Exp.No: 2

RUN A BASIC WORD COUNT MAP REDUCE PROGRAM TO UNDERSTAND MAP REDUCE PARADIGM

AIM:

To run a basic Word Count MapReduce program.

PROCEDURE:

Step 1: Create Data File:

Create a file named "word_count_data.txt" and populate it with text data that you wish to analyse.

Login with your hadoop user.

nano word_count.txt

Output: Type the below content in word_count.txt

Step 2: Mapper Logic - mapper.py:

Create a file named "mapper.py" to implement the logic for the mapper. The mapper will read input data from STDIN, split lines into words, and output each word with its count.

```
nano mapper.py
# Copy and paste the mapper.py code
```

```
#!/usr/bin/env python3
# import sys because we need to read and write data to STDIN and STDOUT
#!/usr/bin/python3
import sys
for line in sys.stdin:
    line = line.strip()    # remove leading and trailing whitespace
    words = line.split()   # split the line into words
    for word in words:

     print( '%s\t%s' % (word, 1))
```

.

Step 3: Reducer Logic - reducer.py:

Create a file named "reducer.py" to implement the logic for the reducer. The reducer will aggregate the occurrences of each word and generate the final output.

```
nano reducer.py
# Copy and paste the reducer.py code
```

reducer.py

#!/usr/bin/python3 from operator import itemgetter import sys current_word = None current_count = 0 word = None for line in sys.stdin: line = line.strip() word, count = line.split('\t', 1) try:

count = int(count) except ValueError: continue if current_word == word: current_count += count else:

if current_word:

print('%s\t%s' % (current_word, current_count)) current_count = count current_word = word if current_word == word: print('%s\t%s' % (current_word, current_count))

Step 4: Prepare Hadoop Environment:

Start the Hadoop daemons and create a directory in HDFS to store your data.

start-all.sh hdfsdfs -mkdir /word_count_in_python hdfsdfs -copyFromLocal /path/to/word_count.txt/word_count_in_python

Step 6: Make Python Files Executable:

Give executable permissions to your mapper.py and reducer.py files.

chmod 777 mapper.py reducer.py

Step 7: Run Word Count using Hadoop Streaming:

Download the latest hadoop-streaming jar file and place it in a location you can easily access.

Then run the Word Count program using Hadoop Streaming.

hadoop jar /path/to/hadoop-streaming-3.3.6.jar \ -input /word_count_in_python/word_count_data.txt \

-output /word_count_in_python/new_output \

-mapper /path/to/mapper.py \

-reducer /path/to/reducer.py

Step 8: Check Output:

Check the output of the Word Count program in the specified HDFS output directory.

hdfs dfs -cat /word_count_in_python/new_output/part-00000

```
The general command line syntax is:
command [genericOptions] [commandOptions]

user@Ubuntu:~$ hdfs dfs -cat //WordCount/Output/part-r-00000
2024-09-20 13:25:34,642 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
cat: No FileSystem for scheme "null"
user@Ubuntu:~$ hdfs dfs -cat /WordCount/Output/part-r-00000
2024-09-20 13:25:55,994 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Hl      1
am      1
are     2
fine    2
hi      1
how     1
i       1
you     1
user@Ubuntu:~$
```

Result:

Thus, the program for basic Word Count Map Reduce has been executed successfully.

Exp.No.: 3 **Map Reduce program to process a weather dataset**

AIM:

To implement MapReduce program to process a weather dataset.

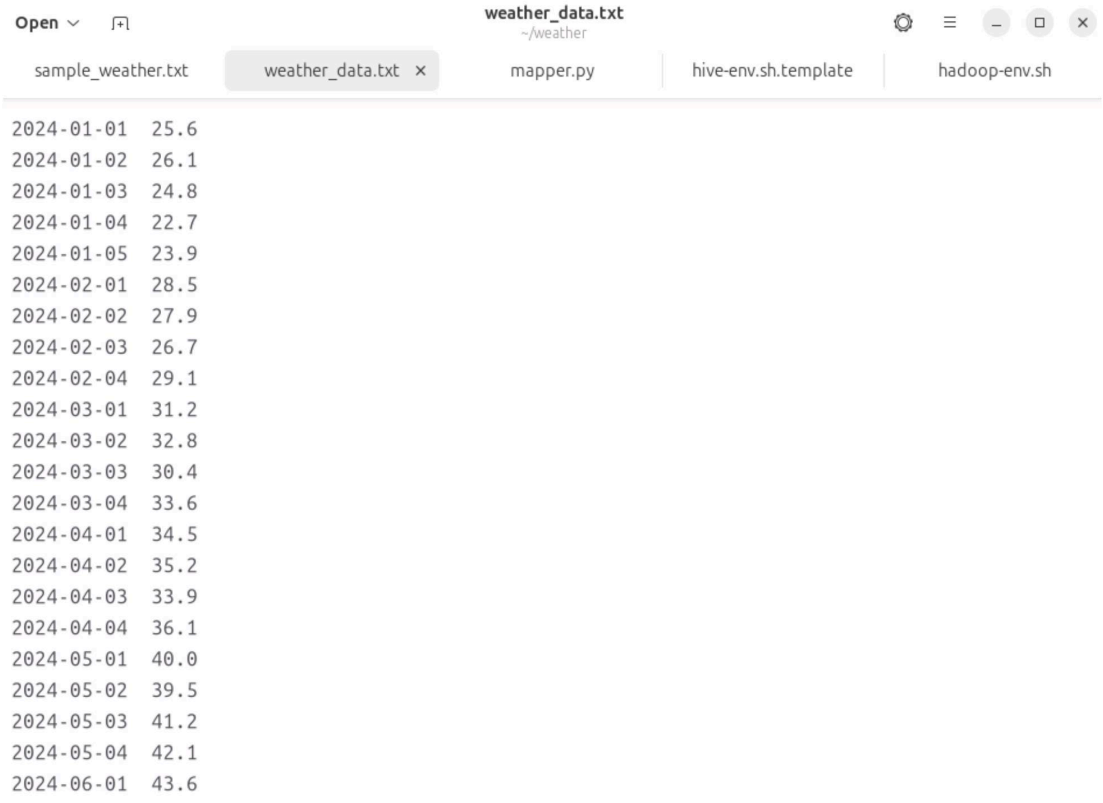
Procedure:

