**Ritika Kumari- A20414073**

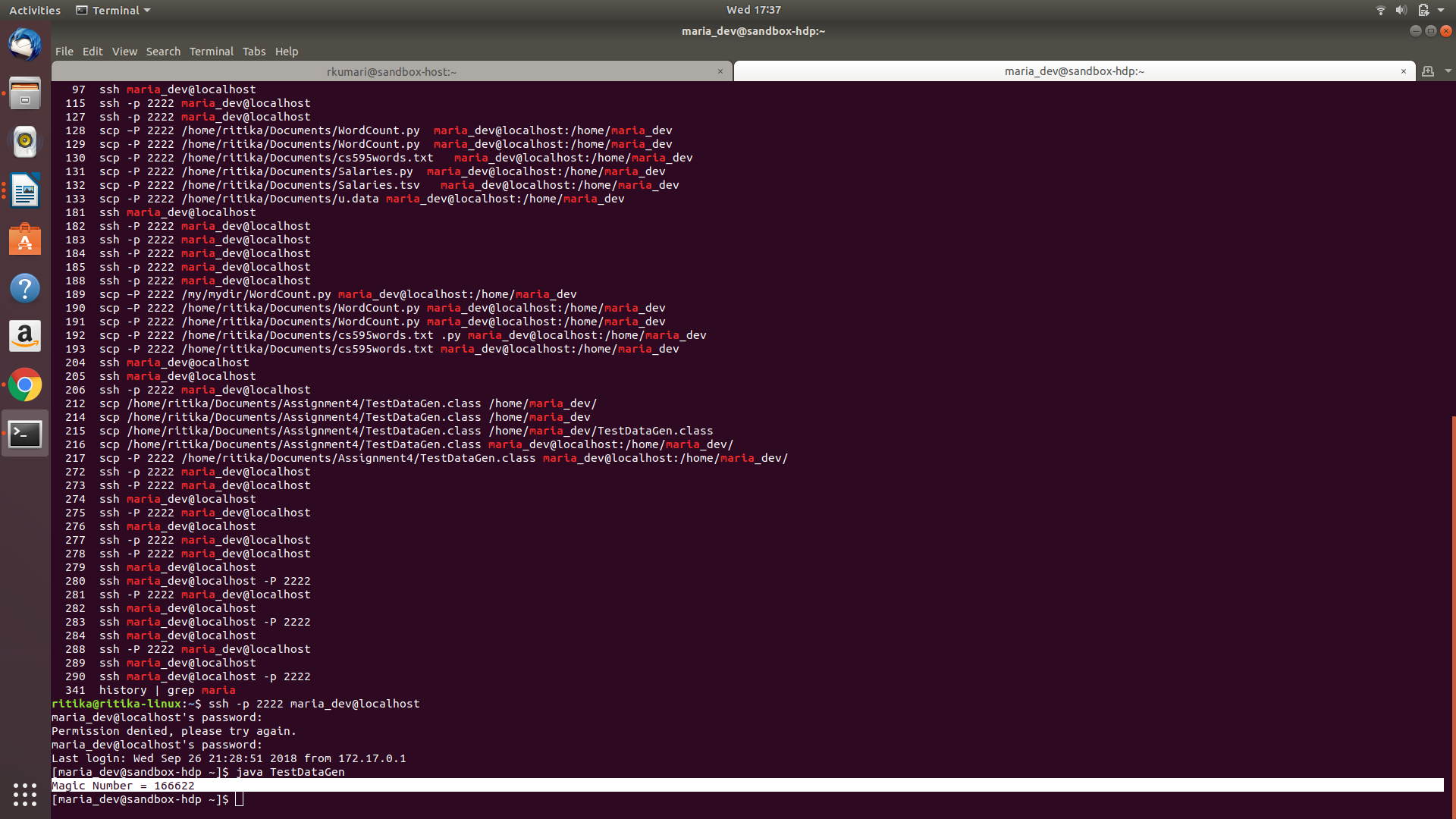
# **CSP554—Big Data Technologies**

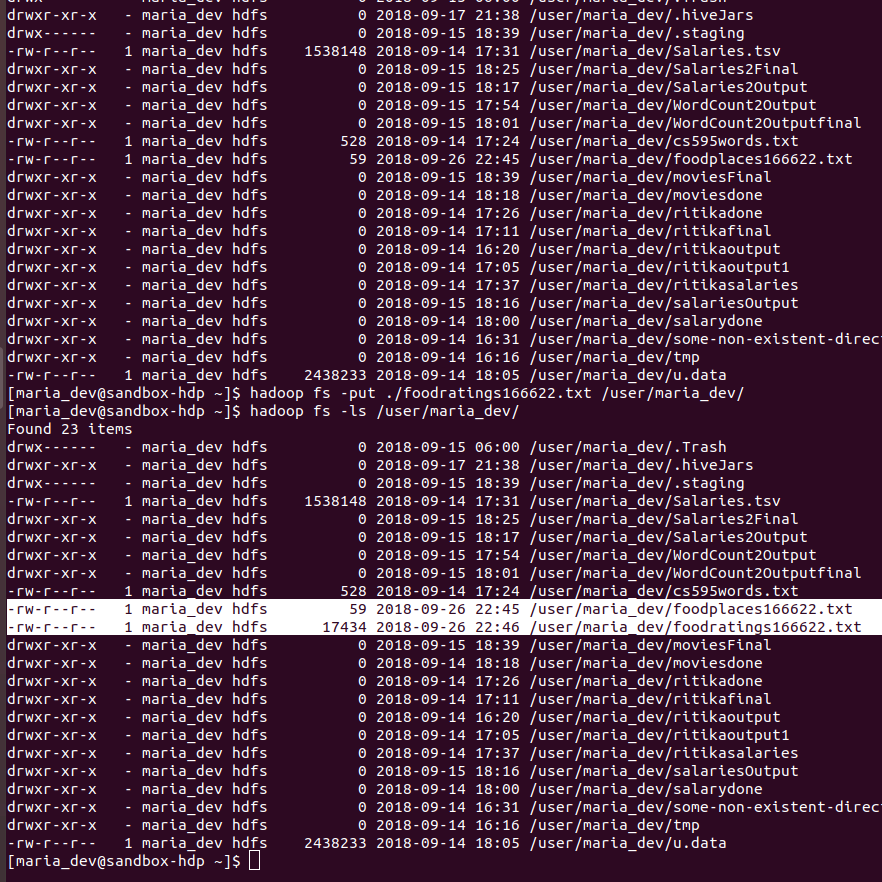
## **Assignment #5 (Modules 05)**

**Recall that the files generated by TestDataGen have comma separated fields.**

Exercise 1)

Create new versions of the foodratings and foodplaces files by using TestDataGen (as described in assignment #4) and copy them to HDFS





Write and execute a sequence of pig latin statements that loads the foodratings file as a relation. Call the relation ‘food\_ratings’. The load command should associate a schema with this relation where the first attribute is referred to as ‘name’ and is of type chararray, the next attributes are referred to as ‘f1’ through ‘f4’ and are of type int, and the last field is refered to a ‘placeid’ and is also of type int.

