Ask Bigger Questions

CAP - Developing with Spark and Hadoop:

Homework Assignment Guide for Students

Homework: Use Spark SQL for ETL

Files and Data Used in this Homework

Exercise Directory: `$DEV1/exercises/spark-sql`

MySQL table: `loudacre.webpage`

Output directory (HDFS): `/loudacre/webpage_files`

In this exercise you will use Spark SQL to load data from MySQL, process it, and store it to HDFS.

Review the Data in MySQL

Review the data currently in the MySQL `loudacre.mysql` table.

1. List the columns and types in the table:

```
$ mysql -utraining -ptraining loudacre \
-e"describe webpage"
```

2. View the first few rows from the table:

```
$ mysql -utraining -ptraining loudacre \
-e"select * from webpage limit 5"
```

Note that the data in the `associated_files` column is a comma-delimited string. Loudacre would like to make this data available in an Impala table, but in order to perform required analysis, the `associated_files` data must be extracted and normalized. Your goal in the next section is to use Spark SQL to extract the data in the column, split the string, and create a new dataset in HDFS containing each web page number, and its associated files in separate rows.

Load the Data from MySQL

3. If necessary, start the Spark Shell.

4. Import the SQLContext class definition, and define a SQL context:

```
scala> import org.apache.spark.sql.SQLContext
scala> val sqlCtx = new SQLContext(sc)
```

```
pyspark> from pyspark.sql import SQLContext
pyspark> sqlCtx = SQLContext(sc)
```

5. Create a new DataFrame based on the `webpage` table from the database:

```
scala> val webpages=sqlCtx.load("jdbc",
Map("url"->
"jdbc:mysql://localhost/loudacre?user=training&password
=training",
"dbtable" -> "webpage"))
```

```
pyspark> webpages=sqlCtx.load(source="jdbc", \
url="jdbc:mysql://localhost/loudacre?user=training&pass
word=training", \
  dbtable="webpage")
```

6. Examine the schema of the new DataFrame by calling `webpages.printSchema()`.

7. Create a new DataFrame by selecting the `web_page_num` and `associated_files` columns from the existing DataFrame:

```
scala> val assocfiles =
webpages.select(webpages("web_page_num"),webpages("asso
ciated_files"))
```

```
python> assocfiles = \
  webpages.select(webpages.web_page_num,\
  webpages.associated_files)
```

8. In order to manipulate the data using Spark, convert the DataFrame into a to a Pair RDD using the map method. The input into the map method is a Row object. They key is the `web_page_num` value (the first value in the Row), and the value is the `associated_files` string (the second value in the Row).

   In Scala, use the correct `get` method for the type of value with the column index:

```
scala> val afilesrdd = assocfiles.map(row =>
(row.getInt(0),row.getString(1)))
```

   In Python, you can dynamically reference the column value of the Row by name:

```
pyspark> afilesrdd = assocfiles.map(lambda row: \
  (row.web_page_num,row.associated_files))
```

9. Now that you have an RDD, you can use the familiar `flatMapValues` transformation to split and extract the filenames in the `associated_files` column:

```
scala> val afilesrdd2 =
  afilesrdd.flatMapValues(filestring =>
  filestring.split(','))
```

```
pyspark> afilesrdd2 = afilesrdd\
.flatMapValues(lambda filestring:filestring.split(','))
```

10. Create a new DataFrame from the RDD:

```
scala> val afiledf = sqlCtx.createDataFrame(afilesrdd2)
```

```
pyspark> afiledf = sqlCtx.createDataFrame(afilesrdd2)
```

11. Call `printSchema` on the new DataFrame. Note that Spark SQL gave the columns generic names: **_1** and **_2**.

12. Create a new DataFrame by renaming the columns to reflect the data they hold.

In Scala, you can use the `toDF` shortcut method to create a new DataFrame based on an existing one with the columns renamed:

```
scala> val finaldf = afiledf.
  toDF("web_page_num","associated_file")
```

In Python, use the `withColumnRenamed` method to rename the two columns:

```
pyspark> finaldf = afiledf. \
  withColumnRenamed('_1','web_page_num'). \
  withColumnRenamed('_2','associated_file')
```

13. Call `printSchema` to confirm that the new DataFrame has the correct column names.

14. Your final DataFrame contains the processed data, so save it in Parquet format (the default) in `/loudacre/webpage_files`. (The code is the same in Scala and Python)

```
> finaldf.save("/loudacre/webpage_files")
```

View the Output

15. Using Hue or the HDFS command line, list the files that were saved by Spark SQL.

16. Execute the following DDL command in Impala to create a table to access the new Parquet dataset:

```
CREATE EXTERNAL TABLE webpage_files LIKE PARQUET
 '/loudacre/webpage_files/part-r-00001.parquet'
  STORED AS PARQUET
  LOCATION '/loudacre/webpage_files'
```

17. Try executing a simple query to confirm the table is set up correctly.

This is the end of the Homework