data Science and Computer Application
Manipal, Karnataka, India
```

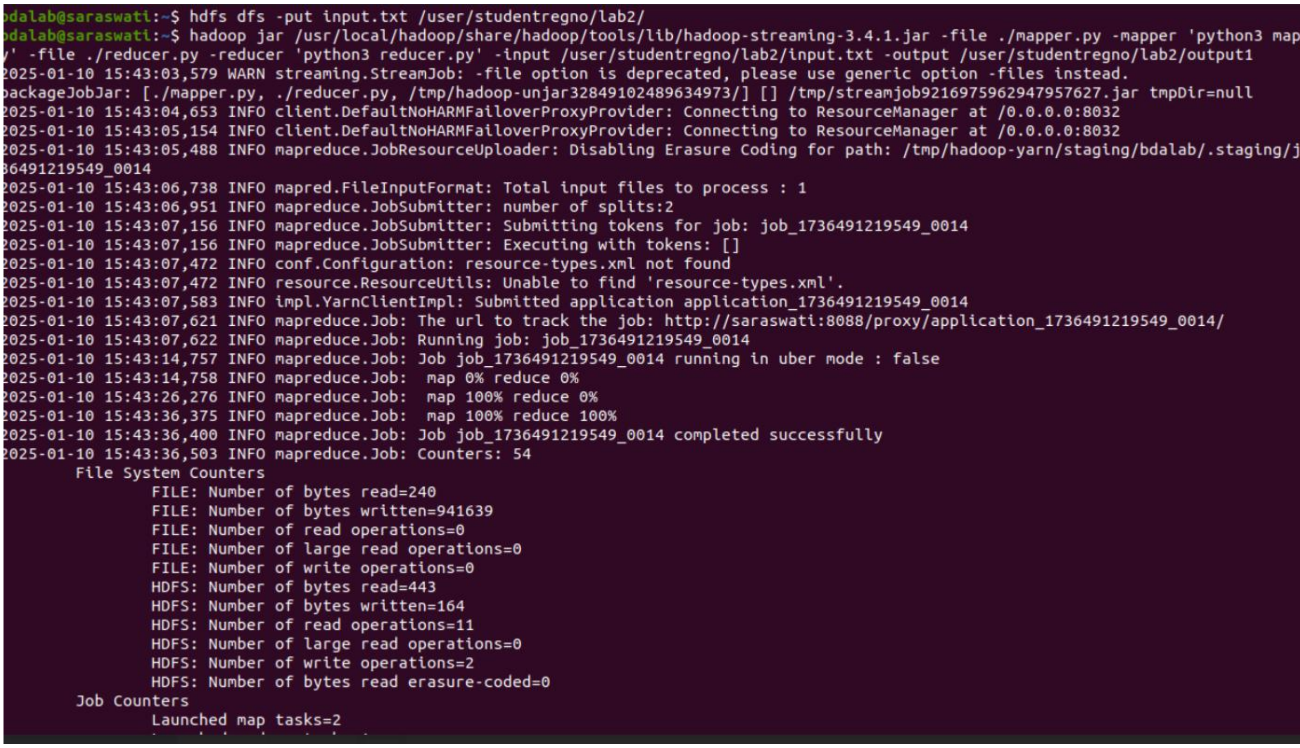
Load input.txt into Hadoop:

```
hdfs dfs -put inputwc.txt /user/studentregno/lab2
```

(iv) Run the MapReduce Job:

Execute the job using Hadoop Streaming:

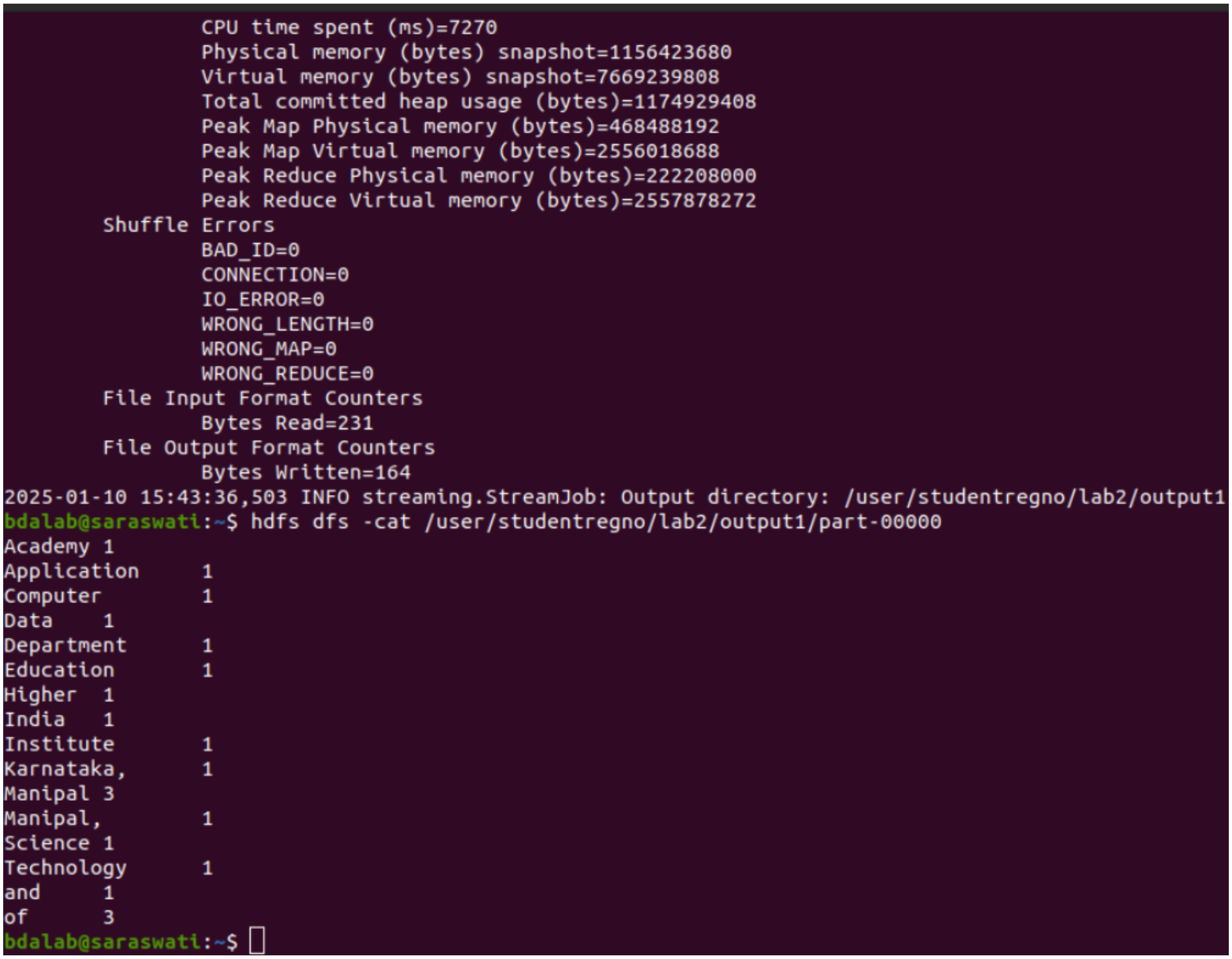
```
hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.4.1.jar -file
./mapper.py -mapper 'python3 mapper.py' -file ./reducer.py -reducer 'python3
reducer.py' -input /user/studentregno/lab2/input.txt -output
/user/studentregno/lab2/output1
```



```
odalab@saraswati:~$ hdfs dfs -put input.txt /user/studentregno/lab2/
odalab@saraswati:~$ hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.4.1.jar -file ./mapper.py -mapper 'python3 map
y' -file ./reducer.py -reducer 'python3 reducer.py' -input /user/studentregno/lab2/input.txt -output /user/studentregno/lab2/output1
2025-01-10 15:43:03,579 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [./mapper.py, ./reducer.py, /tmp/hadoop-unjar32849102489634973/] [] /tmp/streamjob9216975962947957627.jar tmpDir=null
2025-01-10 15:43:04,653 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2025-01-10 15:43:05,154 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2025-01-10 15:43:05,488 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/bdalab/.staging/j
36491219549_0014
2025-01-10 15:43:06,738 INFO mapred.FileInputFormat: Total input files to process : 1
2025-01-10 15:43:06,951 INFO mapreduce.JobSubmitter: number of splits:2
2025-01-10 15:43:07,156 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1736491219549_0014
2025-01-10 15:43:07,156 INFO mapreduce.JobSubmitter: Executing with tokens: []
2025-01-10 15:43:07,472 INFO conf.Configuration: resource-types.xml not found
2025-01-10 15:43:07,472 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2025-01-10 15:43:07,583 INFO impl.YarnClientImpl: Submitted application application_1736491219549_0014
2025-01-10 15:43:07,621 INFO mapreduce.Job: The url to track the job: http://saraswati:8088/proxy/application_1736491219549_0014/
2025-01-10 15:43:07,622 INFO mapreduce.Job: Running job: job_1736491219549_0014
2025-01-10 15:43:14,757 INFO mapreduce.Job: Job job_1736491219549_0014 running in uber mode : false
2025-01-10 15:43:14,758 INFO mapreduce.Job:  map 0% reduce 0%
2025-01-10 15:43:26,276 INFO mapreduce.Job:  map 100% reduce 0%
2025-01-10 15:43:36,375 INFO mapreduce.Job:  map 100% reduce 100%
2025-01-10 15:43:36,400 INFO mapreduce.Job: Job job_1736491219549_0014 completed successfully
2025-01-10 15:43:36,503 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=240
                FILE: Number of bytes written=941639
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=443
                HDFS: Number of bytes written=164
                HDFS: Number of read operations=11
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=2
```

(v) View the Results

hdfs dfs -cat /user/studentregno/lab2/output1/part-00000



```
                CPU time spent (ms)=7270
                Physical memory (bytes) snapshot=1156423680
                Virtual memory (bytes) snapshot=7669239808
                Total committed heap usage (bytes)=1174929408
                Peak Map Physical memory (bytes)=468488192
                Peak Map Virtual memory (bytes)=2556018688
                Peak Reduce Physical memory (bytes)=222208000
                Peak Reduce Virtual memory (bytes)=2557878272
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=231
        File Output Format Counters
                Bytes Written=164
2025-01-10 15:43:36,503 INFO streaming.StreamJob: Output directory: /user/studentregno/lab2/output1
bdalab@saraswati:~$ hdfs dfs -cat /user/studentregno/lab2/output1/part-00000
Academy 1
Application     1
Computer        1
Data    1
Department      1
Education       1
Higher  1
India   1
Institute       1
Karnataka,      1
Manipal 3
Manipal,        1
Science 1
Technology      1
and     1
of      3
bdalab@saraswati:~$
```

Week-3 Exercise:

1. Write a Hadoop MapReduce program to count the frequency of each character in a text file.
2. Write a Hadoop MapReduce program to find the maximum temperature for each year in a weather dataset.
3. Write a Hadoop MapReduce program to find the top N records (e.g., top 3 highest scores) from a dataset.
4. Write a Hadoop MapReduce program to calculate the average value for each key in a dataset

Week-4 Advanced Programs on MapReduce

Solved Program: Implement a MapReduce program to perform secondary sorting, where records are sorted by a primary and secondary key.

Write a MapReduce program to Sort a dataset of employee records by department (primary key) and salary (secondary key) in ascending order.

Sample Input:

```
1,HR,Alice,7000
2,IT,Bob,9000
3,HR,Charlie,8000
4,IT,Dave,9500
```

Expected Output:

```
HR,Alice,7000
HR,Charlie,8000
IT,Bob,9000
IT,Dave,9500
```

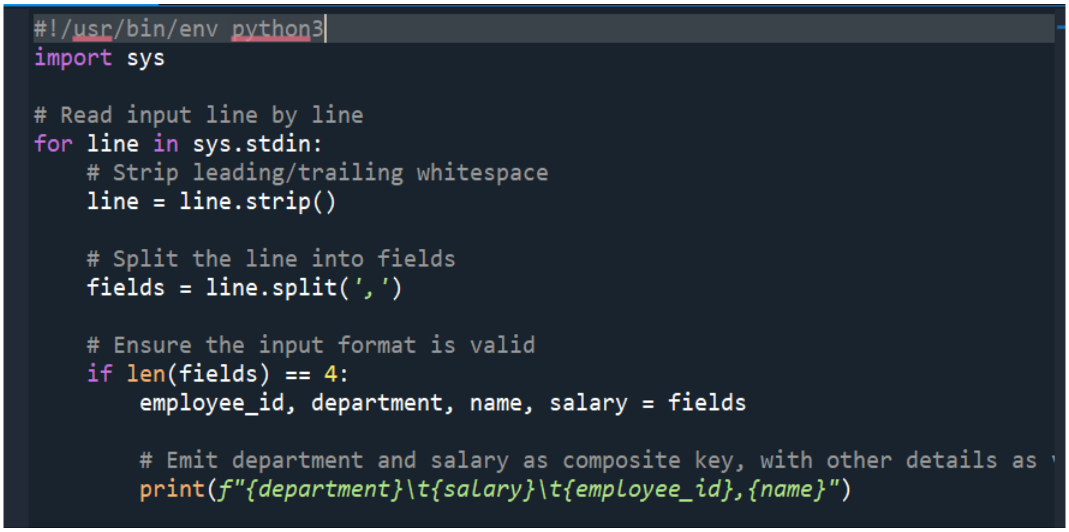
Solution:

In this program, we'll sort employee records by department (primary key) and salary (secondary key) in ascending order.

Step-1 : Write mapper and reducer program as follows:

Mapper (4s_mapper.py):

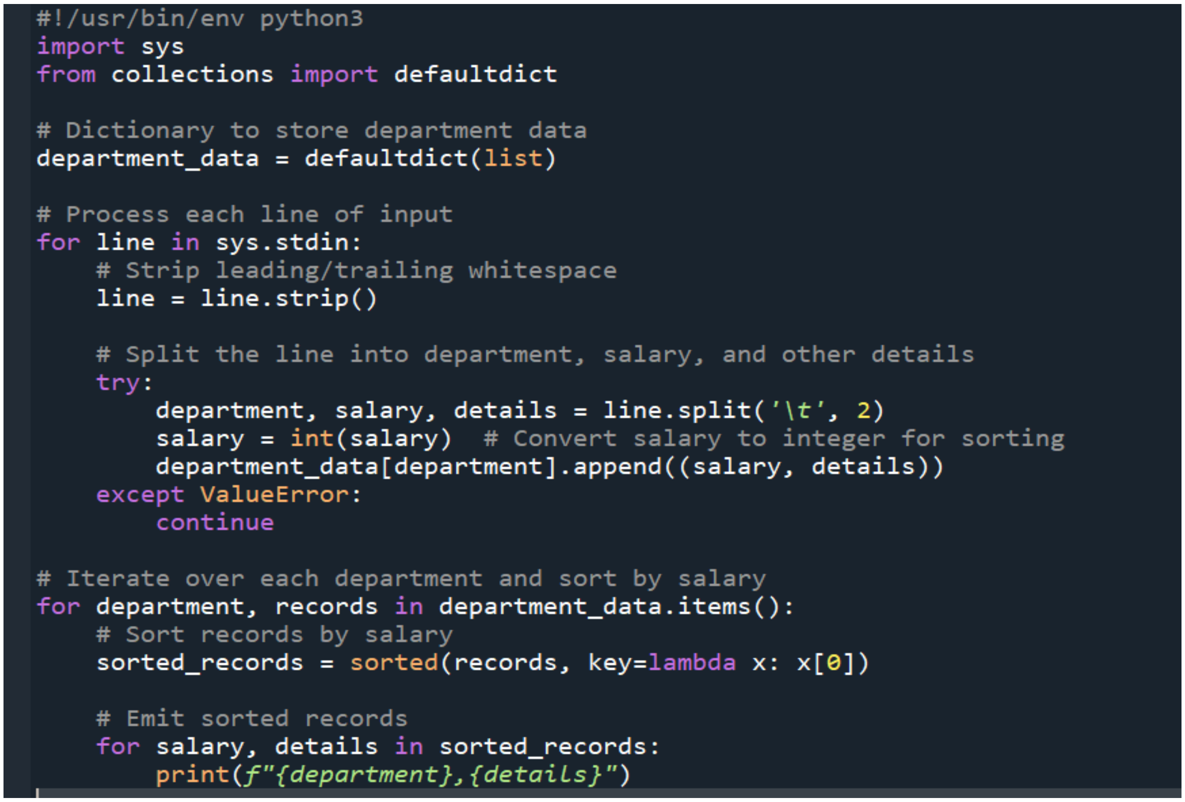
- Emits the department and salary as the key.
- Emits the remaining details (employee ID and name) as the value.
- Sorting is achieved by Hadoop's built-in sorting mechanism, which sorts by key during the shuffle and sort phase.



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

# Read input line by line
for line in sys.stdin:
    # Strip leading/trailing whitespace
    line = line.strip()

    # Split the line into fields
    fields = line.split(',')

    # Ensure the input format is valid
    if len(fields) == 4:
        employee_id, department, name, salary = fields

        # Emit department and salary as composite key, with other details as
        print(f"{department}\t{salary}\t{employee_id},{name}")
```

Reducer (4s_reducer.py):

- Group records by department.
- Sort records within each department by salary using Python’s sorted function.



```python
#!/usr/bin/env python3
import sys
from collections import defaultdict

# Dictionary to store department data
department_data = defaultdict(list)

# Process each line of input
for line in sys.stdin:
    # Strip leading/trailing whitespace
    line = line.strip()

    # Split the line into department, salary, and other details
    try:
        department, salary, details = line.split('\t', 2)
        salary = int(salary)  # Convert salary to integer for sorting
        department_data[department].append((salary, details))
    except ValueError:
        continue

# Iterate over each department and sort by salary
for department, records in department_data.items():
    # Sort records by salary
    sorted_records = sorted(records, key=lambda x: x[0])

    # Emit sorted records
    for salary, details in sorted_records:
        print(f"{department},{details}")
```

Step-2: Create input data (4s_input.txt) and copy it into HDFS:

```
1,HR,Eve,6000
2,HR,Alice,7000
3,HR,Charlie,8000
4,IT,Bob,9000
5,IT,Dave,9500
```

To load file on Hadoop:

```
hdfs dfs -mkdir /user/studentregno/lab4
```

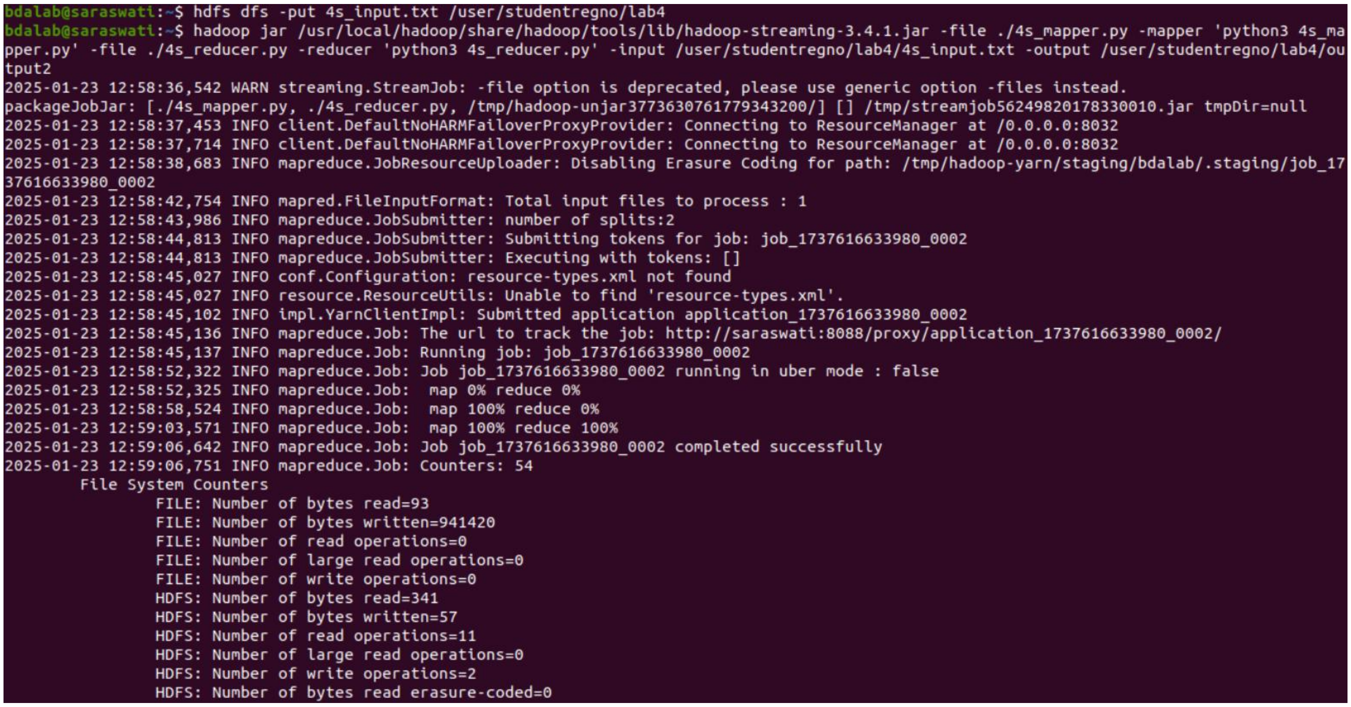
```
hdfs dfs -put 4s_input.txt /user/studentregno/lab4
```

Step-3: Run the program on Hadoop

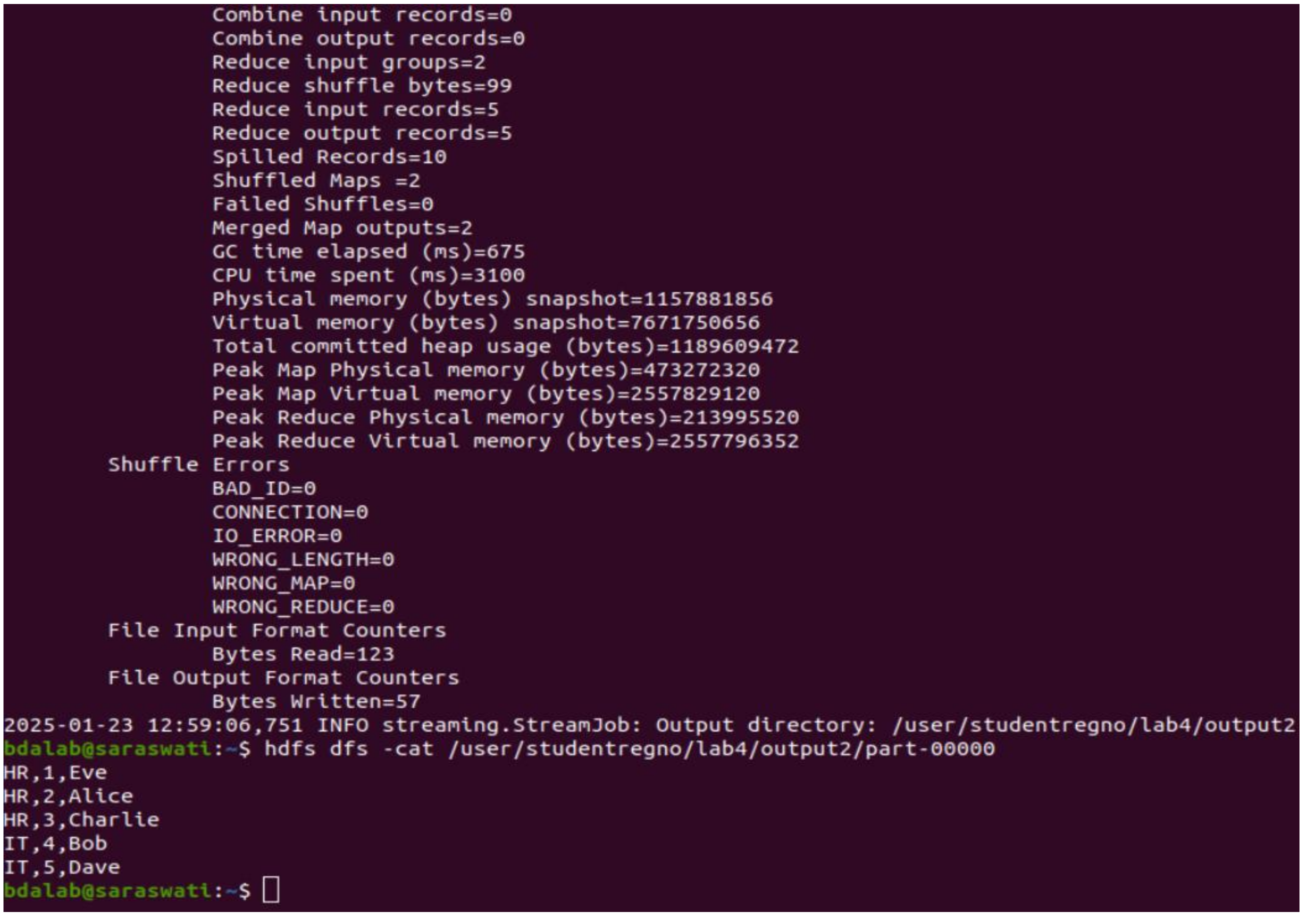
Run the MapReduce Job:

