SHRADDHA PATEL A20499171

CSP554—Big Data Technologies

Assignment #7

Worth: 12 points (2 points for each problem)

Readings:

The mid-term will assume you have read at least the high points of the below. In "Free Books and Chapters"

- Make sure to read/skim "Spark - The Definitive Guide (Excerpts)"
- Make sure to read/skim "Spark - Python API - SQL & DataFrames"
- Also take a look at the two Spark Cheat Sheets

For this assignment you will be using your Hadoop environment including the pyspark CLI.

Some basic notes:

- We will again be using files generated by the program TestDataGen. But even though the files this program generates end is the '.txt' suffix, I want you to treat them as if they were (comma separated) '.csv' files.
- In fact, if you like, when you copy them to HDFS you can change their suffixes from '.txt' to '.csv'. But this is not necessary to complete the exercises.

Demos

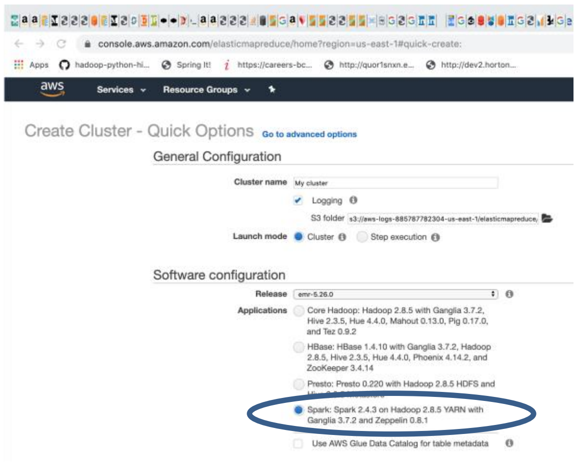
To sets of demo files have been included in this assignment. It is a good idea to look at and execute them before trying the following exercises.

1. One set of demo files provide examples of the use of RDDs. The instructions are in pydemo.txt and the demo files themselves are in pyspark.zip. Use scp to copy pyspark.zip to your EMR master node (to /home/hadoop) and then follow the steps in pydemo.txt
2. Another set of demo files provide examples of the use of DataFrames. The instructions are in dfdemo.txt and the demo files themselves are in sparkdf.zip. Use scp to copy sparkdf.zip to your EMR master node (to /home/hadoop) and then follow the steps in dfdemo.txt

Exercise 1)

Step A

Start up a Hadoop cluster as previously, but instead of choosing the "Core Hadoop" configuration chose the "Spark" configuration (see below), otherwise proceed as before.



console.aws.amazon.com/elasticmapreduce/home?region=us-east-1#quick-create:

Create Cluster - Quick Options

Go to advanced options

General Configuration

Cluster name: My cluster

[✓] Logging

S3 folder: s3://aws-logs-885787782304-us-east-1/elasticmapreduce/

Launch mode: [✓ Cluster] Step execution

Software configuration

Release: emr-5.26.0

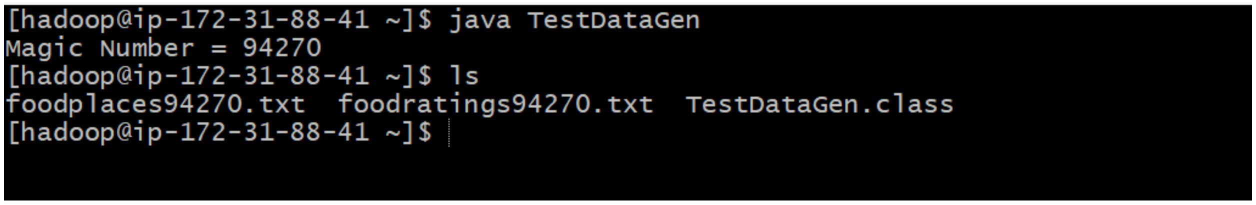
Applications:

- Core Hadoop: Hadoop 2.8.5 with Ganglia 3.7.2, Hive 2.3.5, Hue 4.4.0, Mahout 0.13.0, Pig 0.17.0, and Tez 0.9.2
- HBase: HBase 1.4.10 with Ganglia 3.7.2, Hadoop 2.8.5, Hive 2.3.5, Hue 4.4.0, Phoenix 4.14.2, and ZooKeeper 3.4.14
- Presto: Presto 0.220 with Hadoop 2.8.5 HDFS and Hive 2.3.5 Metastore
- [✓ Spark: Spark 2.4.3 on Hadoop 2.8.5 YARN with Ganglia 3.7.2 and Zeppelin 0.8.1]

