



**PSGR
Krishnammal College for Women**



Affiliated to Bharathiar University \ Autonomous \ College of Excellence \ Accredited with A++ Grade \ Ranked 9th in NIRF

DEPARTMENT OF COMPUTER SCIENCE (PG)

BIG DATA ANALYTICS LAB – (MCS23P4)

2025-2027



**PSGR
Krishnammal College for Women**



Affiliated to Bharathiar University \ Autonomous \ College of Excellence \ Accredited with A++ Grade \ Ranked 9th in NIRF

DEPARTMENT OF COMPUTER SCIENCE(PG)

BIG DATA ANALYTICS LAB (MCS23P4)

REGISTER NUMBER: _____

Certified that this is a bonafide record work done by _____
of I-MSc Computer Science during the year of 2025-2027.

FACULTY INCHARGE

HEAD OF THE DEPARTMENT

Submitted for the Practical examination held on by _____
at PSGR Krishnammal college for women, Coimbatore.

INTERNAL EXAMINER

EXTERNAL EXAMINER

INDEX

S No.	Date	Topics	Page No	Sign
1		File Management Tasks in Hadoop		
2		Electrical Consumption Using MapReduce		
3		Weather Data Using MapReduce		
4		Count Gender and Hobby Using Pig Script		
5		Count Words Using Pig Script		
6		Internal And External Tables in Hive		
7		Partitions And Buckets in Hive		
8		Built In Functions in Hive		
9		Join Operations in Hive		
10		Create Employee Class in Scala Environment		
11		Check Character Sequence in Scala		
12		Min and Max Values in Map Using Scala		
13		Dataframe and Group by Operations in Scala		
14		Key space In Cassandra Using Java Api		
15		Adding a Column in Cassandra using Java Api		

16		Word Count Using Scala		
17		Frequency Calculation Using MapReduce		
18		Display Chart Using Zeppelin		
19		Interactive Queries Using Zeppelin		
20		Student Grade Analysis Using Scala		

Ex. No:01

Date:

FILE MANAGEMENT TASKS IN HADOOP

AIM

To implement the following file management tasks in Hadoop: Adding files and directories, retrieving files, deleting files.

ALGORITHM

Step 1: Start the process.

Step 2: Set up the Hadoop environment.

Step 3: Navigate to the Hadoop directory using `cd hadoop-2.9.1`.

Step 4: List the created directories using `bin/hadoop dfs -ls /`.

Step 5: Copy a file from the local system to Hadoop using `bin/hadoop dfs -copyFromLocal myfile1.txt sample1.txt`.

Step 6: Copy the file from Hadoop back to the local system with a new name using `bin/hadoop dfs -copyToLocal sample1.txt myfile3.txt`.

Step 7: Duplicate the file within Hadoop with a new name using `bin/hadoop dfs -cp sample1.txt datafile1.txt`.

Step 8: Count the number of files using `bin/hadoop dfs -count /`.

Step 9: Remove a specific file using `bin/hadoop dfs -rm sample1.txt`.

Step 10: Delete an entire directory using `bin/hadoop dfs -rmr /`.

Step 11: Display the contents of a file using `bin/hadoop dfs -cat datafile.txt`.

Step 12: Check file storage usage, including space allocation, using `bin/hadoop dfs -du datafile.txt`.

Step 13: Stop the process.

PROGRAM

.BASHRC

```
export JAVA_HOME=/home/grg/jdk1.8.0_45
export HADOOP_HOME=/home/grg/hadoop-2.9.0
export SPARK_HOME=/home/grg/spark-2.1.1-bin-hadoop2.7
export PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:$HIVE_HOME/bin:$PATH
```

TERMINAL

```
cd hadoop-2.9.1
bin/hadoop namenode -format
sbin/start-all.sh
jps
```

CREATING DIRECTORY

```
bin/hadoop dfs -mkdir /mcs1
bin/hadoop dfs -ls /
```

CREATING 2 TEXT FILES IN HOME

```
cd $home
touch file1.txt
touch file2.txt
```

FILE1.TXT,FILE2.TXT WILL BE CREATED IN HOME

IN FILE1.TXT

HELLO WELCOME TO PSGR

IN TERMINAL:

```
cd hadoop-2.9.1
bin/hadoop fs -put /home/grg/file1.txt /file2.txt
bin/hadoop fs -copyFromLocal /home/grg/file1.txt /file2.txt
bin/hadoop fs -copyToLocal /file2.txt /home/grg/file3.txt
```

```
bin/hadoop fs -cat /file2.txt
```

```
bin/hadoop fs -du /file2.txt
```

```
bin/hadoop fs -rm /file2.txt
```

OUTPUT

```
Activities Terminal Feb 20 11:27 grg@LAB4-1: ~/hadoop-2.9.1

localhost: starting nodemanager, logging to /home/grg/hadoop-2.9.1/logs/yarn-grg-nodemanager-LAB4-1.out
grg@LAB4-1:~/hadoop-2.9.1$ jps
2484 SecondaryNameNode
3064 Jps
2281 DataNode
2636 ResourceManager
2940 NodeManager
2093 NameNode
grg@LAB4-1:~/hadoop-2.9.1$ bin/hadoop dfs -mkdir /mcs1
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
grg@LAB4-1:~/hadoop-2.9.1$ bin/hadoop dfs -ls /
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

Found 3 items
drwxr-xr-x - grg supergroup 0 2026-02-16 08:59 /home
drwxr-xr-x - grg supergroup 0 2026-02-20 11:18 /mcs1
drwxr-xr-x - grg supergroup 0 2026-02-16 08:59 /user
grg@LAB4-1:~/hadoop-2.9.1$ cd $home
grg@LAB4-1:~$ touch file1.txt
grg@LAB4-1:~$ touch file2.txt
grg@LAB4-1:~$ cd hadoop-2.9.1
grg@LAB4-1:~/hadoop-2.9.1$ bin/hadoop fs -put/home/grg/file1.txt/file2.txt
-put/home/grg/file1.txt/file2.txt: Unknown command
Usage: hadoop fs [generic options]
[-appendToFile <localsrc> ... <dst>]
[-cat [-ignoreCrc] <src> ...]
[-checksum <src> ...]
[-chgrp [-R] GROUP PATH...]
[-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
[-chown [-R] [OWNER][[:GROUP]] PATH...]
[-copyFromLocal [-f] [-p] [-l] [-d] <localsrc> ... <dst>]
[-copyToLocal [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
[-count [-q] [-h] [-v] [-t <storage type>]] [-u] [-x] <path> ...]
[-cp [-f] [-p] [-p[topax]] [-d] <src> ... <dst>]
[-createSnapshot <snapshotDir> [<snapshotName>]]
[-deleteSnapshot <snapshotDir> <snapshotName>]
[-df [-h] [<path> ...]]
```

```
Activities Terminal Feb 20 11:27 grg@LAB4-1: ~/hadoop-2.9.1

[-moveFromLocal <localsrc> ... <dst>]
[-moveToLocal <src> <localdst>]
[-mv <src> ... <dst>]
[-put [-f] [-p] [-l] [-d] <localsrc> ... <dst>]
[-renameSnapshot <snapshotDir> <oldName> <newName>]
[-rm [-f] [-r] [-R] [-skipTrash] [-safely] <src> ...]
[-rmdir [--ignore-fail-on-non-empty] <dir> ...]
[-setfacl [-R] [{-b|-k} {-m|-x <acl_spec>} <path>][[--set <acl_spec> <path>]]]
[-setfattr {-n name [-v value] | -x name} <path>]
[-setrep [-R] [-w] <rep> <path> ...]
[-stat [format] <path> ...]
[-tail [-f] <file>]
[-test [-defsz] <path>]
[-text [-ignoreCrc] <src> ...]
[-touchz <path> ...]
[-truncate [-w] <length> <path> ...]
[-usage [cmd ...]]

Generic options supported are:
-conf <configuration file>          specify an application configuration file
-D <property=value>                 define a value for a given property
-fs <file:///|hdfs://namenode:port> specify default filesystem URL to use, overrides 'fs.defaultFS' property from configurations.
-it <local|resourcemanager:port>    specify a ResourceManager
-files <file1,...>                  specify a comma-separated list of files to be copied to the map reduce cluster
-lbjars <jar1,...>                  specify a comma-separated list of jar files to be included in the classpath
-archives <archive1,...>            specify a comma-separated list of archives to be unarchived on the compute machines

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

grg@LAB4-1:~/hadoop-2.9.1$ bin/hadoop fs -copyFromLocal /home/grg/file1.txt /file2.txt
grg@LAB4-1:~/hadoop-2.9.1$ bin/hadoop fs -copyToLocal /file2.txt /home/grg/file3.txt
grg@LAB4-1:~/hadoop-2.9.1$ bin/hadoop fs -cat /file2.txt
HELLO WELCOME TO PSGR

grg@LAB4-1:~/hadoop-2.9.1$ bin/hadoop fs -du /file2.txt
23 /file2.txt
grg@LAB4-1:~/hadoop-2.9.1$ bin/hadoop fs -rm /file2.txt
Deleted /file2.txt
grg@LAB4-1:~/hadoop-2.9.1$
```

RESULT

Thus the program is executed and the output is verified.

Ex. No:02

Date:

ELECTRICAL CONSUMPTION USING MAPREDUCE

AIM

To develop a MapReduce program to determine the maximum electrical consumption for each year based on monthly electrical consumption data for each year.

ALGORITHM

Step 1: Create a CSV file named eb.csv to store electrical consumption data.

Step 2: Open the file in a text editor and insert monthly consumption data for multiple years.

Step 3: Create a Python script mapper2.py to read the CSV file and extract year and consumption values.

Step 4: In mapper2.py, skip the header line and print year-consumption pairs separated by a tab.

Step 5: Create another Python script reducer2.py to process the mapper output and find the maximum consumption per year.

Step 6: In reducer2.py, iterate through the input, updating the maximum consumption for each year.

Step 7: When the year changes, print the maximum consumption for the previous year.

Step 8: At the end of the reducer, print the last year's maximum consumption.

Step 9: Execute the MapReduce pipeline using the command:

```
cat eb.csv | python3 mapper2.py | python3 reducer2.py > output2.txt
```

Step 10: Display the final results using cat output2.txt.

PROGRAM

In Terminal

```
touch eb.csv
nano mapper2.py
nano reducer2.py
cat eb.csv
cat eb.csv | python3 mapper2.py | python3 reducer2.py > output2.txt
cat output2.txt
```

eb.csv

```
Year,Month,Consumption
2020,Jan,320
2020,Feb,290
2020,Mar,450
2020,Apr,380
2021,Jan,500
2021,Feb,450
2021,Mar,510
2021,Apr,480
```

mapper2.py

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

for line in sys.stdin:
    line = line.strip()
    parts = line.split(",")
    # Skip header
    if parts[0] == "Year":
        continue
    try:
```

```
year = parts[0]
consumption = int(parts[2]) # Convert consumption to integer
print(f'{year}\t{consumption}')
except ValueError:
continue # Skip invalid lines
```

reducer2.py

```
#!/usr/bin/env python3
import sys
current_year = None
max_consumption = 0
for line in sys.stdin:
line = line.strip()
year, consumption = line.split("\t")
try:
consumption = int(consumption)
if current_year == year:
max_consumption = max(max_consumption, consumption)
else:
if current_year:
print(f'{current_year}\t{max_consumption}') # Print max for previous year
current_year = year
max_consumption = consumption
except ValueError:
continue
# Print last year's max
if current_year:
print(f'{current_year}\t{max_consumption}')
```

OUTPUT

```
grg@LAB4-1:~$ touch eb.csv
grg@LAB4-1:~$ nano mapper2.py
grg@LAB4-1:~$ nano reducer2.py
grg@LAB4-1:~$ cat eb.csv
grg@LAB4-1:~$ cat eb.csv
Year,Month,Consumption
2020,Jan,320
2020,Feb,290
2020,Mar,450
2020,Apr,380
2021,Jan,500
2021,Feb,450
2021,Mar,510
2021,Apr,480
grg@LAB4-1:~$ cat eb.csv | python3 mapper2.py | python3 reducer2.py > output2.txt
grg@LAB4-1:~$ cat output2.txt
2020      450
2021      510
grg@LAB4-1:~$
```

RESULT

Thus the program is executed and the output is verified.

Ex. No:03 Date:	WEATHER DATA USING MAPREDUCE
----------------------------------	-------------------------------------

AIM

To develop a MapReduce program to analyze the weather dataset and print whether the day is a shiny or cool day.

ALGORITHM

Step 1: Create a CSV file named weather.csv to store weather data.

Step 2: Open weather.csv and insert date, temperature, and weather type data.

Step 3: Create a Python script mapper1.py to process the CSV file.

Step 4: In mapper1.py, skip the header line and extract date and temperature values.

Step 5: Define temperature thresholds and classify days as "Shiny" or "Cool" based on temperature.

Step 6: Print the date and classification as tab-separated values.

Step 7: Create another Python script reducer1.py to process the mapper output.

Step 8: In reducer1.py, read the mapper output and format it as date is a category day.

Step 9: Run the MapReduce pipeline using the command:

```
cat weather.csv | python3 mapper1.py | python3 reducer1.py > output.txt
```

Step 10: Display the final categorized weather results using cat output.txt.

PROGRAM

In Terminal

