Advanced Databases
PySpark Tutorial

The objective of this document is to provide you guidelines on how to develop data processing pipelines using PySpark. Certain level of familiarity with the Python programming language is assumed, otherwise we encourage you to take a look at the plethora of tutorials that are available online (e.g., `https://docs.python.org/3/tutorial/index.html`). Before proceeding, you should have installed the necessary software (see the installation manual provided). Also, we assume that you will be running the code seen here in the Python interactive shell.

1 Spark functionalities

Here, we provide you some guidelines and tips that might be useful in the project. We further recommend you to take a look at the following resources:

- Spark programming guide(`https://spark.apache.org/docs/latest/sql-programming-guide.html`)
- PySpark reference (`https://spark.apache.org/docs/latest/api/python/pyspark.html`)

1.1 PySpark

PySpark applications start with initializing a SparkSession which is the entry point of PySpark.

```
>>> from pyspark.sql import SparkSession
>>> if not "spark" in globals():
      spark = SparkSession.builder.getOrCreate()
```

1.2 Dataframes

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 con structed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs. We can create PySpark DataFrame from a list of rows:

```
>>> from datetime import datetime, date
>>> from pyspark.sql import Row
>>> df = spark.createDataFrame([ \
       Row(a=1, b=2., c="string1", d=date(2000, 1, 1), e=datetime
              (2000, 1, 1, 12, 0)), \
       Row(a=2, b=3., c="string2", d=date(2000, 2, 1), e=datetime
              (2000, 1, 2, 12, 0)), \
       Row(a=4, b=5., c="string3", d=date(2000, 3, 1), e=datetime
              (2000, 1, 3, 12, 0))
])
```

Viewing the data

```
# Show the data
>>> df.show()
```

The top rows of a DataFrame can be displayed using DataFrame.show()

```
>>> df.show(10)
```

The rows can also be shown vertically. This is useful when rows are too long to show horizontally

```
>>> df.show(4, vertical=True)
```

You can see the DataFrame's schema and column names as follows

```
>>> df.printSchema()
>>> df.columns
```

Show the summary of the DataFrame using 'describe' function

```
>>> df.select("a", "b", "c").describe().show()
```

Selecting and accessing data

PySpark DataFrame is lazily evaluated and simply selecting a column does not trigger the computation but it returns a Column instance.

```
>>> df.a
```

These Columns can be used to select the columns from a DataFrame. For example, DataFrame.select() takes the Column instances that returns another DataFrame.

```
>>> df.select("c")
```

Note that ‘column‘ from ‘pyspark.sql.functions‘ is a function and ‘Column‘ from ‘pyspark.sql‘ is a type. Then, we can only use ‘column(c)‘ in a select, as ‘Column(c)‘ is not valid since Column is only used to give a data type to any python object

```
>>> from pyspark.sql.functions import column
>>> df.select(column("c")).show()
```

Assign a new Column instance

```
>>> df.withColumn( upper_c  , upper(df.c)).show()
```

Grouping data

PySpark DataFrame also provides a way of handling grouped data by using the common approach, split-apply-combine strategy. It groups the data by a certain condition applies a function to each group and then combines them back to the DataFrame.

```
>>> df.groupby( color  ).avg().show()
```

Getting data in/out

```
# Write data to CSV
>>> df.write.csv(foo. csv , header=True)
# Read from CSV
>>> spark.read.csv(foo. csv , header=True).show()
```

Working with SQL

DataFrame and Spark SQL share the same execution engine so they can be interchangeably used. For example, you can register the DataFrame as a table and run a SQL easily as below:

```
>>> df.createOrReplaceTempView("tableA")
>>> spark.sql("SELECT␣count(*)␣from␣tableA").show()
```

1.3 SparkSQL

SparkSQL is a Spark module for structured data processing. One use of Spark SQL is to execute SQL queries. When running SQL from within another programming language the results will be returned as a DataFrame. To start using SparkSQL we will configure the SparkSession object using a SparkConf object. One of the configurations we will add is the reference to the PostgreSQL driver. First import the required libraries

```
>>> import os
>>> import sys
>>> import pyspark
>>> from pyspark.sql import SparkSession
>>> from pyspark import SparkConf
```

Configure the environment variables. Make sure you edit them according to the configuration in your environment. The "hadoop_home" folder is required for Windows and you can find it in the resources folder inside the CodeSkeleton zip file in Moodle. You have to correctly configure the "filepath" to point to the "hadoop_home" folder.

```
>>> HADOOP_HOME = "filepath/hadoop_home"
>>> PYSPARK_PYTHON = "python3.6"
>>> PYSPARK_DRIVER_PYTHON = "python3.6"

>>> os.environ["HADOOP_HOME"] = HADOOP_HOME
>>> sys.path.append(HADOOP_HOME + "\\bin")
>>> os.environ["PYSPARK_PYTHON"] = PYSPARK_PYTHON
>>> os.environ["PYSPARK_DRIVER_PYTHON"] = PYSPARK_DRIVER_PYTHON
```

Need to provide the path to the PostgreSQL driver to create a connection later on. You can find the postgresql driver in the resources folder inside the CodeSkeleton zip file in Moodle. You have to correctly configure the "filepath" to point to the postgresql driver.

```
>>> POSTGRESQL_DRIVER_PATH = "filepath\postgresql-42.2.8.jar"
```

Create the configuration

```
>>> conf = SparkConf() \
      .set("spark.master","local") \
      .set("spark.app.name","DBALab") \
      .set("spark.jars",POSTGRESQL_DRIVER_PATH)
```

Initialize the Spark Session

```
spark = SparkSession.builder.config(conf=conf).getOrCreate()
```

Define the PostgreSQL database connection properties. Make sure you edit the username and password accordingly.

```
>>> db_properties = { \
       "driver": "org.postgresql.Driver", \
       "url": "jdbc:postgresql://postgresfib.fib.upc.edu␣\
␣␣␣␣␣␣␣␣:6433/AMOS?sslmode=require", \
       "user": "username", \
       "password": "password" \
   }
```

Read data from the PostgreSQL database into a DataFrame

```
data = spark.read.jdbc(url=db_properties["url"], \
       table="public.operationinterruption", \
       properties=db_properties)
```

Data obtained from a database can be manipulated using SparkSQL's operations.

```
>>> df = data.select("flightid","airport","duration")
>>> df.show(10)
```

Stop the Spark session when you're done

```
>>> spark.stop()
```