CSP554—Big Data Technologies

Assignment #4 A20512400 Pradaap Shiva Kumar Shobha

Worth: 18 points

You must test your exercises against randomly generated data files. You will generate these files using the TestDataGen program. To do so please follow the below steps:

- Download the file TestDataGen.class from the Blackboard. It is one of the attachments to the assignment.
- scp the file over to the home directory (/home/hadoop) on your Hadoop VM
- Log on to your VM using ssh and execute the file using "java TestDataGen"
- This will output a magic number which you should copy down and provide with the results of your assignment.
- It will also place the files foodratings<magic number>.txt and foodplaces<magic number>.txt in your VM home directory
- Use them for your exercises
- Note, each time you execute the TestDataGen program it create new files of test data so you can exercise your program using different combinations of data. Make sure to send the magic number of your final test data set. Also, this means that every student will have different data for their assignment.

The foodratings<magic number>.txt file has six comma separated fields. The first field is the name of a food critic. The second through fifth fields are the ratings each critic gives to four food types at each restaurant they review. The ratings are an integer from 1 through 50. The sixth field is the id of the restaurant.

The foodplaces<.magic number>.txt file has two comma separated fields. The first field is the id of a restaurant. The second field is the name of that restaurant.

```
Magic Number: 227448
```

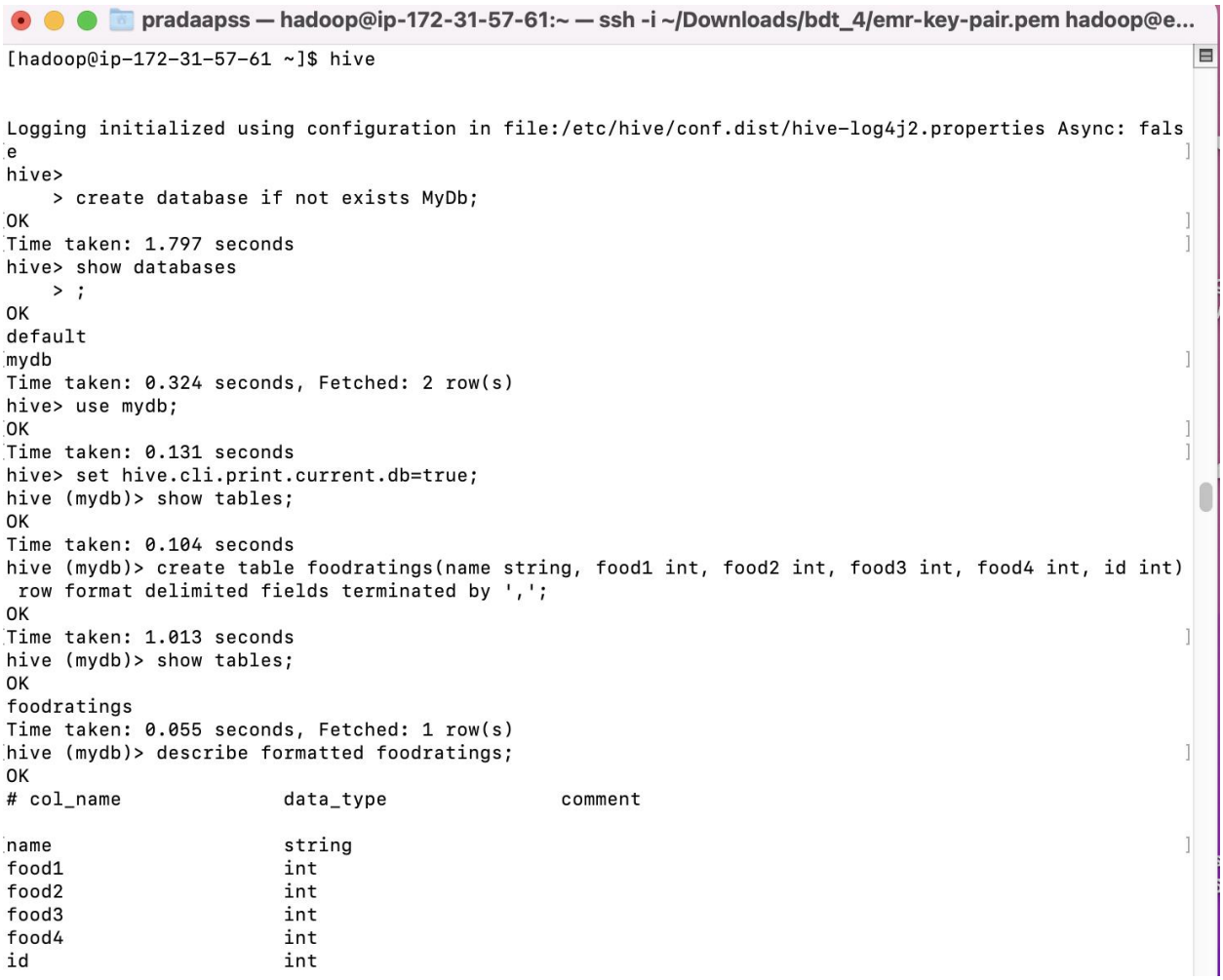
Exercise 1) 2 points

Create a Hive database called “MyDb”.

Note, after you do this the default database is still 'default." So unless you do something specific about this, if you create a table without qualifying it as belonging to MyDb (MyDb.sometable), it is created in the 'default' database. You can change the default database via a hive command. Try to discover which one and execute it now. Or when you create and use a table you must always qualify its name with the name of the database you created.

Now in MyDb create a table with name foodratings having six columns with the name of the first ‘name’ and the type of the first a string and the names of the remaining columns food1, food2, food3, food4 and id and indicate their types each as an integer. The table should have storage format TEXTFILE and column separator a “,”. That is the underlying format should be a CSV file.

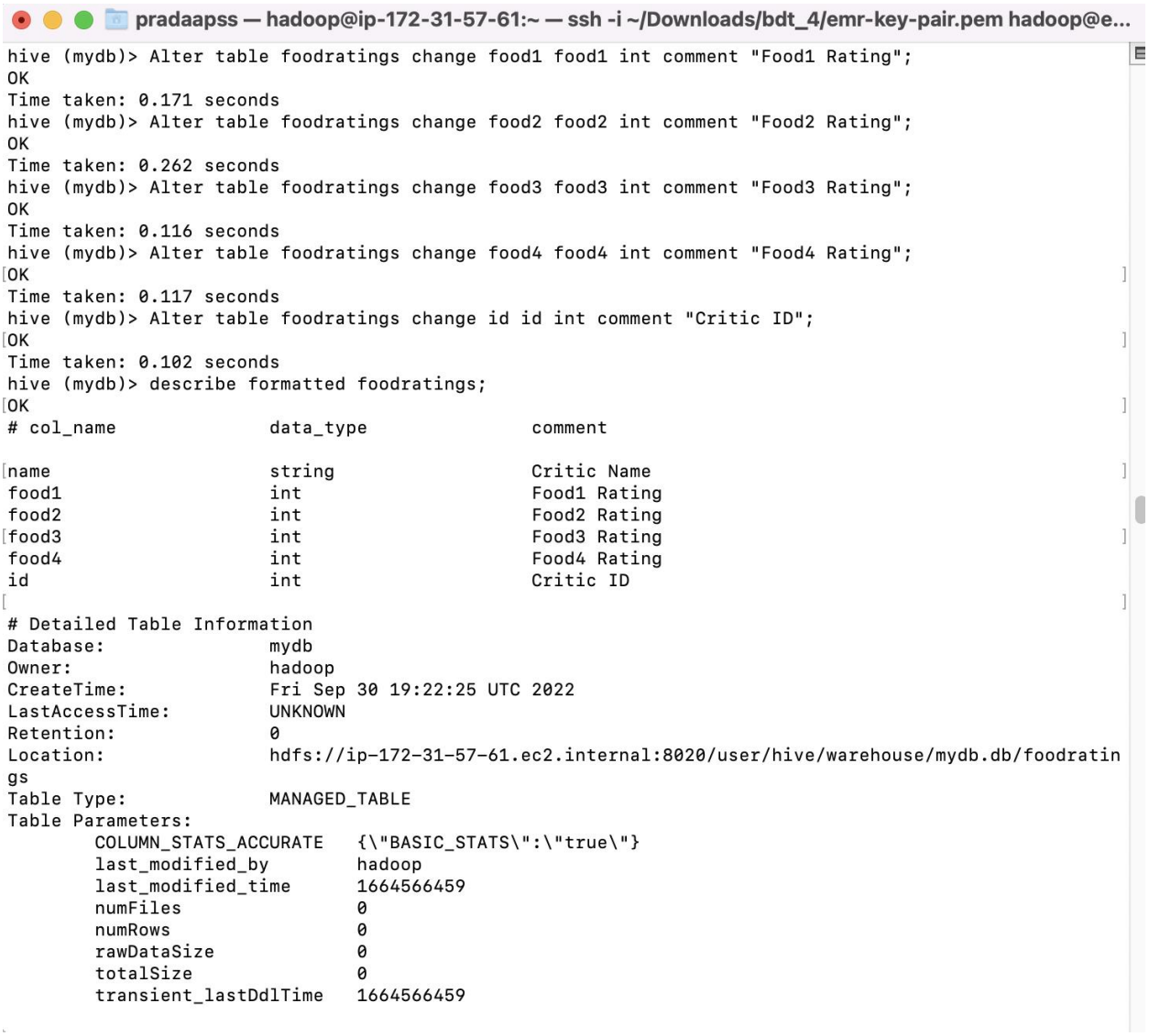
```
create database if not exists MyDb;
show databases
create table foodratings(name string, food1 int, food2 int, food3 int, food4 int, id int) row format delimited fields terminated by ',';
```



