UNIVERSITY of WASHINGTON

Welcome to BIGDATA 210: Introduction to Data Engineering

Week 4 - 1

Week 4 Part 1

What we will do?

- Questions from last week
- Status of Projects
- Spark SQL
- Hands-On

Spark SQL Overview

- A module built on top of Spark Core
- Provides a programming abstraction for distributed processing of large-scale structured data in Spark
- Data is described as a DataFrame with rows, columns and a schema
- Data manipulation and access is available with two mechanisms
  - SQL Queries
  - DataFrames API

The DataFrame Abstraction

- A DataFrame is inspired by the DataFrame concept in R (dplr, DataFrame) or Python (pandas), but stored using RDDs underneath in a distributed manner
- A DataFrame is organized into named columns
  - Underneath: an RDD of "Row" objects
- The DataFrame API is available in Scala, Java, Python, and R

DataFrames can be created from various sources

- DataFrames from HIVE data
  - Reading/writing HIVE tables, including ORC
- DataFrames from files:
  - Built-in: JSON, JDBC, Parquet, HDFS, ORC
  - External plug-in: CSV, HBASE, Avro, memsql, elasticsearch

SQLContext and HiveContext

- To use Spark-SQL you first create a SQLContext

```
python
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
```

Alternatively you can create a HiveContext to connect with HIVE:

```
python
from pyspark.sql import HiveContext
  hc = HiveContext(sc)
```

Example: Using the DataFrames API

```
from pyspark.sql import HiveContext
hc = HiveContext(sc)

hc.sql("use demo")
df1 = hc.table("crimes") \
            .select("year", "month", "day", "category") \
                .filter("year > 2014").head(5)
```

Example: using SQL syntax

```
from pyspark.sql import HiveContext
hc = HiveContext(sc)

hc.sql("use demo")
df1 = hc.sql("""
          SELECT year, month, day, category
          FROM crimes
          WHERE year > 2014""").head(5)
```

Creating a DataFrame: from a table in HIVE

- Load the entire table

```
python
from pyspark.sql import HiveContext
hc = HiveContext(sc)
df=hc.table("patients")
```

- Load using a SQL Query

```
python
hc.sql("Use people")
df1 = hc.sql("SELECT * from patients WHERE age>50")
df2 = hc.sql("""
    SELECT col1 as timestamp, SUBSTR(date,1,4) as year, event
    FROM events
    WHERE year > 2014""")
```

DataFrameReader / DataFrameWriter API

- DataFrameReader
  - Interface used to load a DataFrame from external storage
  - format(str) – supports “orc”, “parquet”, “json”, etc
  - load(path-to-file)
- DataFrameWriter
  - Interface used to store a DataFrame to external storage
  - format(str) – supports “orc”, “parquet”, “json”, etc
  - mode(str) - what to do when file exists: “append”, “ignore”, “overwrite”, “error”
  - save(path-to-file)

Creating a DataFrame from a file

- From a JSON file

```
df = hc.read.json("somefile.json")
```

```
df = hc.read.format("json").load("somefile.json")
```

- From Parquet file

```
df = hc.read.parquet("somefile.parquet")
```

- From a CSV file:

```
df = hc.read.format("com.databricks.spark.csv")
  .options(header='true').load("somefile.csv")
```

Creating a DataFrame from an RDD

- DataFrames can be manually created from RDDs of Row objects.

```
from pyspark.sql import Row
rdd = sc.parallelize([Row(name='Alice', age=12, height=80), \
          Row(name='Bob', age=15, height=120)])
df = rdd.toDF()
```

- Alternatively, we can let Spark SQL infer the schema using createDataFrame()

```
rdd = sc.parallelize([('Alice', 12, 80), ('Bob', 15, 120)])
df = hc.createDataFrame(rdd, ['name', 'age', 'height'])
```

Creating a DataFrame from text

Creating a DataFrame from a text file after splitting it, and then mapping the fields to a row object:

```
from pyspark.sql import SQLContext, Row
sqlContext = SQLContext(sc)
##Want to create a DataFrame of People
##Attributes will be Name, Age

lines = sc.textFile("examples/src/main/resources/people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))

##Create the DataFrame
peopleDF=sqlContext.createDataFrame(people)
```

Example DataFrames

```
df1 = sc.parallelize(
    [Row(cid='101', name='Alice', age=25, state='ca'), \
    Row(cid='102', name='Bob', age=15, state='ny'), \
    Row(cid='103', name='Bob', age=23, state='nc'), \
    Row(cid='104', name='Ram', age=45, state='fl')]).toDF()

df2 = sc.parallelize(
    [Row(cid='101', date='2015-03-12', product='toaster', price=200), \
    Row(cid='104', date='2015-04-12', product='iron', price=120), \
    Row(cid='102', date='2014-12-31', product='fridge', price=850), \
    Row(cid='102', date='2015-02-03', product='cup', price=5)]).toDF()
```

