**KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY**

**Narayanaguda, Hyderabad – 500029**

# (Autonomous)

Logo, company name

Description automatically generated

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING(AI&ML)**

**LAB MANUAL-KR20**

**BIGDATA WITH HADOOP LAB**

(2022-2023)

PREPARED BY

## 

Keshav Memorial Institute of Technology

(Approved by AICTE, Affiliated to JNTUH & Accredited by NBA)

Narayanguda, Hyderabad, Telangana 500029

KR20

(Autonomous)

# B.Tech. II Year SEM II Syllabus (KR20 Regulations)

**COMPUTER SCIENCE AND ENGINEERING (AI & ML)**

# CM603PC BIG DATA WITH HADOOP AND SPARK LAB

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

**III YEAR B.TECH CML II-SEM**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TOPIC** | **PAGE NO** |
| I | List of Experiments | 3 |
| II | V/M /POs/PSOs/PEOs | 4 |
| III | Course Objectives & Course Outcomes | 9 |

**LIST OF EXPERIMENTS**

1. Execute file system commands in HDFS shell and the files and folders verify in Web UI.
2. Write a program to read, write operations from local file system to HDFS using File System API.
3. Write a program to create directories and copy from one directory to another directory using File System API
4. Implement Map-Reduce application to find sum of salaries of employees for each department
5. Write a pig script for log file analysis.
6. Implement a pig script to analyze the data using UDFs.
7. Create Managed table and load a csv file from local storage. Verify the location of table data in HDFS and perform query operations. Delete the table and verify the data in HDFS.
8. Create External table and load a csv file from local storage. The data should be stored in ORC format in specified folder in HDFS. Perform query operations and verify the data in HDFS. Delete the table and verify the data in HDFS.
9. Create partitioned table using Static partitioning technique load the data into corresponding partitions.
10. Create and manage a table with bucketing concept and execute queries with UDFs.
11. Implement flume script for spooling a directory from local file system to HDFS.
12. Import selective data from database into a specific directory in HDFS and analyze it. Export the summarized data into another table of database.
13. Load data into RDD and perform Transformations and Actions in Spark.
14. Implement spark script to demonstrate shared variables and broadcast variables.
15. Load and analyze data using spark sql using sqlcontext and hive context.

**KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY**

(Autonomous)

(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)

3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

**Department of Computer Science and Engineering(AI&ML)**

**Vision & Mission of the Department**

Vision of the Department

To be among the region's premier teaching and research Computer Science and Engineering departments producing globally competent and socially responsible graduates in the most conducive academic environment.

Mission of the Department

* To provide faculty with state of the art facilities for continuous professional development and research, both in foundational aspects and of relevance to emerging computing trends.
* To impart skills that transform students to develop technical solutions for societal needs and inculcate entrepreneurial talents.
* To inculcate an ability in students to pursue the advancement of knowledge in various specializations of Computer Science and Engineering and make them industry-ready.
* To engage in collaborative research with academia and industry and generate adequate resources for research activities for seamless transfer of knowledge resulting in sponsored projects and consultancy.
* To cultivate responsibility through sharing of knowledge and innovative computing solutions that benefit the society-at-large.
* To collaborate with academia, industry and community to set high standards in academic excellence and in fulfilling societal responsibilities.

**KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY**

(Autonomous)

(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)

3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

**Department of Computer Science and Engineering(AI&ML)**

#### 

# PROGRAM OUTCOMES (POs)

1. **Engineering Knowledge:** Apply knowledge of mathematics and science, with fundamentals of Computer Science & Engineering to be able to solve complex engineering problems related to CSE.
2. **Problem Analysis:** Identify, Formulate, review research literature and analyze complex engineering problems related to CSE and reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences
3. **Design/Development of solutions:** Design solutions for complex engineering problems related to CSE and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety and the cultural societal and environmental considerations
4. **Conduct Investigations 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.
5. **Modern Tool Usage:** Create, Select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to computer science related complex engineering activities with an understanding of the limitations
6. **The Engineer and Society:** Apply Reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the CSE professional engineering practice
7. **Environment and Sustainability:** Understand the impact of the CSE professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of, and need for sustainable development
8. **Ethics:** Apply Ethical Principles and commit to professional ethics and responsibilities and norms of the engineering practice
9. **Individual and Team Work:** Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary Settings
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large such as able to comprehend and with write effective reports and design documentation, make effective presentations and give and receive clear instructions.
11. **Project Management and Finance:** Demonstrate knowledge and understanding of the engineering management principles and apply these to one's own work, as a member and

leader in a team, to manage projects and in multi-disciplinary environments

1. **Life-Long Learning:** Recognize the need for and have the preparation and ability to engage in independent and life-long learning the broadest context of technological change.

**KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY**

(Autonomous)

(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)

3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

**Department of Computer Science and Engineering(AI&ML)**

**PROGRAM SPECIFIC OUTCOMES (PSOs)**

**PSO1:** An ability to analyze the common business functions to design and develop appropriate

Computer Science solutions for social upliftment.

**PSO2:** Shall have expertise on the evolving technologies like Python, Machine Learning, Deep Learning, Internet of Things (IOT), Data Science, Full stack development, Social

Networks, Cyber Security, Big Data, Mobile Apps, CRM, ERP etc.

**KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY**

(Autonomous)

(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)

3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

**Department of Computer Science and Engineering(AI&ML)**

**PROGRAM EDUCATIONAL OBJECTIVES (PEOs)**

**PEO 1:** Graduates will endeavor to excel in their chosen careers as professionals, researchers

and entrepreneurs on a global platform.

**PEO 2:** Graduates will demonstrate the ability to solve challenges in the fields of Engineering

and Technology simultaneously catering to societal needs.

**PEO 3**: Graduates will strive to improve their learning curve by practising Continuing

Professional Development (CPD)

**PEO 4:** Graduates will, at all times, adopt a professional demeanor by communicating

effectively, working collaboratively, and maintaining the ethics & core values as befitting their education in interdisciplinary and emerging fields.

**KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY**

(Autonomous)

(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)

3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

**Department of Computer Science and Engineering(AI&ML)**

**COURSE OBJECTIVES**:

* Hands on practice on Bigdata handling using Hadoop and its eco systems.
* Store and Analyze data with Hadoop eco systems
* Hands on Advanced parallel computing concept using Spark
* Hands on sqoop knowledge.
* Working with Hive to analyze large datasets.

