INFO7250- Engg Of Big Data

Final Project

INFO7250- Engg Of Big Data

**Ankit Yadav**

**NUID- 001271369**

**Northeastern University**

**CONTENTS**

|  |  |
| --- | --- |
| Topics | PAge No |
| 0. Overview about the dataset | **2** |
| 1. Analysis of Flight Data using MapReduce on Hadoop | **4** |
| 2. Analysis of Flight Data using Apache PIG on Hadoop | **19** |
| 3. Analysis of Flight Data using Apache HIVE on Hadoop | **19** |
| 4. References | **26** |
| 5. APPENDICES | **27** |

**0. Overview about the dataset**

I am using the data on **Airline On-Time Statistics and Delay Causes from**

<http://stat-computing.org/dataexpo/2009/the-data.html>

This is dataset containing information about airline schedule with following columns:

**Variable descriptions**

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Description** |
| 1 | Year | 1987-2008 |
| 2 | Month | 1-12 |
| 3 | DayofMonth | 1-31 |
| 4 | DayOfWeek | 1 (Monday) - 7 (Sunday) |
| 5 | DepTime | actual departure time (local, hhmm) |
| 6 | CRSDepTime | scheduled departure time (local, hhmm) |
| 7 | ArrTime | actual arrival time (local, hhmm) |
| 8 | CRSArrTime | scheduled arrival time (local, hhmm) |
| 9 | UniqueCarrier | [unique carrier code](http://stat-computing.org/dataexpo/2009/supplemental-data.html) |
| 10 | FlightNum | flight number |
| 11 | TailNum | plane tail number |
| 12 | ActualElapsedTime | in minutes |
| 13 | CRSElapsedTime | in minutes |
| 14 | AirTime | in minutes |
| 15 | ArrDelay | arrival delay, in minutes |
| 16 | DepDelay | departure delay, in minutes |
| 17 | Origin | origin [IATA airport code](http://stat-computing.org/dataexpo/2009/supplemental-data.html) |
| 18 | Dest | destination [IATA airport code](http://stat-computing.org/dataexpo/2009/supplemental-data.html) |
| 19 | Distance | in miles |
| 20 | TaxiIn | taxi in time, in minutes |
| 21 | TaxiOut | taxi out time in minutes |
| 22 | Cancelled | was the flight cancelled? |
| 23 | CancellationCode | reason for cancellation (A = carrier, B = weather, C = NAS, D = security) |
| 24 | Diverted | 1 = yes, 0 = no |
| 25 | CarrierDelay | in minutes |
| 26 | WeatherDelay | in minutes |
| 27 | NASDelay | in minutes |
| 28 | SecurityDelay | in minutes |
| 29 | LateAircraftDelay | in minutes |

The reason of selections this data set is that it has many numbers of columns which will enable me to use various MapReduce algorithms studies in the course for different types of analysis.

Also, the data is evenly segregated in yearly basis. So, in case If I can am unable to load complete data in my computer then too I can do the same analysis on small portion of same data more easily.

**The complete data was downloaded using following script:**



**1. Analysis of Flight Data using MapReduce on Hadoop**

**1- Getting total count of all the data:**

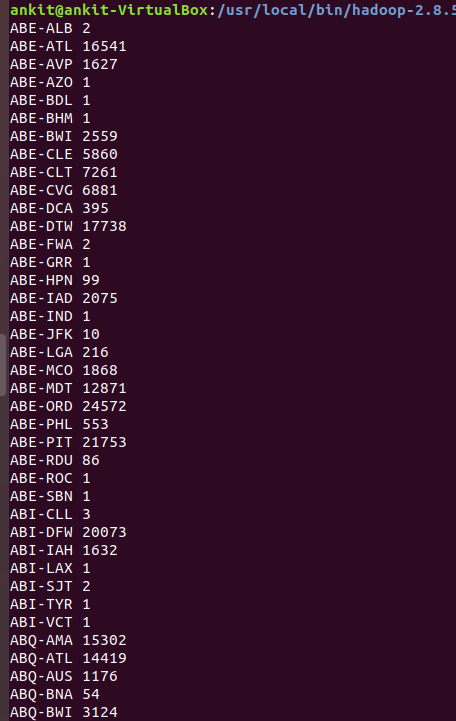
This is a very basic map reduce use case in which we count the whole data to get a sense of how many total records are there:

hadoop jar ~/Downloads/ProjectJars/count.jar hadoop.project.total\_count.MRCount /flight-data /FinalProjectMROutput/2-Total-Data-Count

The final count is: **123534970**

**2- Getting the total flights from all source destinations pairs in from 1987 to 2008:**

This was a huge data and MapReduce made this analysis quite simple and fast:



**3: Top 30 source destination pairs**

Sorting the above data to get top 30 most busy Source Destination pair:



**4: Delay in flight percentage**

We considered the delay greater than or equal to 15 minutes as delay . Now we need to count those flights which had delay greater than or equal to 15 minutes.

Delayed flight Count:

Percentage of departure delayed flights: Total Flight Count/ Departure Delayed Flight count

**= (19690422/123534970) \* 100 = 15.94 %**

So, this shows that the actual delay greater that 15 minutes is very less and generally flights depart on time.

Let’s check the same for arrival delay

Percentage of departure delayed flights: Total Flight Count/ Delayed Flight count

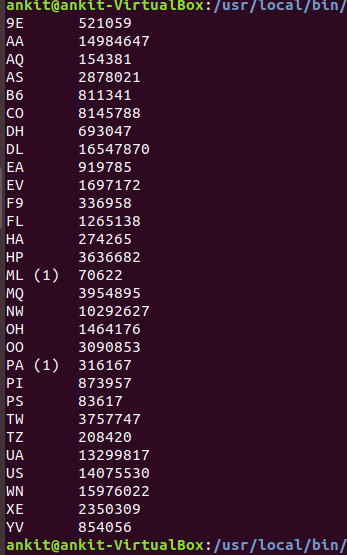
**= (24627925/123534970) \* 100 = 19.9 %**

**So, the delay in departure and arrival is between 15 to 20 % range.**

**So, it shows that overall flights are mostly on time from/to all source destination**

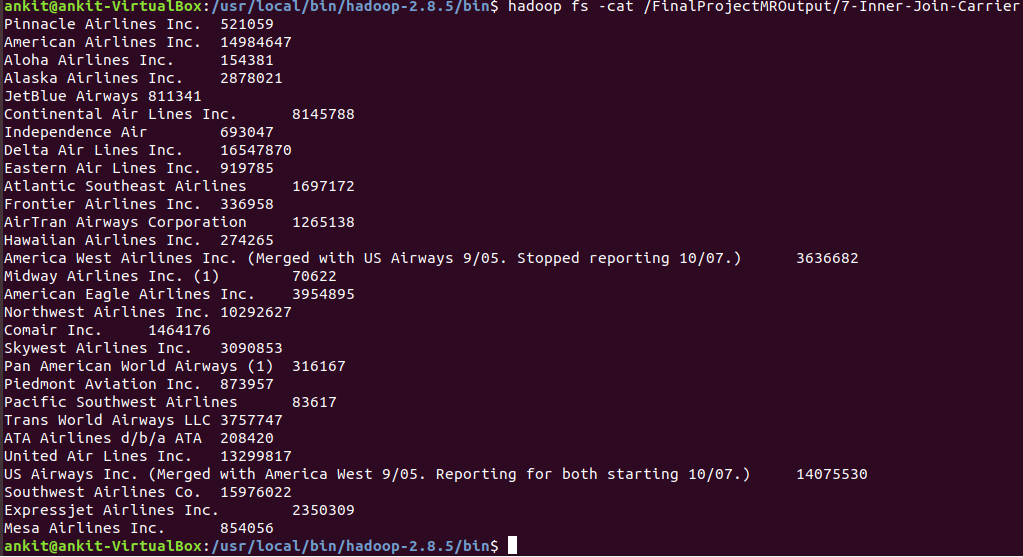
**5- Count of unique carrier’s flights**

The data for unique carriers are as follows:

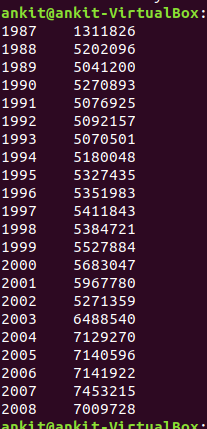


**6- Inner Join to get the full name for unique carriers**

We did inner join with between two files to get carrier names instead of carrier codes

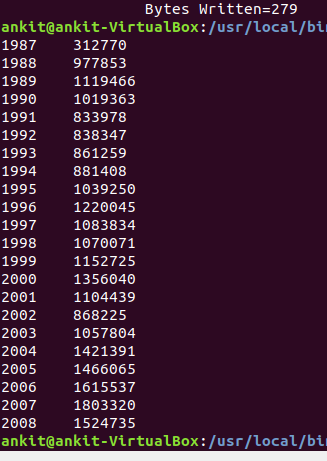


**7- Getting Flight data by year**

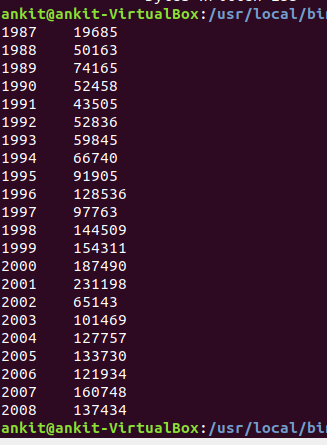


**8- Delayed flights per year**

In this we will check delayed flights per year( we will count flights as delayed only is the delay time if greater than equal to 15 minutes)

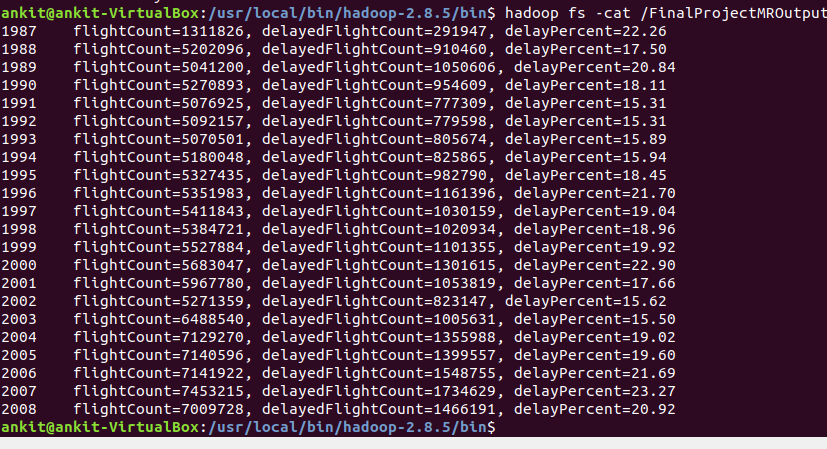


**9- Cancelled flights by year**



**10- Ratio of delayed flights per year to total flights**

We can get percentage of delayed flights per year also

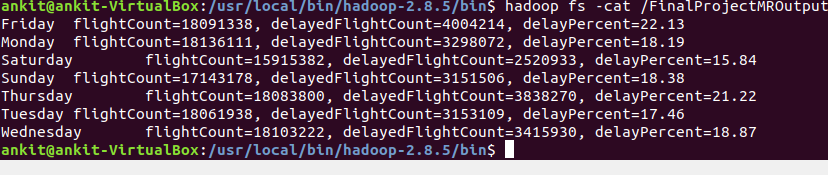


It shows that the years 1991-1994, 2002,2003 were best years to fly as they had least delayed flights (less than 16%).

Years with most delays were- 1987, 2000, 2007 with more than 22% flights delayed

**11- Total flights by day of week and ratio to delayed**

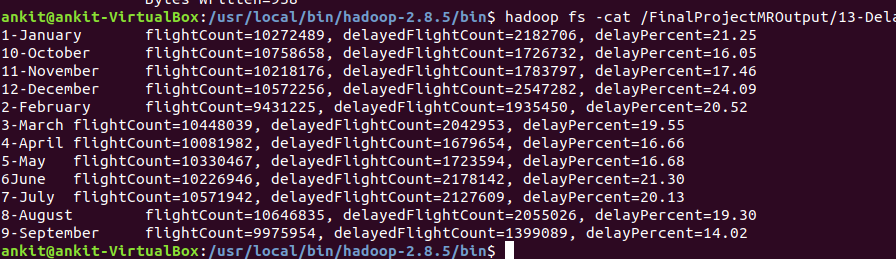
Following is the data of total flights , delayed flights and their ratio.



From this data we can infer that maximum delay is on Thursday and Fridays that is when weekends are starting .

Best day to fly are when the weekends ends like Saturday, Sunday or on weekdays

**12- Total flights by months of year and ratio to delayed**

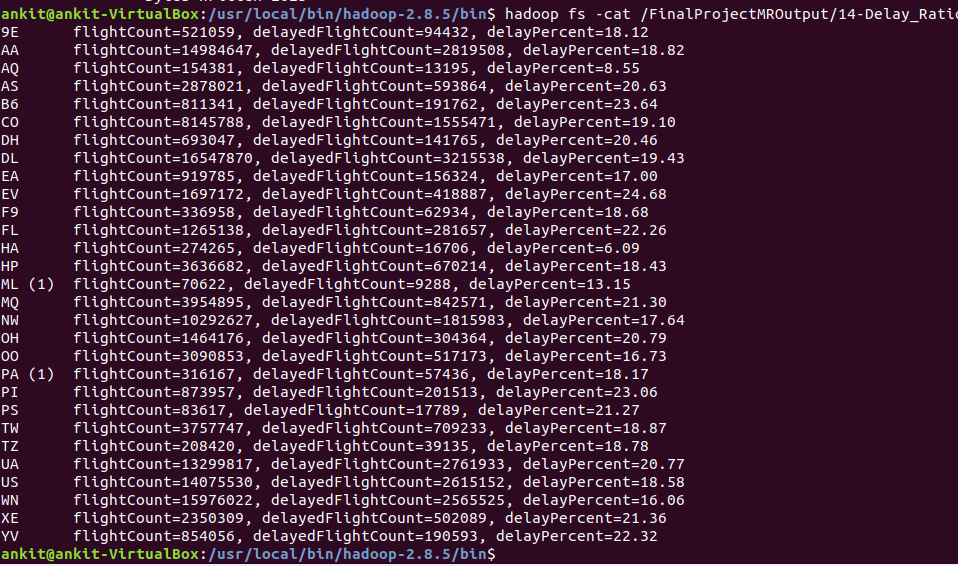


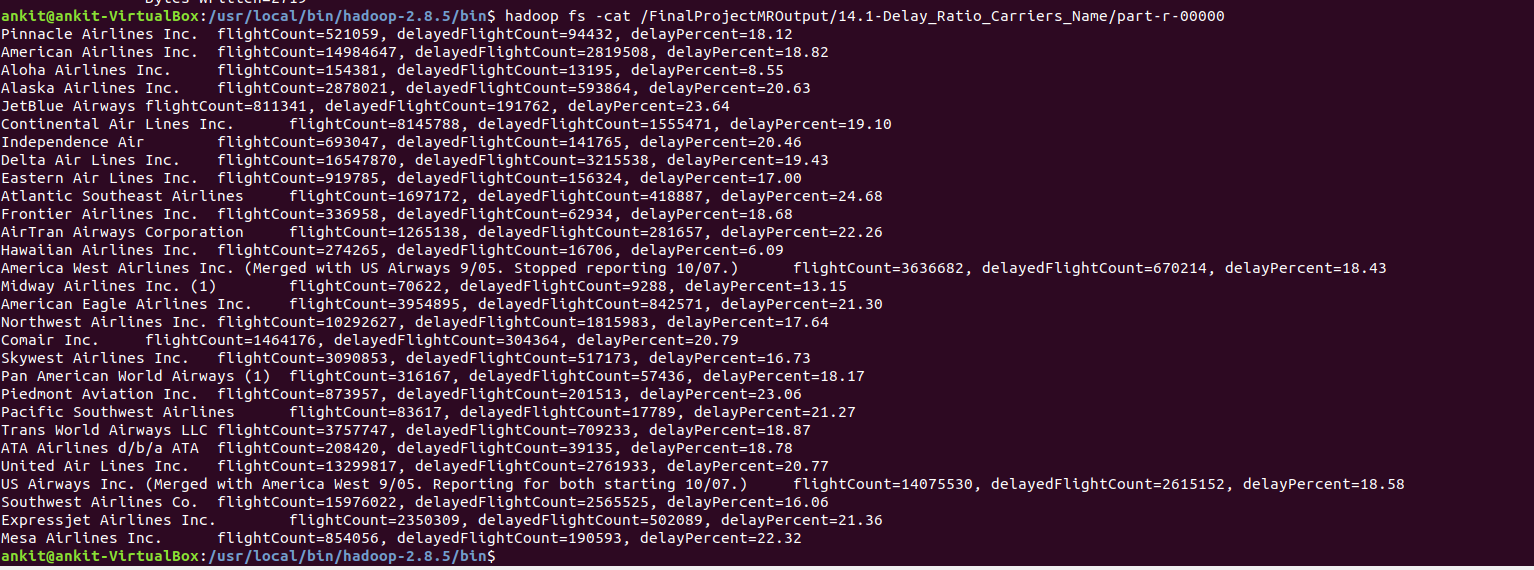
We can infer from this data that best months to fly was September with least delay- 14 %

Other good months were- April, May and October with delay – 16%

Worst month was December- 24% flights delayed. It may be due to big holidays season in December.