Step 1: Create Data File:

Create a file named "word_count_data.txt" and populate it with text data that you wish to analyse. Login with your hadoop user.

Download the dataset (weather data)

Output:



Step 2: Mapper Logic - mapper.py:

Create a file named "mapper.py" to implement the logic for the mapper. The mapper will read input data from STDIN, split lines into words, and output each word with its count.

```
nano mapper.py
# Copy and paste the mapper.py code
```

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

import sys

# input comes from STDIN (standard input)
# the mapper will get daily max temperature and group it by month. so output will be
(month,dailymax_temperature)
```

```
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()     # split
the line into words     words =
line.split()
    #See the README hosted on the weather website  which help us understand how each
position represents a column     month = line[10:12]     daily_max = line[38:45]     daily_max
= daily_max.strip()
    # increase counters     for
word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be go through the shuffle proess and then
        # be the input for the Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; month and daily max temperature as output
print ('%s\t%s' % (month ,daily_max))
```

.

Step 3: Reducer Logic - reducer.py:

Create a file named "reducer.py" to implement the logic for the reducer. The reducer will aggregate the occurrences of each word and generate the final output.

```
nano reducer.py
# Copy and paste the reducer.py code
```

reducer.py

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

from operator import itemgetter import sys
#reducer will get the input from stdid which will be a collection of key, value(Key=month , value=
daily max temperature)
#reducer logic: will get all the daily max temperature for a month and find max temperature for the
month
#shuffle will ensure that key are sorted(month)
current_month = None
current_max = 0 month =
None

# input comes from STDIN for
line in sys.stdin:
    # remove leading and trailing whitespace     line
= line.strip()
    # parse the input we got from mapper.py     month,
daily_max = line.split('\t', 1)

    # convert daily_max (currently a string) to float     try:
```

```
        daily_max = float(daily_max)     except
ValueError:
        # daily_max was not a number, so silently
        # ignore/discard this line
continue

    # this IF-switch only works because Hadoop shuffle process sorts map output
    # by key (here: month) before it is passed to the reducer
if current_month == month:          if daily_max > current_max:
current_max = daily_max     else:         if current_month:
            # write result to STDOUT
            print ('%s\t%s' % (current_month, current_max))
current_max = daily_max
        current_month = month

# output of the last month if current_month == month:
print ('%s\t%s' % (current_month, current_max))
```

Step 4: Prepare Hadoop Environment:

Start the Hadoop daemons and create a directory in HDFS to store your data.

```
start-all.sh
```

Step 6: Make Python Files Executable:

Give executable permissions to your mapper.py and reducer.py files.

```
chmod 777 mapper.py reducer.py
```

```
user@Ubuntu:~$ chmod 777 mapper.py reducer.py
user@Ubuntu:~$ hadoop fs -mkdir -p /weatherdata
2024-09-23 08:17:58,220 WARN util.NativeCodeLoader: Unable to load native-hadoop
 library for your platform... using builtin-java classes where applicable
user@Ubuntu:~$ hdfs dfs -ls /weatherdata
2024-09-23 08:18:31,572 WARN util.NativeCodeLoader: Unable to load native-hadoop
 library for your platform... using builtin-java classes where applicable
Found 2 items
-rw-r--r--   1 user supergroup      51647 2024-09-01 20:07 /weatherdata/dataset.
txt
drwxr-xr-x   - user supergroup          0 2024-09-01 20:40 /weatherdata/output
user@Ubuntu:~$
```

Step 7: Run the program using Hadoop Streaming:

Download the latest hadoop-streaming jar file and place it in a location you can easily access.