```
touch weather.csv
```

```
nano mapper1.py
```

```
nano reducer1.py
```

```
cat weather.csv
```

```
cat weather.csv | python3 mapper1.py | python3 reducer1.py > output.txt
```

```
cat output.txt
```

weather.csv

```
Date,Temperature,WeatherType
```

```
2024-01-01,35,Sunny
```

```
2024-01-02,18,Cloudy
```

```
2024-01-03,42,Sunny
```

```
2024-01-04,12,Rainy
```

```
2024-01-05,30,Sunny
```

mapper1.py

```
#!/usr/bin/env python3
```

```
import sys
```

```
# Define temperature thresholds
```

```
HOT_THRESHOLD = 30 # Above 30°C is considered "Shiny"
```

```
COLD_THRESHOLD = 20 # Below 20°C is considered "Cool"
```

```
# Read input line by line
```

```
for line in sys.stdin:
```

```
    line = line.strip()
```

```
    parts = line.split(",")
```

```
    # Skip header line
```

```
    if parts[0] == "Date":
```

```
        continue
```

```
    # Extract date and temperature
```

```

date = parts[0]
try:
    temperature = int(parts[1]) # Convert temperature to integer
# Categorize the day as "Shiny" or "Cool"
    if temperature > HOT_THRESHOLD:
print(f'{date}\tShiny')
    elif temperature < COLD_THRESHOLD:
print(f'{date}\tCool')
    except ValueError:
continue # Skip lines with invalid data
reduce.py
#!/usr/bin/env python3
import sys

# Read input from standard input (sorted by Hadoop)
for line in sys.stdin:
    line = line.strip()
    try:
        date, category = line.split("\t")
# Output the final classification of the day
print(f'{date} is a {category} day')
    except ValueError:
continue # Skip malformed lines

```

OUTPUT

```
grg@LAB4-1:~$ nano mapper1.py
grg@LAB4-1:~$ nano reducer1.py
grg@LAB4-1:~$ cat weather.csv
Date,Temperature,WeatherType
2024-01-01,35,Sunny
2024-01-02,18,Cloudy
2024-01-03,42,Sunny
2024-01-04,12,Rainy
2024-01-05,30,Sunny
grg@LAB4-1:~$
```

```
2024-01-03,42,Sunny
grg@LAB4-1:~$ cat weather.csv | python3 mapper1.py | python3 reducer1.py > output.txt
grg@LAB4-1:~$ cat output.txt
2024-01-01 is a Shiny day
2024-01-02 is a Cool day
2024-01-03 is a Shiny day
2024-01-04 is a Cool day
grg@LAB4-1:~$
```

RESULT

Thus the program is executed and the output is verified.

Ex. No:04

Date:

COUNT GENDER AND HOBBY USING PIG SCRIPT

AIM

To write a pig script to find the count of Gender and Hobbies.

ALGORITHM

Step 1: Open the terminal and refresh the environment using the appropriate command.

Step 2: Navigate to the Pig directory using the 'cd' command.

Step 3: Start Pig in local mode to run the script without a Hadoop cluster.

Step 4: Load the dataset 'file2.txt', which contains gender and hobby information.

Step 5: Group the data by the hobby field to categorize individuals based on their hobbies.

Step 6: Count the number of individuals in each hobby category.

Step 7: Group the data by the gender field to categorize individuals based on gender.

Step 8: Count the number of individuals for each gender category.

Step 9: Display the results of both hobby and gender counts to view the final output.

Step 10: Execute the script in Pig interactive mode, verify the results, and ensure correctness

PROGRAM

PIG SETUP

.bashrc

```
export JAVA_HOME=/home/grg/jdk1.8.0_45
export PIG_HOME=/home/grg/pig-0.12.0
export PATH=$JAVA_HOME/bin:$PATH/bin:$PIG_HOME
#export HADOOP_HOME=/home/grg/hadoop-2.9.1
#export SPARK_HOME=/home/grg/spark-2.1.1-bin-hadoop2.7
#export PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:$HIVE_HOME/bin:$PATH
#export HIVE_HOME=/home/grg/hive-0.12.0
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64 # Or replace with your desired JDK path
export PIG_HOME=/home/grg/pig-0.12.0
export PATH=$PIG_HOME/bin:$JAVA_HOME/bin:$PATH
```

Core-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://localhost:9000</value>
<name>dfs.namenode.https.address</name>
<value>node:port</value>
</property>
</configuration>
```

Hive-site.xml

```
<configuration>
<property>
<name>hive.metastore.uris</name>
<value>thrift://localhost:9083</value>
```

<description>URI for the Hive Metastore service</description>

</property>

</configuration>

Terminal

```
source .bashrc
```

```
cd pig-0.12.0
```

```
bin/pig -x local
```

Create file2.txt

```
1,alice,female,jogging
```

```
2,anu,male,book
```

```
3,kalai,female,phone
```

```
4,mani,male,tv
```

```
5,laksh,female,book
```

```
6,keerthana,female,painting
```

```
7,vikas,male,reading
```

```
8,rohit,male,gaming
```

CODE

```
H= LOAD' /home/grg/file2.txt' using PigStorage(',') as
```

```
(id:int,name:chararray,gender:chararray,hobby:chararray);
```

```
grp = group H by hobby;
```

```
gr = FOREACH grp GENERATE group,COUNT(H);
```

```
dump gr;
```

```
grp_gender = GROUP H BY gender;
```

```
gender_count = FOREACH grp_gender GENERATE group AS gender, COUNT(H) AS count;
```

```
DUMP gender_count;
```

OUTPUT

```
Activities Terminal Jan 30 11:11 grg@LAB4-1: ~/pig-0.12.0

grunt> H= LOAD '/home/grg/file2.txt' using PigStorage(',') as (id:int,name:chararray,gender:chararray,hobby:chararray);
grunt> grp = group H by hobby;
grunt> gr = FOREACH grp GENERATE group,COUNT(H);
grunt> dump gr;
2026-01-30 10:51:27,359 [main] INFO org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: GROUP_BY
2026-01-30 10:51:27,359 [main] INFO org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach, Col
umnMapKeyPrune, DuplicateForEachColumnRewrite, GroupByConstParallelSetter, ImplicitSplitInserter, LimitOptimizer, LoadTypeCastInsert
er, MergeFilter, MergeForEach, NewPartitionFilterOptimizer, PartitionFilterOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFil
ter, StreamTypeCastInserter], RULES_DISABLED=[FilterLogicExpressionSimplifier]}
2026-01-30 10:51:27,361 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MRCompiler - File concatenation th
reshold: 100 optimistic? false
2026-01-30 10:51:27,362 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.CombinerOptimizer - Choosing to mo
ve algebraic foreach to combiner
2026-01-30 10:51:27,362 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size
before optimization: 1
2026-01-30 10:51:27,362 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size
after optimization: 1
2026-01-30 10:51:27,363 [main] INFO org.apache.pig.tools.pigstats.ScriptState - Pig script settings are added to the job
2026-01-30 10:51:27,363 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - mapred.job.re
duce.markreset.buffer.percent is not set, set to default 0.3
2026-01-30 10:51:27,364 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - Setting up si
ngle store job
2026-01-30 10:51:27,365 [main] INFO org.apache.pig.data.SchemaTupleFrontend - Key [pig.schematuple] is false, will not generate cod
e.
2026-01-30 10:51:27,365 [main] INFO org.apache.pig.data.SchemaTupleFrontend - Starting process to move generated code to distribute
d cache
2026-01-30 10:51:27,365 [main] INFO org.apache.pig.data.SchemaTupleFrontend - Distributed cache not supported or needed in local mo
de. Setting key [pig.schematuple.local.dir] with code temp directory: /tmp/1769750487365-0
2026-01-30 10:51:27,365 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - Reduce phase
detected, estimating # of required reducers.
2026-01-30 10:51:27,365 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - Using reducer
estimator: org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.InputSizeReducerEstimator
2026-01-30 10:51:27,365 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.InputSizeReducerEstimator - BytesP
erReducer=1000000000 maxReducers=999 totalInputFileSize=1
2026-01-30 10:51:27,365 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - Could not est
imate number of reducers and no requested or default parallelism set. Defaulting to 1 reducer.
2026-01-30 10:51:27,365 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - Setting Paral
lelism to 1
2026-01-30 10:51:27,370 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 1 map-reduce j
ob(s) waiting for submission.
```

```
Activities Terminal Jan 30 11:12 grg@LAB4-1: ~/pig-0.12.0

2026-01-30 10:51:33,891 [main] INFO org.apache.pig.tools.pigstats.SimplePigStats - Script Statistics:
HadoopVersion PigVersion UserId StartedAt FinishedAt Features
1.0.0 0.12.0 grg 2026-01-30 10:51:27 2026-01-30 10:51:33 GROUP_BY

Success!

Job Stats (time in seconds):
JobId Alias Feature Outputs GROUP_BY,COMBINER file:/tmp/temp-1722057135/tmp-723133696,
job_local_0002 H,gr,grp

Input(s):
Successfully read records from: "/home/grg/file2.txt"

Output(s):
Successfully stored records in: "file:/tmp/temp-1722057135/tmp-723133696"

Job DAG:
job_local_0002

2026-01-30 10:51:33,891 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2026-01-30 10:51:33,891 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2026-01-30 10:51:33,894 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2026-01-30 10:51:33,894 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process :
1
(tv,1)
(book,2)
(phone,1)
(gaming,1)
(jogging,1)
(reading,1)
(painting,1)
grunt> grp_gender = GROUP H BY gender;
grunt> gender_count = FOREACH grp_gender GENERATE group AS gender, COUNT(H) AS count;
grunt> DUMP gender_count;
2026-01-30 10:54:34,911 [main] INFO org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: GROUP_BY
2026-01-30 10:54:34,911 [main] INFO org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach, Col
umnMapKeyPrune, DuplicateForEachColumnRewrite, GroupByConstParallelSetter, ImplicitSplitInserter, LimitOptimizer, LoadTypeCastInsert
er, MergeFilter, MergeForEach, NewPartitionFilterOptimizer, PartitionFilterOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFil
```

```
Activities Terminal Jan 30 11:12 grg@LAB4-1: ~/pig-0.12.0
2026-01-30 10:54:37,972 [Thread-11] INFO org.apache.hadoop.mapred.LocalJobRunner -
2026-01-30 10:54:37,973 [Thread-11] INFO org.apache.hadoop.mapred.Task - Task attempt_local_0003_r_000000_0 is allowed to commit now
2026-01-30 10:54:37,973 [Thread-11] INFO org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - Saved output of task 'attempt
2026-01-30 10:54:40,966 [Thread-11] INFO org.apache.hadoop.mapred.LocalJobRunner - reduce > reduce
2026-01-30 10:54:40,967 [Thread-11] INFO org.apache.hadoop.mapred.Task - Task 'attempt_local_0003_r_000000_0' done.
2026-01-30 10:54:41,450 [main] WARN org.apache.pig.tools.pigstats.PigStatsUtil - Failed to get RunningJob for job job_local_0003
2026-01-30 10:54:41,450 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 100% complete
2026-01-30 10:54:41,451 [main] INFO org.apache.pig.tools.pigstats.SimplePigStats - Detected Local mode. Stats reported below may be
incomplete
2026-01-30 10:54:41,451 [main] INFO org.apache.pig.tools.pigstats.SimplePigStats - Script Statistics:

HadoopVersion PigVersion UserId StartedAt FinishedAt Features
1.0.0 0.12.0 grg 2026-01-30 10:54:34 2026-01-30 10:54:41 GROUP_BY

Success!

Job Stats (time in seconds):
JobId Alias Feature Outputs
job_local_0003 H,gender_count,grp_gender GROUP_BY,COMBINER file:/tmp/temp-1722057135/tmp-86732645,

Input(s):
Successfully read records from: "/home/grg/file2.txt"

Output(s):
Successfully stored records in: "file:/tmp/temp-1722057135/tmp-86732645"

Job DAG:
job_local_0003

2026-01-30 10:54:41,451 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2026-01-30 10:54:41,451 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2026-01-30 10:54:41,452 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2026-01-30 10:54:41,452 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process :
1
(male,4)
(female,4)
grunt>
```

RESULT

Thus the program is executed and the output is verified.

Ex. No:05 Date:	COUNT WORDS USING PIG SCRIPT
----------------------------------	-------------------------------------

AIM

To write a program to find the number of words using pig script.

ALGORITHM

Step 1: Open the terminal to begin the process.

Step 2: Navigate to the Pig directory using the command `cd pig-0.12.0`.

Step 3: Start Pig in local mode to execute the script without a Hadoop cluster.

Step 4: Load the dataset `text1.txt`, which contains lines of text, using the `TextLoader()` function.

Step 5: Tokenize each line into individual words to prepare for counting.

Step 6: Group all words together to count occurrences across the entire dataset.

Step 7: Count the total number of words present in the dataset.

Step 8: Display the final word count result using the `DUMP` command.

Step 9: Verify the output to ensure accuracy and correctness.

Step 10: Modify the script if necessary to refine or extend the word count analysis.