**13- Total delayed flights by flight carriers and ratio of delayed flights**



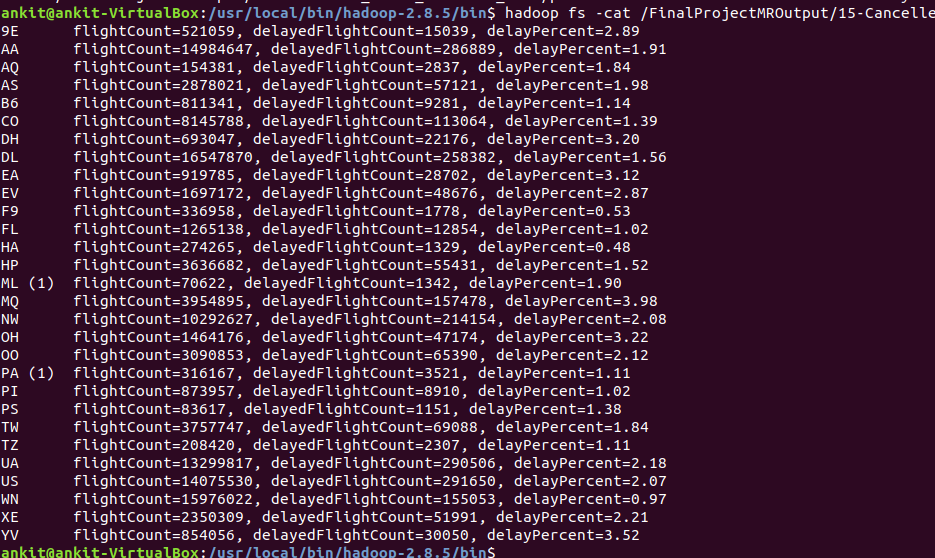


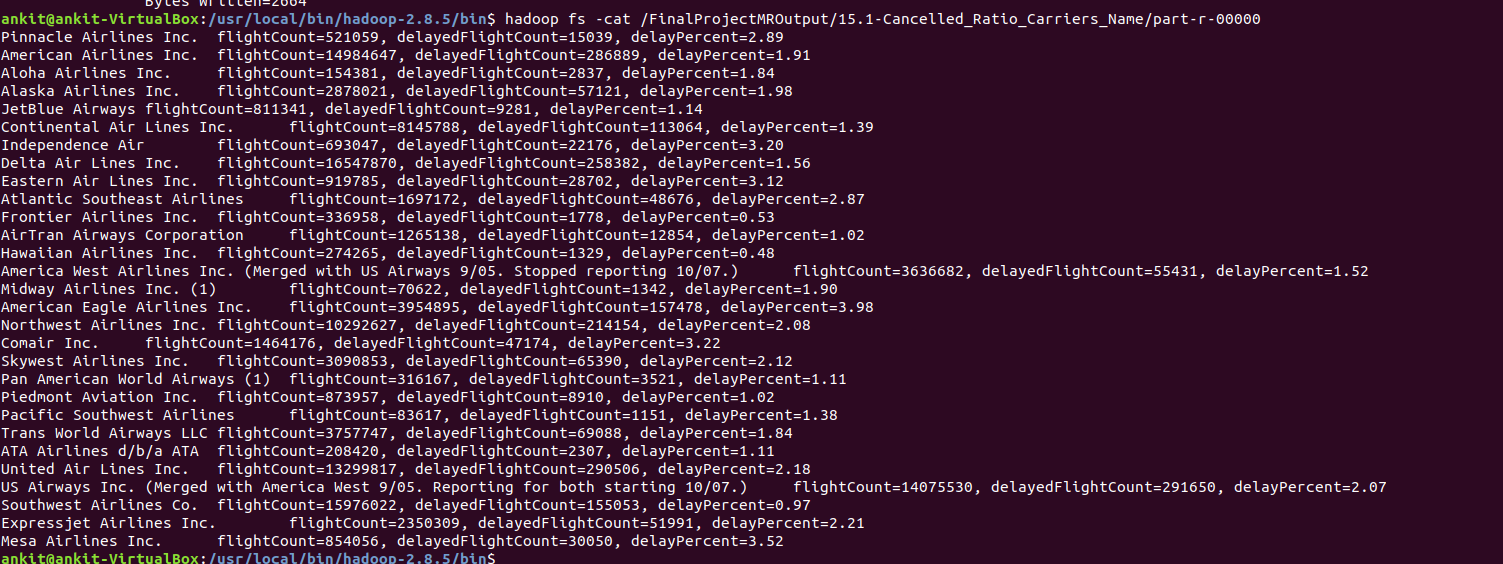
By using this analysis we can check which carriers are more prone to delays and can plan flights with those carriers who are less prone to delays.

Carriers with least delays- **Hawaiian Airlines, Aloha Airlines** with **6%** and **8%** flights delayed respectively.

Carriers with most delays- **JetBlue Airways, Atlantic Southeast Airlines** with around **24%** flights delayed.

**14- Total cancelled flights by flight carriers and ratio of cancelled flights**



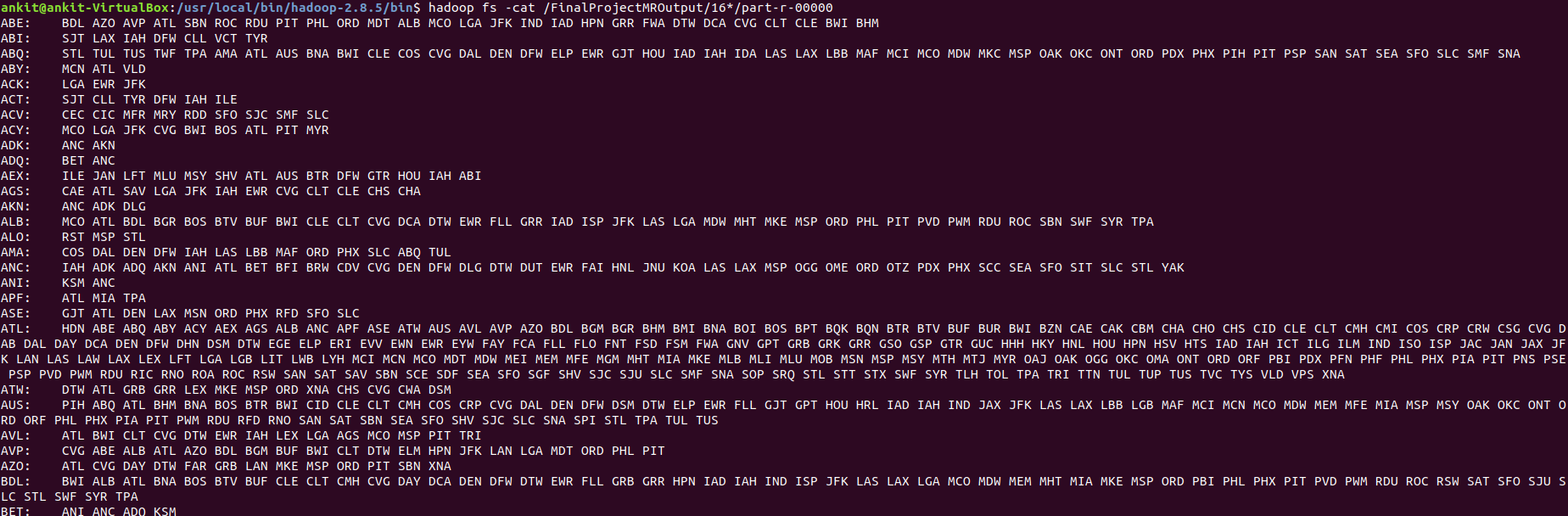


The number of cancelled flights are very less for almost all the carriers less than 4%.

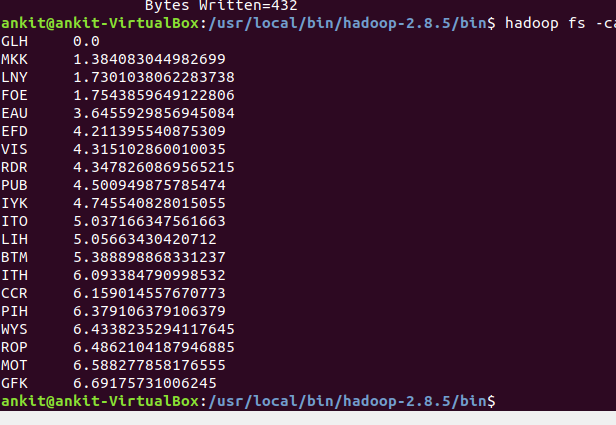
Among them best are **Frontier Airlines, Hawaiian Airlines** with **0.5%** cancelled flights and worst are **American Eagle Airlines, Mesa Airlines** with more than **3.5%** cancelled flights.

**15- Inverted index for all source and destination**

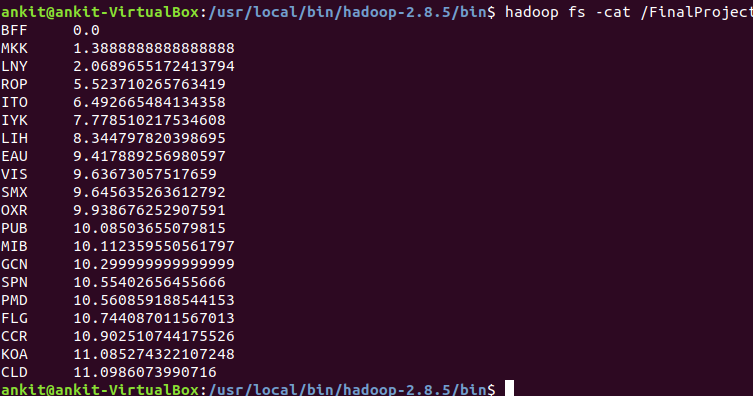
This data can help to search for all the destination stations from a particular source stations.



**16- Top 20 best source station with least departure delayed flight percent**



**17- Top 20 best destination station with least arrival delayed flight percent**



**18- Delay groups- grouping amount of flights per delay groups**



Between 1 hour and 2 hour **2.93**

Between 15 and 30 minutes **7.33**

Between 30 minutes and 1 hour **4.55**

Less than 15 Minutes **84.06**

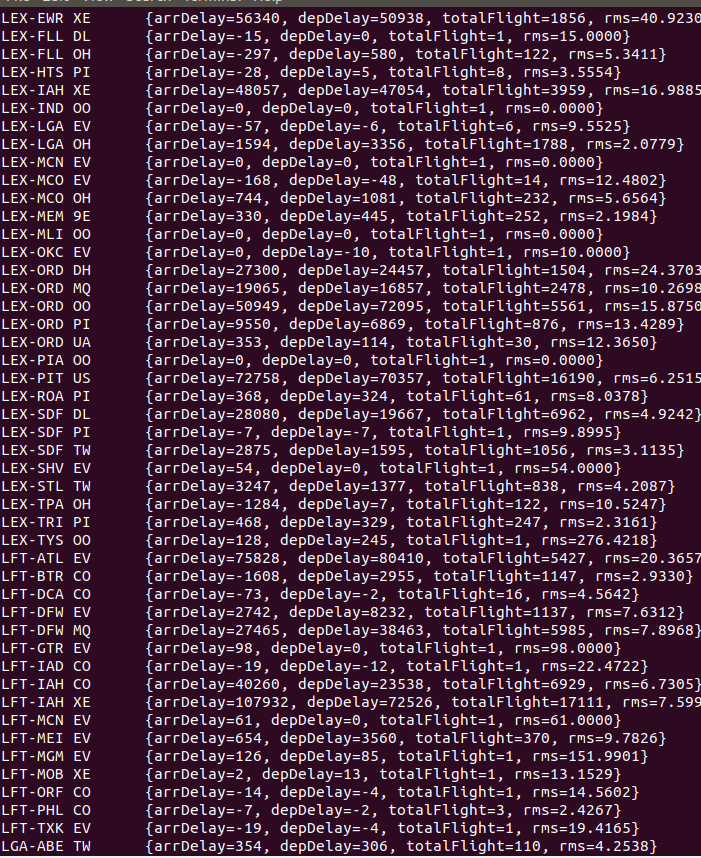
More than 2 hours **1.13**

**19- Recommendation system- Best carrier for a source destination route**

I first calculate average arrival delay and average departure delay for each source destination pair for each carrier.

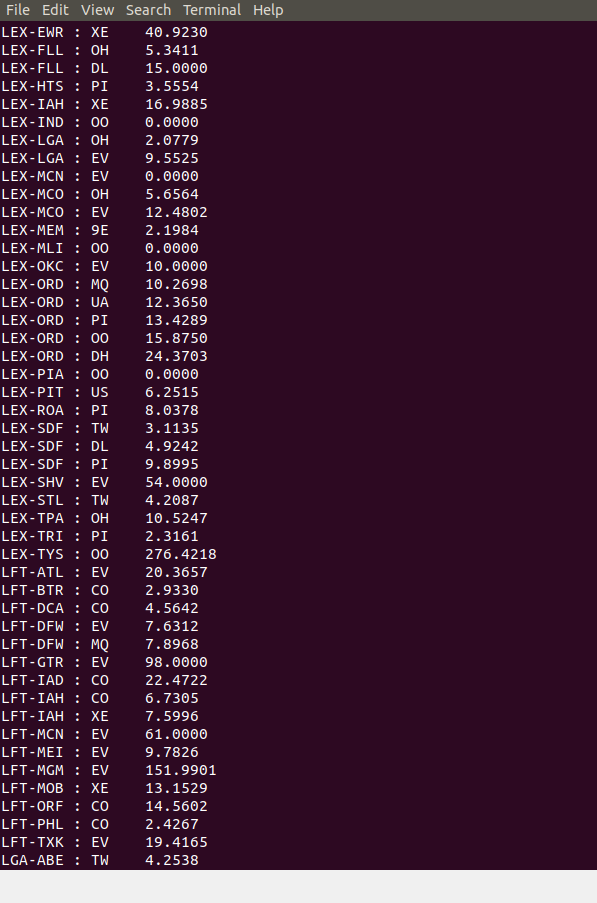
After this I calculate root mean square value for average arrival delay and average departure delay.

**Rms =**

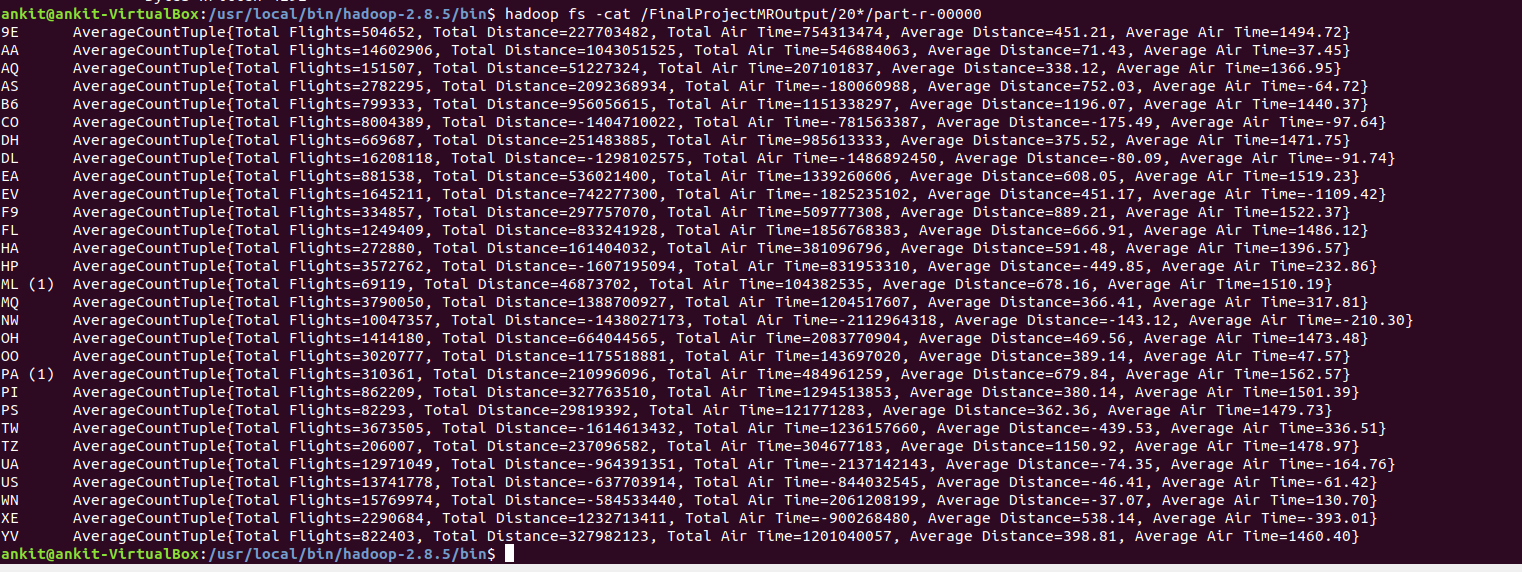


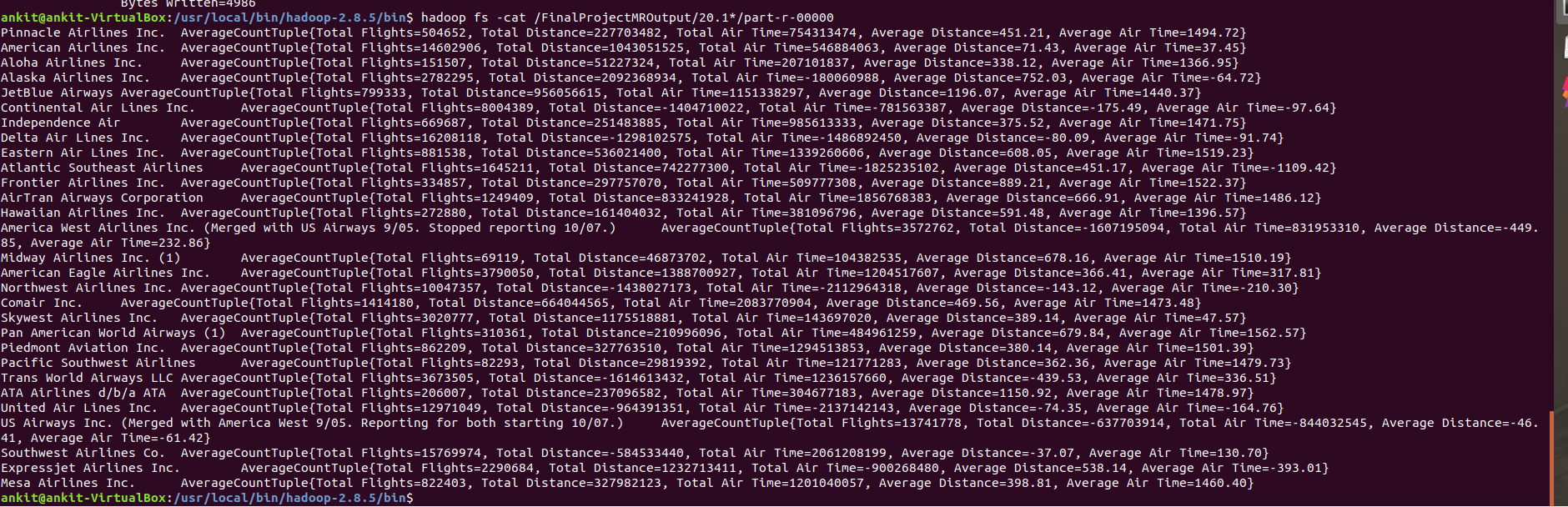
After this I sorted the carriers for all source destination pair in ascending order by RMS value.

It gives user a recommendation for choosing a carrier between a source destination pair with least arrival and/or departure delay.

