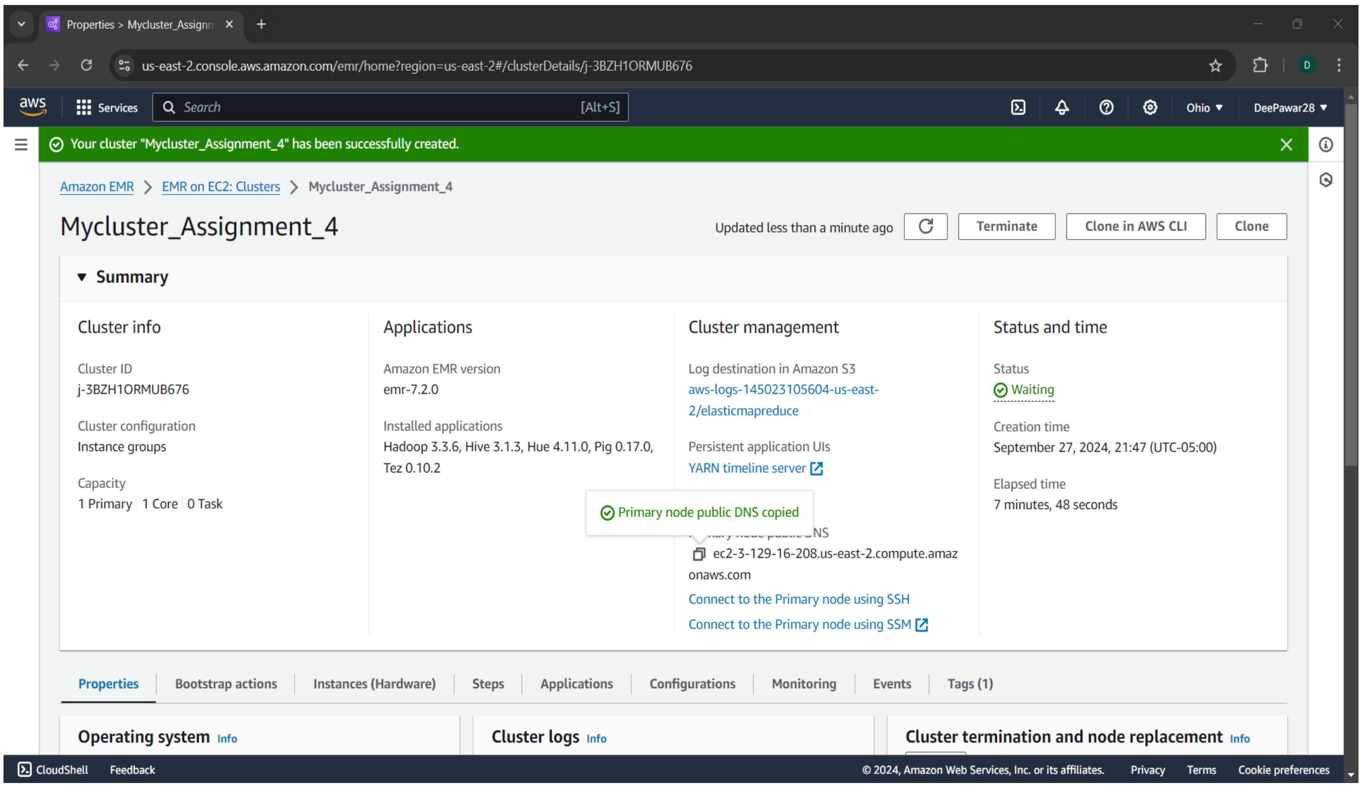
Name: Deep Pawar(A20545137)
Professor: Joseph Rosen
Institute: Illinois Institute of Technology

CSP 554: Big Data Technologies

Fall 2024 - Assignment 4

- **Questions and Answers:**

- Configure a Hadoop (AWS EMR) environment as you have done for previous assignments:



us-east-2.console.aws.amazon.com/emr/home?region=us-east-2#/clusterDetails/j-3BZH1ORMUB676

Your cluster "Mycluster_Assignment_4" has been successfully created.

Amazon EMR > EMR on EC2: Clusters > Mycluster_Assignment_4

Mycluster_Assignment_4

Updated less than a minute ago

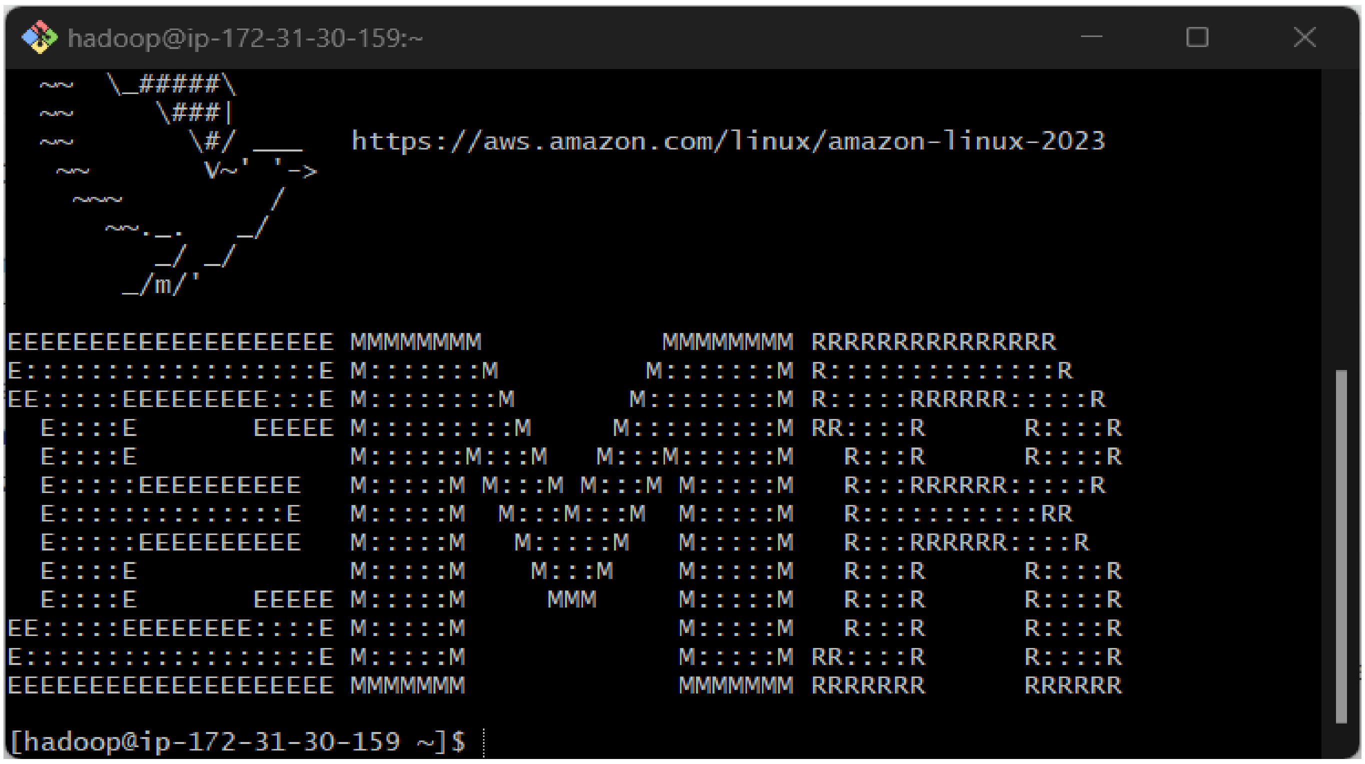
Summary

Cluster info
Cluster ID
j-3BZH1ORMUB676
Cluster configuration
Instance groups
Capacity
1 Primary | 1 Core | 0 Task