Execute the describe command on this relation.

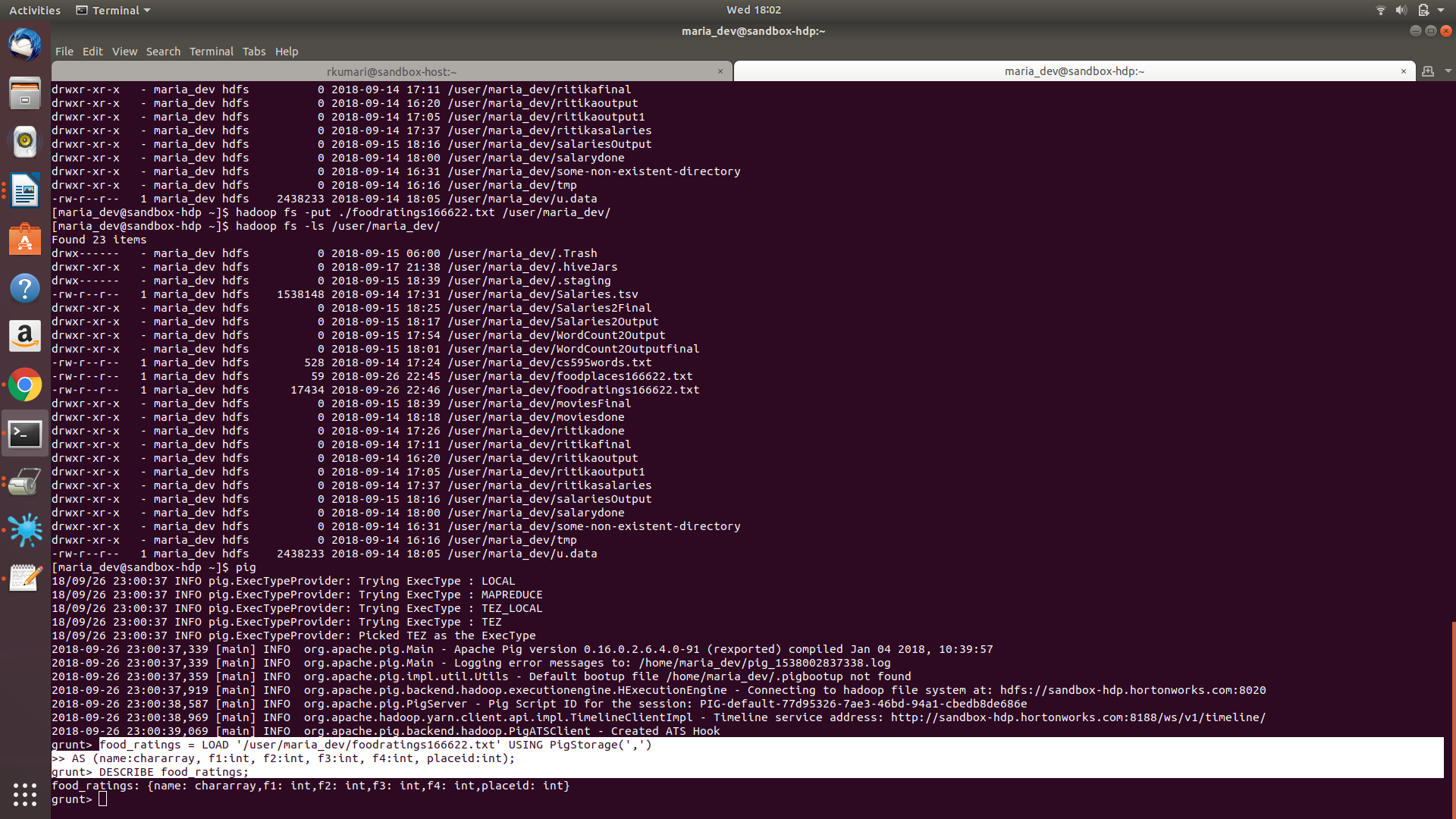
Provide the magic number, the load command you wrote and the output of the describe command as the result of this exercise.

**Magic number: 166622**

**Command:** food\_ratings = LOAD '/user/maria\_dev/foodratings166622.txt' USING PigStorage(',')

AS (name:chararray, f1:int, f2:int, f3:int, f4:int, placeid:int);

DESCRIBE food\_ratings;



Exercise 2)

Now create another relation with two fields of the initial (food\_ratings) relation: ‘name’ and ‘f4’. Call this relation ‘food\_ratings\_subset’.

Store this last relation back to HDFS.

Also write 6 records of this relation out to the console.

Submit the pig latin statements you used and the six records printed out to the console as the result of this exercise.