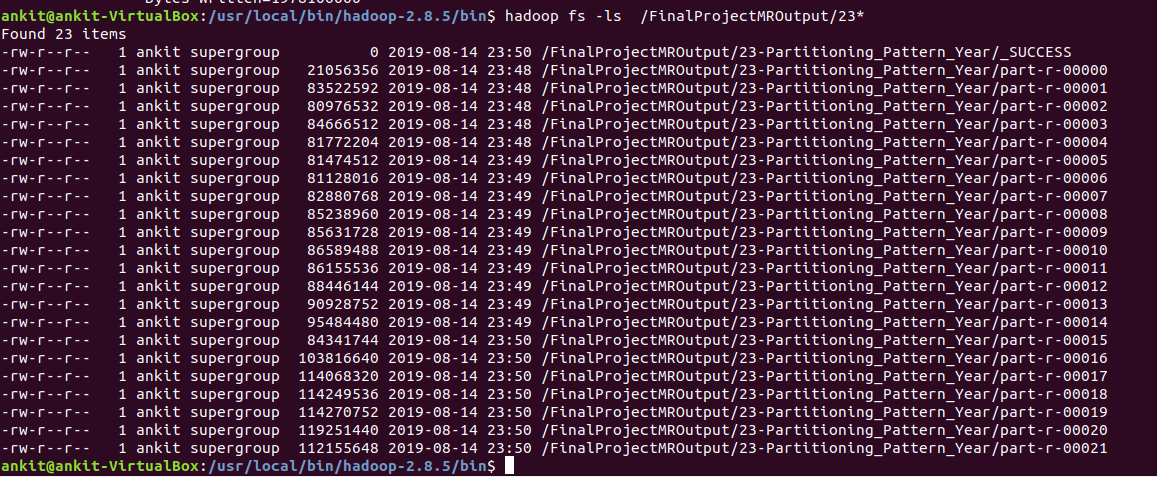


**20- Average flying distance per carrier**





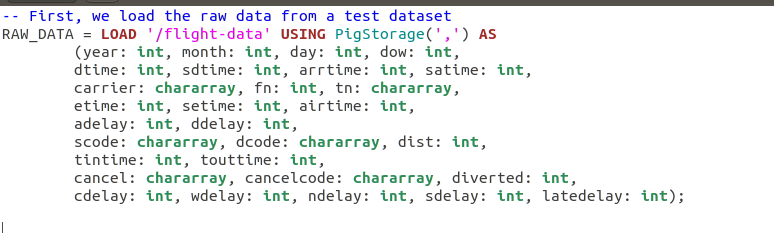
**21- Using partitioning pattern on the basis of year**



**2. Analysis of Flight Data using Apache PIG on Hadoop**

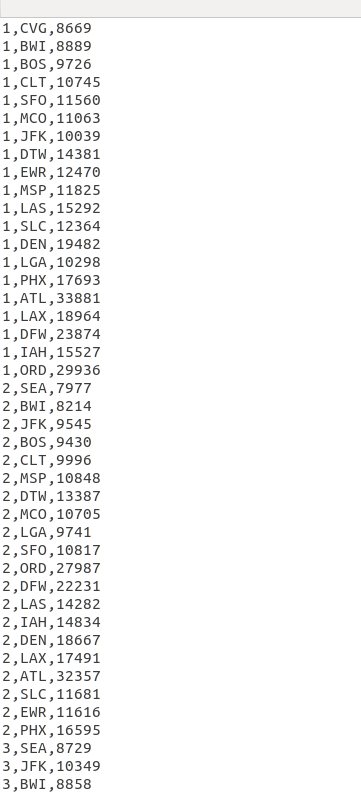
**Analysis 1: Top 20 cities by total volume of flights**

What are the busiest cities by total flight traffic? For each airport code I computed the number of inbound, outbound and all flights.

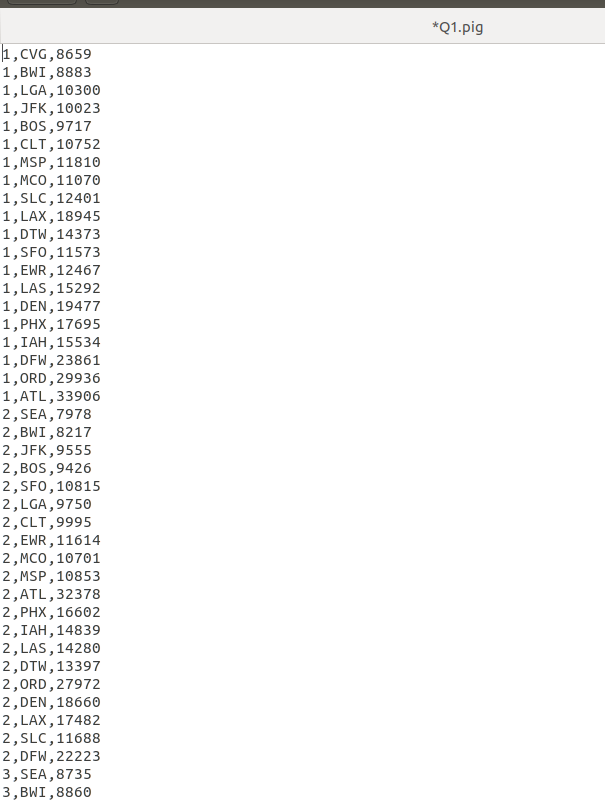




**OUTPUT: INBOUNT\_TOP**

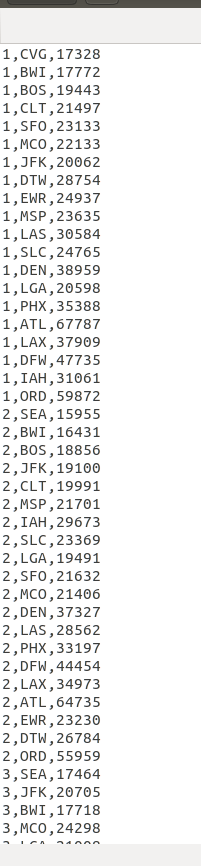


**OUTPUT: OUTBOUND TOP**



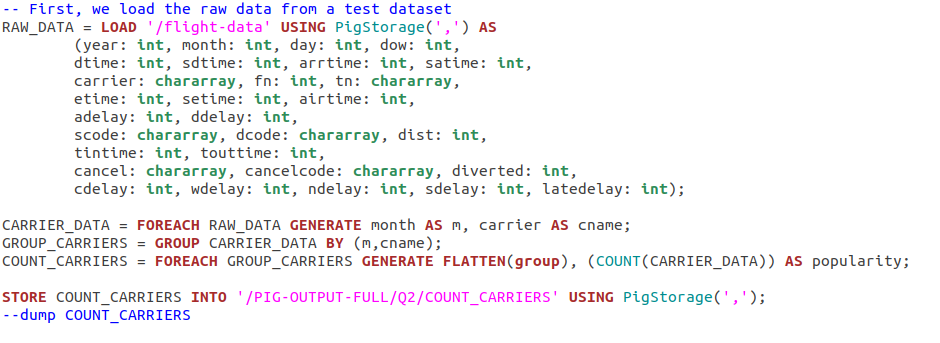


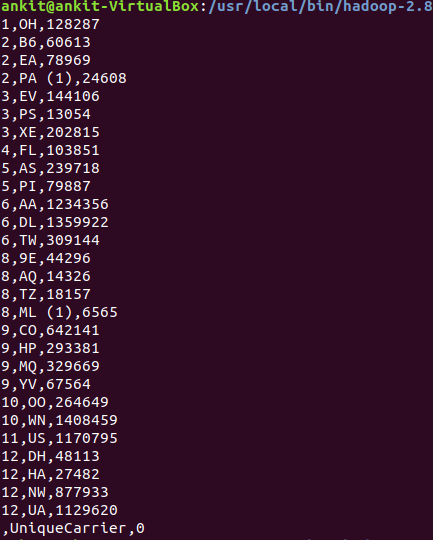
**Output Monthly Traffic Top**



**Analysis 2: Carrier Popularity**

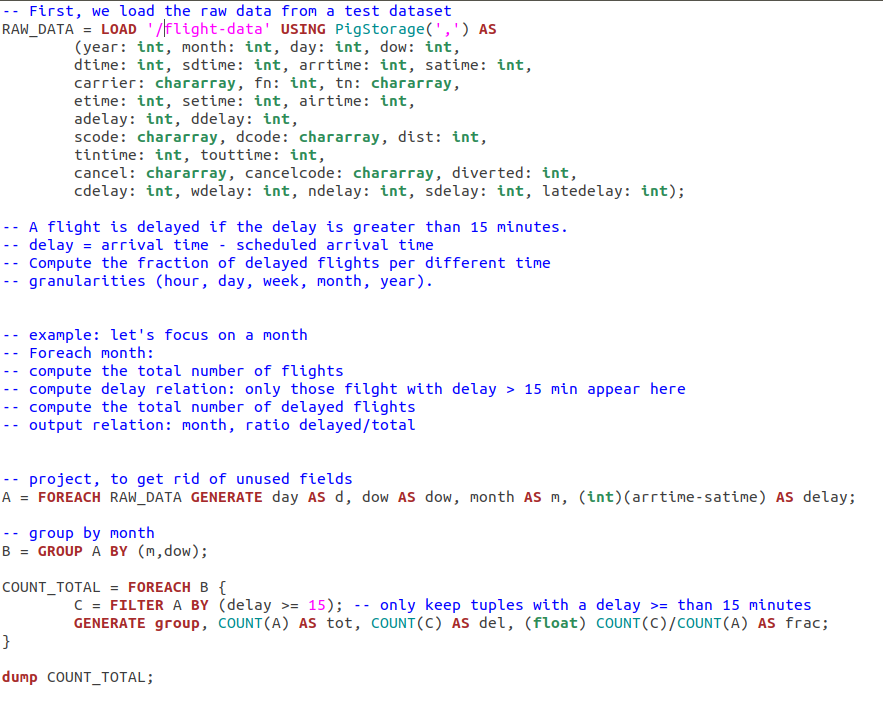
Computing the volume -- total flights -- over each year, by carrier. The carriers are ranked by their median volume (over the 10-year span).

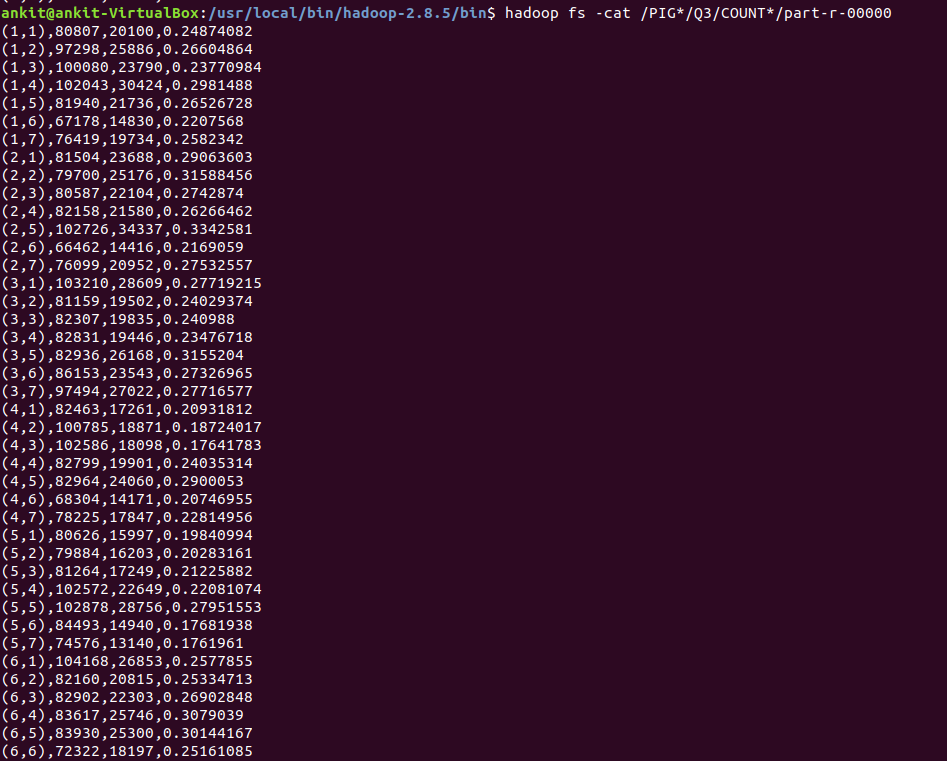




**Analysis 3: Proportion of Flights Delayed**

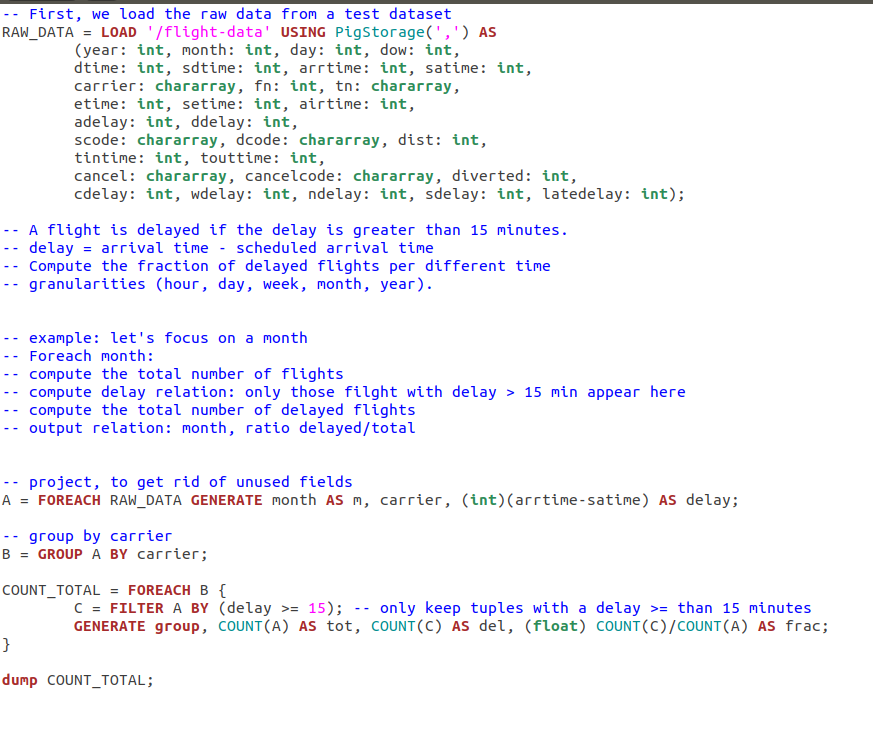
A flight is delayed if the delay is greater than 15 minutes. I am calculating the fraction of delayed flights per different time limits (hour, day, week, month, year).





**Analysis 4: Carrier Delays**

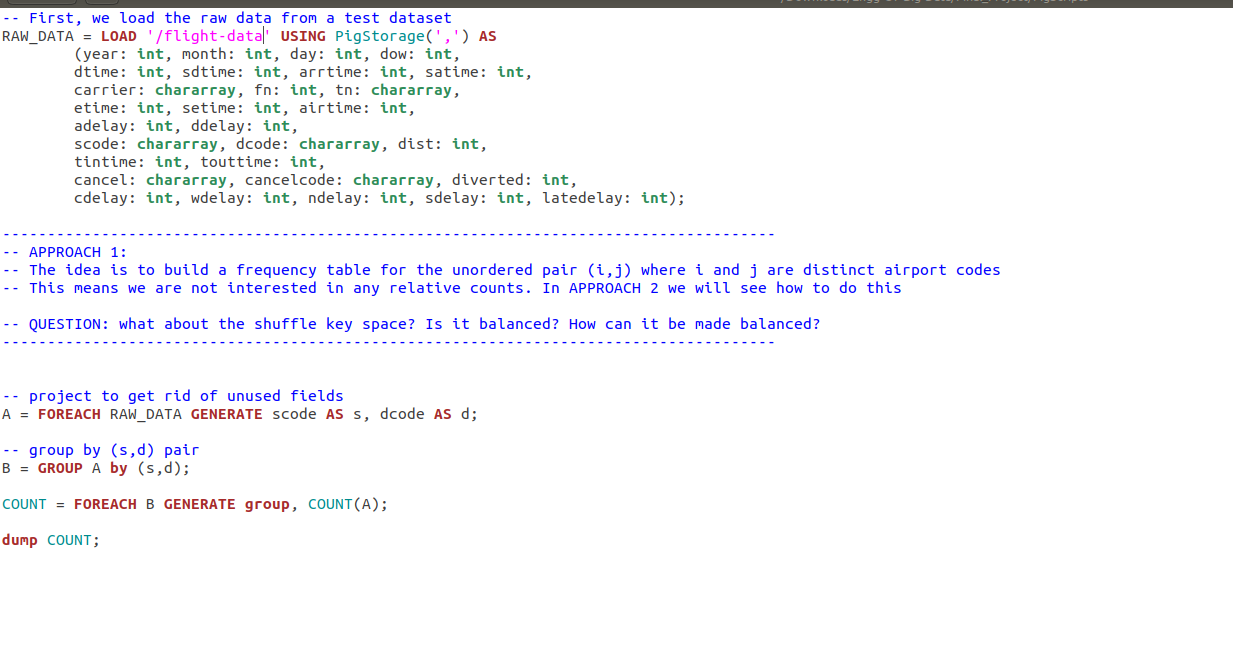
Calculating the proportion of delayed flights by carrier, ranked by carrier, at different time (hour, day, week, month year). Again, a flight is delayed if the delay is greater than 15 minutes.

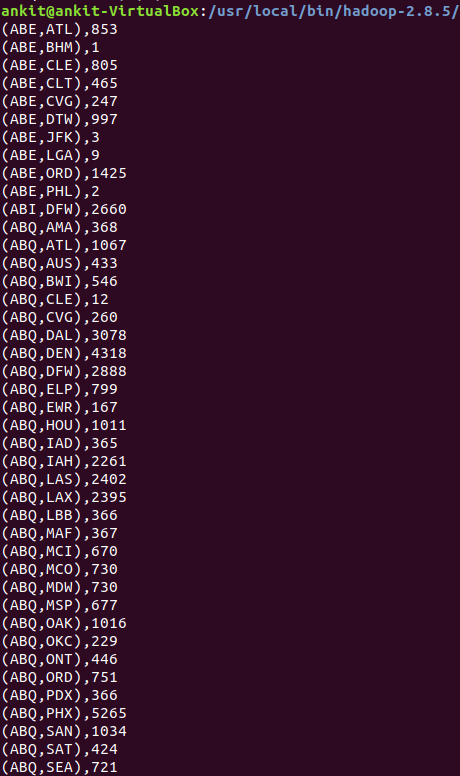




**Analysis 5: Routes that were most busy**

The approach is to create a frequency table for the unordered pair (m,n) where m and n are distinct airport codes which will help in finding the routes that are more busy.



