Girish Rajani
A20503736
CSP-554 Big Data Technologies
Homework 4

```
[hadoop@ip-172-31-58-208 ~]$ java TestDataGen
Magic Number = 37441
```

Figure 1 - Showing the file TestDataGen.class executed using “java TestDataGen” and generating a magic number

```
[hadoop@ip-172-31-58-208 ~]$ ls
foodplaces37441.txt  foodratings37441.txt  TestDataGen.class
[hadoop@ip-172-31-58-208 ~]$ sudo hive
```

Figure 2 - Showing two txt files created in the home directory upon executing the TestDataGen.class file

Exercise 1) 2 points

```
hive> CREATE DATABASE IF NOT EXISTS MyDb;
OK
Time taken: 2.213 seconds
hive> SHOW DATABASES;
OK
default
mydb
Time taken: 0.301 seconds, Fetched: 2 row(s)
```

Figure 3 - Creating a Hive database called “MyDb”

```
hive> USE MyDb;
OK
```

Figure 4 - Showing hive command ‘USE’ used to change the default database to MyDb

```
hive> CREATE TABLE IF NOT EXISTS foodratings (
    > name string COMMENT 'Food Name',
    > food1 int COMMENT 'first food',
    > food2 int COMMENT 'second food',
    > food3 int COMMENT 'third food',
    > food4 int COMMENT 'fourth food',
    > id int COMMENT 'restaurant ID')
    > COMMENT 'Food Ratings Table'
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 1.222 seconds
```

Figure 5 - Creating a table called foodratings as a textfile with appropriate fields, field types, and comments

```
hive> DESCRIBE FORMATTED MyDb.foodratings;
OK
# col_name              data_type               comment

name                    string                  Food Name
food1                   int                     first food
food2                   int                     second food
food3                   int                     third food
food4                   int                     fourth food
id                      int                     restaurant ID

# Detailed Table Information
Database:               mydb
Owner:                  root
CreateTime:             Thu Sep 29 18:48:45 UTC 2022
LastAccessTime:         UNKNOWN
Retention:              0
Location:               hdfs://ip-172-31-58-208.ec2.internal:8020/user/hive/ware
house/mydb.db/foodratings
Table Type:             MANAGED_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   {\"BASIC_STATS\":\"true\"}
        comment                 Food Ratings Table
        numFiles                0
        numRows                 0
        rawDataSize             0
        totalSize               0
        transient_lastDdlTime   1664477325

# 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: 0.297 seconds, Fetched: 37 row(s)
```

Figure 6 - Showing command ‘DESCRIBE FORMATTED MyDb.foodratings;’ executed to show summary format of foodratings table

```
hive> CREATE TABLE IF NOT EXISTS foodplaces (
    > id int, place string)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 0.103 seconds
```

Figure 7 - Creating a table called foodplaces as a textfile with appropriate fields, and field types

```
hive> DESCRIBE FORMATTED MyDb.foodplaces;
OK
# col_name              data_type               comment

id                      int
place                   string

# Detailed Table Information
Database:               mydb
Owner:                  root
CreateTime:             Thu Sep 29 18:51:36 UTC 2022
LastAccessTime:         UNKNOWN
Retention:              0
Location:               hdfs://ip-172-31-58-208.ec2.internal:8020/user/hive/ware
house/mydb.db/foodplaces
Table Type:             MANAGED_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   {\"BASIC_STATS\":\"true\"}
        numFiles                0
        numRows                 0
        rawDataSize             0
        totalSize               0
        transient_lastDdlTime   1664477496

# 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: 0.105 seconds, Fetched: 32 row(s)
```

Figure 8 - Showing command ‘DESCRIBE FORMATTED MyDb.foodplaces;’ executed to show summary format of foodplaces table

Exercise 2) 2 points

```
hive> load data local inpath '/home/hadoop/foodratings37441.txt' into table food
ratings;
Loading data to table mydb.foodratings
OK
Time taken: 1.098 seconds
```

Figure 9 - Loading the foodratings<magic number>.txt file created from figure 2 from the local file system into the foodratings table.