The table itself and each column should include a comment just to show me you know how to use comments (it does not matter what it says).

```
Alter table foodratings change name name string comment "Critic Name"
Alter table foodratings change food1 food1 int comment "Food1 Rating"
Alter table foodratings change food2 food2 int comment "Food2 Rating";
Alter table foodratings change food3 food3 int comment "Food3 Rating";
Alter table foodratings change food4 food4 int comment "Food4 Rating";
Alter table foodratings change id id int comment "Critic ID";
```

Execute a Hive command of ‘DESCRIBE FORMATTED MyDb.foodratings;’ and capture its output as one of the results of this exercise.



Then in MyDb create a table with name foodplaces having two columns with first called ‘id’ with the type of the first an integer, and the second column called ‘place’ with the type of the second a string. This table should also have storage format TEXTFILE and column separator a “,”. That is the underlying format should be a CSV file. No comments are needed for this table.

Execute a Hive command of ‘DESCRIBE FORMATTED MyDb.foodplaces’ and capture its output as another of the results of this exercise.

```
create table foodplaces(id int, place string) row format delimited
fields terminated by ',';
```

```
pradaapss — hadoop@ip-172-31-57-61:~ — ssh -i ~/Downloads/bdt_4/emr-key-pair.pem hadoop@e...
hive (mydb)> create table foodplaces(id int, place string) row format delimited fields terminated by '
,';
OK
Time taken: 0.082 seconds
hive (mydb)> describe formatted foodplaces;
OK
# col_name              data_type               comment

id                      int
place                   string

# Detailed Table Information
Database:               mydb
Owner:                  hadoop
CreateTime:             Fri Sep 30 19:37:58 UTC 2022
LastAccessTime:         UNKNOWN
Retention:              0
Location:               hdfs://ip-172-31-57-61.ec2.internal:8020/user/hive/warehouse/mydb.db/foodplace
s
Table Type:             MANAGED_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   {\"BASIC_STATS\":\"true\"}
        numFiles                0
        numRows                 0
        rawDataSize             0
        totalSize               0
        transient_lastDdlTime   1664566678

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        field.delim             ,
        serialization.format    ,
Time taken: 0.103 seconds, Fetched: 32 row(s)
```

Exercise 2) 2 points

Load the foodratings<magic number>.txt file created using TestDataGen from your local file system into the foodratings table.