Execute the job using Hadoop Streaming:

```
hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.4.1.jar -file
./4s_mapper.py -mapper 'python3 4s_mapper.py' -file ./4s_reducer.py -reducer
'python3 4s_reducer.py' -input /user/studentregno/lab4/4s_input.txt -output
/user/studentregno/lab4/output1
```



```
bdalab@saraswati:~$ hdfs dfs -put 4s_input.txt /user/studentregno/lab4
bdalab@saraswati:~$ hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.4.1.jar -file ./4s_mapper.py -mapper 'python3 4s_ma
pper.py' -file ./4s_reducer.py -reducer 'python3 4s_reducer.py' -input /user/studentregno/lab4/4s_input.txt -output /user/studentregno/lab4/ou
tput2
2025-01-23 12:58:36,542 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [./4s_mapper.py, ./4s_reducer.py, /tmp/hadoop-unjar3773630761779343200/] [] /tmp/streamjob5624982017833 0010.jar tmpDir=null
2025-01-23 12:58:37,453 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2025-01-23 12:58:37,714 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2025-01-23 12:58:38,683 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/bdalab/.staging/job_17
37616633980_0002
2025-01-23 12:58:42,754 INFO mapred.FileInputFormat: Total input files to process : 1
2025-01-23 12:58:43,986 INFO mapreduce.JobSubmitter: number of splits:2
2025-01-23 12:58:44,813 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1737616633980_0002
2025-01-23 12:58:44,813 INFO mapreduce.JobSubmitter: Executing with tokens: []
2025-01-23 12:58:45,027 INFO conf.Configuration: resource-types.xml not found
2025-01-23 12:58:45,027 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2025-01-23 12:58:45,102 INFO impl.YarnClientImpl: Submitted application application_1737616633980_0002
2025-01-23 12:58:45,136 INFO mapreduce.Job: The url to track the job: http://saraswati:8088/proxy/application_1737616633980_0002/
2025-01-23 12:58:45,137 INFO mapreduce.Job: Running job: job_1737616633980_0002
2025-01-23 12:58:52,322 INFO mapreduce.Job: Job job_1737616633980_0002 running in uber mode : false
2025-01-23 12:58:52,325 INFO mapreduce.Job:  map 0% reduce 0%
2025-01-23 12:58:58,524 INFO mapreduce.Job:  map 100% reduce 0%
2025-01-23 12:59:03,571 INFO mapreduce.Job:  map 100% reduce 100%
2025-01-23 12:59:06,642 INFO mapreduce.Job: Job job_1737616633980_0002 completed successfully
2025-01-23 12:59:06,751 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=93
                FILE: Number of bytes written=941420
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=341
                HDFS: Number of bytes written=57
                HDFS: Number of read operations=11
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
```



```
                Combine input records=0
                Combine output records=0
                Reduce input groups=2
                Reduce shuffle bytes=99
                Reduce input records=5
                Reduce output records=5
                Spilled Records=10
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=675
                CPU time spent (ms)=3100
                Physical memory (bytes) snapshot=1157881856
                Virtual memory (bytes) snapshot=7671750656
                Total committed heap usage (bytes)=1189609472
                Peak Map Physical memory (bytes)=473272320
                Peak Map Virtual memory (bytes)=2557829120
                Peak Reduce Physical memory (bytes)=213995520
                Peak Reduce Virtual memory (bytes)=2557796352
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=123
        File Output Format Counters
                Bytes Written=57
2025-01-23 12:59:06,751 INFO streaming.StreamJob: Output directory: /user/studentregno/lab4/output2
bdalab@saraswati:~$ hdfs dfs -cat /user/studentregno/lab4/output2/part-00000
HR,1,Eve
HR,2,Alice
HR,3,Charlie
IT,4,Bob
IT,5,Dave
bdalab@saraswati:~$ 
```

Week-4 Exercise:

- Write a Hadoop MapReduce program to perform a **reduce-side join** to merge two datasets based on a common key.

  Hint: Join two datasets:

  Students dataset:

  StudentID,Name,CourseID

  Courses dataset:

  CourseID,CourseName,Sem

  Perform an **inner join** to output:

  StudentID,Name,CourseName,Sem

- Write a Hadoop MapReduce program to implement a distributed **word count** program that filters out stop words during processing.

Hint: Count the frequency of each word in a text dataset while ignoring stop words (e.g., "is," "the," "and").

Sample Input:

```
Doc1    This is a sample document
Doc2    Another document with sample text
```

Expected Output:

```
Another    1
Document   2
Sample     2
Text       1
```

- Write a Hadoop MapReduce program to create an **inverted index** that maps each word to the list of documents in which it appears.

Hint: Generate an inverted index for a collection of documents.

Sample Input:

```
Doc1    Hadoop MapReduce is powerful
Doc2    MapReduce is scalable
Doc3    Hadoop and Spark are popular
```

Expected Output:

```
Hadoop      Doc1,Doc3
MapReduce   Doc1,Doc2
Powerful    Doc1
Scalable    Doc2
Spark       Doc3
Popular     Doc3
```

Week-5 Exploring Pig

What is Apache Pig?

Apache Pig is a high-level data flow scripting language used to process large datasets in Hadoop. It provides an abstraction over MapReduce, allowing users to write simpler scripts instead of complex Java code. Pig uses its own scripting language called Pig Latin, which is similar to SQL but more flexible for data transformations and ETL (Extract, Transform, Load) operations.

Key Features of Apache Pig

- Ease of Use – Uses a simple, SQL-like language (Pig Latin).
- Abstraction Over MapReduce – No need to write complex Java code.
- Scalability – Runs efficiently on large datasets.
- Extensibility – Supports User-Defined Functions (UDFs) in Python and Java.
- Flexibility – Handles both structured and semi-structured data.

Apache Pig vs. MapReduce

Feature	**Apache Pig**	**MapReduce**
Language	Pig Latin	Java
Complexity	Simple	Complex
Use Case	ETL, data transformation	Custom logic
Performance	Faster than Hive	Slower

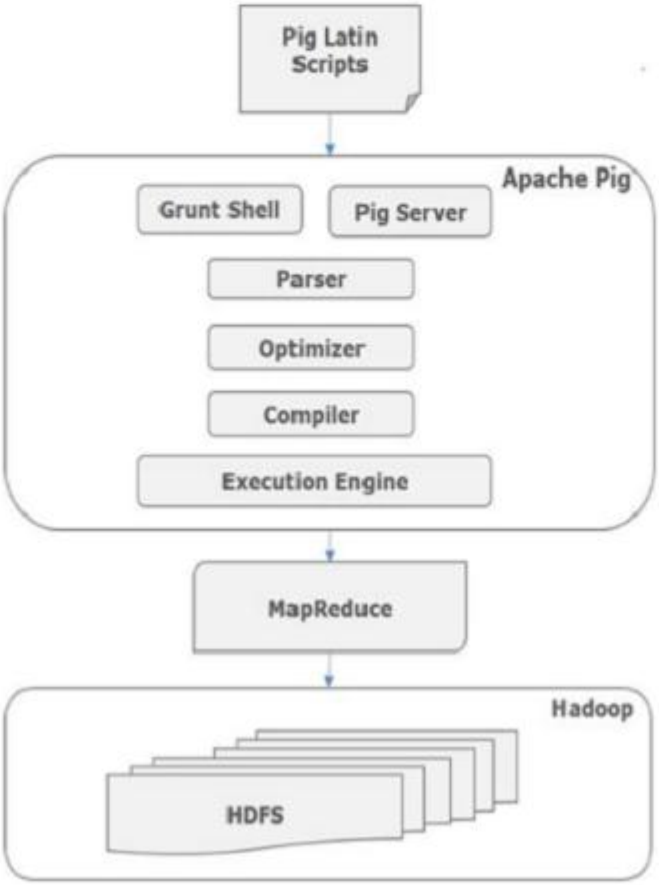
Pig Architecture

Apache Pig consists of:

- Pig Latin Interpreter – Parses and converts scripts into MapReduce jobs.
- Grunt Shell – Interactive command-line interface to run Pig scripts.
- Execution Engine – Converts scripts into optimized execution plans.
- HDFS / Local Mode – Works on Hadoop HDFS or standalone mode.

Modes of Execution:

- Local Mode – Runs without Hadoop (pig -x local).
- MapReduce Mode – Runs on Hadoop (pig -x mapreduce).



When to Use Apache Pig?

- ETL (Extract, Transform, Load) workflows
- Data preprocessing before analysis
- Handling semi-structured data like logs, JSON, CSV
- Quick prototyping compared to MapReduce

Apache Pig is widely used in big data processing pipelines at companies like Yahoo, Twitter, and LinkedIn.

Solved Exercise: First Pig Script: Word Count

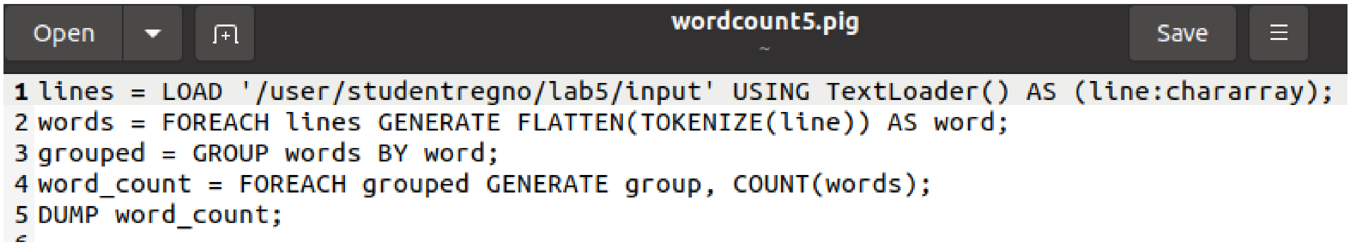
Step 1: Create a Sample Input File (input.txt)

```
Manipal Institute of Technology Manipal
Manipal Academy of Higher Education
Department of Data Science and Computer Application
Manipal, Karnataka, India
```

Load it into hdfs:

```
hdfs dfs -mkdir /user/studentregno/lab5
hdfs dfs -put input.txt /user /studentregno/lab5
```

Step 2: Write Pig Latin Script and save it as wordcount5.pig



```
lines = LOAD '/user/studentregno/lab5/input' USING TextLoader() AS (line:chararray);
words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
grouped = GROUP words BY word;
word_count = FOREACH grouped GENERATE group, COUNT(words);
DUMP word_count;
```