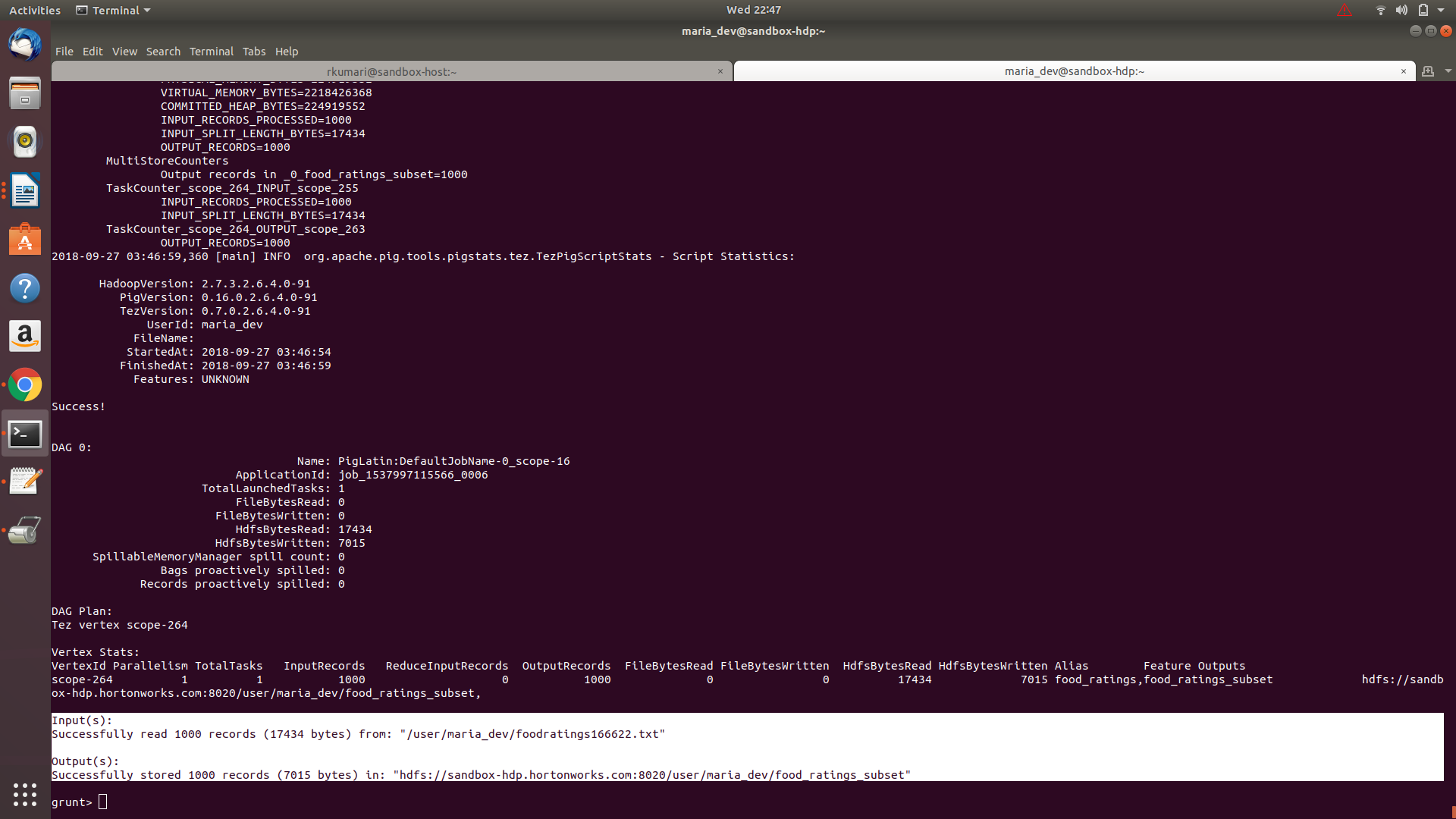
**Magic number: 166622**

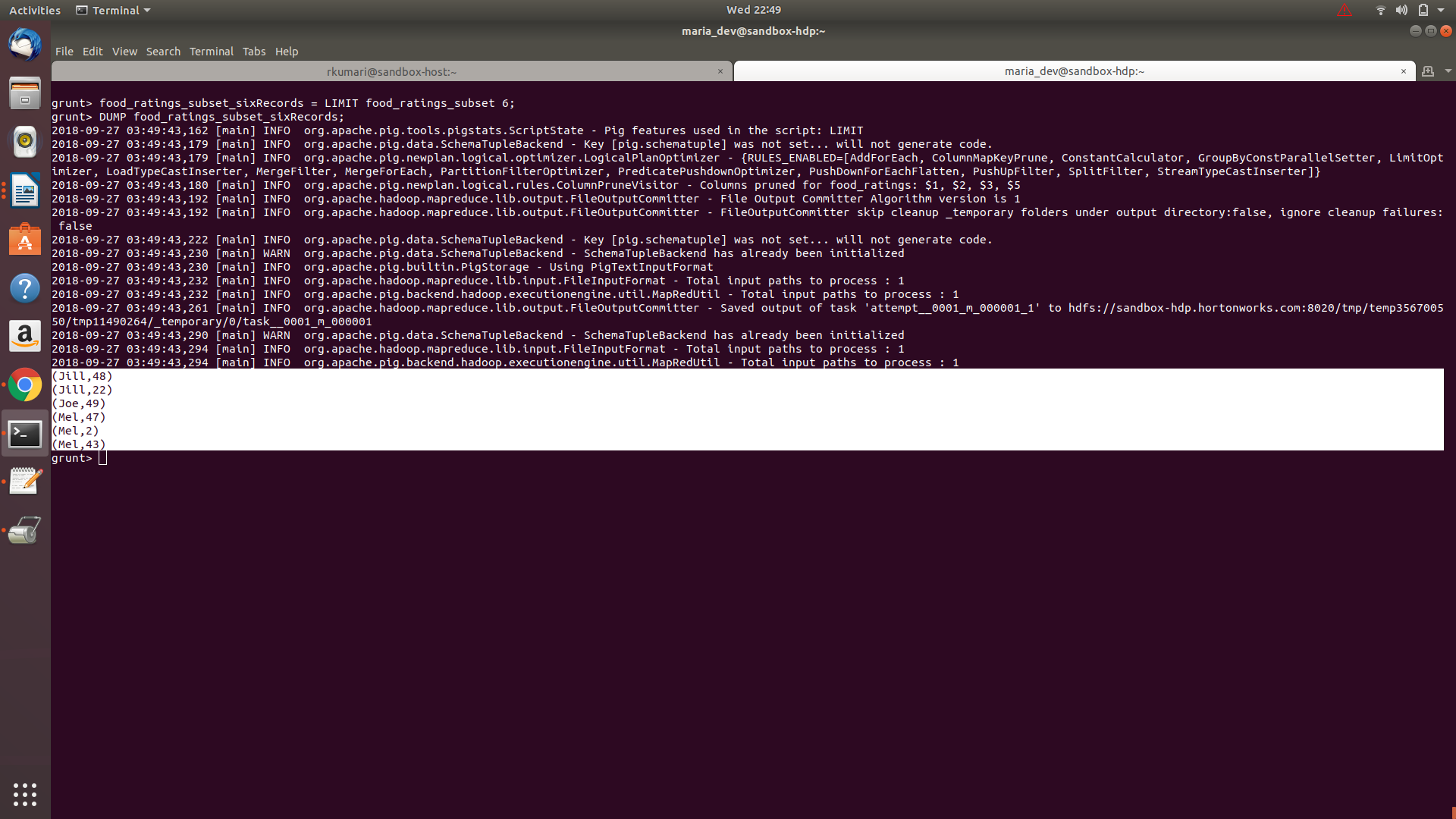
**Command:** food\_ratings\_subset = FOREACH food\_ratings GENERATE name, f4;

STORE food\_ratings\_subset INTO 'food\_ratings\_subset' USING PigStorage ('|');

food\_ratings\_subset\_sixRecords = LIMIT food\_ratings\_subset 6;

DUMP food\_ratings\_subset\_sixRecords;





Exercise 3)

Now create another relation using the initial (food\_ratings) relation. Call this relation ‘food\_ratings\_profile’. The new relation should only have one record. This record should hold the minimum, maximum and average values for the attributes ‘f2’ and ‘f3’. (So this one record will have 6 fields).

Write the record of this relation out to the console.

