

Apache Spark



KEEPCODING
Tech School

Antonio Gutiérrez López
gutierrez.lopez.ant@gmail.com



Spark SQL



Antonio Gutiérrez López
gutierrez.lopez.ant@gmail.com

Spark SQL: Basics

- Spark SQL es un módulo 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 RDD.
- Existen varias formas para trabajar con Spark SQL:
 - **Dataframe:** Conjunto de datos organizados en columnas. Es equivalente a una tabla relacional. Pueden ser contruidos desde: ficheros estructurados, tablas en Hive, base de datos externas, o RDDs 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 sintaxis SQL

Spark SQL: Proyecto Básico

Para empezar un proyecto de Spark, necesitamos añadir sus dependencias en nuestro proyecto de sbt en el IDE:

- Usaremos la última versión de Scala 2.12.x disponible. Actualmente tenemos la versión **2.12.17**
- Añadimos la dependencia de **spark-core** en nuestro **build.sbt**:

```
libraryDependencies += "org.apache.spark" %% "spark-core" % "3.3.0"  
libraryDependencies += "org.apache.spark" %% "spark-sql" % "3.3.0"
```

- Una vez tenemos las dependencias, podemos crear un objeto principal con un método **main**, necesitaremos crear una nueva **sparkSession**.

Spark SQL: Proyecto Básico

```
package io.keepcoding.spark.sql

import org.apache.spark.sql.Session

object SparkSqlBaseProject {
  def main(args: Array[String]): Unit = {
    val spark = Session
      .builder()
      .appName("Spark SQL KeepCoding")
      .getOrCreate()
    import spark.implicits._
    // <code>
    spark.close()
  }
}
```

Exercise 1

Spark SQL: Read Multiple Data Sources

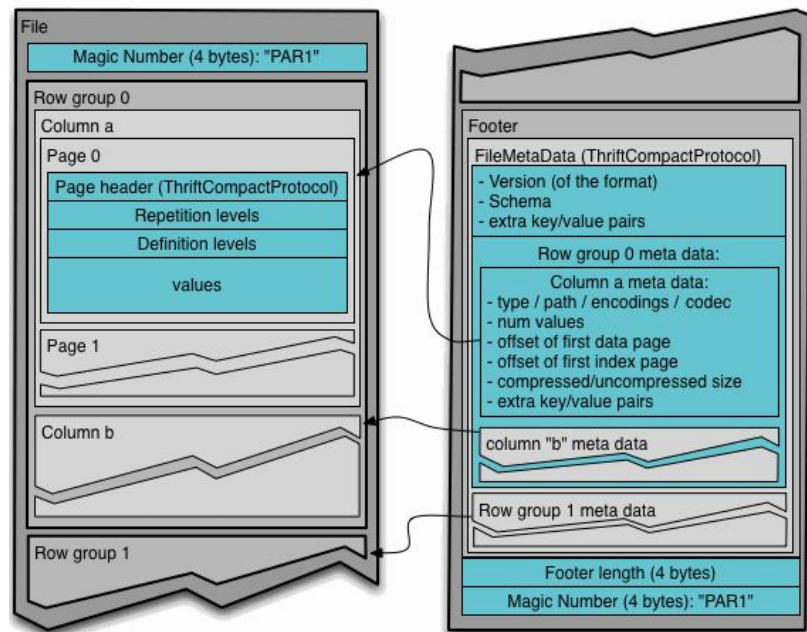
- CSV
- JSON

| Exercise 2

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 está 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, asignando estos datos a distintas tareas, que son distribuidas en el cluster.
 - Si seleccionamos unas columnas específicas, el filtro es empujado hacia el lector de parquet permitiendo leer únicamente esas columnas.



Utilizado para procesamiento
BATCH

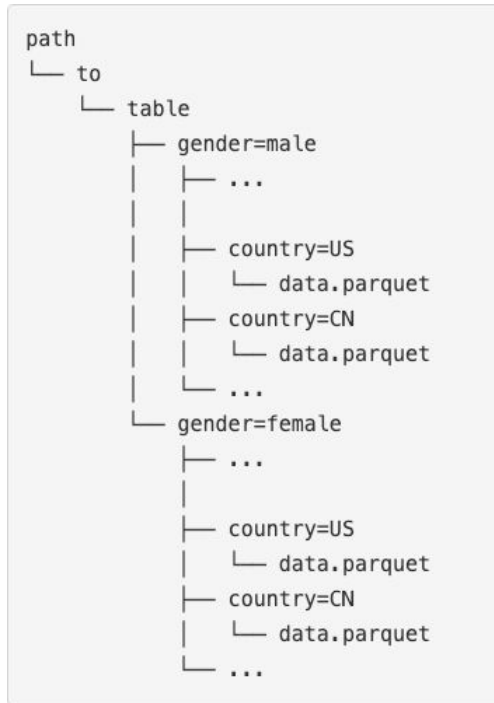
<https://parquet.apache.org>

Spark SQL: 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.



Apache AVRO

- Formato binario para serialización de datos.
- **Funciona con schemas:** cuando se lee un AVRO, el schema con el que se escribió está 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 él, y de esta manera siempre está 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": -1},
    {"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": {}
  ]
}
```

Spark SQL: Apache AVRO

- Spark SQL 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 esquema 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	MapType

Spark SQL: 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)**

| Exercise 3

Spark SQL: Read More Data Sources

- PARQUET
- AVRO

Spark SQL: JDBC Connection

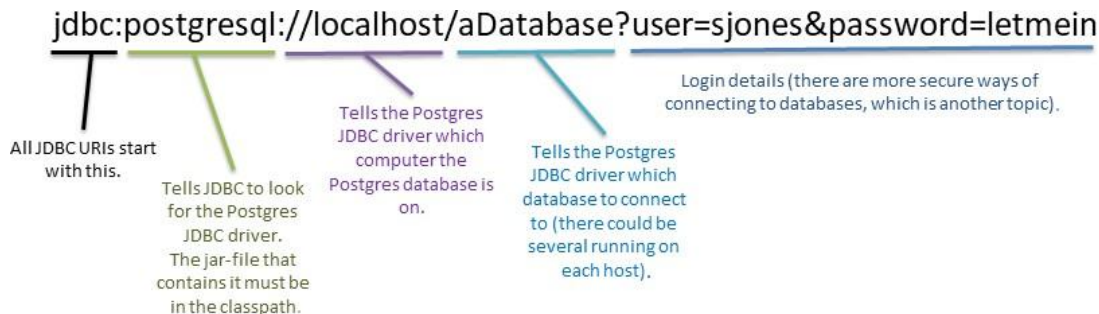
- Spark permite obtener/escribir datos desde/hacia base de datos relacionales mediante conexión JDBC.

- 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.



| Exercise 4

Spark SQL: Read Data from Google SQL

| Exercise 5

Sensor data with SparkSQL

- DataFrame API
- SQL
- Dataset API

| Exercise 6

Spark SQL: Built-In functions

| Exercise 7

Spark SQL: UDFs

| Exercise 8

Spark SQL CLI



KEEPCODING

Tech School

Madrid | Barcelona | Bogotá

Datos de contacto