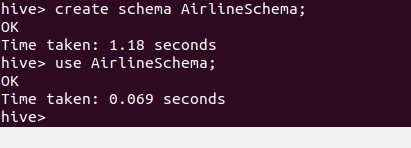


**3. Analysis of Flight Data using Apache HIVE on Hadoop**

**Creating schema for flight data**

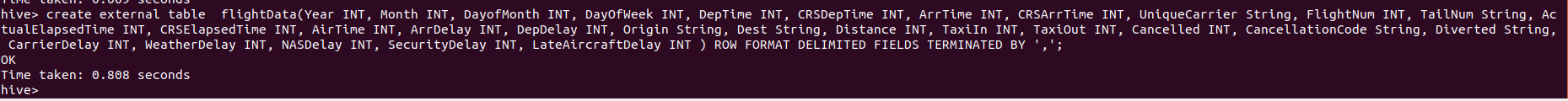
create schema AirlineSchema;

use AirlineSchema;



**Creating table to store flight data**

create external table flightData(Year INT, Month INT, DayofMonth INT, DayOfWeek INT, DepTime INT, CRSDepTime INT, ArrTime INT, CRSArrTime INT, UniqueCarrier String, FlightNum INT, TailNum String, ActualElapsedTime INT, CRSElapsedTime INT, AirTime INT, ArrDelay INT, DepDelay INT, Origin String, Dest String, Distance INT, TaxiIn INT, TaxiOut INT, Cancelled INT, CancellationCode String, Diverted String, CarrierDelay INT, WeatherDelay INT, NASDelay INT, SecurityDelay INT, LateAircraftDelay INT ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';



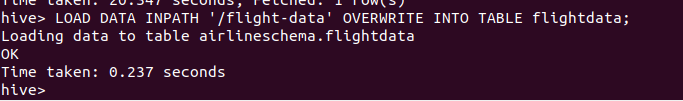
**Set some hive properties**

SET hive.exec.dynamic.partition = true;

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

**Load flight data from HDFS into table**

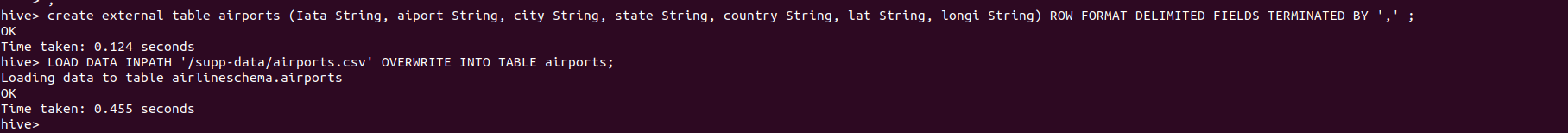
LOAD DATA INPATH '/flight-data' OVERWRITE INTO TABLE flightData;



**Create table and load airports data from HDFS**

create external table airports (Iata String, aiport String, city String, state String, country String, lat String, longi String) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ;

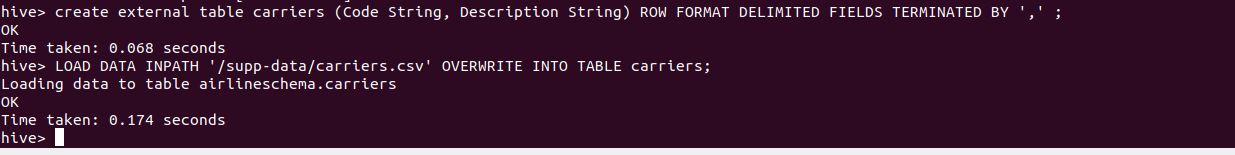
LOAD DATA INPATH '/supp-data/airports.csv' OVERWRITE INTO TABLE airports;



**Create table and load carrier’s data from HDFS**

create external table carriers (Code String, Description String) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ;

LOAD DATA INPATH '/supp-data/carriers.csv' OVERWRITE INTO TABLE carriers;



Now, All data is loaded. So, we can proceed to analysis.

**1: FLIGHTS THAT TRAVELLED LESS THAN OR MORE THAN 500 AIRTIME**

INSERT OVERWRITE DIRECTORY '/HiveMROutput/1.1' select count(\*) from flightData where AirTime > 500;

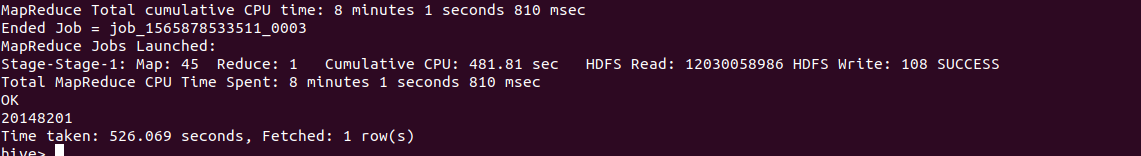
INSERT OVERWRITE DIRECTORY '/HiveMROutput/1.2' select count(\*) from flightData where AirTime >= 500;

**2. COUNT OF ALL THE FLIGHTS THAT WERE ON TIME WHILE ARRIVING AND DEPARTURE**

INSERT OVERWRITE DIRECTORY '/HiveMROutput/2' select Year,Month,DayofMonth,Origin,Dest,AirTime,Distance,TaxiIn,TaxiOut from flightData where DepTime<=CRSDepTime and ArrTime<=CRSArrTime;

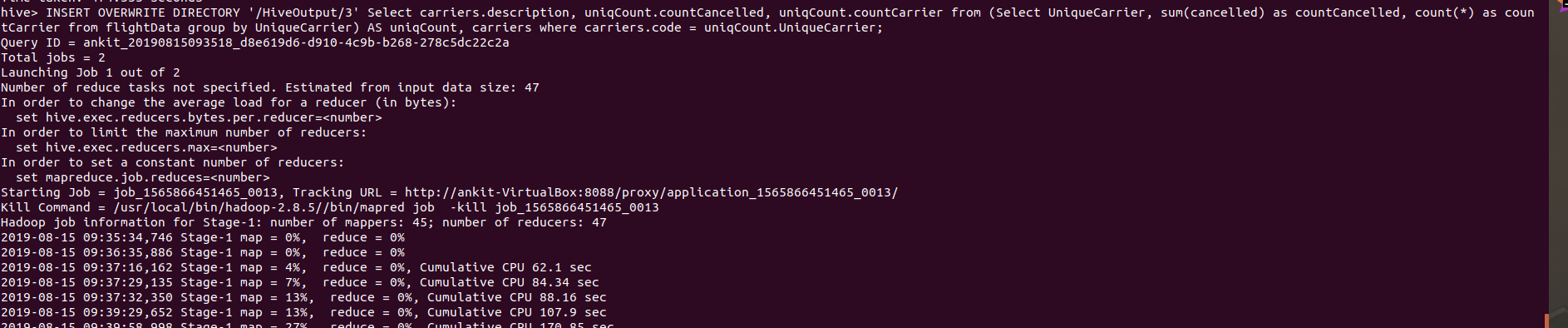
**3: COUNT OF ALL THE FLIGHTS THAT TOOK MORE THAN 30 MINS TO DEP AND ARRIVAL DELAY**

select count(\*) from flightData where ArrDelay + DepDelay >30;



**4: COUNT OF FLIGHTS FOR EACH CARRIER**

INSERT OVERWRITE DIRECTORY '/HiveMROutput/3' Select carriers.description, uniqCount.countCancelled, uniqCount.countCarrier from (Select UniqueCarrier, sum(cancelled) as countCancelled, count(\*) as countCarrier from flightData group by UniqueCarrier) AS uniqCount, carriers where carriers.code = uniqCount.UniqueCarrier;



**4. References**

1. <https://learning.oreilly.com/library/view/mapreduce-design-patterns/9781449341954/>
2. <https://gitlab.eurecom.fr/yonghui.feng/clouds-lab>
3. <https://learning.oreilly.com/library/view/data-algorithms/9781491906170/ch01.html>
4. <http://cs229.stanford.edu/proj2013/MathurNagaoNg-PredictingFlightOnTimePerformance.pdf>

**5. APPENDIX**

The code of this project can be found at GitHub repository for this project at

<https://github.com/ankit08015/Engg-Of-Big-Data/tree/master/Final_Project>

**1. Airport Count**

**package** hadoop.project.airport\_count;  
  
**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.lib.input.FileInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** AirportMain {  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"airportcount"**);  
 job.setJarByClass(AirportMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(IntWritable.**class**);  
  
  
  
  
 job.setMapperClass(AirportMapper.**class**);  
 job.setCombinerClass(AirportReducer.**class**);  
 job.setReducerClass(AirportReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(IntWritable.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.airport\_count;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.LongWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** AirportMapper **extends** Mapper<LongWritable, Text, Text, IntWritable> {  
  
 *// hadoop datatype* Text **word** = **new** Text();  
 IntWritable **one** = **new** IntWritable(1);  
  
 @Override  
 **protected void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  
  
 String line = value.toString();  
 String[] tokens= line.split(**","**);  
 **if**(tokens[0].equals(**"Year"**))**return**;  
  
 String src = tokens[16];  
 **word**.set(src);  
 context.write(**word**,**one**);  
 }  
  
  
}

**package** hadoop.project.airport\_count;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** AirportReducer **extends** Reducer<Text, IntWritable, Text, IntWritable> {  
  
 *// just like in mongoDB values is iterable* @Override  
 **protected void** reduce(Text key, Iterable<IntWritable> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** sum=0;  
 **for**(IntWritable v: values){  
 sum += v.get();  
 *// can we use this-- Integer.parseInt(v.toString());* }  
  
 context.write(key, **new** IntWritable(sum));  
  
 *//super.reduce(key, values, context); //To change body of generated methods, choose Tools | Templates.* }  
  
}

**2- Average Distance Carrier**

**package** hadoop.project.avg\_distance\_carrier;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** AverageCombiner **extends** Reducer<Text, AverageCountTuple, Text, AverageCountTuple> {  
  
 **private** AverageCountTuple **tuple** = **new** AverageCountTuple();  
  
 @Override  
 **protected void** reduce(Text key, Iterable<AverageCountTuple> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** totalFlight=0;  
 **int** totalDist=0;  
 **int** totalAirTime =0;  
  
  
  
 **for**(AverageCountTuple dt: values){  
 totalFlight += dt.getFlightCount();  
 totalDist += dt.getDistCount();  
 totalAirTime += dt.getAirTime();  
  
 }  
  
 **tuple**.setAirTime(totalAirTime);  
 **tuple**.setDistCount(totalDist);  
 **tuple**.setFlightCount(totalFlight);  
  
 context.write(key,**tuple**);  
 }  
}

**package** hadoop.project.avg\_distance\_carrier;  
  
**import** org.apache.hadoop.io.Writable;  
  
**import** java.io.DataInput;  
**import** java.io.DataOutput;  
**import** java.io.IOException;  
  
**public class** AverageCountTuple **implements** Writable {  
  
 **private int flightCount**=0;  
 **private int distCount**=0;  
 **private int airTime**=0;  
 **private double averageDist** =0.0;  
 **private double averageAirTime**=0.0;  
  
 **public int** getAirTime() {  
 **return airTime**;  
 }  
  
 **public void** setAirTime(**int** airTime) {  
 **this**.**airTime** = airTime;  
 }  
  
 **public int** getFlightCount() {  
 **return flightCount**;  
 }  
  
 **public void** setFlightCount(**int** flightCount) {  
 **this**.**flightCount** = flightCount;  
 }  
  
 **public int** getDistCount() {  
 **return distCount**;  
 }  
  
 **public void** setDistCount(**int** distCount) {  
 **this**.**distCount** = distCount;  
 }  
  
 **public double** getAverageDist() {  
 **return averageDist**;  
 }  
  
 **public void** setAverageDist(**double** averageDist) {  
 **this**.**averageDist** = averageDist;  
 }  
  
 **public double** getAverageAirTime() {  
 **return averageAirTime**;  
 }  
  
 **public void** setAverageAirTime(**double** averageAirTime) {  
 **this**.**averageAirTime** = averageAirTime;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "AverageCountTuple{"** +  
 **"Total Flights="** + **flightCount** +  
 **", Total Distance="** + **distCount** +  
 **", Total Air Time="** + **airTime** +  
 **", Average Distance="** + String.*format*(**"%.2f"**, **averageDist**) +  
 **", Average Air Time="** + String.*format*(**"%.2f"**, **averageAirTime**)+  
 **'}'**;  
 }  
  
 @Override  
 **public void** write(DataOutput dataOutput) **throws** IOException {  
 dataOutput.writeInt(**flightCount**);  
 dataOutput.writeInt(**distCount**);  
 dataOutput.writeInt(**airTime**);  
 dataOutput.writeDouble(**averageDist**);  
 dataOutput.writeDouble(**averageAirTime**);  
  
 }  
  
 @Override  
 **public void** readFields(DataInput dataInput) **throws** IOException {  
  
 **flightCount** = dataInput.readInt();  
 **distCount** = dataInput.readInt();  
 **airTime** = dataInput.readInt();  
 **averageDist** = dataInput.readDouble();  
 **averageAirTime** = dataInput.readDouble();  
  
 }  
}

**package** hadoop.project.avg\_distance\_carrier;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** AverageMain {  
  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(AverageMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(AverageCountTuple.**class**);  
  
  
 job.setMapperClass(AverageMapper.**class**);  
 job.setCombinerClass(AverageCombiner.**class**);  
 job.setReducerClass(AverageReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(AverageCountTuple.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.avg\_distance\_carrier;  
  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
  
**public class** AverageMapper **extends** Mapper<Object, Text, Text, AverageCountTuple> {  
  
 **private** AverageCountTuple **tuple** = **new** AverageCountTuple();  
  
 @Override  
 **protected void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
 String [] tokens = value.toString().split(**","**);  
  
 **if**(tokens[0].equals(**"Year"**))**return**;  
  
 String carrier = tokens[8];  
 **int** dist=0;  
 **int** flightTime =0;  
  
 **try** {  
 dist = Integer.*parseInt*(tokens[18]);  
  
 flightTime = Integer.*parseInt*(tokens[6]);  
 } **catch** (Exception e){  
 **return**;  
 }  
 **tuple**.setFlightCount(1);  
 **tuple**.setDistCount(dist);  
 **tuple**.setAirTime(flightTime);  
  
  
 context.write(**new** Text(carrier),**tuple**);  
 }  
}

**package** hadoop.project.avg\_distance\_carrier;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** AverageReducer **extends** Reducer<Text, AverageCountTuple, Text, AverageCountTuple> {  
  
**private** AverageCountTuple **tuple** = **new** AverageCountTuple();  
  
@Override  
**protected void** reduce(Text key, Iterable<AverageCountTuple> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** totalFlight=0;  
 **int** totalDist=0;  
 **int** totalAirTime =0;  
  
  
  
 **for**(AverageCountTuple dt: values){  
 totalFlight += dt.getFlightCount();  
 totalDist += dt.getDistCount();  
 totalAirTime += dt.getAirTime();  
  
 }  
  
 **double** avgDist = (**double**)totalDist/totalFlight;  
 **double** avgAirTime = (**double**)totalAirTime/totalFlight;  
  
 **tuple**.setAirTime(totalAirTime);  
 **tuple**.setDistCount(totalDist);  
 **tuple**.setFlightCount(totalFlight);  
 **tuple**.setAverageDist(avgDist);  
 **tuple**.setAverageAirTime(avgAirTime);  
  
 context.write(key,**tuple**);  
 }  
 }

3 Cancel By Carriers Count

**package** hadoop.project.cancel\_by\_carriers;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** CancelCarrierMain {  
  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(CancelCarrierMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(CancelCountTuple.**class**);  
  
  
 job.setMapperClass(CancelCarrierMapper.**class**);  
 job.setCombinerClass(CancelCarrierReducer.**class**);  
 job.setReducerClass(CancelCarrierReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(CancelCountTuple.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.cancel\_by\_carriers;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
  
**public class** CancelCarrierMapper **extends** Mapper<Object, Text, Text, CancelCountTuple> {  
  
 **private** CancelCountTuple **tuple** = **new** CancelCountTuple();  
  
 @Override  
 **protected void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
 String [] tokens = value.toString().split(**","**);  
  
 **if**(tokens[0].equals(**"Year"**))**return**;  
  
 String carrier = tokens[8];  
  
  
 **try** {  
 String can = tokens[21];  
 **tuple**.setCancelFlightCount(Integer.*parseInt*(can));  
  
 }**catch** (Exception e){  
 **tuple**.setCancelFlightCount(0);  
 }  
  
  
 **tuple**.setFlightCount(1);  
  
 context.write(**new** Text(carrier),**tuple**);  
 }  
}

**package** hadoop.project.cancel\_by\_carriers;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** CancelCarrierReducer **extends** Reducer<Text, CancelCountTuple, Text, CancelCountTuple> {  
  
 **private** CancelCountTuple **res**= **new** CancelCountTuple();  
  
 @Override  
 **protected void** reduce(Text key, Iterable<CancelCountTuple> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** total=0;  
 **int** cancelTotal=0;  
  
 **for**(CancelCountTuple dt: values){  
 total += dt.getFlightCount();  
 cancelTotal +=dt.getCancelFlightCount();  
 }  
  
 **double** percent = ((**double**)cancelTotal/total)\*100;  
  
 **res**.setCancelFlightCount(cancelTotal);  
 **res**.setFlightCount(total);  
 **res**.setCancelPercent(percent);  
  
 context.write(key,**res**);  
 }  
}

**package** hadoop.project.cancel\_by\_carriers;  
  
**import** org.apache.hadoop.io.Writable;  
  
**import** java.io.DataInput;  
**import** java.io.DataOutput;  
**import** java.io.IOException;  
  
**public class** CancelCountTuple **implements** Writable {  
  
 **private int flightCount**=0;  
 **private int cancelFlightCount**=0;  
 **private double cancelPercent** =0.0;  
  
 **public int** getFlightCount() {  
 **return flightCount**;  
 }  
  
 **public void** setFlightCount(**int** flightCount) {  
 **this**.**flightCount** = flightCount;  
 }  
  
 **public int** getCancelFlightCount() {  
 **return cancelFlightCount**;  
 }  
  
 **public void** setCancelFlightCount(**int** cancelFlightCount) {  
 **this**.**cancelFlightCount** = cancelFlightCount;  
 }  
  
 **public double** getCancelPercent() {  
 **return cancelPercent**;  
 }  
  
 **public void** setCancelPercent(**double** cancelPercent) {  
 **this**.**cancelPercent** = cancelPercent;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "flightCount="** + **flightCount** +  
 **", delayedFlightCount="** + **cancelFlightCount** +  
 **", delayPercent="** + String.*format*(**"%.2f"**, **cancelPercent**);  
 }  
  
 @Override  
 **public void** write(DataOutput dataOutput) **throws** IOException {  
 dataOutput.writeInt(**flightCount**);  
 dataOutput.writeInt(**cancelFlightCount**);  
 dataOutput.writeDouble(**cancelPercent**);  
  
 }  
  
 @Override  
 **public void** readFields(DataInput dataInput) **throws** IOException {  
  
 **flightCount** = dataInput.readInt();  
 **cancelFlightCount** = dataInput.readInt();  
 **cancelPercent** = dataInput.readDouble();  
  
 }  
}

4- Cancelled By Year

**package** hadoop.project.cancelled\_by\_year;  
  
**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.lib.input.FileInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** CancMain {  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(CancMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(IntWritable.**class**);  
  
  
 job.setMapperClass(CancMapper.**class**);  
 job.setReducerClass(CancReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(IntWritable.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.cancelled\_by\_year;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.LongWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** CancMapper **extends** Mapper<LongWritable, Text, Text, IntWritable> {  
  
 *// hadoop datatype* Text **word** = **new** Text();  
 IntWritable **one** = **new** IntWritable(1);  
  
 @Override  
 **protected void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  
 String [] values = value.toString().split(**","**);  
 **if**(values[0].equals(**"Year"**))**return**;  
 **try** {  
 String can = values[21];  
 String year = values[0];  
  