### **COURSE OUTCOMES (COs)**

### 

**CO 1**: Store and manage data in HDFS

**CO 2**: Implement basic applications in map-reduce.

**CO 3:** Store and analyze data using PIG scripts

**CO 4:** Able to handle partitioned and bucked tables in Hive.

**CO 5:** Import and export data from databases like mysql or oracle.

**CO 6:** Work with various file formats in hadoop echo systems.

**CO 7:** Implement spark scripts using RDDs

**KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY**

(Autonomous)

(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)

3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

**Department of Computer Science and Engineering(AI&ML)**

### **CO-PO-PSO MAPPING**

##### **EXPERIMENT 1**

1. **Execute file system commands in HDFS shell and the files and folders verify in Web UI.**
2. ***mkdir***

***syntax:*** $ hadoop fs -mkdir  [-p]

***Example:*** $ hadoop fs –mkdir mydir1

1. ***ls***

***syntax:*** $ hadoop fs -ls [-d] [-h] [-R]

***Example:*** $ hadoop fs -ls /user

1. ***put***

***syntax:*** $ hadoop fs -put [-f] [-p] ...

***Example:*** $ hadoop fs -put Desktop/sample.txt /user/data/sampletext.txt

1. ***get***

***syntax:*** $ hadoop fs -get [-f] [-p]

***Example:*** $ hadoop fs -get /user/data/sampletext.txt Desktop/sample1.txt

1. ***cat***

***Example:*** $ hadoop fs -cat /user/data/sampletext.txt

1. ***cp***

***Example:*** $ hadoop fs -cp /user/data/sample1.txt /user/hadoop1

$ hadoop fs -cp /user/data/sample2.txt /user/test/in1

1. ***mv***

***Example:*** $ hadoop fs -mv /user/hadoop/sample1.txt /user/text/

1. ***rm***

***Syntax:*** $ hadoop fs -rm [-f] [-r|-R] [-skipTrash]

***Example:*** $ hadoop fs –rm sample1.txt

1. ***getmerge***

***Example:*** $ hadoop fs -getmerge /user/data

1. ***setrep***

***Syntax***: $ hadoop fs -setrep [-R] [-w]

*Example*: $ hadoop fs –setrep –R 5 /user/mydir1

1. ***appendToFile***

$ hadoop fs -appendToFile sample2.txt sample1.txt

1. ***tail***

$ hadoop fs –tail sample1.txt

1. ***chgrp***

***Syntax:*** $ hadoop fs -chgrp [-R] groupname

***Example:*** $ hadoop fs –chgrp –r hadopgrp mydir1

1. ***chmod***

***Syntax:*** $ hadoop fs -chmod [-R] PATH

***Example:*** $ hadoop fs –chmod –R 755 mydir1

1. ***chown***

***Syntax:*** $ hadoop fs -chown [-R] [OWNER][:[GROUP]] PATH

***Example:*** $ hadoop fs –chown –R hduser /user/mydir1

##### **EXPERIMENT 2**

1. **Write a program to read, write operations from local file system to HDFS using File System API.**

**Program for get command:**

import org.apache.hadoop.conf.\*;

import org.apache.hadoop.fs.\*;

public class Download

{

public static void main(String s[])

{

Configuration conf=new Configuration();

FileSystem fs;

try

{

fs=FileSystem.get(conf);

Path src=new Path(s[0]);

Path dst=new Path(s[1]);

fs.copyToLocalFile(src, dst);

}

catch(Exception e)

{

System.out.println(e);

}

}

}

**Program for put command:**

import org.apache.hadoop.conf.\*;

import org.apache.hadoop.fs.\*;

public class Upload

{

public static void main(String s[])

{

Configuration conf=new Configuration();

FileSystem fs;

try

{

fs=FileSystem.get(conf);

Path src=new Path(s[0]);

Path dst=new Path(s[1]);

fs.copyFromLocalFile(src, dst);

fs.close();

}

catch(Exception e)

{

System.out.println(e);

}

}

}

**Program for read operation:**

import java.io.DataInputStream;

import org.apache.hadoop.conf.\*;

import org.apache.hadoop.fs.\*;

public class ReadFile

{

public static void main(String s[])

{

String filename;

String data = null;

DataInputStream ds=new DataInputStream(System.in);

Configuration conf=new Configuration();

FSDataInputStream fsi;

FileSystem fs;

try

{

System.out.println("Enter the file to be displayed");

filename=ds.readLine();

fs=FileSystem.get(conf);

fsi=fs.open(new Path(filename));

System.out.println("Reading....");

while((data=fsi.readLine())!= null)

{

System.out.println("in progress....");

System.out.println(data);

}

}

catch(Exception e)

{

System.out.println(e);

}

}

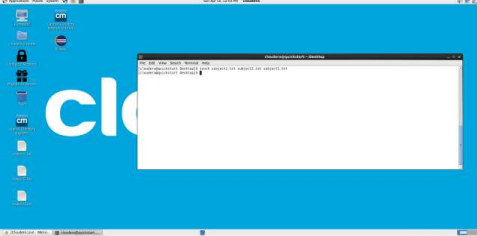
}

##### **EXPERIMENT 3**

1. **Write a program to create directories and copy from one directory to another directory using File System API.**

**Linux Command to create files:**

**[cloudera@quickstart Desktop]$ touch subject1.txt subject2.txt su bject3.txt**

****

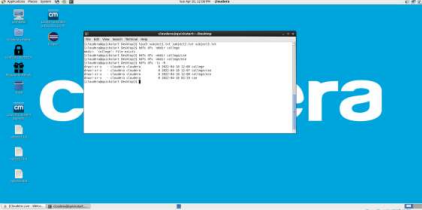
**Create the directory structure in HDFS.**

**HDFS Command for Create Directory Structure:**

**[cloudera@quickstart Desktop]$ hdfs dfs –mkdir college**

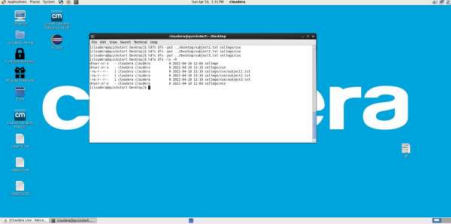
**[cloudera@quickstart Desktop]$ hdfs dfs –mkdir college/cse**

**[cloudera@quickstart Desktop]$ hdfs dfs –mkdir college/ece**

****

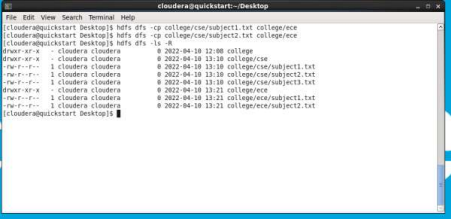
**2. Copy subject1,subject2 and subject3 files onto college/cse directory HDFS Command:**

**[cloudera@quickstart Desktop]$ hdfs dfs –put ../Desktop/subject1.txt college/cse [cloudera@quickstart Desktop]$ hdfs dfs –put ../Desktop/subject2.txt college/cse [cloudera@quickstart Desktop]$ hdfs dfs –put ../Desktop/subject3.txt college/cse**