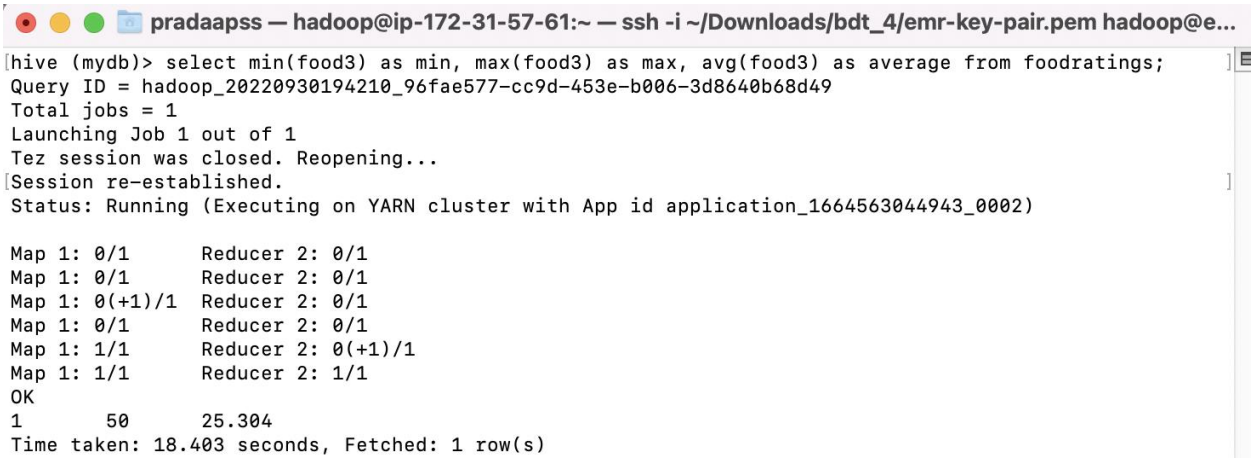
```
load data local inpath '/home/hadoop/foodratings227448.txt' overwrite into table foodratings;
```

```
pradaapss — hadoop@ip-172-31-57-61:~ — ssh -i ~/Downloads/bdt_4/emr-key-pair.pem hadoop@ec2-1...
hive (mydb)> load data local inpath '/home/hadoop/foodratings227448.txt' overwrite into table foodratings;
Loading data to table mydb.foodratings
OK
Time taken: 0.763 seconds
```

Execute a hive command to output the min, max and average of the values of the food3 column of the foodratings table. This should be one hive command, not three separate ones.

A copy of the hive command you wrote, the output of this query and the magic number are the result of this exercise.

```
select min(food3) as min, max(food3) as max, avg(food3) as average
from foodratings;
```



Exercise 3) 2 points

Execute a hive command to output the min, max and average of the values of the food1 column grouped by the first column ‘name’. This should be one hive command, not three separate ones.

The output should look something like:

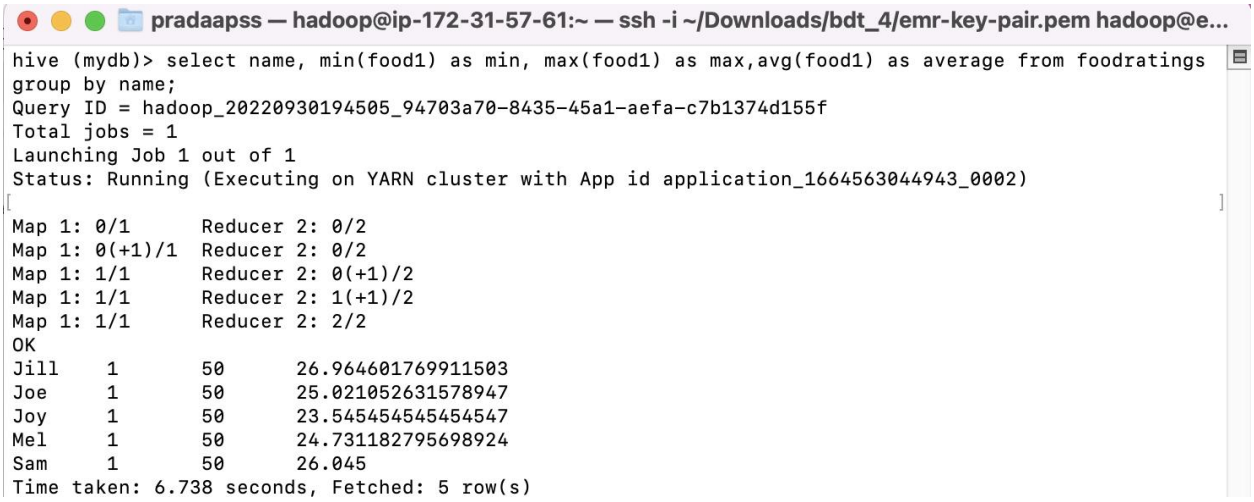
Mel 10 20 15

Bill 20, 30, 24

…

A copy of the hive command you wrote, the output of this query and the magic number are the result of this exercise.

```
select name, min(food1) as min, max(food1) as max,avg(food1) as
average from foodratings group by name;
```