Then run the program using Hadoop Streaming.

```
hadoop fs -mkdir -p /weatherdata

hadoop fs -copyFromLocal /home/sx/Downloads/dataset.txt /weatherdata
```

```
hdfs dfs -ls /weatherdata

hadoop jar /home/sx/hadoop-3.2.3/share/hadoop/tools/lib/hadoop-streaming-3.2.3.jar \
  -input /weatherdata/dataset.txt \
  -output /weatherdata/output \
  -file "/home/sx/Downloads/mapper.py" \
  -mapper "python3 mapper.py" \
  -file "/home/sx/Downloads/reducer.py" \
  -reducer "python3 reducer.py"

hdfs dfs -text /weatherdata/output/* > /home/sx/Downloads/outputfile.txt
```

Step 8: Check Output:

Check the output of the program in the specified HDFS output directory.

```
hdfs dfs -text /weatherdata/output/* > /home/sx/Downloads/output/ /part-00000
```

```
user@Ubuntu:~$ hdfs dfs -cat /weatherdata/output/part-00000
2024-09-23 08:14:37,641 WARN util.NativeCodeLoader: Unable to load native-hadoop
 library for your platform... using builtin-java classes where applicable
01      -2.9
02      9.3
03      10.4
04      15.7
05      20.1
06      28.3
07      28.2
08      28.4
user@Ubuntu:~$
```

After copy and paste the above output in your local file give the below command to remove the directory from hdfs : hadoop fs -rm -r /weatherdata/output

Result:

Thus, the program for weather dataset using Map Reduce has been executed successfully.

Exp.No.: 4 **Create UDF in PIG**

Step-by-step installation of Apache Pig on Hadoop cluster on Ubuntu Pre-requisite:

· Ubuntu 16.04 or higher version running (I have installed Ubuntu on Oracle VM (Virtual Machine) VirtualBox),

· Run Hadoop on ubuntu (I have installed Hadoop 3.2.1 on Ubuntu 16.04). You may refer to my blog "How to install Hadoop installation" click here for Hadoop installation).

Pig installation steps

Step 1: Login into Ubuntu

Step 2: Go to https://pig.apache.org/releases.html and copy the path of the latest version of pig that you want to install. Run the following comment to download Apache Pig in Ubuntu:

$ wget https://dlcdn.apache.org/pig/pig-0.16.0/pig-0.16.0.tar.gz

Step 3: To untar pig-0.16.0.tar.gz file run the following command:

$ tar xvzf pig-0.16.0.tar.gz

Step 4: To create a pig folder and move pig-0.16.0 to the pig folder, execute the following command:

$ sudo mv /home/hadoop/pig-0.16.0 /home/hadoop/pig

Step 5: Now open the .bashrc file to edit the path and variables/settings for pig. Run the following command:

$ sudo nano .bashrc

Add the below given to .bashrc file at the end and save the file.

#PIG settingsexport PIG_HOME=/home/hdoop/pigexport PATH=$PATH:$PIG_HOME/binexport PIG_CLASSPATH=$PIG_HOME/conf:$HADOOP_INSTALL/etc/hadoop/export PIG_CONF_DIR=$PIG_HOME/confexport JAVA_HOME=/usr/lib/jvm/java-8-openjdkamd64export PIG_CLASSPATH=$PIG_CONF_DIR:$PATH#PIG setting ends

```
 GNU nano 7.2                              .bashrc

export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_HOME=/home/hadoop/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"


# PIG settings
export PIG_HOME=/home/hadoop/pig
export PATH=$PATH:$PIG_HOME/bin
export PIG_CLASSPATH=$PIG_HOME/conf:$HADOOP_INSTALL/etc/hadoop
export PIG_CONF_DIR=$PIG_HOME/conf
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export PIG_CLASSPATH=$PIG_CONF_DIR:$PIG_CLASSPATH
# PIG settings end
```

Step 6: Run the following command to make the changes effective in the .bashrc file:

$ source .bashrc

Step 7: To start all Hadoop daemons, navigate to the hadoop-3.2.1/sbin folder and run the following commands:

$ ./start-dfs.sh$ ./start-yarn$ jps

```
user@Ubuntu:~$ nano .bashrc
user@Ubuntu:~$ source ~/.bashrc
user@Ubuntu:~$ jps
5889 Jps
4387 ResourceManager
3941 DataNode
4536 NodeManager
4120 SecondaryNameNode
3823 NameNode
user@Ubuntu:~$
```

