



Dataframes & Spark SQL



Spark SQL

Spark module
for
Structured Data Processing

Spark SQL

Integrated

- Provides **DataFrames**
- Mix SQL queries & Spark programs

Spark SQL

Uniform Data Access

- Source:
 - HDFS,
 - Hive
 - Relational Databases
- Avro, Parquet, ORC, JSON
- You can even join data across these sources.
- Hive Compatibility
- Standard Connectivity

DataFrames

RDD

| |
|-----------|
| 1 sandeep |
| 2 ted |
| 3 thomas |
| 4 priya |
| 5 kush |

Unstructured

Need code for processing

DataFrames

RDD

| | |
|---|---------|
| 1 | sandeep |
| 2 | ted |
| 3 | thomas |
| 4 | priya |
| 5 | kush |

Unstructured

Need code for processing

Data Frame

| ID | Name |
|----|---------|
| 1 | sandeep |
| 2 | ted |
| 3 | thomas |
| 4 | priya |
| 5 | kush |

Structured

Can use SQL or R like syntax:
df.sql("select Id where name = 'priya'")

head(where(df, df\$ID > 2))

Data Frames

| col1 | col2 | col3 | Partition 1 |
|------|------|------|-------------|
| | | | |
| | | | Partition 2 |
| | | | |
| | | | |

- Collection with named columns
- Distributed
- $\langle \rangle$ Same as database table
- $\langle \rangle$ A data frame in R/Python

Data Frames

Structured data:
CSV, JSON

Hive

RDBMS

RDDs

Can be constructed from

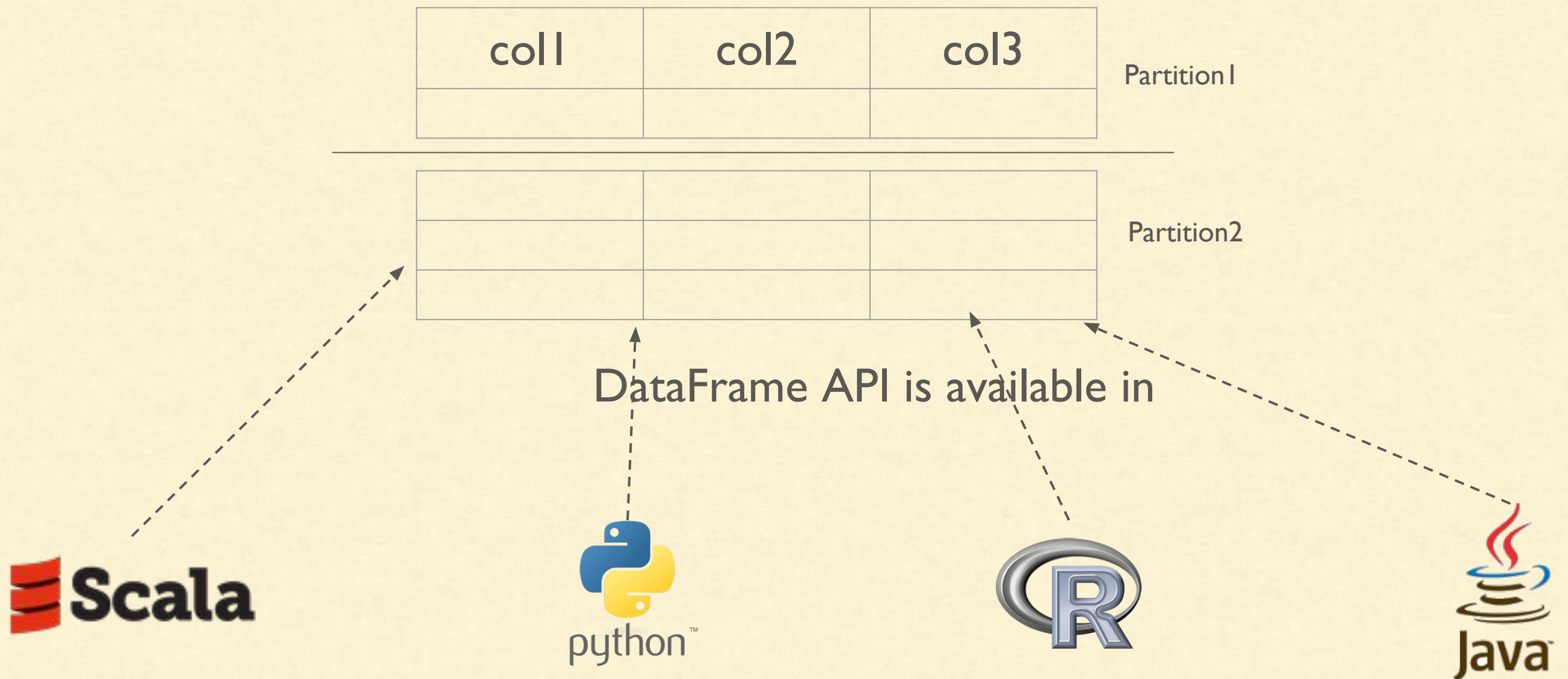
| col1 | col2 | col3 |
|------|------|------|
| | | |