[ ] Use AWS Glue Data Catalog for table metadata

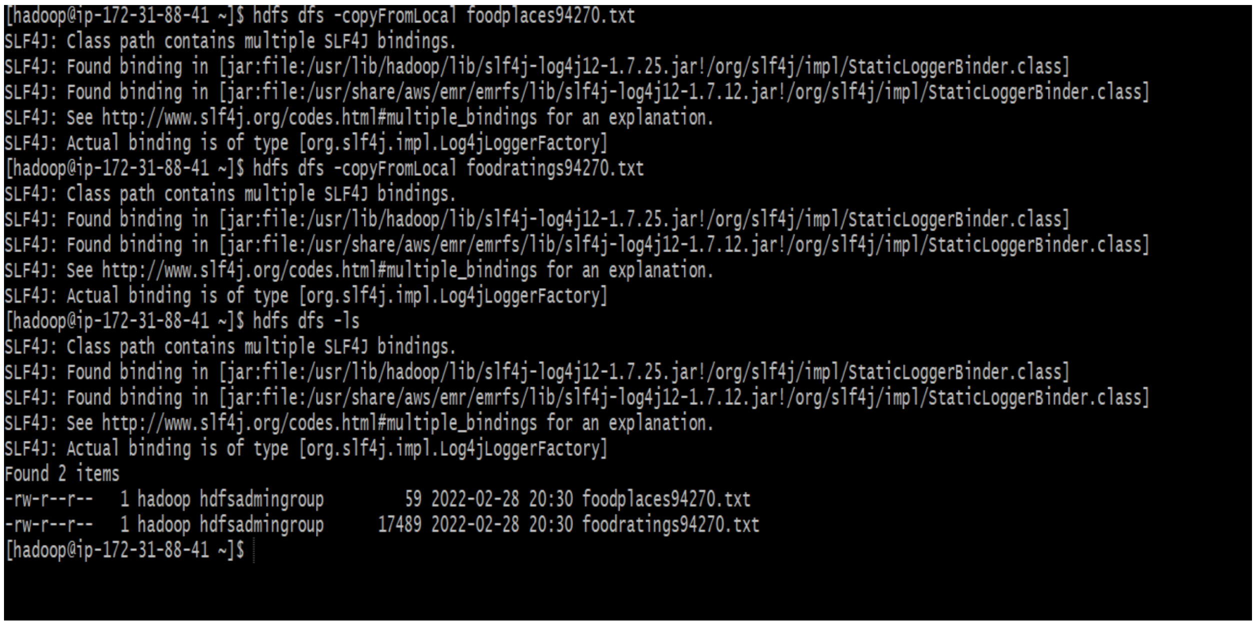
Step B

Use the TestDataGen program from previous assignments to generate new data files.



```
[hadoop@ip-172-31-88-41 ~]$ java TestDataGen
Magic Number = 94270
[hadoop@ip-172-31-88-41 ~]$ ls
foodplaces94270.txt  foodratings94270.txt  TestDataGen.class
[hadoop@ip-172-31-88-41 ~]$
```

Copy both generated files to the HDFS directory “/user/hadoop”



```
[hadoop@ip-172-31-88-41 ~]$ hdfs dfs -copyFromLocal foodplaces94270.txt
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hadoop/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/aws/emr/emrfs/lib/slf4j-log4j12-1.7.12.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
[hadoop@ip-172-31-88-41 ~]$ hdfs dfs -copyFromLocal foodratings94270.txt
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hadoop/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/aws/emr/emrfs/lib/slf4j-log4j12-1.7.12.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
[hadoop@ip-172-31-88-41 ~]$ hdfs dfs -ls
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hadoop/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/aws/emr/emrfs/lib/slf4j-log4j12-1.7.12.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 2 items
-rw-r--r--   1 hadoop hdfsadmingroup         59 2022-02-28 20:30 foodplaces94270.txt
-rw-r--r--   1 hadoop hdfsadmingroup      17489 2022-02-28 20:30 foodratings94270.txt
[hadoop@ip-172-31-88-41 ~]$
```

Step C

Load the ‘foodratings’ file as a ‘csv’ file into a DataFrame called foodratings. When doing so specify a schema having fields of the following names and types:

Field Name	Field Type
name	String
food1	Integer
food2	Integer
food3	Integer
food4	Integer
placeid	Integer

As the results of this exercise provide the magic number, *the code you execute* and screen shots of the following commands:

foodratings.printSchema()

foodratings.show(5)