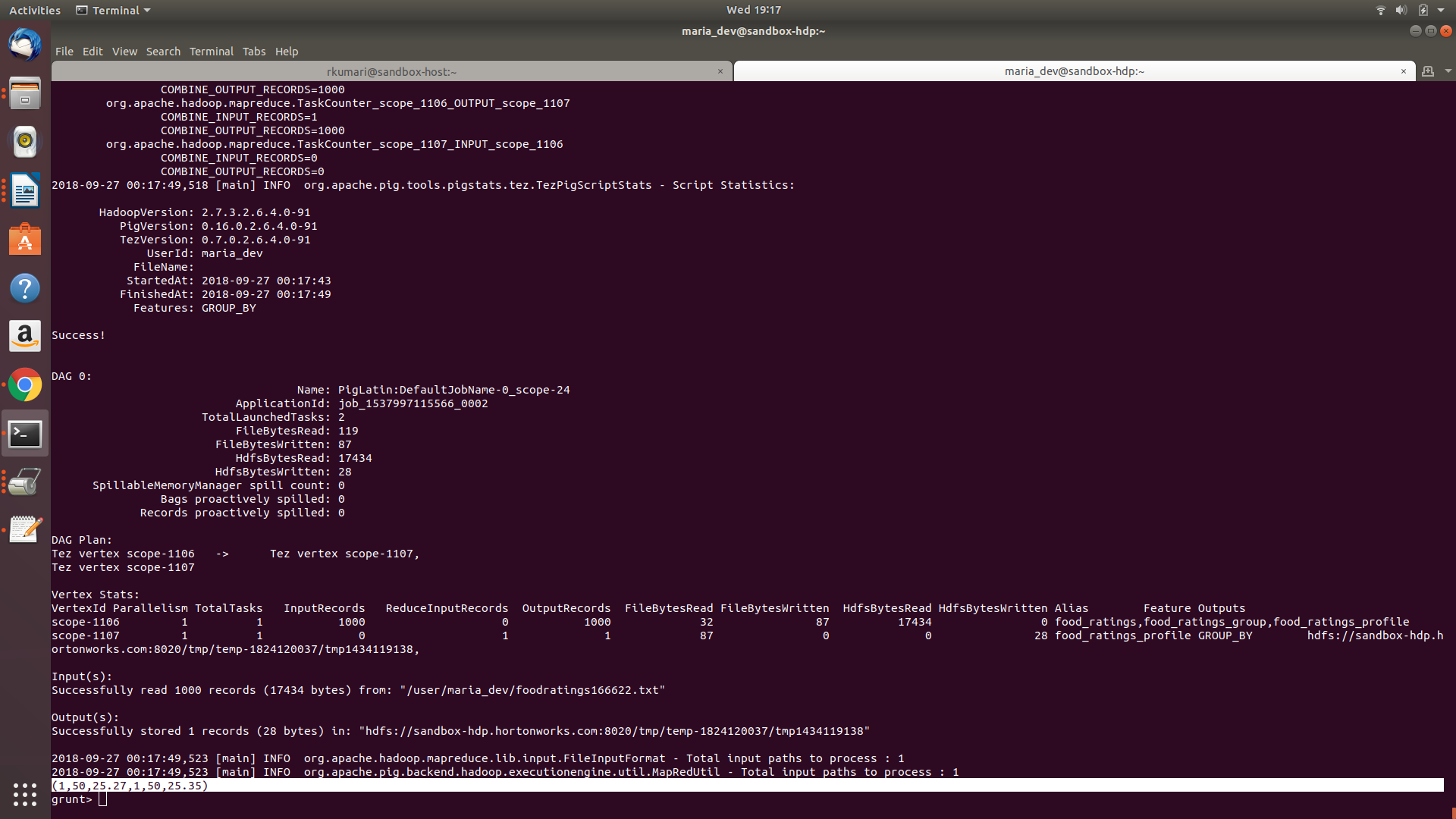
Submit the pig latin statements you used and the record printed out to the console as the result of this exercise.

**Command:** food\_ratings\_all = GROUP food\_ratings ALL;

food\_ratings\_profile = FOREACH food\_ratings\_group GENERATE MIN(food\_ratings.f2) as F2\_MINIMUM, MAX(food\_ratings.f2) as F2\_MAXIMUM, AVG(food\_ratings.f2) as F2\_AVGERGE, MIN(food\_ratings.f3) as F3\_MINIMUM, MAX(food\_ratings.f3) as F3\_MAXIMUM, AVG(food\_ratings.f3) as F3\_AVGERAGE;

DESCRIBE food\_ratings\_profile;

DUMP food\_ratings\_profile;



Exercise 4)

Now create yet another relation from the initial (food\_ratings) relation. This new relation should only include tuples (records) where f1 < 20 and f3 > 5. Call this relation ‘food\_ratings\_filtered’.

Write 6 records of this relation out to the console.

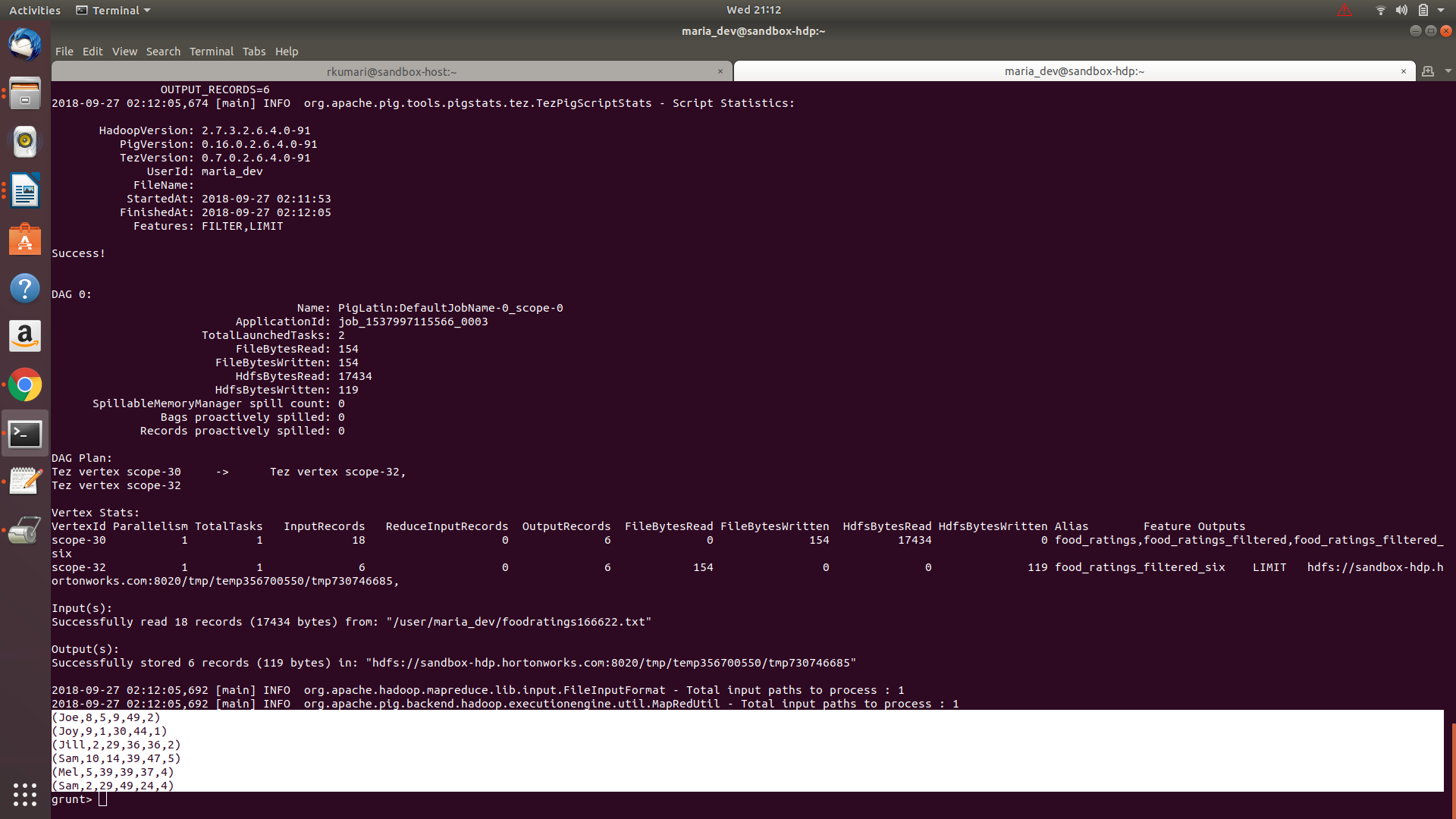
Submit the pig latin statements you used and the six records printed out to the console as the result of this exercise.