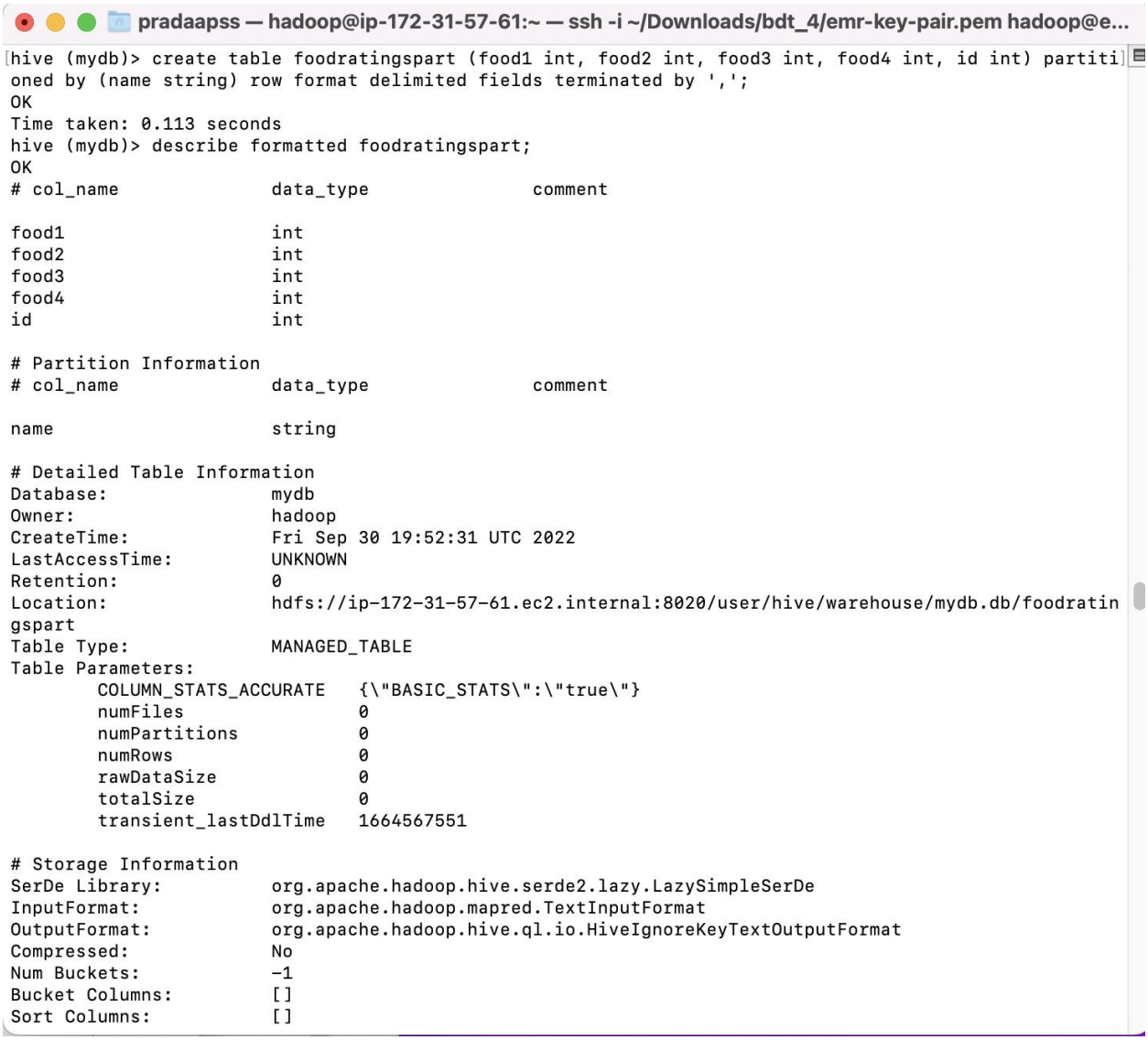
Exercise 4) 2 points

In MyDb create a partitioned table called ‘foodratingspart’

The partition field should be called ‘name’ and its type should be a string. The names of the non-partition columns should be food1, food2, food3, food4 and id and their types each an integer. The table should have storage format TEXTFILE and column separator a “,”. That is the underlying format should be a CSV file. No comments are needed for this table.

Execute a Hive command of ‘DESCRIBE FORMATTED MyDb.foodratingspart;’ and capture its output as the result of this exercise.

```
create table foodratingspart (food1 int, food2 int, food3 int, food4 int, id int) partitioned by (name string) row format delimited fields terminated by ',';
```



Exercise 5) 2 points

Assume that the number of food critics is relatively small, say less than 10 and the number places to eat is very large, say more than 10,000. In a few short sentences explain why using the (critic) name is a good choice for a partition field while using the place id is not.