 **if**(can.equals(**"1"**))  
 context.write(**new** Text(year), **one**);  
 } **catch**(Exception e){  
 **return**;  
 }  
  
 }  
}

**package** hadoop.project.cancelled\_by\_year;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** CancReducer **extends** Reducer<Text, IntWritable, Text, IntWritable> {  
  
 *// just like in mongoDB values is iterable* @Override  
 **protected void** reduce(Text key, Iterable<IntWritable> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** sum=0;  
 **for**(IntWritable v: values){  
 sum += v.get();  
 *// can we use this-- Integer.parseInt(v.toString());* }  
  
 context.write(key, **new** IntWritable(sum));  
  
 *//super.reduce(key, values, context); //To change body of generated methods, choose Tools | Templates.* }  
  
}

5- Delayed Flights By Careers

**package** hadoop.project.delay\_by\_carriers;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** DelayCarrierMain {  
  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(DelayCarrierMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(DelayCountTuple.**class**);  
  
  
 job.setMapperClass(DelayCarrierMapper.**class**);  
 job.setCombinerClass(DelayCarrierReducer.**class**);  
 job.setReducerClass(DelayCarrierReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(DelayCountTuple.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.delay\_by\_carriers;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
  
**public class** DelayCarrierMapper **extends** Mapper<Object, Text, Text, DelayCountTuple> {  
  
 **private** DelayCountTuple **tuple** = **new** DelayCountTuple();  
  
 @Override  
 **protected void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
 String [] tokens = value.toString().split(**","**);  
  
 **if**(tokens[0].equals(**"Year"**))**return**;  
  
 String carrier = tokens[8];  
  
  
 **try** {  
 **int** delay = Integer.*parseInt*(tokens[14]);  
  
  
 **if** (delay > 15) {  
 **tuple**.setDelayedFlightCount(1);  
 } **else** {  
 **tuple**.setDelayedFlightCount(0);  
 }  
 }**catch** (Exception e){  
 **tuple**.setDelayedFlightCount(0);  
 }  
  
 **tuple**.setFlightCount(1);  
  
 context.write(**new** Text(carrier),**tuple**);  
 }  
}

**package** hadoop.project.delay\_by\_carriers;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** DelayCarrierReducer **extends** Reducer<Text, DelayCountTuple, Text, DelayCountTuple> {  
  
 **private** DelayCountTuple **res**= **new** DelayCountTuple();  
  
 @Override  
 **protected void** reduce(Text key, Iterable<DelayCountTuple> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** total=0;  
 **int** delayedTotal=0;  
  
 **for**(DelayCountTuple dt: values){  
 total += dt.getFlightCount();  
 delayedTotal +=dt.getDelayedFlightCount();  
 }  
  
 **double** percent = ((**double**)delayedTotal/total)\*100;  
  
 **res**.setDelayedFlightCount(delayedTotal);  
 **res**.setFlightCount(total);  
 **res**.setDelayPercent(percent);  
  
 context.write(key,**res**);  
 }  
}

**package** hadoop.project.delay\_by\_carriers;  
  
**import** org.apache.hadoop.io.Writable;  
  
**import** java.io.DataInput;  
**import** java.io.DataOutput;  
**import** java.io.IOException;  
  
**public class** DelayCountTuple **implements** Writable {  
  
 **private int flightCount**=0;  
 **private int delayedFlightCount**=0;  
 **private double delayPercent** =0.0;  
  
 **public int** getFlightCount() {  
 **return flightCount**;  
 }  
  
 **public void** setFlightCount(**int** flightCount) {  
 **this**.**flightCount** = flightCount;  
 }  
  
 **public int** getDelayedFlightCount() {  
 **return delayedFlightCount**;  
 }  
  
 **public void** setDelayedFlightCount(**int** delayedFlightCount) {  
 **this**.**delayedFlightCount** = delayedFlightCount;  
 }  
  
 **public double** getDelayPercent() {  
 **return delayPercent**;  
 }  
  
 **public void** setDelayPercent(**double** delayPercent) {  
 **this**.**delayPercent** = delayPercent;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "flightCount="** + **flightCount** +  
 **", delayedFlightCount="** + **delayedFlightCount** +  
 **", delayPercent="** + String.*format*(**"%.2f"**, **delayPercent**);  
 }  
  
 @Override  
 **public void** write(DataOutput dataOutput) **throws** IOException {  
 dataOutput.writeInt(**flightCount**);  
 dataOutput.writeInt(**delayedFlightCount**);  
 dataOutput.writeDouble(**delayPercent**);  
  
 }  
  
 @Override  
 **public void** readFields(DataInput dataInput) **throws** IOException {  
  
 **flightCount** = dataInput.readInt();  
 **delayedFlightCount** = dataInput.readInt();  
 **delayPercent** = dataInput.readDouble();  
  
 }  
}

6- Delay Groups

**package** hadoop.project.delay\_groups;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**import** org.apache.hadoop.io.DoubleWritable;  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** GroupMain {  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(GroupMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(IntWritable.**class**);  
  
  
 job.setMapperClass(GroupMapper.**class**);  
 job.setReducerClass(GroupReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(Text.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.delay\_groups;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** GroupMapper **extends** Mapper<Object, Text, Text, IntWritable> {  
  
  
 @Override  
 **protected void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
 String [] tokens = value.toString().split(**","**);  
  
 **if**(tokens[0].equals(**"Year"**))**return**;  
  
 **int** delay =0;  
  
 **try** {  
 delay = Integer.*parseInt*(tokens[15]);  
  
 }**catch** (Exception e){  
 delay=0;  
 }  
  
 String grpKey=**""**;  
  
 **if**(delay<15)  
 grpKey=**"Less than 15 Minutes"**;  
 **else if**(delay>=15 && delay <=30)  
 grpKey=**"Between 15 abd 30 minutes"**;  
 **else if**(delay>30 && delay<60)  
 grpKey=**"Between 30 minutes and 1 hour"**;  
 **else if**(delay>=60 && delay<=120)  
 grpKey=**"Between 1 hour and 2 hour"**;  
 **else** grpKey=**"More than 2 hours"**;  
  
 context.write(**new** Text(grpKey),**new** IntWritable(1));  
 }  
}

**package** hadoop.project.delay\_groups;  
  
**import** org.apache.hadoop.io.DoubleWritable;  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** GroupReducer **extends** Reducer<Text, IntWritable, Text, Text> {  
  
 *// just like in mongoDB values is iterable* @Override  
 **protected void** reduce(Text key, Iterable<IntWritable> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** sum=0;  
 **for**(IntWritable v: values){  
 sum += v.get();  
 *// can we use this-- Integer.parseInt(v.toString());* }  
  
 **double** total = 123534970.0;  
 *//double total = 1311827;* **double** percent= (sum/total)\*100;  
  
 String res = String.*format*(**"%.2f"**, percent);  
  
 context.write(key, **new** Text(res));  
  
 *//super.reduce(key, values, context); //To change body of generated methods, choose Tools | Templates.* }  
  
}

7- Delay By Month

**package** hadoop.project.delay\_month;  
  
**import** org.apache.hadoop.io.Writable;  
  
**import** java.io.DataInput;  
**import** java.io.DataOutput;  
**import** java.io.IOException;  
  
**public class** DelayCountTuple **implements** Writable {  
  
 **private int flightCount**=0;  
 **private int delayedFlightCount**=0;  
 **private double delayPercent** =0.0;  
  
 **public int** getFlightCount() {  
 **return flightCount**;  
 }  
  
 **public void** setFlightCount(**int** flightCount) {  
 **this**.**flightCount** = flightCount;  
 }  
  
 **public int** getDelayedFlightCount() {  
 **return delayedFlightCount**;  
 }  
  
 **public void** setDelayedFlightCount(**int** delayedFlightCount) {  
 **this**.**delayedFlightCount** = delayedFlightCount;  
 }  
  
 **public double** getDelayPercent() {  
 **return delayPercent**;  
 }  
  
 **public void** setDelayPercent(**double** delayPercent) {  
 **this**.**delayPercent** = delayPercent;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "flightCount="** + **flightCount** +  
 **", delayedFlightCount="** + **delayedFlightCount** +  
 **", delayPercent="** + String.*format*(**"%.2f"**, **delayPercent**);  
 }  
  
 @Override  
 **public void** write(DataOutput dataOutput) **throws** IOException {  
 dataOutput.writeInt(**flightCount**);  
 dataOutput.writeInt(**delayedFlightCount**);  
 dataOutput.writeDouble(**delayPercent**);  
  
 }  
  
 @Override  
 **public void** readFields(DataInput dataInput) **throws** IOException {  
  
 **flightCount** = dataInput.readInt();  
 **delayedFlightCount** = dataInput.readInt();  
 **delayPercent** = dataInput.readDouble();  
  
 }  
}

**package** hadoop.project.delay\_month;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** DelayMonthMain {  
  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(DelayMonthMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(DelayCountTuple.**class**);  
  
  
 job.setMapperClass(DelayMonthMapper.**class**);  
 job.setCombinerClass(DelayMonthReducer.**class**);  
 job.setReducerClass(DelayMonthReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(DelayCountTuple.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.delay\_month;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
  
**public class** DelayMonthMapper **extends** Mapper<Object, Text, Text, DelayCountTuple> {  
  
 **private** String [] **days** ={**""**,**"1-January"**,**"2-February"**,**"3-March"**,**"4-April"**,**"5-May"**,**"6June"**,**"7-July"**,**"8-August"**,**"9-September"**,**"10-October"**,**"11-November"**,**"12-December"**};  
 **private** DelayCountTuple **tuple** = **new** DelayCountTuple();  
  
 @Override  
 **protected void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
 String [] tokens = value.toString().split(**","**);  
  
 **if**(tokens[0].equals(**"Year"**))**return**;  
  
 String month = **days**[Integer.*parseInt*(tokens[1])];  
  
  
 **try** {  
 **int** delay = Integer.*parseInt*(tokens[14]);  
  
  
 **if** (delay > 15) {  
 **tuple**.setDelayedFlightCount(1);  
 } **else** {  
 **tuple**.setDelayedFlightCount(0);  
 }  
 }**catch** (Exception e){  
 **tuple**.setDelayedFlightCount(0);  
 }  
  
 **tuple**.setFlightCount(1);  
  
 context.write(**new** Text(month),**tuple**);  
 }  
}

**package** hadoop.project.delay\_month;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** DelayMonthReducer **extends** Reducer<Text, DelayCountTuple, Text, DelayCountTuple> {  
  
 **private** DelayCountTuple **res**= **new** DelayCountTuple();  
  
 @Override  
 **protected void** reduce(Text key, Iterable<DelayCountTuple> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** total=0;  
 **int** delayedTotal=0;  
  
 **for**(DelayCountTuple dt: values){  
 total += dt.getFlightCount();  
 delayedTotal +=dt.getDelayedFlightCount();  
 }  
  
 **double** percent = ((**double**)delayedTotal/total)\*100;  
  
 **res**.setDelayedFlightCount(delayedTotal);  
 **res**.setFlightCount(total);  
 **res**.setDelayPercent(percent);  
  
 context.write(key,**res**);  
 }  
}

8- Delay per year

**package** hadoop.project.delay\_per\_year;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** DelayedMain {  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(DelayedMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(IntWritable.**class**);  
  
  
 job.setMapperClass(DelayedMapper.**class**);  
 job.setReducerClass(DelayedReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(IntWritable.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.delay\_per\_year;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.LongWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** DelayedMapper **extends** Mapper<LongWritable, Text, Text, IntWritable> {  
  
 *// hadoop datatype* Text **word** = **new** Text();  
 IntWritable **one** = **new** IntWritable(1);  
  
 @Override  
 **protected void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  
 String [] values = value.toString().split(**","**);  
 **if**(values[0].equals(**"Year"**))**return**;  
 **try** {  
 **int** delay = Integer.*parseInt*(values[14]);  
 String year = values[0];  
  
 **if**(delay>=15)  
 context.write(**new** Text(year), **one**);  
 } **catch**(Exception e){  
 **return**;  
 }  
  
 }  
}

**package** hadoop.project.delay\_per\_year;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** DelayedReducer **extends** Reducer<Text, IntWritable, Text, IntWritable> {  
  
 *// just like in mongoDB values is iterable* @Override  
 **protected void** reduce(Text key, Iterable<IntWritable> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** sum=0;  
 **for**(IntWritable v: values){  
 sum += v.get();  
 *// can we use this-- Integer.parseInt(v.toString());* }  
  
 context.write(key, **new** IntWritable(sum));  
  
 *//super.reduce(key, values, context); //To change body of generated methods, choose Tools | Templates.* }  
  
}

9- Delay ratio by year

**package** hadoop.project.delay\_ratio\_year;  
  
**import** org.apache.hadoop.io.Writable;  
  
**import** java.io.DataInput;  
**import** java.io.DataOutput;  
**import** java.io.IOException;  
  
**public class** DelayCountTuple **implements** Writable {  
  
 **private int flightCount**=0;  
 **private int delayedFlightCount**=0;  
 **private double delayPercent** =0.0;  
  
 **public int** getFlightCount() {  
 **return flightCount**;  
 }  
  
 **public void** setFlightCount(**int** flightCount) {  
 **this**.**flightCount** = flightCount;  
 }  
  
 **public int** getDelayedFlightCount() {  
 **return delayedFlightCount**;  
 }  
  
 **public void** setDelayedFlightCount(**int** delayedFlightCount) {  
 **this**.**delayedFlightCount** = delayedFlightCount;  
 }  
  
 **public double** getDelayPercent() {  
 **return delayPercent**;  
 }  
  
 **public void** setDelayPercent(**double** delayPercent) {  
 **this**.**delayPercent** = delayPercent;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "flightCount="** + **flightCount** +  
 **", delayedFlightCount="** + **delayedFlightCount** +  
 **", delayPercent="** + String.*format*(**"%.2f"**, **delayPercent**);  
 }  
  
 @Override  
 **public void** write(DataOutput dataOutput) **throws** IOException {  
 dataOutput.writeInt(**flightCount**);  
 dataOutput.writeInt(**delayedFlightCount**);  
 dataOutput.writeDouble(**delayPercent**);  
  
 }  
  
 @Override  
 **public void** readFields(DataInput dataInput) **throws** IOException {  
  
 **flightCount** = dataInput.readInt();  
 **delayedFlightCount** = dataInput.readInt();  
 **delayPercent** = dataInput.readDouble();  
  
 }  
}

**package** hadoop.project.delay\_ratio\_year;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** DelayYearMain {  
  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(DelayYearMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(DelayCountTuple.**class**);  
  
  
 job.setMapperClass(DelayYearMapper.**class**);  
 job.setCombinerClass(DelayYearReducer.**class**);  
 job.setReducerClass(DelayYearReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(DelayCountTuple.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.delay\_ratio\_year;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
  
**public class** DelayYearMapper **extends** Mapper<Object, Text, Text, DelayCountTuple> {  
  
 **private** DelayCountTuple **tuple** = **new** DelayCountTuple();  
  
 @Override  
 **protected void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
 String [] tokens = value.toString().split(**","**);  
  
 **if**(tokens[0].equals(**"Year"**))**return**;  
  
 String year = tokens[0];  
  
  
 **try** {  
 **int** delay = Integer.*parseInt*(tokens[14]);  
  
  
 **if** (delay > 15) {  
 **tuple**.setDelayedFlightCount(1);  
 } **else** {  
 **tuple**.setDelayedFlightCount(0);  
 }  
 }**catch** (Exception e){  
 **tuple**.setDelayedFlightCount(0);  
 }  
  
 **tuple**.setFlightCount(1);  
  
 context.write(**new** Text(year),**tuple**);  
 }  
}

**package** hadoop.project.delay\_ratio\_year;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** DelayYearReducer **extends** Reducer<Text, DelayCountTuple, Text, DelayCountTuple> {  
  
 **private** DelayCountTuple **res**= **new** DelayCountTuple();  
  
 @Override  
 **protected void** reduce(Text key, Iterable<DelayCountTuple> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** total=0;  
 **int** delayedTotal=0;  
  
 **for**(DelayCountTuple dt: values){  
 total += dt.getFlightCount();  
 delayedTotal +=dt.getDelayedFlightCount();  
 }  
  
 **double** percent = ((**double**)delayedTotal/total)\*100;  
  
 **res**.setDelayedFlightCount(delayedTotal);  
 **res**.setFlightCount(total);  
 **res**.setDelayPercent(percent);  
  
 context.write(key,**res**);  
 }  
}

10- All delayed count

**package** hadoop.project.delayed\_count;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** DelayedMain {  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(DelayedMapper.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(NullWritable.**class**);  
 job.setMapOutputValueClass(IntWritable.**class**);  
  
  
 job.setMapperClass(DelayedMapper.**class**);  
 job.setReducerClass(DelayedReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(IntWritable.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.delayed\_count;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.LongWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** DelayedMapper **extends** Mapper<LongWritable, Text, NullWritable, IntWritable> {  
  
 *// hadoop datatype* Text **word** = **new** Text();  
 IntWritable **one** = **new** IntWritable(1);  
  
 @Override  
 **protected void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  
 String [] values = value.toString().split(**","**);  
 **if**(values[0].equals(**"Year"**))**return**;  
 **try** {  
 **int** delay = Integer.*parseInt*(values[14]);  
  
 **if**(delay>=15)  
 context.write(NullWritable.*get*(), **one**);  
 } **catch**(Exception e){  
 **return**;  
 }  
  
 }  
}

**package** hadoop.project.delayed\_count;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** DelayedReducer **extends** Reducer<NullWritable, IntWritable, NullWritable, IntWritable> {  
  
 *// just like in mongoDB values is iterable* @Override  
 **protected void** reduce(NullWritable key, Iterable<IntWritable> values, Context context) **throws** IOException, InterruptedException {

**int** sum=0;  
 **for**(IntWritable v: values){  
 sum += v.get();  
 *// can we use this-- Integer.parseInt(v.toString());* }  
  
 context.write(key, **new** IntWritable(sum));  
  
 *//super.reduce(key, values, context); //To change body of generated methods, choose Tools | Templates.* }  
  
}

**11-Delay by days**

**package** hadoop.project.delays\_by\_days;  
  
**import** org.apache.hadoop.io.Writable;  
  
**import** java.io.DataInput;  
**import** java.io.DataOutput;  
**import** java.io.IOException;  
  
**public class** DelayCountTuple **implements** Writable {  
  
 **private int flightCount**=0;  
 **private int delayedFlightCount**=0;  
 **private double delayPercent** =0.0;  
  
 **public int** getFlightCount() {  
 **return flightCount**;  
 }  
  
 **public void** setFlightCount(**int** flightCount) {  
 **this**.**flightCount** = flightCount;  
 }  
  
 **public int** getDelayedFlightCount() {  
 **return delayedFlightCount**;  
 }  
  
 **public void** setDelayedFlightCount(**int** delayedFlightCount) {  
 **this**.**delayedFlightCount** = delayedFlightCount;  
 }  
  