Copy of hive command to find min, max and average values of the food3 column:

SELECT MIN(food3) AS MIN_VALUE, MAX(food3) AS MAX_VALUE, AVG(food3) AS AVG_VALUE FROM foodratings;

```
hive>  SELECT MIN(food3) AS MIN_VALUE, MAX(food3) AS MAX_VALUE, AVG(food3) AS AVG_VALUE FROM foodratings;
Query ID = root_20220929185804_8ae547e5-faec-466e-ae71-b6f3d4cc6c32
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1664476392372_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: 5.50 s
----------------------------------------------------------------------------------------------
OK
1       50      25.968
Time taken: 6.266 seconds, Fetched: 1 row(s)
```

Figure 10 - Showing query output where the first value is the min, second value is the max and third value is the average of the values of the food3 column

Magic number is 37441 as shown in figure 1

Exercise 3) 2 points

Copy of hive command to find min, max and average values of the food1 column grouped by the first column ‘name’:

SELECT name, MIN(food1), MAX(food1), AVG(food1) FROM foodratings GROUP BY name;

```
hive> SELECT name, MIN(food1), MAX(food1), AVG(food1) FROM foodratings GROUP BY name;
Query ID = root_20220929190230_166c0dae-8baf-494b-9fda-5b6aa503b121
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1664476392372_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: 5.67 s
--------------------------------------------------------------------------------
OK
Jill    2       50      25.858695652173914
Joe     1       50      25.851162790697675
Joy     1       50      25.22340425531915
Mel     1       50      26.07246376811594
Sam     1       50      25.038834951456312
Time taken: 6.468 seconds, Fetched: 5 row(s)
```

Figure 11 - Showing query output of min, max and average of food1 column grouped by name

Exercise 4) 2 points

```
hive> CREATE TABLE 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: 0.121 seconds
```

Figure 12 - Creating a partitioned table called foodratingspart as a textfile with appropriate fields, and field types using column ‘name’ as the partition field

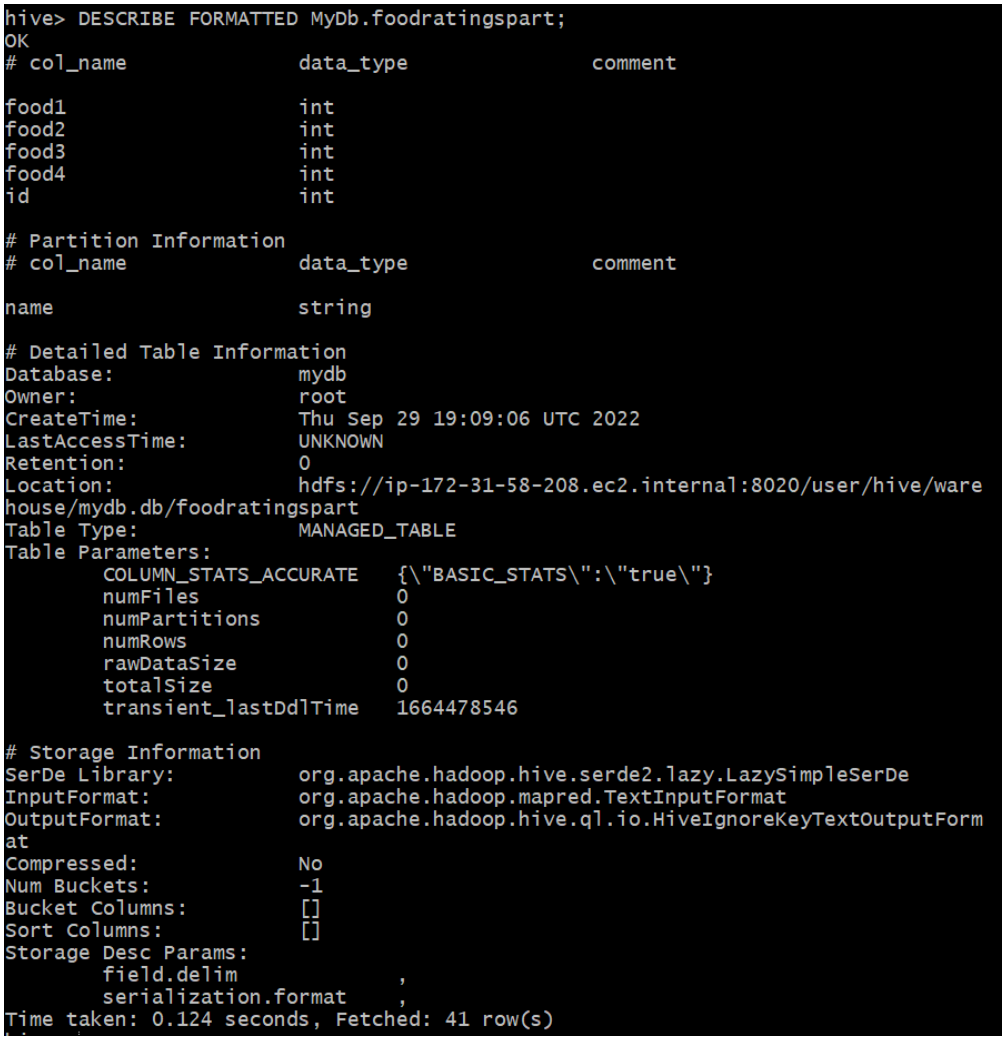

Figure 13 - Showing command 'DESCRIBE FORMATTED MyDb.foodratingspart;' executed to show summary format of foodratingspart partitioned table

Exercise 5) 2 points

Using the critic name as the partition field is a better option than the place id because since there are much fewer critic names (10) than place (10,000), the chances that a query would specify a specific critic is much higher than that of a place which will result in queries taking less time to execute.

