Roll no : 220340325006

Name: Amit Singh

Hive

Q1) Write a program to find the count of customers for each profession.

use amitveersingh;

create table customers (cust\_no int, firstname string, lastname string,age int,profession string)

> row format delimited

>

> fields terminated by ','

>

> stored as textfile;

load data local inpath "custs.txt" overwrite into table customers;

query:

select profession,count(\*) as cust\_count from customers group by profession order by cust\_count;

output:

Total MapReduce CPU Time Spent: 5 seconds 630 msec

OK

Social Worker 1

Writer 101

Artist 175

Environmental scientist 176

Carpenter 181

Dancer 185

Therapist 187

Economist 189

Real estate agent 191

Electrical engineer 192

Nurse 192

Civil engineer 193

Automotive mechanic 193

Psychologist 194

Electrician 194

Agricultural and food scientist 195

Athlete 196

Statistician 196

Judge 196

Doctor 197

Financial analyst 198

Accountant 199

Reporter 200

Secretary 200

Coach 201

Physicist 201

Farmer 201

Actor 202

Architect 203

Computer hardware engineer 204

Teacher 204

Engineering technician 204

Designer 205

Musician 205

Childcare worker 207

Veterinarian 208

Chemist 209

Police officer 210

Recreation and fitness worker 210

Lawyer 212

Social worker 212

Pilot 212

Human resources assistant 212

Pharmacist 213

Computer software engineer 216

Firefighter 217

Librarian 218

Loan officer 221

Photographer 222

Computer support specialist 222

Politician 228

Q2) Write a program to find the top 10 products sales wise.

create table txnrecord (txnno INT, txndate STRING, custno INT, amount DOUBLE,

category STRING, product STRING, city STRING, state STRING, spendby STRING)

row format delimited

fields terminated by ','

stored as textfile;

load data local inpath "txns1.txt" overwrite into table txnrecord;

query:

select product,round(sum(amount),2) as amt from txnrecord group by product order by amt desc limit 10;

output:



Q3) Write a program to create partiioned table on category

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

set hive.exec.dynamic.partition=true;

create table txnbycat(txnno INT, txndate STRING, custno INT, amount DOUBLE,

product STRING, city STRING, state STRING, spendby STRING)

partitioned by (category STRING)

row format delimited

fields terminated by ','

stored as textfile;

hive (amitveersingh)> desc txnbycat;

OK

txnno int

txndate string

custno int

amount double

product string

city string

state string

spendy string

category string

# Partition Information

# col\_name data\_type

INSERT OVERWRITE TABLE txnbycat PARTITION(category) select txn.txnno, txn.txndate,txn.custno, txn.amount,txn.product,txn.city,txn.state, txn.spendby, txn.category from txnrecords txn DISTRIBUTE By category;

PySpark

1. What was the highest number of people travelled in which year?

dataRDD=sc.textFile("/user/bigdatamind43857/airlines.csv")

>>> dataRDD2=dataRDD.map(lambda a : a.encode("ascii","ignore"))

>>> header=dataRDD2.first()

>>> dataRDD3=dataRDD2.map(lambda a : a != header)

>>> dataRDD4=dataRDD3.map(lambda a : a.split(","))

>>> keyword=dataRDD4.map(lambda a : (a[0],int(a[3])))

>>> count=keyword.reduceByKey(lambda a,b : a+b)

>>> sortbyval=count.sortBy(lambda a : -a[1])

>>> for line in sortbyval.collect():

>>> print(line)

>>>

Output:

('2007', 176299)

('2013', 173676)

('2001', 173598)

('1996', 167223)

('2008', 166897)

('2012', 166076)

('2015', 165438)

('2004', 164800)

('2010', 163741)

('2014', 159823)

('1997', 157972)

('2003', 156153)

('2000', 154376)

('2006', 153789)

('2002', 152195)

('2005', 150610)

('2009', 150308)

('1999', 150000)

('1995', 148520)

('2011', 142647)

('1998', 135678)

Q2) 2) Identifying the highest revenue generation for which year.

>>> key\_value = arrayRDD.map(lambda a : (a[0] ,float(a[2]) \* float(a[3])))

>>> add\_total = key\_value.reduceByKey(lambda a,b : a+b)

>>> sortbyval = add\_total.sortBy(lambda a : -a[1])

>>> for i in sortbyval.collect():

... print(i)

Output:

('2013', 66363208.71)

('2014', 62624175.85000001)

('2015', 62378990.57)

('2012', 62199127.28)

('2008', 57653170.760000005)

('2007', 57309216.07)

('2001', 55533779.99999999)

('2010', 54861521.29)

('2000', 52342926.550000004)

('2011', 51888286.22)

('2004', 50631364.949999996)

('2006', 50437898.419999994)

('2003', 49273210.83)

('1999', 48757714.48)