**Command:** food\_ratings\_filtered = FILTER food\_ratings BY (f1 < 20) AND (f3 > 5);

food\_ratings\_filtered\_six = LIMIT food\_ratings\_filtered 6;

DESCRIBE food\_ratings\_filtered\_six;

DUMP food\_ratings\_filtered\_six;



Exercise 5)

Using the initial (food\_ratings) relation, write and execute a sequence of pig latin statements that creates another relation, call it ‘food\_ratings\_2percent’, holding a random selection of 2% of the records in the initial relation.

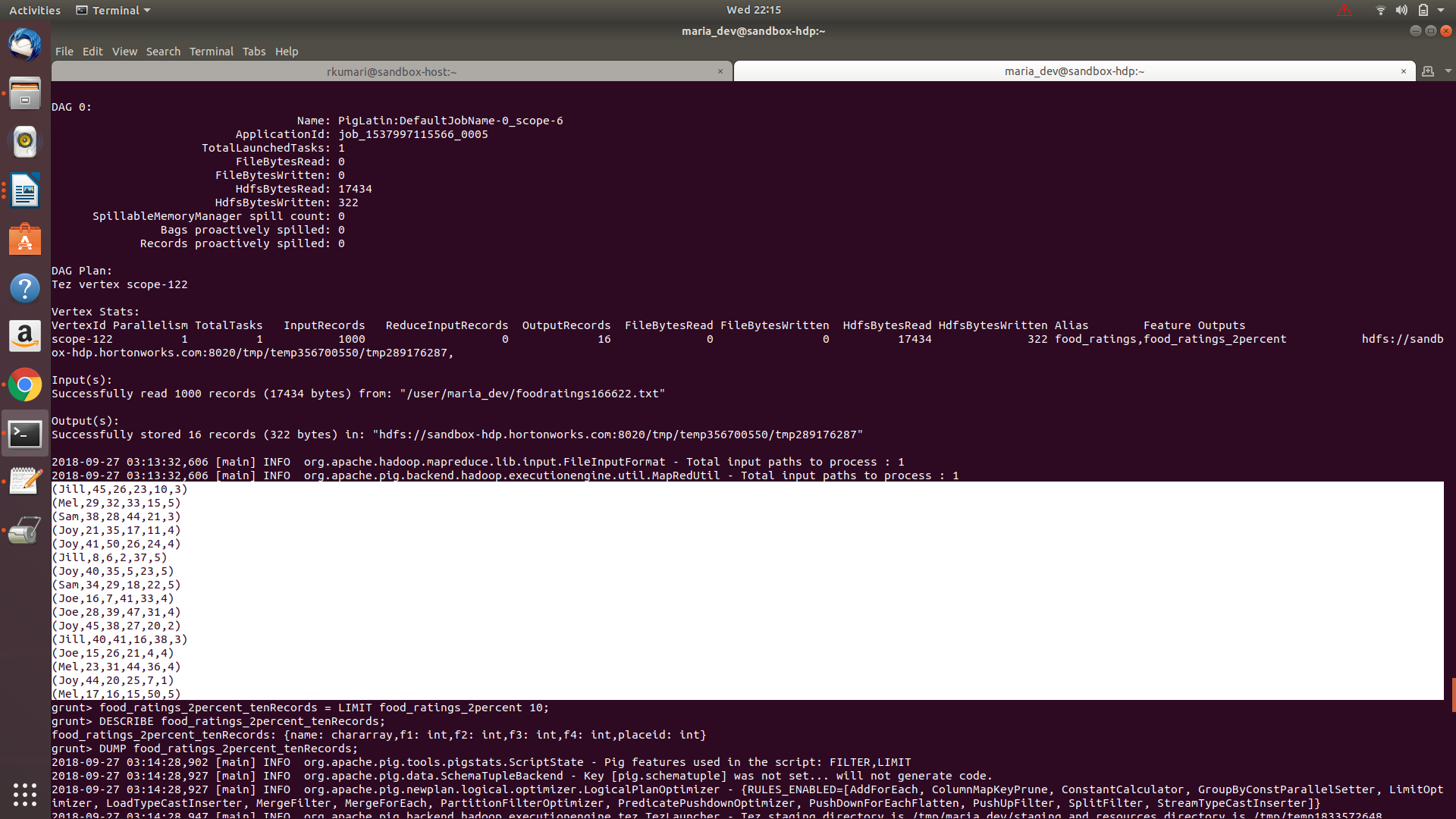
Write 10 of the records out to the console.

Submit the pig latin statements and the records printed out to the console.

**Command:** food\_ratings\_2percent = SAMPLE food\_ratings 0.02;

DESCRIBE food\_ratings\_2percent;