If the place field is used as a partition, then most of the queries will only be specifying a few out of the 10,000 places, therefore making the performance gained from partitioning this way minimal. When partitioning by critic's name, the number of divisions created will be significantly less than that of places field making it much easier to manage.

.

Exercise 6) 2 points

```
hive> set hive.exec.dynamic.partition=true;
hive> set hive.exec.dynamic.partition.mode=nonstrict;
```

Figure 14 - Showing commands used to allow dynamic partition creation

```
hive> INSERT INTO foodratingspart
    > PARTITION(name)
    > SELECT food1, food2, food3, food4, id, name
    > FROM foodratings;
```

Figure 15 - Showing command used to load the 'foodratingspart' table

Copy of hive command to find min, max and average values of the food2 column of the foodratingspart table where the name is either Mel or Jill:

```
SELECT MIN(food2), MAX(food2), AVG(food2)
  > FROM foodratingspart
  > WHERE name='Mel' OR name='Jill';
```

```
hive> SELECT MIN(food2), MAX(food2), AVG(food2)
    > FROM foodratingspart
    > WHERE name='Mel' OR name='Jill';
Query ID = root_20220929195007_0d7b8e61-0c69-43b1-89f5-43a0aeb597e9
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1664476392372_0004)

--------------------------------------------------------------------------------
        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: 5.89 s
--------------------------------------------------------------------------------
OK
1       50      27.485933503836318
Time taken: 6.708 seconds, Fetched: 1 row(s)
```

Figure 16 - Showing query output of min, max and average of food2 column in the foodratingspart table where the name is either Mel or Jill

Exercise 7) 2 points

```
hive> load data local inpath '/home/hadoop/foodplaces37441.txt' into table foodplaces;
Loading data to table mydb.foodplaces
OK
Time taken: 0.945 seconds
```

Figure 17 - Loading the foodplaces<magic number>.txt file created from figure 2 from the local file system into the foodplaces table.

Copy of hive command used to join foodplaces and foodratings and to provide the average rating for field food4 for the restaurant Super Bowl

```
SELECT foodplaces.place, AVG(foodratings.food4)
  > FROM foodratings
  > INNER JOIN foodplaces ON foodratings.id=foodplaces.id
  > WHERE place='Soup Bowl'
  > GROUP BY place;
```

```
hive> SELECT foodplaces.place, AVG(foodratings.food4)
    > FROM foodratings
    > INNER JOIN foodplaces ON foodratings.id=foodplaces.id
    > WHERE place='Soup Bowl'
    > GROUP BY place;
Query ID = root_20220929202805_0a49d53a-0988-43f5-9b97-d669eacf9db0
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1664476392372_0006)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Map 3 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      2          2        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 11.45 s
--------------------------------------------------------------------------------
OK
Soup Bowl       24.76923076923077
Time taken: 12.132 seconds, Fetched: 1 row(s)
```

Figure 18 - Showing output and command used to join foodplaces and foodratings and to provide the average rating for field food4 for the restaurant Super Bowl

Exercise 8) 4 points

a. The most important consideration when choosing a row format is when you need to access one or more records with most or all the columns. This means that if you are writing queries that will access some records and you want most/all the columns of each record, then row format is recommended. When considering a column format, it should be used when you need to analyze and perform computation on specific columns.
b. “Splittability” for a column file format means decomposing the big data sets into smaller chunks and storing them in columns. This enables parallel processing, which means splitting the data across multiple machines. Additionally, splittability for a column file format is possible if queries are related to one column at a time, such as finding the min/max of a specific column.
c. Files stored in column format can achieve better compression rates than those stored in row format. This is because by storing columns of data of the same type next to each other, there is more efficient compression and reduced workload than if you were storing data in row format. Therefore, the files can skip unnecessary data that it does not need to know and simply bypass an entire row when stored in column format to reduce computation times.
d. The “Parquet” column file format would be the best choice to use when analyzing large datasets that contain many columns and running read heavy tasks. Additionally, when you need the data store to support schema evolution, “Parquet” would be the best choice to use since it is supported.