Applications
Amazon EMR version
emr-7.2.0
Installed applications
Hadoop 3.3.6, Hive 3.1.3, Hue 4.11.0, Pig 0.17.0, Tez 0.10.2

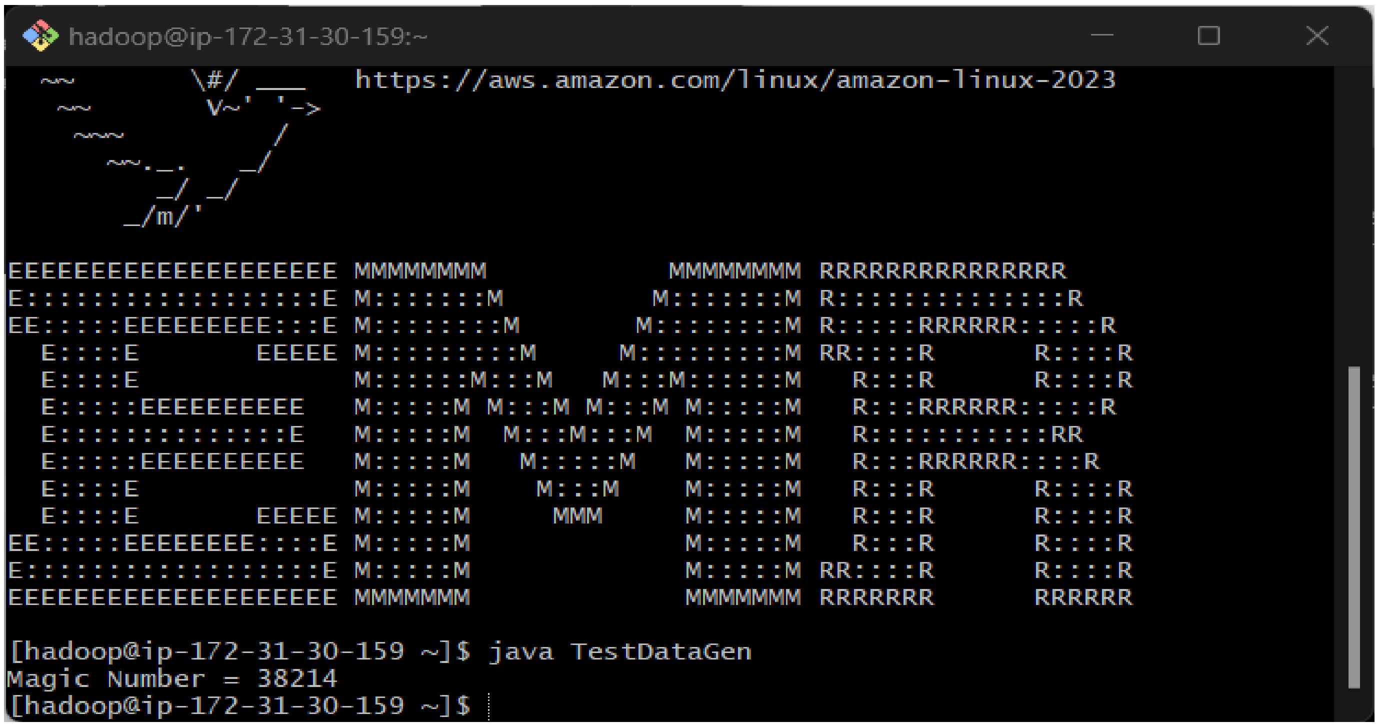
Cluster management
Log destination in Amazon S3
aws-logs-145023105604-us-east-2/elasticmapreduce
Persistent application UIs
YARN timeline server
Primary node public DNS copied
Primary node public DNS
ec2-3-129-16-208.us-east-2.compute.amazonaws.com
Connect to the Primary node using SSH
Connect to the Primary node using SSM

Status and time
Status
Waiting
Creation time
September 27, 2024, 21:47 (UTC-05:00)
Elapsed time
7 minutes, 48 seconds



```
hadoop@ip-172-31-30-159:~
  ~~   \_#####\
  ~~     \###|
  ~~       \#/ ___    https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'

EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-30-159 ~]$
```

- Log on to your VM using ssh and execute the file using “java TestDataGen”



hadoop@ip-172-31-30-159:~

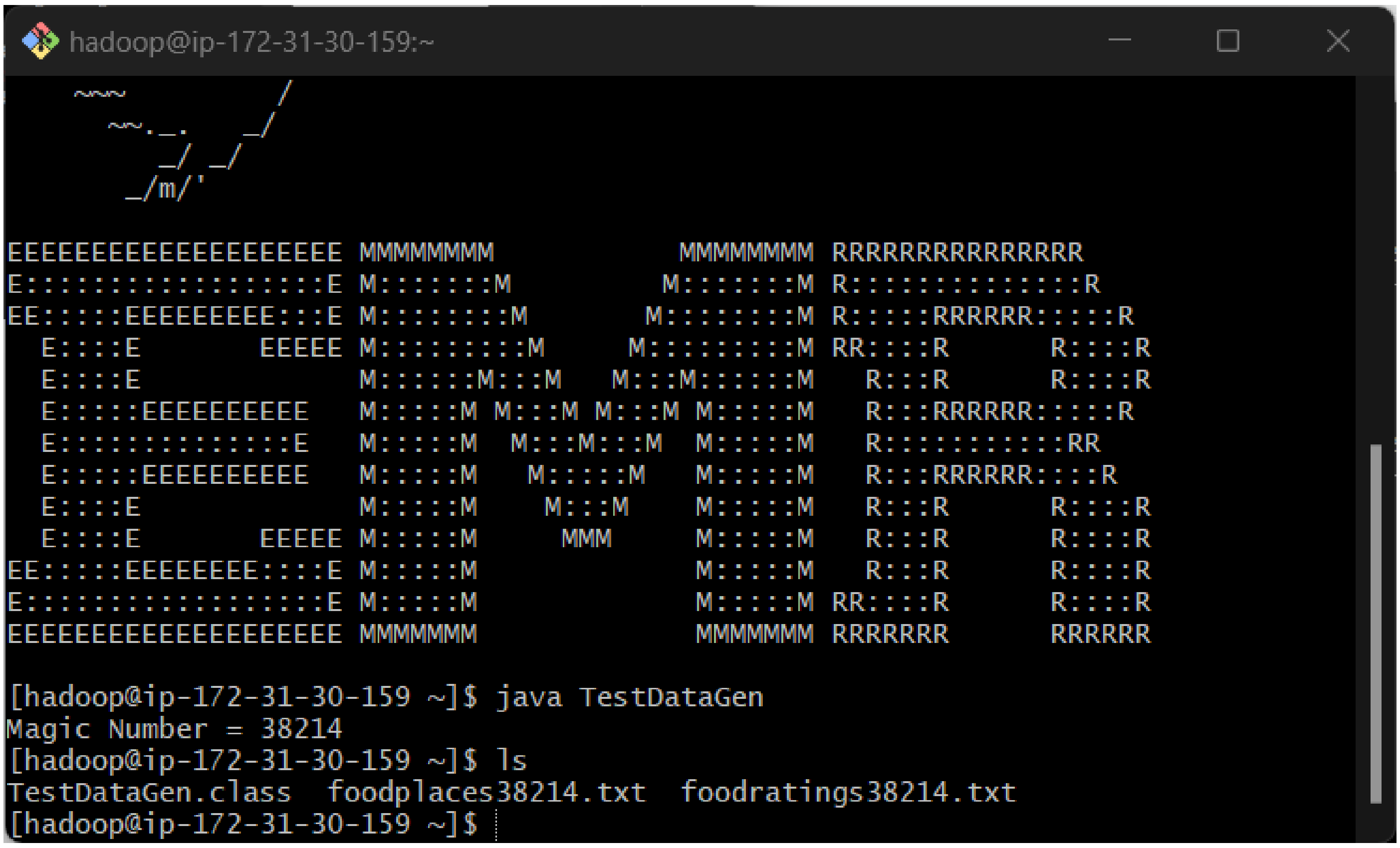
```
   ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
    ~~       V~' '->
     ~~~         /
       ~~._.   _/
          _/ _/
        _/m/'

EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-30-159 ~]$ java TestDataGen
Magic Number = 38214
[hadoop@ip-172-31-30-159 ~]$
```

- This will output a magic number which you should copy down and provide with the results of your assignment.