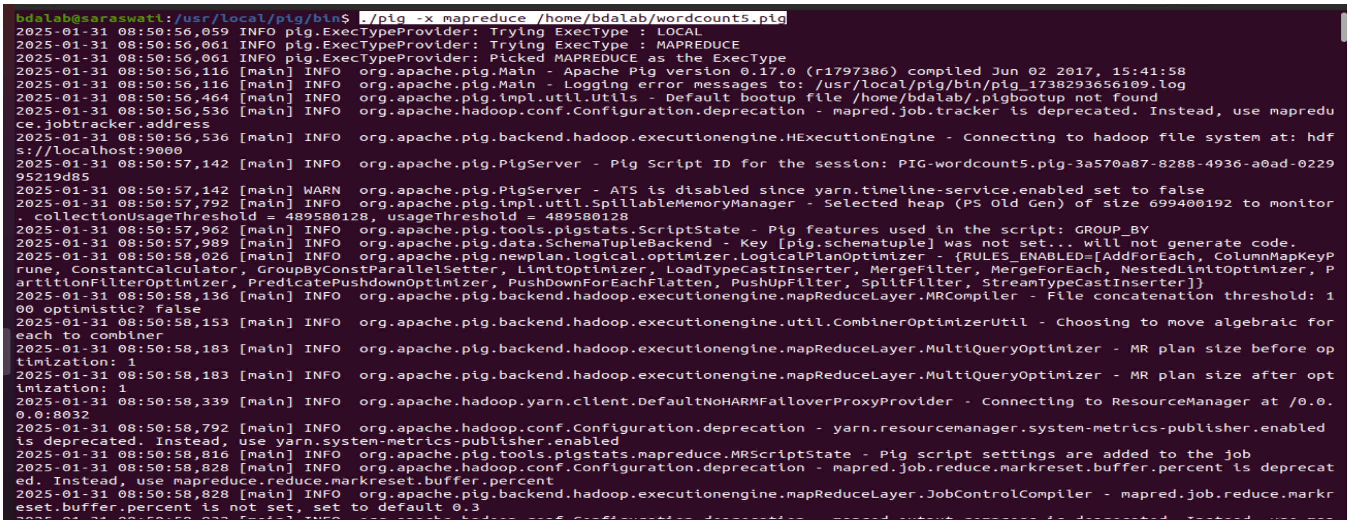
Step 3: Run the Script

Switch to Pig bin directory

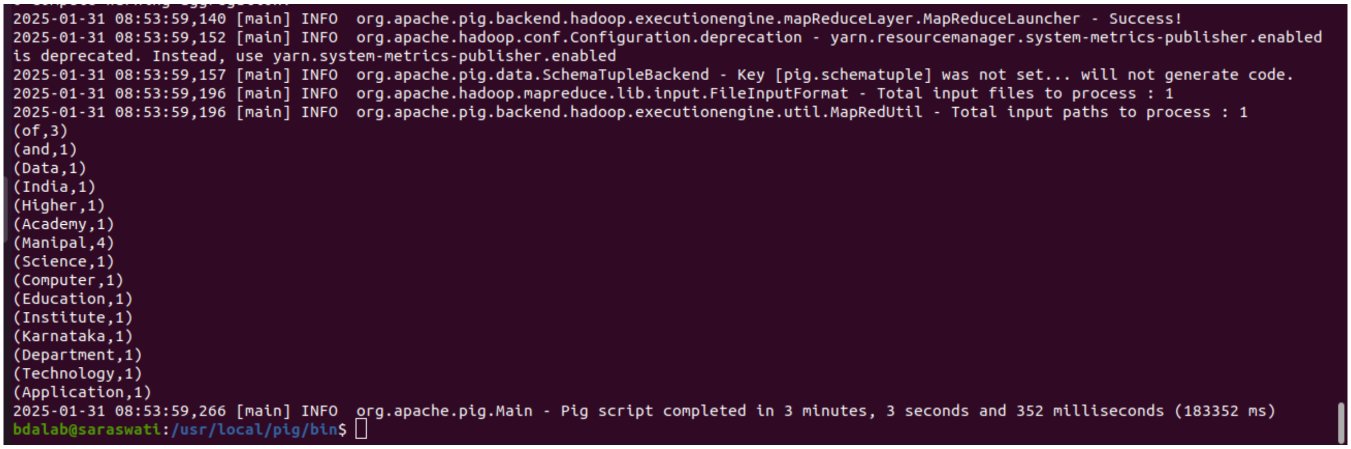
```
cd /usr/local/pig/bin
```

Run the script with following command

```
./pig -x mapreduce /home/bdalab/wordcount5.pig
```



```
bdalab@saraswati:/usr/local/pig/bin$ ./pig -x mapreduce /home/bdalab/wordcount5.pig
2025-01-31 08:50:56,059 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
2025-01-31 08:50:56,061 INFO pig.ExecTypeProvider: Trying ExecType : MAPREDUCE
2025-01-31 08:50:56,061 INFO pig.ExecTypeProvider: Picked MAPREDUCE as the ExecType
2025-01-31 08:50:56,116 [main] INFO  org.apache.pig.Main - Apache Pig version 0.17.0 (r1797386) compiled Jun 02 2017, 15:41:58
2025-01-31 08:50:56,116 [main] INFO  org.apache.pig.Main - Logging error messages to: /usr/local/pig/bin/pig_1738293656109.log
2025-01-31 08:50:56,464 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file /home/bdalab/.pigbootup not found
2025-01-31 08:50:56,536 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapredu
ce.jobtracker.address
2025-01-31 08:50:56,536 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: hdf
s://localhost:9000
2025-01-31 08:50:57,142 [main] INFO  org.apache.pig.PigServer - Pig Script ID for the session: PIG-wordcount5.pig-3a570a87-8288-4936-a0ad-0229
95219d85
2025-01-31 08:50:57,142 [main] WARN  org.apache.pig.PigServer - ATS is disabled since yarn.timeline-service.enabled set to false
2025-01-31 08:50:57,792 [main] INFO  org.apache.pig.impl.util.SpillableMemoryManager - Selected heap (PS Old Gen) of size 699400192 to monitor
. collectionUsageThreshold = 489580128, usageThreshold = 489580128
2025-01-31 08:50:57,962 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: GROUP_BY
2025-01-31 08:50:57,989 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2025-01-31 08:50:58,026 [main] INFO  org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach, ColumnMapKeyP
rune, ConstantCalculator, GroupByConstParallelSetter, LimitOptimizer, LoadTypeCastInserter, MergeFilter, MergeForEach, NestedLimitOptimizer, P
artitionFilterOptimizer, PredicatePushdownOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFilter, StreamTypeCastInserter]}
2025-01-31 08:50:58,136 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MRCompiler - File concatenation threshold: 1
00 optimistic? false
2025-01-31 08:50:58,153 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.CombinerOptimizerUtil - Choosing to move algebraic for
each to combiner
2025-01-31 08:50:58,183 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size before op
timization: 1
2025-01-31 08:50:58,183 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size after opt
imization: 1
2025-01-31 08:50:58,339 [main] INFO  org.apache.hadoop.yarn.client.DefaultNoHARMFailoverProxyProvider - Connecting to ResourceManager at /0.0.
0.0:8032
2025-01-31 08:50:58,792 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - yarn.resourcemanager.system-metrics-publisher.enabled
is deprecated. Instead, use yarn.system-metrics-publisher.enabled
2025-01-31 08:50:58,816 [main] INFO  org.apache.pig.tools.pigstats.mapreduce.MRScriptState - Pig script settings are added to the job
2025-01-31 08:50:58,828 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.reduce.markreset.buffer.percent is deprecat
ed. Instead, use mapreduce.reduce.markreset.buffer.percent
2025-01-31 08:50:58,828 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - mapred.job.reduce.markr
eset.buffer.percent is not set, set to default 0.3
```



```
2025-01-31 08:53:59,140 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2025-01-31 08:53:59,152 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - yarn.resourcemanager.system-metrics-publisher.enabled
is deprecated. Instead, use yarn.system-metrics-publisher.enabled
2025-01-31 08:53:59,157 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2025-01-31 08:53:59,196 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1
2025-01-31 08:53:59,196 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(of,3)
(and,1)
(Data,1)
(India,1)
(Higher,1)
(Academy,1)
(Manipal,4)
(Science,1)
(Computer,1)
(Education,1)
(Institute,1)
(Karnataka,1)
(Department,1)
(Technology,1)
(Application,1)
2025-01-31 08:53:59,266 [main] INFO  org.apache.pig.Main - Pig script completed in 3 minutes, 3 seconds and 352 milliseconds (183352 ms)
bdalab@saraswati:/usr/local/pig/bin$
```

Working with Datasets in Pig:

Step-1:Loading Data into Pig:

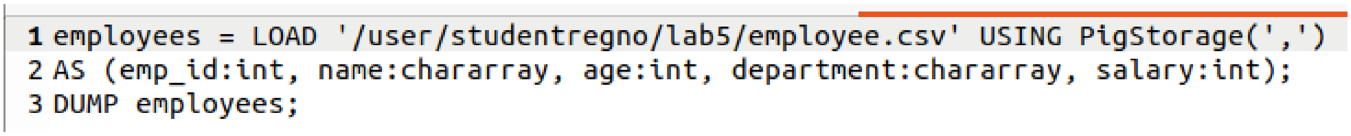
Create a .csv file (employee.csv)

```
101,John,30,HR,50000
102,Alice,25,IT,60000
103,Bob,35,Finance,70000
104,Joy,36,Manager,55000
```

Load it on HDFS

```
hdfs dfs -put employee.csv /user/studentregno/lab5
```

Step 2: Write Pig Latin Script and save it as loadcsv.pig

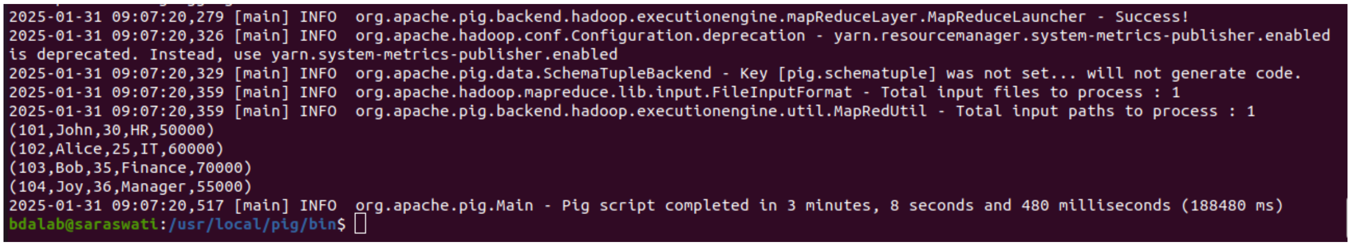


```
employees = LOAD '/user/studentregno/lab5/employee.csv' USING PigStorage(',')
AS (emp_id:int, name:chararray, age:int, department:chararray, salary:int);
DUMP employees;
```

Step-3: Run the script with following command

Loading csv:

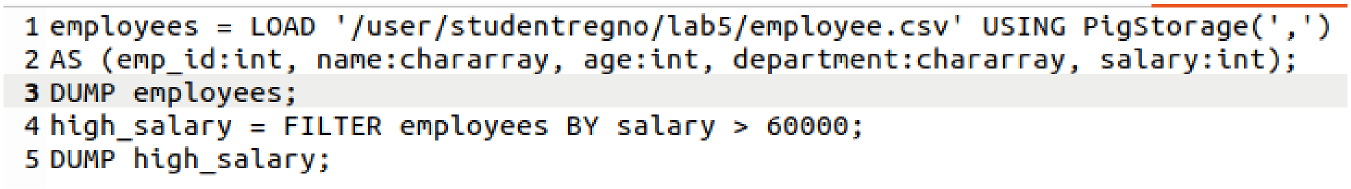
./pig -x mapreduce /home/bdalab/loadcsv.pig



```
2025-01-31 09:07:20,279 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2025-01-31 09:07:20,326 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - yarn.resourcemanager.system-metrics-publisher.enabled
is deprecated. Instead, use yarn.system-metrics-publisher.enabled
2025-01-31 09:07:20,329 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2025-01-31 09:07:20,359 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1
2025-01-31 09:07:20,359 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(101,John,30,HR,50000)
(102,Alice,25,IT,60000)
(103,Bob,35,Finance,70000)
(104,Joy,36,Manager,55000)
2025-01-31 09:07:20,517 [main] INFO  org.apache.pig.Main - Pig script completed in 3 minutes, 8 seconds and 480 milliseconds (188480 ms)
bdalab@saraswati:/usr/local/pig/bin$ 
```

Filtering Data: Find employees with a salary greater than 60,000.

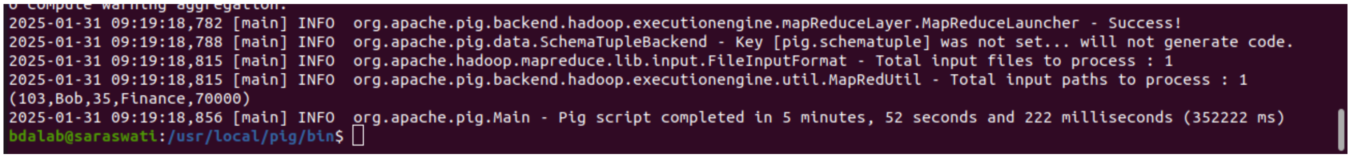
Write Pig Latin Script and save it as filter.pig



```
employees = LOAD '/user/studentregno/lab5/employee.csv' USING PigStorage(',')
AS (emp_id:int, name:chararray, age:int, department:chararray, salary:int);
DUMP employees;
high_salary = FILTER employees BY salary > 60000;
DUMP high_salary;
```

Run the script:

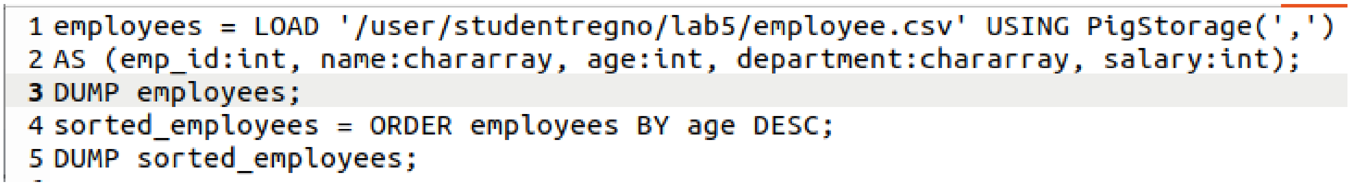
./pig -x mapreduce /home/bdalab/filter.pig



```
o compute warning aggregation.
2025-01-31 09:19:18,782 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2025-01-31 09:19:18,788 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2025-01-31 09:19:18,815 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1
2025-01-31 09:19:18,815 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(103,Bob,35,Finance,70000)
2025-01-31 09:19:18,856 [main] INFO  org.apache.pig.Main - Pig script completed in 5 minutes, 52 seconds and 222 milliseconds (352222 ms)
bdalab@saraswati:/usr/local/pig/bin$ 
```

Sorting Data: Sort employees by age in descending order.

