Spark SQL, DataFrames and Datasets Guide

Spark SQL is a Spark module for structured data processing. Unlike the basic Spark RDD API, the interfaces provided by Spark SQL provide Spark with more information about the structure of both the data and the computation being performed. Internally, Spark SQL uses this extra information to perform extra optimizations. There are several ways to interact with Spark SQL including SQL and the Dataset API. 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 between different APIs based on which provides the most natural way to express a given transformation.

All of the examples on this page use sample data included in the Spark distribution and can be run in the `spark-shell`, `pyspark` shell, or `sparkR` shell.

SQL

One use of Spark SQL is to execute SQL queries. Spark SQL can also be used to read data from an existing Hive installation. For more on how to configure this feature, please refer to the Hive Tables section. When running SQL from within another programming language the results will be returned as a Dataset/DataFrame. You can also interact with the SQL interface using the command-line or over JDBC/ODBC.

Datasets and DataFrames

A Dataset is a distributed collection of data. Dataset is a new interface added in Spark 1.6 that provides the benefits of RDDs (strong typing, ability to use powerful lambda functions) with the benefits of Spark SQL's optimized execution engine. A Dataset can be constructed from JVM objects and then manipulated using functional transformations (`map`, `flatMap`, `filter`, etc.). The Dataset API is available in Scala and Java. Python does not have the support for the Dataset API. But due to Python's dynamic nature, many of the benefits of the Dataset API are already available (i.e. you can access the field of a row by name naturally `row.columnName`). The case for R is similar.

A DataFrame is a *Dataset* 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. The DataFrame API is available in Scala, Java, Python, and R. In Scala and Java, a DataFrame is represented by a Dataset of `Row`s. In the Scala API, `DataFrame` is simply a type alias of `Dataset[Row]`. While, in Java API, users need to use `Dataset<Row>` to represent a `DataFrame`.

Throughout this document, we will often refer to Scala/Java Datasets of `Row`s as DataFrames.