****

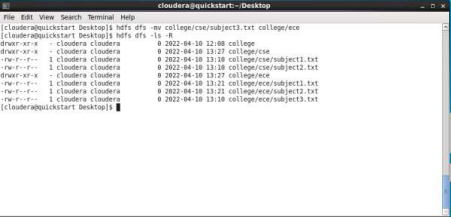
**3. Copy subject1, subject2 files from college/cse to ece directory**

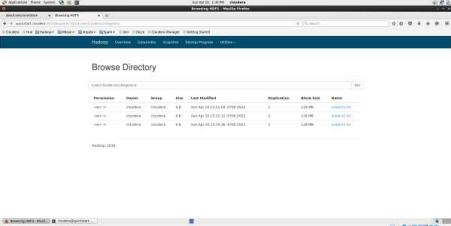
**HDFS Command:**

**[cloudera@quickstart Desktop]$ hdfs dfs –cp college/cse/subject1.txt college/ece [cloudera@quickstart Desktop]$ hdfs dfs –cp college/cse/subject2.txt college/ece**

**4. Move the file subject3 from college/cse to ece directory and  Verify the created structure on web interface**

**HDFS Command:**

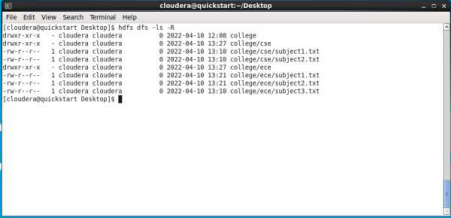
**[cloudera@quickstart Desktop]$ Hdfs dfs –mv college/cse/subject3.txt college/ece Web interface:**

****

**5. Verify the structure on command prompt.**

**HDFS Command:**

**[cloudera@quickstart Desktop]$ Hdfs dfs –ls -R**

****

##### **EXPERIMENT 4**

1. **Implement Map-Reduce application to find sum of salaries of employees for each department.**

Data set: Empsalary.txt

empno, name, salary, agegroup, dept

1001,RAMESH,9000,A,CSE

1002,SOMESH,8000,A,ECE

1013,VANITA,45000,B,CSE

1014,KIRAN,50000,B,ECE

1023,SHRAVAN,70000,C,EEE

1024,AKASH,75000,C,IT

1029,ANANTH,150000,D,IT

1030,BHASKAR,125000,D,EEE

...

Develop a map reduce application to find the maximum salary each age group.

Output:

A xxxx

B yyyy

C zzzz

d mmmm

**EmpWritable.java**

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.Writable;

public class EmpWritable implements Writable

{

private LongWritable empID = new LongWritable();

private Text empName = new Text();

private LongWritable salary = new LongWritable();

private Text ageGroup = new Text();

private Text empDepartment = new Text();

public EmpWritable()

{

this.empID = new LongWritable();

this.empName =  new Text();

this.salary = new LongWritable();

this.ageGroup = new Text();

this.empDepartment = new Text();

}

public void set (long empID, String empName, long salary,String ageGroup,String empDepartment)

{

this.empID.set(empID);

this.empName.set(empName);

this.salary.set(salary);

this.ageGroup.set(ageGroup);

this.empDepartment.set(empDepartment);

}

public void readFields(DataInput in) throws IOException {

empID.readFields(in);

empName.readFields(in);

salary.readFields(in);

ageGroup.readFields(in);

empDepartment.readFields(in);

}

public void write(DataOutput out) throws IOException {

empID.write(out);

empName.write(out);

salary.write(out);

ageGroup.write(out);

empDepartment.write(out);

}

public LongWritable getEmpID()

{

return empID;

}

public Text getEmpName()

{

return empName;

}

public LongWritable getSalary() {

return salary;

}

public Text getAgeGroup() {

return ageGroup;

}

public Text getEmpDepartment() {

return empDepartment;

}

}

**EmpProcessor.java**

import java.net.URI;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.jobcontrol.ControlledJob;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class EmpProcessor extends Configured implements Tool {

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

int res = ToolRunner.run(new Configuration(), new EmpProcessor(),

args);

System.exit(res);

}

@Override

public int run(String[] args) throws Exception {

if (args.length < 3) {

System.err.println("Usage:  <input\_path> <output\_path> <num\_reduce\_tasks>");

System.exit(-1);

}

/\* input parameters \*/

String inputPath = args[0];

String outputPath = args[1];

int numReduce = Integer.parseInt(args[2]);

@SuppressWarnings("deprecation")

Job job = new Job(getConf(), "employee-data-analysis");

job.setJarByClass(EmpProcessor.class);

job.setMapperClass(EmpProcessorMap.class);

job.setReducerClass(EmpProcessorReducer.class);

    job.setOutputKeyClass(Text.class);

    job.setOutputValueClass(LongWritable.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(EmpWritable.class);//specifying EmpWritable as a (Map output) value type

FileInputFormat.setInputPaths(job, new Path(inputPath));

FileOutputFormat.setOutputPath(job, new Path(outputPath));

job.setNumReduceTasks(numReduce);

int exitStatus = job.waitForCompletion(true) ? 0 : 1;

return exitStatus;

}

}

**EmpProcessorMap.java**

import java.io.File;

import java.io.IOException;

import org.apache.hadoop.io.\*;

import java.net.URL;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

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.Mapper;

public class EmpProcessorMap extends Mapper<Object, Text, Text, EmpWritable > {

EmpWritable empValue;

public void map(Object key, Text value, Context context)

throws IOException, InterruptedException {

String logEntryPattern = "^(\\S+),(\\S+),(\\S+),(\\S+),(\\S+)";

Pattern p = Pattern.compile(logEntryPattern);

Matcher matcher = p.matcher(value.toString());

if (!matcher.matches()) {

return;

}

String name = matcher.group(2);

int eid = Integer.parseInt(matcher.group(1));

String dept = matcher.group(5);

String age = matcher.group(4);

int sal = Integer.parseInt(matcher.group(3));

empValue = new EmpWritable();

empValue.set(eid, name, sal, age,dept );

context.write(empValue. getEmpDepartment(),empValue);

// Write your code here

}

}

**EmpProcessorReduce.java**

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.io.\*;

public class EmpProcessorReducer extends

Reducer<Text,EmpWritable,Text,LongWritable> {

   private LongWritable result = new LongWritable();

   public void reduce(Text key, Iterable<EmpWritable> values,

                      Context context) throws IOException, InterruptedException {

     // Write your code here

     //maximum salary of each ageGroup

     long max = 0;long number=0;

     for (EmpWritable val : values) {

number =val.getSalary().get();

        if(number>max)

{

max=number;//update max variable if you found input value is greater than max

}

     }

     result.set(max);

     context.write(key, result);

   }

}

##### **EXPERIMENT 5**

**5.** **Write a pig script for log file analysis.**

CompanyData.log is available in HDFS at /user/student

1.1. Create a managed table company in hive for the data available in CompanyData.log.

Fields

companyId

companyName

companyAddress

PreviousYrs\_Revenues

growth\_Preferences

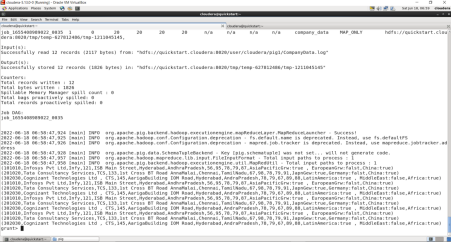
1.2. Store the hive table data in /user/student/your\_directory/warehouse/company/  1.3. Use appropriate Collection data types.

