RN SHETTY TRUST®

RNS INSTITUTE OF TECHNOLOGY

Autonomous Institution, Affiliated to VTU, Recognized by GOK, Approved by AICTE,
(NAAC 'A+ Grade' Accredited, NBA Accredited (UG - CSE, ECE, ISE, EIE and EEE))
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098
Ph:(080)28611880,28611881 URL: www.rnsit.ac.in

DEPARTMENT OF CSE (DATA SCIENCE)

BIG DATA AND ANALYTICS LAB MANUAL

(BAD601)

(As per Visvesvaraya Technological University Course type- IPCC)

Compiled by

DEPARTMENT OF CSE (DATA SCIENCE)

R N S Institute of Technology

Bengaluru-98

Name: ______________________

USN: ______________________

RN SHETTY TRUST®

RNS INSTITUTE OF TECHNOLOGY

Autonomous Institution Affiliated to VTU, Recognized by GOK, Approved by AICTE
(NAAC 'A+ Grade' Accredited, NBA Accredited (UG - CSE, ECE, ISE, EIE and EEE))
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098
Ph:(080)28611880,28611881 URL: www.rnsit.ac.in

Estd : 2001

DEPARTMENT OF CSE (DATA SCIENCE)

VISION OF THE DEPARTMENT

Empowering students to solve complex real-time computing problems involving high volume multi-dimensional data.

MISSION OF THE DEPARTMENT

- Provide quality education in both theoretical and applied Computer Science to solve real world problems.
- Conduct research to develop algorithms that solve complex problems involving multi-dimensional high volume data through intelligent inferencing.
- Develop good linkages with industry and research organizations to expose students to global problems and find optimal solutions.
- Creating confident Graduates who can contribute to the nation through high levels of commitment following ethical practices and with integrity.

Disclaimer

Trademark

Estd : 2001

Edition: 2024- 25

Document Owner

The primary contact for questions regarding this document is:

Author(s):	1. Prof. Sunil G L
	2. Prof. Vidya Shirodkar
Department:	**CSE (Data Science)**
Contact email ids :	sunil.gl@rnsit.ac.in
	vidyashirodkar@rnsit.ac.in

COURSE OUTCOMES

Course Outcomes: At the end of this course, students are able to:
CO1- Identify and list various Big Data concepts, tools and applications.
CO2- Develop programs using HADOOP framework.
CO3- Make use of Hadoop Cluster to deploy Map Reduce jobs, PIG, HIVE and Spark programs.
CO4- Analyze the given data set and identify deep insights from the data set.
CO5- Demonstrate Text, Web Content and Link Analytics.

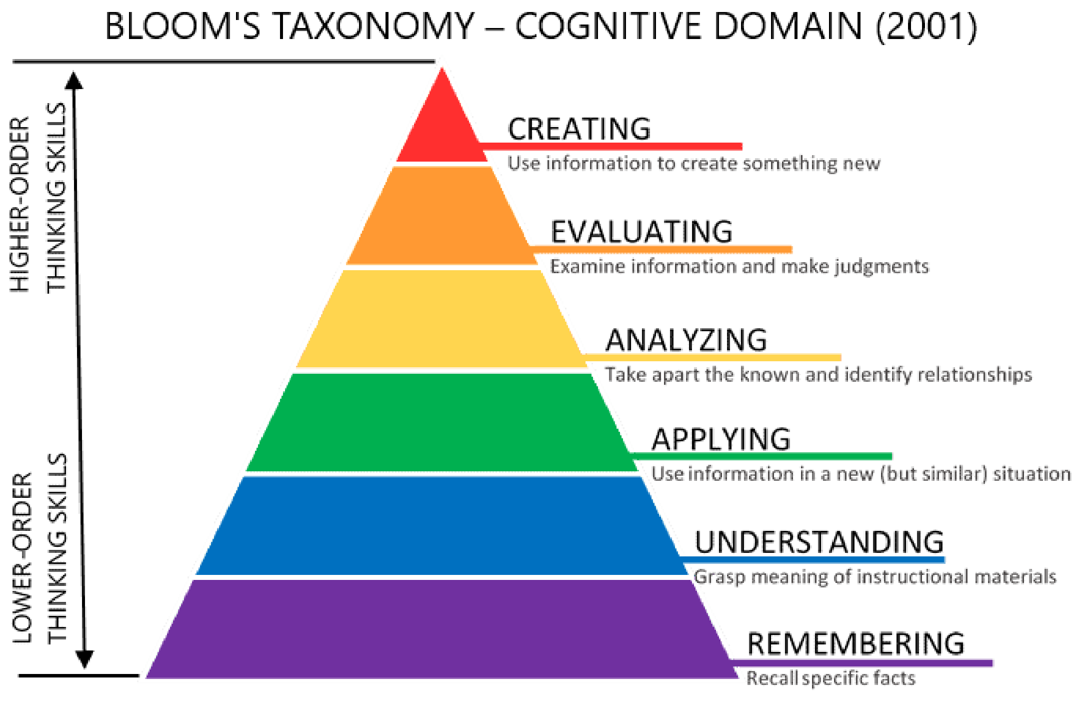
COs and POs Mapping of lab Component

COURSE OUTCOMES	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
CO1	3	2	2	1	3							2				
CO2	3	3	3	1	3							2				
CO3	3	3	3	1	3							2				
CO4	3	3	3	1	3							2				
	3	3	2	1	3							2				

Mapping of ‘Graduate Attributes’ (GAs) and ‘Program Outcomes’ (POs)

Graduate Attributes (GAs) (As per Washington Accord Accreditation)	**Program Outcomes (POs) (As per NBA New Delhi)**
Engineering Knowledge	Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems
Problem Analysis	Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
Design/Development of solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate considerations for the public health and safety and the cultural, societal and environmental consideration.
Conduct Investigation of complex problems	Use research – based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.
Modern Tool Usage	Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
The engineer and society	Apply reasoning informed by the contextual knowledge to assess society, health, safety, legal and cultural issues and the consequential responsibilities relevant to the professional engineering practice.
Environment and sustainability	Understand the impact of the professional engineering solutions in societal and environmental context and demonstrate the knowledge of and need for sustainable development.
Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
Individual and team work	Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary settings.
Communication	Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions.
Project management & finance	Demonstrate knowledge and understanding of the engineering and management principles and apply these to ones won work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
Life Long Learning	Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