PROGRAM

```
inputfile = LOAD '/home/grg/input.txt' AS (line:chararray);  
DUMP inputfile;  
words = FOREACH inputfile GENERATE FLATTEN(TOKENIZE(line)) AS word;  
filtered_words = FILTER words BY word MATCHES '\\w+';  
word_groups = GROUP filtered_words BY word;  
word_count = FOREACH word_groups GENERATE group AS word, COUNT(filtered_words) AS  
count;  
sorted_word_count = ORDER word_count BY count DESC;  
DUMP sorted_word_count;
```

OUTPUT

```
Activities Terminal Jan 30 11:22 grg@LAB4-1: ~/pig-0.12.0

grunt> inputfile = LOAD '/home/grg/input.txt' AS (line:chararray);
grunt> DUMP inputfile;
2026-01-30 11:19:34,342 [main] INFO org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: UNKNOWN
2026-01-30 11:19:34,342 [main] INFO org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach, Col
umnMapKeyPrune, DuplicateForEachColumnRewrite, GroupByConstParallelSetter, ImplicitSplitInserter, LimitOptimizer, LoadTypeCastInsert
er, MergeFilter, MergeForEach, NewPartitionFilterOptimizer, PartitionFilterOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFil
ter, StreamTypeCastInserter], RULES_DISABLED=[FilterLogicExpressionSimplifier]}
2026-01-30 11:19:34,343 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MRCompiler - File concatenation th
reshold: 100 optimistic? false
2026-01-30 11:19:34,343 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size
before optimization: 1
2026-01-30 11:19:34,343 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size
after optimization: 1
2026-01-30 11:19:34,343 [main] INFO org.apache.pig.tools.pigstats.ScriptState - Pig script settings are added to the job
2026-01-30 11:19:34,344 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - mapred.job.re
duce.markreset.buffer.percent is not set, set to default 0.3
2026-01-30 11:19:34,345 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - Setting up si
ngle store job
2026-01-30 11:19:34,345 [main] INFO org.apache.pig.data.SchemaTupleFrontend - Key [pig.schematuple] is false, will not generate cod
e.
2026-01-30 11:19:34,345 [main] INFO org.apache.pig.data.SchemaTupleFrontend - Starting process to move generated code to distribute
d cache
2026-01-30 11:19:34,345 [main] INFO org.apache.pig.data.SchemaTupleFrontend - Distributed cache not supported or needed in local mo
de. Setting key [pig.schematuple.local.dir] with code temp directory: /tmp/1769752174345-0
2026-01-30 11:19:34,345 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - Map only job,
skipping reducer estimation
2026-01-30 11:19:34,347 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 1 map-reduce j
ob(s) waiting for submission.
2026-01-30 11:19:34,350 [JobControl] WARN org.apache.hadoop.mapred.JobClient - No job jar file set. User classes may not be found.
See JobConf(Class) or JobConf#setJar(String).
2026-01-30 11:19:34,352 [JobControl] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2026-01-30 11:19:34,352 [JobControl] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to proc
ess : 1
2026-01-30 11:19:34,352 [JobControl] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths (combin
ed) to process : 1
2026-01-30 11:19:34,365 [Thread-17] INFO org.apache.hadoop.mapred.Task - Using ResourceCalculatorPlugin : org.apache.hadoop.util.L
inuxResourceCalculatorPlugin$14041603
2026-01-30 11:19:34,366 [Thread-17] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.PigRecordReader - Current spl
it being processed file:/home/grg/input.txt:0+103
2026-01-30 11:19:34,368 [Thread-17] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not gen
```

```
ies Terminal Jan 30 11:23 grg@LAB4-1: ~/pig-0.12.0

HadoopVersion PigVersion UserId StartedAt FinishedAt Features
1.0.0 0.12.0 grg 2026-01-30 11:19:34 2026-01-30 11:19:37 UNKNOWN

Success!

Job Stats (time in seconds):
JobId Alias Feature Outputs
job_local_0006 inputfile MAP_ONLY file:/tmp/temp-1722057135/tmp510871105,

Input(s):
Successfully read records from: "/home/grg/input.txt"

Output(s):
Successfully stored records in: "file:/tmp/temp-1722057135/tmp510871105"

Job DAG:
job_local_0006

2026-01-30 11:19:37,860 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2026-01-30 11:19:37,860 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2026-01-30 11:19:37,861 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2026-01-30 11:19:37,861 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process :
1
(Apache Pig is a high-level platform run on Hadoop. The language for this platform is called)
(Pig Latin.)
grunt> words = FOREACH inputfile GENERATE FLATTEN(TOKENIZE(line)) AS word;
grunt> filtered_words = FILTER words BY word MATCHES '\\w+';
grunt> word_groups = GROUP filtered_words BY word;
grunt> word_count = FOREACH word_groups GENERATE group AS word, COUNT(filtered_words) AS count;
grunt> sorted_word_count = ORDER word_count BY count DESC;
grunt> DUMP sorted_word_count;
2026-01-30 11:21:46,154 [main] INFO org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: GROUP_BY,ORDER_BY,
FILTER
2026-01-30 11:21:46,155 [main] INFO org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach, Col
umnMapKeyPrune, DuplicateForEachColumnRewrite, GroupByConstParallelSetter, ImplicitSplitInserter, LimitOptimizer, LoadTypeCastInsert
er, MergeFilter, MergeForEach, NewPartitionFilterOptimizer, PartitionFilterOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFil
ter, StreamTypeCastInserter], RULES_DISABLED=[FilterLogicExpressionSimplifier]}
2026-01-30 11:21:46,159 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MRCompiler - File concatenation th
```

```
es Terminal Jan 30 11:24
grg@LAB4-1: ~/pig-0.12.0
1.0.0 0.12.0 grg 2026-01-30 11:21:46 2026-01-30 11:22:05 GROUP_BY,ORDER_BY,FILTER
Success!
Job Stats (time in seconds):
JobId Alias Feature Outputs
job_local_0007 filtered_words,inputfile,word_count,word_groups,words GROUP_BY,COMBINER
job_local_0008 sorted_word_count SAMPLER
job_local_0009 sorted_word_count ORDER_BY file:/tmp/temp-1722057135/tmp-1113243181,
Input(s):
Successfully read records from: "/home/grg/input.txt"
Output(s):
Successfully stored records in: "file:/tmp/temp-1722057135/tmp-1113243181"
Job DAG:
job_local_0007 -> job_local_0008,
job_local_0008 -> job_local_0009,
job_local_0009
2026-01-30 11:22:05,778 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2026-01-30 11:22:05,778 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2026-01-30 11:22:05,779 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2026-01-30 11:22:05,779 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process :
1
(is,2)
(Pig,2)
(platform,2)
(a,1)
(on,1)
(The,1)
(for,1)
(run,1)
(this,1)
(Apache,1)
(called,1)
(language,1)
JJ grunt>
```

RESULT

Thus the program is executed and the output is verified.

Ex. No:06

Date:

INTERNAL AND EXTERNAL TABLES IN HIVE

AIM

To write a program to create, alter and drop Internal and External tables in Hive.

ALGORITHM

Step 1: Open the terminal.

Step 2: Navigate to the home directory using `cd $HOME`.

Step 3: Start the Hive shell using `$HIVE_HOME/bin/hive`.

Step 4: Create an internal table named `internal_table` with columns `id`, `name`, and `age`.

Step 5: Create an external table named `external_table` with the same columns, specifying a file location.

Step 6: Describe both tables to check their structure.

Step 7: Alter the `internal_table` by changing the data type of the `age` column to `BIGINT`.

Step 8: Rename `internal_table` to `managed_table`.

Step 9: Describe `managed_table` to verify the changes.

Step 10: Drop the `external_table` to remove it from Hive metadata.

PROGRAM

HIVE SETUP

.bashrc

```
export JAVA_HOME=/home/grg/jdk1.8.0_45
export HADOOP_HOME=/home/grg/hadoop-2.9.0
export SPARK_HOME=/home/grg/spark-2.1.1-bin-hadoop2.7
export PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:$HIVE_HOME/bin:$PATH
export JAVA_HOME=/usr/lib/jpm/java-8-openjdk_amd64
export PIG_HOME=/home/grg/pig-0.12.0
export HADOOP_HOME=/home/grg/hadoop-2.9.1
export HIVE_HOME=/home/grg/hive-0.12.0
export PATH=$HIVE_HOME/bin:$PATH
```

In Terminal

```
cd $home
$HIVE_HOME/bin/hive
```

```
CREATE TABLE internal_table (
  id INT,
  name STRING,
  age INT
)
STORED AS TEXTFILE;
CREATE EXTERNAL TABLE external_table (
  id INT,
  name STRING,
  age INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION '/home/grg/external_data/';
DESCRIBE internal_table;
```

```
DESCRIBE external_table;  
ALTER TABLE internal_table CHANGE COLUMN age age BIGINT;  
ALTER TABLE internal_table RENAME TO managed_table;  
DESCRIBE managed_table;  
DROP TABLE external_table;  
DESCRIBE external_table;
```

OUTPUT

```
grg@LAB4-1:~$ cd $HOME
grg@LAB4-1:~$ SHIVE_HOME/bin/hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/grg/hadoop-2.9.1/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/grg/hive-0.12.0/lib/slf4j-log4j12-1.6.1.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.slf4j.impl.Log4jLoggerFactory]
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/grg/hadoop-2.9.1/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/grg/hive-0.12.0/lib/slf4j-log4j12-1.6.1.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.slf4j.impl.Log4jLoggerFactory]
26/01/03 10:51:11 INFO Configuration.deprecation: mapred.input.dir.recursive is deprecated. Instead, use mapreduce.input.fileinputformat.input.dir.recursive
26/01/03 10:51:11 INFO Configuration.deprecation: mapred.max.split.size is deprecated. Instead, use mapreduce.input.fileinputformat.split.maxsize
26/01/03 10:51:11 INFO Configuration.deprecation: mapred.min.split.size is deprecated. Instead, use mapreduce.input.fileinputformat.split.minsize
26/01/03 10:51:11 INFO Configuration.deprecation: mapred.min.split.size.per.rack is deprecated. Instead, use mapreduce.input.fileinputformat.split.minsize.per.rack
26/01/03 10:51:11 INFO Configuration.deprecation: mapred.min.split.size.per.node is deprecated. Instead, use mapreduce.input.fileinputformat.split.minsize.per.node
26/01/03 10:51:11 INFO Configuration.deprecation: mapred.reduce.tasks is deprecated. Instead, use mapreduce.job.reduces
26/01/03 10:51:11 INFO Configuration.deprecation: mapred.reduce.tasks.speculative.execution is deprecated. Instead, use mapreduce.reduce.speculative
```

```
26/01/03 10:51:11 INFO Configuration.deprecation: mapred.reduce.tasks is deprecated. Instead, use mapreduce.job.reduces
26/01/03 10:51:11 INFO Configuration.deprecation: mapred.reduce.tasks.speculative.execution is deprecated. Instead, use mapreduce.reduce.speculative

Logging initialized using configuration in jar:file:/home/grg/hive-0.12.0/lib/hive-common-0.12.0.jar!/hive-log4j.properties
hive> CREATE TABLE internal_table (
  >   id INT,
  >   name STRING,
  >   age INT
  > )
  > STORED AS TEXTFILE;
OK
Time taken: 3.231 seconds
hive> CREATE EXTERNAL TABLE external_table (
  >   id INT,
  >   name STRING,
  >   age INT
  > )
  > ROW FORMAT DELIMITED
  > FIELDS TERMINATED BY ','
  > LOCATION '/home/grg/external_data/';
OK
Time taken: 0.025 seconds
hive> DESCRIBE internal_table;
OK
id                int                None
name              string            None
age               int                None
Time taken: 0.141 seconds, Fetched: 3 row(s)
hive> DESCRIBE external_table;
OK
id                int                None
name              string            None
age               int                None
Time taken: 0.031 seconds, Fetched: 3 row(s)
hive> ALTER TABLE internal_table CHANGE COLUMN age age BIGINT;
OK
Time taken: 0.071 seconds
hive> ALTER TABLE internal_table RENAME TO managed_table;
```

```

Time taken: 3.231 seconds
hive> CREATE EXTERNAL TABLE external_table (
  >   id INT,
  >   name STRING,
  >   age INT
  > )
  > ROW FORMAT DELIMITED
  > FIELDS TERMINATED BY ','
  > LOCATION '/home/grg/external_data/';
OK
Time taken: 0.025 seconds
hive> DESCRIBE internal_table;
OK
id                int                None
name              string              None
age               int                None
Time taken: 0.141 seconds, Fetched: 3 row(s)
hive> DESCRIBE external_table;
OK
id                int                None
name              string              None
age               int                None
Time taken: 0.031 seconds, Fetched: 3 row(s)
hive> ALTER TABLE internal_table CHANGE COLUMN age age BIGINT;
OK
Time taken: 0.071 seconds
hive> ALTER TABLE internal_table RENAME TO managed_table;
OK
Time taken: 0.198 seconds
hive> DESCRIBE managed_table;
OK
id                int                None
name              string              None
age               bigint             None
Time taken: 0.051 seconds, Fetched: 3 row(s)
hive> DROP TABLE external_table;
OK
Time taken: 0.395 seconds
hive>

```

RESULT

Thus the program is executed and the output is verified.

Ex. No:07

Date:

PARTITIONS AND BUCKETS IN HIVE

AIM

To write a program to implement tables, partitions and buckets in Hive.

ALGORITHM

Step 1: Open MySQL terminal with root access.

Step 2: Create the database and switch to it.

Step 3: Drop existing Students and StudentMarks tables if they exist.

Step 4: Create the Students table with range partitioning on AdmissionDate.

Step 5: Create the StudentMarks table with hash partitioning on StudentID.

Step 6: Insert sample data into the Students table.

Step 7: Insert sample data into the StudentMarks table.

Step 8: Retrieve all records from the Students table.

Step 9: Retrieve all records from StudentMarks table.

Step 10: Check partition details in INFORMATION_SCHEMA.PARTITIONS.