1.4. Load the data from CompanyData.log into company table in pig

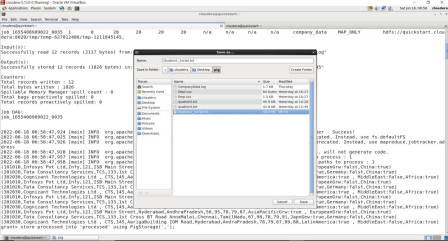
grunt> company\_data = load 'pig1/CompanyData.log' using PigStorage('|') as (companyId:int, companyName:chararray, companyAddress:chararray, PYR:chararray, GP:chararray);

1.5. Display all the records from table

grunt>dump company\_data;



1.6. Store your commands/script in /user/student/your\_directory/company1.txt



After save import to HDFS

[cloudera@quickstart ~]$ hdfs dfs -put '/home/cloudera/Desktop/pig/Quation1\_Script.txt'  pig\_scripts

[cloudera@quickstart ~]$ hdfs dfs -ls pig\_scripts

In this log file are two different separates ‘|’ and ‘,’

First load file using ‘|’ and store using pigStorage(‘,’)

store company\_data into 'cd' using PigStorage(',');



After that load that stored file using PigStorage(‘,’)

grunt>cmprv = load 'cd/part-m-00000' using PigStorage(',') as

(CompanyId:int,Company\_Actual\_name:chararray,Company\_Alias\_name:chararray,Company\_roa d\_num:int,Company\_street:chararray,Company\_city:chararray,Company\_state:chararray,PYR:cha rarray,GP:chararray);



##### **EXPERIMENT 6**

**Implement a pig script to analyze the data using UDFs.**

Create employee and dept tables for the available files emp1.csv and dept.csv.

Colnames

Emp: Empno, name, job, mgr, hdate, sal, comm.,deptno

deptno Dept: dno, name, loc

Write the queries for the following

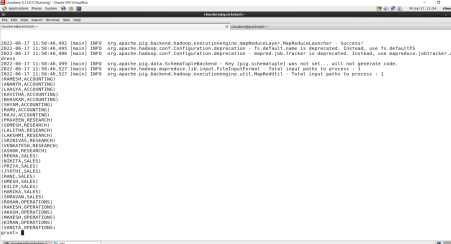
Retrieve employee name and department name

grunt> dept\_emp = join emp by deptno,dept by deptno;

grunt> e\_d = foreach dept\_emp generate emp::name,dept::dept\_name;

grunt> dump e\_d;





Retrieve the employe names who are working in ‘dallas’

grunt>emp\_dallas = filter dept\_emp by (dept::location =='CHICAGO') ;

Retrieve total salaries to be paid for the employees working in ‘chicago’ grunt>emp\_sal = group emp\_dallas by $10;

grunt>Result = FOREACH emp\_sal generate group,SUM(emp\_dallas.$5);

##### **EXPERIMENT 7**

**7. Create Managed table and load a csv file from local storage. Verify the location of table data in HDFS and perform query operations. Delete the table and verify the data in HDFS.**

***Step 1: Create a Database.***

1. ***Create a database named “company” by running the create command:***

               create database company;

***1.2. Next, verify the database is created by running the show command:***

show databases;

***1.3. Find the “company” database in the list:***

***1.4. Open the “company” database by using the following command:***

use company;

***Step 2: Create a Table in Hive***

***2.1 .  Create*** ***an “employees.txt” file in the****/hdoop****directory.***

***2.2. Arrange the data from the “employees.txt” file in columns. The column names in our example are:***

* ***ID***
* ***Name***
* ***Country***
* ***Department***
* ***Salary***

***2.3. Use column names when creating a table. Create the table by running the following command:***

create table employees (id int, name string, country string, department string, salary int)

row format delimited fields terminated by '-';

***The terminal prints out a confirmation message after creating the table successfully.***

***2.4. Create a logical schema that arranges data from the .txt file to the corresponding columns. In the “employees.txt” file, data is separated by a '-'. To create a logical schema type:***

***2.5. Verify if the table is created by running the show command:***

show tables;

***Step 3: Load Data from a File***

***3.1. Load data by running the load command:***

load data inpath '/hdoop/employees.txt' overwrite into table employees;

***3.2. Verify if the data is loaded by running the select command:***

select \* from employees;

***The terminal prints out data imported from the employees.txt file.***

##### **EXPERIMENT 8**

**8. Create External table and load a csv file from local storage. The data should be stored in ORC format in specified folder in HDFS. Perform query operations and verify the data in HDFS. Delete the table and verify the data in HDFS.**

**Step 1: Prepare the Data File**

***1. Create a CSV file titled ‘countries.csv’***

sudo nano countries.csv

***2. For each country in the list, write a row number, the country’s name, its capital city, and its population in millions***

1,USA,Washington,328

2,France,Paris,67

3,Spain,Madrid,47

4,Russia,Moscow,145

5,Indonesia,Jakarta,267

6,Nigeria,Abuja,196

***3. Save the file and make a note of its location.***

**Step 2: Import the File to HDFS**

***1. Create an HDFS directory. You will use this directory as an HDFS location of the file you created.***

hdfs dfs -mkdir [hdfs-directory-name]

***2. Import the CSV file into HDFS:***

hdfs dfs -put [original-file-location] [hdfs-directory-name]

***3. Use the****-ls****command to verify that the file is in the HDFS folder:***

hdfs dfs -ls [hdfs-directory-name]

**Step 3: Create an External Table**

***1. After you import the data file to HDFS, initiate Hive and use the syntax explained above to create an external table.***

***2. To verify that the external table creation was successful, type:***

select \* from [external-table-name];

***The output should list the data from the CSV file you imported into the table***

***3. If you wish to create a managed table using the data from an external table, type:***

create table if not exists [managed-table-name](

[column1-name] [column1-type], [column2-name] [var2-name], …)

comment '[comment]';

***4. Next, import the data from the external table:***

insert overwrite table [managed-table-name] select \* from [external-table-name];

***5. Verify that the data is successfully inserted into the managed table.***

select \* from [managed-table-name];

