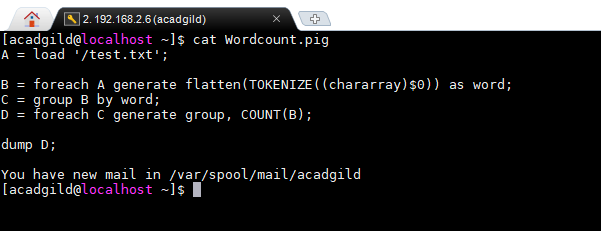
**Task 1 : Write a program to implement wordcount using Pig.**

The program is coded inside PIG script Wordcount.pig



In relation A we are loading the text ‘test.txt’ for which the word count has to be done

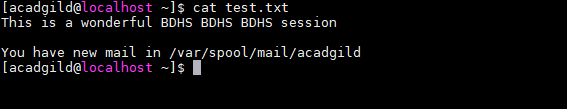
In relation B, each word in a line is tokenized and then it is converted to a single column with multiple rows using FLATTEN

Relation C, group by on words

In relation D, for each group of words generate word count

Dump D prints the wordcount on the Console

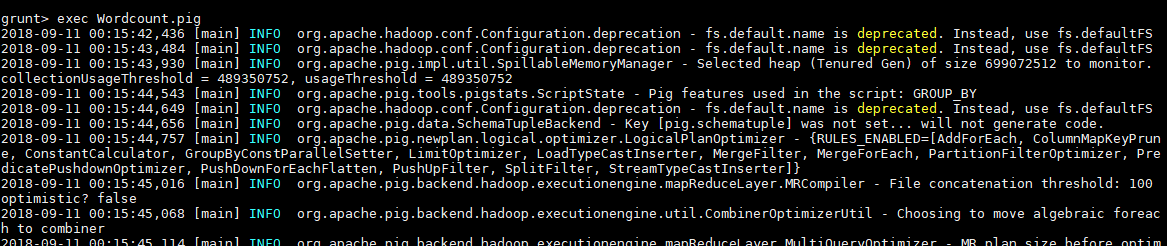
Input file:

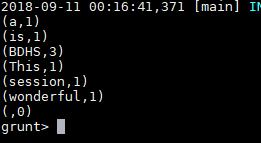


Running the program

Enter to pig grunt shell by giving ‘PIG’command

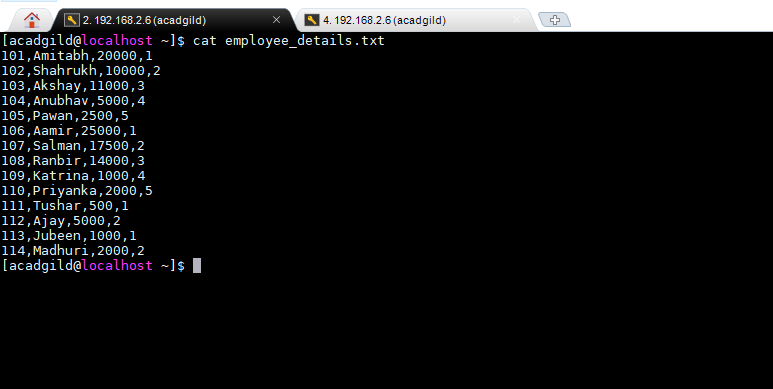
Inside the Grunt shell run the PIG script



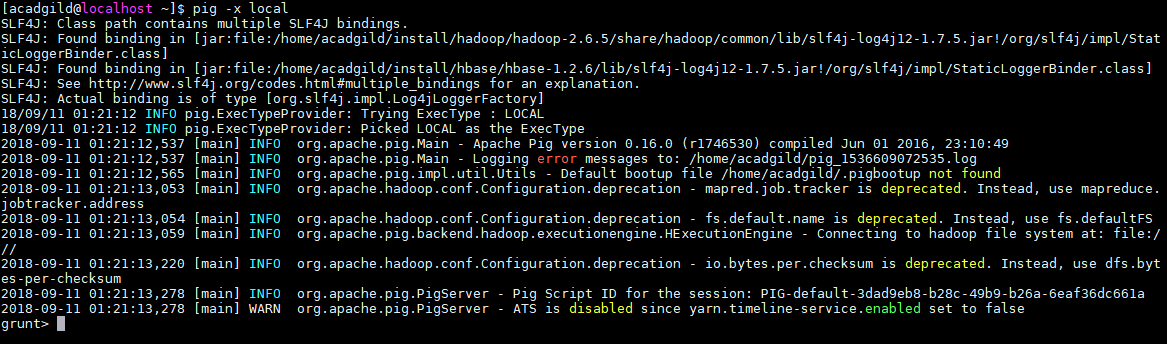


**Task 2: (a) Top 5 employees (employee id and employee name) with highest rating. (In case two employees have same rating, employee with name coming first in dictionary should get preference)**

