

# 732A54/TDDE31 Big Data Analytics

## Introduction of Spark SQL

# DataFrames

A DataFrame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood. DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs.

<http://spark.apache.org/docs/1.6.0/sql-programming-guide.html>

<http://spark.apache.org/docs/1.6.0/api/python/pyspark.sql.html>

# SQLContext & HiveContext

- Start with obtaining `SparkContext` object and then `SQLContext` from it

```
sc = SparkContext()  
sqlContext = SQLContext(sc)
```

- `HiveContext` provides additional features to `SQLContext` (likely not needed for the lab assignment)

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

# Imports

Don't forget to import relevant classes first!

```
from pyspark import SparkContext
from pyspark.sql import SQLContext, Row
from pyspark.sql import functions as F
```

# Create a DataFrame from a RDD

- Two ways:
  - Inferring the schema using reflection
  - Specifying the schema programatically
- Then register the table

# Create a DataFrame from a RDD – way I

```
# Load a text file and convert each line to a Row.  rdd  
= sc.textFile("FILENAME")
```

```
parts = rdd.map(lambda l: l.split(";"))
```

```
tempReadings = parts.map(lambda p: Row(station=p[0],  
date=p[1], year=p[1].split("-")[0], time=p[2],  
value=float(p[3]), quality=p[4] ))
```

- Inferring the schema and registering the DataFrame as a table

```
schemaTempReadings =
```

```
sqlContext.createDataFrame(tempReadings)
```

```
schemaTempReadings.registerTempTable("tempReadings")
```

```
# Can run queries now
```

# Create a DataFrame from a RDD – way II

```
# Load a text file and convert each line to a tuple.  
rdd = sc.textFile("FILENAME")  
  
parts = rdd.map(lambda l: l.split(";"))  
  
tempReadingsRow = parts.map(lambda p: (p[0], p[1], int(p[1].split("-")[0]),  
int(p[1].split("-")[1]), p[2], float(p[3]), p[4] ))
```

- Specifying the schema programatically and registering the DataFrame as a table

```
tempReadingsString = ["station", "date", "year", "month", "time", "value",  
"quality"]
```

```
# Apply the schema to the RDD.
```

```
schemaTempReadings = sqlContext.createDataFrame(tempReadingsRow,  
tempReadingsString)
```

```
# Register the DataFrame as a table.
```

```
schemaTempReadings.registerTempTable("tempReadingsTable")
```

```
# Can run queries now
```

# Access columns' values

- In pyspark it's possible to access a DataFrame's columns either by attribute (`dataframe.age`) or by indexing (`dataframe['age']`). While the former is convenient for interactive data exploration, users are highly encouraged to **use the latter form**, which is future proof and won't break with column names that are also attributes on the DataFrame class.



# Run SQL queries

- Two ways:
  - Write regular SQL over DataFrames registered as tables using `sqlContext.sql("Your regular SQL query")`
  - Use API methods, such as `select()`, `filter()`, `groupBy()`, `agg()`, etc.

**The results of the SQL queries are DataFrames, to convert to RDD you can use:**

```
dataFrame.rdd
```

**and then apply operations for RDDs**

# Run SQL queries – regular SQL – way I

These examples use **temperature-readings.csv**

```
# SQL can be run over DataFrames that have been  
registered as a table.
```

```
max1950 = sqlContext.sql("SELECT max(value) as value  
FROM tempReadings WHERE year=1950")
```

```
largerThan10Degrees = sqlContext.sql("SELECT year,  
month, count(value) as value FROM tempReadingsTable  
WHERE year=1950 and value>=10.0 group by year, month")
```

# Run SQL queries – API methods – way II

These examples use `temperature-readings.csv`

```
schemaTempReadingsMin =  
schemaTempReadings.groupBy('year', 'month', 'day',  
'station').agg(F.min('value').alias('dailymin'))  
.orderBy(['year', 'month', 'day', 'station'],  
ascending=[0,0,0,1])
```

# Run SQL queries – API methods – way II

This example uses `temperature-readings.csv` and `stations-Ostergotland.csv`

**Note: Partial code !!!!! - it demonstrates the join not the entire code needed before**

```
# Define the schema
```

```
stationsString = ["stationNum", "stationName"]
```

```
precReadingsString = ["station", "date", "year", "month", "day",  
"time", "value", "quality"]
```

```
# Apply the schema to the RDD.
```

```
schemaStations = sqlContext.createDataFrame(stationsRow,  
stationsString)
```

```
schemaPrecReadings = sqlContext.createDataFrame(precReadingsRow,  
precReadingsString)
```

```
precStationOst = schemaStations.join(schemaPrecReadings,  
schemaStations['stationNum']==schemaPrecReadings['station'], 'inner')
```