##### **EXPERIMENT 9**

1. **Create partitioned table using Static partitioning technique load the data into corresponding partitions.**

**Steps for creating static partitioning using an example:**

1. create table student (id int, name string, age int,  institute string)

      partitioned by (course string)

      row format delimited

      fields terminated by ‘,’

      lines terminated by ‘\n’;

2.      load data local inpath ‘home/hduser/tvgstcousrse’ into table st\_part\_course partition(course=”java”);

3.      load data local inpath ‘home/hduser/tvgstcousrse’ into table st\_part\_course partition(course=”hadoop”);

4.      load data local inpath ‘home/hduser/tvgstcousrse’ into table st\_part\_course partition(course=”hive”);

5.      browse the directory /apps/hive/warehouse/tvgdb.db/st\_part\_course using

hdfs dfs -ls /apps/hive/warehouse/tvgdb.db/st\_part\_course

* go to hive, and browse the complete data using

select \* from st\_part\_course;

* browse the data of a partition by giving

select \* from st\_part\_course where course=”java”;

*Procedure for creating dynamic partitioning using an example*

* Use database;
* Enable dynamic partitioning by using the following commands:

     hive> set hive.exec.dynamic.partition = true;

     hive>set hive.exec.dynamic.partition.mode = nonstrict;

* Create a dummy table to store the data
* Now load data into the table
* Create partitioning table using the following command

      create table table-name(col1 datatype,col2 datatype,…)

partitioned by(col datatype)

row format delimited

fields terminated by ‘ ‘

lines terminated by ‘ ‘

* Now insert data of dummy table into partition table

insert into partition-table

partition (col)

select list of fields from dummytable-name

##### **EXPERIMENT 10**

1. **Create and manage a table with bucketing concept and execute queries with UDFs.**

**Procedure for creating Bucketing example**

* **set hive.enforce.bucketing = true;**
* **Create a bucketed table using**

CREATE TABLE bucketed\_student(firstname VARCHAR(64), lastname  VARCHAR(64),  
       city  VARCHAR(64), state  VARCHAR(64))  COMMENT ‘A bucketed sorted user table’  
       PARTITIONED BY (country VARCHAR(64)) CLUSTERED BY (state) SORTED BY (city) INTO 4 BUCKETS  
        STORED AS SEQUENCEFILE;

* **Creating a temporary table**

CREATE TEMPORARY TABLE temp\_user(firstname VARCHAR(64), lastname VARCHAR(64), country VARCHAR(64), city VARCHAR(64), state VARCHAR(64))

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n’ STORED AS TEXTFILE;

* **Load data into the temporary table from a file .**

LOAD DATA LOCAL INPATH '/home/user/user\_table.txt' INTO TABLE temp\_user;

* **Inserting data into bucketed table**

INSERT OVERWRITE TABLE bucketed\_user PARTITION (country)

SELECT firstname, lastname, city, state, country FROM temp\_user;

##### **EXPERIMENT 11**

1. **Implement flume script for spooling a directory from local file system to HDFS.**

local.conf:

a1.sources = src-1

a1.channels = c1

a1.sinks = k1

a1.sources.src-1.type = spooldir

a1.sources.src-1.spoolDir = /home/cloudera/Desktop/flumetest

a1.sources.src-1.fileHeader = true

a1.sources.src-1.channels = c1

a1.channels.c1.type = memory

a1.channels.c1.capacity = 1000

a1.sinks.k1.type = hdfs

a1.sinks.k1.hdfs.path = /flume/events/%y-%m-%d/%H%M/%S

a1.sinks.k1.hdfs.filePrefix = events-

a1.sinks.k1.hdfs.round = true

a1.sinks.k1.hdfs.roundValue = 10

a1.sinks.k1.hdfs.roundUnit = minute

a1.sinks.k1.hdfs.writeFormat = Text

a1.sinks.k1.hdfs.fileType = DataStream

a1.sinks.k1.hdfs.useLocalTimeStamp = true

a1.sources.src-1.channels = c1

a1.sinks.k1.channel = c1

flume-ng agent --conf conf --conf-file Desktop/local.conf --name a1 -Dflume.root.logger=INFO,console

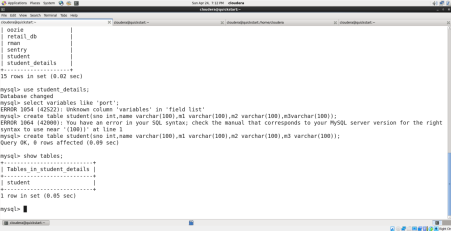
##### **EXPERIMENT 12**

**12. Import selective data from database into a specific directory in HDFS and analyze it. Export the summarized data into another table of database.**

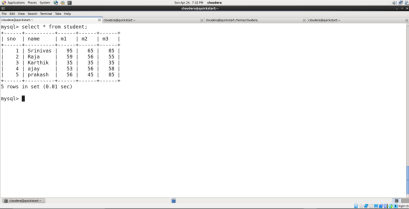
Create mysql table student with sno, name, m1, m2, m3 and populate  with data Import student table into HDFS at yourdirectory/query1.  Display the data from command prompt.

Create Table Query in MySQL:

mysql>CREATE TABLE student(sno int,name varchar(100) , m1  varchar(100) , m2 varchar(100), m3 varchar(100));



mysql>SELECT \* FROM student;

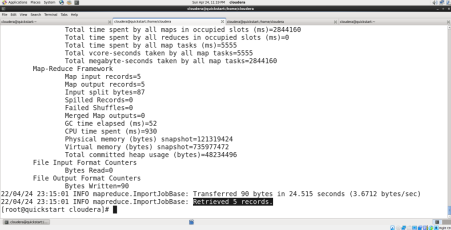
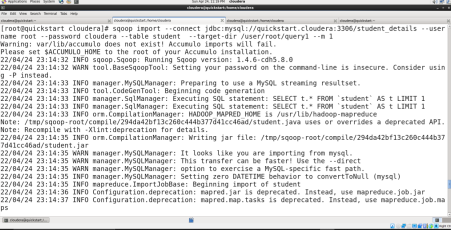


Import student table into HDFS

[root@quickstart cloudera]# sqoop import - -connect

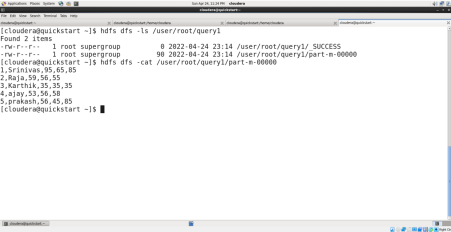
jdbc:mysql://quickstart.cloudera:3306/student\_details - -user root - - password cloudera - -table student

- -target-dir /user/root/query1 --m 1



Display the data from command prompt.

