Spark SQL

Introduction

- Spark SQL is a module for structured data processing. It provides Spark with more information about the both the data and the computation being performed.
- We can interact with Spark SQL with *SQL*, *`Dataset`* (and *`DataFrames`*) API and other ways.
    - We can interact with the SQL interface via a programming language and the data set will be returned as a `Dataset`/`DataFrame`. We can also interact with the SQL interface using the *command-line* or over JDBC/ODBC.
- When computing a result the same execution engine is used, independent of which API/language you are using to express the computation. This unification means that developers can easily switch back and forth.

Creating DataFrames

- With a SparkSession, we can create DataFrames from an existing RDD, from a Hive table or from other Spark data sources.
- JavaSparkSQLExample1.java in the folder Spark/java-sql-example
- DataFrames provide a domain-specific language for structured data manipulation. DataFrames are just Dataset of Rows in Scala and Java API. These operations are also referred as "untyped transformations" in contrast to "typed transformations" that come with strongly typed Scala/Java Datasets.
- See same example as above for some basic examples of structured data processing using Datasets. Here are some code snippets:

```java
df.printSchema();

// Select only the "name" column
df.select("name").show();

// Select everybody, but increment the age by 1
df.select(col("name"), col("age").plus(1)).show();

// Select people older than 21
df.filter(col("age").gt(21)).show();

// Count people by age
df.groupBy("age").count().show();
```

Running SQL Queries Programmatically

- The `sql` function on a `SparkSession` enables applications to run SQL queries programmatically and returns the result as a `Dataset<Row>`. See same example as before. Here is a code snippet.

```
df.createOrReplaceTempView("people");
Dataset<Row> sqlDF = spark.sql("SELECT * FROM people"
    );
sqlDF.show();
```

- Create a global temporary view to have one that is shared among all sessions while a Spark application runs. Temporary views (like above) are session-scoped and will disappear if the session that creates it terminates.

```
df.createGlobalTempView("people");
spark.sql("SELECT * FROM global_temp.people").show();
spark.newSession().sql("SELECT * FROM global_temp.
    people").show();
```

Creating `Datasets`

- `Dataset`s are similar to RDDs, however, instead of using Java serialization or Kryo they use a specialized `Encoder` to serialize the objects for processing or transmitting over the network
- Encoders are code generated dynamically and use a format that allows Spark to perform many operations like filtering, sorting and hashing without deserializing the bytes back into an object.
- See example: JavaSQLExample2.java

Interoperability with RDDs (1)

- Spark SQL supports two different methods for converting existing RDDs into `Dataset`s.
  - Use reflection to infer the schema of an RDD that contains specific types of objects. This reflection-based approach leads to more concise code and works well when we already know the schema.
  - Use a programmatic interface that allows us to construct a schema and then apply it to an existing RDD. While this method is more verbose, it allows us to construct `Dataset`s when the columns and their types are not known until run time.

Interoperability with RDDs (2)

- Spark SQL supports automatically converting an RDD of `JavaBeans` into a `DataFrame`.
- The `BeanInfo`, obtained using reflection, defines the schema of the table. Currently, Spark SQL does not support `JavaBeans` that contain Map field(s). Nested `JavaBeans` and `List` or `Array` fields are supported though.
- See the method `runInferSchemaExample` in JavaSQLExample3.java

What is a JavaBean?

- It is a reusable software component written in Java.
- JavaBeans are classes that encapsulate many objects into a single object (the bean) by following a standard. The name "Bean" was given to encompass this standard, which is defined below:
  - Must implement Serializable
  - It should have a public no-arg constructor
  - All attributes must be private with public getters and setter methods
  - Rules for setter methods:
    - It should be public
    - The return-type should be void
    - The method name should be prefixed with set
    - It should take some argument i.e. it should not be no-arg method
  - Rules for getter methods:
    - It should be public
    - The return-type should not be void
    - The method name should be prefixed with get
    - For boolean attributes, the name can be prefixed with “get" or “is" but preferred prefix would be “is"
    - It should not take any argument

Interoperability with RDDs (3)

- When `JavaBean` classes cannot be defined ahead of time, a `Dataset`<Row> can be created programmatically with three steps.
  - **Step 1**: Create an RDD of Rows from the original RDD
  - **Step 2**: Create the schema represented by a StructType matching the structure of Rows in the RDD created in Step 1
  - **Step 3**: Apply the schema to the RDD of Rows via `createDataFrame` method provided by SparkSession
- See the method `runProgrammaticSchemaExample` in JavaSQLExample3.java