Apache Spark

Andrés Gómez Ferrer andresgomezfrr@gmail.com



Spark SQL



Spark SQL: Basics

- Spark SQL es un modulo para el procesamiento de datos estructurados.
- Las interfaces proporcionan mayor información sobre la estructura y la computación de los datos que la API de Spark RRD.
- Existen varias formas para trabajar con Spark SQL:
- **Dataframe:** Conjunto de datos organizados en columnas. Es equivalente a una tabla relacional. Pueden ser construidos desde: ficheros estructurados, tablas en Hive, base de datos externas, o RRDs existentes. Scala/Java/Python/R
- Dataset: Conjunto de datos distribuidos, permite usar Spark SQL con los beneficios de los RDDs: tipado fuerte, transformaciones funcionales, etc. Scala/Java
 - SQL: Lenguaje con sintasix SQL



Spark SQL: Base Project

- Para empezar un proyecto de Spark, necesitamos añadir sus dependencias en nuestro proyecto de sbt en el IDE.
- IMPORTANTE: SCALA 2.12.12

```
libraryDependencies ++= Seq(
   "org.apache.spark" %% "spark-core" % "3.0.1",
   "org.apache.spark" %% "spark-sql" % "3.0.1"
)
```

- Una vez tenemos las dependencias, podemos crear un objeto principal con un método main, donde crearemos un nuevo contexto de spark.
- Desde este punto podemos ejecutar en el play del IDE y ejecuta spark en modo local[1], modo local, usando 1CPU de la maquina.
- En este caso usaremos una instancia de SparkSession en lugar de SparkContext

```
package io.keepcoding.spark.sql
import org.apache.spark.sql.SparkSession
object SparkSqlBaseProject {
 def main(args: Array[String]): Unit = {
  val spark = SparkSession
    .builder()
    .appName("Spark SQL KeepCoding")
    .getOrCreate()
  import spark.implicits.
  // <code>
  spark.close()
```

Read Multiple Data Sources

- > CSV
- > JSON

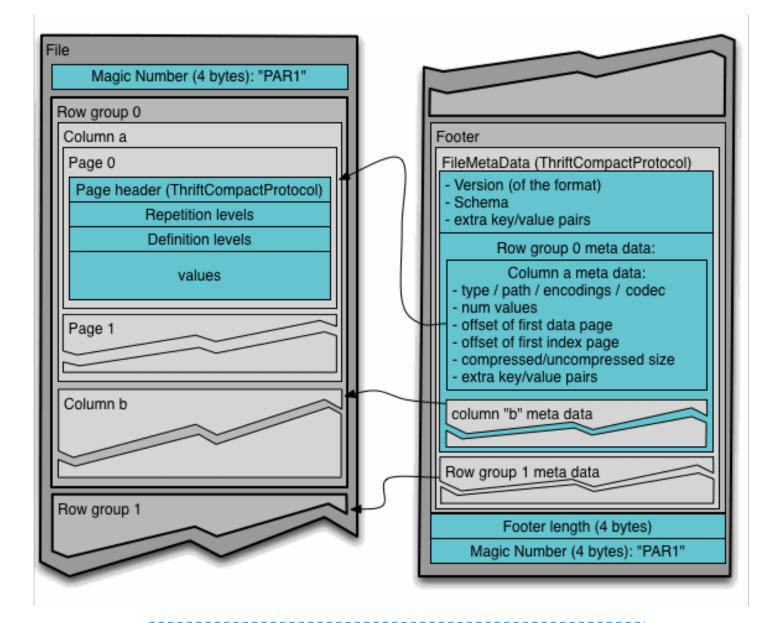


Spark SQL: DataFrame Basics



Apache Parquet

- El formato Parquet es un formato de almacenamiento en columnas enfocado en optimizar el procesamiento y modelado de datos.
- Un fichero de parquet esta compuesto por tres piezas:
 - Row Group: conjunto de filas en formato columnar.
 - Column chunk: datos de una columna en un grupo, se puede leer de manera independiente para optimizar las lecturas.
 - Page: almacenamiento de los datos.
- Spark optimiza las lecturas:
 - Permite leer los distintos row group de manera independiente, asignado estos datos a distintas tareas, que son distribuidas en el cluster.
 - Si seleccionamos unas columnas especificas, el filtro es empujado hacia el lector de parquet permitiendo leer únicamente esas columnas.



Utilizado para procesamiento en BATCH

https://parquet.apache.org/

SparkSQL: Apache Parquet

- Spark soporta de manera nativa la lectura y escritura en ficheros Parquet.
- Leyendo Parquet las columnas son marcadas como nullable, por razones de compatibilidad.