Hdfs command:

[cloudera@quickstart ~]$ hdfs dfs –cat /user/root/query1/part-m-00000

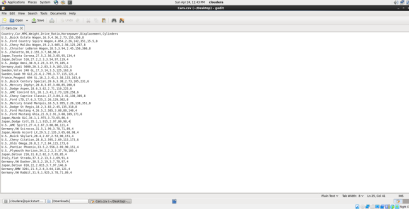
2. Cars.csv is available in the shared drive.

1. Create a suitable table “cars” in mysql for cars database. 2. Load the cars.csv into cars table

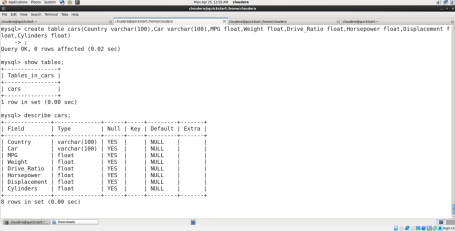
3. Import the cars data into hdfs .

a. Data should be stored in /user/cloudera/sqoopdata/cars b. Save the data in text format.

Create a suitable table “cars” in mysql for cars database. Cars.csv file

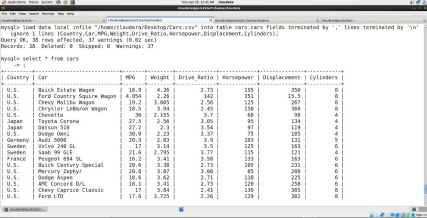


Mysql> CREATE TABLE CARS(Country varchar(100),Car varchar(100),MPG  float,Weight float,Drive\_Ratio float,Horsepower float,Displacement float,Cylinders  float);



Load the cars.csv into cars table

Mysql>load data local infile “/home/cloudera/Desktop/Cars.csv” into table  cars.cars fields terminated by ‘,’ line terminated by ‘\n’ ignore 1 lines  (Country,Car,MPG,Weight,Drive\_Ratio,Horsepower,Displacement,Cylinders);

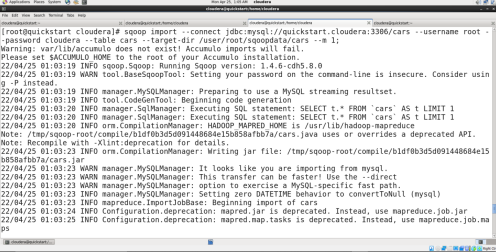


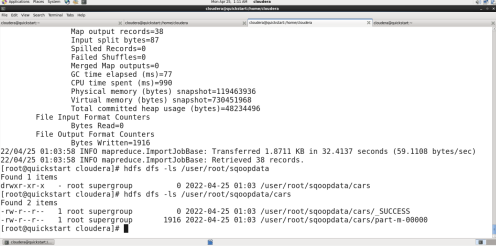
Import the cars data into hdfs.

Data should be stored in /user/cloudera/sqoopdata/cars

[root@quickstart cloudera]# Sqoop import –connect

jdbc:mysql://quickstart.cloudera:3306/cars --username root --password  cloudera –table student –target-dir /user/root/sqoopdata/cars --m 1;



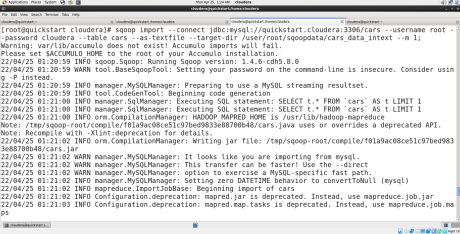


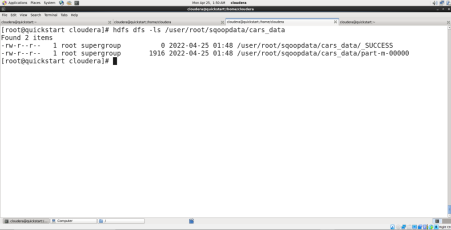
Save the data in text format.

[root@quickstart cloudera]#sqoop import –connect

jdbc:mysql://quickstart.cloudera:3306 /cars - -username root - -password  cloudera - -table cars - -as-textfile - -target-dir

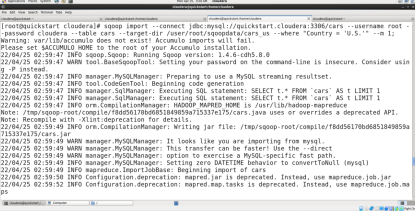
/user/root/sqoopdata/cars\_data\_intext --m 1;



  
Import the data for only US cars and Store the data in  /user/cloudera/sqoopdata/cars\_us

[root@quickstart cloudera]# sqoop import –connect  jdbc:mysql://quickstart.cloudera:3306/cars/ - -username root - -password cloudera  - -table cars - -target-dir /user/root/sqoopdata/cars\_us

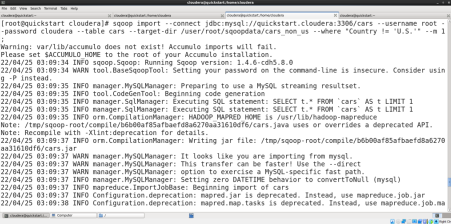
- -where “Country=’U.S’ ” - -m 1;

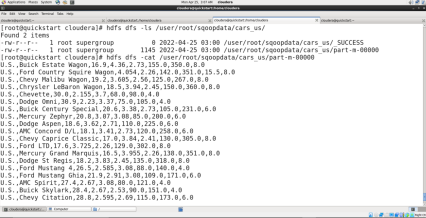


Import the data for all country cars except US and store the data  in/user/cloudera/sqoopdata/cars\_non\_us

[root@quickstart cloudera]# sqoop import –connect

jdbc:mysql://quickstart.cloudera:3306/cars/ - -username root - -password cloudera  - -table cars - -target-dir /user/root/sqoopdata/cars\_us - -where “Country !=’U.S’ ”  - -m 1;





3. Cars database is available in MySQL

1. An analysis team want to import the data into hdfs based on cylinders  with their average mileage per gallon.

2. They want to maintain the data in 2 parts for less than 5 cylinders and 5  or more cylinders separately.

3. They also want to store the resulting table into database back by  creating suitable table in database.

4. Solve this problem and display the records from 2 parts and mysql table.

An analysis team want to import the data into hdfs based on cylinders  with their average mileage per gallon.

[cloudera@quickstart ~]$ sqoop import - -connect

jdbc:mysql://quickstart.cloudera/cars - -username root - -password cloudera  - - target-dir sqoopdata/cars\_split\_by\_cylinders - -split-by “Cylinderts” –m 5

They want to maintain the data in 2 parts for less than 5 cylinders and 5  or more cylinders separately.