Magic Number: 94270

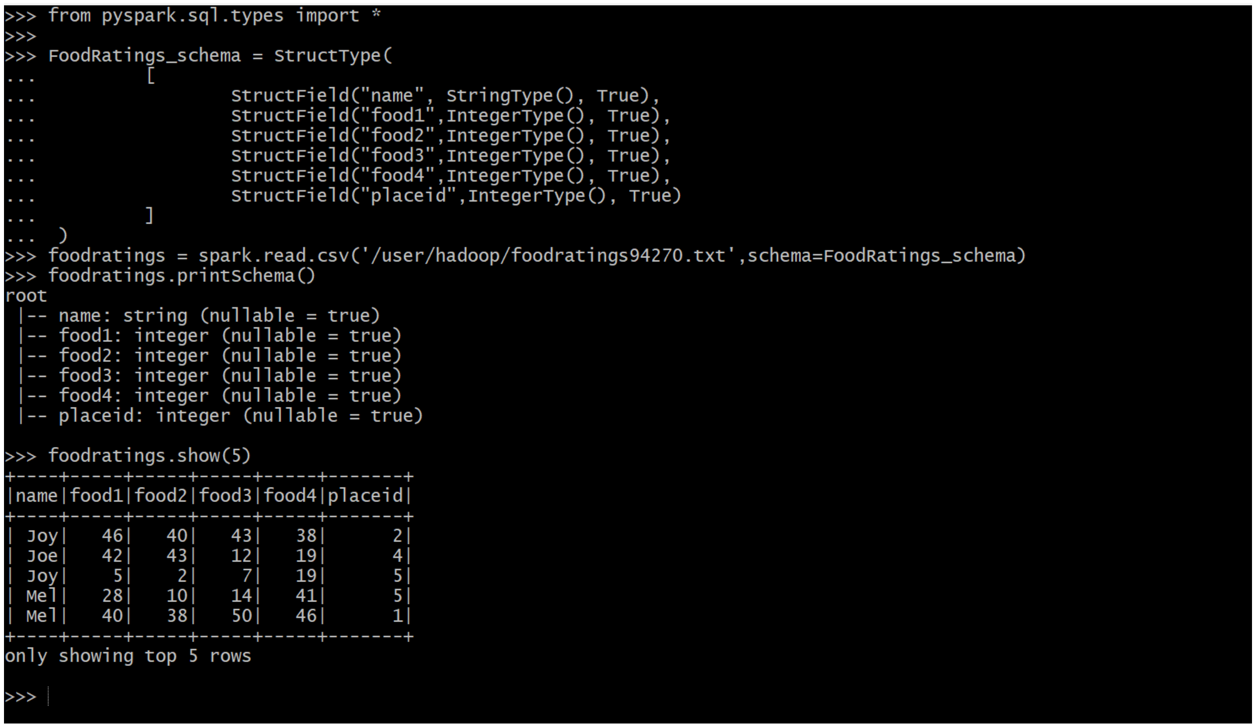
Code:

```
from pyspark.sql.types import *

FoodRatings_schema = StructType(
         [
                 StructField("name", StringType(), True),
                 StructField("food1",IntegerType(), True),
                 StructField("food2",IntegerType(), True),
                 StructField("food3",IntegerType(), True),
                 StructField("food4",IntegerType(), True),
                 StructField("placeid",IntegerType(), True)
         ]
 )

foodratings =
spark.read.csv('/user/hadoop/foodratings94270.txt',schema=FoodRatings_schema)
foodratings.printSchema()
foodratings.show(5)
```

OUTPUT:



```
>>> from pyspark.sql.types import *
>>>
>>> FoodRatings_schema = StructType(
...         [
...                 StructField("name", StringType(), True),
...                 StructField("food1",IntegerType(), True),
...                 StructField("food2",IntegerType(), True),
...                 StructField("food3",IntegerType(), True),
...                 StructField("food4",IntegerType(), True),
...                 StructField("placeid",IntegerType(), True)
...         ]
...  )
>>> foodratings = spark.read.csv('/user/hadoop/foodratings94270.txt',schema=FoodRatings_schema)
>>> foodratings.printSchema()
root
 |-- name: string (nullable = true)
 |-- food1: integer (nullable = true)
 |-- food2: integer (nullable = true)
 |-- food3: integer (nullable = true)
 |-- food4: integer (nullable = true)
 |-- placeid: integer (nullable = true)

>>> foodratings.show(5)
+----+-----+-----+-----+-----+-------+
|name|food1|food2|food3|food4|placeid|
+----+-----+-----+-----+-----+-------+
| Joy|   46|   40|   43|   38|      2|
| Joe|   42|   43|   12|   19|      4|
| Joy|    5|    2|    7|   19|      5|
| Mel|   28|   10|   14|   41|      5|
| Mel|   40|   38|   50|   46|      1|
+----+-----+-----+-----+-----+-------+
only showing top 5 rows

>>>
```