PROGRAM

In Terminal

mysql -u root -p

Password : root

```
CREATE DATABASE IF NOT EXISTS UniversityDB;
```

```
USE UniversityDB;
```

```
DROP TABLE IF EXISTS StudentMarks;
```

```
DROP TABLE IF EXISTS Students;
```

```
CREATE TABLE Students (
```

```
StudentID INT NOT NULL, -- Removed AUTO_INCREMENT
```

```
Name VARCHAR(100) NOT NULL,
```

```
DateOfBirth DATE NOT NULL,
```

```
AdmissionDate DATE NOT NULL,
```

```
Course VARCHAR(50),
```

```
PRIMARY KEY (StudentID, AdmissionDate) -- Required for partitioning
```

```
)
```

```
PARTITION BY RANGE (YEAR(AdmissionDate)) (
```

```
PARTITION p1 VALUES LESS THAN (2020),
```

```
PARTITION p2 VALUES LESS THAN (2022),
```

```
PARTITION p3 VALUES LESS THAN (2024),
```

```
PARTITION p4 VALUES LESS THAN MAXVALUE
```

```
);
```

```
CREATE TABLE StudentMarks (
```

```
MarkID INT NOT NULL AUTO_INCREMENT,
```

```
StudentID INT NOT NULL,
```

```
Subject VARCHAR(50),
```

```
Marks INT,
```

```
PRIMARY KEY (MarkID, StudentID) -- Includes StudentID to satisfy partitioning requirements
```

```
)
```

```
PARTITION BY HASH(StudentID) PARTITIONS 4;
```

```
INSERT INTO Students (StudentID, Name, DateOfBirth, AdmissionDate, Course) VALUES
(1, 'Alice Johnson', '2002-05-14', '2019-07-15', 'Computer Science'),
(2, 'Bob Smith', '2001-08-21', '2021-09-01', 'Mechanical Engineering'),
(3, 'Charlie Brown', '2003-02-11', '2023-08-20', 'Electrical Engineering');
INSERT INTO StudentMarks (StudentID, Subject, Marks) VALUES
(1, 'Math', 85),
(1, 'Physics', 78),
(2, 'Math', 90),
(2, 'Chemistry', 88),
(3, 'Physics', 82),
(3, 'Biology', 91);
SELECT * FROM Students;
SELECT * FROM StudentMarks;
SELECT PARTITION_NAME, TABLE_NAME, TABLE_SCHEMA
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_NAME = 'StudentMarks';
```

OUTPUT

```
es Terminal Feb 5 13:53 grg@LAB4-1: ~
mysql> CREATE DATABASE IF NOT EXISTS UniversityDB;
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> USE UniversityDB;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> DROP TABLE IF EXISTS StudentMarks;
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> DROP TABLE IF EXISTS Students;
Query OK, 0 rows affected (0.03 sec)
```

```
es Terminal Feb 5 13:57 grg@LAB4-1: ~
mysql> CREATE TABLE Students (
->
->   StudentID INT NOT NULL, -- Removed AUTO_INCREMENT
->   Name VARCHAR(100) NOT NULL,
->   DateOfBirth DATE NOT NULL,
->   AdmissionDate DATE NOT NULL,
->   Course VARCHAR(50),
->   PRIMARY KEY (StudentID, AdmissionDate) -- Required for partitioning
-> )
->
-> PARTITION BY RANGE (YEAR(AdmissionDate)) (
->   PARTITION p1 VALUES LESS THAN (2020),
->   PARTITION p2 VALUES LESS THAN (2022),
->   PARTITION p3 VALUES LESS THAN (2024),
->   PARTITION p4 VALUES LESS THAN MAXVALUE
-> );
Query OK, 0 rows affected (0.11 sec)
```

```
mysql> CREATE TABLE StudentMarks (
->   MarkID INT NOT NULL AUTO_INCREMENT,
->   StudentID INT NOT NULL,
->   Subject VARCHAR(50),
->   Marks INT,
->   PRIMARY KEY (MarkID, StudentID) -- Includes StudentID to satisfy partitioning requirements
-> )
->
-> PARTITION BY HASH(StudentID) PARTITIONS 4;
Query OK, 0 rows affected (0.11 sec)
```

```
mysql> INSERT INTO Students (StudentID, Name, DateOfBirth, AdmissionDate, Course) VALUES
-> (1, 'Alice Johnson', '2002-05-14', '2019-07-15', 'Computer Science'),
-> (2, 'Bob Smith', '2001-08-21', '2021-09-01', 'Mechanical Engineering'),
-> (3, 'Charlie Brown', '2003-02-11', '2023-08-20', 'Electrical Engineering');
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> INSERT INTO StudentMarks (StudentID, Subject, Marks) VALUES
-> (1, 'Math', 85),
-> (1, 'Physics', 78),
-> (2, 'Math', 90),
-> (2, 'Chemistry', 88),
-> (3, 'Physics', 82),
-> (3, 'Biology', 91);
Query OK, 6 rows affected (0.01 sec)
Records: 6 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT * FROM Students;
+-----+-----+-----+-----+-----+
| StudentID | Name       | DateOfBirth | AdmissionDate | Course          |
+-----+-----+-----+-----+-----+
| 1 | Alice Johnson | 2002-05-14 | 2019-07-15 | Computer Science |
| 2 | Bob Smith    | 2001-08-21 | 2021-09-01 | Mechanical Engineering |
| 3 | Charlie Brown | 2003-02-11 | 2023-08-20 | Electrical Engineering |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT * FROM StudentMarks;
+-----+-----+-----+-----+
| MarkID | StudentID | Subject | Marks |
+-----+-----+-----+-----+
| 1 | 1 | Math | 85 |
| 2 | 1 | Physics | 78 |
| 3 | 2 | Math | 90 |
| 4 | 2 | Chemistry | 88 |
| 5 | 3 | Physics | 82 |
| 6 | 3 | Biology | 91 |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

```
mysql> SELECT PARTITION_NAME, TABLE_NAME, TABLE_SCHEMA
-> FROM INFORMATION_SCHEMA.PARTITIONS
-> WHERE TABLE_NAME = 'StudentMarks';
+-----+-----+-----+
| PARTITION_NAME | TABLE_NAME | TABLE_SCHEMA |
+-----+-----+-----+
| p3 | StudentMarks | UniversityDB |
| p2 | StudentMarks | UniversityDB |
| p1 | StudentMarks | UniversityDB |
| p0 | StudentMarks | UniversityDB |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

RESULT

Thus the program is executed and the output is verified.

Ex. No:08	BUILT IN FUNCTIONS IN HIVE
Date:	

AIM

To write a program to implement built-in functions in Hive for employee and student details.

ALGORITHM

Step 1: Open MySQL terminal with root access.

Step 2: Create the CompanyDB database and switch to it.

Step 3: Create the Employees table with necessary columns.

Step 4: Create the Students table with relevant attributes.

Step 5: Insert sample data into the Employees table.

Step 6: Insert sample data into the Students table.

Step 7: Retrieve all records from the Employees table.

Step 8: Retrieve all records from the Students table.

Step 9: Convert employee names to uppercase using UPPER().

Step 10: Calculate the average salary of employees using AVG().

Step 11: Compute student ages using TIMESTAMPDIFF().

Step 12: Concatenate employee names with department details using CONCAT().

PROGRAM

```
CREATE DATABASE IF NOT EXISTS CompanyDB;
USE CompanyDB;
CREATE TABLE Employees (
EmpID INT AUTO_INCREMENT PRIMARY KEY,
Name VARCHAR(100),
Salary DECIMAL(10,2),
Department VARCHAR(50),
JoiningDate DATE
);
CREATE TABLE Students (
StudentID INT AUTO_INCREMENT PRIMARY KEY,
Name VARCHAR(100),
DateOfBirth DATE,
AdmissionDate DATE,
Course VARCHAR(50)
);
INSERT INTO Employees (Name, Salary, Department, JoiningDate) VALUES
('Alice Johnson', 60000.00, 'IT', '2020-06-15'),
('Bob Smith', 75000.50, 'HR', '2019-09-10'),
('Charlie Brown', 50000.75, 'Finance', '2021-03-22');
INSERT INTO Students (Name, DateOfBirth, AdmissionDate, Course) VALUES
('David Miller', '2002-07-10', '2020-08-15', 'Computer Science'),
('Emma Wilson', '2001-05-25', '2019-07-10', 'Mechanical Engineering'),
('Sophia Anderson', '2003-09-30', '2021-09-01', 'Electrical Engineering');
SELECT * FROM Employees;
SELECT * FROM Students;
SELECT UPPER(Name) AS UpperName FROM Employees;
SELECT AVG(Salary) AS AvgSalary FROM Employees;
SELECT Name, TIMESTAMPDIFF(YEAR, DateOfBirth, CURDATE()) AS Age FROM Students;
SELECT CONCAT(Name, ' works in ', Department) AS EmployeeInfo FROM Employees;
```

OUTPUT

```
es Terminal Feb 5 14:10
grg@LAB4-1: ~

mysql> CREATE DATABASE IF NOT EXISTS CompanyDB;
Query OK, 1 row affected (0.03 sec)

mysql> USE CompanyDB;
Database changed

mysql> CREATE TABLE Employees (
->
->     EmpID INT AUTO_INCREMENT PRIMARY KEY,
->
->     Name VARCHAR(100),
->
->     Salary DECIMAL(10,2),
->
->     Department VARCHAR(50),
->
->     JoiningDate DATE
-> );
Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE Students (
->
->     StudentID INT AUTO_INCREMENT PRIMARY KEY,
->
->     Name VARCHAR(100),
->
->     DateOfBirth DATE,
->
->     AdmissionDate DATE,
->
->     Course VARCHAR(50)
-> );
Query OK, 0 rows affected (0.05 sec)

es Terminal Feb 5 14:11
grg@LAB4-1: ~

mysql> INSERT INTO Employees (Name, Salary, Department, JoiningDate) VALUES
->
->     ('Alice Johnson', 60000.00, 'IT', '2020-06-15'),
->
->     ('Bob Smith', 75000.50, 'HR', '2019-09-10'),
->
->     ('Charlie Brown', 50000.75, 'Finance', '2021-03-22');
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> INSERT INTO Students (Name, DateOfBirth, AdmissionDate, Course) VALUES
->
->     ('David Miller', '2002-07-10', '2020-08-15', 'Computer Science'),
->
->     ('Emma Wilson', '2001-05-25', '2019-07-10', 'Mechanical Engineering'),
->
->     ('Sophia Anderson', '2003-09-30', '2021-09-01', 'Electrical Engineering');
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM Employees;
+-----+-----+-----+-----+-----+
| EmpID | Name       | Salary | Department | JoiningDate |
+-----+-----+-----+-----+-----+
| 1     | Alice Johnson | 60000.00 | IT         | 2020-06-15 |
| 2     | Bob Smith    | 75000.50 | HR         | 2019-09-10 |
| 3     | Charlie Brown | 50000.75 | Finance    | 2021-03-22 |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT * FROM Students;
+-----+-----+-----+-----+-----+
| StudentID | Name       | DateOfBirth | AdmissionDate | Course          |
+-----+-----+-----+-----+-----+
| 1         | David Miller | 2002-07-10 | 2020-08-15 | Computer Science |
| 2         | Emma Wilson  | 2001-05-25 | 2019-07-10 | Mechanical Engineering |
| 3         | Sophia Anderson | 2003-09-30 | 2021-09-01 | Electrical Engineering |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```
es Terminal Feb 5 14:11
grg@LAB4-1: ~
mysql> INSERT INTO Employees (Name, Salary, Department, JoiningDate) VALUES
->
-> ('Alice Johnson', 60000.00, 'IT', '2020-06-15'),
->
-> ('Bob Smith', 75000.50, 'HR', '2019-09-10'),
->
-> ('Charlie Brown', 50000.75, 'Finance', '2021-03-22');
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> INSERT INTO Students (Name, DateOfBirth, AdmissionDate, Course) VALUES
->
-> ('David Miller', '2002-07-10', '2020-08-15', 'Computer Science'),
->
-> ('Emma Wilson', '2001-05-25', '2019-07-10', 'Mechanical Engineering'),
->
-> ('Sophia Anderson', '2003-09-30', '2021-09-01', 'Electrical Engineering');
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM Employees;
+-----+-----+-----+-----+-----+
| EmpID | Name       | Salary | Department | JoiningDate |
+-----+-----+-----+-----+-----+
| 1     | Alice Johnson | 60000.00 | IT         | 2020-06-15 |
| 2     | Bob Smith    | 75000.50 | HR         | 2019-09-10 |
| 3     | Charlie Brown | 50000.75 | Finance    | 2021-03-22 |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT * FROM Students;
+-----+-----+-----+-----+-----+
| StudentID | Name       | DateOfBirth | AdmissionDate | Course           |
+-----+-----+-----+-----+-----+
| 1         | David Miller | 2002-07-10 | 2020-08-15 | Computer Science |
| 2         | Emma Wilson  | 2001-05-25 | 2019-07-10 | Mechanical Engineering |
| 3         | Sophia Anderson | 2003-09-30 | 2021-09-01 | Electrical Engineering |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

RESULT

Thus the program is executed and the output is verified.

Ex. No:09

Date:

JOIN OPERATIONS IN HIVE

AIM

To write a Hive program to perform four types of join operations for customer table.

ALGORITHM

Step 1: Open MySQL terminal with root access.

Step 2: Create the Customers table with CustomerID, Name, and City.

Step 3: Create Orders table with OrderID, CustomerID, and OrderAmount using foreign key.

Step 4: Insert sample data into the Customers table.

Step 5: Insert sample data into the Orders table.

Step 6: Retrieve all records from the Customers table.

Step 7: Retrieve all records from the Orders table.

Step 8: Perform an INNER JOIN to fetch matching customer orders.

Step 9: Perform a LEFT JOIN to get all customers, including those without orders.

Step 10: Perform a RIGHT JOIN to get all orders, including those without customers.

Step 11: Use UNION to combine LEFT JOIN and RIGHT JOIN to get a full outer join.

PROGRAM

```
CREATE TABLE Customers (  
    CustomerID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    City VARCHAR(50)  
);
```

```
CREATE TABLE Orders (  
    OrderID INT PRIMARY KEY,  
    CustomerID INT,  
    OrderAmount DECIMAL(10,2),  
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)  
);
```

```
INSERT INTO Customers (CustomerID, Name, City) VALUES  
(1, 'Alice', 'New York'),  
(2, 'Bob', 'Los Angeles'),  
(3, 'Charlie', 'Chicago');
```

```
INSERT INTO Orders (OrderID, CustomerID, OrderAmount) VALUES  
(101, 1, 250.50), -- Matches Alice (ID 1)  
(102, 2, 180.75); -- Matches Bob (ID 2)
```

```
SELECT * FROM Customers;
```

```
SELECT * FROM Orders;
```

```
SELECT C.CustomerID, C.Name, C.City, O.OrderID, O.OrderAmount
```

```
FROM Customers C
```

```
INNER JOIN Orders O
```

```
ON C.CustomerID = O.CustomerID;