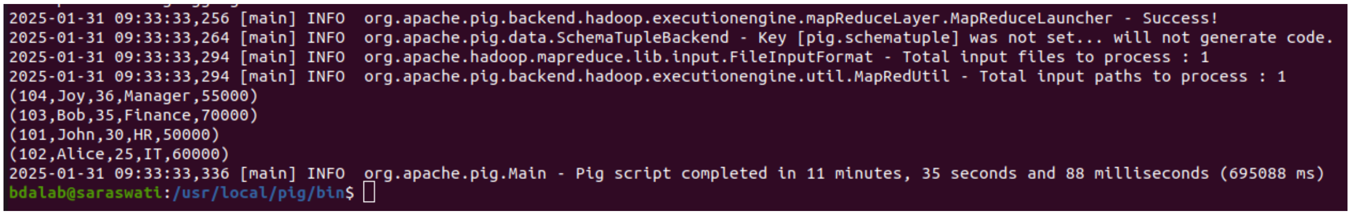
Write Pig Latin Script and save it as sortd.pig



```
employees = LOAD '/user/studentregno/lab5/employee.csv' USING PigStorage(',')
AS (emp_id:int, name:chararray, age:int, department:chararray, salary:int);
DUMP employees;
sorted_employees = ORDER employees BY age DESC;
DUMP sorted_employees;
```

Run the script:

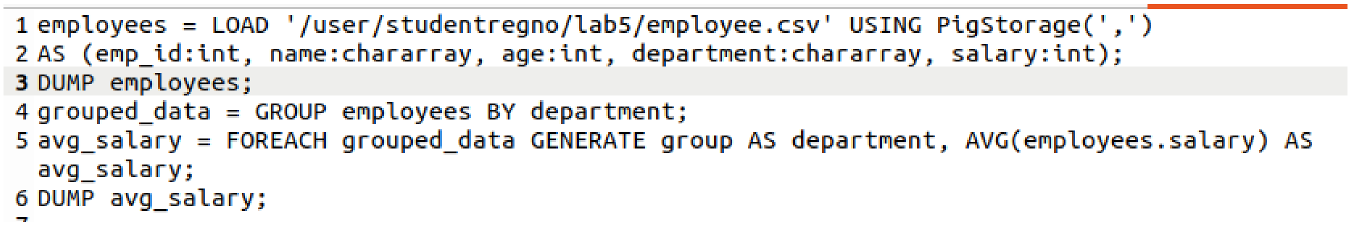
./pig -x mapreduce /home/bdalab/sortd.pig



```
2025-01-31 09:33:33,256 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2025-01-31 09:33:33,264 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2025-01-31 09:33:33,294 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1
2025-01-31 09:33:33,294 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(104,Joy,36,Manager,55000)
(103,Bob,35,Finance,70000)
(101,John,30,HR,50000)
(102,Alice,25,IT,60000)
2025-01-31 09:33:33,336 [main] INFO  org.apache.pig.Main - Pig script completed in 11 minutes, 35 seconds and 88 milliseconds (695088 ms)
bdalab@saraswati:/usr/local/pig/bin$ 
```

Aggregations: Average Salary by Department

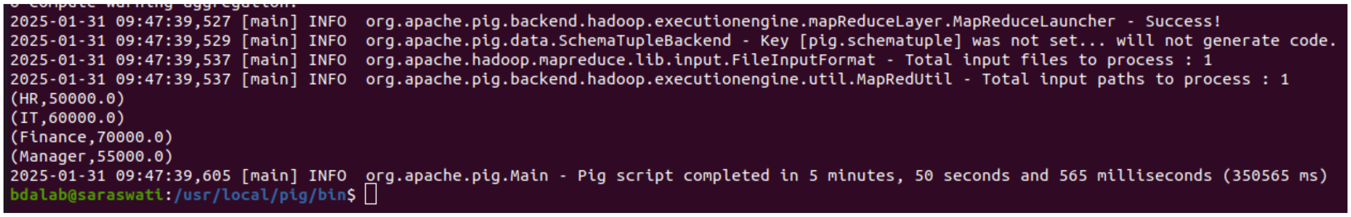
Write Pig Latin Script and save it as average.pig



```
employees = LOAD '/user/studentregno/lab5/employee.csv' USING PigStorage(',')
AS (emp_id:int, name:chararray, age:int, department:chararray, salary:int);
DUMP employees;
grouped_data = GROUP employees BY department;
avg_salary = FOREACH grouped_data GENERATE group AS department, AVG(employees.salary) AS
avg_salary;
DUMP avg_salary;
```

Run the script:

./pig -x mapreduce /home/bdalab/average.pig



```
2025-01-31 09:47:39,527 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2025-01-31 09:47:39,529 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2025-01-31 09:47:39,537 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1
2025-01-31 09:47:39,537 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(HR,50000.0)
(IT,60000.0)
(Finance,70000.0)
(Manager,55000.0)
2025-01-31 09:47:39,605 [main] INFO  org.apache.pig.Main - Pig script completed in 5 minutes, 50 seconds and 565 milliseconds (350565 ms)
bdalab@saraswati:/usr/local/pig/bin$ 
```

Joining Two Datasets

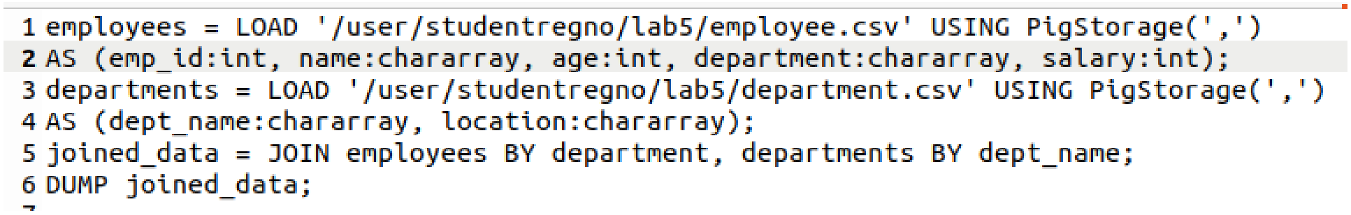
Create a .csv file (department.csv)

```
HR,Bangalore
IT,Bangalore
Finance,Mumbai
Manager,NewYork
```

Load it on HDFS

hdfs dfs -put department.csv /user/studentregno/lab5

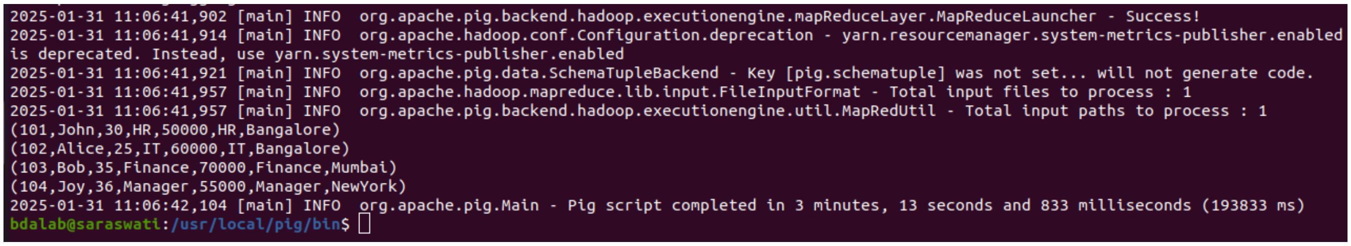
Write Pig Latin Script and save it as join.pig



```
employees = LOAD '/user/studentregno/lab5/employee.csv' USING PigStorage(',')
AS (emp_id:int, name:chararray, age:int, department:chararray, salary:int);
departments = LOAD '/user/studentregno/lab5/department.csv' USING PigStorage(',')
AS (dept_name:chararray, location:chararray);
joined_data = JOIN employees BY department, departments BY dept_name;
DUMP joined_data;
```

Run the script:

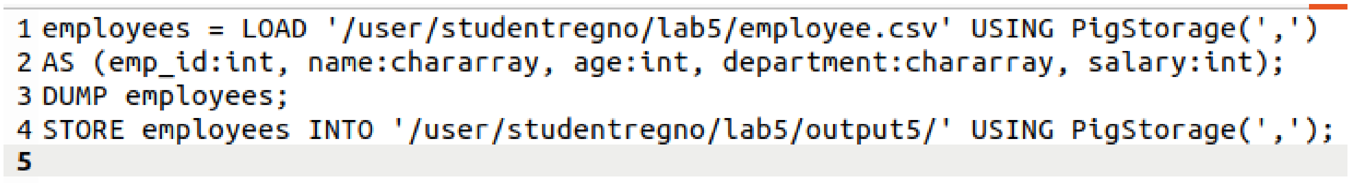
./pig -x mapreduce /home/bdalab/join.pig



```
2025-01-31 11:06:41,902 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2025-01-31 11:06:41,914 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - yarn.resourcemanager.system-metrics-publisher.enabled
is deprecated. Instead, use yarn.system-metrics-publisher.enabled
2025-01-31 11:06:41,921 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2025-01-31 11:06:41,957 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1
2025-01-31 11:06:41,957 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(101,John,30,HR,50000,HR,Bangalore)
(102,Alice,25,IT,60000,IT,Bangalore)
(103,Bob,35,Finance,70000,Finance,Mumbai)
(104,Joy,36,Manager,55000,Manager,NewYork)
2025-01-31 11:06:42,104 [main] INFO  org.apache.pig.Main - Pig script completed in 3 minutes, 13 seconds and 833 milliseconds (193833 ms)
bdalab@saraswati:/usr/local/pig/bin$ 
```

Store Processed Data to HDFS

Write Pig Latin Script and save it as storehdfs.pig



```
employees = LOAD '/user/studentregno/lab5/employee.csv' USING PigStorage(',')
AS (emp_id:int, name:chararray, age:int, department:chararray, salary:int);
DUMP employees;
STORE employees INTO '/user/studentregno/lab5/output5/' USING PigStorage(',');
```

Run the script:

```
./pig -x mapreduce /home/bdalab/storehdfs.pig
```

After completion can see the data in HDFS output directory:

```
hdfs dfs -cat /user/studentregno/lab5/output5/part-m-00000
```

Week-5 Exercise (use Mapreduce mode):

1. Create a sales dataset (order_id,product,category,amount). Load it into Pig. Display the first 10 records
2. Load the sales dataset. Filter and display products where the amount is greater than 5000.
3. Load the sales dataset. Sort the products in descending order of amount. Display the sorted results.
4. Load the sales dataset. Group sales by category. Find the total sales amount for each category.
5. Create customer dataset (order_id,customer_name,city). Load sales dataset and customers datasets. Perform a JOIN operation on the order_id column. Display the results.
6. Find the top 3 most expensive products from the sales dataset.
7. Store the filtered high-value transactions (amount > 5000) in an HDFS output directory.

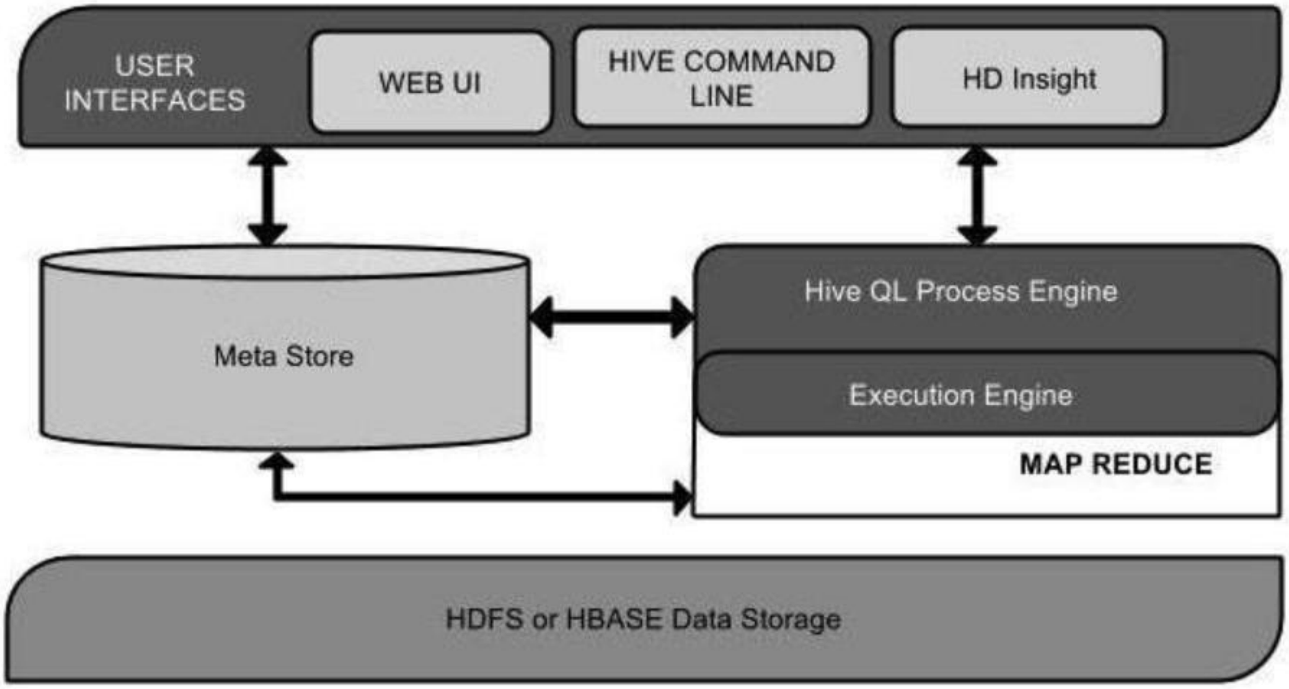
Week-6 Exploring Hive

Apache Hive is a data warehouse infrastructure built on top of Apache Hadoop that enables SQL-like querying on large datasets stored in HDFS (Hadoop Distributed File System). It is primarily used for data analysis, ETL (Extract, Transform, Load) operations, and business intelligence.