REVISED BLOOMS TAXONOMY (RBT)


BLOOM'S TAXONOMY – COGNITIVE DOMAIN (2001)
HIGHER-ORDER THINKING SKILLS
LOWER-ORDER THINKING SKILLS
CREATING
Use information to create something new
EVALUATING
Examine information and make judgments
ANALYZING
Take apart the known and identify relationships
APPLYING
Use information in a new (but similar) situation
UNDERSTANDING
Grasp meaning of instructional materials
REMEMBERING
Recall specific facts

Rubrics:

Lab Marks (25)	
Record	05 (Min-2)
Observation	10 (Min-4)
Test	10 (Min-4)
Total	**25 (Min-10)**

PROGRAM LIST

Program 1:

AIM: Install Hadoop and implement the following file management tasks in Hadoop

Adding Files and Directories

Retrieving Files

Delete files and directories

Hint: A typical Hadoop workflow creates data files (such as log files) elsewhere and copies them into HDFS using one of the above command line utilities.

Source code:

Download and install Oracle virtual box & Cloudera and execute the following commands in terminal

1. **Create a directory**

   ```
   $mkdir Hadoop
   ```

2. **Create a file in a Hadoop directory of file name hello.txt**

   ```
   $cd Hadoop/
   $vi hello.txt
   ```

   (Type the file content and Press Esc : wq)

3. **Display the content of the File**

   ```
   $cat hello.txt
   ```

4. **Create directory in HDFS**

   ```
   $hdfs dfs -mkdir  /today
   ```

5. **Copy the file from Hadoop to hdfs**

   ```
   $hdfs dfs -put  hello.txt  /today/data.txt
   ```

6. **Copy the file within hdfs**

   ```
    $hdfs dfs -mkdir /input
   $hdfs dfs -cp  /today/data.txt /input/data.txt
   $hdfs dfs -cat /input/data.txt
   ```

7. **Copy the file from hdfs to Hadoop**

   ```
   $ hdfs dfs -get /input/data.txt
   ```

8. **Delete the files**

   ```
   $hdfs dfs -rm /today/data.txt
   ```

9. **Remove directory**

   ```
   $hdfs dfs -rmdir /today
   ```

Program 2:
AIM: Develop a Map Reduce program for Matrix Multiplication.

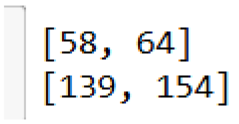
Source Code:

Execute in Jupiter/Spyder

```
from functools import reduce

# Function to create key-value pairs for the Map step
def mapper(matrix1, matrix2):
    # List to store key-value pairs
    key_value_pairs = []

    # Iterate over the rows of the first matrix
    for i in range(len(matrix1)):
        # Iterate over the columns of the second matrix
        for j in range(len(matrix2[0])):
            for k in range(len(matrix2)):
                # Create key-value pair (i, j) -> matrix1[i][k] * matrix2[k][j]
                key_value_pairs.append(((i, j), matrix1[i][k] * matrix2[k][j]))

    return key_value_pairs

# Function to sum values for the Reduce step
def reducer(key, values):
    return sum(values)

# Function to perform MapReduce
def map_reduce(matrix1, matrix2):
    # Apply the mapper function to create key-value pairs
    key_value_pairs = mapper(matrix1, matrix2)

    # Group values by key
    grouped_values = {}
    for key, value in key_value_pairs:
        if key not in grouped_values:
            grouped_values[key] = []
        grouped_values[key].append(value)

    # Apply the reducer function to get final results
    result = {}
    for key in grouped_values:
        result[key] = reducer(key, grouped_values[key])
    return result
```

```
# Function to convert result dictionary to matrix
def result_to_matrix(result, rows, cols):
    matrix = [[0 for _ in range(cols)] for _ in range(rows)]
    for key, value in result.items():
        matrix[key[0]][key[1]] = value
    return matrix

# Example matrices
matrix1 = [
    [1, 2, 3],
    [4, 5, 6]
]

matrix2 = [
    [7, 8],
    [9, 10],
    [11, 12]
]

# Perform MapReduce matrix multiplication
result = map_reduce(matrix1, matrix2)

# Convert result dictionary to matrix
result_matrix = result_to_matrix(result, len(matrix1), len(matrix2[0]))

# Print result matrix
for row in result_matrix:
    print(row)
```

Sample Output:



```
[58, 64]
[139, 154]
```

Program 3:

AIM: Develop a map reduce program that mines weather data and display appropriate message indicating the weather conditions of the day

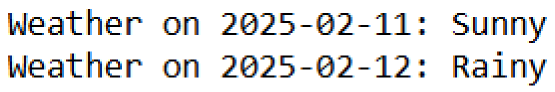
Source Code:

Execute in Jupiter/Spyder Environment

```
def mapper(line):
    date, condition = line.strip().split(',')
    return (date, condition)

def reducer(date, conditions):
    # Assuming conditions is a list of weather conditions for the given date
    condition_counts = {}
    for condition in conditions:
        if condition in condition_counts:
            condition_counts[condition] += 1
        else:
            condition_counts[condition] = 1

    # Determine the most frequent weather condition
    most_frequent_condition = max(condition_counts, key=condition_counts.get)
    return (date, most_frequent_condition)

def main(input_file):
    # Read input data
    with open(input_file, 'r') as f:
        lines = f.readlines()

    # Apply mapper function
    mapped_data = [mapper(line) for line in lines]

    # Group by date
    grouped_data = {}
    for date, condition in mapped_data:
        if date in grouped_data:
            grouped_data[date].append(condition)
        else:
            grouped_data[date] = [condition]

    # Apply reducer function
    reduced_data = {date: reducer(date, conditions) for date, conditions in
grouped_data.items()}

    # Display results
    for date, (date_key, condition) in reduced_data.items():
        print(f"Weather on {date_key}: {condition}")
```

```
# Example usage
if __name__ == "__main__":
    input_file = 'weather.txt'
    main(input_file)
```

Sample Output:



```
Weather on 2025-02-11: Sunny
Weather on 2025-02-12: Rainy
```

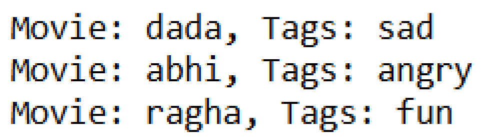
Program 4:
AIM: Develop a Map Reduce program to find the tags associated with each movie by analyzing movie lens data.