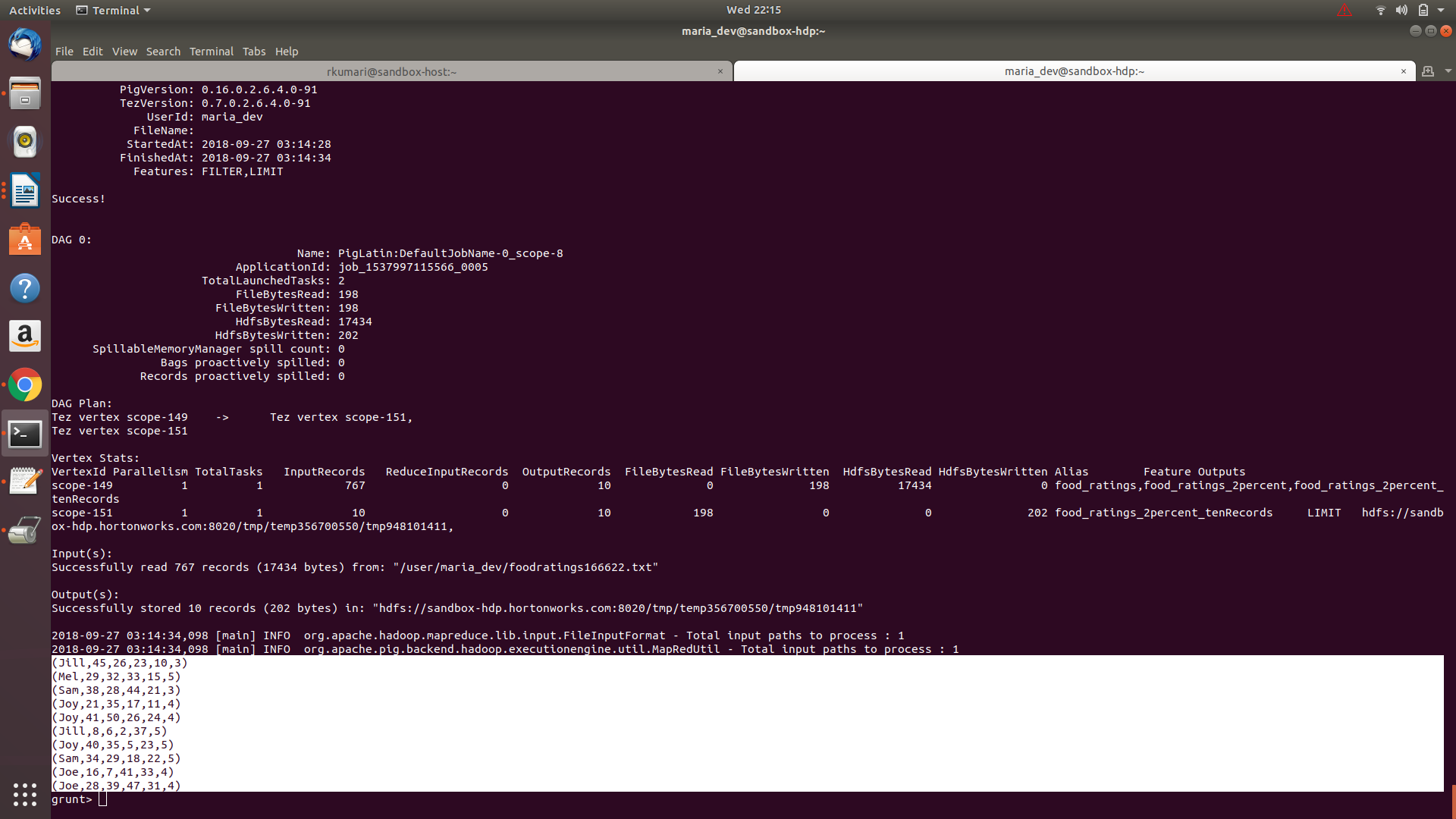
DUMP food\_ratings\_2percent;



**Command:** food\_ratings\_2percent\_tenRecords = LIMIT food\_ratings\_2percent 10;

DESCRIBE food\_ratings\_2percent\_tenRecords;

DUMP food\_ratings\_2percent\_tenRecords;



Exercise 6)

Write and execute a sequence of pig latin statements that loads the foodplaces file as a relation. Call the relation ‘food\_places’. The load command should associate a schema with this relation where the first attribute is referred to as ‘placeid’ and is of type int and the second attribute is referred to as ‘placename’ and is of type chararray.

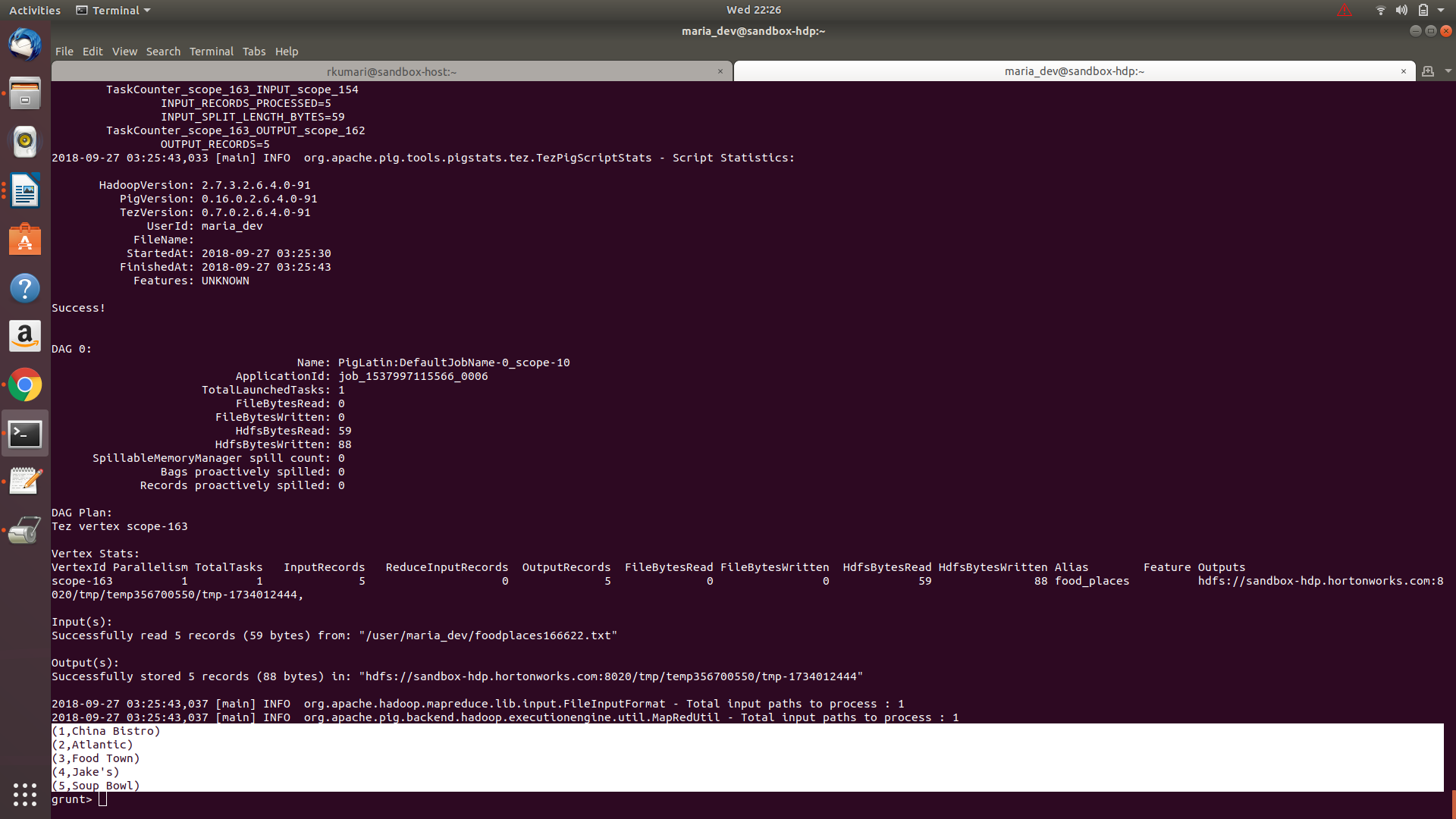
Execute the describe command on this relation.

**Command:** food\_places = LOAD '/user/maria\_dev/foodplaces166622.txt' USING PigStorage(',')

AS (placeid:int, placename:chararray);

DESCRIBE food\_places;

DUMP food\_places;



Now perform a join between the initial place\_ratings relation and the food\_places relation on the placeid attributes to create a new relation called ‘food\_ratings\_w\_place\_names’. This new relation should have all the attributes (columns) of both relations. The new relation will allow us to work with place ratings and place names together.