Step 8: Now you can launch pig by executing the following command: $ pig

```
user@Ubuntu:~$ pig
2024-09-23 10:18:38,911 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
2024-09-23 10:18:38,913 INFO pig.ExecTypeProvider: Trying ExecType : MAPREDUCE
2024-09-23 10:18:38,913 INFO pig.ExecTypeProvider: Picked MAPREDUCE as the ExecType
2024-09-23 10:18:39,047 [main] INFO  org.apache.pig.Main - Apache Pig version 0.17.0 (
r1797386) compiled Jun 02 2017, 15:41:58
2024-09-23 10:18:39,047 [main] INFO  org.apache.pig.Main - Logging error messages to:
/home/sai/pig_1727066919036.log
2024-09-23 10:18:39,131 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup f
ile /home/sai/.pigbootup not found
2024-09-23 10:18:39,537 [main] WARN  org.apache.hadoop.util.NativeCodeLoader - Unable
to load native-hadoop library for your platform... using builtin-java classes where ap
plicable
2024-09-23 10:18:39,554 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation
- mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2024-09-23 10:18:39,554 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HEx
ecutionEngine - Connecting to hadoop file system at: hdfs://localhost:9000
2024-09-23 10:18:40,411 [main] INFO  org.apache.pig.PigServer - Pig Script ID for the
session: PIG-default-158ed3f2-9071-4282-ae3a-6211c47c9b42
2024-09-23 10:18:40,411 [main] WARN  org.apache.pig.PigServer - ATS is disabled since
yarn.timeline-service.enabled set to false
grunt>
```

Step 9: Now you are in pig and can perform your desired tasks on pig. You can come out of the pig by the quit command:

> quit;

Reg.No:210701277

CREATE USER DEFINED FUNCTION(UDF)

Aim :

To create User Define Function in Apache Pig and execute it on map reduce.

PROCEDURE:

Create a sample text file

hadoop@Ubuntu:~/Documents$ nano sample.txt

Paste the below content to sample.txt

1,John

2,Jane

3,Joe

4,Emma

hadoop@Ubuntu:~/Documents$ hadoop fs -put sample.txt /home/hadoop/piginput/

Create PIG File

hadoop@Ubuntu:~/Documents$ nano demo_pig.pig

paste the below the content to demo_pig.pig

-- Load the data from HDFS

data = LOAD '/home/hadoop/piginput/sample.txt' USING PigStorage(',') AS (id:int>

-- Dump the data to check if it was loaded correctly

DUMP data;

---------- **Run the above file**

hadoop@Ubuntu:~/Documents$ pig demo_pig.pig

```
user@Ubuntu:~$ hadoop fs -ls /piginput
2024-09-23 10:30:30,590 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Found 1 items
-rw-r--r--   1 user supergroup         27 2024-09-01 22:11 /piginput/sample.txt
user@Ubuntu:~$ nano demo_pig.pig
user@Ubuntu:~$ pig demo_pig.pig
2024-09-23 10:32:30,584 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
2024-09-23 10:32:30,587 INFO pig.ExecTypeProvider: Trying ExecType : MAPREDUCE
2024-09-23 10:32:30,587 INFO pig.ExecTypeProvider: Picked MAPREDUCE as the ExecType
2024-09-23 10:32:30,649 [main] INFO  org.apache.pig.Main - Apache Pig version 0.17.0 (
r1797386) compiled Jun 02 2017, 15:41:58
```

Create udf file an save as uppercase_udf.py

uppercase_udf.py

```
def uppercase(text): return text.upper()

if __name__ == "__main__":

import sys for line in
sys.stdin:
        line = line.strip() result =
        uppercase(line)
        print(result)
```

Create the udfs folder on hadoop

hadoop@Ubuntu:~/Documents$ hadoop fs -mkdir /home/hadoop/udfs

put the upppercase_udf.py in to the abv folder

hadoop@Ubuntu:~/Documents$ hdfs dfs -put uppercase_udf.py /home/hadoop/udfs/

hadoop@Ubuntu:~/Documents$ nano udf_example.pig copy and paste the below content on udf_example.pig

```
-- Register the Python UDF script

REGISTER 'hdfs:///home/hadoop/udfs/uppercase_udf.py' USING jython AS udf;


-- Load some data

data = LOAD 'hdfs:///home/hadoop/sample.txt' AS (text:chararray);


-- Use the Python UDF

uppercased_data = FOREACH data GENERATE udf.uppercase(text) AS uppercase_text;


-- Store the result

STORE uppercased_data INTO 'hdfs:///home/hadoop/pig_output_data';
```

---

place sample.txt file on hadoop

hadoop@Ubuntu:~/Documents$ hadoop fs -put sample.txt /home/hadoop/

To Run the pig file

hadoop@Ubuntu:~/Documents$ pig -f udf_example.pig

```
user@Ubuntu:~$ pig -f udf_example.pig
2024-09-23 10:35:53,534 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
2024-09-23 10:35:53,535 INFO pig.ExecTypeProvider: Trying ExecType : MAPREDUCE
2024-09-23 10:35:53,535 INFO pig.ExecTypeProvider: Picked MAPREDUCE as the ExecType
2024-09-23 10:35:53,580 [main] INFO  org.apache.pig.Main - Apache Pig version 0.17.0 (
r1797386) compiled Jun 02 2017, 15:41:58
```

To check the output file is created

hadoop@Ubuntu:~/Documents$ hdfs dfs -ls /home/hadoop/pig_output_data

Found 2 items

If you need to examine the files in the output folder, use:

To view the output

hadoop@Ubuntu:~/Documents$ hdfs dfs -cat /home/hadoop/pig_output_data/part-m00000

```
user@Ubuntu:~$ hdfs dfs -cat /pig_output_data/part-m-00000
2024-09-23 10:17:46,126 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
1,JOHN
2,JANE
3,JOE
4,EMMA
```

Result:

Thus the program to create User Define Function in Apache Pig and execute it on map reduce has been done successfully.