Why Use Hive?

- SQL-Like Queries – Uses HiveQL, which is similar to SQL.
- Handles Big Data – Efficiently processes petabytes of structured/unstructured data.
- Scalability – Runs on top of Hadoop, making it highly scalable.
- Schema-on-Read – Unlike traditional databases, data is interpreted at query time.
- Supports Various File Formats – Works with CSV, ORC, Parquet, JSON, and more.

Hive Architecture



Hive consists of several key components:

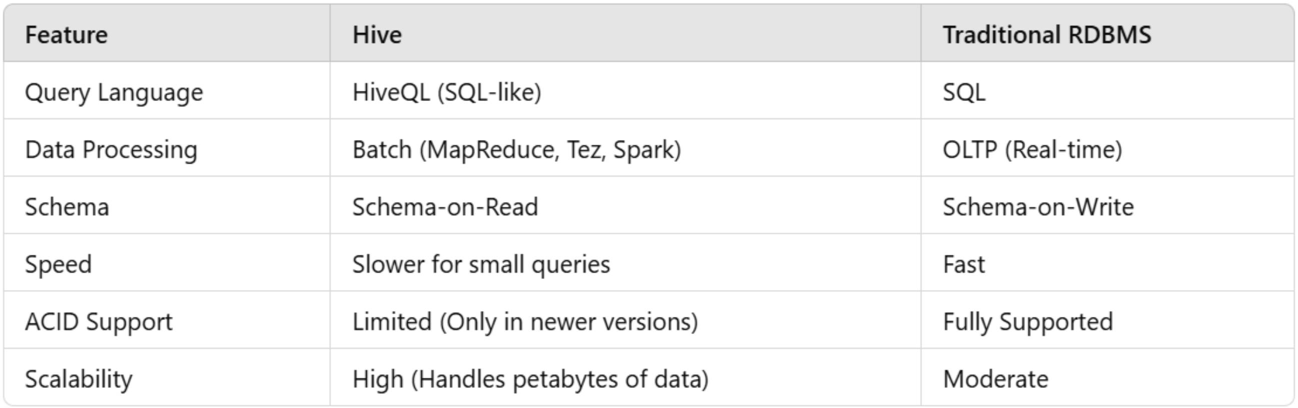
1. **Metastore** – Stores metadata (schema, table definitions, partitions).
2. **Driver** – Manages query execution lifecycle.
3. **Query Engine** – Converts HiveQL into execution plans.

4. **Execution Engine** – Runs queries using MapReduce, Tez, or Spark.
5. **HDFS** – Stores the actual data.

Key Features of Hive

- **Tables and Partitions** – Data is stored in tables that can be partitioned for faster queries.
- **User-Defined Functions (UDFs)** – Allows custom functions in Java/Python.
- **Joins and Aggregations** – Supports complex queries, joins, and analytics.
- **Optimization Features** – Partitioning, bucketing, indexing, and ORC file format.
- **Integration** – Works with Spark, Presto, Impala, and BI tools like Tableau.

Hive vs. Traditional Databases



Feature	Hive	Traditional RDBMS
Query Language	HiveQL (SQL-like)	SQL
Data Processing	Batch (MapReduce, Tez, Spark)	OLTP (Real-time)
Schema	Schema-on-Read	Schema-on-Write
Speed	Slower for small queries	Fast
ACID Support	Limited (Only in newer versions)	Fully Supported
Scalability	High (Handles petabytes of data)	Moderate

When to Use Hive?

- Processing large-scale data (e.g., logs, clickstream, IoT).
- Running batch analytics on Hadoop.
- ETL workloads and data preprocessing.
- Data exploration and reporting.
- Not ideal for: Real-time transactions, low-latency queries, or updates**.**

Working with Hive:

Start Hadoop services

start-dfs.sh

start-yarn.sh

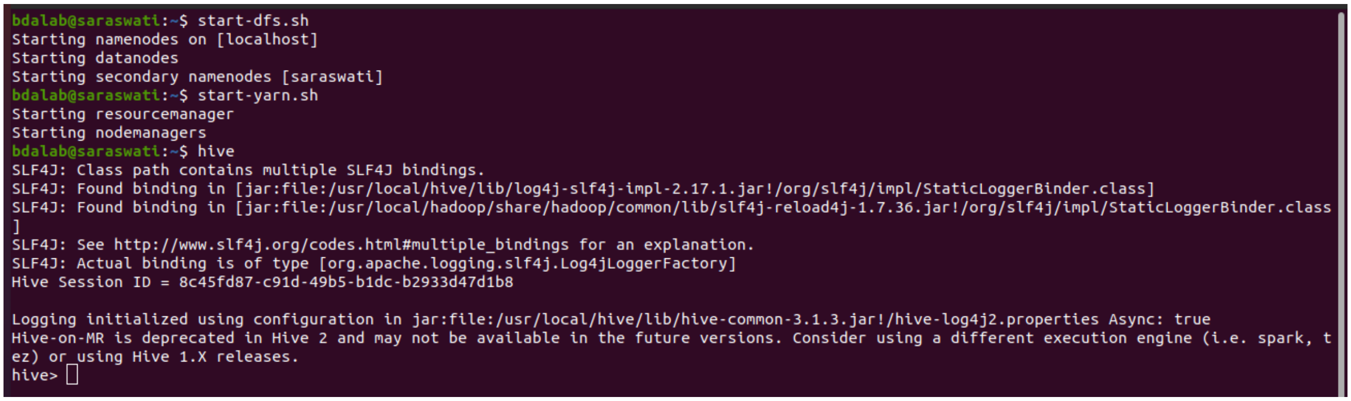
Execute the following command to give the all permission to hive directory:

hdfs dfs -chmod 777 /tmp

Start Hive:

Type following command in terminal

Hive



```
bdalab@saraswati:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [saraswati]
bdalab@saraswati:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
bdalab@saraswati:~$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hive/lib/log4j-slf4j-impl-2.17.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class
]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = 8c45fd87-c91d-49b5-b1dc-b2933d47d1b8

Logging initialized using configuration in jar:file:/usr/local/hive/lib/hive-common-3.1.3.jar!/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, t
ez) or using Hive 1.X releases.
hive> 
```

Hive Database Operations:

Create a new database

CREATE DATABASE sales_db;

List databases

SHOW DATABASES;

Use a database

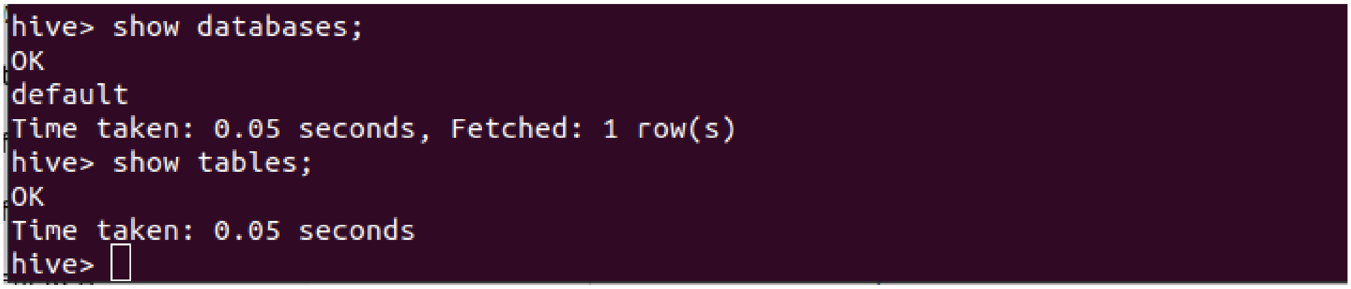
USE sales_db;

Show Table list

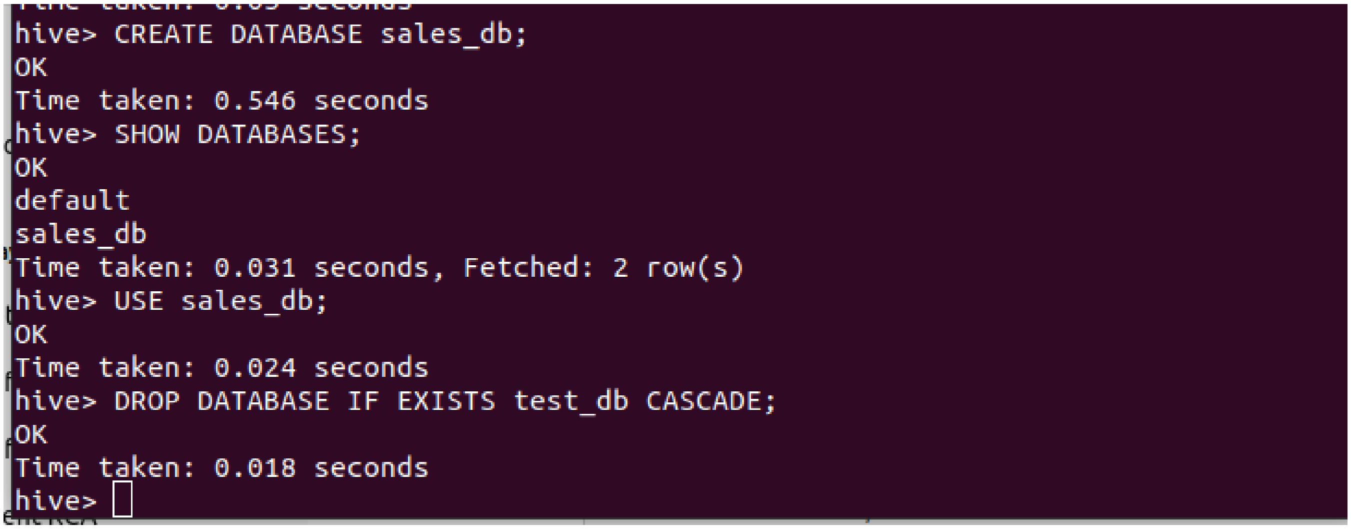
SHOW TABLES;

Drop a database

DROP DATABASE sales_db CASCADE;



```
hive> show databases;
OK
default
Time taken: 0.05 seconds, Fetched: 1 row(s)
hive> show tables;
OK
Time taken: 0.05 seconds
hive> 
```

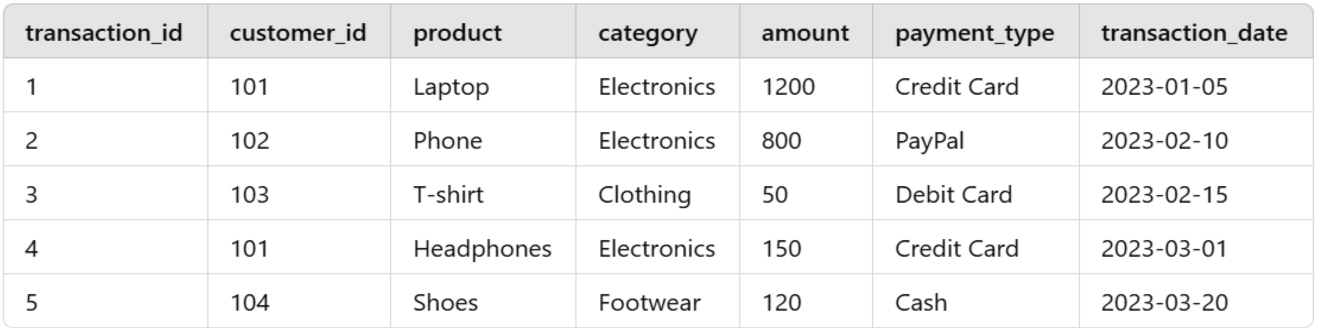


```
hive> CREATE DATABASE sales_db;
OK
Time taken: 0.546 seconds
hive> SHOW DATABASES;
OK
default
sales_db
Time taken: 0.031 seconds, Fetched: 2 row(s)
hive> USE sales_db;
OK
Time taken: 0.024 seconds
hive> DROP DATABASE IF EXISTS test_db CASCADE;
OK
Time taken: 0.018 seconds
hive> 
```

Solved Exercise:

E-Commerce Sales Analytics: An e-commerce company wants to analyze sales trends and customer purchases. The data is stored in Hadoop, and they need insights into total revenue, top customers, and monthly sales trends.