Exercise 2)

Load the ‘foodplaces’ file as a ‘csv’ file into a DataFrame called foodplaces. When doing so specify a schema having fields of the following names and types:

Field Nampee	Field Type
placeid	Integer
placename	String

As the results of this exercise provide *the code you execute* and screen shots of the following commands:

foodratings.printSchema()

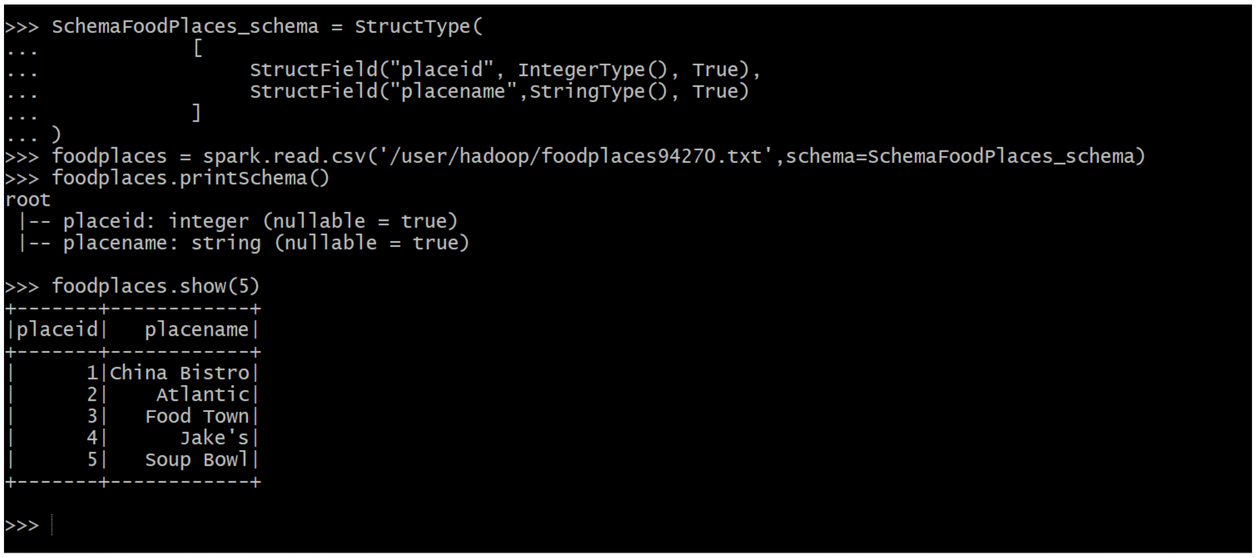
foodratings.show(5)

code:

```
SchemaFoodPlaces_schema = StructType(
            [
                StructField("placeid", IntegerType(), True),
                StructField("placename",StringType(), True)
            ]
)

foodplaces =
spark.read.csv('/user/hadoop/foodplaces94270.txt',schema=SchemaFoodPlaces_schema)
foodplaces.printSchema()
foodplaces.show(5)
```

OUTPUT:



```
>>> SchemaFoodPlaces_schema = StructType(
...             [
...                 StructField("placeid", IntegerType(), True),
...                 StructField("placename",StringType(), True)
...             ]
... )
>>> foodplaces = spark.read.csv('/user/hadoop/foodplaces94270.txt',schema=SchemaFoodPlaces_schema)
>>> foodplaces.printSchema()
root
 |-- placeid: integer (nullable = true)
 |-- placename: string (nullable = true)

>>> foodplaces.show(5)
+-------+------------+
|placeid|   placename|
+-------+------------+
|      1|China Bistro|
|      2|    Atlantic|
|      3|   Food Town|
|      4|      Jake's|
|      5|   Soup Bowl|
+-------+------------+

>>>
```

Exercise 3)

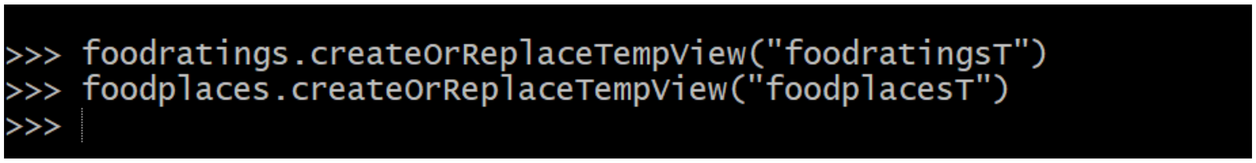
<u>Step A</u>

Register the DataFrames created in exercise 1 and 2 as tables called “foodratingsT” and “foodplacesT”