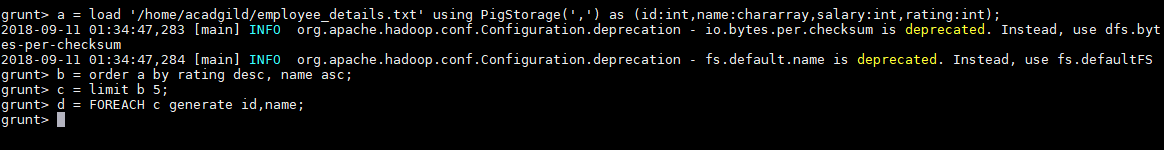
File – employee\_details.txt



Using local mode to run PIG command



Run the following commands:



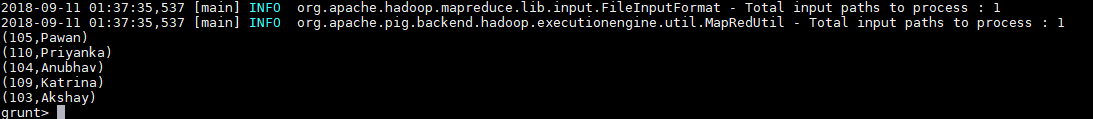
Relation a loads the employee\_details.txt

Relation b,sorts the relation a by rating in descending and name in ascending

Relation c, limits the output by 5

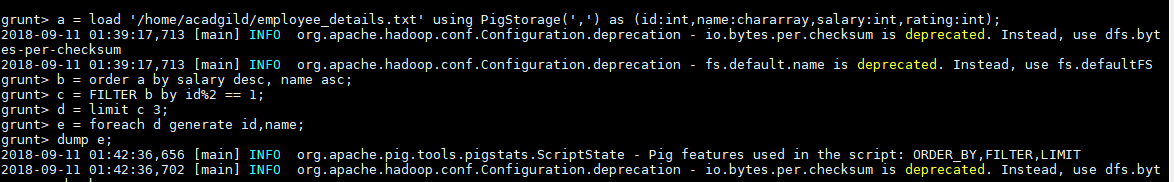
Relation d, picks only the desired columns id and name

Dump d will give the output of top 5 employees (employee id and employee name) with highest rating.



**Task 2 : (b) Top 3 employees (employee id and employee name) with highest salary, whose employee id is an odd number. (In case two employees have same salary, employee with name coming firs in dictionary should get preference)**

Run the following commands:

****

Relation a loads the employee\_details.txt

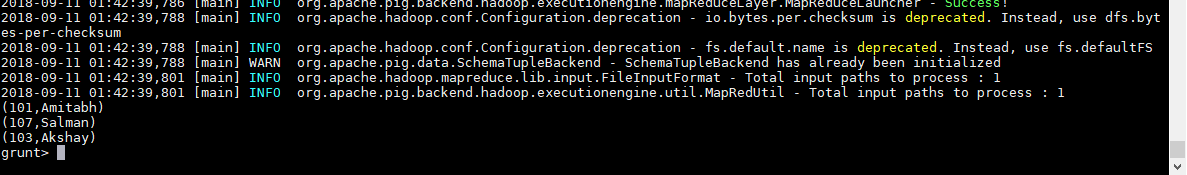
Relation b,sorts the relation a by salary in descending and name in ascending