SELECT C.CustomerID, C.Name, C.City, O.OrderID, O.OrderAmount

FROM Customers C

LEFT JOIN Orders O

ON C.CustomerID = O.CustomerID;

SELECT C.CustomerID, C.Name, C.City, O.OrderID, O.OrderAmount

FROM Customers C

RIGHT JOIN Orders O

ON C.CustomerID = O.CustomerID;

SELECT C.CustomerID, C.Name, C.City, O.OrderID, O.OrderAmount

FROM Customers C

LEFT JOIN Orders O

ON C.CustomerID = O.CustomerID

UNION

SELECT C.CustomerID, C.Name, C.City, O.OrderID, O.OrderAmount

FROM Customers C

RIGHT JOIN Orders O

ON C.CustomerID = O.CustomerID;
```

OUTPUT

```
es Terminal Feb 5 14:59 grg@LAB4-1: ~
mysql> CREATE TABLE Customers (
-> CustomerID INT PRIMARY KEY,
-> Name VARCHAR(100),
-> City VARCHAR(50)
-> );
Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE Orders (
-> OrderID INT PRIMARY KEY,
-> CustomerID INT,
-> OrderAmount DECIMAL(10,2),
-> FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
-> );
Query OK, 0 rows affected (0.04 sec)
```

```
es Terminal Feb 5 14:59 grg@LAB4-1: ~
mysql> INSERT INTO Customers (CustomerID, Name, City) VALUES
-> (1, 'Alice', 'New York'),
-> (2, 'Bob', 'Los Angeles'),
-> (3, 'Charlie', 'Chicago');
Query OK, 3 rows affected (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> INSERT INTO Orders (OrderID, CustomerID, OrderAmount) VALUES
-> (101, 1, 250.50), -- Matches Alice (ID 1)
-> (102, 2, 180.75); -- Matches Bob (ID 2)
Query OK, 2 rows affected (0.03 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM Customers;
+-----+-----+-----+
| CustomerID | Name | City |
+-----+-----+-----+
| 1 | Alice | New York |
| 2 | Bob | Los Angeles |
| 3 | Charlie | Chicago |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT * FROM Orders;
+-----+-----+-----+
| OrderID | CustomerID | OrderAmount |
+-----+-----+-----+
| 101 | 1 | 250.50 |
| 102 | 2 | 180.75 |
+-----+-----+-----+
2 rows in set (0.01 sec)

mysql> SELECT C.CustomerID, C.Name, C.City, 0.OrderID, 0.OrderAmount
->
-> FROM Customers C
```

```
mysql> SELECT C.CustomerID, C.Name, C.City, O.OrderID, O.OrderAmount
->
-> FROM Customers C
->
-> INNER JOIN Orders O
->
-> ON C.CustomerID = O.CustomerID;
+-----+-----+-----+-----+-----+
| CustomerID | Name | City | OrderID | OrderAmount |
+-----+-----+-----+-----+-----+
| 1 | Alice | New York | 101 | 250.50 |
| 2 | Bob | Los Angeles | 102 | 180.75 |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> SELECT C.CustomerID, C.Name, C.City, O.OrderID, O.OrderAmount
->
-> FROM Customers C
->
-> LEFT JOIN Orders O
->
-> ON C.CustomerID = O.CustomerID;
+-----+-----+-----+-----+-----+
| CustomerID | Name | City | OrderID | OrderAmount |
+-----+-----+-----+-----+-----+
| 1 | Alice | New York | 101 | 250.50 |
| 2 | Bob | Los Angeles | 102 | 180.75 |
| 3 | Charlie | Chicago | NULL | NULL |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql> SELECT C.CustomerID, C.Name, C.City, O.OrderID, O.OrderAmount
->
-> FROM Customers C
->
-> RIGHT JOIN Orders O
->
-> ON C.CustomerID = O.CustomerID;
+-----+-----+-----+-----+-----+
| CustomerID | Name | City | OrderID | OrderAmount |
+-----+-----+-----+-----+-----+
| 1 | Alice | New York | 101 | 250.50 |
| 2 | Bob | Los Angeles | 102 | 180.75 |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> SELECT C.CustomerID, C.Name, C.City, O.OrderID, O.OrderAmount
->
-> FROM Customers C
->
-> LEFT JOIN Orders O
->
-> ON C.CustomerID = O.CustomerID
-> UNION
->
-> SELECT C.CustomerID, C.Name, C.City, O.OrderID, O.OrderAmount
->
-> FROM Customers C
->
-> RIGHT JOIN Orders O
->
-> ON C.CustomerID = O.CustomerID;
+-----+-----+-----+-----+-----+
| CustomerID | Name | City | OrderID | OrderAmount |
+-----+-----+-----+-----+-----+
| 1 | Alice | New York | 101 | 250.50 |
| 2 | Bob | Los Angeles | 102 | 180.75 |
| 3 | Charlie | Chicago | NULL | NULL |
+-----+-----+-----+-----+-----+
3 rows in set (0.02 sec)
```

RESULT

Thus the program is executed and the output is verified.

Ex. No: 10

Date:

CREATE EMPLOYEE CLASS IN SCALA ENVIRONMENT

AIM

To write a Scala program that creates a class Employee with properties like name, age and designation. Implement a method to display employee details.

ALGORITHM

Step 1: Open terminal.

Step 2: Navigate to the home directory using `cd $home`.

Step 3: Change directory to Spark installation: `cd spark-2.1.1-bin-hadoop2.7`.

Step 4: Start Spark Shell: `bin/spark-shell`.

Step 5: Define an EmployeeApp object inside the Scala environment.

Step 6: Create an Employee class with attributes `employeeId`, `name`, `department`, and `age`.

Step 7: Implement a method `displayDetails()` to print employee details.

Step 8: Inside `main()`, create an instance of the Employee class with sample data.

Step 9: Call `displayDetails()` to print employee information.

Step 10: Run the application using `EmployeeApp.main(null)` to view the output.

PROGRAM

```
object EmployeeApp {  
  class Employee(val employeeId: Int, val name: String, val department: String, val age: Int) {  
    def displayDetails(): Unit = {  
      println(s"Employee ID: $employeeId")  
      println(s"Name: $name")  
      println(s"Department: $department")  
      println(s"Age: $age")  
    }  
  }  
  
  def main(args: Array[String]): Unit = {  
    val emp1 = new Employee(101, "John Doe", "Finance", 30)  
    emp1.displayDetails()  
  }  
}  
  
EmployeeApp.main{null}
```

OUTPUT

```
scala> object EmployeeApp {  
  |   class Employee(val employeeId: Int, val name: String, val department: String, val age: Int) {  
  |     def displayDetails(): Unit = {  
  |       println(s"Employee ID: $employeeId")  
  |       println(s"Name: $name")  
  |       println(s"Department: $department")  
  |       println(s"Age: $age")  
  |     }  
  |   }  
  |  
  |   def main(args: Array[String]): Unit = {  
  |     val emp1 = new Employee(101, "John Doe", "Finance", 30)  
  |     emp1.displayDetails()  
  |   }  
  | }  
defined object EmployeeApp  
  
scala> EmployeeApp.main(null)  
Employee ID: 101  
Name: John Doe  
Department: Finance  
Age: 30
```

RESULT

Thus the program is executed and the output is verified.

Ex. No: 11	CHECK CHARACTER SEQUENCE IN SCALA
Date:	

AIM

To write a Scala program to test if a given string contains the specified sequence of character values.

ALGORITHM

Step 1: Open terminal.

Step 2: Navigate to the home directory using `cd $home`.

Step 3: Change directory to Spark installation: `cd spark-2.1.1-bin-hadoop2.7`.

Step 4: Start Spark Shell: `bin/spark-shell`.

Step 5: Define an object `StringChecker` inside the Scala environment.

Step 6: Implement the method `containsSequence()` to check if string contains a given sequence.

Step 7: Inside `main()`, define an `inputString` and `sequenceToCheck`.

Step 8: Call `containsSequence()` and store the result in a variable.

Step 9: Print appropriate messages based on whether the sequence is found.

Step 10: Run the application using `StringChecker.main(null)` to view the result.

SCALA SETUP

In Terminal

```
cd $home
```

```
cd spark-2.1.1-bin-hadoop2.7
```

```
bin/spark-shell
```

```
bash: ./usr/share/bash-completion/bash_completion: No such file or directory
grggLAB4-1: $ cd $home
grggLAB4-1: $ cd spark-2.1.1-bin-hadoop2.7
grggLAB4-1: ~/spark-2.1.1-bin-hadoop2.7 $ bin/spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/12/24 09:34:16 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
25/12/24 09:34:16 WARN Utils: Your hostname, LAB4-1 resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s3)
25/12/24 09:34:16 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
25/12/24 09:34:19 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1766549056989).
Spark session available as 'spark'.
Welcome to

  ____              __
 / ___/  ___  ____  /  _/   ____  ____
/___/  /___/  /___/  /___/   /___/  /___/
version 2.1.1

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_45)
Type in expressions to have them evaluated.
Type :help for more information.
```

PROGRAM

```
object StringChecker {
  def containsSequence(str: String, sequence: String): Boolean = {
    str.contains(sequence)
  }

  def main(args: Array[String]): Unit = {
    val inputString = "Hello, world!"
    val sequenceToCheck = "world"
    val contains = containsSequence(inputString, sequenceToCheck)
    if(contains) {
      println(s"The string '$inputString' contains the sequence '$sequenceToCheck'.")
    } else {
```

```
println(s"The string '$inputString' does not contain the sequence '$sequenceToCheck'.")
}

}

}
```

TO VIEW OUTPUT

```
StringChecker.main{null}
```

OUTPUT

```
scala> object StringChecker {  
  |   def containsSequence(str: String, sequence: String): Boolean = {  
  |     str.contains(sequence)  
  |   }  
  |  
  |   def main(args: Array[String]): Unit = {  
  |     val inputString = "Hello, world!"  
  |     val sequenceToCheck = "world"  
  |     val contains = containsSequence(inputString, sequenceToCheck)  
  |  
  |     if (contains) {  
  |       println(s"The string '$inputString' contains the sequence '$sequenceToCheck'.")  
  |     } else {  
  |       println(s"The string '$inputString' does not contain the sequence '$sequenceToCheck'.")  
  |     }  
  |   }  
  | }  
defined object StringChecker  
  
scala> StringChecker.main{null}  
The string 'Hello, world!' contains the sequence 'world'.
```

Activate Win

RESULT

Thus the program is executed and the output was verified.

Ex. No: 12	MIN AND MAX VALUES IN MAP USING SCALA
Date:	

AIM:

To write a Scala program to create a map and find the minimum and maximum values in the map.

ALGORITHM

Step 1: Open the terminal.

Step 2: Navigate to the home directory: `cd $home`.

Step 3: Change directory to Spark installation: `cd spark-2.1.1-bin-hadoop2.7`.

Step 4: Start Spark Shell: `bin/spark-shell`.

Step 5: Define an object MapMinMax inside the Scala environment.

Step 6: Create a Map containing string keys and integer values.

Step 7: Find the minimum value in the map using `.values.min`.

Step 8: Find the maximum value in the map using `.values.max`.

Step 9: Print the minimum and maximum values.

Step 10: Run the application using `MapMinMax.main(null)` to view the output.

PROGRAM

```
object MapMinMax {  
  
  def main(args: Array[String]): Unit = {  
  
    val numbersMap = Map("one" -> 1, "two" -> 2, "three" -> 3, "four" -> 4, "five" -> 5)  
    val minValue = numbersMap.minBy(_._2) // Get the pair with the minimum value  
    val maxValue = numbersMap.maxBy(_._2) // Get the pair with the maximum value  
    println(s"Minimum Value in Map: ${minValue._2}")  
    println(s"Maximum Value in Map: ${maxValue._2}")  
  
  }  
  
}
```

To View the Output:

```
MapMinMax.main{null}
```

OUTPUT

```
scala> object MapMinMax {  
  |   def main(args: Array[String]): Unit = {  
  |     val numbersMap = Map("one" -> 1, "two" -> 2, "three" -> 3, "four" -> 4, "five" -> 5)  
  |     val minValue = numbersMap.minBy(_._2) // Get the pair with the minimum value  
  |     val maxValue = numbersMap.maxBy(_._2) // Get the pair with the maximum value  
  |     println(s"Minimum Value in Map: ${minValue._2}")  
  |     println(s"Maximum Value in Map: ${maxValue._2}")  
  |   }  
  | }  
defined object MapMinMax  
  
scala> MapMinMax.main(null)  
Minimum Value in Map: 1  
Maximum Value in Map: 5
```

RESULT

Thus the program is executed and the output was verified.

Ex. No: 13

**DATAFRAME AND GROUP BY OPERATIONS IN
SCALA**

Date:

AIM

To write a Scala program to create a dataframe and perform group by operations with summer olympic data.

ALGORITHM

Step 1: Open terminal.

Step 2: Navigate to the home directory: `cd $home`.

Step 3: Change directory to Spark installation: `cd spark-2.1.1-bin-hadoop2.7`.

Step 4: Start Spark Shell: `bin/spark-shell`.

Step 5: Read the dataset `olympics.csv` using Spark's DataFrame API with inferred schema and proper delimiters.

Step 6: Filter and count occurrences of "Gold" and "Silver" medals grouped by city.

Step 7: Group and count records by city, medal type, and year.

Step 8: Group and count records by city, medal type, and gender.

Step 9: Filter only female athletes and group the count by city and medal type.

Step 10: Display the results using `.show()`.

PROGRAM

Have to create **olympics.csv** file