Partition1

| | | |
|--|--|--|
| | | |
| | | |
| | | |

Partition2

Data Frames



Getting Started

- Available in Spark 2.0x onwards.
- Using usual interfaces
 - Spark-shell
 - Spark Application
 - Pyspark
 - Java
 - etc.

Getting Started

```
$ export HADOOP_CONF_DIR=/etc/hadoop/conf/  
$ export YARN_CONF_DIR=/etc/hadoop/conf/
```

Getting Started

```
$ export HADOOP_CONF_DIR=/etc/hadoop/conf/
```

```
$ export YARN_CONF_DIR=/etc/hadoop/conf/
```

```
$ ls /usr/
```

```
bin  games  include  jdk64  lib64    local  share  spark1.6  
spark2.0.2  tmp  etc  hdp    java    lib    libexec  sbin  spark1.2.1  
spark2.0.1  src
```

Getting Started

```
$ export HADOOP_CONF_DIR=/etc/hadoop/conf/
```

```
$ export YARN_CONF_DIR=/etc/hadoop/conf/
```

```
$ ls /usr/
```

```
bin  games  include  jdk64  lib64      local  share      spark1.6  
spark2.0.2  tmp  etc  hdp    java      lib    libexec  sbin    spark1.2.1  
spark2.0.1  src
```

```
$ /usr/spark2.0.2/bin/spark-shell
```

Getting Started

Spark context Web UI available at <http://172.31.60.179:4040>

```
Spark context available as 'sc' (master = local[*], app id = local-1498489557917).
```

```
Spark session available as 'spark'.
```

Welcome to

[illegible]

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_91)

Type in expressions to have them evaluated.

Type :help for more information.

```
scala>
```

Getting Started

Spark context Web UI available at <http://172.31.60.179:4040>

```
Spark context available as 'sc' (master = local[*], app id = local-1498489557917).
```

```
Spark session available as 'spark'.
```

Welcome to

```
/ _ /  
 \ V _ V _ ` / _ /  
/_/ . _/\_,_/_/_/_/_/  
/_/
```

version 2.0.2

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_91)

Type in expressions to have them evaluated.