Relation c, filters the employee id with odd number

Relation d, limits the output by 3

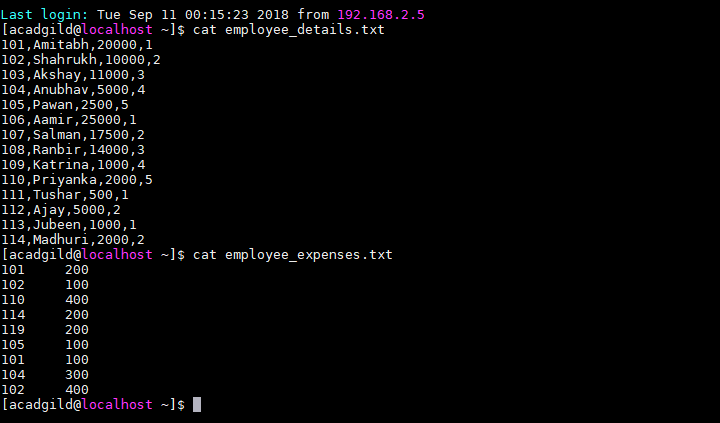
Relation e, picks only the desired columns- id and name

Dump e will give the output of top 5 employees (employee id and employee name) with highest rating.

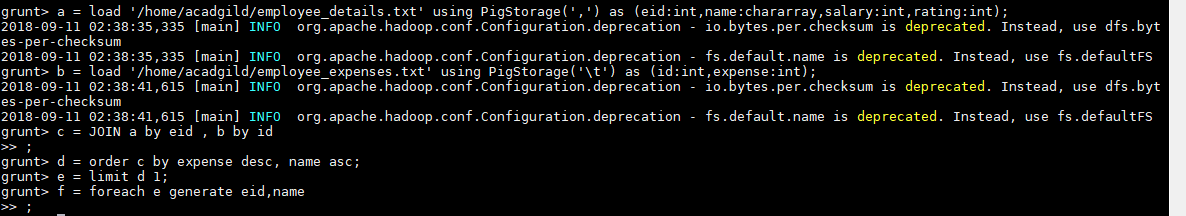


**Task 2(c) Employee (employee id and employee name) with maximum expense (In case two employees have same expense, employee with name coming first in dictionary should get preference)**

**Reading the two input files**

****

**Run the below commands**

****

Relation a : loads the employee details file

Relation b: Loads the employee expenses file

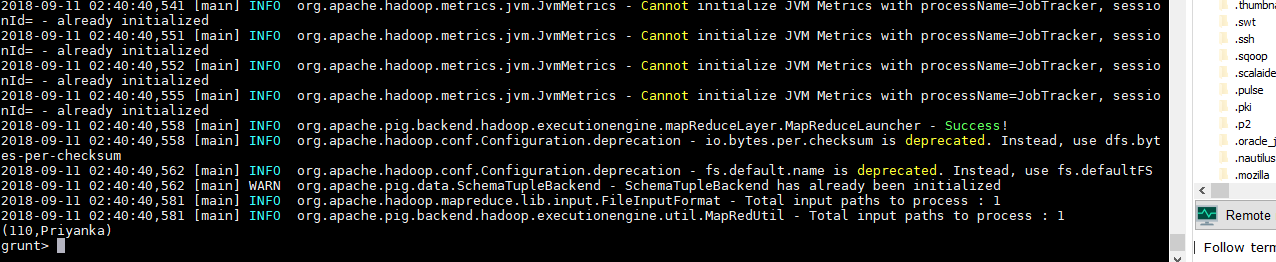
Relation c : Joins the two files on the id

Relation d: sorts the relation c on expense in descending and name in ascending

Relation e: limits the output to1 as we need only the employee with maximum expense

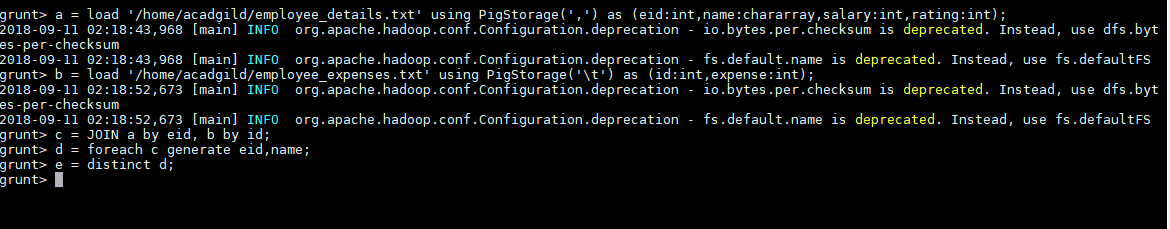
Relation f: picks the required columns(id and name)

dump f will give the output to the console.