```
val df2 = spark.read.option("inferSchema", "true").option("delimiter", ",").option("header",  
"true").csv("/home/grg/olymbics.csv")  
df2.filter($"Medal" === "Silver" || $"Medal" === "Gold").groupBy($"City").count().show()  
df2.groupBy($"City", $"Medal", $"Year").count().show()  
df2.groupBy($"City", $"Medal", $"Sex").count().show()  
df2.filter($"Sex" === "F").groupBy($"City", $"Medal").count().show()
```

OUTPUT

```
scala> df2.filter($"Medal" === "Silver" || $"Medal" === "Gold").groupBy($"City").count().show()
+-----+-----+
|      City|count|
+-----+-----+
|   Beijing| 1338|
|   Grenoble| 136|
| Stockholm| 664|
| Squaw Valley| 93|
| Los Angeles| 1393|
| Sarajevo| 148|
| Oslo| 86|
| Berlin| 613|
| Sochi| 398|
| Chamonix| 93|
| London| 2399|
| St. Louis| 332|
| Sydney| 1323|
| Tokyo| 677|
| Montreal| 866|
| Mexico City| 692|
| Paris| 981|
| Albertville| 211|
| Roma| 596|
| Lillehammer| 218|
+-----+-----+
only showing top 20 rows
```

```
scala> df2.groupBy($"City", $"Medal", $"Year").count().show()
+-----+-----+-----+-----+
|      City|      Medal|      Year|count|
+-----+-----+-----+-----+
| Summer|Rowing Men's Sing...|1952 Summer| 1|
| Summer|Canoeing Men's Ka...|1964 Summer| 1|
| Summer|Wrestling Men's L...|1968 Summer| 1|
| Summer|Athletics Men's 1...|2012 Summer| 1|
| Summer|Swimming Women's ...|1936 Summer| 1|
| Summer|Athletics Men's 4...|1968 Summer| 1|
| Winter|Speed Skating Wom...|1976 Winter| 1|
| Summer|Athletics Men's 4...|1988 Summer| 1|
| Grenoble|Silver|1968| 70|
| Summer|Football Men's Fo...|1912 Summer| 1|
| Summer|Athletics Men's D...|1968 Summer| 1|
| Winter|Alpine Skiing Wom...|1952 Winter| 1|
| Summer|Swimming Men's 20...|1996 Summer| 1|
| Summer|Athletics Women's...|1992 Summer| 1|
| Summer|Equestrianism Men...|1952 Summer| 2|
| Albertville|Bronze|1992| 106|
| Summer|Equestrianism Men...|1912 Summer| 1|
| Summer|Cycling Men's Tea...|1956 Summer| 1|
| Summer|Athletics Men's D...|1960 Summer| 1|
| Summer|Athletics Men's 1...|1948 Summer| 1|
+-----+-----+-----+-----+
only showing top 20 rows
```

```
scala> df2.groupBy($"City", $"Medal", $"Sex").count().show()
+-----+-----+-----+
| City | Medal | Sex | count |
+-----+-----+-----+
| Beijing | Silver | M | 360 |
| Montreal | Bronze | M | 317 |
| Oslo | Silver | F | 6 |
| Winter | Luge Mixed (Men)'... | Jr. | 3 |
| Summer | Handball Men's Ha... | Jr. | 3 |
| 1900 | Golf | Wright | 1 |
| Summer | Boxing Men's Ligh... | Jr. | 1 |
| Vancouver | NA | F | 1618 |
| Mexico City | Bronze | M | 285 |
| Summer | Equestrianism Mix... | III | 1 |
| Summer | Swimming Women's ... | -Greer | 1 |
| Rio de Janeiro | Silver | M | 335 |
| Moskva | NA | F | 1322 |
| Garmisch-Partenkl... | Silver | F | 3 |
| Summer | Gymnastics Women'... | -Zuschneid | 1 |
| Los Angeles | NA | M | 8047 |
| Stockholm | Gold | M | 328 |
| Melbourne | NA | M | 3228 |
| Berlin | Bronze | M | 263 |
| Summer | Rowing Men's Doub... | Jr. | 5 |
+-----+-----+-----+
only showing top 20 rows
```

```
scala> df2.filter($"Sex" === "F").groupBy($"City", $"Medal").count().show()
+-----+-----+
| City | Medal | count |
+-----+-----+
| Sankt Moritz | NA | 143 |
| Mexico City | Gold | 75 |
| Sankt Moritz | Gold | 7 |
| Lillehammer | Bronze | 35 |
| St. Louis | Gold | 6 |
| Garmisch-Partenkl... | Silver | 3 |
| London | Bronze | 354 |
| Amsterdam | NA | 305 |
| Stockholm | Bronze | 12 |
| Antwerpen | Bronze | 14 |
| Sapporo | Bronze | 14 |
| Albertville | Gold | 33 |
| Cortina d'Ampezzo | Silver | 9 |
| Sochi | NA | 1758 |
| Paris | Silver | 20 |
| Stockholm | Gold | 10 |
| Beijing | Bronze | 319 |
| Helsinki | Bronze | 46 |
| Seoul | Bronze | 191 |
| Moskva | NA | 1322 |
+-----+-----+
only showing top 20 rows
```

Activate Windows

RESULT

Thus the program is executed and the output was verified.

Ex. No: 14	KEYSPACE IN CASSANDRA USING JAVA API
Date:	

AIM:

To write a Program to create and use a keyspace in cassandra using Java API.

ALGORITHM

Step 1: Open terminal.

Step 2: Ensure Cassandra is running.

Step 3: Install necessary dependencies: `sudo apt install python3-pip`.

Step 4: Install the Cassandra driver: `pip3 install cassandra-driver`.

Step 5: Create a Python script (`create_keyspace.py`) containing the keyspace creation logic.

Step 6: Run the script: `python3 create_keyspace.py`.

Step 7: Verify that the keyspace has been created in Cassandra.

Step 8: Exit the terminal session if necessary: `exit`.

PROGRAM

In This PC

Open C: -> Cassandra -> bin

Command

Cassandra.bat

In notepad create check.java

```
import java.io.IOException;

public class check {

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

        String cmd = "C:\\apache-cassandra-3.11.9\\bin\\cqlsh.bat -e "
        + "\"CREATE KEYSPACE IF NOT EXISTS mykeyspace "
        + "WITH replication = {'class':'SimpleStrategy','replication_factor':1};\"";

        Runtime.getRuntime().exec(cmd);

        System.out.println("Keyspace Created Successfully");

    }

}
```

Open C: -> Cassandra -> bin -> type cmd

Command

cqlsh.py

DESCRIBE KEYSPACES;

OUTPUT

```
C:\Windows\System32\cmd.e  x  +  v
Microsoft Windows [Version 10.0.26100.4946]
(c) Microsoft Corporation. All rights reserved.

C:\Users\1MSCCS15\Desktop>javac keySpace.java
Note: keySpace.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Users\1MSCCS15\Desktop>java keySpace
Keyspace Created Successfully
```

```
C:\Windows\System32\cmd.e  x  +  v
Microsoft Windows [Version 10.0.26100.4946]
(c) Microsoft Corporation. All rights reserved.

C:\apache-cassandra-3.11.9\bin>cqlsh.bat

WARNING: console codepage must be set to cp65001 to support utf-8 encoding
If you experience encoding problems, change your console codepage with 'ch
cqlsh.

Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.9 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing.  Install to enable tab completion.
cqlsh> describe keyspaces;

system_schema  system          system_distributed
system_auth    mykeyspace      system_traces
```

RESULT

Thus the program is executed and the output is verified.

Ex. No: 15

Date:

ADDING A COLUMN IN CASSANDRA USING JAVA API

AIM

To write a program to create a column to an existing table in Cassandra using Java API.

ALGORITHM

Step 1: Navigate to the Documents directory and enter the cass_programs folder.

Step 2: Create a virtual environment and activate it.

Step 3: Create a Python script (add_col.py) to manage Cassandra database operations.

Step 4: Open a terminal and connect to the Cassandra container using Docker.

Step 5: List the available keyspaces to verify the database structure.

Step 6: Switch to the my_kspace keyspace to perform operations.

Step 7: Check if the employees table exists and view its current structure.

Step 8: Run the add_col.py script to create the table, insert data, and add a new column.

Step 9: Query the employees table to verify that the new column has been added and updated.

Step 10: Exit the Cassandra shell and close the connection.

Step 11: Deactivate the virtual environment if necessary.

PROGRAM

In This PC

Open C: -> Cassandra -> bin

Command

Cassandra.bat

In notepad create db.java

```
import java.io.*;

public class CreateTable {

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

        String cmd = "C:\\\\apache-cassandra-3.11.9\\\\bin\\\\cqlsh.bat -e "
        + "\"USE mykeyspace; "
        + "CREATE TABLE IF NOT EXISTS student (id int PRIMARY KEY, name text); "
        + "ALTER TABLE student ADD age int;\"";

        Runtime.getRuntime().exec(cmd);

        System.out.println("Table Created AND Column Added Successfully");

    }

}
```

Open C: -> Cassandra -> bin -> type cmd

Command

cqlsh.py

USE mykeyspace;

DESCRIBE TABLE student;

OUTPUT

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.26100.4946]
(c) Microsoft Corporation. All rights reserved.

C:\Users\1MSCCS15\Desktop>javac keySpace.java
Note: keySpace.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Users\1MSCCS15\Desktop>java keySpace
Keyspace Created Successfully

C:\Users\1MSCCS15\Desktop>javac createTable.java
Note: createTable.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Users\1MSCCS15\Desktop>java createTable
Table Created AND Column Added Successfully

C:\Users\1MSCCS15\Desktop>
```

```
C:\Windows\System32\cmd.exe
cqlsh> use mykeyspace;
cqlsh:mykeyspace> describe table student;

CREATE TABLE mykeyspace.student (
  id int PRIMARY KEY,
  age int,
  name text
) WITH bloom_filter_fp_chance = 0.01
   AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
   AND comment = ''
   AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
   AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
   AND crc_check_chance = 1.0
   AND dclocal_read_repair_chance = 0.1
   AND default_time_to_live = 0
   AND gc_grace_seconds = 864000
   AND max_index_interval = 2048
   AND memtable_flush_period_in_ms = 0
   AND min_index_interval = 128
   AND read_repair_chance = 0.0
   AND speculative_retry = '99PERCENTILE';

cqlsh:mykeyspace> select * from student;

 id | age | name
----+----+-----
(0 rows)
cqlsh:mykeyspace>
```

RESULT

Thus the program is executed and output is verified successfully

Ex. No: 16	WORD COUNT USING SCALA
Date:	

AIM:

To write a program to create a column to an existing table in Cassandra using Java API.

ALGORITHM

Step 1: Open the terminal.

Step 2: Navigate to the Spark installation directory using `cd spark-2.1.1-bin-hadoop2.7`.

Step 3: Start the Spark shell in Scala mode by running `bin/spark-shell`.

Step 4: Load the text file (data101.txt) from the specified path into an RDD using `sc.textFile("/home/grg/data101.txt")`.

Step 5: Split each line of the text file into individual words using the `flatMap` transformation.

Step 6: Map each word to a key-value pair where the key is the word and the value is 1, representing its occurrence.

Step 7: Use the `reduceByKey` transformation to sum up the values for each word, effectively counting the occurrences of each unique word.

Step 8: Collect the final word count results from the RDD and display them using `collect()`.

Step 9: Analyze the output to verify the word frequency distribution in the given text file.

Step 10: Stop the program.

PROGRAM

For this program requires **word.txt** file

```
val a = sc.textFile("/home/grg/word.txt")
val splitdata = a.flatMap(line => line.split(" "))
val napdata = splitdata.map(word => (word, 1))
val reducedata = napdata.reduceByKey(_ + _)
reducedata.collect()
```

OR

```
val a = sc.textFile("/home/grg/word.txt")
val splitdata = a.flatMap(line => line.split(" "))
val napdata = splitdata.map(word => (word, 1))
val reducedata = napdata.reduceByKey(_ + _)
reducedata.collect().foreach(println) // Print the results to the console
```

OUTPUT

```
scala> val a = sc.textFile("/home/grg/word.txt")
a: org.apache.spark.rdd.RDD[String] = /home/grg/word.txt MapPartitionsRDD[41] at textFile at <console>:24

scala> val splitdata = a.flatMap(line => line.split(" "))
splitdata: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[42] at flatMap at <console>:26

scala> val napdata = splitdata.map(word => (word, 1))
napdata: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[43] at map at <console>:28

scala> val reducedata = napdata.reduceByKey(_ + _)
reducedata: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[44] at reduceByKey at <console>:30

scala> reducedata.collect().foreach(println) // Print the results to the console
(hello,3)
(welcome,1)
(hi,3)
```

RESULT

Thus the program is executed successfully and the output was verified.

Ex. No: 17	FREQUENCY CALCULATION USING MAPREDUCE
Date:	

AIM

To write a simple YARN client App to develop a MapReduce program to calculate the frequency of a given word in the given file.

ALGORITHM

Step 1: Create a new text file named frequency.txt using the touch command.

Step 2: Open mapper3.py in a text editor using nano mapper3.py and write the mapper script.

Step 3: Open reducer3.py in a text editor using nano reducer3.py and write the reducer script.

Step 4: Add text content to frequency.txt, which will be processed using MapReduce.

Step 5: Display the content of frequency.txt using cat frequency.txt to verify the input data.

Step 6: Run the MapReduce process by passing the content of frequency.txt through mapper3.py,

then piping the output to reducer3.py, and storing the final result in result.txt.

Step 7: Display the word count results using cat result.txt to verify the output.

Step 8: Stop the program.