('2002', 47499146.5)

('2009', 46746446.59)

('2005', 46376786.24)

('1996', 46358778.03)

('1997', 45385236.16)

('1995', 43494243.22)

('1998', 42035717.78)

Q3) Identifying the highest revenue generation for which year and quarter (Common group)

>>> key\_value = arrayRDD.map(lambda a : ((a[0]+" "+a[1]),float(a[2]) \* float(a[3])))

>>> add\_total = key\_value.reduceByKey(lambda a,b : a+b)

>>> sortbyval = add\_total.sortBy(lambda a : -a[1])

>>> for i in sortbyval.collect():

... print(i)

Output:

('2014 4', 18819408.48)

('2013 1', 18572613.990000002)

('2013 3', 18177814.2)

('2015 2', 17316167.61)

('2000 1', 16385136.57)

('2010 1', 16300345.36)

('2012 4', 16087025.010000002)

('2014 3', 15956667.09)

('2012 3', 15947048.32)

('1999 1', 15742058.22)

('2004 1', 15698315.06)

('2008 1', 15626301.65)

('2007 2', 15539975.62)

('2015 4', 15486025.28)

('2008 4', 15459962.940000001)

('2012 2', 15447962.530000001)

('2006 3', 15339355.92)

('2001 1', 15247249.57)

('2010 4', 15157499.28)

('2011 3', 15124745.98)

('2005 4', 14985094.08)

('2015 1', 14899061.76)

('2013 2', 14861070.0)

('2013 4', 14751710.520000001)

('2012 1', 14717091.42)

('2015 3', 14677735.920000002)

('2007 4', 14177801.61)

('2001 2', 14148586.16)

('2007 1', 14082536.879999999)

('2014 1', 13995861.6)

('2002 3', 13988802.6)

('2009 1', 13866450.52)

('2014 2', 13852238.68)

('1995 1', 13823960.899999999)

('2001 3', 13717715.399999999)

('2004 2', 13665002.549999999)

('2008 3', 13652979.34)

('1996 1', 13576037.760000002)

('2007 3', 13508901.96)

('1997 2', 13477773.6)

('2003 1', 13409491.09)

('2011 2', 13331400.16)

('2006 4', 13120918.4)

('2000 2', 12999663.64)

('2008 2', 12913926.83)

('2000 3', 12720698.100000001)

('2003 3', 12626803.799999999)

('2003 4', 12598591.459999999)

('1999 2', 12594949.62)

('2011 4', 12560625.44)

('2001 4', 12420228.87)

('2002 1', 12372293.219999999)

('2010 2', 12159615.360000001)

('2004 4', 12111781.76)

('1998 3', 12016699.5)

('1996 2', 11864055.6)

('2009 4', 11716105.5)

('2006 2', 11429949.959999999)

('2009 3', 11357456.95)

('2010 3', 11244061.29)

('1998 4', 11190558.74)

('2002 2', 11129457.58)

('1995 2', 11113082.4)

('1997 4', 10993123.54)

('2005 3', 10991040.39)

('1997 3', 10976351.219999999)

('2011 1', 10871514.64)

('2005 2', 10755267.6)

('2003 2', 10638324.479999999)

('2006 1', 10547674.139999999)

('1996 3', 10497174.48)

('1999 3', 10483486.56)

('1996 4', 10421510.19)

('2000 4', 10237428.24)

('2002 4', 10008593.100000001)

('1997 1', 9937987.799999999)

('1999 4', 9937220.08)

('1995 3', 9812141.28)

('2009 2', 9806433.62)

('2005 1', 9645384.17)

('1998 1', 9542933.1)

('1998 2', 9285526.440000001)

('2004 3', 9156265.58)

('1995 4', 8745058.639999999)

MapReduce

Question 2 : Find all time High price for each stock.

Java code:

import java.io.\*;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.DoubleWritable;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

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

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

import org.apache.hadoop.mapreduce.lib.input.\*;

import org.apache.hadoop.mapreduce.lib.output.\*;

public class alltimehigh {

public static class MapClass extends Mapper<LongWritable,Text,Text,DoubleWritable>

{

private Text stock\_id = new Text();

private DoubleWritable High = new DoubleWritable();

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

{

try{

String[] str = value.toString().split(",");

double high = Double.parseDouble(str[4]);

stock\_id.set(str[1]);

High.set(high);

//context.write(new Text(str[1]),new LongWritable(vol));

context.write(stock\_id, High);

}

catch(Exception e)

{

System.out.println(e.getMessage());

}

}

}

public static class ReduceClass extends Reducer<Text,DoubleWritable,Text,DoubleWritable>

{

private DoubleWritable result = new DoubleWritable();

public void reduce(Text key, Iterable<DoubleWritable> values,Context context) throws IOException, InterruptedException {

double maxValue=0;

double temp\_val=0;

for (DoubleWritable value : values) {

temp\_val = value.get();

if (temp\_val > maxValue) {

maxValue = temp\_val;