****

**Task 2(d) List of employees (employee id and employee name) having entries in employee\_expenses file.**

**Run the below commands**

****

Relation a : loads the employee details file

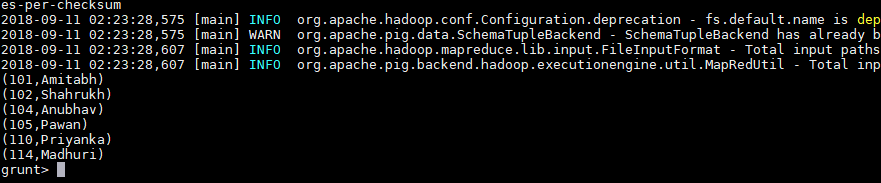
Relation b: Loads the employee expenses file

Relation C : Inner Joins the two files on the id

Relation D: picks only the required columns

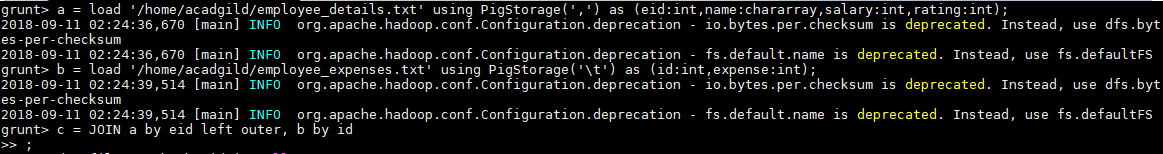
Relation E: removes the duplicates and gives the distinct set of id and name

Dump e gives the output in console



**Task (e) : List of employees (employee id and employee name) having no entry in employee\_expenses file.**

Run the below commands

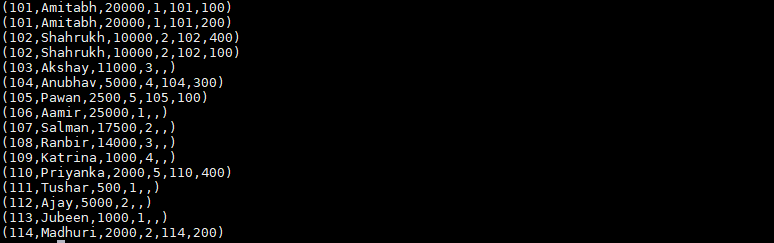
****

Relation a : loads the employee details file

Relation b: Loads the employee expenses file

Relation c : Outer Joins the two files on the id

dump c : gives a full outer join of the two tables . For the records present in first file and not in second file will be populates as null. So to extract the non matching records, we need to give the filter condition as null

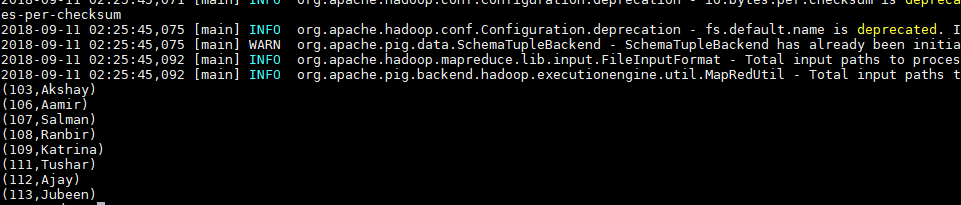
****

****

Relation d: filters the null records on the id column from relation c(with gives the non-matching records)

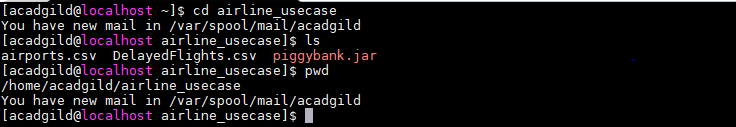
Relation e: picks only the required columns

Dump e gives the output to the console

****

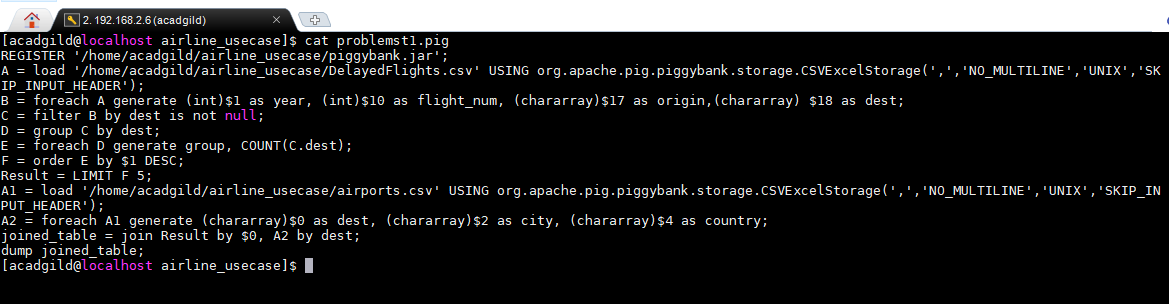
**Task 3: Implement the use case present in below blog link and share the complete steps along with screenshot(s) from your end.**