Type :help for more information.

```
scala>
```

Starting Point: SparkSession

```
import org.apache.spark.sql.SparkSession

val spark = SparkSession
  .builder()
  .appName("Spark SQL basic example")
  .config("spark.some.config.option", "some-value")
  .getOrCreate()
```

Starting Point: SparkSession

```
import org.apache.spark.sql.SparkSession

val spark = SparkSession
  .builder()
  .appName("Spark SQL basic example")
  .config("spark.some.config.option", "some-value")
  .getOrCreate()

//For implicit conversions, e.g. RDDs to DataFrames
import spark.implicits._
```

Creating DataFrames from JSON

In web console or ssh:

```
$ hadoop fs -cat /data/spark/people.json
```

```
{"name": "Michael"}
```

```
{"name": "Andy", "age": 30}
```

```
{"name": "Justin", "age": 19}
```

Creating DataFrames from JSON

```
var df = spark.read.json("/data/spark/people.json")

// Displays the content of the DataFrame to stdout
df.show()
```

```
scala> df.show()
```

```
+-----+-----+
|  age |   name |
+-----+-----+
| null | Michael |
|   30 |    Andy |
|   19 |   Justin |
+-----+-----+
```

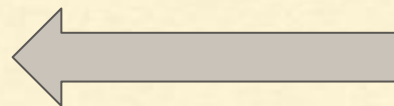
Creating DataFrames from JSON

```
var df = spark.read.json("/data/spark/people.json")

# Displays the content of the DataFrame to stdout
df.show()
```

```
scala> df.show()
```

```
+-----+-----+
|  age  |  name  |
+-----+-----+
| null  | Michael|
|   30  |   Andy |
|   19  |  Justin|
+-----+-----+
```



Original JSON:

```
{"name": "Michael"}
{"name": "Andy", "age": 30}
{"name": "Justin", "age": 19}
```


DataFrame Operations

```
# Print the schema in a tree format  
df.printSchema()
```

```
root
```

```
 |-- age: long (nullable = true)  
 |-- name: string (nullable = true)
```

```
{"name": "Michael"}  
{"name": "Andy", "age": 30}  
{"name": "Justin", "age": 19}
```

DataFrame Operations

```
# Select only the "name" column  
df.select("name").show()
```

```
+-----+  
|  name  |  
+-----+  
|Michael|  
|  Andy  |  
| Justin |  
+-----+
```

```
{"name": "Michael"}  
{"name": "Andy", "age": 30}  
{"name": "Justin", "age": 19}
```

DataFrame Operations

```
# Increment the age by 1
df.select($"name", $"age" + 1).show()
```

```
+-----+-----+
|  name | (age + 1) |
+-----+-----+
|Michael|      null |
|  Andy |       31 |
| Justin|       20 |
+-----+-----+
```

```
{"name": "Michael"}
{"name": "Andy", "age": 30}
{"name": "Justin", "age": 19}
```

DataFrame Operations

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

```
+---+-----+
|age|name|
+---+-----+
| 30|Andy|
+---+-----+
```

```
{"name":"Michael"}
{"name":"Andy", "age":30}
{"name":"Justin", "age":19}
```


DataFrame Operations

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

```
+-----+-----+
|  age | count |
+-----+-----+
|   19 |      1 |
| null |      1 |
|   30 |      1 |
+-----+-----+
```

#SQL Equivalent

```
Select age, count(*) from df group by age
```

```
{"name": "Michael"}
{"name": "Andy", "age": 30}
{"name": "Justin", "age": 19}
```

Running SQL Queries Programmatically

Running SQL Queries Programmatically

```
// Register the DataFrame as a SQL temporary view  
df.createOrReplaceTempView("people")
```

Running SQL Queries Programmatically

```
// Register the DataFrame as a SQL temporary view  
df.createOrReplaceTempView("people")
```

```
val sqlDF = spark.sql("SELECT * FROM people")
```


Running SQL Queries Programmatically

```
// Register the DataFrame as a SQL temporary view  
df.createOrReplaceTempView("people")
```

```
val sqlDF = spark.sql("SELECT * FROM people")  
sqlDF.show()
```

```
+-----+-----+  
|  age |   name |  
+-----+-----+  
| null | Michael |  
|   30 |    Andy |  
|   19 |   Justin |  
+-----+-----+
```

Datasets

Datasets

- Similar to RDDs
- instead Java serialization or Kryo
- use a specialized Encoder
- use Encoder to serialize