Code:

```
foodratings.createOrReplaceTempView("foodratingsT")
foodplaces.createOrReplaceTempView("foodplacesT")
```

OUTPUT:



```
>>> foodratings.createOrReplaceTempView("foodratingsT")
>>> foodplaces.createOrReplaceTempView("foodplacesT")
>>>
```

<u>Step B</u>

Use a SQL query on the table “foodratingsT” to create a new DataFrame called foodratings_ex3a holding records which meet the following condition: food2 < 25 and food4 > 40. Remember, when defining conditions in your code use maximum parentheses.

As the results of this step *provide the code you execute* and screen shots of the following commands:

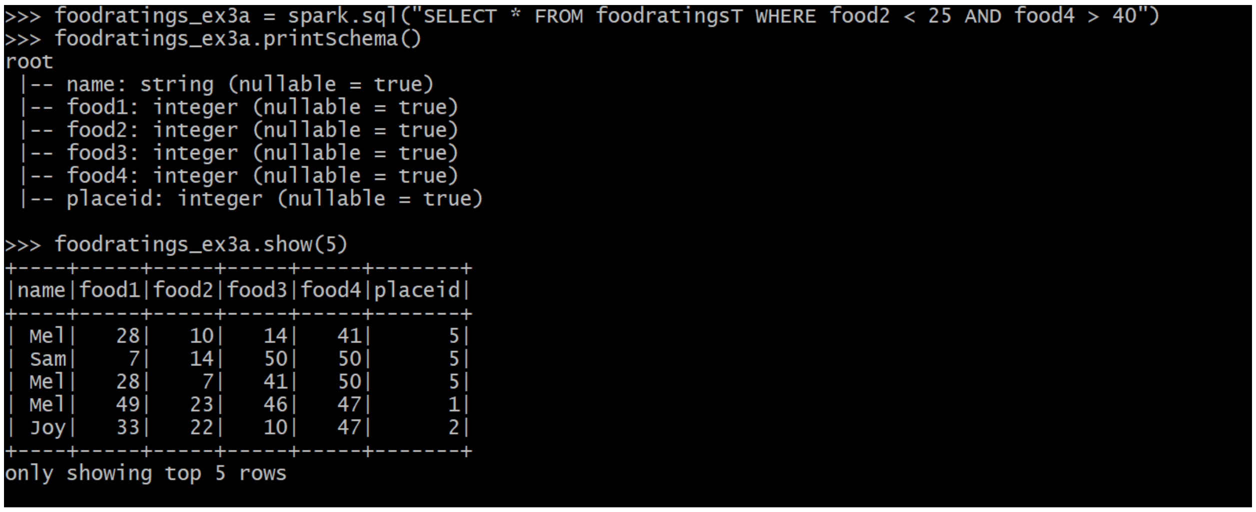
foodratings_ex3a.printSchema()

foodratings_ex3a.show(5)

Code:

```
foodratings_ex3a = spark.sql("SELECT * FROM foodratingsT WHERE food2 < 25 AND
food4 > 40")
foodratings_ex3a.printSchema()
foodratings_ex3a.show(5)
```

OUTPUT:



```
>>> foodratings_ex3a = spark.sql("SELECT * FROM foodratingsT WHERE food2 < 25 AND food4 > 40")
>>> foodratings_ex3a.printSchema()
root
 |-- name: string (nullable = true)
 |-- food1: integer (nullable = true)
 |-- food2: integer (nullable = true)
 |-- food3: integer (nullable = true)
 |-- food4: integer (nullable = true)
 |-- placeid: integer (nullable = true)

>>> foodratings_ex3a.show(5)
+----+-----+-----+-----+-----+-------+
|name|food1|food2|food3|food4|placeid|
+----+-----+-----+-----+-----+-------+
| Mel|   28|   10|   14|   41|      5|
| Sam|    7|   14|   50|   50|      5|
| Mel|   28|    7|   41|   50|      5|
| Mel|   49|   23|   46|   47|      1|
| Joy|   33|   22|   10|   47|      2|
+----+-----+-----+-----+-----+-------+
only showing top 5 rows
```