All the files required for implementing the Aviation Data Analysis use case is present in the below directory

****

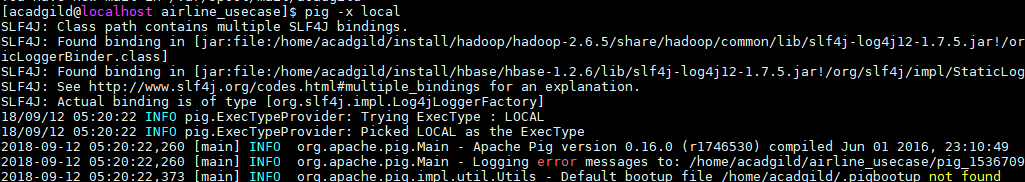
**Problem Statement 1: Find out the top 5 most visited destinations.**

Creates a pig script in the location - /home/acadgild/airline\_usecase/problemst1.pig

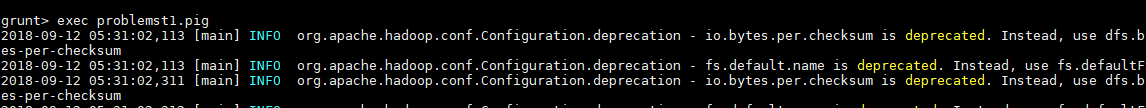


**In Line 1**: We are registering the piggybank jar in order to use the CSVExcelStorage class.  
In relation **A**, we are loading the dataset using CSVExcelStorage because of its effective technique to handle double quotes and headers.  
In relation **B**, we are generating the columns that are required for processing and explicitly typecasting each of them.  
In relation **C**, we are filtering the null values from the “dest” column.  
In relation **D**, we are grouping relation C by “dest.”  
In relation **E**, we are generating the grouped column and the count of each.  
Relation **F** and **Result** is used to order and limit the result to top 5.  
These are the steps to find the top 5 most visited destinations. However, adding few more steps in this process, we will be using another table to find the city name and country as well.  
In relation **A1**, we are loading another table to which we will look-up and find the city as well as the country.  
In relation **A2**, we are generating dest, city, and country from the previous relation.  
In relation **joined\_table**, we are joining Result and A2 based on a common column, i.e., “dest”  
Finally, using dump, we are printing the result.