  Ans: Magic Number = 38214

- It will also place the files foodratings<magic number>.txt and foodplaces<magic number>.txt in your VM home directory

  Ans:



hadoop@ip-172-31-30-159:~

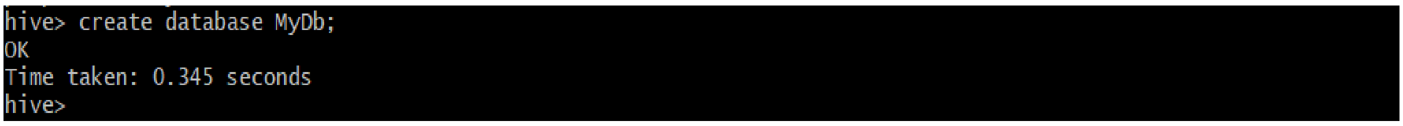
```
     ~~~         /
       ~~._.   _/
          _/ _/
        _/m/'

EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-30-159 ~]$ java TestDataGen
Magic Number = 38214
[hadoop@ip-172-31-30-159 ~]$ ls
TestDataGen.class  foodplaces38214.txt  foodratings38214.txt
[hadoop@ip-172-31-30-159 ~]$
```

Exercise 1) 2 points

a. Create a Hive database called “MyDb”.

Ans:

- **Command Used:** create database MyDb;

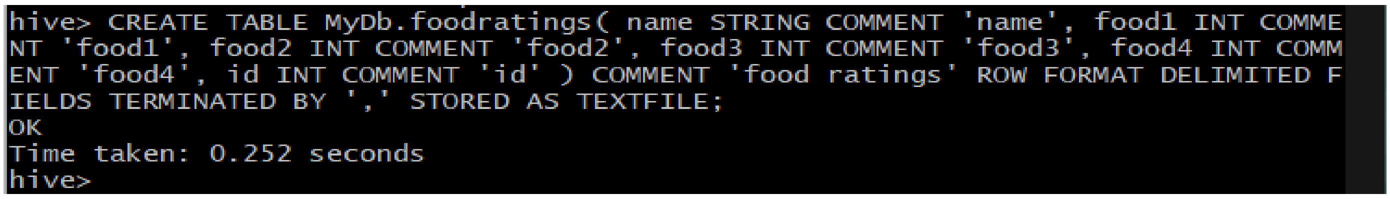


```
hive> create database MyDb;
OK
Time taken: 0.345 seconds
hive>
```

b. Now in MyDb create a table with name foodratings

Ans:

- **Command Used:** CREATE TABLE MyDb.foodratings(name STRING COMMENT 'name', food1 INT COMMENT 'food1', food2 INT COMMENT 'food2', food3 INT COMMENT 'food3', food4 INT COMMENT 'food4', id INT COMMENT 'id') COMMENT 'food ratings' ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

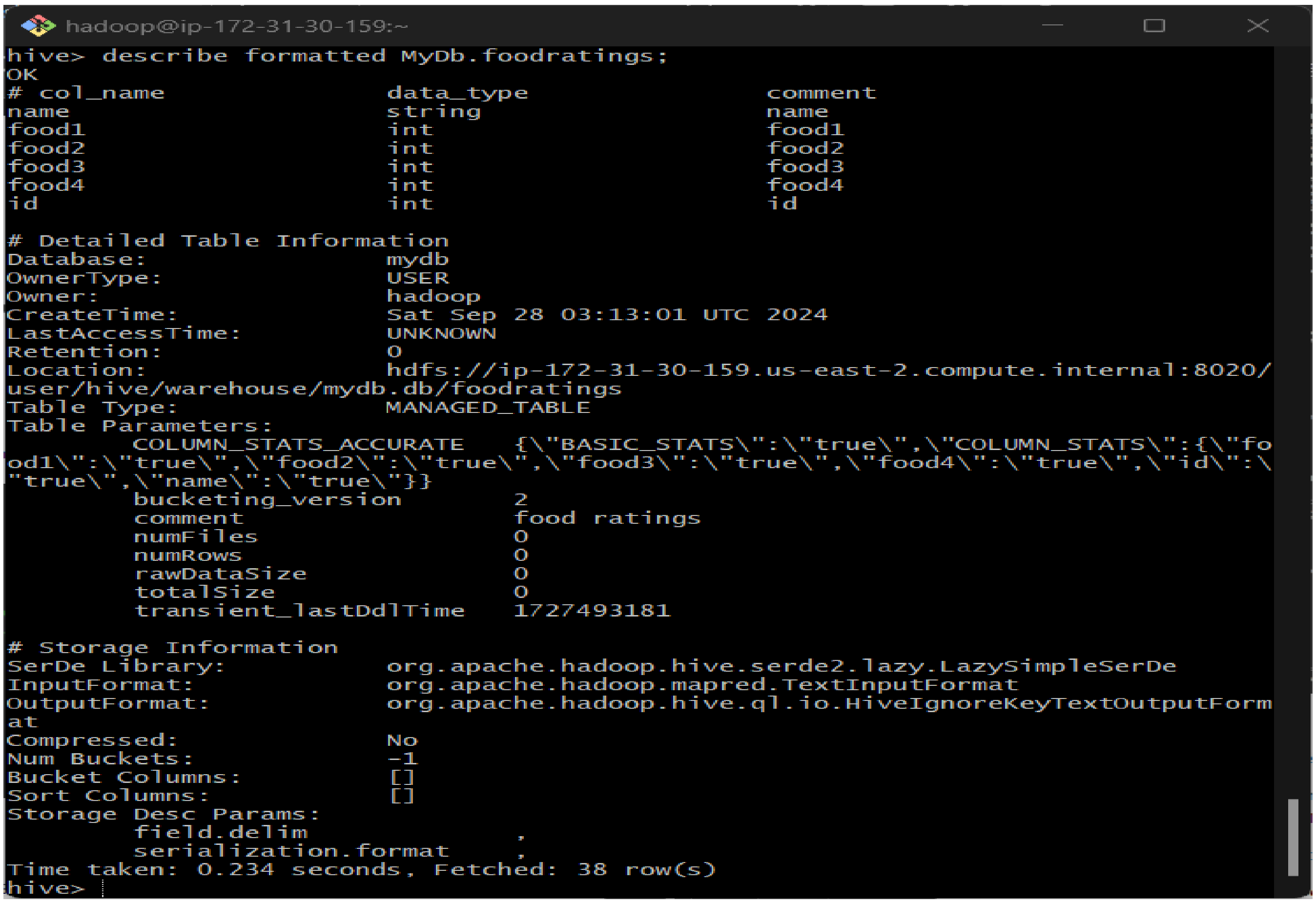


```
hive> CREATE TABLE MyDb.foodratings( name STRING COMMENT 'name', food1 INT COMME
NT 'food1', food2 INT COMMENT 'food2', food3 INT COMMENT 'food3', food4 INT COMM
ENT 'food4', id INT COMMENT 'id' ) COMMENT 'food ratings' ROW FORMAT DELIMITED F
IELDS TERMINATED BY ',' STORED AS TEXTFILE;
OK
Time taken: 0.252 seconds
hive>
```