Exp.No.:5 **Installation of Hive on Ubuntu**

Aim:

To Download and install Hive, Understanding Startup scripts, Configuration files.

Procedure:

Step 1: Download and extract it

Download the Apache hive and extract it use tar, the commands given below:

$wgethttps://downloads.apache.org/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz

```
hadoop@priyav-VirtualBox:~$ wget https://archive.apache.org/dist/hive/hive-3.1.2
/apache-hive-3.1.2-bin.tar.gz
--2024-09-02 12:26:15--  https://archive.apache.org/dist/hive/hive-3.1.2/apache-
hive-3.1.2-bin.tar.gz
Resolving archive.apache.org (archive.apache.org)... 65.108.204.189, 2a01:4f9:1a
:a084::2
Connecting to archive.apache.org (archive.apache.org)|65.108.204.189|:443... con
nected.
HTTP request sent, awaiting response... 200 OK
Length: 278813748 (266M) [application/x-gzip]
Saving to: ‘apache-hive-3.1.2-bin.tar.gz’

apache-hive-3.1.2-b 100%[===================>] 265.90M  1.20MB/s    in 2m 57s

2024-09-02 12:29:13 (1.50 MB/s) - ‘apache-hive-3.1.2-bin.tar.gz’ saved [27881374
8/278813748]
```

$ tar –xvf apache-hive-3.1.2-bin.tar.gz

```
hadoop@priyav-VirtualBox:~$ tar -xvf apache-hive-3.1.2-bin.tar.gz
apache-hive-3.1.2-bin/LICENSE
apache-hive-3.1.2-bin/NOTICE
apache-hive-3.1.2-bin/RELEASE_NOTES.txt
apache-hive-3.1.2-bin/binary-package-licenses/asm-LICENSE
apache-hive-3.1.2-bin/binary-package-licenses/com.google.protobuf-LICENSE
apache-hive-3.1.2-bin/binary-package-licenses/com.ibm.icu.icu4j-LICENSE
apache-hive-3.1.2-bin/binary-package-licenses/com.sun.jersey-LICENSE
apache-hive-3.1.2-bin/binary-package-licenses/com.thoughtworks.paranamer-LICENSE
apache-hive-3.1.2-bin/binary-package-licenses/javax.transaction.transaction-api-
LICENSE
apache-hive-3.1.2-bin/binary-package-licenses/javolution-LICENSE
apache-hive-3.1.2-bin/binary-package-licenses/jline-LICENSE
apache-hive-3.1.2-bin/binary-package-licenses/NOTICE
apache-hive-3.1.2-bin/binary-package-licenses/org.abego.treelayout.core-LICENSE
apache-hive-3.1.2-bin/binary-package-licenses/org.antlr-LICENSE
apache-hive-3.1.2-bin/binary-package-licenses/org.antlr.antlr4-LICENSE
```

Step 2: Place different configuration properties in Apache Hive

In this step, we are going to do two things ○ Placing

Hive Home path in bashrc file

$*nano .bashrc*

And append the below lines in it

```
#HIVE settings
export HIVE_HOME=/home/hadoop/apache-hive-3.1.2
export PATH=$PATH:$HIVE_HOME/bin
#HIVE settings end
```

2. Exporting **Hadoop path in Hive-config.sh** (To communicate with the Hadoop eco system we are defining Hadoop Home path in hive config field) **Open the hiveconfig.sh as shown in below** *$cd apache-hive-3.1.2-bin/bin*
$cp hive-env.sh.template hive-env.sh
$nano hive-env.sh
Append the below commands on it export HADOOP_HOME=/home/Hadoop/Hadoop
export HIVE_CONF_DIR=/home/Hadoop/apache-hive-3.1.2/conf

```
# Set HADOOP_HOME to point to a specific hadoop install directory
# HADOOP_HOME=${bin}/../../hadoop
export HADOOP_HOME=/home/hadoop/hadoop


# Hive Configuration Directory can be controlled by:
# export HIVE_CONF_DIR=
export HIVE_CONF_DIR=/home/hadoop/apache-hive-3.1.2-bin/conf
# Folder containing extra libraries required for hive compilation/execution can be controlled by:
```

Step 3: Install mysql

1. Install mysql in Ubuntu by running this command:

$sudo apt update
$sudo apt install mysql-server

2. *Alter username and password for MySQLby running below commands:*
$sudomysql

Pops command line interface for MySQLand run the below SQL queries to change username and set password
mysql> SELECT user, host, plugin FROM mysql.user WHERE user = 'root';
mysql> ALTER USER 'root'@'localhost' IDENTIFIED WITH 'mysql_native_password' BY 'your_new_password';
mysql> FLUSH PRIVILEGES;