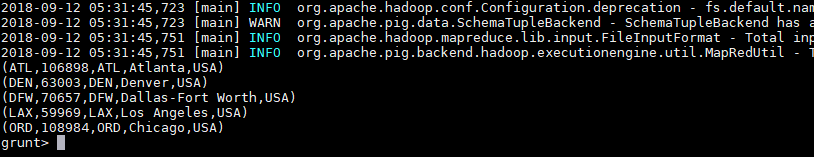
Entering to local shell for pig



Executing the pig script

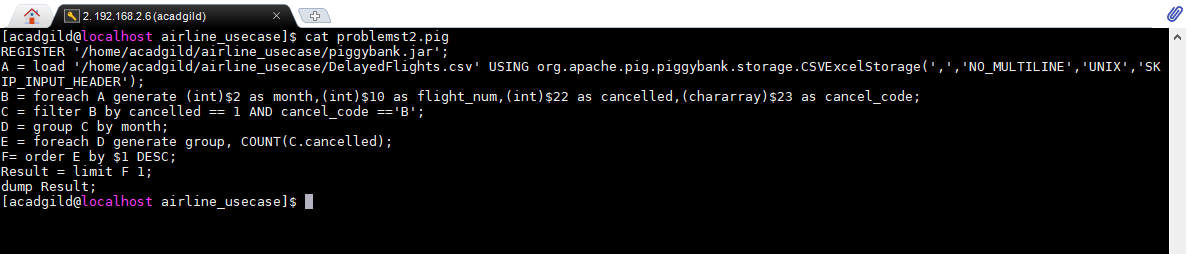


Output to the console



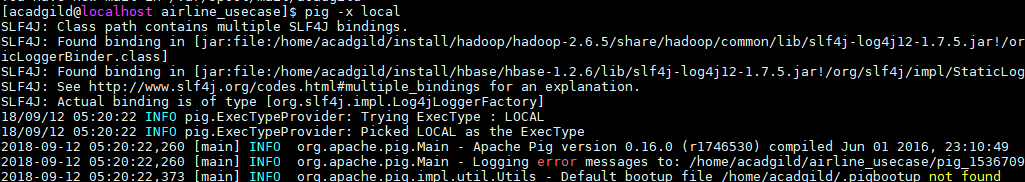
**Problem Statement 2:** Which month has seen the most number of cancellations due to bad weather?

Creates a pig script in the location /home/acadgild/airline\_usecase/problemst2.pig

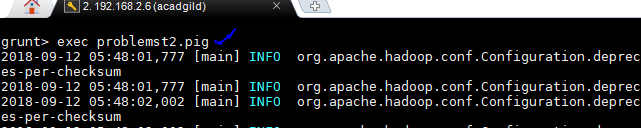


**In Line 1**: We are registering piggybank jar in order to use the CSVExcelStorage class.  
In relation **A**, we are loading the dataset using CSVExcelStorage because of its effective technique to handle double quotes and header.  
In relation **B**, we are generating the columns which are required for processing and explicitly typecasting each of them.  
In relation **C**, we are filtering the data based on cancellation and cancellation code, i.e.,  canceled = 1 means flight have been canceled and cancel\_code = ‘B’ means the reason for cancellation is “weather.” So relation C will point to the data which consists of canceled flights due to bad weather.  
In relation**D**, we are grouping the relation C based on every month.  
In relation **E**, we are finding the count of canceled flights every month.  
Relation **F** and **Result** is for ordering and finding the top month based on cancellation.

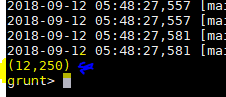
Entering to local shell for pig



Executing the pig script

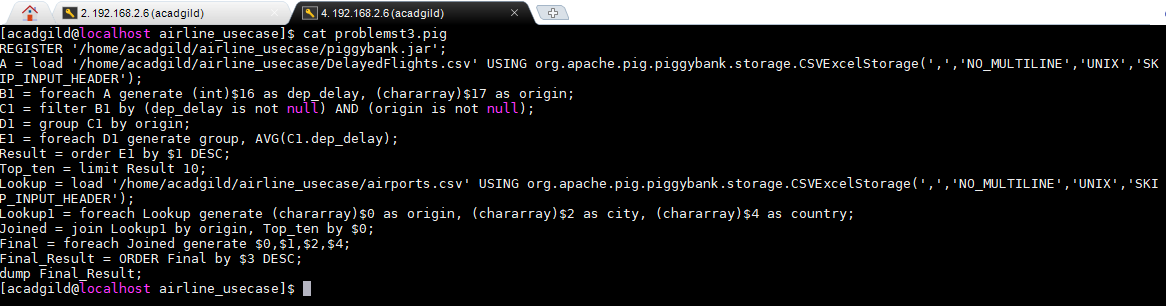


Output to the console

****

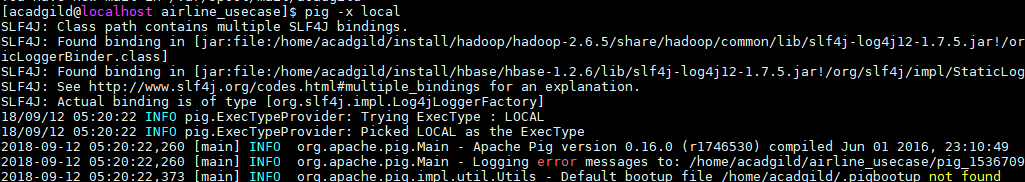
**Problem Statement 3:** Top ten origins with the highest AVG departure delay

Creates a pig script in the location /home/acadgild/airline\_usecase/problemst3.pig