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

Ans:

- **Command Used:** describe formatted MyDb.foodratings;

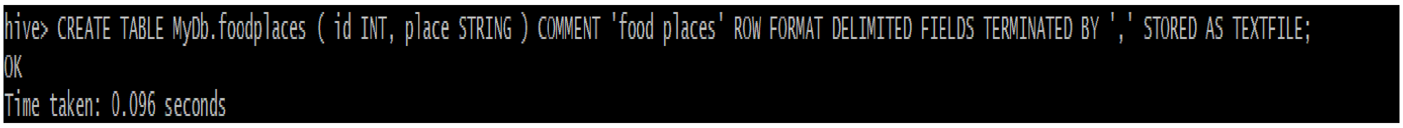


```
hive> describe formatted MyDb.foodratings;
OK
# col_name              data_type               comment
name                    string                  name
food1                   int                     food1
food2                   int                     food2
food3                   int                     food3
food4                   int                     food4
id                      int                     id

# Detailed Table Information
Database:               mydb
OwnerType:              USER
Owner:                  hadoop
CreateTime:             Sat Sep 28 03:13:01 UTC 2024
LastAccessTime:         UNKNOWN
Retention:              0
Location:               hdfs://ip-172-31-30-159.us-east-2.compute.internal:8020/
user/hive/warehouse/mydb.db/foodratings
Table Type:             MANAGED_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   {\"BASIC_STATS\":\"true\",\"COLUMN_STATS\":{\"fo
od1\":\"true\",\"food2\":\"true\",\"food3\":\"true\",\"food4\":\"true\",\"id\":\
"true\",\"name\":\"true\"}}
        bucketing_version       2
        comment                 food ratings
        numFiles                0
        numRows                 0
        rawDataSize             0
        totalSize               0
        transient_lastDdlTime   1727493181

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputForm
at
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        field.delim             ,
        serialization.format    ,
Time taken: 0.234 seconds, Fetched: 38 row(s)
hive>
```

d. Then in MyDb create a table with name foodplaces

Ans:

- **Command Used:** CREATE TABLE MyDb.foodplaces (id INT, place STRING) COMMENT 'food places' ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

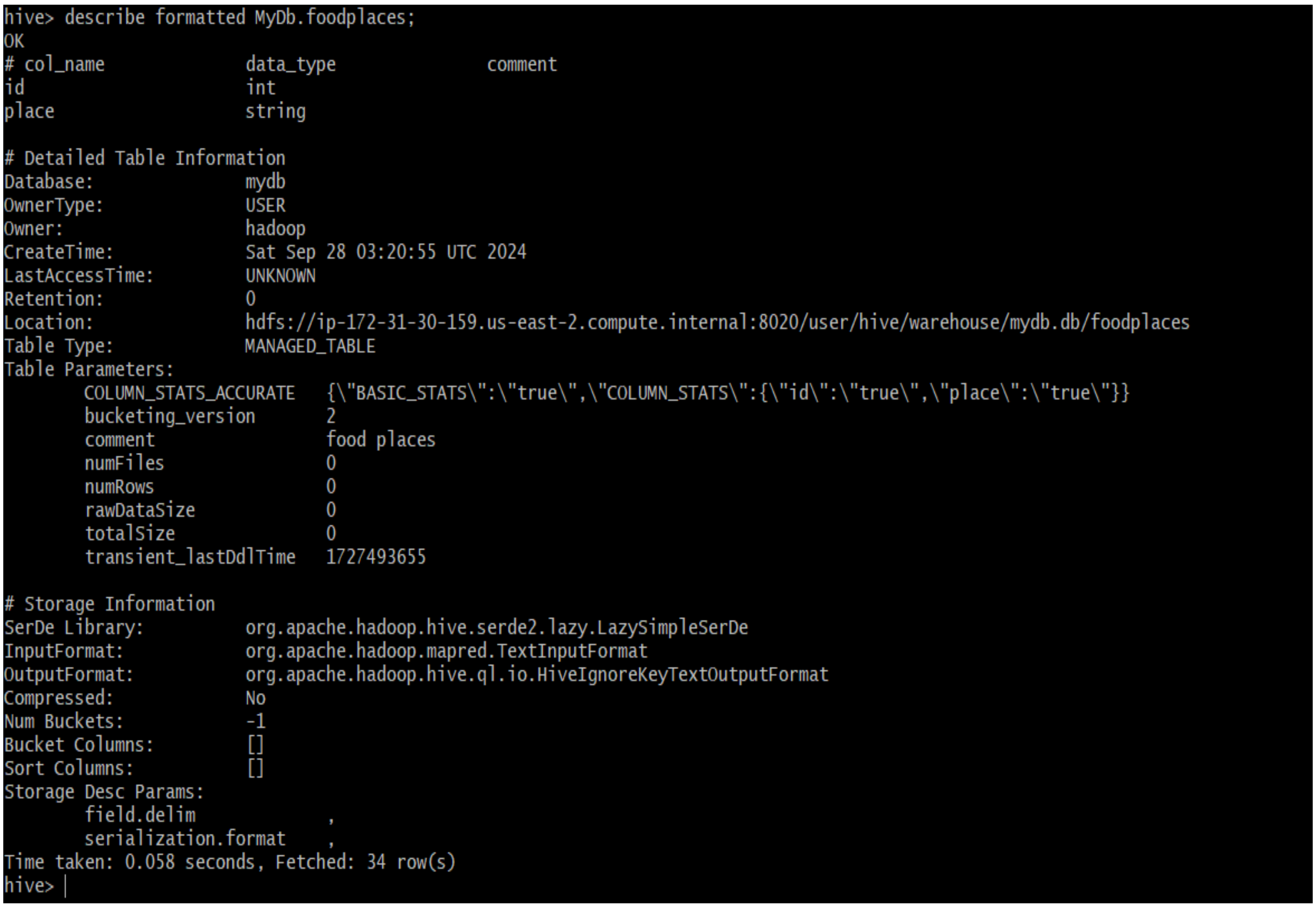


```
hive> CREATE TABLE MyDb.foodplaces ( id INT, place STRING ) COMMENT 'food places' ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
OK
Time taken: 0.096 seconds
```

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

Ans:

- **Command Used:** describe formatted MyDb.foodplaces;



```
hive> describe formatted MyDb.foodplaces;
OK
# col_name              data_type               comment
id                      int
place                   string

# Detailed Table Information
Database:               mydb
OwnerType:              USER
Owner:                  hadoop
CreateTime:             Sat Sep 28 03:20:55 UTC 2024
LastAccessTime:         UNKNOWN
Retention:              0
Location:               hdfs://ip-172-31-30-159.us-east-2.compute.internal:8020/user/hive/warehouse/mydb.db/foodplaces
Table Type:             MANAGED_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   {\"BASIC_STATS\":\"true\",\"COLUMN_STATS\":{\"id\":\"true\",\"place\":\"true\"}}
        bucketing_version       2
        comment                 food places
        numFiles                0
        numRows                 0
        rawDataSize             0
        totalSize               0
        transient_lastDdlTime   1727493655

# 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: 0.058 seconds, Fetched: 34 row(s)
hive> |
```

Exercise 2) 2 points

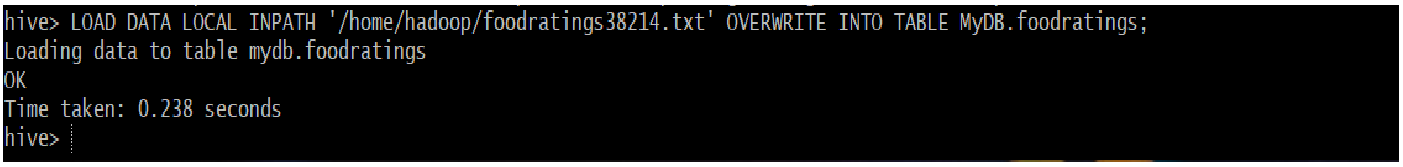
a. Load the foodratings<magic number>.txt file created using TestDataGen from your local file system into the foodratings table.