Step C

Use a SQL query on the table “foodplacesT” to create a new DataFrame called foodplaces_ex3b holding records which meet the following condition: placeid > 3

As the results of this step *provide the code you execute* and screen shots of the following commands:

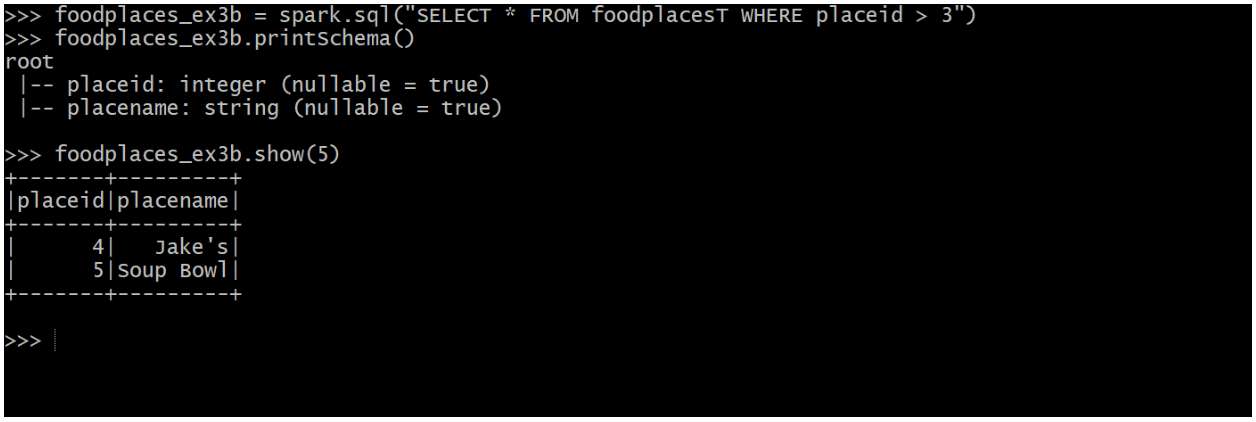
foodplaces_ex3b.printSchema()

foodplaces_ex3b.show(5)

Code:

```
foodplaces_ex3b = spark.sql("SELECT * FROM foodplacesT WHERE placeid > 3")
foodplaces_ex3b.printSchema()
foodplaces_ex3b.show(5)
```

OUTPUT:



```
>>> foodplaces_ex3b = spark.sql("SELECT * FROM foodplacesT WHERE placeid > 3")
>>> foodplaces_ex3b.printSchema()
root
 |-- placeid: integer (nullable = true)
 |-- placename: string (nullable = true)

>>> foodplaces_ex3b.show(5)
+-------+---------+
|placeid|placename|
+-------+---------+
|      4|   Jake's|
|      5|Soup Bowl|
+-------+---------+

>>>
```

Exercise 4)

Use a transformation (not a SparkSQL query) on the DataFrame ‘foodratings’ created in exercise 1 to create a new DataFrame called foodratings_ex4 that includes only those records (rows) where the ‘name’ field is “Mel” and food3 < 25.

As the results of this step provide the code you execute and screen shots of the following commands:

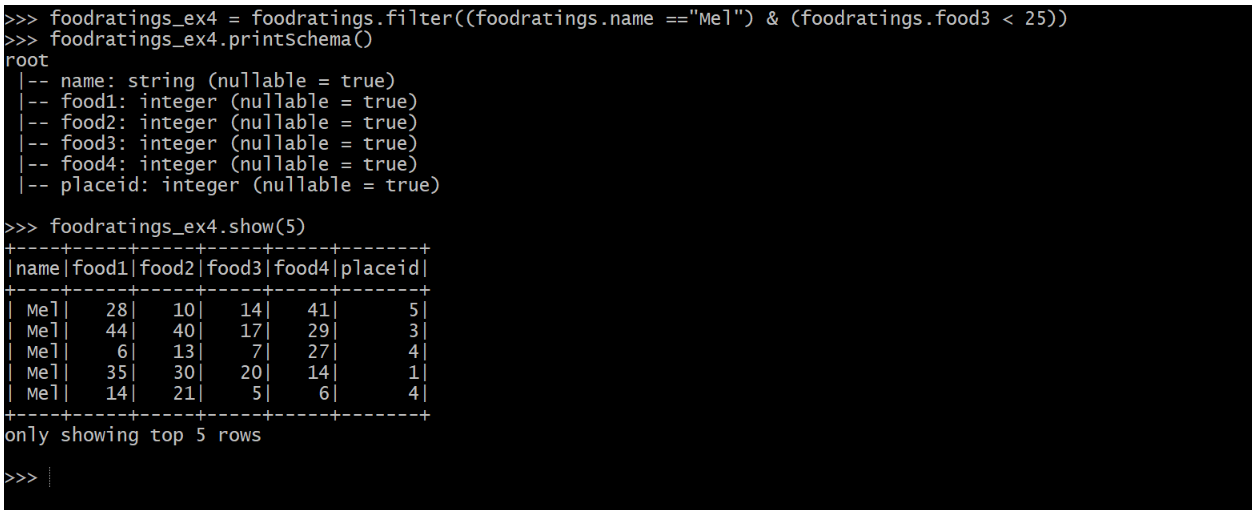
foodratings_ex4.printSchema()