Source Code:

Execute in Jupiter/Spyder Environment

```
import csv
from functools import reduce

# Function to create key-value pairs for the Map step
def mapper(tags):
    key_value_pairs = []
    for row in tags:
        movie_id = row['movieId']
        tag = row['tag']
        key_value_pairs.append((movie_id, tag))
    return key_value_pairs

# Function to collect tags for the Reduce step
def reducer(key, values):
    return list(set(values))

# Function to perform MapReduce
def map_reduce(tags):
    # Apply the mapper function to create key-value pairs
    key_value_pairs = mapper(tags)

    # Group values by key (movieId)
    grouped_values = {}
    for key, value in key_value_pairs:
        if key not in grouped_values:
            grouped_values[key] = []
        grouped_values[key].append(value)

    # Apply the reducer function to get final results
    result = {}
    for key in grouped_values:
        result[key] = reducer(key, grouped_values[key])

    return result

# Function to load tags data from CSV file
def load_tags(file_path):
    tags = []
    with open(file_path, mode='r') as file:
        reader = csv.DictReader(file)
        for row in reader:
            tags.append(row)
    return tags

# Function to load movie titles from CSV file
def load_movies(file_path):
    movies = {}
    with open(file_path, mode='r') as file:
        reader = csv.DictReader(file)
        for row in reader:
            movies[row['movieId']] = row['title']
    return movies

# Load MovieLens data
tags = load_tags('tags.csv')
movies = load_movies('movies.csv')

# Perform MapReduce to find tags associated with each movie
result = map_reduce(tags)

# Display results
for movie_id, tags in result.items():
    movie_title = movies[movie_id]
    print(f'Movie: {movie_title}, Tags: {", ".join(tags)}')
```

OUTPUT:



```
Movie: dada, Tags: sad
Movie: abhi, Tags: angry
Movie: ragha, Tags: fun
```

Program 5:

AIM: Implement functions: Count-Sort-Limit-Skip- Aggregate using mango DB.

Source Code:

Install MongoDB on Fedora using below commands

```
$sudo dnf install -y mongodb mongodb-server
$sudo systemctl start mongod
$sudo systemctl enable mongod
$sudo systemctl status mongod
$sudo dnf install mongodb-mongosh
```

Start the **MongoDB shell** by running:

```
$mangosh
```

Select a database

```
$use testDB
```

Sample Data:

```
[
  { "_id": 1, "name": "Alice", "age": 22, "marks": 85 },
  { "_id": 2, "name": "Bob", "age": 21, "marks": 78 },
  { "_id": 3, "name": "Charlie", "age": 23, "marks": 92 },
  { "_id": 4, "name": "David", "age": 20, "marks": 88 },
  { "_id": 5, "name": "Eve", "age": 22, "marks": 76 }
]
```

Create students collection using above sample data

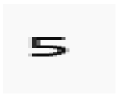
```
db.createCollection('Students');
db.student.insertOne({ "_id": 1, "name": "Alice", "age": 22, "marks": 85 })
```

(i) Count

Count the number of students age is greater than 20

```
$ db.student.count({ age: { $gt: 20 } })
```

Output:



```
5
```

ii) Sort

Sort documents by age in descending order

```
$ db.students.find().sort({ marks: -1 })
```

```
[
  { "_id": 3, "name": "Charlie", "age": 23, "marks": 92 },
  { "_id": 4, "name": "David", "age": 20, "marks": 88 },
  { "_id": 1, "name": "Alice", "age": 22, "marks": 85 },
  { "_id": 2, "name": "Bob", "age": 21, "marks": 78 },
  { "_id": 5, "name": "Eve", "age": 22, "marks": 76 }
]
```

iii) Limit

Limit the number of returned documents to 5:

$db.students.find().sort({ marks: -1 }).limit(3)

Output:

```
[
  { "_id": 3, "name": "Charlie", "age": 23, "marks": 92 },
  { "_id": 4, "name": "David", "age": 20, "marks": 88 },
  { "_id": 1, "name": "Alice", "age": 22, "marks": 85 }
]
```

iv)Skip

$db.students.find().sort({ marks: -1 }).skip(2)

Output:

```
[
  { "_id": 1, "name": "Alice", "age": 22, "marks": 85 },
  { "_id": 2, "name": "Bob", "age": 21, "marks": 78 },
  { "_id": 5, "name": "Eve", "age": 22, "marks": 76 }
]
```

iv) Aggregate Data

db.students.aggregate([{ $group: { _id: "$age", avgMarks: { $avg: "$marks" } } }])

```
[
  { "_id": 20, "avgMarks": 88 },
  { "_id": 21, "avgMarks": 78 },
  { "_id": 22, "avgMarks": 80.5 },
  { "_id": 23, "avgMarks": 92 }
]
```

Program 6:

AIM: Develop a Pig Latin Scripts to sort, group, join, Project and filter the data

Source Code:

1. Sample Data

Let's assume we have two datasets:

- employees.txt (Employee ID, Name, Age, Department ID, Salary)
- departments.txt (Department ID, Department Name)

employees.txt (stored in HDFS at /user/cloudera/employees.txt)

```
101,John,30,1,50000

102,Sam,28,2,60000

103,Anna,32,1,75000

104,David,29,3,62000

105,Lily,27,2,58000
```

departments.txt (stored in HDFS at /user/cloudera/departments.txt)

```
1,HR

2,Finance

3,IT
```