Ans:

- **Magic number** = 38214
- **Files Generated** = foodratings38214.txt and foodplaces38214.txt

Here in this case, we are using foodrating38214.txt file dataset to load into MyDb.foodratings table.

- **Command Used:** LOAD DATA LOCAL INPATH '/home/hadoop/foodratings38214.txt' OVERWRITE INTO TABLE MyDB.foodratings;

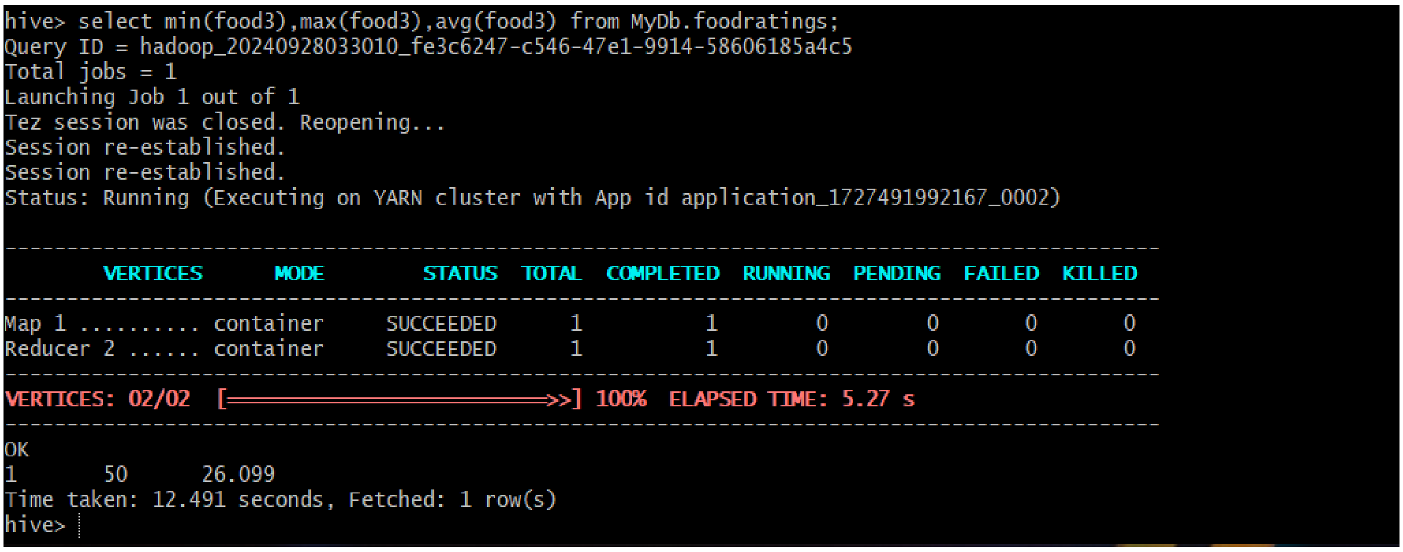


```
hive> LOAD DATA LOCAL INPATH '/home/hadoop/foodratings38214.txt' OVERWRITE INTO TABLE MyDB.foodratings;
Loading data to table mydb.foodratings
OK
Time taken: 0.238 seconds
hive>
```

b. Execute a single 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.

Ans:

- **Command Used:** select min(food3),max(food3),avg(food3) from MyDb.foodratings;



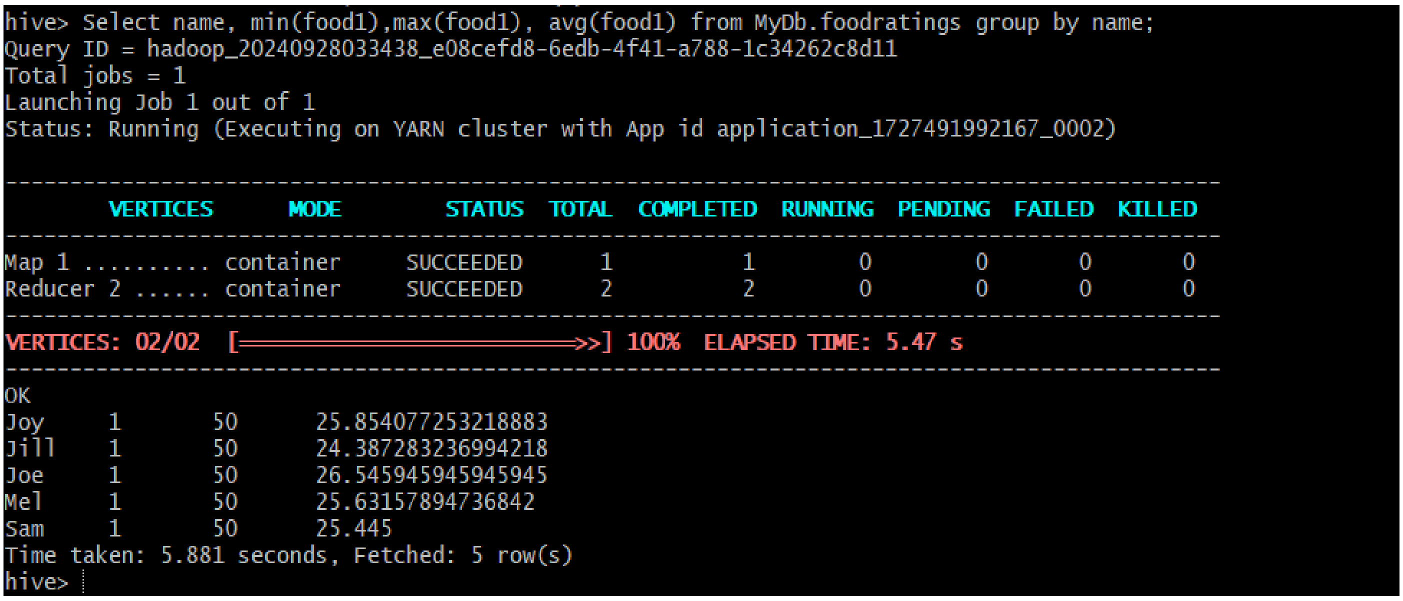
```
hive> select min(food3),max(food3),avg(food3) from MyDb.foodratings;
Query ID = hadoop_20240928033010_fe3c6247-c546-47e1-9914-58606185a4c5
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1727491992167_0002)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 5.27 s
----------------------------------------------------------------------------------------------
OK
1       50      26.099
Time taken: 12.491 seconds, Fetched: 1 row(s)
hive>
```

Exercise 3) 2 points

a. Execute a hive command to output the min, max and average of the values of the food1 column grouped by the first column ‘name’.