Sample data: (sales_data.csv)



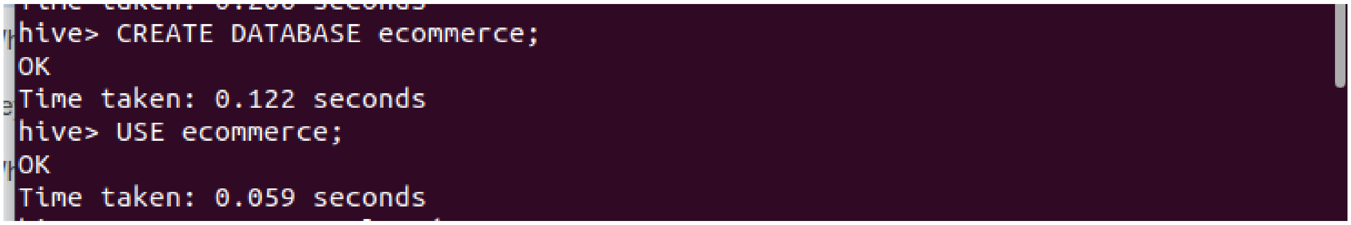
transaction_id	customer_id	product	category	amount	payment_type	transaction_date
1	101	Laptop	Electronics	1200	Credit Card	2023-01-05
2	102	Phone	Electronics	800	PayPal	2023-02-10
3	103	T-shirt	Clothing	50	Debit Card	2023-02-15
4	101	Headphones	Electronics	150	Credit Card	2023-03-01
5	104	Shoes	Footwear	120	Cash	2023-03-20

Create sales_data.csv and load it on HDFS

```
hdfs dfs -mkdir /user/studentregno/lab6
hdfs dfs -put sales_data.csv /user/studentregno/lab6
```

a. Create a Hive database and table

```
CREATE DATABASE ecommerce;
USE ecommerce;
```



```
hive> CREATE DATABASE ecommerce;
OK
Time taken: 0.122 seconds
hive> USE ecommerce;
OK
Time taken: 0.059 seconds
```

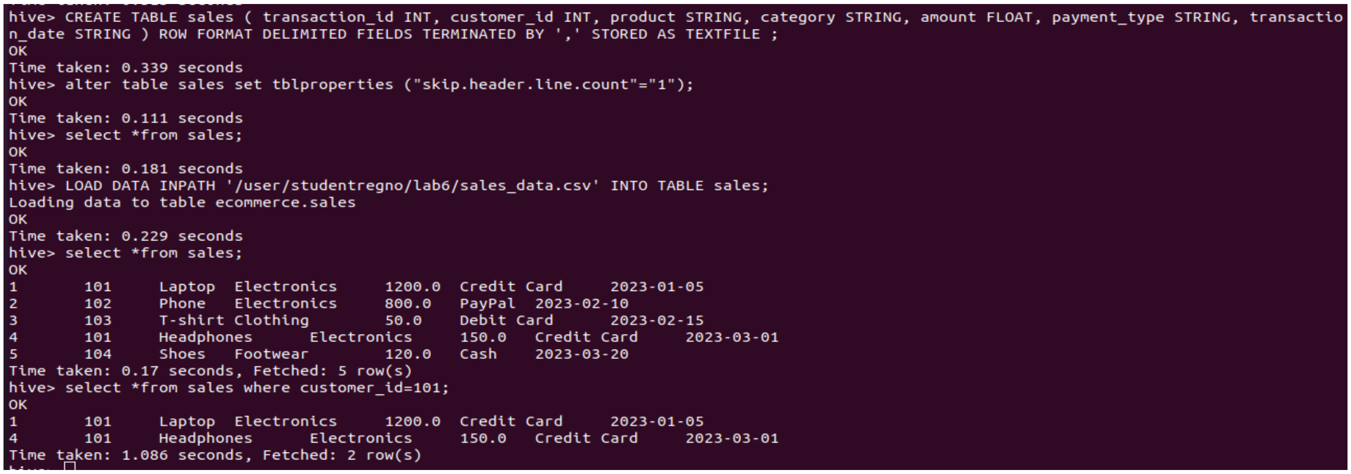
```
CREATE TABLE sales ( transaction_id INT, customer_id INT, product STRING, category STRING, amount FLOAT, payment_type STRING, transaction_date STRING ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE ;
```

To skip the header information:

```
alter table sales set tblproperties ("skip.header.line.count"="1");
```

b. Load Data from HDFS

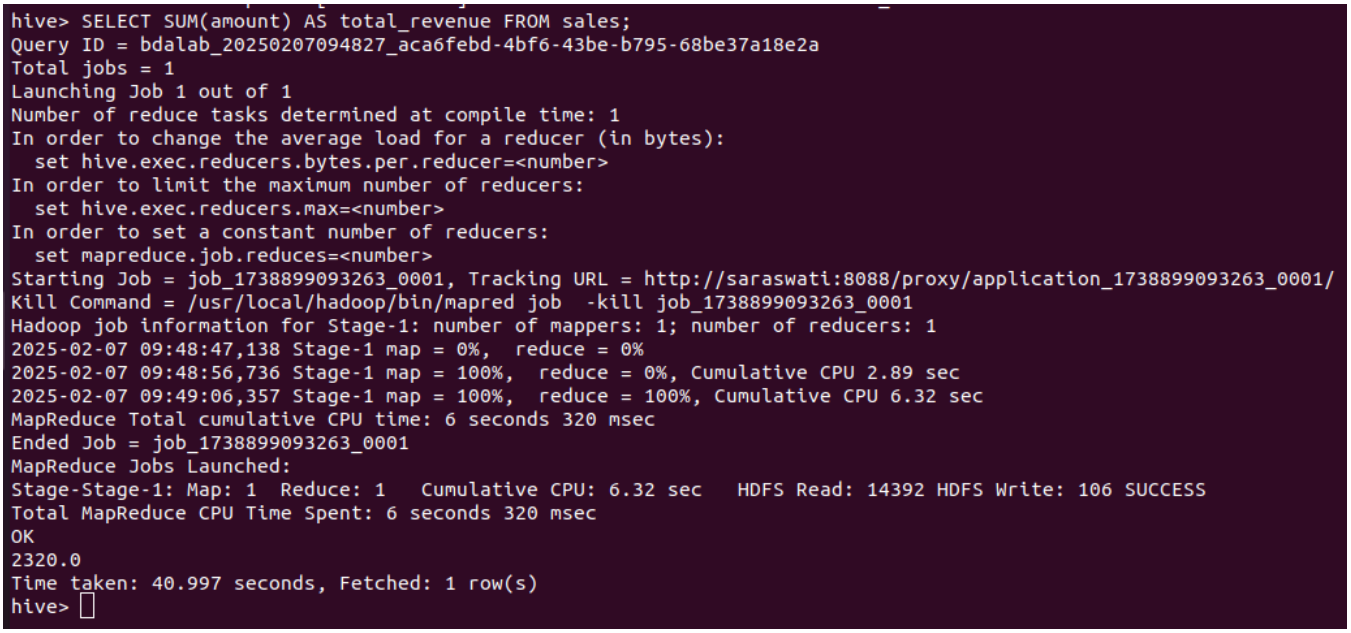
```
LOAD DATA INPATH '/user/studentregno/lab6/sales_data.csv' INTO TABLE sales;
```



```
hive> CREATE TABLE sales ( transaction_id INT, customer_id INT, product STRING, category STRING, amount FLOAT, payment_type STRING, transactio
n_date STRING ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE ;
OK
Time taken: 0.339 seconds
hive> alter table sales set tblproperties ("skip.header.line.count"="1");
OK
Time taken: 0.111 seconds
hive> select *from sales;
OK
Time taken: 0.181 seconds
hive> LOAD DATA INPATH '/user/studentregno/lab6/sales_data.csv' INTO TABLE sales;
Loading data to table ecommerce.sales
OK
Time taken: 0.229 seconds
hive> select *from sales;
OK
1       101     Laptop  Electronics     1200.0  Credit Card     2023-01-05
2       102     Phone   Electronics     800.0   PayPal  2023-02-10
3       103     T-shirt Clothing        50.0    Debit Card      2023-02-15
4       101     Headphones      Electronics     150.0   Credit Card     2023-03-01
5       104     Shoes   Footwear        120.0   Cash    2023-03-20
Time taken: 0.17 seconds, Fetched: 5 row(s)
hive> select *from sales where customer_id=101;
OK
1       101     Laptop  Electronics     1200.0  Credit Card     2023-01-05
4       101     Headphones      Electronics     150.0   Credit Card     2023-03-01
Time taken: 1.086 seconds, Fetched: 2 row(s)
```

c. Analyze Total Revenue:

SELECT SUM(amount) AS total_revenue FROM sales;



```
hive> SELECT SUM(amount) AS total_revenue FROM sales;
Query ID = bdalab_20250207094827_aca6febd-4bf6-43be-b795-68be37a18e2a
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 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_1738899093263_0001, Tracking URL = http://saraswati:8088/proxy/application_1738899093263_0001/
Kill Command = /usr/local/hadoop/bin/mapred job  -kill job_1738899093263_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-02-07 09:48:47,138 Stage-1 map = 0%,  reduce = 0%
2025-02-07 09:48:56,736 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.89 sec
2025-02-07 09:49:06,357 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.32 sec
MapReduce Total cumulative CPU time: 6 seconds 320 msec
Ended Job = job_1738899093263_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.32 sec   HDFS Read: 14392 HDFS Write: 106 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 320 msec
OK
2320.0
Time taken: 40.997 seconds, Fetched: 1 row(s)
hive> 
```

d. Identify the Most Popular Product

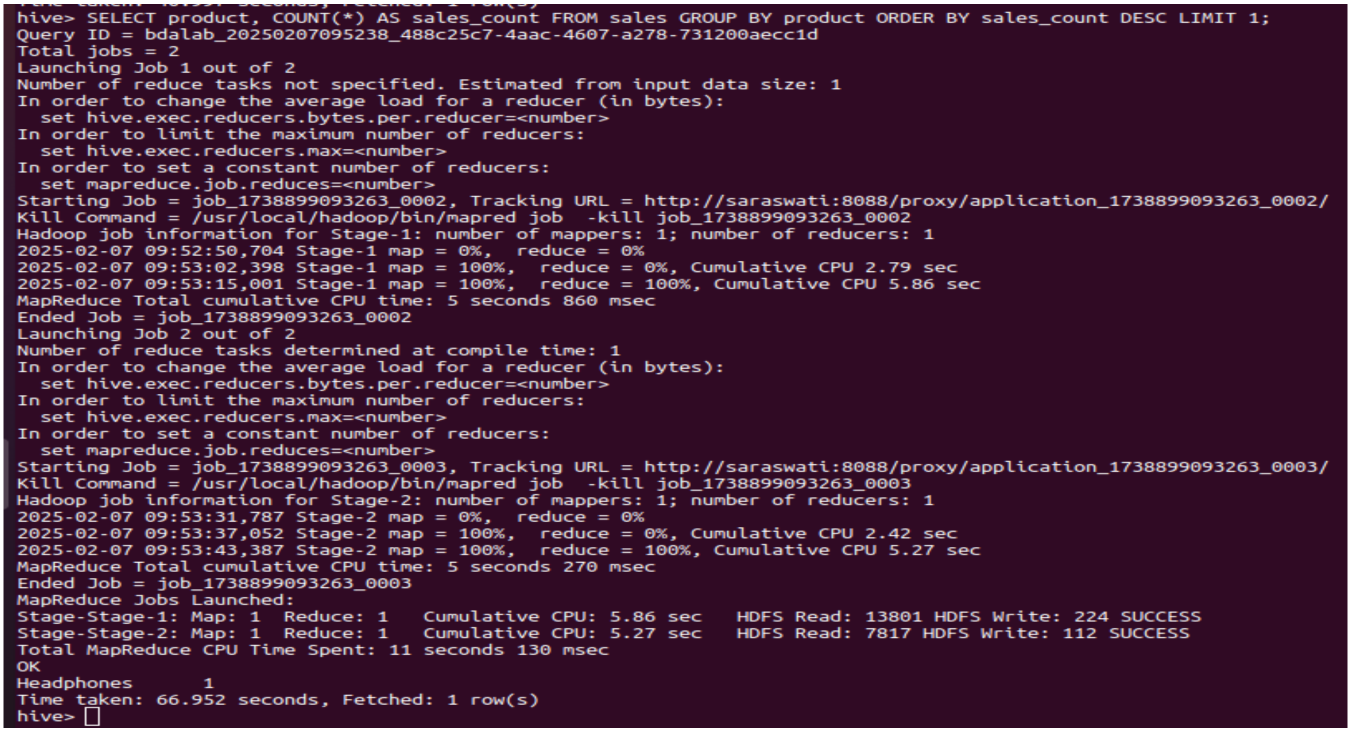
SELECT product, COUNT(*) AS sales_count

FROM sales

GROUP BY product

ORDER BY sales_count DESC

LIMIT 1;