}

}

result.set(maxValue);

context.write(key, result);

//context.write(key, new LongWritable(sum));

}

}

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

Configuration conf = new Configuration();

//conf.set("name", "value")

//conf.set("mapreduce.input.fileinputformat.split.minsize", "134217728");

Job job = Job.getInstance(conf, "Highest Price for each stock");

job.setJarByClass(AllTimeHigh.class);

job.setMapperClass(MapClass.class);

//job.setCombinerClass(ReduceClass.class);

job.setReducerClass(ReduceClass.class);

job.setNumReduceTasks(1);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(DoubleWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

Output:

AA 94.62

AAI 57.88

AAN 35.21

AAP 83.65

AAR 25.25

AAV 24.78

AB 94.94

ABA 27.94

ABB 33.39

ABC 84.35

ABD 28.58

ABG 30.06

ABK 96.1

ABM 41.63

ABR 34.45

ABT 93.37

ABV 107.5

ABVT 100.0

ABX 54.74

ACC 37.0

ACE 104.0

ACF 64.9

ACG 12.63

ACH 111.6

ACI 112.89

ACL 178.56

ACM 38.25

ACN 44.03

ACO 42.7

ACS 109.55

ACV 65.32

ADC 37.7

ADI 185.5

ADM 48.95

ADP 84.31

ADS 80.79

ADX 40.56

ADY 44.0

AEA 23.94

AEB 26.5

AEC 17.6

AED 26.12

AEE 56.77

AEF 27.0

AEG 148.32

AEH 26.64

AEL 14.6

AEM 83.45

AEO 88.13

AEP 53.31

AER 32.82

AES 92.5

AET 154.67

AEV 26.78

AF 63.09

AFB 17.03

AFC 25.15

AFE 26.7

AFF 25.15

AFG 54.65

AFL 74.94

AFN 11.99

AGC 20.2

AGCO 71.95

AGD 25.5

AGL 44.67

AGM 80.0

AGN 125.0

AGO 31.99

AGP 80.89

AGU 113.88

AHC 16.35

AHD 47.12

AHL 30.8

AHS 37.4

AHT 13.48

AI 28.7

AIB 125.0

AIG 157.19

AIN 43.62

AIQ 15.4

AIR 46.58

AIT 59.0

AIV 65.79

AIZ 71.31

AJG 68.5

AKF 26.42

AKP 17.45

AKR 29.0

AKS 73.07

AKT 26.25

ALB 86.52

ALC 27.86

ALD 33.35

ALE 51.7

ALEX 44.52

ALF 26.75

ALG 29.23

ALJ 47.1

ALK 62.56

ALL 100.25

ALM 27.79

ALQ 28.5

ALU 86.25

ALV 65.09

ALX 471.0

ALY 28.1

ALZ 26.25

AM 62.88

AMB 66.86

AMD 97.0

AME 53.12

AMG 136.51

AMN 138.65

AMP 69.25

AMR 69.01

AMT 55.5

AMX 69.15

AN 53.93

ANF 101.5

ANH 16.65

ANN 53.06

ANR 119.3

ANW 48.63

AOB 14.48

AOD 21.85

AOI 23.38

AOL 27.0

AON 75.56

AOS 58.06

AP 54.46

APA 149.23

APB 36.14

APC 113.95

APD 106.06

APF 24.53

APH 121.06

APL 56.88

APU 42.94

APX 12.38

ARB 55.63

ARD 71.08

ARE 116.5

ARG 65.45

ARI 19.2

ARJ 48.02

ARK 8.29

ARL 22.25

ARM 32.5

ARO 47.82

ARP 39.0

ART 28.1

ARW 64.12

ASA 92.6

ASF 89.12

ASG 12.56

ASH 76.25

ASI 24.21

ASP 16.87

ASR 63.54

ASX 7.49

ATE 58.02

ATI 119.7

ATK 120.9

ATO 33.47

ATR 76.98

ATT 27.14

ATU 70.17

ATV 32.33

ATW 126.92

AU 62.2

AUO 28.5

AUY 19.93

AV 15.05

AVA 67.76

AVB 149.94

AVD 51.0

AVF 27.0

AVK 29.75

AVP 90.45

AVT 81.12

AVX 100.0

AVY 78.5

AWC 32.85

AWF 15.46

AWH 53.48

AWI 57.48

AWK 23.77

AWP 20.55

AWR 48.0

AXA 80.94

AXE 88.4

AXL 42.1

AXP 169.5

AXR 149.99

AXS 43.35

AYE 65.48

AYI 66.89

AYN 15.42

AYR 41.31

AZN 145.41

AZO 169.99

AZZ 59.2