Pig Latin Script

Save the script as employee_analysis.pig and execute it in Cloudera.

```
-- Load the employees dataset

employees = LOAD 'hdfs://localhost:9000/user/cloudera/employees.txt'

USING PigStorage(',')

AS (emp_id:int, name:chararray, age:int, dept_id:int, salary:int);


-- Load the departments dataset
departments = LOAD 'hdfs://localhost:9000/user/cloudera/departments.txt'
USING PigStorage(',')
AS (dept_id:int, dept_name:chararray);

-- 1. FILTER: Select employees with age greater than 28
filtered_employees = FILTER employees BY age > 28;

-- 2. PROJECT: Select only emp_id, name, and salary columns
projected_employees = FOREACH filtered_employees GENERATE emp_id, name, salary;

-- 3. SORT: Order employees by salary in descending order
sorted_employees = ORDER projected_employees BY salary DESC;
```

```
-- 4. GROUP: Group employees by department ID
grouped_by_department = GROUP employees BY dept_id;

-- 5. JOIN: Join employees with department names using dept_id
joined_data = JOIN employees BY dept_id, departments BY dept_id;

-- STORE results in HDFS
STORE sorted_employees INTO 'hdfs://localhost:9000/user/cloudera/output/sorted_employees' USING
PigStorage(',');

STORE grouped_by_department INTO
'hdfs://localhost:9000/user/cloudera/output/grouped_by_department' USING PigStorage(',');
STORE joined_data INTO 'hdfs://localhost:9000/user/cloudera/output/joined_data' USING PigStorage(',');

-- DISPLAY the results on the screen
DUMP sorted_employees;
DUMP grouped_by_department;
DUMP joined_data;
```

Upload the data in HDFS

```
hdfs dfs -mkdir -p /user/cloudera
hdfs dfs -put employees.txt /user/cloudera/
hdfs dfs -put departments.txt /user/cloudera/
```

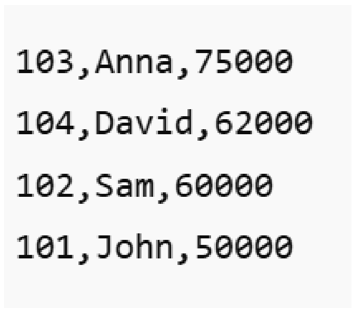
Run the Script

```
pig -x mapreduce employee_analysis.pig
```

ouput commands

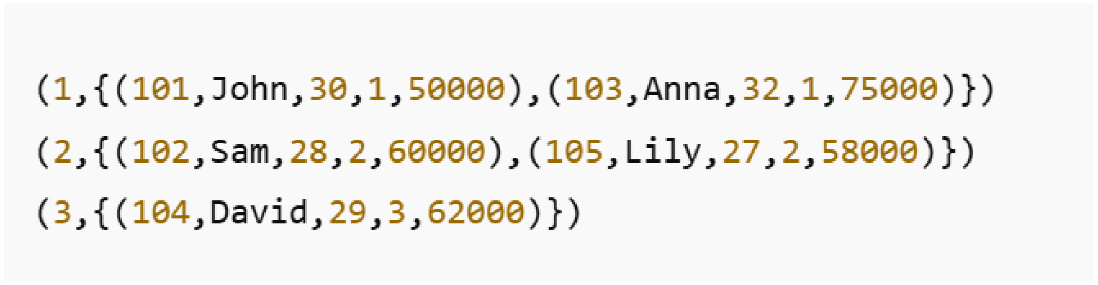
```
hdfs dfs -cat /user/cloudera/output/sorted_employees/part-r-00000
hdfs dfs -cat /user/cloudera/output/grouped_by_department/part-r-00000
hdfs dfs -cat /user/cloudera/output/joined_data/part-r-00000
```

OUTPUT :Sorted Employee by Salary



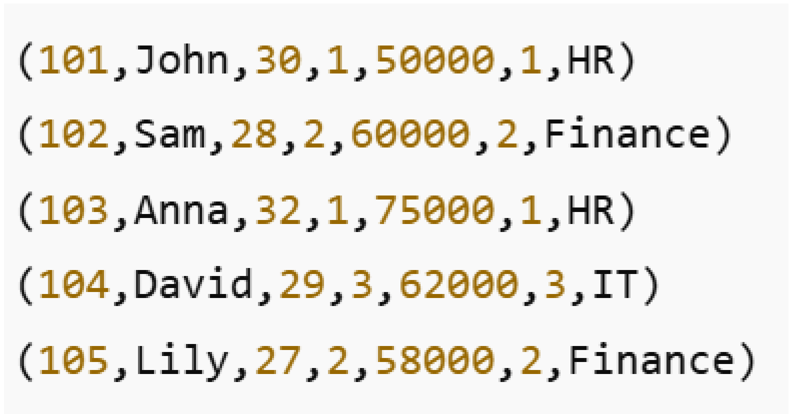
```
103,Anna,75000
104,David,62000
102,Sam,60000
101,John,50000
```

Grouped Employees by Departments



```
(1,{(101,John,30,1,50000),(103,Anna,32,1,75000)})
(2,{(102,Sam,28,2,60000),(105,Lily,27,2,58000)})
(3,{(104,David,29,3,62000)})
```

Joined Employees with Departments



```
(101,John,30,1,50000,1,HR)
(102,Sam,28,2,60000,2,Finance)
(103,Anna,32,1,75000,1,HR)
(104,David,29,3,62000,3,IT)
(105,Lily,27,2,58000,2,Finance)
```

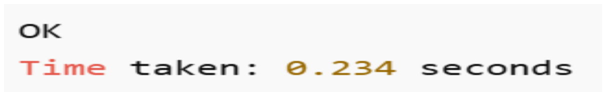
Program 7:

AIM: Use HIVE to create, alter, and drop databases, tables, views, functions and indexes

Create a Database:

CREATE DATABASE employee_db;

Output:

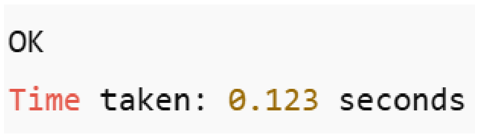


```
OK
Time taken: 0.234 seconds
```

Use the database:

USE employee_db;

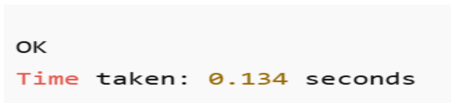
Output:



```
OK
Time taken: 0.123 seconds
```

Alter a Database:

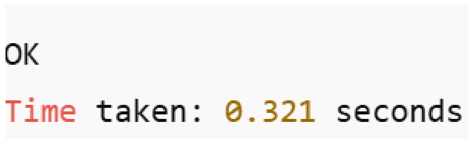
ALTER DATABASE employee_db SET DBPROPERTIES ('Owner'='Admin');



```
OK
Time taken: 0.134 seconds
```