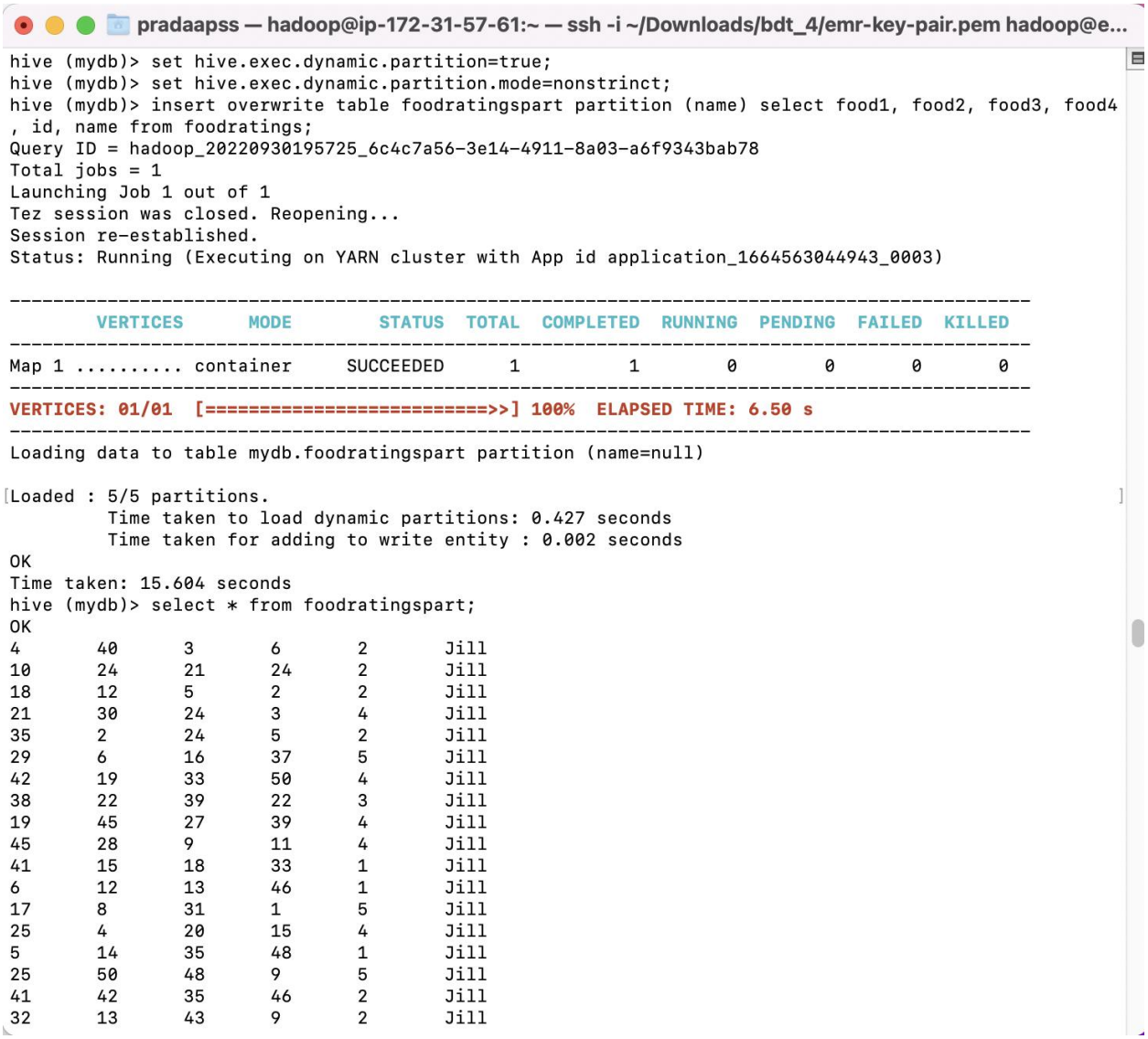
There are generally few critics, so the information can be organized effectively at the segment level.