Step 4:Config hive-site.xml

Config the hive-site.xml by appending this xml code and change the username and password according to your MySQL.
$cd apache-hive-3.1.2-bin/bin
$cp hive-default.xml.template hive-site.xml
$nano hive-site.xml
Append these lines into it
Replace root as your username of MySQL
Replaceyour_new_password as with your password of MySQL
<configuration>
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:mysql://localhost/metastore?createDatabaseIfNotExist=true</value>
</property>

```
<property>
<name>javax.jdo.option.ConnectionDriverName</name>
<value>com.mysql.cj.jdbc.Driver</value>
</property>

<property>
<name>javax.jdo.option.ConnectionUserName</name>
<value>root</value>
</property>

<property>
<name>javax.jdo.option.ConnectionPassword</name>
<value>your_new_password</value>
</property>

<property>
<name>datanucleus.autoCreateSchema</name>
<value>true</value>
</property>

<property>
<name>datanucleus.fixedDatastore</name>
<value>true</value>
</property>

<property>
<name>datanucleus.autoCreateTables</name>
<value>True</value>
</property>

</configuration>
```

Step 5: Setup MySQL java connector:

First, you'll need to download the MySQL Connector/J, which is the JDBC driver for MySQL. You can download it from the below link
https://drive.google.com/file/d/1QFhB7Kvcat7a4LzDRe6GcmZva1yAxKz/view?usp=drive_link
Copy the downloaded MySQL Connector/J JAR file to the Hive library directory. By default, the Hive library directory is usually located at*/path/to/apache-hive-3.1.2/lib/*on Ubuntu. Use the following command to copy the JAR file:
$sudo cp /path/to/mysql-connector-java-8.0.15.jar /path/to/apache-hive-3.1.2/lib/ Replace /path/to/ with the actual path to the JAR file.

Step 6:Initialize the Hive Metastore Schema:

Run the following command to initialize the Hive metastore schema:
$$HIVE_HOME/bin/schematool -initSchema -dbTypemysql

```
hadoop@priyav-VirtualBox:~$ hdfs dfs -chmod g+w /tmp
hadoop@priyav-VirtualBox:~$ hdfs dfs -mkdir -p /user/hive/warehouse
hadoop@priyav-VirtualBox:~$ hdfs dfs -chmod g+w /user/hive/warehouse
hadoop@priyav-VirtualBox:~$ schematool -initSchema -dbType derby
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hadoop/apache-hive-3.1.2/lib/log4j-slf4j-impl-2.10.
0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hadoop/hadoop/share/hadoop/common/lib/slf4j-reload4
j-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]
Metastore connection URL:        jdbc:derby:;databaseName=metastore_db;create=true
Metastore Connection Driver :    org.apache.derby.jdbc.EmbeddedDriver
Metastore connection User:       APP
```

Step 7: Start hive:

You can test Hive by running the Hive shell: Copy code hive You should be able to run Hive queries, and metadata will be stored in your MySQL database. *$hive*

```
user@Ubuntu:~$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/sai/Desktop/hive/apache-hive-3.1.2-bin/lib/log
4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/sai/hadoop-3.4.0/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 = 0eb28a97-0716-440e-a747-ac1f95d523cd

Logging initialized using configuration in jar:file:/home/sai/Desktop/hive/apache-hive
-3.1.2-bin/lib/hive-common-3.1.2.jar!/hive-log4j2.properties Async: true
Hive Session ID = fdfd86b0-1756-4526-9f14-b8c518b83b6c
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Co
nsider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases
.
hive>
```

Result:

Thus, the Apache Hive installation is completed successfully on Ubuntu.

Exp.No.: 5a

Design and test various schema models to optimize data storage and retrieval Using Hive

Aim:

To Design and test various schema models to optimize data storage and retrieval Using Hbase.

Procedure:

Step 1: Start Hive

Open a terminal and start Hive by running:

$hive

Step 2: Create a Database

Create a new database in Hive: hive>CREATE DATABASE financials;

```
hive> CREATE DATABASE financials;
OK
Time taken: 0.063 seconds
```

***Step 3: Use the Database**:*

Switch to the newly created database: hive>use financials;

```
hive> use financials;
OK
Time taken: 0.57 seconds
```

Step 4: Create a Table:

Create a simple table in your database:

hive>CREATE TABLE finance_table(id INT, name STRING);

```
hive> CREATE TABLE finance_table( id INT, name STRING );
OK
Time taken: 2.013 seconds
```

Step 5: Load Sample Data:

You can insert sample data into the table:

hive>INSERT INTO finance_tableVALUES (1, 'Alice'), (2, 'Bob'), (3, 'Charlie');

```
hive> INSERT INTO finance_table VALUES
    > (1,'Alice')
    > ,
    > (2,'Bob'),
    > (3,'Charlie');
Query ID = hadoop_20240911171244_304f3e60-6937-434c-acb2-d71be2797182
Total jobs = 3
Launching Job 1 out of 3
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>
Job running in-process (local Hadoop)
2024-09-11 17:12:54,138 Stage-1 map = 0%,  reduce = 0%
2024-09-11 17:12:57,541 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local1825573535_0001
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/financials.db/finance_table/.hive-staging_hive_2024-
9-11_17-12-44_558_5675160086864575725-1/-ext-10000
Loading data to table financials.finance_table
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 0 HDFS Write: 208 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Time taken: 13.965 seconds
```

Step 6: Query Your Data

Use SQL-like queries to retrieve data from your table:
hive>CREATE VIEW myview AS SELECT name, id FROM finance_table;

```
hive> CREATE VIEW myview AS SELECT name, id FROM finance_table;
OK
Time taken: 0.244 seconds
```

Step 7: View the data:

*To see the data in the view, you would need to query the view hive>SELECT*FROM myview;*

```
hive> SELECT*FROM myview;
OK
Alice   1
Bob     2
Charlie 3
Time taken: 0.22 seconds, Fetched: 3 row(s)
```

Step 8: Describe a Table:

You can describe the structure of a table using the DESCRIBE command:
hive>DESCRIBE finance_table;

Step 9: Alter a Table:

```
hive> DESCRIBE finance_table;
OK
id                      int
name                    string
age                     int
Time taken: 0.729 seconds, Fetched: 3 row(s)
```

You can alter the table structure by adding a new column: hive>ALTER TABLE finance_table ADD COLUMNS (age INT);

```
hive> ALTER TABLE finance_table ADD COLUMNS (age INT);
OK
Time taken: 0.188 seconds
```

Step 10: Quit Hive:

To exit the Hive CLI, simply type: hive>quit;

```
hive> quit;
user@Ubuntu:~$
```

Result:

Thus, the usage of various commands in Hive has been successfully completed.

Ex.No.: 6

Import a JASON file from the command line. Apply the following actions with the data present in the JASON file where, projection, aggregation, remove, count, limit, skip and sort

AIM:

To import a JASON file from the command line and apply the following actions with the data present in the JASON file where, projection, aggregation, remove, count, limit, skip and sort.

PROCEDURE:

1. Download and install MongoDB:

1. Visit the MongoDB Download Center and choose the version suitable for your operating system (Windows).
2. Select MSI as the package, and click "Download".
3. Run the downloaded MSI installer. Follow the prompts in the installation wizard.
4. Choose "Complete" for the setup type to install all components.
5. Select the option to install MongoDB as a service (recommended), which allows MongoDB to start automatically with your system.
6. Check the version installed by using the command
   $ mongod –version

```
C:\Users\tamil>mongod --version
db version v7.0.14
Build Info: {
    "version": "7.0.14",
    "gitVersion": "ce59cfc6a3c5e5c067dca0d30697edd68d4f5188",
    "modules": [],
    "allocator": "tcmalloc",
    "environment": {
        "distmod": "windows",
        "distarch": "x86_64",
        "target_arch": "x86_64"
    }
}
```

2. Create a sample people.json file with the following content:

```
[{"name": "Alice",
   "age": 30,
   "city": "New York"},
 {"name": "Bob",
   "age": 25,
   "city": "Los Angeles" },
 {"name": "Charlie",
   "age": 35,
   "city": "Chicago"},
 {"name": "David",
   "age": 28,
   "city": "Houston"},
 {"name": "Eve",
   "age": 22,
   "city": "Phoenix"}]
```

Import the JSON File

To import the JSON file into your MongoDB database, open your terminal or command prompt and use the mongoimport command:

>mongoimport --db mydb --collection people --file path_to_your_json/people.json –jsonArray

3. Download and install MongoShell:

```
C:\Users\tamil>mongoimport --db mydb --collection people --file "C:\Users\tamil\OneDrive\Desktop\people.json" --jsonArray
2024-09-13T11:53:19.890+0530    connected to: mongodb://localhost/
2024-09-13T11:53:19.937+0530    5 document(s) imported successfully. 0 document(s) failed to import.
```

1. Visit the MongoDB Shell https://www.mongodb.com/try/download/shell Download Page.
2. Select your operating system as Windows and download the MSI package. Run the downloaded MSI installer.
3. Follow the installation prompts to complete the installation.
4. Ensure the option to add MongoDB Shell to your PATH is checked during installation.
5. To verify, open Command Prompt or PowerShell and type the following command to check if mongosh is installed correctly:

   $ mongosh --version
6. If installed correctly, it will display the version of mongosh.
7. To start MongoDB Shell open Command Prompt Type mongosh to start the MongoDB Shell. It will connect to the default MongoDB server (localhost:27017).