Write 6 records of this relation out to the console.

Submit the pig latin statements you used and the six records printed out to the console as the result of this exercise.

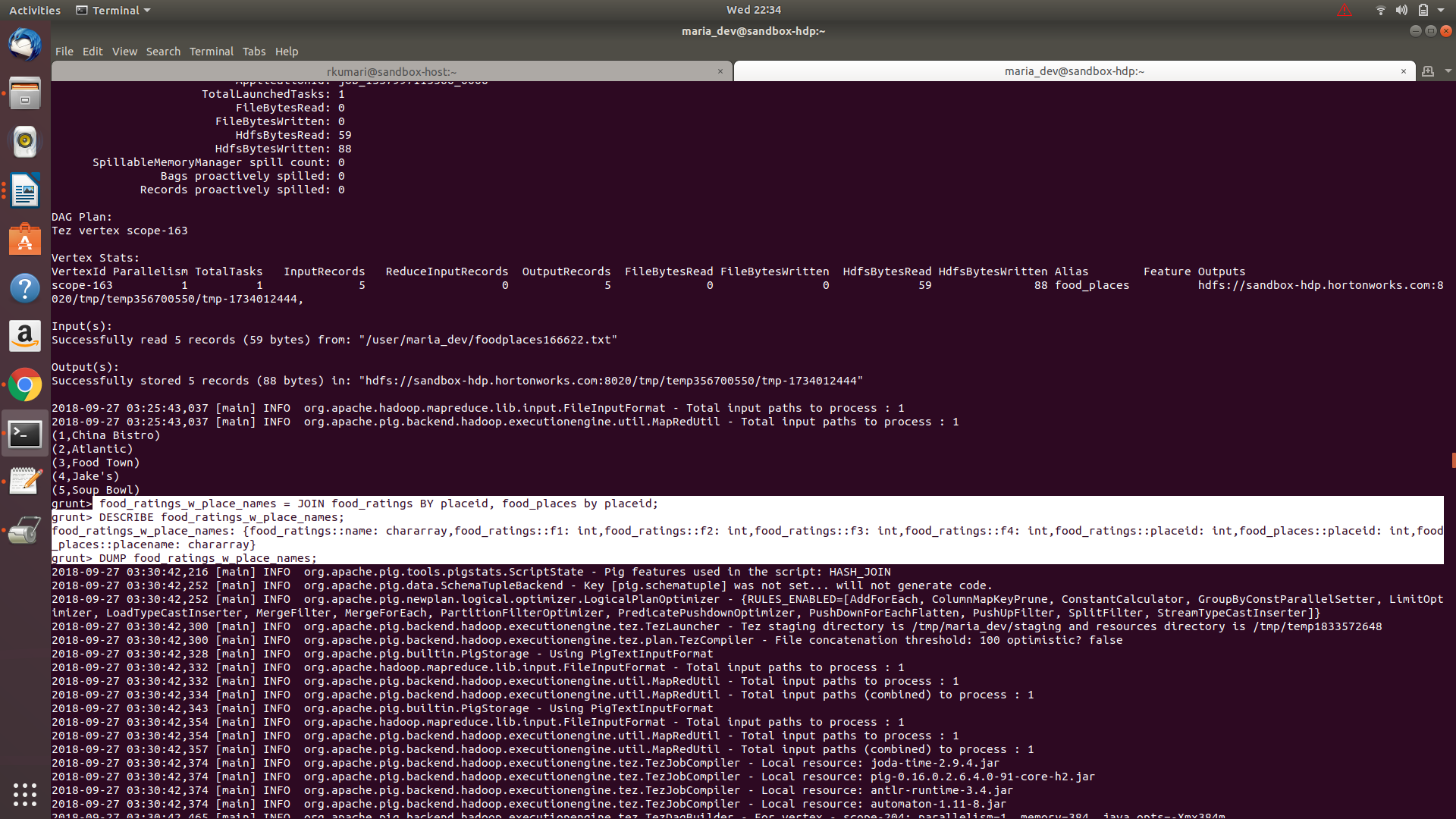
**Command:** food\_ratings\_w\_place\_names = JOIN food\_ratings BY placeid, food\_places by placeid;

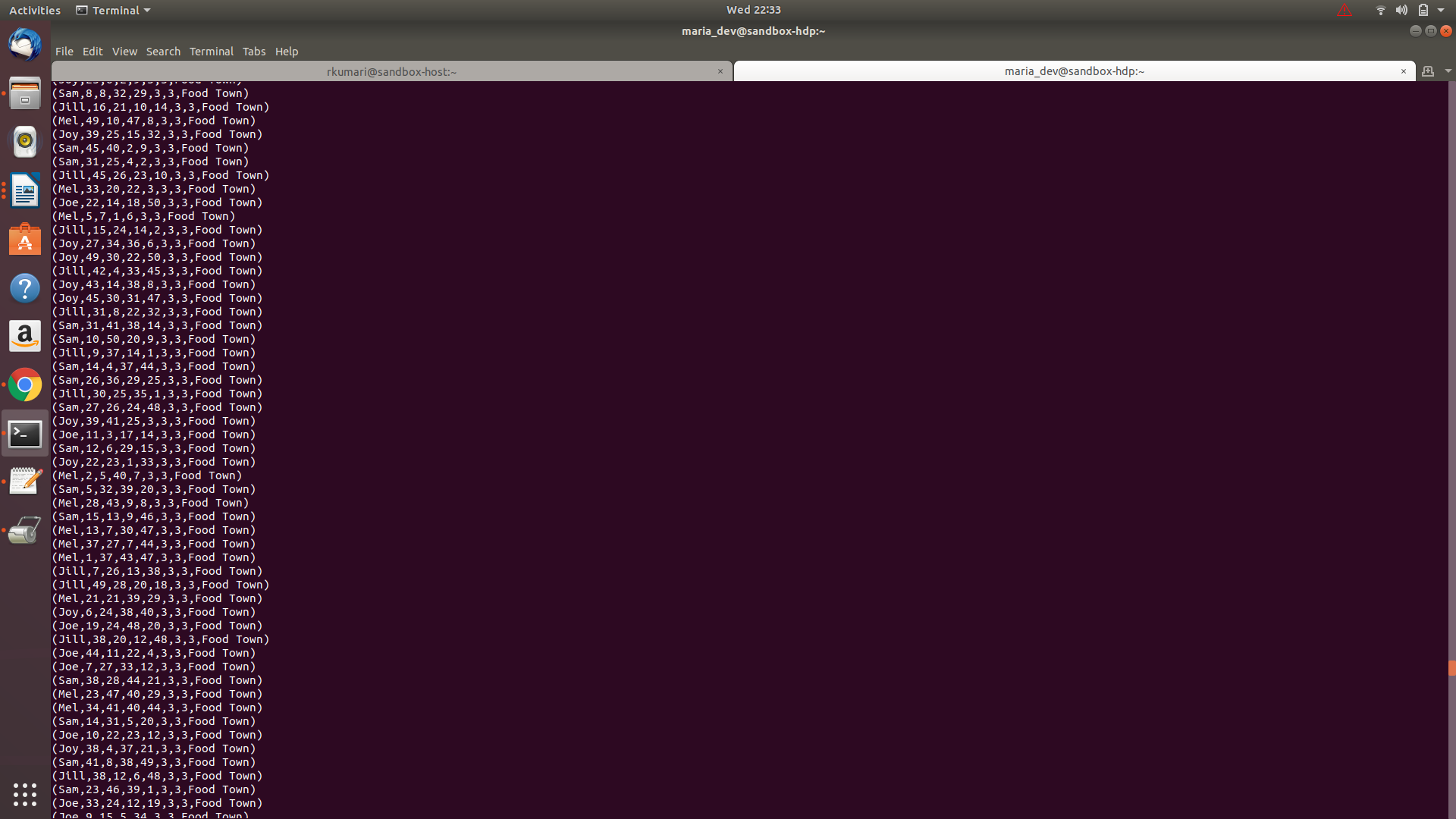
DESCRIBE food\_ratings\_w\_place\_names;