Drop a Data Base

DROP DATABASE employee_db CASCADE;



```
OK
Time taken: 0.321 seconds
```

Create a Table

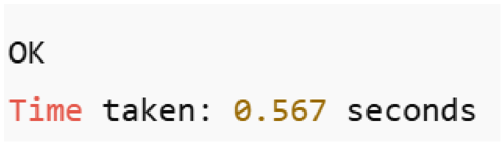
```
CREATE TABLE employees (
    emp_id INT,
    name STRING,
    age INT,
    dept_id INT,
    salary FLOAT
)
```

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

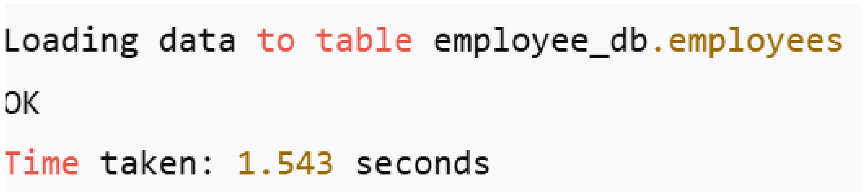
Output:



```
OK
Time taken: 0.567 seconds
```

LOAD DATA INPATH '/user/cloudera/employees.txt' INTO TABLE employees;

Output:



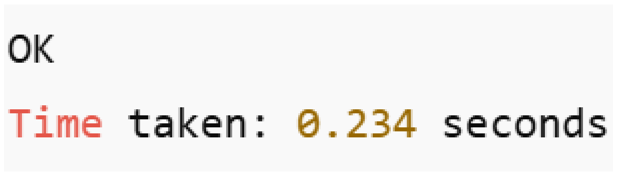
```
Loading data to table employee_db.employees
OK
Time taken: 1.543 seconds
```

Alter a Table

Add a New Column

ALTER TABLE employees ADD COLUMNS (email STRING);

Output:



```
OK
Time taken: 0.234 seconds
```

Rename:

ALTER TABLE employees RENAME TO employees_new;

Drop Table:

DROP TABLE employees_new;

Create a View

CREATE VIEW high_salary_employees AS

SELECT emp_id, name, salary

FROM employees

WHERE salary > 50000;

Alter a View:

ALTER VIEW high_salary_employees AS

SELECT emp_id, name, age, salary

FROM employees

WHERE salary > 60000;

Drop View:

DROP VIEW high_salary_employees;

Create a Function

Add a JAR file containing a Java-based UDF:

ADD JAR /user/cloudera/custom_udf.jar;

CREATE FUNCTION to_upper AS 'com.example.hiveudf.ToUpperUDF';

```
Added /user/cloudera/custom_udf.jar to class path
OK
Time taken: 0.568 seconds
```

Use the Function

SELECT to_upper(name) FROM employees;

Output:

```
JOHN
SAM
ANNA
DAVID
LILY
OK
Time taken: 0.345 seconds
```

Drop Function

DROP FUNCTION to_upper;

Create an Index

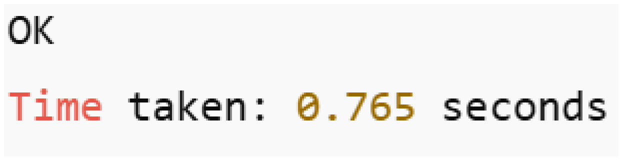
CREATE INDEX emp_dept_idx

ON TABLE employees (dept_id)

AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'

WITH DEFERRED REBUILD;

Output:



```
OK
Time taken: 0.765 seconds
```

Rebuild the Index:

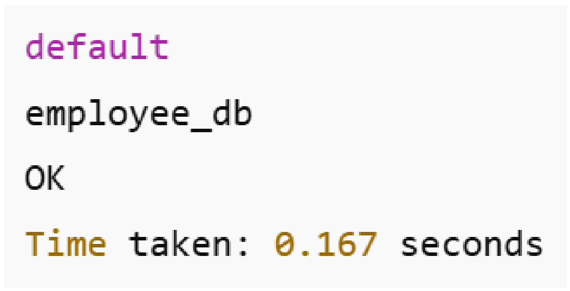
ALTER INDEX emp_dept_idx ON employees REBUILD;

Drop Index

DROP INDEX emp_dept_idx ON employees;

Check all tables in the current database:

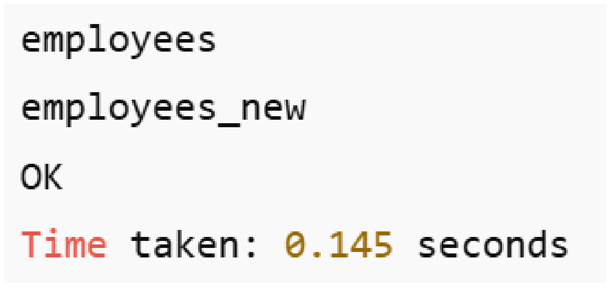
SHOW TABLES;



```
default
employee_db
OK
Time taken: 0.167 seconds
```

Check all tables in the current database:

SHOW TABLES;



```
employees
employees_new
OK
Time taken: 0.145 seconds
```

DESCRIBE employees;

```
emp_id      int
name        string
age         int
dept_id     int
salary      float
email       string
OK
Time taken: 0.234 seconds
```

Display the table data

SELECT * FROM employees LIMIT 5;

```
101  John  30  1  50000.0  NULL
102  Sam   28  2  60000.0  NULL
103  Anna  32  1  75000.0  NULL
104  David 29  3  62000.0  NULL
105  Lily  27  2  58000.0  NULL
OK
Time taken: 0.459 seconds
```

Program 8:

AIM: Implement word count program in Hadoop and spark

Source Code;

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;
import java.util.StringTokenizer;

public class WordCount {

    public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException
{
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }
    public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Compile and Package the Java Code

```
javac -classpath `hadoop classpath` -d . WordCount.java
jar cf wc.jar WordCount*.class
```

Run the Hadoop Word Count Job

Upload the input file to HDFS:

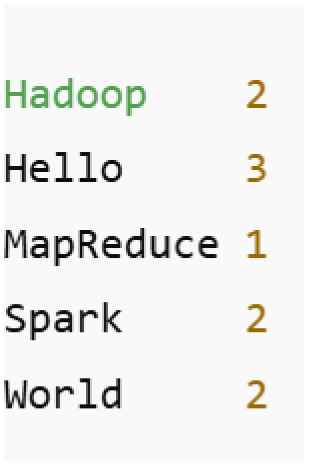
```
hdfs dfs -mkdir -p /user/cloudera/input
hdfs dfs -put sample.txt /user/cloudera/input/
```

Run the Map Reduce job

```
hadoop jar wc.jar WordCount /user/cloudera/input /user/cloudera/output
```

View the Output:

```
hdfs dfs -cat /user/cloudera/output/part-r-00000
```



```
Hadoop    2
Hello     3
MapReduce 1
Spark     2
World     2
```

Word Count in Apache Spark (PySpark)

1.Prepare the Input File

Upload the Sample.txt

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

hdfs dfs -put sample.txt /user/cloudera/input/

sample input File:

```
Hello Hadoop
Hello Spark
Hello World
Spark is fast
Hadoop is slow
```

2. Create a new script file using nano or vi:

```
nano wordcount.py
```

3.code

```
from pyspark.sql import SparkSession

# Initialize Spark Session
spark = SparkSession.builder.appName("WordCount").getOrCreate()
```

```
# Read text file from HDFS
text_file = spark.sparkContext.textFile("hdfs://localhost:9000/user/cloudera/input/sample.txt")

# Process data
word_counts = (text_file
              .flatMap(lambda line: line.split(" "))   # Split lines into words
              .map(lambda word: (word, 1))             # Map each word to (word, 1)
              .reduceByKey(lambda a, b: a + b))        # Reduce by key (word) and sum counts

# Save the output to HDFS
word_counts.saveAsTextFile("hdfs://localhost:9000/user/cloudera/output_spark")

# Print results
for word, count in word_counts.collect():
    print(f"{word}: {count}")

# Stop Spark Session
spark.stop()
```

3.Execute the Script

```
spark-submit wordcount.py
```

4.View the output:

```
hdfs dfs -ls /user/cloudera/output_spark
hdfs dfs -cat /user/cloudera/output_spark/part-00000
```

```
Hello 3
Hadoop 2
Spark 2
World 1
is 2
fast 1
slow 1
```

Program 9:

AIM: Use CDH (Cloudera Distribution for Hadoop) and HUE (Hadoop User Interface) to analyze the data and generate reports for sample data sets

Steps and Source Code:

1.Start Cloudera services:

```
sudo service cloudera-scm-server start
sudo service cloudera-scm-agent star
```

Check the status:

```
sudo service --status-all | grep cloudera
```

2. Access HUE Web Interface

Open your browser and navigate to:

http://localhost:8888

Login using HUE credentials:

Username: cloudera

Password: cloudera

3. Upload Sample Dataset to HDFS

Example Dataset: Employee Data (employees.csv)

```
id,name,department,salary
1,John,IT,70000
2,Alice,HR,60000
3,Bob,IT,75000
4,Charlie,Finance,80000
5,David,HR,62000
6,Eva,IT,72000
7,Frank,Finance,81000
8,Grace,HR,65000
```

Upload employees.csv to HDFS using HUE

1. **Navigate to HUE → File Browser → HDFS**
2. **Create a new directory** /user/cloudera/data
3. Click **Upload** → Select employees.csv → Upload it

Or through terminal

```
hdfs dfs -mkdir -p /user/cloudera/data
 hdfs dfs -put employees.csv /user/cloudera/data/
```

Create a Hive Table in HUE

Open HUE → Click Query Editors → Select Hive

4. Create a Hive table for the dataset:

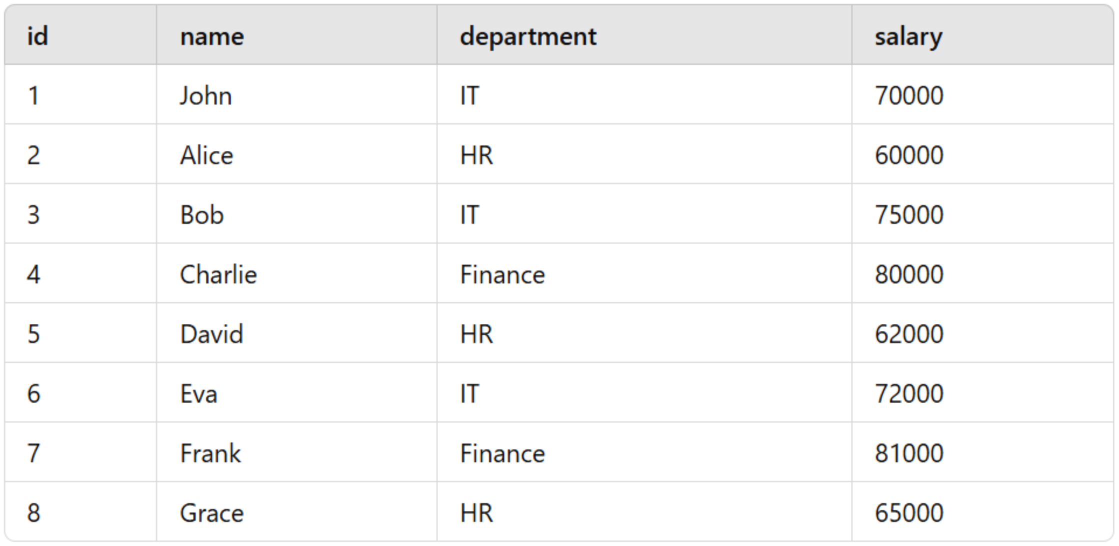
```
CREATE DATABASE IF NOT EXISTS company;
USE company;

CREATE TABLE employees (
    id INT,
    name STRING,
    department STRING,
    salary INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Load data into the Hive table:

```
LOAD DATA INPATH '/user/cloudera/data/employees.csv' INTO TABLE employees;
```

Verify the data:

