Assignment: Flavors of Spark Data-Models

Avraham Leff

COM3580: Spring 2024

1 Assignment-Specific Packaging

The general packaging is unchanged from the basic "Homework Requirements" (see slides from first lecture and "Homework Policies for COM 3580" on Piazza).

> In this iteration, I **will not** pull-and-execute your code.
> However: you must check in your **compilable Java** code and specified **output directories**.

This assignment's "DIR" **must be named** *FlavorsOfSpark*, and the writeup file must be named *FlavorsOfSpark.pdf*, residing in the specified DIR.

> This assignment **must be done in Java**, so please review the general requirements document in how to do such assignments. The package name is `edu.yu.mdm.spark` (remember to use the correct directory structure per the "Homework Policies" document), and your application class must be named *FlavorsOfSpark.java*. I plan to grade based on the writeup file, but will "check your work" as necessary by examining, and possibly running, the code.

1.1 Software Versions

You will use the following versions of the software stack. Let me know if you think that the list is under-specified.

```
<dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-core_2.12</artifactId>
<version>3.5.1</version>
</dependency>

<dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-sql_2.12</artifactId>
<version>3.5.1</version>
</dependency>
```

2 Context & Motivation

In lecture we discussed how Spark includes several *data-models*:

- RDDs
- Dataset
- DataFrame

Each of these data-models has strengths and weaknesses relative to one another (although RDDs are definitely "deprecated" from a programming point of view).

This assignment will give you "introductory" experience with using Spark and its data models. I hope that the assignment will give you insight as to the tradeoffs between the representations.

3 Requirements

Your program will consist of the following phases:

1. Ingest a csv file (filename to be supplied on the command-line invocation, the csv "header" is shown below, along with some sample data).

   Download the input file (from data/books.csv.)

   The file contains n lines: the first is a "header line" matching the line below.

   ```
   id,authorId,title,releaseDate,link
   ```

   The input format is straight-forward, but note:

   - A given row may be missing fields (indicated by the field being omitted except for the delimiting comma). The Spark "ingester" is up to the task; your implementation is responsible for taking possibly missing fields into account.
   - Be sure to handle the `releaseDate` formats correctly!

2. Ingest the input file, and construct the required Spark data-sets (details below). You decide how the ingestion is done, and how/whether to convert one data-model instance to another during your program's execution. You may constructs as many intermediary stages as you please: I only want to see the final results.

   You'll be required to do specified transformations and to output specified data-sets to output directories.

3. Translate a set of "natural language" queries into invocations of a given data-set, showing me your formulated query and the results.

3.1 Output

- All program output must look "pretty": see this useful tip.

1. Create a $JavaRDD < String >$ containing only the book titles from the input file. You may include the "title" from the csv header line.

   Write that JavaRDD to your writeup file.

2. Create a $JavaRDD < String >$ containing only book titles that contain "Harry Potter" (anywhere in the title field).

   Write that JavaRDD to your writeup file, and save to a text files in a top-level directory rooted at DIR and named `rddOutput`. Your writeup file must contain the nicely formatted code that produced this result.

3. Create a $Dataset < Row >$ from the input file.

   Use `df.showString(5, 0, true)` to write the specified contents to your writeup file.

4. Invoking `printSchema` on the previous data-set should show that `releaseDate` is typed as String. You'll now fix that while creating a $Dataset < Book >$ that will have explicit specification of the following typed properties

   - id: integer
   - authorId: integer
   - link: String
   - title: String
   - releaseDate: `java.util.Date`

   Create that $Dataset < Book >$, and invoke `showString(5, 0, true)` to write tuples to your writeup file. Do the same for `printSchema`.

   Write the Dataset, in JSON format, to a directory named `dataSet` rooted in DIR.

5. Examining `printSchema` on the above Dataset should show that `releaseDate` looks bizarre. Convert this Dataset to a $Dataset < Row >$ whose columns have been transformed to

   - Not include the `releaseDate` column
   - Include a `niceDate` column formatted as `yyyy-MM-dd`.

   Create that $Dataset < Row >$, and invoke `showString(25, 0, true)` to write tuples to your writeup file. Do the same for `printSchema`.

   Write the Dataset, in CSV format, to a directory named `niceDate` rooted in DIR.

6. Output the entire previous Dataset, sorted by descending `niceDate`.

7. Create a $Dataset < Row >$ containing only the rows with a `null` "niceDate". Invoke `showString(25, 0, true)` to write the Dataset to your writeup file.

8. Formulate a query that returns the `id, count` of the author with the most books.

   Invoke `showString(1, 0, true)` to write the Dataset to your writeup file.