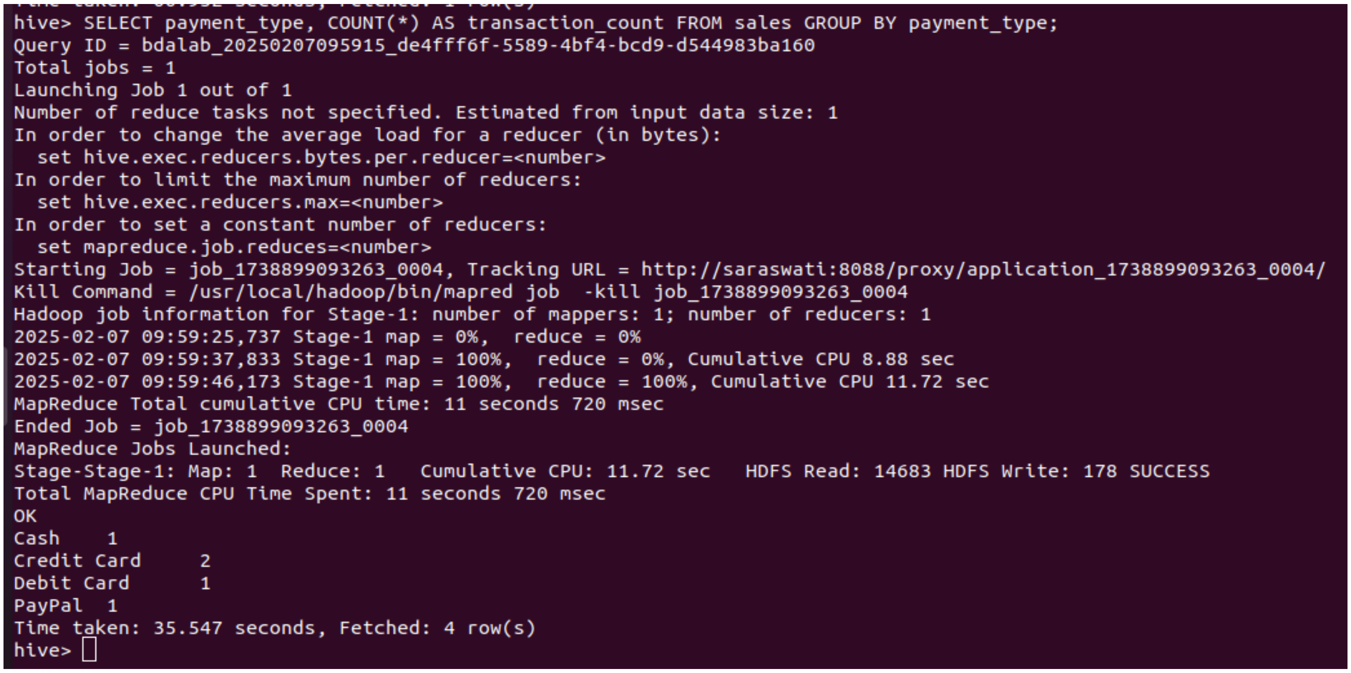
```
hive> SELECT product, COUNT(*) AS sales_count FROM sales GROUP BY product ORDER BY sales_count DESC LIMIT 1;
Query ID = bdalab_20250207095238_488c25c7-4aac-4607-a278-731200aecc1d
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order 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_1738899093263_0002, Tracking URL = http://saraswati:8088/proxy/application_1738899093263_0002/
Kill Command = /usr/local/hadoop/bin/mapred job  -kill job_1738899093263_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-02-07 09:52:50,704 Stage-1 map = 0%,  reduce = 0%
2025-02-07 09:53:02,398 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.79 sec
2025-02-07 09:53:15,001 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 5.86 sec
MapReduce Total cumulative CPU time: 5 seconds 860 msec
Ended Job = job_1738899093263_0002
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 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_1738899093263_0003, Tracking URL = http://saraswati:8088/proxy/application_1738899093263_0003/
Kill Command = /usr/local/hadoop/bin/mapred job  -kill job_1738899093263_0003
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2025-02-07 09:53:31,787 Stage-2 map = 0%,  reduce = 0%
2025-02-07 09:53:37,052 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 2.42 sec
2025-02-07 09:53:43,387 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 5.27 sec
MapReduce Total cumulative CPU time: 5 seconds 270 msec
Ended Job = job_1738899093263_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.86 sec   HDFS Read: 13801 HDFS Write: 224 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 5.27 sec   HDFS Read: 7817 HDFS Write: 112 SUCCESS
Total MapReduce CPU Time Spent: 11 seconds 130 msec
OK
Headphones      1
Time taken: 66.952 seconds, Fetched: 1 row(s)
hive> 
```

e. Analyze Sales by Payment Method

SELECT payment_type, COUNT(*) AS transaction_count

FROM sales

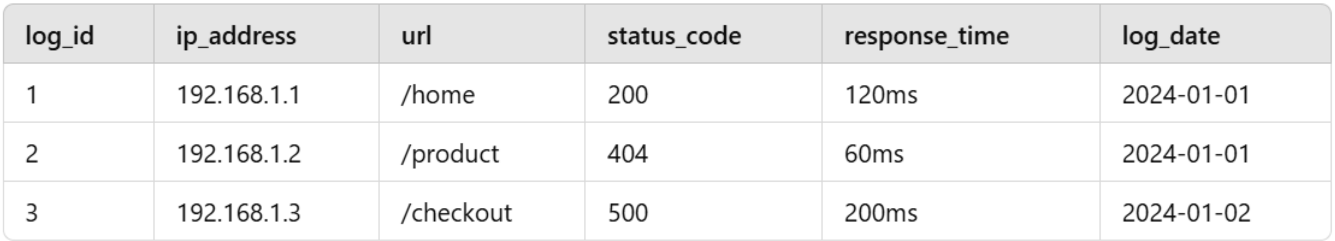
GROUP BY payment_type;



```
hive> SELECT payment_type, COUNT(*) AS transaction_count FROM sales GROUP BY payment_type;
Query ID = bdalab_20250207095915_de4fff6f-5589-4bf4-bcd9-d544983ba160
Total jobs = 1
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_1738899093263_0004, Tracking URL = http://saraswati:8088/proxy/application_1738899093263_0004/
Kill Command = /usr/local/hadoop/bin/mapred job  -kill job_1738899093263_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-02-07 09:59:25,737 Stage-1 map = 0%,  reduce = 0%
2025-02-07 09:59:37,833 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 8.88 sec
2025-02-07 09:59:46,173 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 11.72 sec
MapReduce Total cumulative CPU time: 11 seconds 720 msec
Ended Job = job_1738899093263_0004
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 11.72 sec   HDFS Read: 14683 HDFS Write: 178 SUCCESS
Total MapReduce CPU Time Spent: 11 seconds 720 msec
OK
Cash    1
Credit Card     2
Debit Card      1
PayPal  1
Time taken: 35.547 seconds, Fetched: 4 row(s)
hive> 
```

Week-6 Exercises:

1. A tech company wants to analyze web server logs to monitor website traffic, detect peak usage times, and track 404 errors. Do the Log analysis of server for the dataset (weblogs.csv)

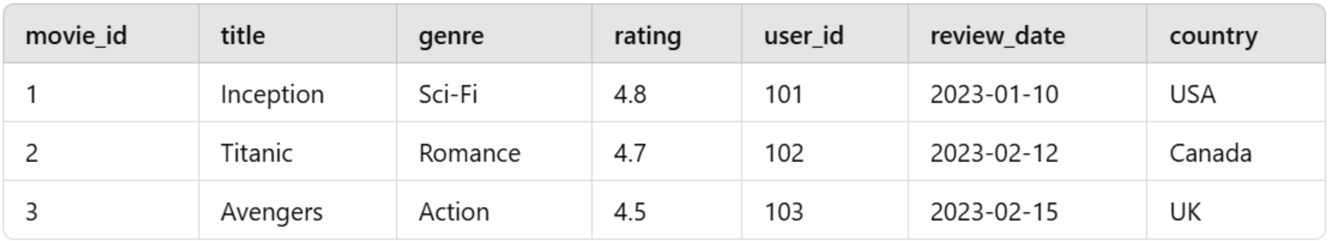


log_id	ip_address	url	status_code	response_time	log_date
1	192.168.1.1	/home	200	120ms	2024-01-01
2	192.168.1.2	/product	404	60ms	2024-01-01
3	192.168.1.3	/checkout	500	200ms	2024-01-02

   a) Create a Hive table for log data. Load Data from HDFS
   b) Find the Most Visited URLs
   c) Detect Peak Traffic Hours
   d) Find 404 Errors

2. Analyze movie ratings, identify top movies, and understand user behavior using **IMDb-style data**.

   Dataset: movie_ratings.csv

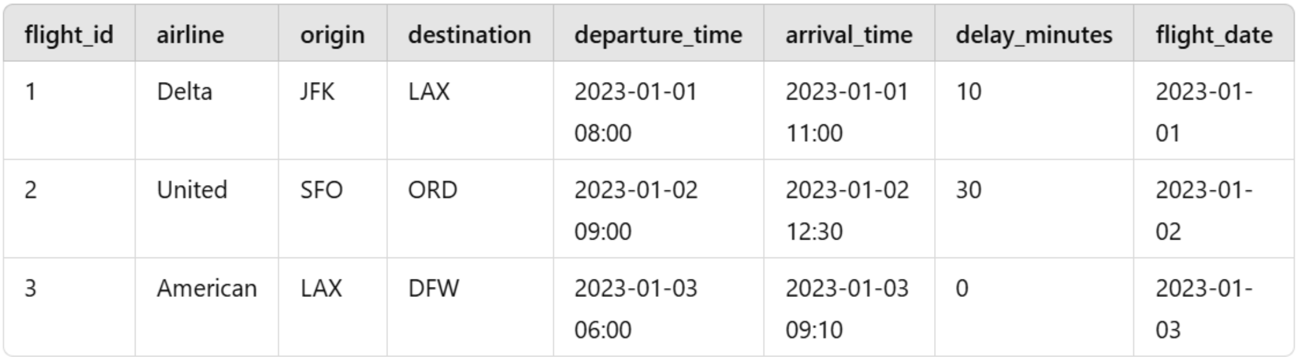


movie_id	title	genre	rating	user_id	review_date	country
1	Inception	Sci-Fi	4.8	101	2023-01-10	USA
2	Titanic	Romance	4.7	102	2023-02-12	Canada
3	Avengers	Action	4.5	103	2023-02-15	UK

   a) Create a Hive table for movie. Load Data from HDFS
   b) Find the Top 5 Highest Rated Movies
   c) Find the Most Popular Genre
   d) Extract Yearly Trends (Extract Year from Review Date)

3. Analyze flight delays, on-time performance, and airport efficiency using real-world airline data.

   Dataset: flight_data.csv



flight_id	airline	origin	destination	departure_time	arrival_time	delay_minutes	flight_date
1	Delta	JFK	LAX	2023-01-01 08:00	2023-01-01 11:00	10	2023-01-01
2	United	SFO	ORD	2023-01-02 09:00	2023-01-02 12:30	30	2023-01-02
3	American	LAX	DFW	2023-01-03 06:00	2023-01-03 09:10	0	2023-01-03

a) Create a Hive table for flight data. Load Data from HDFS

b) Find the Airline with the Most Delays

c) Identify Airports with the Most Delayed Departures

d) Calculate Average Delay per Airline