Encoders

- Are dynamically generated code
- Perform operations with deserializing

Creating Datasets

```
// Encoders for most common types are automatically  
provided by importing spark.implicits._  
val primitiveDS = Seq(1, 2, 3).toDS()  
primitiveDS.map(_ + 1).collect() // Returns: Array(2, 3, 4)
```

Creating Datasets

```
case class Person(name: String, age: Long)

// Encoders are created for case classes
val caseClassDS = Seq(Person("Andy", 32)).toDS()
caseClassDS.show()
// +-----+-----+
// |name|age|
// +-----+-----+
// |Andy| 32|
// +-----+-----+
```

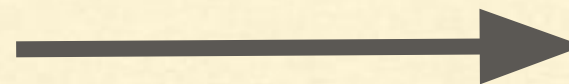

Creating Datasets

```
val path = "/data/spark/people.json"
val peopleDS = spark.read.json(path).as[Person]
peopleDS.show()
// +-----+-----+
// | age | name |
// +-----+-----+
// |null|Michael|
// | 30 | Andy |
// | 19 | Justin|
// +-----+-----+
```

Interoperating with RDDs

RDD

| | |
|---|---------|
| 1 | sandeep |
| 2 | ted |
| 3 | thomas |
| 4 | priya |
| 5 | kush |



Data Frame

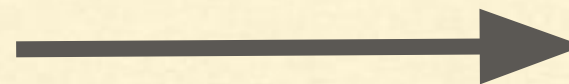
| ID | Name |
|----|---------|
| 1 | sandeep |
| 2 | ted |
| 3 | thomas |
| 4 | priya |
| 5 | kush |

How to convert an RDD into dataframe?

Interoperating with RDDs

RDD

| | |
|---|---------|
| 1 | sandeep |
| 2 | ted |
| 3 | thomas |
| 4 | priya |
| 5 | kush |



Data Frame

| ID | Name |
|----|---------|
| 1 | sandeep |
| 2 | ted |
| 3 | thomas |
| 4 | priya |
| 5 | kush |

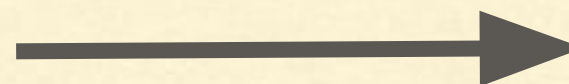
Two ways to convert RDDs to DF:

- a. Inferring the Schema Using Reflection
- b.

Interoperating with RDDs

RDD

| | |
|---|---------|
| 1 | sandeep |
| 2 | ted |
| 3 | thomas |
| 4 | priya |
| 5 | kush |



Data Frame

| ID | Name |
|----|---------|
| 1 | sandeep |
| 2 | ted |
| 3 | thomas |
| 4 | priya |
| 5 | kush |

Two ways to convert RDDs to DF:

- a. Inferring the Schema Using Reflection
- b. Programmatically Specifying the Schema


Inferring the Schema Using Reflection


- Spark SQL can convert an RDDs with case classes to a DataFrame
- The names of case class arguments are read using reflection and become columns
- Case classes can be nested or contain complex types
- Let us try to convert people.txt into dataframe

```
people.txt:  
Michael, 29  
Andy, 30  
Justin, 19
```

Inferring the Schema Using Reflection


GitHub, Inc. [US] https://github.com/cloudxlab/bigdata/blob/master/spark/examples/dataframes/rdd_to_df.scala

 Features Business Explore Marketplace Pricing This repository Search

 **cloudxlab / bigdata**
forked from [singhabhinav/cloudxlab](#) Watch 1

<> Code Pull requests 0 Projects 0 Insights ▾

Branch: **master ▾** **bigdata / spark / examples / dataframes / rdd_to_df.scala**