Exercise 6) 2 points

Configure Hive to allow dynamic partition creation as described in the lecture. Now, use a hive command to copy from MyDB.foodratings into MyDB.foodratingspart to create a partitioned table from a non-partitioned one.

```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrinct;
insert overwrite table foodratingspart partition (name) select food1,
food2, food3, food4, id, name from foodratings;

select * from foodratingspart;
```



```
pradaapss — hadoop@ip-172-31-57-61:~ — ssh -i ~/Downloads/bdt_4/emr-key-pair.pem hadoop@e...
hive (mydb)> set hive.exec.dynamic.partition=true;
hive (mydb)> set hive.exec.dynamic.partition.mode=nonstrinct;
hive (mydb)> insert overwrite table foodratingspart partition (name) select food1, food2, food3, food4
, id, name from foodratings;
Query ID = hadoop_20220930195725_6c4c7a56-3e14-4911-8a03-a6f9343bab78
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1664563044943_0003)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 6.50 s
--------------------------------------------------------------------------------
Loading data to table mydb.foodratingspart partition (name=null)

Loaded : 5/5 partitions.
         Time taken to load dynamic partitions: 0.427 seconds
         Time taken for adding to write entity : 0.002 seconds
OK
Time taken: 15.604 seconds
hive (mydb)> select * from foodratingspart;
OK
4       40      3       6       2       Jill
10      24      21      24      2       Jill
18      12      5       2       2       Jill
21      30      24      3       4       Jill
35      2       24      5       2       Jill
29      6       16      37      5       Jill
42      19      33      50      4       Jill
38      22      39      22      3       Jill
19      45      27      39      4       Jill
45      28      9       11      4       Jill
41      15      18      33      1       Jill
6       12      13      46      1       Jill
17      8       31      1       5       Jill
25      4       20      15      4       Jill
5       14      35      48      1       Jill
25      50      48      9       5       Jill
41      42      35      46      2       Jill
32      13      43      9       2       Jill
```

Hint: The ‘name’ column from MyDB.foodratings should be mentioned last in this command (whatever it is).

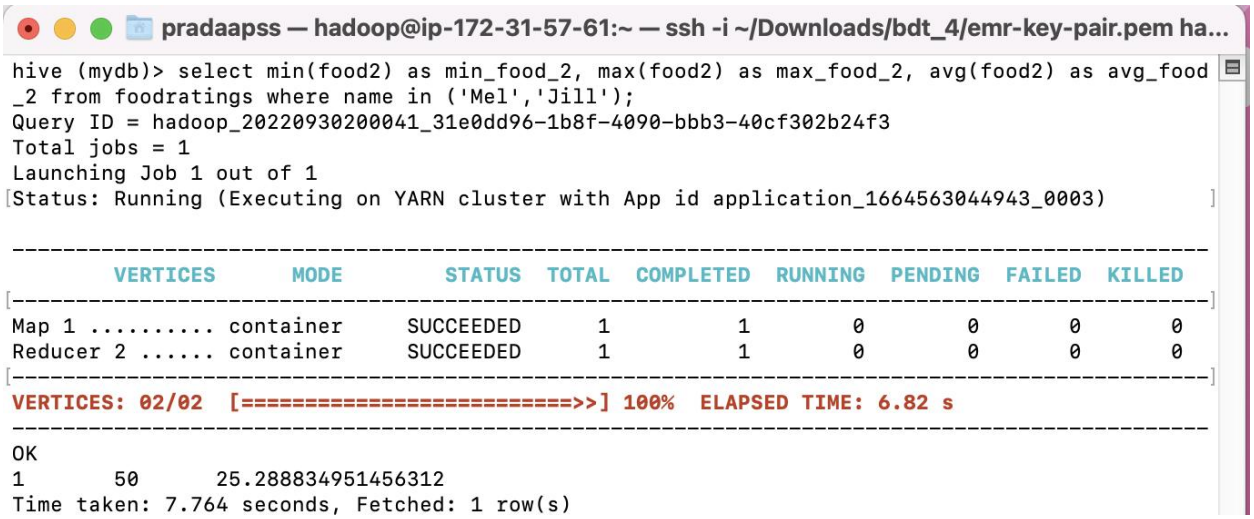
Provide a copy of the command you use to load the ‘foodratingspart’ table as a result of this exercise.

Execute a hive command to output the min, max and average of the values of the food2 column of MyDB.foodratingspart where the food critic ‘name’ is either Mel or Jill.