foodratings_ex4.show(5)

Code:

```
foodratings_ex4 = foodratings.filter((foodratings.name =="Mel") &
(foodratings.food3 < 25))
foodratings_ex4.printSchema()
foodratings_ex4.show(5)
```

OUTPUT:



```
>>> foodratings_ex4 = foodratings.filter((foodratings.name =="Mel") & (foodratings.food3 < 25))
>>> foodratings_ex4.printSchema()
root
 |-- name: string (nullable = true)
 |-- food1: integer (nullable = true)
 |-- food2: integer (nullable = true)
 |-- food3: integer (nullable = true)
 |-- food4: integer (nullable = true)
 |-- placeid: integer (nullable = true)

>>> foodratings_ex4.show(5)
+----+-----+-----+-----+-----+-------+
|name|food1|food2|food3|food4|placeid|
+----+-----+-----+-----+-----+-------+
| Mel|   28|   10|   14|   41|      5|
| Mel|   44|   40|   17|   29|      3|
| Mel|    6|   13|    7|   27|      4|
| Mel|   35|   30|   20|   14|      1|
| Mel|   14|   21|    5|    6|      4|
+----+-----+-----+-----+-----+-------+
only showing top 5 rows

>>>
```

Exercise 5)

Use a transformation (**not a SparkSQL query**) on the DataFrame ‘foodratings’ created in exercise 1 to create a new DataFrame called foodratings_ex5 that includes only the columns (fields) ‘name’ and ‘placeid’

As the results of this step provide the code you execute and screen shots of the following commands:

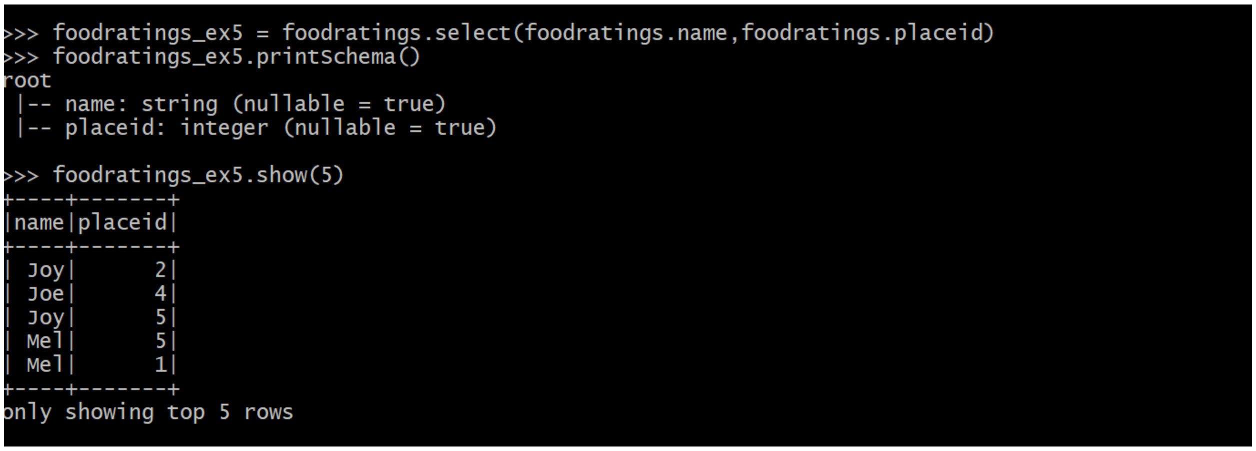
foodratings_ex5.printSchema()

foodratings_ex5.show(5)

Code:

```
foodratings_ex5 = foodratings.select(foodratings.name,foodratings.placeid)
foodratings_ex5.printSchema()
foodratings_ex5.show(5)
```