Ans:

- **Magic number** = 38214
- **Command Used:** select name, min(food1),max(food1), avg(food1) from MyDb.foodratings group by name;



```
hive> Select name, min(food1),max(food1), avg(food1) from MyDb.foodratings group by name;
Query ID = hadoop_20240928033438_e08cefd8-6edb-4f41-a788-1c34262c8d11
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1727491992167_0002)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      2          2        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 5.47 s
----------------------------------------------------------------------------------------------
OK
Joy     1       50      25.854077253218883
Jill    1       50      24.387283236994218
Joe     1       50      26.545945945945945
Mel     1       50      25.63157894736842
Sam     1       50      25.445
Time taken: 5.881 seconds, Fetched: 5 row(s)
hive>
```

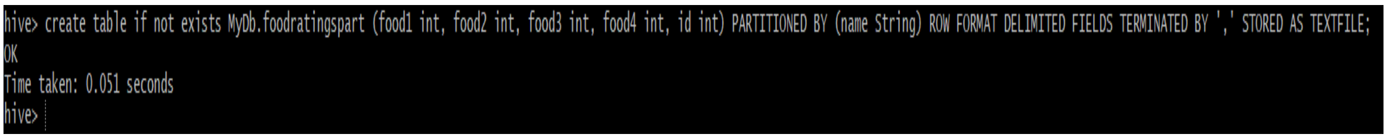
Exercise 4) 2 points

a. In MyDb create a partitioned table called 'foodratingspart'

Ans:

Here In MyDb, we will create a partitioned table called 'foodratingspart'. The partition field is '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.

- **Command Used:** create table if not exists MyDb.foodratingspart (food1 int, food2 int, food3 int, food4 int, id int) PARTITIONED BY (name String) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

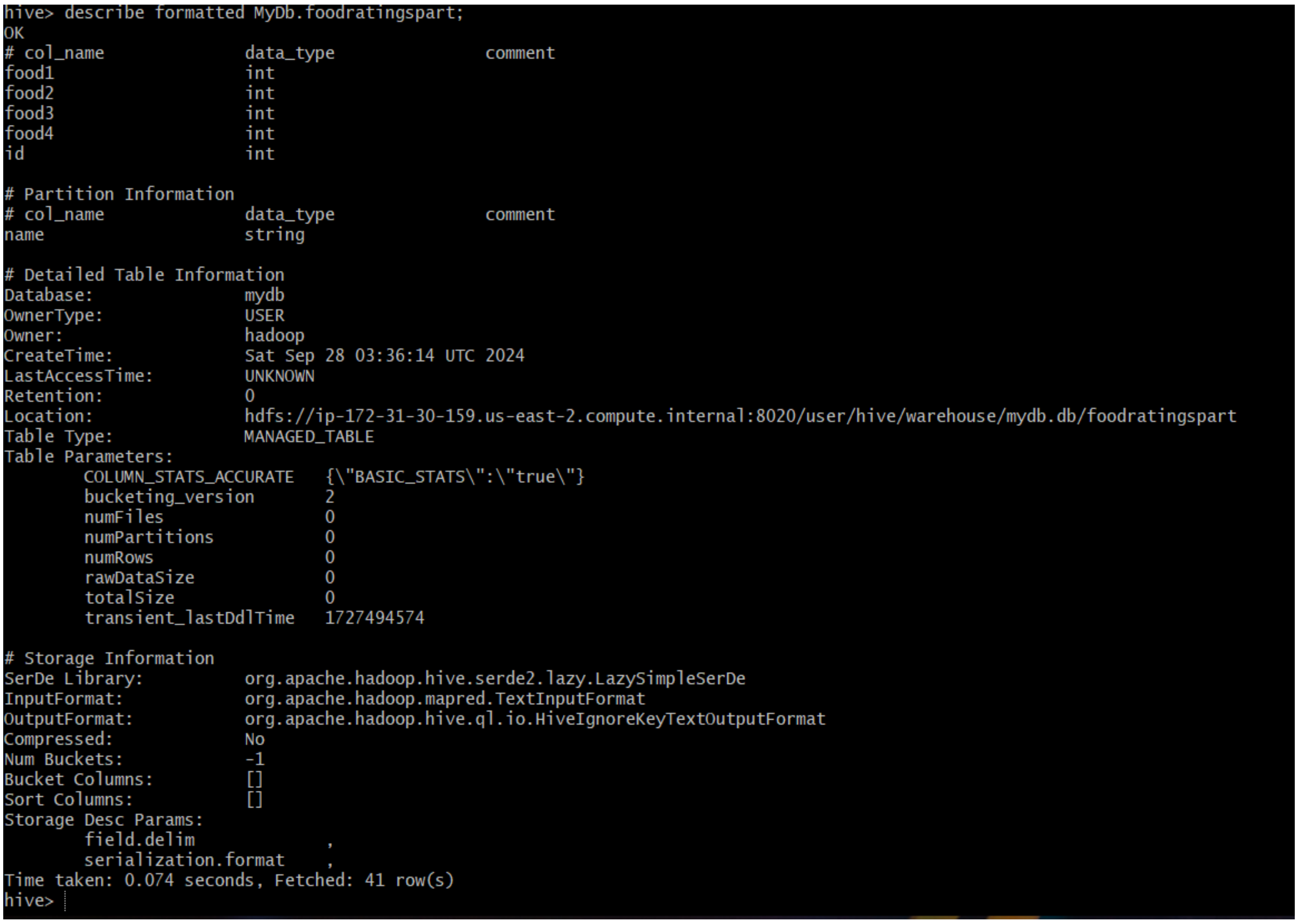


```
hive> create table if not exists MyDb.foodratingspart (food1 int, food2 int, food3 int, food4 int, id int) PARTITIONED BY (name String) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
OK
Time taken: 0.051 seconds
hive>
```

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

Ans:

- **Command Used:** describe formatted MyDb.foodratingspart;



```
hive> describe formatted MyDb.foodratingspart;
OK
# col_name              data_type               comment
food1                   int
food2                   int
food3                   int
food4                   int
id                      int

# Partition Information
# col_name              data_type               comment
name                    string

# Detailed Table Information
Database:               mydb
OwnerType:              USER
Owner:                  hadoop
CreateTime:             Sat Sep 28 03:36:14 UTC 2024
LastAccessTime:         UNKNOWN
Retention:              0
Location:               hdfs://ip-172-31-30-159.us-east-2.compute.internal:8020/user/hive/warehouse/mydb.db/foodratingspart
Table Type:             MANAGED_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   {\"BASIC_STATS\":\"true\"}
        bucketing_version       2
        numFiles                0
        numPartitions           0
        numRows                 0
        rawDataSize             0
        totalSize               0
        transient_lastDdlTime   1727494574

# 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: 0.074 seconds, Fetched: 41 row(s)
hive>
```

Exercise 5) 2 points

Q. 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.

Ans:

Partitioning a table according to a column such as the critic's name is efficient since it reduces the number of partitions and speeds up the execution of queries that filter by critic name. In this case, the column contains fewer unique values less than 10. This is because the amount of data examined can be decreased by assigning each partition to a certain critic.

On the other hand, because there are a lot of places more than 10,000 using the location id as a partition field is not a smart idea. When a highly cardinal column (such as place id) is used for partitioning, this leads to an excessive number of partitions and suboptimal query performance. Keeping track of so many partitions can be expensive and time-consuming, and query optimization benefits drop as one grows in number.

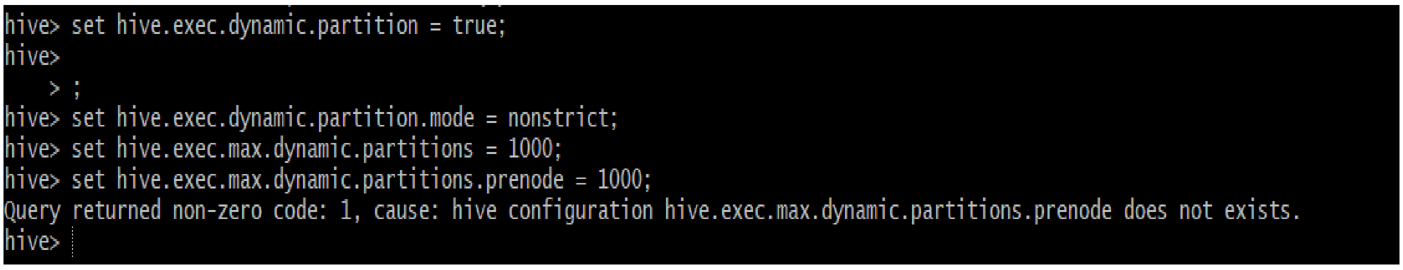
Exercise 6) 2 points

a. Configure Hive to allow dynamic partition creation as described in the lecture.