 **girisandeep** Create rdd_to_df.scala

1 contributor

9 lines (6 sloc) | 285 Bytes Raw Blame

```
1 import spark.implicits._
2
3 case class Person(name: String, age: Long)
4
5 val txtRDD = sc.textFile("/data/spark/people.txt")
6 val arrayRDD = txtRDD.map(_.split(","))
7 val personRDD = arrayRDD.map(attributes => Person(attributes(0), attributes(1).trim.toInt))
8 val peopleDF = personRDD.toDF()
```

https://github.com/cloudxlab/bigdata/blob/master/spark/examples/dataframes/rdd_to_df.scala

Inferring the Schema Using Reflection

```
scala> import spark.implicits._  
import spark.implicits._
```

Inferring the Schema Using Reflection

```
scala> import spark.implicits._  
import spark.implicits._  
  
scala> case class Person(name: String, age: Long)  
defined class Person
```


Inferring the Schema Using Reflection

```
scala> import spark.implicits._  
import spark.implicits._
```

```
scala> case class Person(name: String, age: Long)  
defined class Person
```

```
scala> val textRDD = sc.textFile("/data/spark/people.txt")  
textRDD: org.apache.spark.rdd.RDD[String] = /data/spark/people.txt  
MapPartitionsRDD[3] at textFile at <console>:30
```

Inferring the Schema Using Reflection

```
scala> import spark.implicits._  
import spark.implicits._
```

```
scala> case class Person(name: String, age: Long)  
defined class Person
```

```
scala> val textRDD = sc.textFile("/data/spark/people.txt")  
textRDD: org.apache.spark.rdd.RDD[String] = /data/spark/people.txt  
MapPartitionsRDD[3] at textFile at <console>:30
```

```
scala> val arrayRDD = textRDD.map(_.split(","))  
arrayRDD: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[4] at  
map at <console>:32
```

Inferring the Schema Using Reflection

```
scala> import spark.implicits._  
import spark.implicits._
```

```
scala> case class Person(name: String, age: Long)  
defined class Person
```

```
scala> val textRDD = sc.textFile("/data/spark/people.txt")  
textRDD: org.apache.spark.rdd.RDD[String] = /data/spark/people.txt  
MapPartitionsRDD[3] at textFile at <console>:30
```

```
scala> val arrayRDD = textRDD.map(_.split(","))  
arrayRDD: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[4] at  
map at <console>:32
```

```
scala> val personRDD = arrayRDD.map(attributes => Person(attributes(0),  
attributes(1).trim.toInt))  
personRDD: org.apache.spark.rdd.RDD[Person] = MapPartitionsRDD[5] at map  
at <console>:36
```

Inferring the Schema Using Reflection

```
scala> val peopleDF = personRDD.toDF()  
peopleDF: org.apache.spark.sql.DataFrame = [name: string, age: bigint]
```


Inferring the Schema Using Reflection

```
scala> val peopleDF = personRDD.toDF()  
peopleDF: org.apache.spark.sql.DataFrame = [name: string, age: bigint]
```

```
scala> peopleDF.show()
```

```
+-----+-----+  
|  name | age |  
+-----+-----+  
|Michael|  29 |  
|  Andy |  30 |  
| Justin|  19 |  
+-----+-----+
```

Inferring the Schema Using Reflection

```
// Register the DataFrame as a temporary view
peopleDF.createOrReplaceTempView("people")

// SQL statements can be run by using the sql methods provided by Spark
val teenagersDF = spark.sql("SELECT name, age FROM people WHERE age BETWEEN 13 AND 19")
```

Inferring the Schema Using Reflection

```
// Register the DataFrame as a temporary view
peopleDF.createOrReplaceTempView("people")

// SQL statements can be run by using the sql methods provided by Spark
val teenagersDF = spark.sql("SELECT name, age FROM people WHERE age BETWEEN 13 AND 19")

// The columns of a row in the result can be accessed by field index
teenagersDF.map(teenager => "Name: " + teenager(0)).show()
// +-----+
// |      value|
// +-----+
// |Name: Justin|
// +-----+
```