 More than 5 cylinders

[cloudera@quickstart ~]$ sqoop import –connect

jdbc:mysql://quickstart.cloudera:3306/cars - -username root - -password cloudera  - -query ‘select \* from cars where Cylinders>=5 and $CONDISIONS’  - -target-dir sqoopdata/cars/more\_\_than\_5 - - split-by “Cylinders” –m 1

 Less than 5 cylinders

[cloudera@quickstart ~]$ sqoop import –connect

jdbc:mysql://quickstart.cloudera:3306/cars - -username root - -password cloudera  - -query ‘select \* from table where Cylinders<5 and $CONDISIONS’

- -target-dir sqoopdata/cars/less\_\_than\_5 - - split-by “Cylinders” –m 1

They also want to store the resulting table into database back by creating  suitable table in database.

Mysql> CREATE TABLE less\_than\_5(Country varchar(100),Car varchar(100),MPG  float,Weight float,Drive\_Ratio float,Horsepower float,Displacement float,Cylinders float);

[cloudera@quickstart ~]$sqoop export - -connect

jdbc:mysql://quickstart.cloudera:3306/cars - -username root - -password cloudera  - -table less\_than\_5 - -export-dir sqoopdata/new\_car/lessthan\_5 -m 1

Mysql>select \* from less\_than\_5;

Mysql>create table more\_than\_5 like less\_than\_5;

[cloudera@quickstart ~]$sqoop export –connect

jdbc:mysql://quickstart.cloudera:3306/cars –username root –password cloudera –table  more\_than\_5 –export-dir sqoopdata/new\_car/more\_than\_5 –m 1

Mysql>select \* from more\_than\_5;

##### **EXPERIMENT 13**

1. **Load data into RDD and perform Transformations and Actions in Spark.**

student marks are stored in hdfs://Hmaster/training/dump/stdmarks1.txt

Input format: sno, name, m1, m2, m3, branch

create an rdd and display the student names of students belongs to branch: cse

Display the names of students using println.

format of output:

xxxx

yyyy

...

val stdrdd = sc.textFile("hdfs://Hmaster/training/dump/stdmarks1.txt")

val stdfilter = stdrdd.filter(x => x.toString.split(",")(5).equals("cse"))

val stdresult= stdfilter.map(x=> x.toString.split(",")(1))

stdresult.collect.foreach(println)

System.exit(0)

1. Employee data is available in

hdfs://Hmaster/training/datasets/avgsalassignment/empdata.csv

Write the spark script find the average salary of all employees and display the result.

Note:  String can be converted as Int by using String.toInt function.

        Any value can be converted as String by using toString function.

Output format:

xxxxxx

val emprdd = sc.textFile("hdfs://Hmaster/training/datasets/avgsalassignment/empdata.csv")

val empsals=emprdd.map(x=>x.split(",")(2).trim())

val sume=empsals.reduce((x,y)=> (x.toInt+y.toInt).toString)

val avge=sume.toInt/empsals.count

println(avge)

System.exit(0)

##### **EXPERIMENT 14**

1. **Implement spark script to demonstrate shared variables and broadcast variables.**

<https://sparkbyexamples.com/spark/spark-broadcast-variables/>

<https://sparkbyexamples.com/spark/spark-accumulators/>

import org.apache.spark.sql.SparkSession

object RDDBroadcast extends App {

  val spark = SparkSession.builder()

    .appName("SparkByExamples.com")

    .master("local")

    .getOrCreate()

  val states = Map(("NY","New York"),("CA","California"),("FL","Florida"))

  val countries = Map(("USA","United States of America"),("IN","India"))

  val broadcastStates = spark.sparkContext.broadcast(states)

  val broadcastCountries = spark.sparkContext.broadcast(countries)

  val data = Seq(("James","Smith","USA","CA"),

    ("Michael","Rose","USA","NY"),

    ("Robert","Williams","USA","CA"),

    ("Maria","Jones","USA","FL")

  )

  val rdd = spark.sparkContext.parallelize(data)

  val rdd2 = rdd.map(f=>{

    val country = f.\_3

    val state = f.\_4

    val fullCountry = broadcastCountries.value.get(country).get

    val fullState = broadcastStates.value.get(state).get

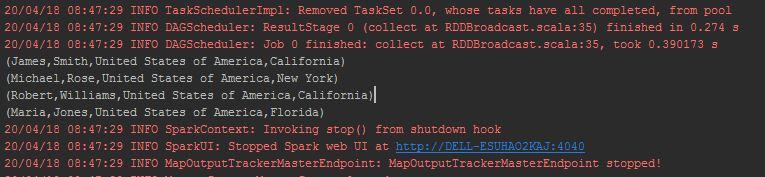
    (f.\_1,f.\_2,fullCountry,fullState)

  })

  println(rdd2.collect().mkString("\n"))

}

Output:



Shared variables: Accumulator

scala> val accum = sc.longAccumulator("SumAccumulator")

accum: org.apache.spark.util.LongAccumulator = LongAccumulator(id: 0, name: Some(SumAccumulator), value: 0)

scala> sc.parallelize(Array(1, 2, 3)).foreach(x => accum.add(x))

-----

-----

scala> accum.value

res2: Long = 6

//Long Accumulator

def longAccumulator : org.apache.spark.util.LongAccumulator

def longAccumulator(name : scala.Predef.String) : org.apache.spark.util.LongAccumulator

  val spark = SparkSession.builder()

    .appName("SparkByExample")

    .master("local")

    .getOrCreate()

  val longAcc = spark.sparkContext.longAccumulator("SumAccumulator")

  val rdd = spark.sparkContext.parallelize(Array(1, 2, 3))

  rdd.foreach(x => longAcc.add(x))

  println(longAcc.value)

##### **EXPERIMENT 15**

1. **Load and analyze data using spark sql using sqlcontext and hive context.**

SQL-1

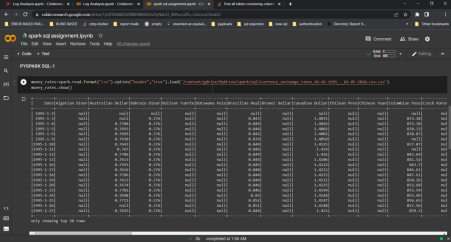
(using google Colab,PySpark)

1. Write and execute a spark script to be run in the spark shell that satisfies the following  criteria:

1.1. The input is all of the rows from the exchangedata table

money\_rates=spark.read.format("csv").option("header","true").load('/content/gdrive/M yDrive/spark/sql/currency\_exchange\_rates\_02-01-1995\_-\_02-05-2018.csv.csv')

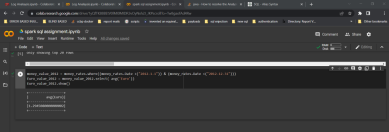
money\_rates.show()

