EXPONAM DATA SOURCE FOR SPARK 2

Version 2.5

OVERVIEW

This document describes the Exponam's .BIG integration with Spark version 2.x.

Obtaining the Data Source

The data source is available as a free download from Exponam.

1) Navigate to https://www.exponam.com and click the Developer Tools button.
2) Add the "Exponam Data Source for Spark 2" to your cart.
3) Proceed to checkout.
4) Enter the details on the checkout form and place the order.
5) You will receive an email with a download link for the jar file.
6) Download the jar file to an appropriate location on your computer.

Obtaining a license file

To unlock the full capabilities of the data source, obtain a license file from Exponam. Contact Exponam at either:

- https://www.exponam.com/#contact, or
- sales@exponam.com

In the absence of a license file, the data source is still functional. Load operations using the data source are fully available even without a license file. However, without a license file the save operations are narrowed to a demonstration mode. Without a license file, save operations with the data source will:

- Apply an upper limit to the number of rows in the output file of 3 million.
- Restrict encryption.
- Restrict the incorporation of custom story files, instead applying an Exponam-standard story file to each file.
- Mark the output file producer as Exponam.

Using the Data Source

In summary, the data source enables you to load data from .BIG files into a Spark cluster, and save data from the Spark cluster out to newly created .BIG files.

Load and save operations each have mandatory and optional parameters that are easily expressed using Spark syntax and operations, per the descriptions below. Note that Java syntax is used here by way of example; Spark's polyglot interfaces allow the documented options in all supported languages.

Loading From .BIG Files

When data is loaded into Spark, the schema for the DataFrame is created based on the column data types for the source .BIG file sheet. In other words, the data content of the .BIG file is faithfully carried into the cluster. Additionally, the data source implements advanced Spark features including column filtering and predicate pushdown. This means that work that would otherwise need to be performed within Spark is performed within the data source itself, yielding faster load performance and reducing the time and cost within the cluster itself.

The following options are available using standard SparkSession load syntax:

```
SparkSession spark = …as appropriate for your Spark installation…
Dataset dataset = spark.read()
   .format("big")     // mandatory to indicate Exponam data source
   .option(password)  // optional password
   .load(filePath)    // mandatory file path
   .filter(…)         // optional Column filter(s); default is all rows
   .select(…)         // optional column(s); default is all columns
```

As a specific example, the following loads the Start and End columns from sample.big with password XYZ, for those rows where Start > 100:

```
Dataset dataset = spark.read()
   .format("big")
   .option("XYZ")
   .load("sample.big")
   .filter(new Column("Start).gt(100))
   .select("Start", "End");
```

Saving To .BIG Files

When data is saved from Spark, one or more .BIG files are produced, subject to how the source data is partitioned. One .BIG file is produced for each partition. It is of course possible in Spark to repartition a DataFrame to a single partition, which will yield a single .BIG file as output. However, it can also be advantageous for large data egress to use multiple partitions, and thereby parallelize the operation within Spark.

The following options are available using standard SparkSession save syntax (example assumes an existing Dataset):

```
dataset
   .write()
   .mode(mode)                        // see explanation below
   .format("big")                     // mandatory to indicate Exponam data source
   .option("license", filePath)       // optional license file
   .option("password", password)      // optional password
   .option("worksheetName", name)     // optional name for the sheet created
   .option("partitionIdWidth", width) // optional; see explanation below
   .option("optimizationLevel", level)// optional; see explanation below
   .option("provenanceLevel", level)  // optional; see explanation below
   .save(filePath);                   // mandatory output file path
```

- Explanation of parameters:
  - mode
    - Supported: SaveMode.Ignore, SaveMode.ErrorIfExists, SaveMode.Overwrite
    - Not supported: SaveMode.Append
  - partitionIdWidth (default is 6)

        - Output file names are created using the filePath given to the save operation. Each output file name includes the partition id for which the file was generated. The partitionIdWidth is a value from 1 to 6 that determines the number of characters that will be used to carry that partition id into the file name. For example, if a file is generated for partition 30, and the partitionIdWidth is 5, then this component of the output file name is left-padded to be "00030".
    - optimizationLevel is the degree of optimization to be performed during file generation, per the following (default is 1)
        - 1 : fast output generation with some file size reduction - simple data exchange where speed and small output size are both priorities
        - 2 : smallest possible output file size - simple data exchange where small output size is the priority
        - 3 : small output size with minimal search indexes - data exchange and edge data with minor gains in query performance
        - 4 : small output size with minimal search indexes - data exchange and edge data with incrementally greater gains in query performance
        - 5 : small output size with minimal search indexes - data exchange and edge data with incrementally greater gains in query performance
        - 6 : medium output size with light search indexes - data exchange and edge data with incrementally greater gains in query performance
        - 7 : medium output size with light search indexes - data exchange and edge data with incrementally greater gains in query performance
        - 8 : medium output size with light search indexes - data exchange and edge data with incrementally greater gains in query performance
        - 9 : medium output size with light search indexes - data exchange and edge data with incrementally greater gains in query performance
        - 10 : larger output size with moderate search indexes - for more demanding query environments
        - 11 : larger output size with moderate search indexes - for even more demanding query environments
        - 12 : larger output size with moderate search indexes - for even more demanding query environments
        - 13 : larger output size with comprehensive search indexes - for stand-alone data warehouse extracts and similar usages
        - 14 : larger output size with comprehensive search indexes - for stand-alone data warehouse extracts and similar usages
        - 15 : largest output size with fullest search indexes - for the most demanding query environments
    - provenanceLevel governs the level of detail emitted into provenance structures, per the following (default is 1):
        - 0 : summary only
        - 1 : summary and high level details
        - 2 : summary and all details

In addition to specifying parameters for the save operation as shown above, certain parameters can be set within the Spark configuration. This means that these parameters do not have to explicitly set within each save command. The use of configuration settings can simplify your logic, and is recommended as a best practice. The supported options are shown by way of example, where spark is a SparkSession instance:

```
spark.conf().set("spark.exponam.big.license", licenseFile)
spark.conf().set("spark.exponam.big.optimization.level", level)
spark.conf().set("spark.exponam.big.provenance.level", level)
spark.conf().set("spark.exponam.big.partitionIdWidth", width)
```