 **public double** getDelayPercent() {  
 **return delayPercent**;  
 }  
  
 **public void** setDelayPercent(**double** delayPercent) {  
 **this**.**delayPercent** = delayPercent;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "flightCount="** + **flightCount** +  
 **", delayedFlightCount="** + **delayedFlightCount** +  
 **", delayPercent="** + String.*format*(**"%.2f"**, **delayPercent**);  
 }  
  
 @Override  
 **public void** write(DataOutput dataOutput) **throws** IOException {  
 dataOutput.writeInt(**flightCount**);  
 dataOutput.writeInt(**delayedFlightCount**);  
 dataOutput.writeDouble(**delayPercent**);  
  
 }  
  
 @Override  
 **public void** readFields(DataInput dataInput) **throws** IOException {  
  
 **flightCount** = dataInput.readInt();  
 **delayedFlightCount** = dataInput.readInt();  
 **delayPercent** = dataInput.readDouble();  
  
 }  
}

**package** hadoop.project.delays\_by\_days;  
  
**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.lib.input.FileInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** DelayDayMain {  
  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(DelayDayMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(DelayCountTuple.**class**);  
  
  
 job.setMapperClass(DelayDayMapper.**class**);  
 job.setCombinerClass(DelayDayReducer.**class**);  
 job.setReducerClass(DelayDayReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(DelayCountTuple.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.delays\_by\_days;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
  
**public class** DelayDayMapper **extends** Mapper<Object, Text, Text, DelayCountTuple> {  
  
 **private** String [] **days** ={**""**,**"Monday"**,**"Tuesday"**,**"Wednesday"**,**"Thursday"**,**"Friday"**,**"Saturday"**,**"Sunday"**};  
 **private** DelayCountTuple **tuple** = **new** DelayCountTuple();  
  
 @Override  
 **protected void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
 String [] tokens = value.toString().split(**","**);  
  
 **if**(tokens[0].equals(**"Year"**))**return**;  
  
 String day = **days**[Integer.*parseInt*(tokens[3])];  
  
  
 **try** {  
 **int** delay = Integer.*parseInt*(tokens[14]);  
  
  
 **if** (delay > 15) {  
 **tuple**.setDelayedFlightCount(1);  
 } **else** {  
 **tuple**.setDelayedFlightCount(0);  
 }  
 }**catch** (Exception e){  
 **tuple**.setDelayedFlightCount(0);  
 }  
  
 **tuple**.setFlightCount(1);  
  
 context.write(**new** Text(day),**tuple**);  
 }  
}

**package** hadoop.project.delays\_by\_days;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** DelayDayReducer **extends** Reducer<Text, DelayCountTuple, Text, DelayCountTuple> {  
  
 **private** DelayCountTuple **res**= **new** DelayCountTuple();  
  
 @Override  
 **protected void** reduce(Text key, Iterable<DelayCountTuple> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** total=0;  
 **int** delayedTotal=0;  
  
 **for**(DelayCountTuple dt: values){  
 total += dt.getFlightCount();  
 delayedTotal +=dt.getDelayedFlightCount();  
 }  
  
 **double** percent = ((**double**)delayedTotal/total)\*100;  
  
 **res**.setDelayedFlightCount(delayedTotal);  
 **res**.setFlightCount(total);  
 **res**.setDelayPercent(percent);  
  
 context.write(key,**res**);  
 }  
}

**12 Hierarchical Pattern**

**package** hadoop.project.hier\_pattern;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** AirportMapper **extends** Mapper<Object, Text, Text, Text> {  
  
 **private** Text **outkey** = **new** Text();  
 **private** Text **outvalue** = **new** Text();  
  
 **public void** map(Object key, Text value, Mapper.Context context)  
 **throws** IOException, InterruptedException {  
  
 String[] tokens = value.toString().split(**","**);  
 **if** (tokens[0].equals(**"iata"**))  
 **return**;  
  
 *// The foreign join key is the post ID* **outkey**.set(tokens[0]);  
  
  
 *// Flag this record for the reducer and then output* StringBuffer sb = **new** StringBuffer();  
 sb.append(tokens[1]);  
 sb.append(**" "**);  
 sb.append(tokens[2]);  
 sb.append(**" "**);  
 sb.append(tokens[3]);  
 sb.append(**" "**);  
 sb.append(tokens[4]);  
  
 **outvalue**.set(**"P"** + sb.toString());  
 context.write(**outkey**, **outvalue**);  
 }  
}

**package** hadoop.project.hier\_pattern;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Job;  
**import** org.apache.hadoop.mapreduce.lib.input.MultipleInputs;  
**import** org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**public class** HierMain {  
 **public static void** main(String[] args) **throws** Exception {  
 Configuration conf = **new** Configuration();  
 Job job = **new** Job(conf, **"Hierarchy"**);  
 job.setJarByClass(HierMain.**class**);  
  
 MultipleInputs.*addInputPath*(job, **new** Path(args[0]), TextInputFormat.**class**, AirportMapper.**class**);  
  
 MultipleInputs.*addInputPath*(job, **new** Path(args[1]), TextInputFormat.**class**, SrcCarrierMapper.**class**);  
  
 job.setReducerClass(HierReducer.**class**);  
  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
 TextOutputFormat.*setOutputPath*(job, **new** Path(args[2]));  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(Text.**class**);  
  
 System.*exit*(job.waitForCompletion(**true**) ? 0 : 2);  
 }  
}

**package** hadoop.project.hier\_pattern;  
  
  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
**import** org.w3c.dom.Attr;  
**import** org.w3c.dom.Document;  
**import** org.w3c.dom.Element;  
**import** org.w3c.dom.NamedNodeMap;  
**import** org.xml.sax.InputSource;  
  
**import** javax.xml.parsers.DocumentBuilder;  
**import** javax.xml.parsers.DocumentBuilderFactory;  
**import** javax.xml.transform.OutputKeys;  
**import** javax.xml.transform.Transformer;  
**import** javax.xml.transform.TransformerFactory;  
**import** javax.xml.transform.dom.DOMSource;  
**import** javax.xml.transform.stream.StreamResult;  
**import** java.io.IOException;  
**import** java.io.StringReader;  
**import** java.io.StringWriter;  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** HierReducer **extends** Reducer<Text, Text, Text, NullWritable> {  
  
 **private** ArrayList<String> **comments** = **new** ArrayList<String>();  
 **private** DocumentBuilderFactory **dbf** = DocumentBuilderFactory.*newInstance*();  
 **private** String **post** = **null**;  
  
 @Override  
 **public void** reduce(Text key, Iterable<Text> values, Context context)  
 **throws** IOException, InterruptedException {  
 *// Reset variables* **post** = **null**;  
 **comments**.clear();  
  
 *// For each input value* **for** (Text t : values) {  
 *// If this is the post record, store it, minus the flag* **if** (t.charAt(0) == **'P'**) {  
 **post** = t.toString().substring(1).trim();  
 } **else** {  
 *// Else, it is a comment record. Add it to the list, minus  
 // the flag* **comments**.add(t.toString().substring(1).trim());  
 }  
 }  
  
 *// If post is not null* **if** (post != **null**) {  
 *// nest the comments underneath the post element* String postWithCommentChildren = nestElements(post, comments);  
 System.out.println(postWithCommentChildren);  
 context.write(**new** Text(postWithCommentChildren), NullWritable.get());  
  
 }  
 }  
  
 **private** String nestElements(String post, List<String> comments) {  
 **try** {  
 *// Create the new document to build the XML* DocumentBuilder bldr = dbf.newDocumentBuilder();  
 Document doc = bldr.newDocument();  
  
 *// Copy parent node to document* Element postEl = getXmlElementFromString(post);  
 Element toAddPostEl = doc.createElement(**"Airport"**);  
  
 *// Copy the attributes of the original post element to the new  
 // one* copyAttributesToElement(postEl.getAttributes(), toAddPostEl);  
  
 *// For each comment, copy it to the "post" node* **for** (String commentXml : comments) {  
 Element commentEl = getXmlElementFromString(commentXml);  
 Element toAddCommentEl = doc.createElement(**"Carriers"**);  
  
 *// Copy the attributes of the original comment element to  
 // the new one* copyAttributesToElement(commentEl.getAttributes(),  
 toAddCommentEl);  
  
 *// Add the copied comment to the post element* toAddPostEl.appendChild(toAddCommentEl);  
 }  
  
 *// Add the post element to the document* doc.appendChild(toAddPostEl);  
  
 *// Transform the document into a String of XML and return* **return** transformDocumentToString(doc);  
  
 } **catch** (Exception e) {  
 **return null**;  
 }  
 }  
  
 **private** Element getXmlElementFromString(String xml) {  
 **try** {  
 *// Create a new document builder* DocumentBuilder bldr = dbf.newDocumentBuilder();  
  
 *// Parse the XML string and return the first element* **return** bldr.parse(**new** InputSource(**new** StringReader(xml)))  
 .getDocumentElement();  
 } **catch** (Exception e) {  
 **return null**;  
 }  
 }  
  
 **private void** copyAttributesToElement(NamedNodeMap attributes,  
 Element element) {  
  
 *// For each attribute, copy it to the element* **for** (**int** i = 0; i < attributes.getLength(); ++i) {  
 Attr toCopy = (Attr) attributes.item(i);  
 element.setAttribute(toCopy.getName(), toCopy.getValue());  
 }  
 }  
  
 **private** String transformDocumentToString(Document doc) {  
 **try** {  
 TransformerFactory tf = TransformerFactory.newInstance();  
 Transformer transformer = tf.newTransformer();  
 transformer.setOutputProperty(OutputKeys.OMIT\_XML\_DECLARATION,  
 **"yes"**);  
 StringWriter writer = **new** StringWriter();  
 transformer.transform(**new** DOMSource(doc), **new** StreamResult(  
 writer));  
 *// Replace all new line characters with an empty string to have  
 // one record per line.* **return** writer.getBuffer().toString().replaceAll(**"\n|\r"**, **""**);  
 } **catch** (Exception e) {  
 **return null**;  
 }  
 }  
}

**package** hadoop.project.hier\_pattern;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** SrcCarrierMapper **extends** Mapper<Object, Text, Text, Text> {  
 **private** Text **outkey** = **new** Text();  
 **private** Text **outvalue** = **new** Text();  
  
 **public void** map(Object key, Text value, Mapper.Context context) **throws** IOException, InterruptedException {  
  
 String[] tokens = value.toString().split(**"\t"**);  
  
 *// The foreign join key is the post ID* **outkey**.set(tokens[0]);  
  
 *// Flag this record for the reducer and then output* **outvalue**.set(**"C"** + tokens[1]);  
 context.write(**outkey**, **outvalue**);  
 }  
}

**13- Inner Join on Carriers**

**package** hadoop.project.inner\_join\_carriers;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.LongWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** CarrierMapper **extends** Mapper<LongWritable, Text, Text, Text> {  
  
 *// hadoop datatype* Text **word** = **new** Text();  
 IntWritable **one** = **new** IntWritable(1);  
  
 @Override  
 **protected void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  
  
 String line = value.toString();  
 String[] tokens= line.split(**","**);  
 **if**(tokens[0].equals(**"Code"**))**return**;  
 String newKey = tokens[0];  
 **word**.set(newKey);  
 String outValue= **"A"**+tokens[1];  
 context.write(**word**,**new** Text(outValue));  
 }  
  
}

**package** hadoop.project.inner\_join\_carriers;  
  
**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.lib.input.FileInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.input.MultipleInputs;  
**import** org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** FlightMain {  
  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 Job job = **new** Job(conf, **"Reduce-side join"**);  
 job.setJarByClass(FlightMain.**class**);  
 job.setReducerClass(FlightReducer.**class**);  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(Text.**class**);  
  
 MultipleInputs.*addInputPath*(job, **new** Path(args[0]),TextInputFormat.**class**, CarrierMapper.**class**);  
 MultipleInputs.*addInputPath*(job, **new** Path(args[1]),TextInputFormat.**class**, FlightMapper.**class**);  
 Path outputPath = **new** Path(args[2]);  
  
 FileOutputFormat.*setOutputPath*(job, outputPath);  
 outputPath.getFileSystem(conf).delete(outputPath);  
 System.*exit*(job.waitForCompletion(**true**) ? 0 : 1);  
  
 }  
}

**package** hadoop.project.inner\_join\_carriers;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.LongWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** FlightMapper **extends** Mapper<LongWritable, Text, Text, Text> {  
  
 *// hadoop datatype* Text **word** = **new** Text();  
 IntWritable **one** = **new** IntWritable(1);  
  
 @Override  
 **protected void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  
  
 String line = value.toString();  
 String[] tokens= line.split(**"\t"**);  
 **if**(tokens[0].equals(**"Code"**))**return**;  
 String newKey = tokens[0];  
 **word**.set(newKey);  
 String outValue= **"B"**+tokens[1];  
 context.write(**word**,**new** Text(outValue));  
 }  
  
}

**package** hadoop.project.inner\_join\_carriers;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
**import** java.util.ArrayList;  
**import** java.util.Iterator;  
  
**public class** FlightReducer **extends** Reducer<Text, Text, Text, Text> {  
  
 **private static final** Text ***EMPTY\_TEXT*** = **new** Text(**" "**);  
 **private** Text **tmp** = **new** Text();  
 **private** ArrayList<Text> **listA** = **new** ArrayList<>();  
 **private** ArrayList<Text> **listB** = **new** ArrayList<>();  
 **private** String **joinType** = **null**;  
  
 @Override  
 **protected void** setup(Context context) **throws** IOException, InterruptedException {  
 *//joinType = context.getConfiguration().get("join.type");* **joinType**=**"inner"**;  
 }  
  
 @Override  
 **protected void** reduce(Text key, Iterable<Text> values, Context context) **throws** IOException, InterruptedException {  
 **listA**.clear();  
 **listB**.clear();  
 Iterator<Text> itr = values.iterator();  
 **while**(itr.hasNext()){  
 **tmp**= itr.next();  
  
 **if**(**tmp**.charAt(0)==**'A'**){  
 **listA**.add(**new** Text(**tmp**.toString().substring(1)));  
 }**else if**(**tmp**.charAt(0)==**'B'**){  
 **listB**.add(**new** Text(**tmp**.toString().substring(1)));  
 }  
 }  
  
 executeJoinLogic(context);  
 }  
  
 **private void** executeJoinLogic(Context context) **throws** IOException, InterruptedException {  
 **if**(**joinType**.equals(**"inner"**)){  
 **if**(!**listA**.isEmpty() && !**listB**.isEmpty()){  
 **for**(Text textA:**listA**){  
 **for**(Text textB:**listB**){  
 context.write(textA,textB);  
 }  
 }  
 }  
 }  
 }  
}

**14- Inverted Index**

**package** hadoop.project.inverted\_index;  
  
**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.lib.input.FileInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** IndexMain {  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(IndexMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(Text.**class**);  
  
  
  
  
 job.setMapperClass(IndexMapper.**class**);  
  
 job.setReducerClass(IndexReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(Text.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.inverted\_index;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.LongWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** IndexMapper **extends** Mapper<LongWritable, Text, Text, Text> {  
  
 @Override  
 **protected void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  
  
 String line = value.toString();  
 String[] tokens= line.split(**","**);  
 **if**(tokens[0].equals(**"Year"**))**return**;  
 String src = tokens[16];  
 String dest = tokens[17];  
  
 context.write(**new** Text(src+**":"**),**new** Text(dest));  
 }  
  
  
}

**package** hadoop.project.inverted\_index;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
**import** java.util.HashSet;  
  
**public class** IndexReducer **extends** Reducer<Text, Text, Text, Text> {  
  
 *// just like in mongoDB values is iterable* HashSet<String> **hs** = **new** HashSet<>();  
 @Override  
 **protected void** reduce(Text key, Iterable<Text> values, Context context) **throws** IOException, InterruptedException {  
 **hs**.clear();  
 StringBuffer sb = **new** StringBuffer(**""**);  
 **for**(Text v: values){  
 **hs**.add(v.toString());  
 *// can we use this-- Integer.parseInt(v.toString());* }  
  
 **for**(String v: **hs**){  
 sb.append(v);  
 sb.append(**" "**);  
 }  
  
 context.write(key, **new** Text(sb.toString()));  
  
 *//super.reduce(key, values, context); //To change body of generated methods, choose Tools | Templates.* }  
  
}

**15- Inverted Index Helper**

**package** hadoop.project.inverted\_index\_count;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** IndexMain {  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(IndexMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(Text.**class**);  
  
  
  
  
 job.setMapperClass(IndexMapper.**class**);  
  
 job.setReducerClass(IndexReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(Text.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.inverted\_index\_count;  
  
**import** org.apache.hadoop.io.LongWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** IndexMapper **extends** Mapper<LongWritable, Text, Text, Text> {  
  
 @Override  
 **protected void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  
  
 String line = value.toString();  
 String[] tokens= line.split(**"\t"**);  
 String pair[] = tokens[0].split(**"-"**);  
 String src = pair[0];  
 String dest = pair[1];  
  
 context.write(**new** Text(src+**":"**),**new** Text(dest));  
 }  
  
  
}

**package** hadoop.project.inverted\_index\_count;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
**import** java.util.HashSet;  
  
**public class** IndexReducer **extends** Reducer<Text, Text, Text, Text> {  
  
 *// just like in mongoDB values is iterable* HashSet<String> **hs** = **new** HashSet<>();  
 @Override  
 **protected void** reduce(Text key, Iterable<Text> values, Context context) **throws** IOException, InterruptedException {  
 **hs**.clear();  
 StringBuffer sb = **new** StringBuffer(**""**);  
*// for(Text v: values){  
// hs.add(v.toString());  
// // can we use this-- Integer.parseInt(v.toString());  
// }* **for**(Text v: values){  
 sb.append(v.toString());  
 sb.append(**" "**);  
 }  
  
 context.write(key, **new** Text(sb.toString()));  
  
 *//super.reduce(key, values, context); //To change body of generated methods, choose Tools | Templates.* }  
  
}

**16- Partitioning Pattern on Year**

**package** hadoop.project.partitioning\_pattern\_year;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** PartitionMain {  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(PartitionMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(IntWritable.**class**);  
 job.setMapOutputValueClass(Text.**class**);  
  
  
 job.setMapperClass(PartitionMapper.**class**);  
 *// Set custom partitioner and min last access date* job.setPartitionerClass(PartitionPartitioner.**class**);  
 PartitionPartitioner.*setMinLastAccessDate*(job, 1987);  
  
*// Last access dates span between 2008-2011, or 4 years* job.setNumReduceTasks(22);  
  
 job.setReducerClass(PartitionReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(NullWritable.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.partitioning\_pattern\_year;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.LongWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
**import** java.text.SimpleDateFormat;  
  
**public class** PartitionMapper **extends** Mapper<Object, Text, IntWritable, Text> {  
  
  
 **private** IntWritable **outkey** = **new** IntWritable();  
  
 **protected void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
  
 String [] tokens = value.toString().split(**","**);  
 **if**(tokens[0].equals(**"Year"**))  
 **return**;  
  