Ans:

- **Configuration code commands used:**

set hive.exec.dynamic.partition = true;
set hive.exec.dynamic.partition.mode = nonstrict;
set hive.exec.max.dynamic.partitions = 1000;
set hive.exec.max.dynamic.partitions.prenode = 1000;

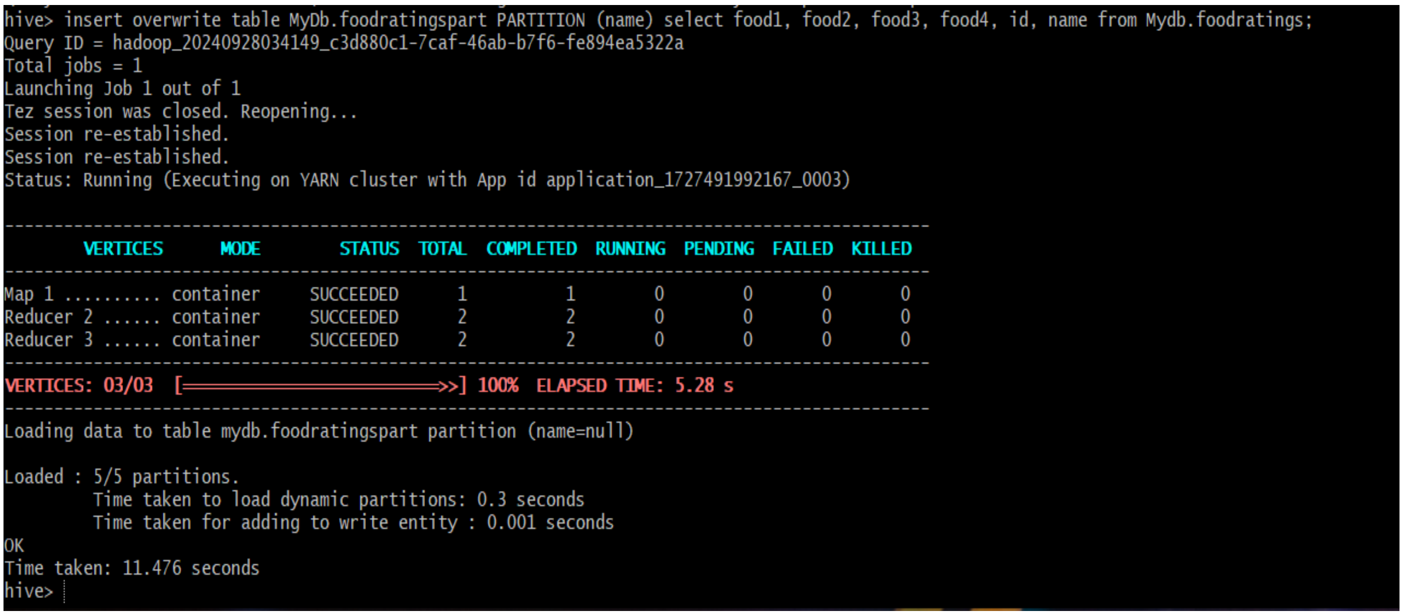


```
hive> set hive.exec.dynamic.partition = true;
hive>
    > ;
hive> set hive.exec.dynamic.partition.mode = nonstrict;
hive> set hive.exec.max.dynamic.partitions = 1000;
hive> set hive.exec.max.dynamic.partitions.prenode = 1000;
Query returned non-zero code: 1, cause: hive configuration hive.exec.max.dynamic.partitions.prenode does not exists.
hive>
```

b. Now, use a hive command to copy from MyDB.foodratings into MyDB.foodratingspart to create a partitioned table from a non-partitioned one.

Ans:

- **Command Used:** insert overwrite table MyDb.foodratingspart PARTITION (name) select food1, food2, food3, food4, id, name from Mydb.foodratings;

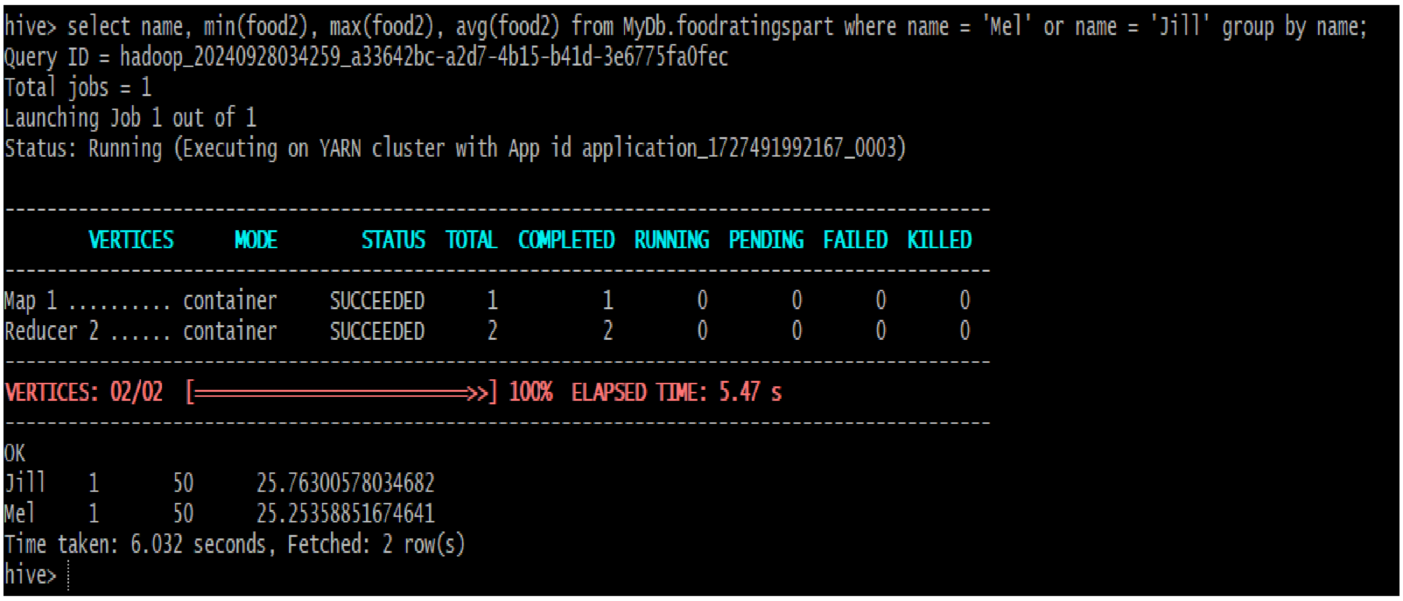


```
hive> insert overwrite table MyDb.foodratingspart PARTITION (name) select food1, food2, food3, food4, id, name from Mydb.foodratings;
Query ID = hadoop_20240928034149_c3d880c1-7caf-46ab-b7f6-fe894ea5322a
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1727491992167_0003)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      2          2        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      2          2        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 5.28 s
----------------------------------------------------------------------------------------------
Loading data to table mydb.foodratingspart partition (name=null)

Loaded : 5/5 partitions.
         Time taken to load dynamic partitions: 0.3 seconds
         Time taken for adding to write entity : 0.001 seconds
OK
Time taken: 11.476 seconds
hive>
```

c. 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.