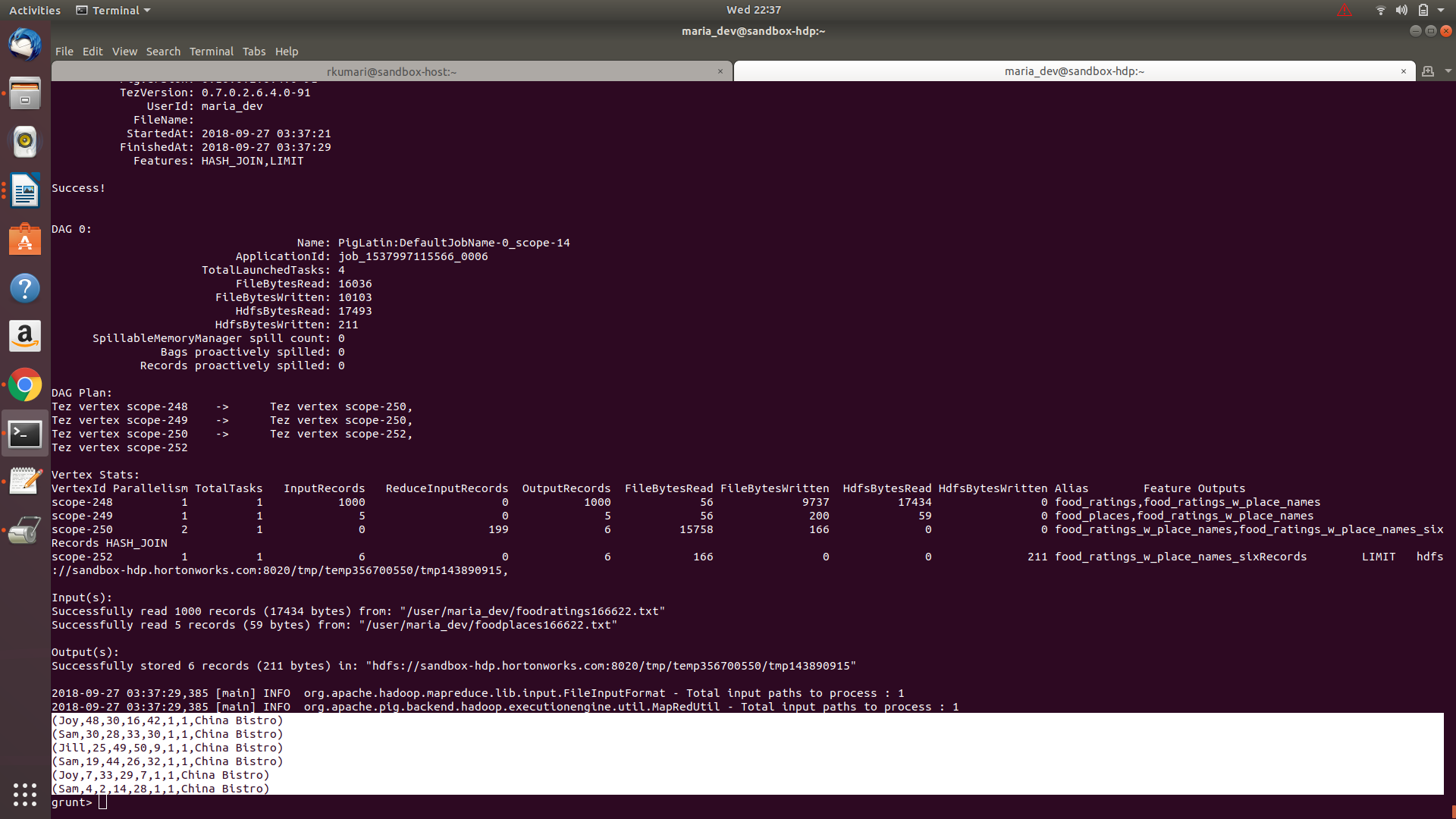
food\_ratings\_w\_place\_names\_sixRecords= LIMIT food\_ratings\_w\_place\_names 6;

DESCRIBE food\_ratings\_w\_place\_names\_sixRecords;

DUMP food\_ratings\_w\_place\_names\_sixRecords;







Read the article about Spark available on the blackboard in the ‘Articles’ section:

“Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing”

Provide a half-page summary and include some of your own comments.

Answer: The article describes the architecture of Resilient Distributed Datasets, the problems they can be used to solve, how they perform on different models and how they are different from existing solutions. Many common clusters computing frameworks lack in two fields: 1. Iterative algorithms 2. Interactive data analysis where users run ad-hoc queries on the data.

One way around these problems is to use specialized frameworks like Pregel. But this leads to loss of generality. RDDs are immutable partitioned collections that are created through deterministic operations on data in stable storage or other RDDs. They keep enough information about how they are derived from other sources called lineage. This lineage ensures that RDDs can be easily rebuilt in case of failures without having to perform explicit checkpointing.

RDDs provide an interface based on fine-grained reads and coarse-grained update transformation. These transformations can be logged to build lineage graph to provide fault tolerance. But this update nature makes RDDs unsuitable for applications like incremental web crawler that needs asynchronous fine-grained updates to a shared state. In such cases, Distributed Shared Memory (DSM) would be a better choice since it provides fine-grained reads and writes. Although RDDs offer many advantages over DSM.

The most interesting aspect of this representation is how dependencies are expressed. Dependencies belong to one of the two classes: Narrow Dependencies   where each partition of the parent node is used by at most one child partition and Wide Dependencies — where multiple child partitions use a single parent partition.

RDDs have been implemented in Spark to provide a language integrated API. The article proposes a graph-based representation of RDDs where an RDD is expressed through a common interface that exposes five functions: partition, dependencies, iterator, partitioner, and preferredLocation. RDDs can be used to express many existing models like MapReduce, Dryad LINQ, Pregel, Batched Stream Processing, etc. This seems surprising given that RDDs offer only a limited interface due to their immutable nature and coarse-grained transformations. But these limitations have a negligible impact on many parallel applications.