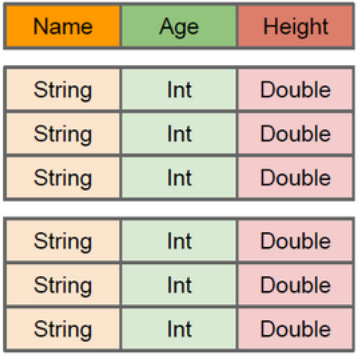
Spark DataFrames

Anurag Nagar

Big Data Class

Introduction

- DataFrames are part of **Spark SQL**.
- Like RDDs, DataFrames (DF) are **immutable, distributed, partitioned** collection of data
- They have all the properties of RDDs, such as lazy evaluation, recovery through lineage graphs, etc.
- They contain specialized APIs for working with **tabular** data, and have **named columns.**

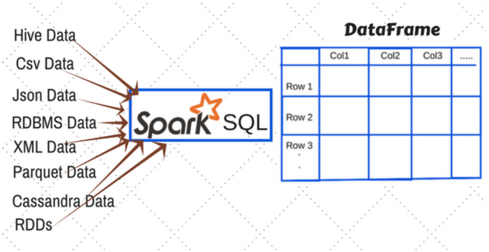


Name	Age	Height
String	Int	Double
String	Int	Double
String	Int	Double
String	Int	Double
String	Int	Double
String	Int	Double

DataFrame

Creating DataFrames

- DataFrames are well suited for large structured or semi-structured data.
- Data can be loaded easily from a wide variety of sources
- DF contain named columns, and a list of tuples



Loading Data into DataFrames

spark.read is the starting point to read data into DF. More details can be found at this link.

- To read a simple CSV file with header

```
val df = spark.read.option("header", "true")
      .csv(FILEPATH)
```

- To read in a file with custom delimiter

```
val df = spark.read.option("header", "true")
      .option("delimiter","|") .load(FILEPATH)
```

Loading Data into DataFrames

spark.read is the starting point to read data into DF. More details can be found at this link.

- RDDs can be converted to DF

```
// define a class that corresponds to each row of data
case class Person(name: String, age: Long)
// Create an RDD of Person objects from a text file ,
    convert it to a Dataframe
val peopleDF = sc
  . textFile ("examples/src/main/resources/people.txt")
  .map(_.split(","))
  .map(attributes => Person(attributes(0),
       attributes (1).trim.toInt ))
  .toDF()
```

Loading Data into DataFrames

- To extract few columns

```
val filtered = df.select("column1", "column2").show()
```

- To filter data with conditions:

```
val selected = df.filter($"column" > value).show()
// example
val selected = df.filter($"age" > 21).show()
```

Loading Data into DataFrames

- To group by a column and get count of groups:

```
val groupCountd = df.groupBy("column").count()
```

- To group by a column and show average of another column by group

```
val groupAge = df.groupBy("column").avg("col2")
```

- To find other stats

```
import org.apache.spark.sql.functions.{avg,mean,stddev}
val stats = cars.groupBy("automatic").agg(avg("mpg"),
    stddev("torque"))
```

Loading Data into DataFrames

- To join two DF

```
val df = left.join(right, left.col("name") ===
      right.col("name"))
```

- To do left/right outer join

```
val df = left.join(right, left.col("name") ===
      right.col("name"), joinType="param")
```

where **param** could be one of the following: *inner*, *outer*, *left_outer*, *right_outer*