Returning rows of dataframe

- first() returns the first row, take(n) returns n rows

```
df1.first()
Row(age=23, cid=u'104', name=u'Bob', state=u'nc')

df1.take(2)
[Row(age=45, cid=u'104', name=u'Ram', state=u'fl')
Row(age=15, cid=u'102', name=u'Bob', state=u'ny')]
```

Returning rows of dataframe

- limit(n): reduce the DataFrame to n rows
  - Result is still a DataFrame, not a python result list
- show(n): prints the first n rows to the console

```
df1.show(3)
+---+---+-----+-----+
|age|cid| name|state|
+---+---+-----+-----+
| 25|101|Alice|   ca|
| 15|102|  Bob|   ny|
| 23|103|  Bob|   nc|
+---+---+-----+-----+

df1.limit(2).show()
+---+---+-----+-----+
|age|cid| name|state|
+---+---+-----+-----+
| 15|102|  Bob|   ny|
| 45|101|Alice|   ca|
+---+---+-----+-----+
```

Inspecting Schema

```
df1.columns  #Display column names
[u'age', u'cid', u'name', u'state']

df1.dtypes  #Display column names and types
[('age', 'bigint'), ('cid', 'string'), ('name', 'string'), ('state', 'string')]

df1.schema  #Display detailed schema
StructType(List(StructField(age,LongType,true),
StructField(cid,StringType,true),
StructField(name,StringType,true),
StructField(state,StringType,true)))
```

Counting rows

Count all the rows in a DataFrame

```
df1.count()
```

4

Note

count() returns number of non-duplicate rows

df1.rdd.count() returns number of actual rows

Extracting a smaller sample of the RDD

```
raw_rdd_sample = raw_rdd.sample(False, 0.1, 1234)
print raw_rdd_sample.count()
print raw_rdd.count()
```

Summary Statistics

```
df1.describe().show()
+-------+------------------+
|summary|               age|
+-------+------------------+
| count |                 4|
| mean  |              27.0|
| stddev|11.045361017187261|
| min   |                15|
| max   |                45|
+-------+------------------+
```

Describe() shows statistics for all numeric columns, ignoring others

Removing duplicates

```
df1.distinct().show()
+---+---+-----+-----+
|age|cid| name|state|
+---+---+-----+-----+
| 23|103|  Bob|   nc|
| 15|102|  Bob|   ny|
| 45|104|  Ram|   fl|
| 25|101|Alice|   ca|
+---+---+-----+-----+
```

Removing duplicate rows by specific columns

- Removing duplicate rows by key, drops every row with the same key but the first occurrence

```
df1.drop_duplicates(["name"]).show()
```

```
+---+---+-----+-----+
|age|cid| name|state|
+---+---+-----+-----+
| 15|102|  Bob|   ny|
| 45|104|  Ram|   fl|
| 25|101|Alice|   ca|
+---+---+-----+-----+
```

Sorting rows

```
df1.sort(df1["age"].desc()).
show()
```

```
+---+---+-----+-----+
|age|cid| name|state|
+---+---+-----+-----+
| 45|104|  Ram|   fl|
| 25|101|Alice|   ca|
| 23|103|  Bob|   nc|
| 15|102|  Bob|   ny|
+---+---+-----+-----+
```

```
df1.sort("age", ascending=True).show()
```

```
+---+---+-----+-----+
|age|cid| name|state|
+---+---+-----+-----+
| 15|102|  Bob|   ny|
| 23|103|  Bob|   nc|
| 25|101|Alice|   ca|
| 45|104|  Ram|   fl|
+---+---+-----+-----+
```

Adding a column

Here is an example of creating a new column, from an existing one by applying a transformation.

```
df1.withColumn('age-dog-years', df1.age*7).show()
```

```
 +---+---+-----+-----+-------------+
|age|cid| name|state|age-dog-years|
+---+---+-----+-----+-------------+
| 25|101|Alice|   ca|          175|
| 15|102|  Bob|   ny|          105|
| 23|103|  Bob|   nc|          161|
| 45|104|  Ram|   fl|          315|
+---+---+-----+-----+-------------+
```