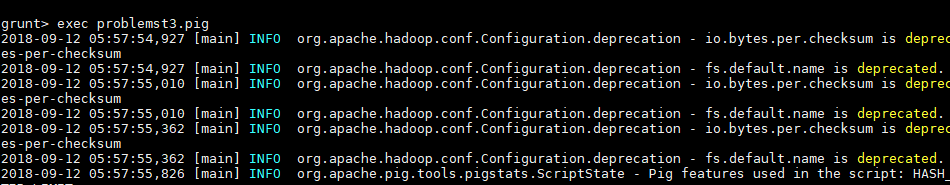
****

Explanation of first 3 lines are the same as explained in the previous 2 problem statements.  
In relation**C1**, we are removing the null values fields present if any.  
In relation **D1**, we are grouping the data based on column “origin.”  
In relation **E1**, we are finding average delay from each unique origin.  
Relations named **Result** and **Top\_ten** are ordering the results in descending order and printing the top ten values.  
These steps are good enough to find the top ten origins with the highest average departure delay.  
However, rather than generating just the code of origin, we will be following a few more steps to find some more details like country and city.  
In the relation **Lookup**, we are loading another table to which we will look up and find the city as well as the country.  
In the relation L**ookup1,** we are generating the destination, city, and country from the previous relation.  
In the relation **Joined**, we are joining relation Top\_ten and Lookup1 based on common a column, i.e., “origin.”  
In the relation**Final,**we are generating required columns from the Joined table.  
Finally, we are ordering and printing the results.

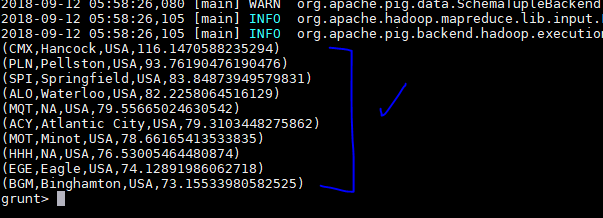
Entering to local shell for pig



Executing the pig script

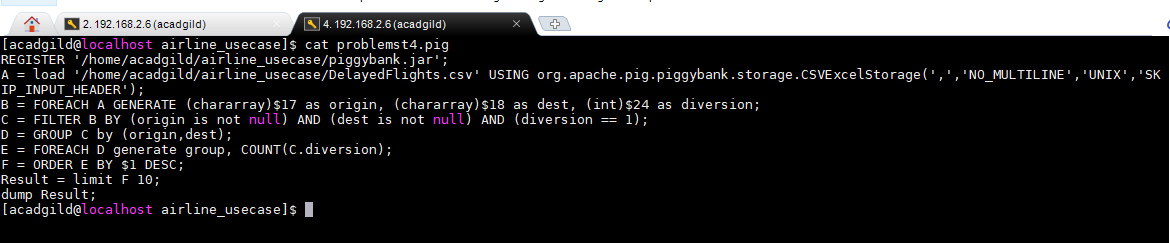


Output to the console

****

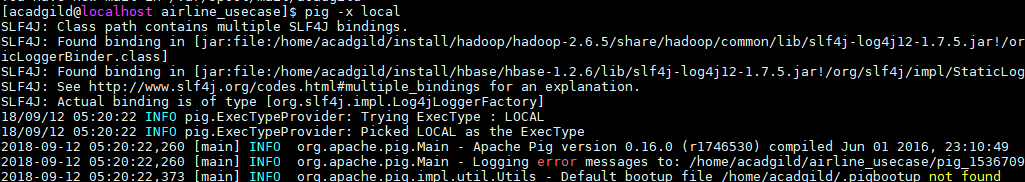
**Problem Statement 4:** Which route (origin & destination) has seen the maximum diversion?

Creates a pig script in the location /home/acadgild/airline\_usecase/problemst4.pig

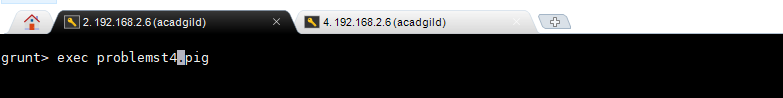
****

**In Line 1**: We are registering piggybank jar in order to use CSVExcelStorage class.  
In relation **A**, we are loading the dataset using CSVExcelStorage because of its effective technique to handle double quotes and headers.  
In relation **B**, we are generating the columns which are required for processing and explicitly type-casting each of them.  
In relation **C**, we are filtering the data based on “not null” and diversion =1. This will remove the null records, if any, and give the data corresponding to the diversion taken.  
In relation **D**, we are grouping the data based on origin and destination.  
Relation **D** finds the count of diversion taken per unique origin and destination.  
Relations **F** and **Result** orders the result and produces top 10 results.

Entering to local shell for pig



Executing the pig script

****

Output to console