```
SELECT * FROM employees;
```

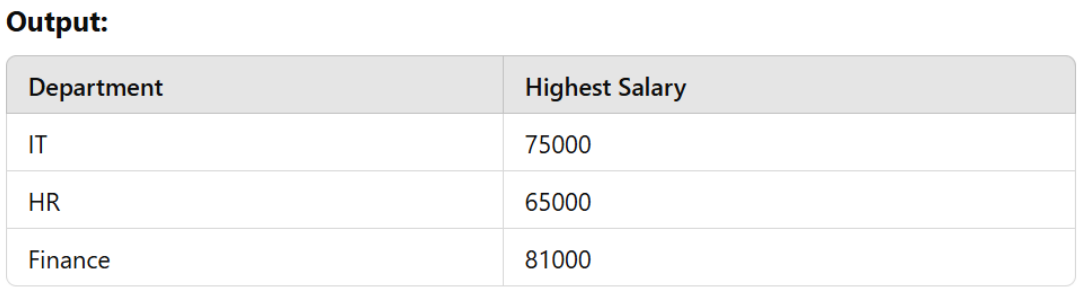


id	name	department	salary
1	John	IT	70000
2	Alice	HR	60000
3	Bob	IT	75000
4	Charlie	Finance	80000
5	David	HR	62000
6	Eva	IT	72000
7	Frank	Finance	81000
8	Grace	HR	65000

Data Analysis Using Hive Queries in HUE

Find the Highest Salary in Each Department

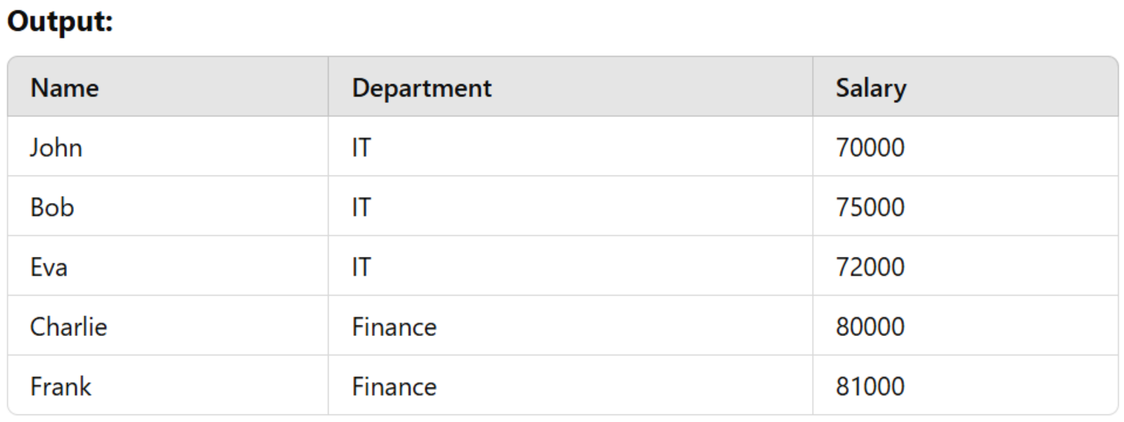
```
SELECT department, MAX(salary) AS highest_salary
FROM employees
GROUP BY department;
```



Output:

Department	Highest Salary
IT	75000
HR	65000
Finance	81000

Get Employees with Salary Greater Than 65000;



Output:

Name	Department	Salary
John	IT	70000
Bob	IT	75000
Eva	IT	72000
Charlie	Finance	80000
Frank	Finance	81000

Generate Reports in HUE

Step 1: Export Query Results

1. Run any of the above SQL queries in HUE Query Editor
2. Click Export → Choose format (CSV, Excel, JSON)
3. Download the report

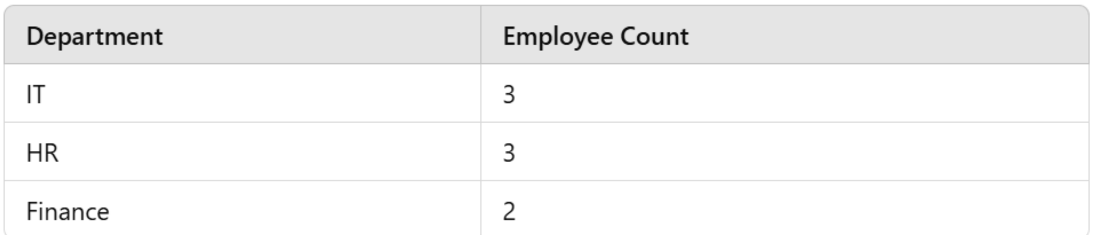
Step 2: Create HUE Dashboard for Visualization

1. Open HUE → Click Dashboard
2. Click Create New Dashboard
3. Click Add Widget → Select Chart Type (Bar Chart, Pie Chart, etc.)

Enter Query → Example for Employee Count:

SELECT department, COUNT(*) AS employee_count FROM employees GROUP BY department;

Click "Run Query" → The visualization will be generated



Department	Employee Count
IT	3
HR	3
Finance	2

Viva Questions:

1. What is Hadoop and why is it used?
2. What are the main components of Hadoop?
3. What is HDFS? Explain its architecture.
4. What is the function of the NameNode in HDFS?
5. What is the function of the DataNode in HDFS?
6. What is the difference between NameNode and Secondary NameNode?
7. What is a block in HDFS, and how large is the default block size?
8. What is the function of YARN in Hadoop?
9. Explain the architecture of YARN.
10. What is MapReduce and how does it work?
11. What is a JobTracker in Hadoop 1.x?
12. What is a TaskTracker in Hadoop 1.x?
13. What is the difference between Hadoop 1.x and Hadoop 2.x?
14. What are the advantages of Hadoop over traditional RDBMS?
15. What is the role of ResourceManager and NodeManager in YARN?
16. What is data replication in HDFS? How is data redundancy handled?
17. What is the function of the JobHistory Server in Hadoop?
18. What are the main differences between HDFS and traditional file systems?
19. What is a combiner in MapReduce?
20. What is the difference between a combiner and a reducerWhat is MongoDB?
21. What are the advantages of using MongoDB over relational databases?
22. What are the data types supported by MongoDB?
23. What is a collection in MongoDB?
24. What is a document in MongoDB?
25. How does MongoDB handle data replication?
26. What is Sharding in MongoDB?
27. What is the use of an index in MongoDB?
28. What is the Aggregation framework in MongoDB?
29. How does MongoDB ensure fault tolerance?
30. What is the difference between MongoDB and RDBMS?
31. What are the different types of indexes in MongoDB?
32. What is the purpose of the find() method in MongoDB?

33. What is Hive and how is it different from traditional RDBMS?

34. What is the architecture of Hive?

35. What is HiveQL?

36. What is the function of the Hive Metastore?

37. What are the different types of tables in Hive?

38.What are the advantages of using Hive over SQL databases?

39. What are partitions in Hive?

40. What are buckets in Hive?

41. What is the difference between internal and external tables in Hive?

42. What file formats does Hive support?

43. What is Apache Spark?

44. What are the core components of Spark?

45. What is an RDD (Resilient Distributed Dataset) in Spark?

46. What are the advantages of using Spark over Hadoop MapReduce?

47. What is the role of Driver and Executor in Spark?

48. What are the types of transformations in Spark?

49. What are the types of actions in Spark?

50 .What is Spark Streaming?

51. What is the purpose of Spark SQL?

52. What is the difference between map() and flatMap() in Spark?

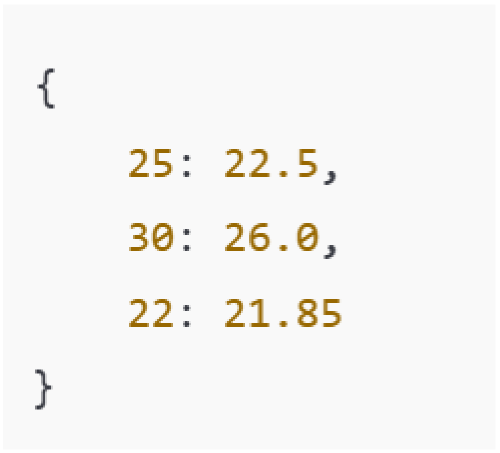
Additional Program

1. **Develop a Map Reduce program for sample health data monitoring data set**