Renaming a column

```
df1.withColumnRenamed('age', 'age2').show()
```

```
+----+---+-----+-----+
|age2|cid| name|state|
+----+---+-----+-----+
|  25|101|Alice|   ca|
|  15|102|  Bob|   ny|
|  23|103|  Bob|   nc|
|  45|104|  Ram|   fl|
+----+---+-----+-----+
```

select() operator

DataFrames allow the developer to only select some columns of the DataFrame.

```
df1.select(“name”, “age”).show()
+-----+---+
| name|age|
+-----+---+
|Alice| 25|
|  Bob| 15|
|  Bob| 23|
|  Ram| 45|
+-----+---+

df1.select(df1.name, df1.age*7).show()
+-----+---------+
| name|(age * 7)|
+-----+---------+
|Alice|      175|
|  Bob|      105|
|  Bob|      161|
|  Ram|      315|
+-----+---------+
```

selectExpr() operator

The select expression accepts a sql query

```
df1.selectExpr("substr(name,1,3)", "age*7").collect()
+------------------+---------+
|SUBSTR(name, 1, 3)|(age * 7)|
+------------------+---------+
|               Ali|      175|
|               Bob|      105|
|               Bob|      161|
|               Ram|      315|
+------------------+---------+
```

Column expression

Operations can be done on column objects, here are some example:

Cast to type: `df1.age.cast("string")`
Rename a column: `df1.age.alias("age2")`
Sort a column: `df1.age.asc() or df.age.desc()`
Substring: `df1.name.substr(1,3)`
Between: `df1.age.between(25, 34) # 25<=age<=34`

Dropping columns

DataFrame Operations allow for dropping comumns with the drop("column_name") api

```
df1.drop("age").show()
```

```
+---+-----+-----+
|cid| name|state|
+---+-----+-----+
|101|Alice|   ca|
|102|  Bob|   ny|
|103|  Bob|   nc|
|104|  Ram|   fl|
+---+-----+-----+
```

Filtering rows

DataFrames support filtering rows, like with the Core RDD library

```
df1.filter(df1.age>21).show()
OR

df1.filter(df1["age"]>21).show()
```

```
+---+---+-----+-----+
|age|cid| name|state|
+---+---+-----+-----+
| 25|101|Alice|   ca|
| 23|103|  Bob|   nc|
| 45|104|  Ram|   fl|
+---+---+-----+-----+
```

groupBy()

DataFrames provide a groupBY api. After using a groupBY, ensure an aggregator comes next

```
df1.groupBy('name').count().show()
```

```
+-----+-----+
| name|count|
+-----+-----+
|  Ram|    1|
|Alice|    1|
|  Bob|    2|
+-----+-----+
```

groupBy and sum()

```
df2.select(df2["date"].substr(1,4).alias("year"), df2["price"]).groupBy("year").sum().show()
```

```
+----+----------+
|year|SUM(price)|
+----+----------+
|2014|       850|
|2015|       325|
+----+----------+
```

groupBy and agg()

DataFrames provide a generic function for implementing aggregations after groupBy. The agg() takes a dictionary mapping column names to aggregate functions (min, max, count, avg, sum). There can be multiple columns in the groupBy as well, just separate them with commas

```
df2.select(df2["date"].substr(1,4).alias("year"), df2["price"]).groupBy("year").agg({"price": "avg", "year": "count"}).show()
```

```
+----+-----------------+-----------+
|year|       AVG(price)|COUNT(year)|
+----+-----------------+-----------+
|2014|            850.0|          1|
|2015|108.3333333|          3|
+----+-----------------+-----------+
```

Inner Join with Data Frames

```
df1.join(df2, df1["cid"]==df2["cid"], "inner").show()
+---+---+-----+-----+---+----------+-----+-------+
|age|cid| name|state|cid|      date|price|product|
+---+---+-----+-----+---+----------+-----+-------+
| 25|101|Alice|   ca|101|2015-03-12|  200|toaster|
| 15|102|  Bob|   ny|102|2014-12-31|  850| fridge|
| 15|102|  Bob|   ny|102|2015-02-03|    5|    cup|
| 45|104|  Ram|   fl|104|2015-04-12|  120|   iron|
+---+---+-----+-----+---+----------+-----+-------+
```

Other types of join

DataFrames support all types of joins: inner, outer, left*outer*, *right*outer and semijoin.

To join two DataFrames, use the following syntax:

```
df1.join(df2, joinExpr, joinTyper)
```

joinType is from the list of joins above.

```
joinExpr can be written in two ways
df1.join(df2, "cid", "inner")
df1.join(df2, df1["cid"]==df2["cid"], "inner")
```

Multiple Conditions in joinExpr

DataFrames also supports multiple condititions in the joinExpr

```
df1.join(df2, (df1["cid"]==df2["cid"] & (df2["price"] > 200), "inner").show()
```

```
+---+---+----+-----+---+----------+-----+-------+
|age|cid|name|state|cid|      date|price|product|
+---+---+----+-----+---+----------+-----+-------+
| 15|102| Bob|   ny|102|2014-12-31| 850 | fridge|
+---+---+----+-----+---+----------+-----+-------+
```