The query and the output of this query are other results of this exercise. It should look something like

10 20 15

```
select min(food2) as min_food_2, max(food2) as max_food_2, avg(food2) as avg_food_2 from foodratings where name in ('Mel','Jill');
```



Exercise 7) 2 points

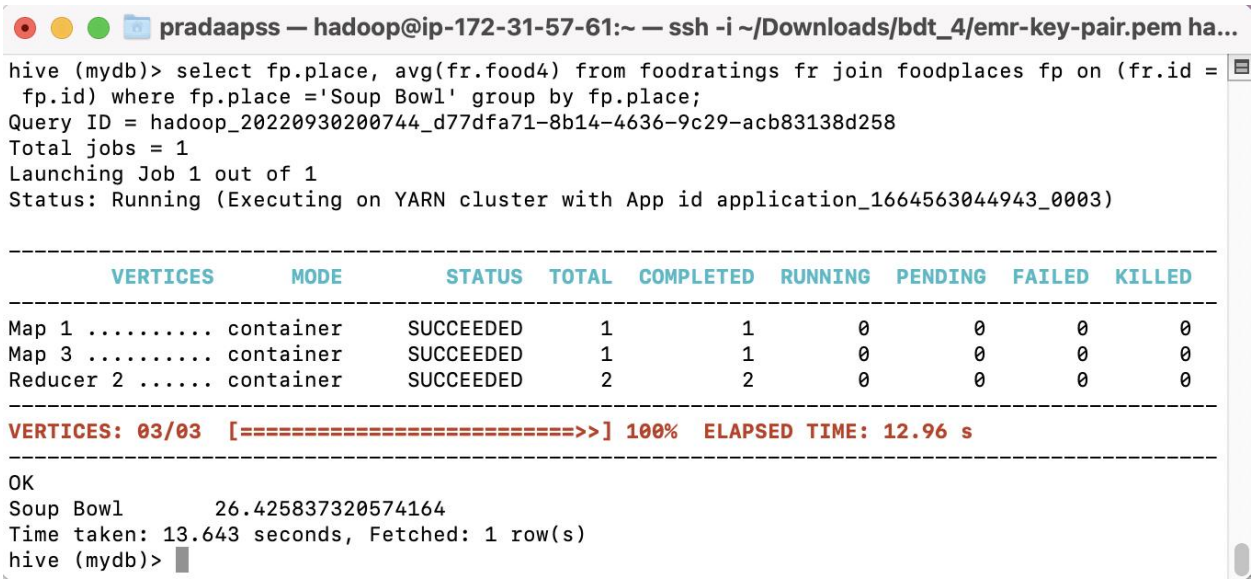
Load the foodplaces<.magic number>.txt file created using TestDataGen from your local file system into the foodplaces table.

Use a join operation between the two tables (foodratings and foodplaces) to provide the average rating for field food4 for the restaurant ‘Soup Bowl’

The output of this query is the result of this exercise. It should look something like

Soup Bowl 20

```
select fp.place, avg(fr.food4) from foodratings fr join foodplaces fp on (fr.id = fp.id) where fp.place ='Soup Bowl' group by fp.place;
```



Exercise 8) 4 points

Read the article “An Introduction to Big Data Formats” found on the blackboard in section “Articles” and provide short (2 to 4 sentence) answers to the following questions:

a) When is the most important consideration when choosing a row format and when a column format for your big data file?

Line information is used when your query needs to reach every section of the column. Section-based organization is chosen for examination inquiries that require just a few segments of data.

b) What is “splittability” for a column file format and why is it important when processing large volumes of data?

Splitability refers to the ability to break up data into smaller records that can be handled independently. As a result, large volumes of data could be processed efficiently. It usually involves breaking the project down into parts and outsourcing them to separate processors. Large-scale parallel processing is crucial to performance. The files in your dataset, for example, won't be "splittable", i.e. decomposable into smaller records that can be dealt with separately, if they contain massive XML structure or JSON records.

c) What can files stored in column format achieve better compression than those stored in row format?

Information organized in sections can achieve better pressure rates than information organized along lines. When you store attributes by segment, with items of the same kind close to one another, the client can exert more professional pressure than if you were to store lines of information.

d) Under what circumstances would it be the best choice to use the “Parquet” column file format?

Parquet excels at analyzing large datasets with multiple segments. Each Parquet document contains information that is paired and organized by "line bunch." The information values are organized by segment for each line bunch. For important tasks and read-weighty responsibilities, parquet is a good option.