```
# Simulated health data as a list of tuples (PatientID, Age, Gender, BMI)
health_data = [
    (101, 25, 'M', 22.1),
    (102, 30, 'F', 24.5),
    (103, 22, 'M', 21.8),
    (104, 25, 'F', 23.0),
    (105, 30, 'M', 27.5),
    (106, 22, 'F', 21.9),
    (107, 30, 'M', 28.0),
    (108, 25, 'F', 22.8)
]

# Step 1: Mapper function to emit key-value pairs (age, bmi)
def mapper(data):
    output = []
    for entry in data:
        age = entry[1]
        bmi = entry[3]
        output.append((age, bmi))  # Emit age as key, and BMI as value
    return output

# Step 2: Shuffle and Sort phase (simulated by grouping by age)
def shuffle_and_sort(mapped_data):
    from collections import defaultdict
    grouped_data = defaultdict(list)

    for age, bmi in mapped_data:
        grouped_data[age].append(bmi)

    return grouped_data
```

```
# Step 3: Reducer function to calculate the average BMI for each age group
def reducer(grouped_data):
    averages = {}
    for age, bmi_list in grouped_data.items():
        avg_bmi = sum(bmi_list) / len(bmi_list)
        averages[age] = avg_bmi
    return averages

# Running the MapReduce-like steps in the notebook
mapped_data = mapper(health_data)  # Step 1: Map
grouped_data = shuffle_and_sort(mapped_data)  # Step 2: Shuffle and Sort
result = reducer(grouped_data)  # Step 3: Reduce

# Displaying the result
result
```

Output:



```
{
    25: 22.5,
    30: 26.0,
    22: 21.85
}
```

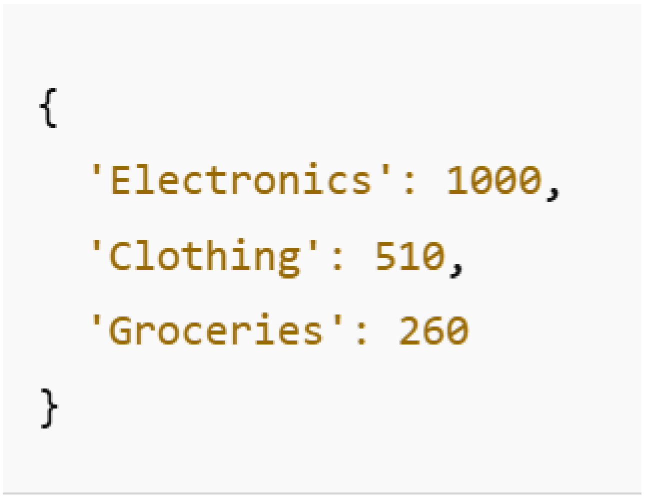
Program 2:

AIM: Develop a Map Reduce program for sample retail data analysis

Source Code:

```
# Sample retail data (ProductID, ProductCategory, SalesAmount, QuantitySold)
retail_data = [
    (101, 'Electronics', 200, 2),
    (102, 'Clothing', 50, 3),
    (103, 'Electronics', 150, 4),
    (104, 'Clothing', 30, 5),
    (105, 'Groceries', 20, 10),
    (106, 'Groceries', 15, 7),
    (107, 'Clothing', 60, 2)
]

# Step 1: Mapper function to calculate total sales for each transaction
def mapper(data):
    output = []
    for entry in data:
        product_category = entry[1]
        sales_amount = entry[2]
        quantity_sold = entry[3]
        total_sales = sales_amount * quantity_sold  # Calculate total sales
        output.append((product_category, total_sales))  # Emit product category as key, and total
sales as value
    return output

# Step 2: Shuffle and Sort phase (simulated by grouping by product category)
def shuffle_and_sort(mapped_data):
    from collections import defaultdict
    grouped_data = defaultdict(list)

    for product_category, total_sales in mapped_data:
        grouped_data[product_category].append(total_sales)
        return grouped_data
# Step 3: Reducer function to calculate total sales per product category
def reducer(grouped_data):
    total_sales_per_category = {}
    for category, sales_list in grouped_data.items():
        total_sales_per_category[category] = sum(sales_list)  # Sum up all sales for the category
    return total_sales_per_category

# Running the MapReduce-like steps in the notebook
mapped_data = mapper(retail_data)  # Step 1: Map
grouped_data = shuffle_and_sort(mapped_data)  # Step 2: Shuffle and Sort
result = reducer(grouped_data)  # Step 3: Reduce

# Displaying the result
result
```

Output:



```
{
  'Electronics': 1000,
  'Clothing': 510,
  'Groceries': 260
}
```