1.2. Use spark SQL to compute the average of all the values of euro for the year 2012

money\_value\_2012 = money\_rates.where((money\_rates.Date >("2012-1- 1")) & (money\_rates.Date <("2012-12-31")))

Euro\_value\_2012 = money\_value\_2012.select( avg('Euro'))

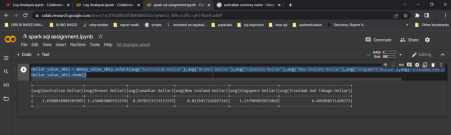
Euro\_value\_2012.show()



1.3. Similarly, compute the average of all the values of dollar for the year 2012

Dollar\_value\_2012 = money\_value\_2012.select(avg("Australian Dollar"),avg("Brunei Dollar"),avg(" Canadian Dollar"),avg("New Zealand Dollar"),avg("Singapore Dollar"),avg("Trinidad And Tobago D ollar"))

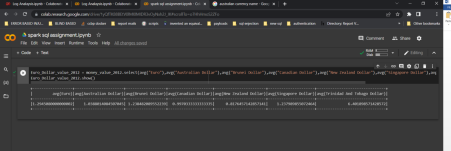
Dollar\_value\_2012.show()



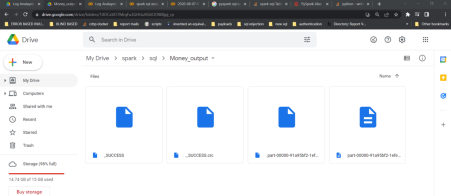
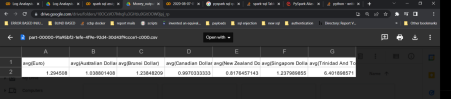
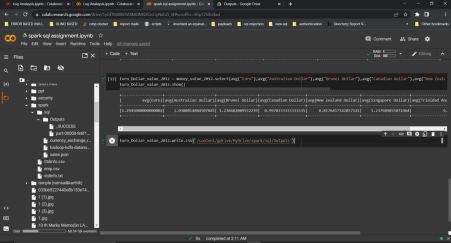
1.4. The output consists for two values separated by a tab: the average of euro for the  year 2012 and the average of dollar for the year 2012

Euro\_Dollar\_value\_2012 = money\_value\_2012.select(avg("Euro"),avg("Australian Dollar"), avg("Brunei Dollar"),avg("Canadian Dollar"),avg("New Zealand Dollar"),avg("Singapore Do llar"),avg("Trinidad And Tobago Dollar"))

Euro\_Dollar\_value\_2012.show()



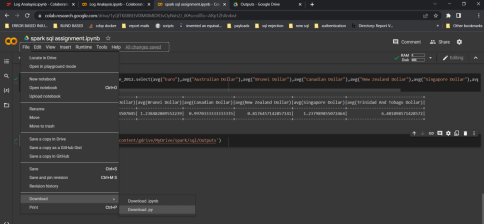
1.5. Output the result to the /user/hduser/task5 folder in HDFS

Euro\_Dollar\_value\_2012.write.csv('/content/gdrive/MyDrive/spark/sql/Outputs')

1.6. Save your script in the existing task5 file in the /home/hduser/solutions/. Folder in  Ubuntu client

To save the script in Google CoLab:

File🡪 Download🡪select any file format(.py & .ipynb)



SQL-2

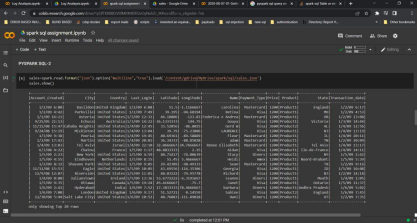
(using google Colab,PySpark)

Write and execute a spark script to be run in spark shell that satisfies the following criteria:  2.1. The input is all the data stored in the /user/hduser/balances folder in HDFS.

2.2. Each row in the output consists of two fields separated by a comma: a Payment\_Type, and the  average payment for payment type in that payment\_type whose price is greater than or equal to  $1200. Your output should consists of number of unique payment\_type records.

sales=spark.read.format("json").option("multiline","true").load('/content/gdrive/MyDrive/spark/ sql/sales.json')

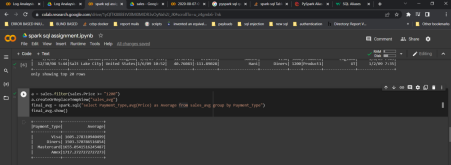
sales.show()

a = sales.filter(sales.Price >= "1200")

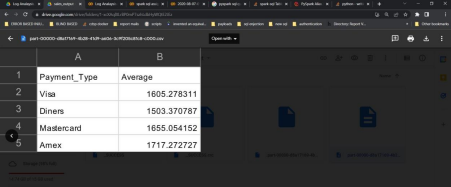
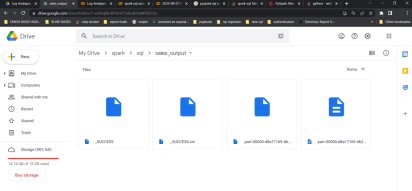
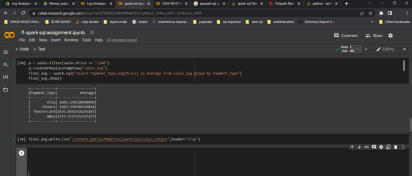
a.createOrReplaceTempView("sales\_avg")

spark.sql("select Payment\_Type,avg(Price) as

Average from sales\_avg group by Payment\_Type").show()



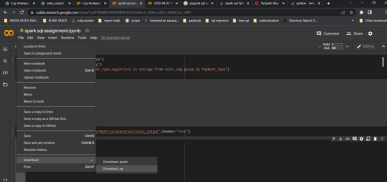
2.3. The output is saved in a folder in HDFS named /user/hduser/task2

final\_avg.write.csv("/content/gdrive/MyDrive/spark/sql/sales\_output",header=’true’)

2.4. Save your script in the existing task2 file in the /home/hduser/solutions/ folder on the  Ubuntu client

To save the script in Google CoLab:

File🡪 Download🡪select any file format(.py & .ipynb)



SQL-3

(using google Colab,PySpark)

Write and execute a spark script to be run in the spark shell that satisfies the following  criteria:

3.1. The input is /user/hduser/hadoop-hdfs-datanode-hmaster.log in HDFS

log\_file = "/content/gdrive/MyDrive/spark/sql/hadoop-hdfs-datanode-hmaster.log" spark = SparkSession.builder.appName("SimpleApp").getOrCreate()

logData = spark.read.text(log\_file).cache()



3.2. Using an accumulator, count the number of ERROR events in the log file. A log event is  an ERROR event if the string following the date and time equals ERROR

3.3. Output your result should be a single integer value, to the console using println  command

Error\_value = logData.filter(logData.value.contains("ERROR")).count() print("Error Lines :%i" %(Error\_value))