PROGRAM

TERMINAL

```
touch frequency.txt
nano mapper3.py
nano reducer3.py
cat frequency.txt
cat frequency.txt | python3 mapper3.py | python3 reducer3.py > result.txt
cat result.txt
```

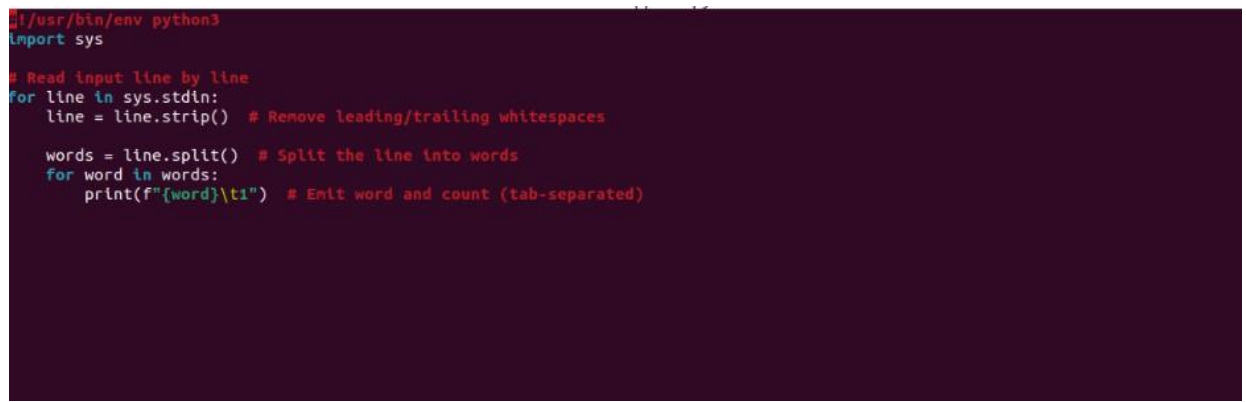
CREATE INTO frequency.txt ——TEXT EDITOR

MapReduce is a parallel computing paradigm and software framework for processing large datasets across many computers, designed for tasks like data extraction, transformation, and loading. It works by dividing the job into two key phases: "Map" and "Reduce".

INSERT INTO mapper3.py

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

# Read input line by line
for line in sys.stdin:
    line = line.strip() # Remove leading/trailing whitespaces
    words = line.split() # Split the line into words
    for word in words:
        print(f"{word}\t1") # Emit word and count (tab-separated)
```

A screenshot of a terminal window with a dark background. The text is color-coded: blue for comments, green for variable names, and red for function names and print statements. The code is the same as shown in the previous block.

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


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

    words = line.split() # Split the line into words
    for word in words:
        print(f"{word}\t1") # Emit word and count (tab-separated)
```

INSERT INTO reducer.py

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

word_counts = {} # Dictionary to store word counts
# Read input from mapper
for line in sys.stdin:
    word, count = line.strip().split("\t") # Split word and count
    count = int(count) # Convert count to integer
# Aggregate counts
if word in word_counts:
    word_counts[word] += count
else:
    word_counts[word] = count
# Print final word frequencies
for word, count in word_counts.items():
    print(f"{word}\t{count}")
```



The screenshot shows a terminal window with the nano 6.2 text editor open. The file being edited is named 'reducer3.py'. The code in the file is identical to the code block above, but with syntax highlighting: comments are in red, keywords like 'for', 'if', 'else', and 'print' are in blue, and strings are in green. The terminal background is dark purple.

```
GNU nano 6.2 reducer3.py
#!/usr/bin/env python3
import sys

word_counts = {} # Dictionary to store word counts

# Read input from mapper
for line in sys.stdin:
    word, count = line.strip().split("\t") # Split word and count
    count = int(count) # Convert count to integer

# Aggregate counts
if word in word_counts:
    word_counts[word] += count
else:
    word_counts[word] = count

# Print final word frequencies
for word, count in word_counts.items():
    print(f"{word}\t{count}")
```

OUTPUT

```
grg@LAB4-1:~$ touch frequency.txt
grg@LAB4-1:~$ nano mapper3.py
grg@LAB4-1:~$ nano reducer3.py
grg@LAB4-1:~$ cat frequency.txt
grg@LAB4-1:~$ cat frequency.txt
MapReduce is a parallel computing paradigm and software framework for processing large datasets across many computers, designed for
tasks like data extraction, transformation, and loading. It works by dividing the job into two key phases: "Map" and "Reduce"
grg@LAB4-1:~$ cat frequency.txt | python3 mapper3.py | python3 reducer3.py > result.txt
```

```
grg@LAB4-1:~$ cat result.txt
```

```
MapReduce      1
is              1
a              1
parallel       1
computing       1
paradigm        1
and             3
software        1
framework       1
for            2
processing      1
large           1
datasets        1
across          1
many            1
computers,      1
designed        1
tasks           1
like            1
data            1
extraction,     1
transformation, 1
loading.        1
It              1
works           1
by              1
dividing        1
the             1
job             1
into            1
two             1
key             1
phases:         1
"Map"           1
"Reduce"        1
```

RESULT

Thus the program is executed and the output was verified.

Ex. No: 18	DISPLAY CHART USING ZEPPELIN
Date:	

AIM

To analyze the cybercrime data set and display the results in charts using Zeppelin.

ALGORITHM

Step 1: Open Zeppelin Notebook and create a new notebook.

Step 2: Import necessary Spark SQL libraries in a new paragraph.

Step 3: Define the schema with column names: id, name, crime_type, gender, and crime_location.

Step 4: Read the dataset crimedata.txt from the specified location using sc.textFile.

Step 5: Convert the raw data into an RDD of Rows, ensuring each row has exactly five elements.

Step 6: Initialize SQLContext and create a DataFrame using the schema and transformed RDD.

Step 7: Register the DataFrame as a temporary table named cybercrime.

Step 8: Run an SQL query using Spark SQL to count occurrences of each crime_type.

Step 9: Display the results using .show().

Step 10: Add a new paragraph in Zeppelin using %sql and execute a similar SQL query to group crimes by type and count them.

Step 11: Stop the program.

ZEPPELIN SETUP

```
export JAVA_HOME=/home/grg/jdk1.8.0_45
export HADOOP_HOME=/home/grg/hadoop-2.9.0
export SPARK_HOME=/home/grg/spark-2.1.1-bin-hadoop2.7
export PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:$HIVE_HOME/bin:$PATH
#export JAVA_HOME=/usr/lib/jpm/java-8-openjdk_amd64
export PIG_HOME=/home/grg/pig-0.12.0
export HADOOP_HOME=/home/grg/hadoop-2.9.1
export HIVE_HOME=/home/grg/hive-0.12.0
export PATH=$HIVE_HOME/bin:$PATH
```

search firefox :<http://localhost:8080/>

```
grg@LAB4-1: ~/spark-2.1.1-bin-hadoop2.7
grg@LAB4-1: ~/zeppelin-0.8.2-bin-all
grg@LAB4-1: ~/spark-2.1.1-bin-hadoop2.7$ cd $HOME
grg@LAB4-1: $ cd zeppelin-0.8.2-bin-all
bash: cd: zeppelin-0.8.2-bin-all: No such file or directory
grg@LAB4-1: $ cd zeppelin-0.8.2-bin-all
grg@LAB4-1: ~/zeppelin-0.8.2-bin-all$ bin/zeppelin-daemon.sh start
Zeppelin start [ OK ]
Zeppelin process died [ FAILED ]
grg@LAB4-1: ~/zeppelin-0.8.2-bin-all$ sudo bin/zeppelin-daemon.sh restart
[sudo] password for grg:
Zeppelin stop [ OK ]
Zeppelin start [ OK ]
grg@LAB4-1: ~/zeppelin-0.8.2-bin-all$
```

create **crimedata.txt** — create new file in texteditor

1,John,Phishing,Male,Delhi
2,Alice,Hacking,Female,Mumbai
3,Rahul,Identity Theft,Male,Bangalore
4,Sneha,Phishing,Female,Delhi
5,Amit,Online Fraud,Male,Chennai
6,Priya,Hacking,Female,Bangalore
7,Rohit,Phishing,Male,Mumbai
8,Neha,Online Fraud,Female,Delhi
9,Karan,Identity Theft,Male,Pune
10,Anjali,Hacking,Female,Chennai

CODE

```
import org.apache.spark.sql.{Row, SQLContext};
import org.apache.spark.sql.types.{StructType, StructField, StringType}
val schemaString = "id, name, crime_type, gender, crime_location"
val schema = StructType(
  schemaString.split(",").map(fieldName =>
    StructField(fieldName.trim, StringType, true)
  )
)
```

```

val cybercrimeData = sc.textFile("file:///home/grg/crimeData.txt")
val rowRDD = cybercrimeData.map(_ .split(",")).flatMap(p =>
if (p.length == 5) {
Some(Row(p(0).trim, p(1).trim, p(2).trim, p(3).trim, p(4).trim))
} else {
None
}
)

val sqlContext = new SQLContext(sc)
val cybercrimeDF = sqlContext.createDataFrame(rowRDD, schema)
cybercrimeDF.createOrReplaceTempView("cybercrime")
val crimeTypeCountsDF = sqlContext.sql(
""""SELECT crime_type, COUNT(*) AS count
FROM cybercrime GROUP
BY crime_type""""
)

crimeTypeCountsDF.show()

```

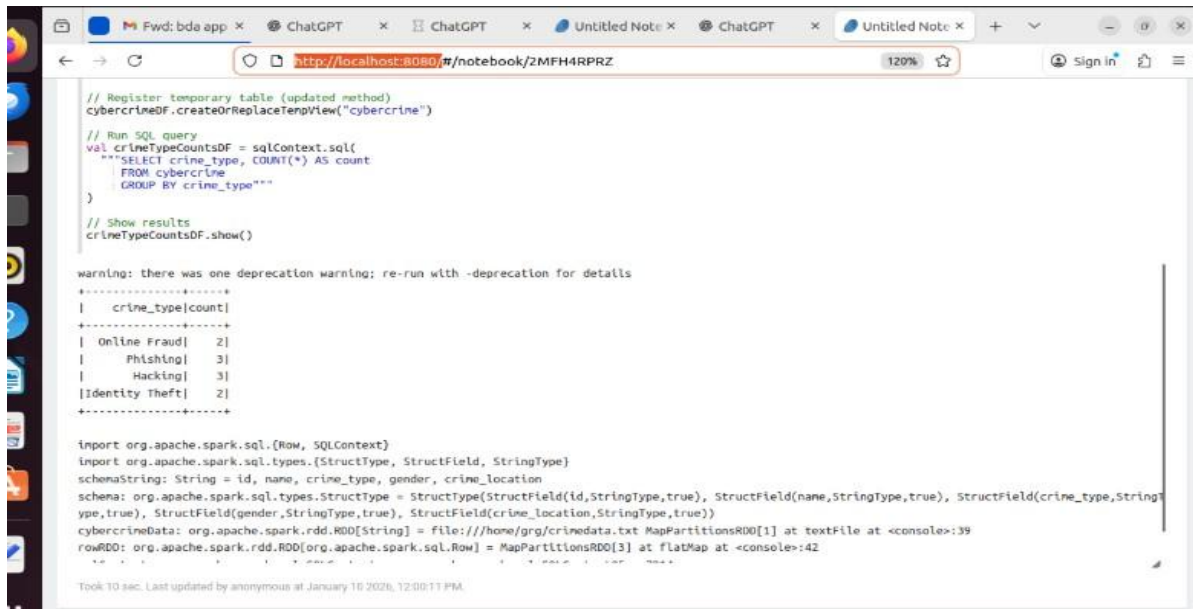
To view the Output:

```

%sql
SELECT crime_type, COUNT(*) AS crime_count
FROM cybercrime
GROUP BY crime_type

```

OUTPUT



The screenshot shows a Jupyter Notebook interface with a browser window at `http://localhost:8080/#/notebook/2MFH4RPZ`. The code in the notebook is as follows:

```
// Register temporary table (updated method)
cybercrimeDF.createOrReplaceTempView("cybercrime")

// Run SQL query
val crimeTypeCountsDF = sqlContext.sql(
  """SELECT crime_type, COUNT(*) AS count
    FROM cybercrime
    GROUP BY crime_type"""
)

// Show results
crimeTypeCountsDF.show()
```

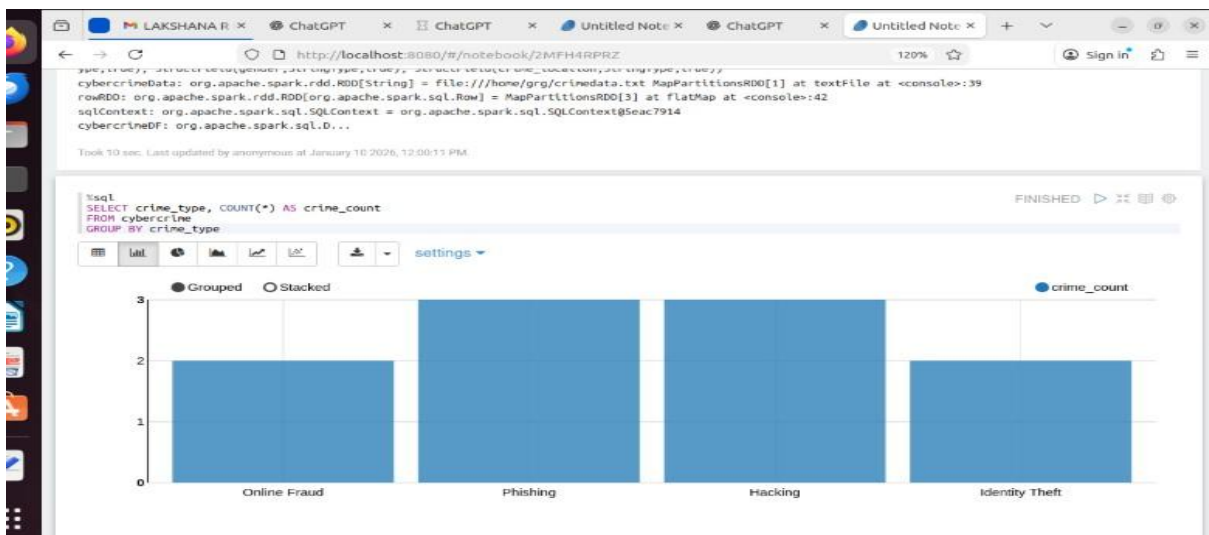
The output of the code is a warning message and a table of crime types and their counts:

```
warning: there was one deprecation warning; re-run with -deprecation for details
+-----+
| crime_type|count|
+-----+
| Online Fraud| 2|
| Phishing| 3|
| Hacking| 3|
| Identity Theft| 2|
+-----+
```

Below the table, there is a block of Scala code defining the schema and data for the `cybercrime` DataFrame:

```
import org.apache.spark.sql.{Row, SQLContext}
import org.apache.spark.sql.types.{StructType, StructField, StringType}
schemaString: String = id, name, crime_type, gender, crime_location
schema: org.apache.spark.sql.types.StructType = StructType(StructField(id,StringType,true), StructField(name,StringType,true), StructField(crime_type,StringType,true), StructField(gender,StringType,true), StructField(crime_location,StringType,true))
cybercrimeData: org.apache.spark.rdd.RDD[String] = file:///home/grg/crime_data.txt MapPartitionsRDD[1] at textFile at <console>:39
rowRDD: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[3] at flatMap at <console>:42
```

The bottom of the screenshot shows the execution time: "Took 10 sec. Last updated by anonymous at January 10 2026, 12:00:11 PM."



RESULT

Thus the program is executed and the output was verified.

Ex. No: 19

Date:

INTERACTIVE QUERIES USING ZEPPELIN

AIM

To develop interactive queries and prepare charts using Zeppelin.

ALGORITHM

Step 1: Open the terminal and navigate to the Zeppelin installation directory.

Step 2: Start Zeppelin using `bin/zeppelin-daemon.sh start`.

Step 3: If the setup fails, check running processes with `ps -ef | grep "zeppelin"` kill any existing Zeppelin processes, and restart using `sudo bin/zeppelin-daemon.sh restart`.

Step 4: Create a text file named `datagen.txt` and insert patient data in CSV format.

Step 5: Open Zeppelin Notebook and create a new notebook.

Step 6: Define a schema with column names: `id`, `name`, `dName`, `gender`, and `amount`.

Step 7: Import required Spark SQL libraries and read `datagen.txt` using `sc.textFile`.

Step 8: Define the schema using `StructType` and map the CSV data to Rows.

Step 9: Create a `DataFrame` from the RDD and schema.

Step 10: Register the `DataFrame` as a temporary table named `patient1`.

Step 11: Add a new paragraph in Zeppelin using `%sql` magic and execute a query to sum amount for each `dName`.

Step 12: Stop the program.

PROGRAM

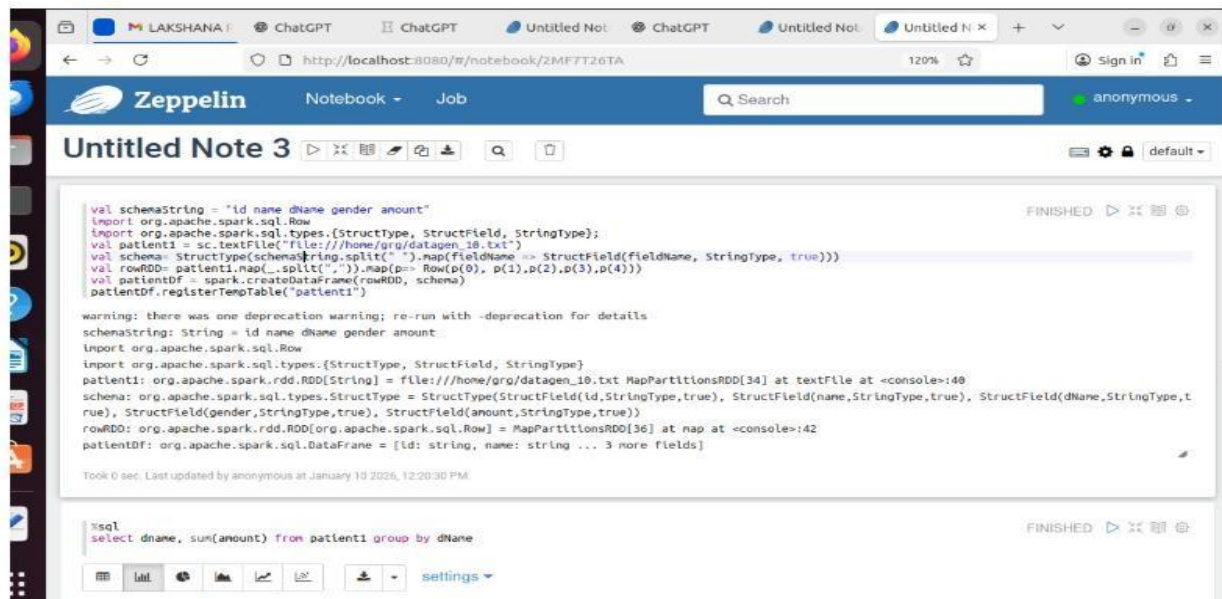
It can need **datagen_10.txt**

```
val schemaString = "id name dName gender amount"
import org.apache.spark.sql.Row
import org.apache.spark.sql.types.{StructType, StructField, StringType};
val patient1 = sc.textFile("file:///home/grg/datagen_10.txt")
val schema= StructType(schemaString.split(" ").map(fieldName => StructField(fieldName,
StringType, true)))
val rowRDD= patient1.map(_ .split(",")).map(p=> Row(p(0), p(1),p(2),p(3),p(4)))
val patientDf = spark.createDataFrame(rowRDD, schema)
patientDf.registerTempTable("patient1")
```

To view the Output

```
%sql
select dname, sum(amount) from patient1 group by dName
```

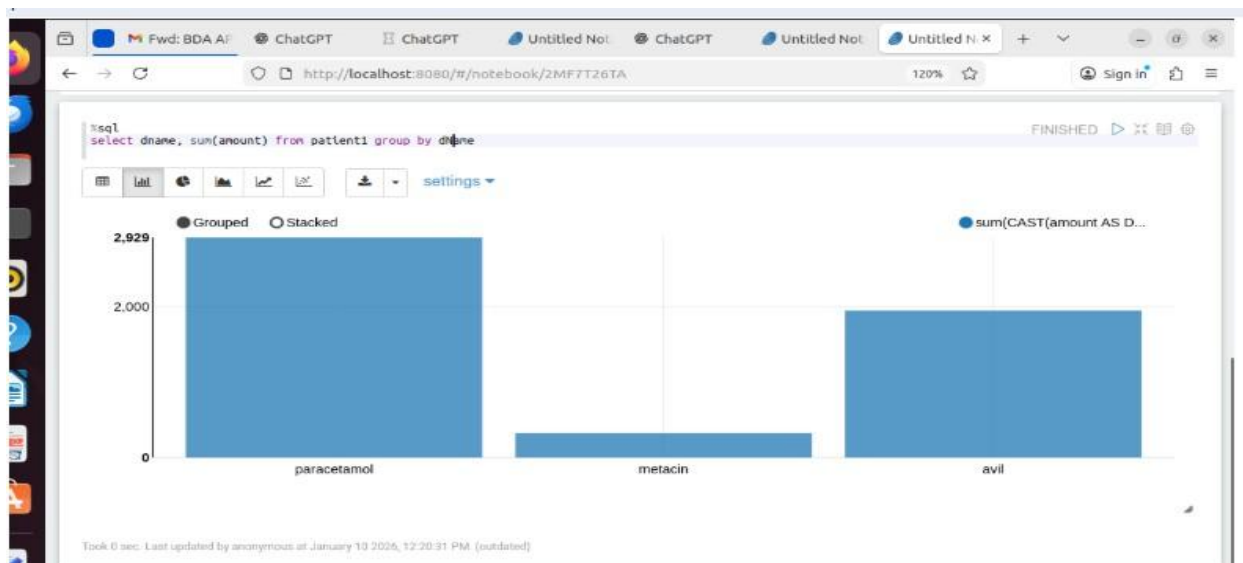
OUTPUT



```
val schemaString = "id name dName gender amount"
import org.apache.spark.sql.Row
import org.apache.spark.sql.types.{StructType, StructField, StringType}
val patient1 = sc.textFile("file:///home/gro/dataseten_10.txt")
val schema = StructType(schemaString.split(" ").map(fieldName => StructField(fieldName, StringType, true)))
val rowRDD = patient1.map(_.split(",")).map(p => Row(p(0), p(1), p(2), p(3), p(4)))
val patientDF = spark.createDataFrame(rowRDD, schema)
patientDF.registerTempTable("patient1")

warning: there was one deprecation warning; re-run with -deprecation for details
schemaString: String = id name dName gender amount
import org.apache.spark.sql.Row
import org.apache.spark.sql.types.{StructType, StructField, StringType}
patient1: org.apache.spark.rdd.RDD[String] = file:///home/gro/dataseten_10.txt MapPartitionsRDD[34] at textFile at <console>:40
schema: org.apache.spark.sql.types.StructType = StructType(StructField(id,StringType,true), StructField(name,StringType,true), StructField(dName,StringType,true), StructField(gender,StringType,true), StructField(amount,StringType,true))
rowRDD: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[36] at map at <console>:42
patientDF: org.apache.spark.sql.DataFrame = [id: string, name: string ... 3 more fields]
```

Took 0 sec. Last updated by anonymous at January 10 2026, 12:20:30 PM.



RESULT

Thus the program is executed successfully and the output was verified.

Ex. No: 20	STUDENT GRADE ANALYSIS USING SCALA
Date:	

AIM

To develop a scala application to generate student grade analysis.

ALGORITHM

Step 1: Open terminal and navigate to home directory using `cd $home`.

Step 2: Source the `.bashrc` file using `source .bashrc`.

Step 3: Navigate to Spark directory using `cd spark-2.1.1-bin-hadoop2.7`.

Step 4: Start Spark shell using `bin/spark-shell`.

Step 5: Define `StudentGradeApp` object in Scala environment.

Step 6: Create a `Student` class with attributes `studentId`, `name`, `department`, and `grade`.

Step 7: Implement `displayDetails()` method to print student details.

Step 8: Define `averageGrade()` function to calculate the average grade.

Step 9: Define `highestGrade()` function to find the highest-scoring student.

Step 10: Define `lowestGrade()` function to find the lowest-scoring student.

Step 11: Implement `main()` function to create and analyze student data.

Step 12: Execute `StudentGradeApp.main(Array())` to display the results.

PROGRAM

```
object StudentGradeAnalysis {

  case class Student(name: String, grade: Int) val
  students = List(
    Student("Alice", 85),
    Student("Bob", 72),
    Student("Charlie", 90),
    Student("David", 68),
    Student("Eva", 95),
    Student("Frank", 77),
    Student("Grace", 83)
  )

  def calculateAverageGrade(students: List[Student]): Double = {
    val totalGrades = students.map(_grade).sum
    totalGrades.toDouble / students.size
  }

  def calculateHighestGrade(students: List[Student]): Student = {
    students.maxBy(_grade)
  }

  def calculateLowestGrade(students: List[Student]): Student = {
    students.minBy(_grade)
  }

  def generateGradeReport(students: List[Student]): Unit = { val
    average = calculateAverageGrade(students)
    val highest = calculateHighestGrade(students) val
```

```
lowest = calculateLowestGrade(students)
println(s"Total Students: ${students.size}")
println(s"Average Grade: %.2f".format(average))
println(s"Highest Grade: ${highest.name} with ${highest.grade}")
println(s"Lowest Grade: ${lowest.name} with ${lowest.grade}")
println("\nGrade Distribution:")
```

```
students.sortBy(_.grade).foreach(student => {
println(s"${student.name}: ${student.grade}")
})
}
```

```
def main(args: Array[String]): Unit = {
println("Student Grade Analysis\n")
generateGradeReport(students)
}
}
```

To view Output

```
StudentGradeAnalysis.main(null)
```

OUTPUT

```
scala> object StudentGradeAnalysis {
  case class Student(name: String, grade: Int)
  val students = List(
    Student("Alice", 85),
    Student("Bob", 72),
    Student("Charlie", 90),
    Student("David", 68),
    Student("Eva", 95),
    Student("Frank", 77),
    Student("Grace", 83)
  )
  def calculateAverageGrade(students: List[Student]): Double = {
    val totalGrades = students.map(_.grade).sum
    totalGrades.toDouble / students.size
  }
  def calculateHighestGrade(students: List[Student]): Student = {
    students.maxBy(_.grade)
  }
  def calculateLowestGrade(students: List[Student]): Student = {
    students.minBy(_.grade)
  }
  def generateGradeReport(students: List[Student]): Unit = {
    val average = calculateAverageGrade(students)
    val highest = calculateHighestGrade(students)
    val lowest = calculateLowestGrade(students)
    println(s"Total Students: ${students.size}")
    println(s"Average Grade: %.2f".format(average))
    println(s"Highest Grade: ${highest.name} with ${highest.grade}")
    println(s"Lowest Grade: ${lowest.name} with ${lowest.grade}")
    println("\nGrade Distribution:")
    students.sortBy(_.grade).foreach(student => {
      println(s"${student.name}: ${student.grade}")
    })
  }
  def main(args: Array[String]): Unit = {
    println("Student Grade Analysis\n")
    generateGradeReport(students)
  }
}
```

```
scala> StudentGradeAnalysis.main(null)
Student Grade Analysis

Total Students: 7
Average Grade: 81.43
Highest Grade: Eva with 95
Lowest Grade: David with 68

Grade Distribution:
David: 68
Bob: 72
Frank: 77
Grace: 83
Alice: 85
Charlie: 90
Eva: 95
```

RESULT

Thus the program is executed and the output was verified.