 StringBuffer sb = **new** StringBuffer();  
 sb.append(tokens[0]);  
 sb.append(**"\t"**);  
 sb.append(tokens[8]);  
 sb.append(**"\t"**);  
 sb.append(tokens[16]);  
 sb.append(**"\t"**);  
 sb.append(tokens[17]);  
  
 **int** year = Integer.*parseInt*(tokens[0]);  
  
 **outkey**.set(year);  
  
 *// Write out the year with the input value* context.write(**outkey**, **new** Text(sb.toString()));  
 }  
}

**package** hadoop.project.partitioning\_pattern\_year;  
  
**import** org.apache.hadoop.conf.Configurable;  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Job;  
**import** org.apache.hadoop.mapreduce.Partitioner;  
  
**public class** PartitionPartitioner **extends** Partitioner<IntWritable, Text> **implements** Configurable {  
  
 **private static final** String ***MIN\_LAST\_ACCESS\_DATE\_YEAR*** =  
 **"min.last.access.date.year"**;  
  
 **private** Configuration **conf** = **null**;  
 **private int minLastAccessDateYear** = 0;  
  
 **public int** getPartition(IntWritable key, Text value, **int** numPartitions) {  
 **return** key.get() - **minLastAccessDateYear**;  
 }  
  
 **public** Configuration getConf() {  
 **return conf**;  
 }  
  
 **public void** setConf(Configuration conf) {  
 **this**.**conf** = conf;  
 **minLastAccessDateYear** = conf.getInt(***MIN\_LAST\_ACCESS\_DATE\_YEAR***, 0);  
 }  
  
 **public static void** setMinLastAccessDate(Job job,  
 **int** minLastAccessDateYear) {  
 job.getConfiguration().setInt(***MIN\_LAST\_ACCESS\_DATE\_YEAR***,  
 minLastAccessDateYear);  
 }  
}

**package** hadoop.project.partitioning\_pattern\_year;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** PartitionReducer **extends** Reducer<IntWritable, Text, Text, NullWritable> {  
  
 **protected void** reduce(IntWritable key, Iterable<Text> values,  
 Context context) **throws** IOException, InterruptedException {  
 **for** (Text t : values) {  
 context.write(t, NullWritable.*get*());  
 }  
 }  
}

**17- Recommendation System**

*/\*  
 \* To change this license header, choose License Headers in Project Properties.  
 \* To change this template file, choose Tools | Templates  
 \* and open the template in the editor.  
 \*/***package** hadoop.project.recommendation\_system;  
  
**import** org.apache.hadoop.io.WritableComparable;  
  
**import** java.io.DataInput;  
**import** java.io.DataOutput;  
**import** java.io.IOException;  
  
*/\*\*  
 \*  
 \** ***@author*** *ankit  
 \*/***public class** CompositeKey **implements** WritableComparable<CompositeKey>{  
  
 **private** String **srcDest**;  
 **private** String **carrierInfo**;  
  
 **public** CompositeKey() {  
 **super**();  
 }  
  
 **public** String getSrcDest() {  
 **return srcDest**;  
 }  
  
 **public void** setSrcDest(String srcDest) {  
 **this**.**srcDest** = srcDest;  
 }  
  
 **public** String getCarrierInfo() {  
 **return carrierInfo**;  
 }  
  
 **public void** setCarrierInfo(String carrierInfo) {  
 **this**.**carrierInfo** = carrierInfo;  
 }  
  
 **public** CompositeKey(String srcDest, String carrierInfo) {  
 **this**.**srcDest** = srcDest;  
 **this**.**carrierInfo** = carrierInfo;  
 }  
  
 @Override  
 **public void** write(DataOutput d) **throws** IOException {  
 d.writeUTF(**srcDest**);  
 d.writeUTF(**carrierInfo**);  
 }  
  
 @Override  
 **public void** readFields(DataInput di) **throws** IOException {  
 **srcDest** = di.readUTF();  
 **carrierInfo** = di.readUTF();  
 }  
  
 @Override  
 **public int** compareTo(CompositeKey o) {  
 **int** result = **this**.**srcDest**.compareTo(o.getSrcDest());  
 **if**(result==0){  
 String c1 = **this**.**carrierInfo**;  
 Double rms1 = Double.*parseDouble*(c1.split(**"\t"**)[1]);  
  
 String c2 = o.getCarrierInfo();  
 Double rms2 = Double.*parseDouble*(c2.split(**"\t"**)[1]);  
 **return** rms1.compareTo(rms2);  
 }  
   
 **return** result;  
 }  
  
 @Override  
 **public** String toString() {  
 **return srcDest** + **" : "** + **carrierInfo**;  
 }  
}

*/\*  
 \* To change this license header, choose License Headers in Project Properties.  
 \* To change this template file, choose Tools | Templates  
 \* and open the template in the editor.  
 \*/***package** hadoop.project.recommendation\_system;  
  
*/\*\*  
 \*  
 \** ***@author*** *ankit  
 \*/***import** org.apache.hadoop.io.WritableComparator;  
  
**public class** GroupComparator **extends** WritableComparator{  
  
 **protected** GroupComparator() {  
 **super**(CompositeKey.**class**, **true**);  
 }  
  
 @Override  
 **public int** compare(Object a, Object b) {  
   
 CompositeKey ckw1 = (CompositeKey)a;  
 CompositeKey ckw2 = (CompositeKey)b;  
   
 **return** ckw1.getSrcDest().compareTo(ckw2.getSrcDest());  
 }  
}

*/\*  
 \* To change this license header, choose License Headers in Project Properties.  
 \* To change this template file, choose Tools | Templates  
 \* and open the template in the editor.  
 \*/***package** hadoop.project.recommendation\_system;  
  
*/\*\*  
 \*  
 \** ***@author*** *ankit  
 \*/***import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.mapreduce.Partitioner;  
  
**public class** KeyPartition **extends** Partitioner<CompositeKey, NullWritable>{  
  
 @Override  
 **public int** getPartition(CompositeKey key, NullWritable value, **int** numPartitions) {  
   
 **return** key.getSrcDest().hashCode()%numPartitions;  
   
 }  
  
}

*/\*  
 \* To change this license header, choose License Headers in Project Properties.  
 \* To change this template file, choose Tools | Templates  
 \* and open the template in the editor.  
 \*/***package** hadoop.project.recommendation\_system;  
  
*/\*\*  
 \*  
 \** ***@author*** *ankit  
 \*/***import** org.apache.hadoop.io.WritableComparable;  
**import** org.apache.hadoop.io.WritableComparator;  
  
**import** java.text.ParseException;  
**import** java.text.SimpleDateFormat;  
**import** java.util.Date;  
**import** java.util.logging.Level;  
**import** java.util.logging.Logger;  
  
**public class** SecondarySortComparator **extends** WritableComparator {  
   
 **private final static** SimpleDateFormat ***frmt*** = **new** SimpleDateFormat(**"yyyy-MM-dd"**);  
  
 **protected** SecondarySortComparator() {  
 **super**(CompositeKey.**class**, **true**);  
 }  
  
 @Override  
 **public int** compare(WritableComparable a, WritableComparable b) {  
   
   
  
 CompositeKey ck1 = (CompositeKey) a;  
 CompositeKey ck2 = (CompositeKey) b;  
  
 **int** result = ck1.getSrcDest().compareTo(ck2.getSrcDest());  
  
 **if** (result == 0) {  
 String c1 = ck1.getCarrierInfo();  
 Double rms1 = Double.*parseDouble*(c1.split(**"\t"**)[1]);  
  
 String c2 = ck2.getCarrierInfo();  
 Double rms2 = Double.*parseDouble*(c2.split(**"\t"**)[1]);  
 result = rms1.compareTo(rms2);  
   
 }  
  
 **return** result;  
 }  
  
   
   
}

*/\*  
 \* To change this license header, choose License Headers in Project Properties.  
 \* To change this template file, choose Tools | Templates  
 \* and open the template in the editor.  
 \*/***package** hadoop.project.recommendation\_system;  
  
**import** org.apache.hadoop.fs.FileSystem;  
**import** org.apache.hadoop.fs.Path;  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.mapreduce.Job;  
**import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
  
**import** java.io.IOException;  
  
*/\*\*  
 \*  
 \** ***@author*** *ankit  
 \*/***public class** SecondarySortDriver {  
   
 **public static void** main( String[] args ) **throws** IOException, ClassNotFoundException, InterruptedException  
 {  
 Job job = Job.*getInstance*();  
   
 job.setJarByClass(SecondarySortDriver.**class**);  
   
 job.setGroupingComparatorClass(GroupComparator.**class**);  
 job.setSortComparatorClass(SecondarySortComparator.**class**);  
 job.setPartitionerClass(KeyPartition.**class**);  
   
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 Path outDir = **new** Path(args[1]);  
 FileOutputFormat.*setOutputPath*(job, outDir);  
   
 job.setMapperClass(SecondarySortMapper.**class**);  
 job.setReducerClass(SecondarySortReducer.**class**);  
   
 job.setNumReduceTasks(1);  
   
 job.setOutputKeyClass(CompositeKey.**class**);  
 job.setOutputValueClass(NullWritable.**class**);  
   
 FileSystem fs = FileSystem.*get*(job.getConfiguration());  
 **if**(fs.exists(outDir)) {  
 fs.delete(outDir, **true**);  
 }  
   
 job.waitForCompletion(**true**);  
 }  
}

*/\*  
 \* To change this license header, choose License Headers in Project Properties.  
 \* To change this template file, choose Tools | Templates  
 \* and open the template in the editor.  
 \*/***package** hadoop.project.recommendation\_system;  
  
**import** org.apache.hadoop.io.LongWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
*/\*\*  
 \*  
 \** ***@author*** *ankit  
 \*/***public class** SecondarySortMapper **extends** Mapper<LongWritable, Text, CompositeKey, NullWritable>{  
  
 @Override  
 **protected void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  
 *//To change body of generated methods, choose Tools | Templates.* String [] tokens = value.toString().split(**"\t"**,2);  
  
 **try** {  
 String srcDest = tokens[0];  
 String carrInfo = tokens[1];  
  
 CompositeKey coKey = **new** CompositeKey(srcDest, carrInfo);  
  
 context.write(coKey, NullWritable.*get*());  
 }**catch**(Exception e){  
 e.getStackTrace();  
 }  
   
 }  
   
   
}

*/\*  
 \* To change this license header, choose License Headers in Project Properties.  
 \* To change this template file, choose Tools | Templates  
 \* and open the template in the editor.  
 \*/***package** hadoop.project.recommendation\_system;  
  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
*/\*\*  
 \*  
 \** ***@author*** *ankit  
 \*/***public class** SecondarySortReducer **extends** Reducer<CompositeKey, NullWritable, CompositeKey, NullWritable>{  
  
 @Override  
 **protected void** reduce(CompositeKey key, Iterable<NullWritable> values, Context context) **throws** IOException, InterruptedException {  
 *//To change body of generated methods, choose Tools | Templates.* **for**(NullWritable v:values){  
 context.write(key, v);  
 }  
 }  
   
}

**18- RMS for src and dest delays**

**package** hadoop.project.rms\_src\_dest\_carriers;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** RMSCombiner **extends** Reducer<Text, RMSCountTuple, Text, RMSCountTuple> {  
  
 **private** RMSCountTuple **res**= **new** RMSCountTuple();  
  
 @Override  
 **protected void** reduce(Text key, Iterable<RMSCountTuple> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** total=0;  
 **int** arrDelay=0;  
 **int** depDelay=0;  
  
 **for**(RMSCountTuple tup : values){  
 total +=tup.getTotalFlight();  
 arrDelay +=tup.getArrDelay();  
 depDelay +=tup.getDepDelay();  
 }  
  
 **res**.setTotalFlight(total);  
 **res**.setArrDelay(arrDelay);  
 **res**.setDepDelay(depDelay);  
  
 context.write(key,**res**);  
 }  
}

**package** hadoop.project.rms\_src\_dest\_carriers;  
  
**import** org.apache.hadoop.io.Writable;  
  
**import** java.io.DataInput;  
**import** java.io.DataOutput;  
**import** java.io.IOException;  
  
**public class** RMSCountTuple **implements** Writable {  
  
 **private int arrDelay**=0;  
 **private int depDelay**=0;  
 **private int totalFlight**=0;  
 **private double rms** =0.0;  
  
 **public int** getArrDelay() {  
 **return arrDelay**;  
 }  
  
 **public void** setArrDelay(**int** arrDelay) {  
 **this**.**arrDelay** = arrDelay;  
 }  
  
 **public int** getDepDelay() {  
 **return depDelay**;  
 }  
  
 **public void** setDepDelay(**int** depDelay) {  
 **this**.**depDelay** = depDelay;  
 }  
  
 **public int** getTotalFlight() {  
 **return totalFlight**;  
 }  
  
 **public void** setTotalFlight(**int** totalFlight) {  
 **this**.**totalFlight** = totalFlight;  
 }  
  
 **public double** getRms() {  
 **return rms**;  
 }  
  
 **public void** setRms(**double** rms) {  
 **this**.**rms** = rms;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "{"** +  
 **"arrDelay="** + **arrDelay** +  
 **", depDelay="** + **depDelay** +  
 **", totalFlight="** + **totalFlight** +  
 **", rms="** + String.*format*(**"%.4f"**, **rms**)+  
 **'}'**;  
 }  
  
*//  
// @Override  
// public String toString() {  
// return String.format("%.4f", rms);  
// }* @Override  
 **public void** write(DataOutput dataOutput) **throws** IOException {  
 dataOutput.writeInt(**arrDelay**);  
 dataOutput.writeInt(**depDelay**);  
 dataOutput.writeInt(**totalFlight**);  
 dataOutput.writeDouble(**rms**);  
  
 }  
  
 @Override  
 **public void** readFields(DataInput dataInput) **throws** IOException {  
  
 **arrDelay** = dataInput.readInt();  
 **depDelay** = dataInput.readInt();  
 **totalFlight** = dataInput.readInt();  
 **rms** = dataInput.readDouble();  
  
 }  
}

**package** hadoop.project.rms\_src\_dest\_carriers;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** RMSMain {  
  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(RMSMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(RMSCountTuple.**class**);  
  
  
 job.setMapperClass(RMSMapper.**class**);  
 job.setCombinerClass(RMSCombiner.**class**);  
 job.setReducerClass(RMSReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(RMSCountTuple.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.rms\_src\_dest\_carriers;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
  
**public class** RMSMapper **extends** Mapper<Object, Text, Text, RMSCountTuple> {  
  
 **private** RMSCountTuple **tuple** = **new** RMSCountTuple();  
  
 @Override  
 **protected void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
 String [] tokens = value.toString().split(**","**);  
  
 **if**(tokens[0].equals(**"Year"**))**return**;  
  
 String src = tokens[16];  
 String dest = tokens[17];  
 String carrier = tokens[8];  
  
 **int** arrDelay=0;  
 **int** depDelay=0;  
  
  
  
 **try** {  
 arrDelay = Integer.*parseInt*(tokens[14]);  
 depDelay = Integer.*parseInt*(tokens[15]);  
 }**catch** (Exception e){  
  
 }  
  
 String newKey = src+**"-"** + dest +**"\t"**+carrier;  
  
 **tuple**.setArrDelay(arrDelay);  
 **tuple**.setDepDelay(depDelay);  
 **tuple**.setTotalFlight(1);  
  
 context.write(**new** Text(newKey),**tuple**);  
 }  
}

**package** hadoop.project.rms\_src\_dest\_carriers;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** RMSReducer **extends** Reducer<Text, RMSCountTuple, Text, RMSCountTuple> {  
  
 **private** RMSCountTuple **res**= **new** RMSCountTuple();  
  
 @Override  
 **protected void** reduce(Text key, Iterable<RMSCountTuple> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** total=0;  
 **int** arrDelay=0;  
 **int** depDelay=0;  
  
 **for**(RMSCountTuple tup : values){  
 total +=tup.getTotalFlight();  
 arrDelay +=tup.getArrDelay();  
 depDelay +=tup.getDepDelay();  
 }  
  
 **double** avgArrDelay = (**double**)arrDelay/total;  
 **double** avgDepDelay = (**double**)depDelay/total;  
  
 **double** rms = Math.*sqrt*((avgArrDelay\*avgArrDelay) + (avgDepDelay\*avgDepDelay));  
  
 **res**.setTotalFlight(total);  
 **res**.setArrDelay(arrDelay);  
 **res**.setDepDelay(depDelay);  
 **res**.setRms(rms);  
  
  
  
 context.write(key,**res**);  
 }  
}

**19- SRC Carrier Map**

**package** hadoop.project.src\_carrier\_map;  
  
**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.lib.input.FileInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** MRCount {  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(SrcCarrierMapper.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(Text.**class**);  
  
  
  
  
 job.setMapperClass(SrcCarrierMapper.**class**);  
 job.setReducerClass(SrcCarrierReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(Text.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.src\_carrier\_map;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.LongWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** SrcCarrierMapper **extends** Mapper<LongWritable, Text, Text, Text> {  
  
 *// hadoop datatype* Text **word** = **new** Text();  
  
 @Override  
 **protected void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  
  
 String line = value.toString();  
 String[] tokens= line.split(**","**);  
 **if**(tokens[0].equals(**"Year"**))**return**;  
 String orig = tokens[16];  
 String carrier = tokens[8];  
 **word**.set(orig);  
 context.write(**word**,**new** Text(carrier));  
 }  
  
  
}

**package** hadoop.project.src\_carrier\_map;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
**import** java.util.HashSet;  
  
**public class** SrcCarrierReducer **extends** Reducer<Text, Text, Text, Text> {  
  
 *// just like in mongoDB values is iterable* **private** HashSet<String> **carriers** = **new** HashSet<>();  
 @Override  
 **protected void** reduce(Text key, Iterable<Text> values, Context context) **throws** IOException, InterruptedException {  
  
  
 **for**(Text v: values){  
 **carriers**.add(v.toString());  
 *// can we use this-- Integer.parseInt(v.toString());* }  
 **for**(String car: **carriers**)  
 context.write(key, **new** Text(car));  
  
 *//super.reduce(key, values, context); //To change body of generated methods, choose Tools | Templates.* }  
  
}

**21- Src-Dest pair count**

**package** hadoop.project.src\_dest\_count;  
  
**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.lib.input.FileInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** MRCount {  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(SrcDestMapper.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(IntWritable.**class**);  
  
  
  
  
 job.setMapperClass(SrcDestMapper.**class**);  
 job.setReducerClass(SrcDestReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(IntWritable.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.src\_dest\_count;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.LongWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** SrcDestMapper **extends** Mapper<LongWritable, Text, Text, IntWritable> {  
  
 *// hadoop datatype* Text **word** = **new** Text();  
 IntWritable **one** = **new** IntWritable(1);  
  
 @Override  
 **protected void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  
  
 String line = value.toString();  
 String[] tokens= line.split(**","**);  
 **if**(tokens[0].equals(**"Year"**))**return**;  
 String orig\_dest = tokens[16]+**"-"** + tokens[17];  
 **word**.set(orig\_dest);  
 context.write(**word**,**one**);  
 }  
  