PARTICIONADO

- Parquet permite el particionado de datos para optimizar sus lecturas.
- Los datos son almacenados en distintos directorios, codificando los valores en el nombre del directorio, para cada partición.

```
path
└─ to
    └── table
           gender=male
               country=US
                └─ data.parquet
                country=CN
                └─ data.parquet
            gender=female
               country=US
                └─ data.parquet
                country=CN
                └─ data.parquet
```



Apache AVRO

- Formato binario para serialización de datos.
- Funciona con schemas: cuando se lee un AVRO, el schema con el que se escribió esta siempre presente.
 Permitiendo no producir overheads y realizar la serialización rápida y con poco coste de tamaño.
- Normalmente cuando se almacena un fichero de AVRO, el esquema se guarda con el, y de esta manera siempre esta presente. Aunque puede ser leído por otro schema compatible.

 Los schemas de AVRO están escritos en formato JSON.

Utilizado para procesamiento en Streaming, soporta procesamiento BATCH en ficheros

```
"type": "record",
"name": "userInfo",
"namespace": "my.example",
"fields": [
 { "name": "username", "type": "string", "default": "NONE" },
 { "name": "age", "type": "int", "default": - | },
 { "name": "phone", "type": "string", "default": "NONE" },
 { "name": "housenum", "type": "string", "default": "NONE" },
 { "name": "address", "type":
    "type": "record", "name": "mailing_address", "fields":
       { "name": "street", "type": "string", "default": "NONE" },
       { "name": "city", "type": "string", "default": "NONE" },
       { "name": "state_prov", "type": "string", "default": "NONE" },
       { "name": "country", "type": "string", "default": "NONE" },
       { "name": "zip", "type": "string", "default": "NONE" }
   }, "default": {}
```



SparkSQL: Apache AVRO

 SparkSQL necesita de una extensión para poder trabajar con Avro y poder hacer .format("avro")

```
libraryDependencies += "org.apache.spark"
%% "spark-avro" % "3.0.1"
```

 Avro permite indicar el tema mediante options a la hora de hacer los read/write, mediante la propiedad avroSchema.

Avro type	Spark SQL type
boolean	BooleanType
int	IntegerType
long	LongType
float	FloatType
double	DoubleType
string	StringType
enum	StringType
fixed	BinaryType
bytes	BinaryType
record	StructType
array	ArrayType
map	МарТуре



SparkSQL: Apache AVRO

Avro también tiene tipos lógicos para soportar estructuras de datos más complejas.

Avro logical type	Avro type	Spark SQL type
date	int	DateType
timestamp-millis	long	TimestampType
timestamp-micros	long	TimestampType
decimal	fixed	DecimalType
decimal	bytes	DecimalType

- El tipo union en Avro se utiliza para indicar que un campo puede tener distintos valores de tipo, spark realiza las siguientes traducciones:
 - union(int, long) → LongType
 - union(float, double) → DoubleType
 - union(any, null) → El equivalente en spark, pero la columna es marcada como nullable.
 - Si la union es de dos tipos distintos, son consideradas complejas y se traducen en una estructura con distintos miembros. union(int, string) → member0 (int), member1 (string)



Read More Data Sources

- > PARQUET
- > AVRO

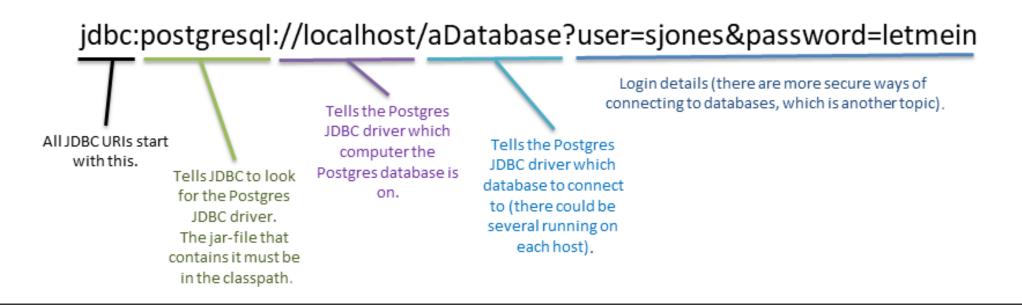


JDBC Connection

- Spark permite obtener/escribir datos desde/hacia base de datos relacionales mediante conexión JDBC.
- Necesita incluir el driver JDBC como librería adicional

```
libraryDependencies += "org.postgresql" % "postgresql" % "42.2.16"
libraryDependencies += "mysql" % "mysql-connector-java" % "8.0.21"
```

- El driver se indica mediante la opción driver, tanto en la lectura como escritura.
- La opción url es usada para indicar la cadena de conexión en formato URI.



Read Data From Google SQL



Sensor data with SparkSQL

- DataFrame API
- > SQL
- Dataset API



Built-In Functions



UDFs



spark-sql CLI

