SparkSQL
Where are we?

- Pig Latin
  - Pig
  - Hadoop MapReduce
  - HDFS

- HiveQL
  - Hive
  - ...?
  - Spark RDD

- ???
Where are we?

- Pig Latin
- HiveQL
- Pig
- Hive
- ...  
- SQL
- ???
- Hadoop MapReduce
- Spark RDD
- HDFS
Shark (Spark on Hive)

- A small side project that aimed to running RDD jobs on Hive data using HiveQL
- Still limited to the data model of Hive
- Tied to the Hadoop world
SparkSQL

- Redesigned to consider Spark query model
- Supports all the popular relational operators
- Can be intermixed with RDD operations
- Uses the Dataframe API as an enhancement to the RDD API

Dataframe = RDD + schema
Dataframes

- SparkSQL’s counterpart to relations or tables in RDMBS
- Consists of rows and columns
- A dataframe is **NOT** in 1NF
  - Why?
- Can be created from various data sources
  - CSV file
  - JSON file
  - MySQL database
  - Hive
Dataframe Vs RDD

**Dataframe**
- Lazy execution
- Spark is aware of the data model
- Spark is aware of the query logic
- Can optimize the query

**RDD**
- Lazy execution
- The data model is hidden from Spark
- The transformations and actions are black boxes
- Cannot optimize the query
Built-in operations in SprkSQL

- Filter (Selection)
- Select (Projection)
- Join
- GroupBy (Aggregation)
- Load/Store in various formats
- Cache
- Conversion between RDD (back and forth)
SparkSQL Examples
Project Setup

# In dependencies pom.xml
<!--
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.11</artifactId>
  <version>2.2.1</version>
</dependency>
Code Setup

SparkSession sparkS = SparkSession
    .builder()
    .appName("Spark SQL examples")
    .master("local")
    .getOrCreate();

Dataset<Row> log_file = sparkS.read()
    .option("delimiter", "\t")
    .option("header", "true")
    .option("inferSchema", "true")
    .csv("nasa_log.tsv");

log_file.show();
Filter Example

# Select OK lines
Dataset<Row> ok_lines = log_file.filter("response=200");
long ok_count = ok_lines.count();
System.out.println("Number of OK lines is " + ok_count);

# Grouped aggregation using SQL
Dataset<Row> bytesPerCode = log_file.sqlContext().sql("SELECT response, sum(bytes) from log_lines GROUP BY response");
SparkSQL Features

- Catalyst query optimizer
- Code generation
- Integration with libraries
DataFrames and SQL share the same optimization/execution pipeline

Credits: M. Armbrust
Catalyst Query Optimizer

- Extensible rule-based optimizer
- Users can define their own rules
Code Generation

- Shift from black-box UDF to **Expressions**

**Example**

- # Filter
  - Dataset<Row> ok_lines = log_file.filter("response=200");
- # Grouped aggregation
  - Dataset<Row> bytesPerCode = log_file.sqlContext().sql("SELECT response, sum(bytes) from log_lines GROUP BY response");

- **SparkSQL** understand the logic of user queries and rewrites them in a more concise way
Integration

- SparkSQL is integrated with other high-level interfaces such as MLlib, PySpark, and SparkR
- SparkSQL is also integrated with the RDD interface and they can be mixed in one program
Further Reading

- Documentation

- SparkSQL paper
  - M. Armbrust *et al.* "Spark sql: Relational data processing in spark." SIGMOD 2015