  
}

**package** hadoop.project.src\_dest\_count;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** SrcDestReducer **extends** Reducer<Text, IntWritable, Text, IntWritable> {  
  
 *// just like in mongoDB values is iterable* @Override  
 **protected void** reduce(Text key, Iterable<IntWritable> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** sum=0;  
 **for**(IntWritable v: values){  
 sum += v.get();  
 *// can we use this-- Integer.parseInt(v.toString());* }  
  
 context.write(key, **new** IntWritable(sum));  
  
 *//super.reduce(key, values, context); //To change body of generated methods, choose Tools | Templates.* }  
  
}

**package** hadoop.project.src\_dest\_count;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.LongWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** WordMapper **extends** Mapper<LongWritable, Text, Text, IntWritable> {  
  
 *// hadoop datatype* Text **word** = **new** Text();  
 IntWritable **one** = **new** IntWritable(1);  
  
 @Override  
 **protected void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  
  
 String line = value.toString();  
 String[] tokens= line.split(**","**);  
 **if**(tokens[0].equals(**"Year"**))**return**;  
 String orig\_dest = tokens[16]+**"-"** + tokens[17];  
 **word**.set(orig\_dest);  
 context.write(**word**,**one**);  
 }  
  
  
}

**package** hadoop.project.src\_dest\_count;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** WordReducer **extends** Reducer<Text, IntWritable, Text, IntWritable> {  
  
 *// just like in mongoDB values is iterable* @Override  
 **protected void** reduce(Text key, Iterable<IntWritable> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** sum=0;  
 **for**(IntWritable v: values){  
 sum += v.get();  
 *// can we use this-- Integer.parseInt(v.toString());* }  
  
 context.write(key, **new** IntWritable(sum));  
  
 *//super.reduce(key, values, context); //To change body of generated methods, choose Tools | Templates.* }  
  
}

**22- Top 20 SRC Dest**

**package** hadoop.project.top20\_src\_dest;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**import** org.apache.hadoop.io.NullWritable;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** DelaySrcMain {  
  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(DelaySrcMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(NullWritable.**class**);  
 job.setMapOutputValueClass(Text.**class**);  
  
  
 job.setMapperClass(DelaySrcMapper.**class**);  
 *//job.setCombinerClass(DelaySrcReducer.class);* job.setReducerClass(DelaySrcReducer.**class**);  
  
 job.setOutputKeyClass(NullWritable.**class**);  
 job.setOutputValueClass(Text.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.top20\_src\_dest;  
  
  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
  
**public class** DelaySrcMapper **extends** Mapper<Object, Text, NullWritable,Text> {  
  
  
 @Override  
 **protected void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
  
 context.write(NullWritable.*get*(),value);  
 }  
}

**package** hadoop.project.top20\_src\_dest;  
  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
**import** java.util.TreeMap;  
  
**public class** DelaySrcReducer **extends** Reducer<NullWritable, Text, NullWritable, Text> {  
 **private** TreeMap<Double, Text> **countMap** = **new** TreeMap<>();  
  
 @Override  
 **protected void** reduce(NullWritable key, Iterable<Text> values, Context context) **throws** IOException, InterruptedException {  
 **for** (Text value : values) {  
  
 String[] tokens = value.toString().split(**"\t"**);  
 **double** perc = Double.*parseDouble*(tokens[1]);  
  
 **countMap**.put(perc, **new** Text(value));  
  
  
 **if** (**countMap**.size() > 20)  
 **countMap**.remove(**countMap**.lastKey());  
 }  
  
 **for** (Text t : **countMap**.values())  
 context.write(NullWritable.*get*(), t);  
 }  
}

**23- Top 30 Busy SRC Dest Pair**

**package** hadoop.project.top30\_busy\_src\_dest\_pair;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**import** org.apache.hadoop.io.NullWritable;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** TopNMain {  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(TopNMapper.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(NullWritable.**class**);  
 job.setMapOutputValueClass(Text.**class**);  
  
  
 job.setMapperClass(TopNMapper.**class**);  
 job.setReducerClass(TopNReducer.**class**);  
  
 job.setOutputKeyClass(NullWritable.**class**);  
 job.setOutputValueClass(Text.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.top30\_busy\_src\_dest\_pair;  
  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
**import** java.util.TreeMap;  
  
**public class** TopNMapper **extends** Mapper<Object, Text, NullWritable, Text> {  
  
 *// store a map of number of trips to src-dest pair* **private** TreeMap<Integer,Text> **countMap** = **new** TreeMap<>();  
  
 @Override  
 **protected void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
 String[] val= value.toString().split(**"\t"**);  
  
 **if**(val.**length**==2){  
 String pair = val[0];  
 **int** count = Integer.*parseInt*(val[1]);  
 **countMap**.put(count,**new** Text(value));  
 }  
  
 **if**(**countMap**.size()>30)  
 **countMap**.remove(**countMap**.firstKey());  
 }  
  
 @Override  
 **protected void** cleanup(Context context) **throws** IOException, InterruptedException {  
 **for**(Text t: **countMap**.values())  
 context.write(NullWritable.*get*(),t);  
 }  
}

**package** hadoop.project.top30\_busy\_src\_dest\_pair;  
  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
**import** java.util.TreeMap;  
  
**public class** TopNReducer **extends** Reducer<NullWritable, Text, NullWritable, Text> {  
 **private** TreeMap<Integer, Text> **countMap** = **new** TreeMap<>();  
  
 @Override  
 **protected void** reduce(NullWritable key, Iterable<Text> values, Context context) **throws** IOException, InterruptedException {  
 **for**(Text value: values){  
 String[] val= value.toString().split(**"\t"**);  
  
 **if**(val.**length**==2){  
 String pair = val[0];  
 **int** count = Integer.*parseInt*(val[1]);  
 **countMap**.put(count,**new** Text(value));  
 }  
  
 **if**(**countMap**.size()>30)  
 **countMap**.remove(**countMap**.firstKey());  
 }  
  
 **for**(Text t: **countMap**.descendingMap().values())  
 context.write(NullWritable.*get*(),t);  
 }  
}

**24- Top dst**

**package** hadoop.project.top\_dst;  
  
**import** org.apache.hadoop.io.Writable;  
  
**import** java.io.DataInput;  
**import** java.io.DataOutput;  
**import** java.io.IOException;  
  
**public class** DelayCountTuple **implements** Writable {  
  
 **private int flightCount**=0;  
 **private int delayedFlightCount**=0;  
 **private double delayPercent** =0.0;  
  
 **public int** getFlightCount() {  
 **return flightCount**;  
 }  
  
 **public void** setFlightCount(**int** flightCount) {  
 **this**.**flightCount** = flightCount;  
 }  
  
 **public int** getDelayedFlightCount() {  
 **return delayedFlightCount**;  
 }  
  
 **public void** setDelayedFlightCount(**int** delayedFlightCount) {  
 **this**.**delayedFlightCount** = delayedFlightCount;  
 }  
  
 **public double** getDelayPercent() {  
 **return delayPercent**;  
 }  
  
 **public void** setDelayPercent(**double** delayPercent) {  
 **this**.**delayPercent** = delayPercent;  
 }  
  
 @Override  
 **public** String toString() {  
 **return ""**+**delayPercent**;  
 }  
  
 @Override  
 **public void** write(DataOutput dataOutput) **throws** IOException {  
 dataOutput.writeInt(**flightCount**);  
 dataOutput.writeInt(**delayedFlightCount**);  
 dataOutput.writeDouble(**delayPercent**);  
  
 }  
  
 @Override  
 **public void** readFields(DataInput dataInput) **throws** IOException {  
  
 **flightCount** = dataInput.readInt();  
 **delayedFlightCount** = dataInput.readInt();  
 **delayPercent** = dataInput.readDouble();  
  
 }  
}

**package** hadoop.project.top\_dst;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**import** org.apache.hadoop.io.DoubleWritable;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** DelayDestMain {  
  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(DelayDestMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(DelayCountTuple.**class**);  
  
  
 job.setMapperClass(DelayDestMapper.**class**);  
 *//job.setCombinerClass(DelaySrcReducer.class);* job.setReducerClass(DelayDestReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(DoubleWritable.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.top\_dst;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
  
**public class** DelayDestMapper **extends** Mapper<Object, Text, Text, DelayCountTuple> {  
  
 **private** DelayCountTuple **tuple** = **new** DelayCountTuple();  
  
 @Override  
 **protected void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
 String [] tokens = value.toString().split(**","**);  
  
 **if**(tokens[0].equals(**"Year"**))**return**;  
  
 String dest = tokens[17];  
  
 **try** {  
 **int** delay = Integer.*parseInt*(tokens[14]);  
  
  
 **if** (delay > 15) {  
 **tuple**.setDelayedFlightCount(1);  
 } **else** {  
 **tuple**.setDelayedFlightCount(0);  
 }  
 }**catch** (Exception e){  
 **tuple**.setDelayedFlightCount(0);  
 }  
  
 **tuple**.setFlightCount(1);  
  
 context.write(**new** Text(dest),**tuple**);  
 }  
}

**package** hadoop.project.top\_dst;  
  
**import** org.apache.hadoop.io.DoubleWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** DelayDestReducer **extends** Reducer<Text, DelayCountTuple, Text, DoubleWritable> {  
  
 **private** DelayCountTuple **res**= **new** DelayCountTuple();  
  
 @Override  
 **protected void** reduce(Text key, Iterable<DelayCountTuple> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** total=0;  
 **int** delayedTotal=0;  
  
 **for**(DelayCountTuple dt: values){  
 total += dt.getFlightCount();  
 delayedTotal +=dt.getDelayedFlightCount();  
 }  
  
 **double** percent = ((**double**)delayedTotal/total)\*100;  
  
 **res**.setDelayedFlightCount(delayedTotal);  
 **res**.setFlightCount(total);  
 **res**.setDelayPercent(percent);  
  
 context.write(key,**new** DoubleWritable(percent));  
 }  
}

**25- Top SRC**

**package** hadoop.project.top\_src;  
  
**import** org.apache.hadoop.io.Writable;  
  
**import** java.io.DataInput;  
**import** java.io.DataOutput;  
**import** java.io.IOException;  
  
**public class** DelayCountTuple **implements** Writable {  
  
 **private int flightCount**=0;  
 **private int delayedFlightCount**=0;  
 **private double delayPercent** =0.0;  
  
 **public int** getFlightCount() {  
 **return flightCount**;  
 }  
  
 **public void** setFlightCount(**int** flightCount) {  
 **this**.**flightCount** = flightCount;  
 }  
  
 **public int** getDelayedFlightCount() {  
 **return delayedFlightCount**;  
 }  
  
 **public void** setDelayedFlightCount(**int** delayedFlightCount) {  
 **this**.**delayedFlightCount** = delayedFlightCount;  
 }  
  
 **public double** getDelayPercent() {  
 **return delayPercent**;  
 }  
  
 **public void** setDelayPercent(**double** delayPercent) {  
 **this**.**delayPercent** = delayPercent;  
 }  
  
 @Override  
 **public** String toString() {  
 **return ""**+**delayPercent**;  
 }  
  
 @Override  
 **public void** write(DataOutput dataOutput) **throws** IOException {  
 dataOutput.writeInt(**flightCount**);  
 dataOutput.writeInt(**delayedFlightCount**);  
 dataOutput.writeDouble(**delayPercent**);  
  
 }  
  
 @Override  
 **public void** readFields(DataInput dataInput) **throws** IOException {  
  
 **flightCount** = dataInput.readInt();  
 **delayedFlightCount** = dataInput.readInt();  
 **delayPercent** = dataInput.readDouble();  
  
 }  
}

**package** hadoop.project.top\_src;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**import** org.apache.hadoop.io.DoubleWritable;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** DelaySrcMain {  
  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(DelaySrcMain.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(DelayCountTuple.**class**);  
  
  
 job.setMapperClass(DelaySrcMapper.**class**);  
 *//job.setCombinerClass(DelaySrcReducer.class);* job.setReducerClass(DelaySrcReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(DoubleWritable.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.top\_src;  
  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
  
**public class** DelaySrcMapper **extends** Mapper<Object, Text, Text, DelayCountTuple> {  
  
 **private** DelayCountTuple **tuple** = **new** DelayCountTuple();  
  
 @Override  
 **protected void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
 String [] tokens = value.toString().split(**","**);  
  
 **if**(tokens[0].equals(**"Year"**))**return**;  
  
 String src = tokens[16];  
  
 **try** {  
 **int** delay = Integer.*parseInt*(tokens[15]);  
  
  
 **if** (delay > 15) {  
 **tuple**.setDelayedFlightCount(1);  
 } **else** {  
 **tuple**.setDelayedFlightCount(0);  
 }  
 }**catch** (Exception e){  
 **tuple**.setDelayedFlightCount(0);  
 }  
  
 **tuple**.setFlightCount(1);  
  
 context.write(**new** Text(src),**tuple**);  
 }  
}

**package** hadoop.project.top\_src;  
  
**import** org.apache.hadoop.io.DoubleWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** DelaySrcReducer **extends** Reducer<Text, DelayCountTuple, Text, DoubleWritable> {  
  
 **private** DelayCountTuple **res**= **new** DelayCountTuple();  
  
 @Override  
 **protected void** reduce(Text key, Iterable<DelayCountTuple> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** total=0;  
 **int** delayedTotal=0;  
  
 **for**(DelayCountTuple dt: values){  
 total += dt.getFlightCount();  
 delayedTotal +=dt.getDelayedFlightCount();  
 }  
  
 **double** percent = ((**double**)delayedTotal/total)\*100;  
  
 **res**.setDelayedFlightCount(delayedTotal);  
 **res**.setFlightCount(total);  
 **res**.setDelayPercent(percent);  
  
 context.write(key,**new** DoubleWritable(percent));  
 }  
}

**26- Total Count**

**package** hadoop.project.total\_count;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** MRCount {  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(WordMapper.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(NullWritable.**class**);  
 job.setMapOutputValueClass(IntWritable.**class**);  
  
  
 job.setMapperClass(WordMapper.**class**);  
 job.setReducerClass(WordReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(IntWritable.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.total\_count;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.LongWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** WordMapper **extends** Mapper<LongWritable, Text, NullWritable, IntWritable> {  
  
 *// hadoop datatype* Text **word** = **new** Text();  
 IntWritable **one** = **new** IntWritable(1);  
  
 @Override  
 **protected void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  
  
  
 context.write(NullWritable.*get*(),**one**);  
 }  
  
  
}

**package** hadoop.project.total\_count;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** WordReducer **extends** Reducer<NullWritable, IntWritable, NullWritable, IntWritable> {  
  
 *// just like in mongoDB values is iterable* @Override  
 **protected void** reduce(NullWritable key, Iterable<IntWritable> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** sum=0;  
 **for**(IntWritable v: values){  
 sum += v.get();  
 *// can we use this-- Integer.parseInt(v.toString());* }  
  
 context.write(key, **new** IntWritable(sum));  
  
 *//super.reduce(key, values, context); //To change body of generated methods, choose Tools | Templates.* }  
  
}

**26- Unique Carrier Count**

**package** hadoop.project.unique\_carrier\_count;  
  
**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.lib.input.FileInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** CarrierMain {  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(CarrierMapper.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(IntWritable.**class**);  
  
  
  
  
 job.setMapperClass(CarrierMapper.**class**);  
 job.setReducerClass(CarrierReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(IntWritable.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.unique\_carrier\_count;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.LongWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** CarrierMapper **extends** Mapper<LongWritable, Text, Text, IntWritable> {  
  
 *// hadoop datatype* Text **word** = **new** Text();  
 IntWritable **one** = **new** IntWritable(1);  
  
 @Override  
 **protected void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  
  
 String line = value.toString();  
 String[] tokens= line.split(**","**);  
 **if**(tokens[0].equals(**"Year"**))**return**;  
  
 String carrier = tokens[8];  
 **word**.set(carrier);  
 context.write(**word**,**one**);  
 }  
  
  
}

**package** hadoop.project.unique\_carrier\_count;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** CarrierReducer **extends** Reducer<Text, IntWritable, Text, IntWritable> {  
  
 *// just like in mongoDB values is iterable* @Override  
 **protected void** reduce(Text key, Iterable<IntWritable> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** sum=0;  
 **for**(IntWritable v: values){  
 sum += v.get();  
 *// can we use this-- Integer.parseInt(v.toString());* }  
  
 context.write(key, **new** IntWritable(sum));  
  
 *//super.reduce(key, values, context); //To change body of generated methods, choose Tools | Templates.* }  
  
}

**27 Yearly Data**

**package** hadoop.project.yearly\_data;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.Path;  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**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.input.TextInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
**import** java.io.IOException;  
  
**public class** MRCount {  
 **public static void** main(String[] args) **throws** IOException, InterruptedException, ClassNotFoundException{  
  
  
 Configuration conf = **new** Configuration();  
 *// Create a new Job* Job job = Job.*getInstance*(conf,**"wordcount"**);  
 job.setJarByClass(MRCount.**class**);  
  
 *// Specify various job-specific parameters* job.setJobName(**"myjob"**);  
  
  
 FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  
  
  
 job.setInputFormatClass(TextInputFormat.**class**);  
 job.setOutputFormatClass(TextOutputFormat.**class**);  
  
 job.setMapOutputKeyClass(Text.**class**);  
 job.setMapOutputValueClass(IntWritable.**class**);  
  
  
 job.setMapperClass(YearMapper.**class**);  
 job.setReducerClass(YearReducer.**class**);  
  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(IntWritable.**class**);  
  
 *// Submit the job, then poll for progress until the job is complete* System.*exit*(job.waitForCompletion(**true**)?0:1);  
  
 }  
}

**package** hadoop.project.yearly\_data;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.LongWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**import** java.io.IOException;  
  
**public class** YearMapper **extends** Mapper<LongWritable, Text, Text, IntWritable> {  
  
 *// hadoop datatype* Text **word** = **new** Text();  
 IntWritable **one** = **new** IntWritable(1);  
  
 @Override  
 **protected void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  
  
 String line = value.toString();  
 String[] tokens= line.split(**","**);  
 **if**(tokens[0].equals(**"Year"**))**return**;  
 String year = tokens[0];  
 **word**.set(year);  
 context.write(**word**,**one**);  
 }  
  
  
}

**package** hadoop.project.yearly\_data;  
  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.NullWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**import** java.io.IOException;  
  
**public class** YearReducer **extends** Reducer<Text, IntWritable, Text, IntWritable> {  
  
 *// just like in mongoDB values is iterable* @Override  
 **protected void** reduce(Text key, Iterable<IntWritable> values, Context context) **throws** IOException, InterruptedException {  
  
 **int** sum=0;  
 **for**(IntWritable v: values){  
 sum += v.get();  
 *// can we use this-- Integer.parseInt(v.toString());* }  
  
 context.write(key, **new** IntWritable(sum));  
  
 *//super.reduce(key, values, context); //To change body of generated methods, choose Tools | Templates.* }  
  
}