4. Create and Switch to the Database:

```
C:\Users\tamil>mongosh
Current Mongosh Log ID: 66e3d532e78e7b0457c73bf7
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2
.3.1
Using MongoDB:          7.0.14
Using Mongosh:          2.3.1

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2024-09-11T23:09:26.958+05:30: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
------

test>
```

Use the use command to create and switch to the new database. Then, switch to your database and check the collection:

test> use mydb

mydb> show collections

```
test> show dbs;
admin   40.00 KiB
config  72.00 KiB
local   72.00 KiB
mydb     8.00 KiB
test> use mydb;
switched to db mydb
mydb>
```

```
mydb> show collections;
demo
people
mydb>
```

>db.people.find().pretty()

```
mongosh mongodb://127.0.(
mydb> db.people.find().pretty()
[
  {
    _id: ObjectId('66e3da575d4c1f6e0314811f'),
    name: 'Charlie',
    age: 35,
    city: 'Chicago'
  },
  {
    _id: ObjectId('66e3da575d4c1f6e03148120'),
    name: 'Eve',
    age: 22,
    city: 'Phoenix'
  },
  {
    _id: ObjectId('66e3da575d4c1f6e03148121'),
    name: 'Alice',
    age: 30,
    city: 'New York'
  },
  {
    _id: ObjectId('66e3da575d4c1f6e03148122'),
    name: 'David',
    age: 28,
    city: 'Houston'
  },
  {
    _id: ObjectId('66e3da575d4c1f6e03148123'),
    name: 'Bob',
    age: 25,
    city: 'Los Angeles'
  }
]
mydb>
```

Step 4:

Perform Actions on the JSON Data

Now, we will apply the following operations: where, projection, aggregation, remove, count, limit, skip, and sort.

1. **Where (Filter):** Find records where the city is "New York":
   >db.people.find({ city: "New York" })

```
mydb> db.people.find({ city: "New York" })
[
  {
    _id: ObjectId('66e3da575d4c1f6e03148121'),
    name: 'Alice',
    age: 30,
    city: 'New York'
  }
]
```

2. **Projection:** Select only the name and age fields from the records:
   >db.people.find({}, { name: 1, age: 1, _id: 0 })

```
mydb> db.people.find({}, { name: 1, age: 1, _id: 0 })
[
  { name: 'Charlie', age: 35 },
  { name: 'Eve', age: 22 },
  { name: 'Alice', age: 30 },
  { name: 'David', age: 28 },
  { name: 'Bob', age: 25 }
]
```

3. **Aggregation:** Calculate the average age of all people in the collection:
   >db.people.aggregate([

{ $group: { _id: null, averageAge: { $avg: "$age" } } }
])

```
mydb> db.people.aggregate([ { $group: { _id: null, averageAge: { $avg: "$age" } } }] )
[ { _id: null, averageAge: 28 } ]
mydb>
```

4. **Remove:** Delete people who live in "Chicago":
   >db.people.deleteMany({ city: "Chicago" })

```
mydb> db.people.deleteMany({ city: "Chicago" })
{ acknowledged: true, deletedCount: 1 }
```

5. **Count:** Count the total number of people in the collection:
   >db.people.countDocuments({})

```
mydb> db.people.countDocuments({})
4
mydb>
```

6. **Limit:** Limit the results to 2 records:
   >db.people.find().limit(2)

```
mydb> db.people.find().limit(2)
[
  {
    _id: ObjectId('66e3da575d4c1f6e03148120'),
    name: 'Eve',
    age: 22,
    city: 'Phoenix'
  },
  {
    _id: ObjectId('66e3da575d4c1f6e03148121'),
    name: 'Alice',
    age: 30,
    city: 'New York'
  }
]
```

7. **Skip:** Skip the first record and show the rest:
   >db.people.find().skip(1)

```
mydb> db.people.find().skip(1)
[
  {
    _id: ObjectId('66e3da575d4c1f6e03148121'),
    name: 'Alice',
    age: 30,
    city: 'New York'
  },
  {
    _id: ObjectId('66e3da575d4c1f6e03148122'),
    name: 'David',
    age: 28,
    city: 'Houston'
  },
  {
    _id: ObjectId('66e3da575d4c1f6e03148123'),
    name: 'Bob',
    age: 25,
    city: 'Los Angeles'
  }
]
```

8. **Sort:** Sort the records by age in ascending order:
   >db.people.find().sort({ age: 1 })

```
mydb> db.people.find().sort({ age: 1 })
[
  {
    _id: ObjectId('66e3da575d4c1f6e03148120'),
    name: 'Eve',
    age: 22,
    city: 'Phoenix'
  },
  {
    _id: ObjectId('66e3da575d4c1f6e03148123'),
    name: 'Bob',
    age: 25,
    city: 'Los Angeles'
  },
  {
    _id: ObjectId('66e3da575d4c1f6e03148122'),
    name: 'David',
    age: 28,
    city: 'Houston'
  },
  {
    _id: ObjectId('66e3da575d4c1f6e03148121'),
    name: 'Alice',
    age: 30,
    city: 'New York'
  }
]
```

RESULT:

Thus to import a JASON file from the command line and apply the following actions with the data present in the JASON file where, projection, aggregation, remove, count, limit, skip and sort has been executed and verified successfully.