Ans:

- **Command Used:** select name, min(food2), max(food2), avg(food2) from MyDb.foodratingspart where name = 'Mel' or name = 'Jill' group by name;



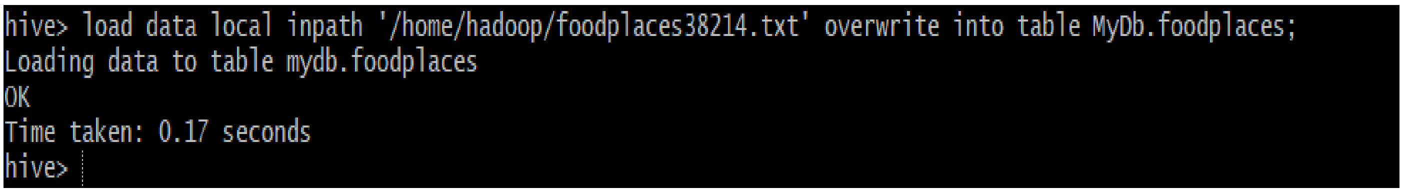
```
hive> select name, min(food2), max(food2), avg(food2) from MyDb.foodratingspart where name = 'Mel' or name = 'Jill' group by name;
Query ID = hadoop_20240928034259_a33642bc-a2d7-4b15-b41d-3e6775fa0fec
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1727491992167_0003)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      2          2        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 5.47 s
----------------------------------------------------------------------------------------------
OK
Jill    1       50      25.76300578034682
Mel     1       50      25.25358851674641
Time taken: 6.032 seconds, Fetched: 2 row(s)
hive>
```

Exercise 7) 2 points

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

Ans:

- **Command Used:** load data local inpath '/home/hadoop/foodplaces38214.txt' overwrite into table MyDb.foodplaces;

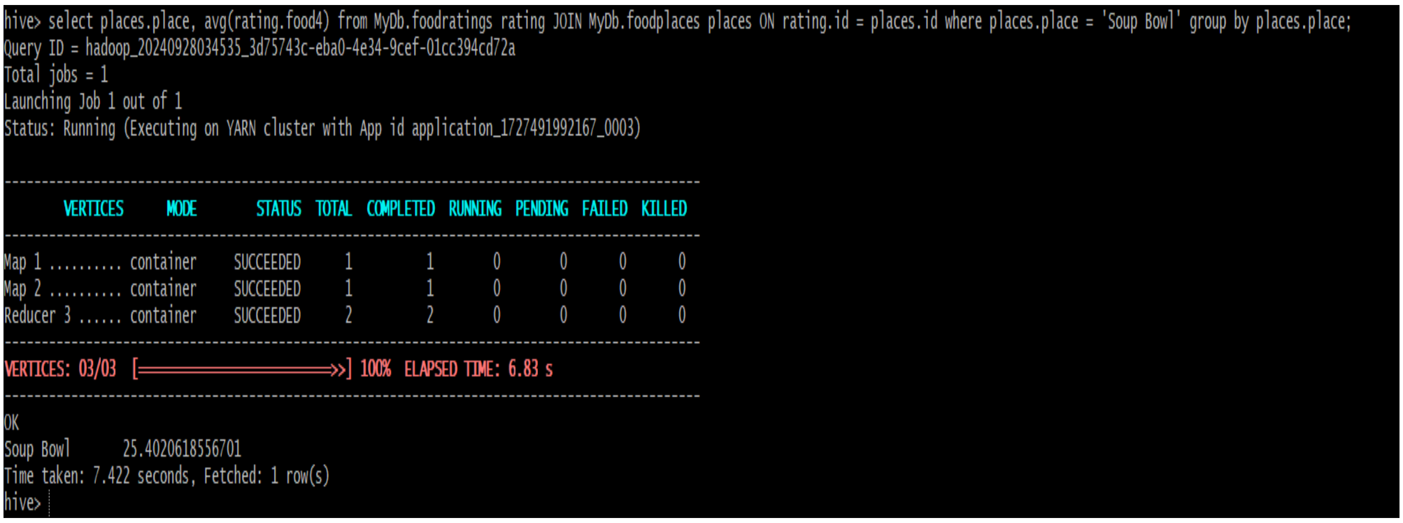


```
hive> load data local inpath '/home/hadoop/foodplaces38214.txt' overwrite into table MyDb.foodplaces;
Loading data to table mydb.foodplaces
OK
Time taken: 0.17 seconds
hive>
```

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

Ans:

- **Command Used:** select places.place, avg(rating.food4) from MyDb.foodratings rating JOIN MyDb.foodplaces places ON rating.id = places.id where places.place = 'Soup Bowl' group by places.place;



```
hive> select places.place, avg(rating.food4) from MyDb.foodratings rating JOIN MyDb.foodplaces places ON rating.id = places.id where places.place = 'Soup Bowl' group by places.place;
Query ID = hadoop_20240928034535_3d75743c-eba0-4e34-9cef-01cc394cd72a
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1727491992167_0003)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Map 2 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      2          2        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 6.83 s
----------------------------------------------------------------------------------------------
OK
Soup Bowl       25.4020618556701
Time taken: 7.422 seconds, Fetched: 1 row(s)
hive>
```

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?

Ans:

Row Format works well if you need to access most or all of the columns for each row in your data. It works well with transactional data or situations where many columns are requested at once. For example, retrieving every field in a record is necessary when presenting a customer's whole transaction history. Row-based storage is usually done with formats like Avro, especially for workloads that are write-heavy.

Column Format works well for analytics, particularly in workloads that require a lot of reading, and is only necessary for a subset of columns. Column-based storage enables effective retrieval by scanning only the pertinent columns when your query spans a huge dataset but only requires a small number of columns. For these kinds of situations, formats like Parquet or ORC are ideal since they provide faster reads and higher compression.

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

Ans:

Splittability determines the ability to process parts of a file independently which in turn enables parallel processing in Hadoop. Because it allows for parallel processing, in which several processors tackle different portions of the file simultaneously, this feature is especially crucial for processing huge datasets because it dramatically speeds up data operations.

Splittability improves performance in columnar formats by enabling systems to separate data by pertinent columns instead of reading entire rows. This minimizes resource utilization in distributed computing settings like Hadoop, where data must be handled across numerous nodes, and speeds up query execution by removing unnecessary data. Big files would create performance bottlenecks without splittability, resulting in longer processing times and more resource usage.

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

Ans:

Files stored in a column format can achieve better compression than those stored in a row format because similar types of data are consecutively placed in columns. Columnar formats make it easier for compression algorithms to find patterns and redundancy in the data since each column in the format only contains values of one type of data (e.g., all integers or dates in one column). When many data types (e.g., dates, numbers, and texts) are stored next to each other in a row-based format, the potential for compression is limited. In contrast, a column containing repeated values or ranges of comparable numbers can be compressed much more efficiently.

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

Ans:

When working with huge datasets, where queries usually concentrate on evaluating individual columns rather than full rows, the Parquet column file format is the ideal option. Parquet is quite effective for workloads that require a heavy reading, such as analytical queries for business intelligence, data warehousing, or machine learning applications.

Parquet's column-based data organization makes it possible for computers to read just the relevant columns, excluding irrelevant data and greatly enhancing read performance. This makes it especially helpful for large datasets with lots of columns, as analysis typically only requires a subset of the columns. parquet is a great option as well when compression and storage efficiency are top concerns. It can achieve high compression ratios while keeping rapid read performance by organizing similar data types into columns. This lowers storage expenses.