Inferring the Schema Using Reflection

```
// Register the DataFrame as a temporary view
peopleDF.createOrReplaceTempView("people")

// SQL statements can be run by using the sql methods provided by Spark
val teenagersDF = spark.sql("SELECT name, age FROM people WHERE age BETWEEN 13 AND 19")

// The columns of a row in the result can be accessed by field index
teenagersDF.map(teenager => "Name: " + teenager(0)).show()
// +-----+
// |      value|
// +-----+
// |Name: Justin|
// +-----+

// or by field name
teenagersDF.map(teenager => "Name: " + teenager.getAs[String]("name")).show()
// +-----+
// |      value|
// +-----+
// |Name: Justin|
// +-----+
```

Inferring the Schema Using Reflection

```
// No pre-defined encoders for Dataset[Map[K,V]], define explicitly
implicit val mapEncoder = org.apache.spark.sql.Encoders.kryo[Map[String, Any]]
// Primitive types and case classes can be also defined as
// implicit val stringIntMapEncoder: Encoder[Map[String, Any]] = ExpressionEncoder()

// row.getValuesMap[T] retrieves multiple columns at once into a Map[String, T]
teenagersDF.map(teenager => teenager.getValuesMap[Any](List("name", "age"))).collect()
// Array(Map("name" -> "Justin", "age" -> 19))
```

Programmatically Specifying the Schema

- When case classes can't be defined during time of coding
 - a. E.g. The fields expected in case classes are passed as arguments
- We need to programmatically create the dataframe:

Programmatically Specifying the Schema

- When case classes can't be defined during time of coding
 - a. E.g. The fields expected in case classes are passed as arguments
- We need to programmatically create the dataframe:
 1. Create RDD of Row objects
 2. Create schema represented by StructType
 3. Apply schema with createDataFrame

Programmatically Specifying the Schema

people.txt:

Michael, 29

Andy, 30

Justin, 19

```
val schemaString = "name age"
```


Programmatically Specifying the Schema

```
import org.apache.spark.sql.types._  
import org.apache.spark.sql._
```

Programmatically Specifying the Schema

```
import org.apache.spark.sql.types._
import org.apache.spark.sql._

// The schema is encoded in a string
// User provided variable
val schemaString = "name age"
val filename = "/data/spark/people.txt"

val fieldsArray = schemaString.split(" ")
val fields = fieldsArray.map(
    f => StructField(f, StringType, nullable = true)
)
val schema = StructType(fields)
```

Programmatically Specifying the Schema

```
import org.apache.spark.sql.types._
import org.apache.spark.sql._

// The schema is encoded in a string
// User provided variable
val schemaString = "name age"
val filename = "/data/spark/people.txt"

val fieldsArray = schemaString.split(" ")
val fields = fieldsArray.map(
    f => StructField(f, StringType, nullable = true)
)
val schema = StructType(fields)

val peopleRDD = spark.sparkContext.textFile(filename)
val rowRDD = peopleRDD.map(_._split(",")).map(
    attributes => Row.fromSeq(attributes)
)
val peopleDF = spark.createDataFrame(rowRDD, schema)
```

Programmatically Specifying the Schema

```
import org.apache.spark.sql.types._
import org.apache.spark.sql._

// The schema is encoded in a string
// User provided variable
val schemaString = "name age"
val filename = "/data/spark/people.txt"

val fieldsArray = schemaString.split(" ")
val fields = fieldsArray.map(
    f => StructField(f, StringType, nullable = true)
)
val schema = StructType(fields)

val peopleRDD = spark.sparkContext.textFile(filename)
val rowRDD = peopleRDD.map(_.split(",")).map(attributes => Row.fromSeq(attributes))
val peopleDF = spark.createDataFrame(rowRDD, schema)
peopleDF.show()
+-----+-----+
|   name|age|
+-----+-----+
|Michael| 29|
|   Andy| 30|
|  Justin| 19|
+-----+-----+
```




Spark SQL & Dataframes

Thank you!