OUTPUT:



```
>>> foodratings_ex5 = foodratings.select(foodratings.name,foodratings.placeid)
>>> foodratings_ex5.printSchema()
root
 |-- name: string (nullable = true)
 |-- placeid: integer (nullable = true)

>>> foodratings_ex5.show(5)
+----+-------+
|name|placeid|
+----+-------+
| Joy|      2|
| Joe|      4|
| Joy|      5|
| Mel|      5|
| Mel|      1|
+----+-------+
only showing top 5 rows
```

Exercise 6)

Use a transformation (**not a SparkSQL query**) to create a new DataFrame called ex6 which is the inner join, on placeid, of the DataFrames 'foodratings' and 'foodplaces' created in exercises 1 and 2

As the results of this step provide the code you execute and screen shots of the following commands:

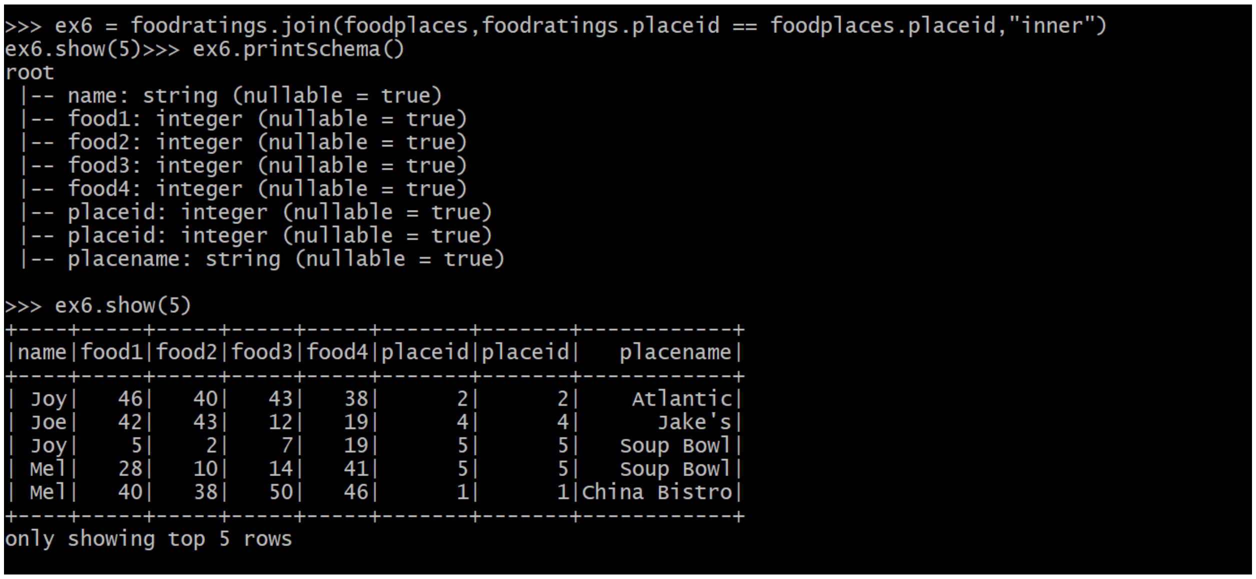
ex6.printSchema()

ex6.show(5)

Code:

```
ex6 = foodratings.join(foodplaces,foodratings.placeid ==
foodplaces.placeid,"inner")
ex6.printSchema()
ex6.show(5)
```

OUTPUT:



```
>>> ex6 = foodratings.join(foodplaces,foodratings.placeid == foodplaces.placeid,"inner")
ex6.show(5)>>> ex6.printSchema()
root
 |-- name: string (nullable = true)
 |-- food1: integer (nullable = true)
 |-- food2: integer (nullable = true)
 |-- food3: integer (nullable = true)
 |-- food4: integer (nullable = true)
 |-- placeid: integer (nullable = true)
 |-- placeid: integer (nullable = true)
 |-- placename: string (nullable = true)

>>> ex6.show(5)
+----+-----+-----+-----+-----+-------+-------+------------+
|name|food1|food2|food3|food4|placeid|placeid|   placename|
+----+-----+-----+-----+-----+-------+-------+------------+
| Joy|   46|   40|   43|   38|      2|      2|    Atlantic|
| Joe|   42|   43|   12|   19|      4|      4|      Jake's|
| Joy|    5|    2|    7|   19|      5|      5|   Soup Bowl|
| Mel|   28|   10|   14|   41|      5|      5|   Soup Bowl|
| Mel|   40|   38|   50|   46|      1|      1|China Bistro|
+----+-----+-----+-----+-----+-------+-------+------------+
only showing top 5 rows
```