Dataframes

- Dataframes are a restricted sub-type of RDDS
- Restricting the type allows for more optimization

Dataframes store two dimensional data, similar to the type of data stored in a spreadsheet

- Each column in a dataframe can have a different type
- Each row contains a record

Similar to pandas dataframes and R dataframes

Constructing a DataFrame from an RDD of Rows

Each row defines its own fields, the schema is *inferred*

In [5]:
```
# One way to create a DataFrame is to first define an RDD from a list c
_list=[Row(name=u"John", age=19),
       Row(name=u"Smith", age=23),
       Row(name=u"Sarah", age=18)]
some_rdd = sc.parallelize(_list)
some_rdd.collect()
```

Out[5]:
```
[Row(name='John', age=19),
 Row(name='Smith', age=23),
 Row(name='Sarah', age=18)]
```

In [6]:

```
# The DataFrame is created from the RDD or Rows
# Infer schema from the first row, create a DataFrame and print the sch
some_df = sqlContext.createDataFrame(_list)
```

In [7]:
```
some_df.printSchema()
```

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

In [8]:

```
# A dataframe is an RDD of rows plus information on the schema.
# performing **collect()* on either the RDD or the DataFrame gives the
print(type(some_rdd),type(some_df))
print('some_df =',some_df.collect())
print('some_rdd=',some_rdd.collect())
```

```
<class 'pyspark.rdd.RDD'> <class 'pyspark.sql.dataframe.DataFra
me'>
some_df = [Row(name='John', age=19), Row(name='Smith', age=23),
Row(name='Sarah', age=18)]
some_rdd= [Row(name='John', age=19), Row(name='Smith', age=23),
Row(name='Sarah', age=18)]
```

Defining the Scheme explicitly

The advantage of creating a DataFrame using a predefined scheme allows the content of the RDD to be simple tuples, rather than rows.

In [9]:

```
# In this case we create the dataframe from an RDD of tuples (rather th
another_rdd = sc.parallelize([("John", 19), ("Smith", 23), ("Sarah", 18
# Schema with two fields - person_name and person_age
schema = StructType([StructField("person_name", StringType(), False),
                     StructField("person_age", IntegerType(), False)])

# Create a DataFrame by applying the schema to the RDD and print the sc
another_df = sqlContext.createDataFrame(another_rdd, schema)
another_df.printSchema()
# root
#  |-- age: binteger (nullable = true)
#  |-- name: string (nullable = true)
```

```
root
 |-- person_name: string (nullable = false)
 |-- person_age: integer (nullable = false)
```

Loading DataFrames from disk

There are many methods to load DataFrame from Disk. Here we will discuss three of these methods.

1. Parquet
2. JSON (on your own)
3. CSV (on your own)

In addition, there are API's for connecting Spark to an external database. We will not discuss this type of connection in this class.

Parquet Files

- Parquet is a popular columnar format
- Spark SQL allows SQL queries retrieve a subset of the rows without reading the whole file
- Compatible with HDFS: allows parallel retrieval on a cluster
- Parquet compresses the data in each column
- `<reponame>.parquet` is usually a **directory** with many files or subdirectories

In addition, there are API's for connecting Spark to an external database. We will not discuss this type of connection in this class.

In [26]:

```
#load a Parquet file
print(parquet_file)
df = sqlContext.read.load(parquet_file)
df.show()
```

```
../../../Data/users.parquet
+------+--------------+----------------+
|  name|favorite_color|favorite_numbers|
+------+--------------+----------------+
|Alyssa|          null|  [3, 9, 15, 20]|
|   Ben|           red|              []|
+------+--------------+----------------+
```

In [27]:
```
df2=df.select("name", "favorite_color")
df2.show()
```

```
+------+--------------+
|  name|favorite_color|
+------+--------------+
|Alyssa|          null|
|   Ben|           red|
+------+--------------+
```

In [28]:

```
outfilename="namesAndFavColors.parquet"
!rm -rf $dir/$outfilename
df2.write.save(dir+"/"+outfilename)
!ls -ld $dir/$outfilename
```

```
drwxr-xr-x  6 yoavfreund  staff  192 Apr 16 16:46 ../../../Dat
a/namesAndFavColors.parquet
```

Let’s have a look at a real-world dataframe

The dataframe is a small part from a small part from a large dataframe (15GB) which stores meteorological data from stations around the world

In [16]:

```
#List is a list of Rows. Stored as a pickle file.
df=sqlContext.createDataFrame(List)
print(df.count())
df.show(1)
```

```
12373
+---------+--------+---------+-----------+-----------+------+--
------------------+------+--------+
|elevation|latitude|longitude|measurement|    station|undefs|
vector|  year|   label|
+---------+--------+---------+-----------+-----------+------+--
------------------+------+--------+
|    181.4| 41.0092| -87.8242|       PRCP|USC00111458|     8|[0
0 00 00 00 00 0...|1991.0|BBSSBBSS|
+---------+--------+---------+-----------+-----------+------+--
------------------+------+--------+
only showing top 1 row
```

In [17]:

```
#selecting a subset of the rows so it fits in slide.
df.select('station','year','measurement').show(5)
```

```
+-----------+------+-----------+
|    station|  year|measurement|
+-----------+------+-----------+
|USC00111458|1991.0|       PRCP|
|USC00111458|1994.0|       PRCP|
|USC00111458|1995.0|       PRCP|
|USC00111458|1996.0|       PRCP|
|USC00111458|1997.0|       PRCP|
+-----------+------+-----------+
only showing top 5 rows
```

In [18]:

```
### Save dataframe as Parquet repository
filename='%s/US_Weather_%s.parquet'%(data_dir,file_index)
!rm -rf $filename
df.write.save(filename)
```

Summary

- Dataframes are an efficient way to store data tables
- All of the values in a column have the same type
- A good way to store a dataframe in disk is to use a Parquet file
- Next: Operations on dataframes