lab2_4

```
import org.apache.spark.sql.SQLContext

val sqlContext = new SQLContext(sc)

val df = sqlContext.read.json("/home/akash/kth/data_intensive/labs/id2221/lab2/data/pe
df.show()
```

Took: 4 seconds 869 milliseconds, at 2016-9-29 19:44

```
df.printSchema
```

Took: 1 second 444 milliseconds, at 2016-9-29 19:44

```
df.select("name").show()
```

Took: 1 second 302 milliseconds, at 2016-9-29 19:44

```
// Select everybody, but increment the age by 1
df.select($"name",($"age")+1).show()
```

Took: 1 second 402 milliseconds, at 2016-9-29 19:44

```
// Select people older than 21
df.filter($"age">21).show()

// Count people by age
df.groupBy("age").count().show()
```

Took: 4 seconds 210 milliseconds, at 2016-9-29 19:44

```
df.registerTempTable("df")
sqlContext.sql("select * from df").show()
```

Took: 1 second 695 milliseconds, at 2016-9-29 19:44

```
// TODO: Replace <FILL IN> with appropriate code
val sqlC = new org.apache.spark.sql.SQLContext(sc)
import sqlC.implicits._
case class Person(name: String, age: Int)

// Load a text file and convert each line to a Row.
val lines = sc.textFile("/home/akash/kth/data_intensive/labs/id2221/lab2/data/people/p
val parts = lines.map(l => l.split(","))
val people = parts.map(p => Person(p(0), p(1).trim.toInt))


// Infer the schema, and register the DataFrame as a table.
val schemaPeople = people.toDF()
schemaPeople.registerTempTable("people")

// SQL can be run over DataFrames that have been registered as a table. Complete the f
// to return teenagers, i.e., age >= 13 and age <= 19.
val teenagers = sqlContext.sql("SELECT name FROM people WHERE age >=13 and aGE<=19")

// The results of SQL queries are DataFrames and support all the normal RDD operations
// The columns of a row in the result can be accessed by field index:
teenagers.map(t => "Name: " + t(0)).collect().foreach(println)

// or by field name:
teenagers.map(t => "Name: " + t.getAs("name").toString).collect().foreach(println)
```

Took: 5 seconds 56 milliseconds, at 2016-9-29 19:44

```
// Just run this code
// Create an RDD
val people = sc.textFile("/home/akash/kth/data_intensive/labs/id2221/lab2/data/people/

// The schema is encoded in a string
val schemaString = "name age"

// Import Row.
import org.apache.spark.sql.Row;

// Import Spark SQL data types
import org.apache.spark.sql.types.{StructType,StructField,StringType};

// Generate the schema based on the string of schema
val schema =
  StructType(
    schemaString.split(" ").map(fieldName => StructField(fieldName, StringType, true))

// Convert records of the RDD (people) to Rows.
val rowRDD = people.map(_.split(",")).map(p => Row(p(0), p(1).trim))

// Apply the schema to the RDD.
val peopleDataFrame = sqlContext.createDataFrame(rowRDD, schema)

// Register the DataFrames as a table.
peopleDataFrame.registerTempTable("people")

// SQL statements can be run by using the sql methods provided by sqlContext.
val results = sqlContext.sql("SELECT name FROM people")

// The results of SQL queries are DataFrames and support all the normal RDD operations
// The columns of a row in the result can be accessed by field index or by field name.
results.map(t => "Name: " + t(0)).collect().foreach(println)
```

Took: 2 seconds 283 milliseconds, at 2016-9-29 19:44

```
// Just run this code
// Load data from a parquet file
val df = sqlContext.read.load("/home/akash/kth/data_intensive/labs/id2221/lab2/data/pe
df.select("name", "favorite_color").write.mode("overwrite").save("namesAndFavColors.pa

// Manually specify the data source type, e.g., json, parquet, jdbc.
val jdf = sqlContext.read.format("json").load("/home/akash/kth/data_intensive/labs/id2
jdf.select("name", "age").write.format("parquet").mode("overwrite").save("namesAndAges
```

Took: 2 seconds 348 milliseconds, at 2016-9-29 19:44

```
// Just run this code
// The RDD is implicitly converted to a DataFrame by implicits, allowing it to be stor
schemaPeople.write.parquet("people.parquet")

// Read in the parquet file created above.  Parquet files are self-describing so the s
// The result of loading a Parquet file is also a DataFrame.
val parquetFile = sqlContext.read.parquet("people.parquet")

//Parquet files can also be registered as tables and then used in SQL statements.
parquetFile.registerTempTable("parquetFile")
val teenagers = sqlContext.sql("SELECT name FROM parquetFile WHERE age >= 13 AND age <=
teenagers.map(t => "Name: " + t(0)).collect().foreach(println)
```

Took: 1 second 826 milliseconds, at 2016-9-29 19:44

```
// Just run this code
// A JSON dataset is pointed to by path.
// The path can be either a single text file or a directory storing text files.
val people = sqlContext.read.json("/home/akash/kth/data_intensive/labs/id2221/lab2/data

// The inferred schema can be visualized using the printSchema() method.
people.printSchema()
// root
//  |-- age: integer (nullable = true)
//  |-- name: string (nullable = true)

// Register this DataFrame as a table.
people.registerTempTable("people")

// SQL statements can be run by using the sql methods provided by sqlContext.
val teenagers = sqlContext.sql("SELECT name FROM people WHERE age >= 13 AND age <= 19"

// Alternatively, a DataFrame can be created for a JSON dataset represented by
// an RDD[String] storing one JSON object per string.
val anotherPeopleRDD = sc.parallelize(
  """{"name":"Yin","address":{"city":"Columbus","state":"Ohio"}}""" :: Nil)
val anotherPeople = sqlContext.read.json(anotherPeopleRDD)
```

Took: 820 milliseconds, at 2016-9-29 19:44

```

```

Took: 939 milliseconds, at 2016-9-29 19:44

Build: | **buildTime**-*Sun Sep 25 12:54:11 UTC 2016* | **formattedShaVersion**-*0.7.0-SNAPSHOT-f6cd60a95cfc19cbae80285f187da9baec04e436* | **sbtVersion**-*0.13.8* | **scalaVersion**-*2.10.6* | **sparkNotebookVersion**-*0.7.0-SNAPSHOT* | **hadoopVersion**-*2.2.0* | **jets3tVersion**-*0.7.1* | **jlineDef**-*(org.scala-lang,2.10.6)* | **sparkVersion**-*2.0.0* | **withHive**-*false* |.