Lab : Apache Spark Date/Time Functions

Pre-reqs:

- Google Chrome (Recommended)

Prerequisites

We need following packages to perform the lab exercise:

- Java Development Kit
- pyspark

JAVA

Verify the installation with: `java -version`

You'll see the following output:

```
java version "1.8.0_201"
Java(TM) SE Runtime Environment (build 1.8.0_201-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode)
```

Date/Time functions

Let us now look at Date/Time functions to manipulate, extract and perform arithmetic operations on date and time. As in collection functions, we shall be using the Spark shell to demonstrate Date/Time functions as well.

Fire up the spark-shell from the terminal `spark-shell`

Step 1: Let us first create a collection with data as shown below. Please make sure you have the imports from the previous section already imported. You will have to import them again if you have closed the Spark Session.

```
val dates = Seq(
  (1, "Ernesto", "2015-09-24"),
  (2, "Lee", "1985-05-16"),
  (3, "John", "2012-07-16"),
  (4, "Doe", "1914-08-02"))
```

Next, let us convert the collection to dataset using the toDS method and rename the column as shown below using the withColumnRenamed method. The default column names for dataset are monotonically increasing numbers like _1, _2, _3 etc.

```
val datesDS = dates.toDS().withColumnRenamed("_1", "id").withColumnRenamed("_2", "name").withColumnRenamed("_3", "date")
```

Let us check the schema using the printSchema method, so that we can compare the datatype for date column in the next step.

```
datesDS.printSchema()
```

As you can see, the date is of type String.

Step 2: Let us cast the date column and convert it to date type using the cast function as shown below.

`val casted = datesDS.select($"id", $"name", $"date".cast("date")).cache()`

Let us print the schema to check if we were able to successfully convert the date column from String type to Date type. Let us also use the show function to view the dataset.

```
casted.printSchema()
casted.show()
```

As you can see, we have successfully casted the date column as date type.

Step 3: Let us now extract the indivudual attributes from the date object such as day, month, year etc. We shall be using various functions to add columns for each function using the withColumn method.

Enter into the paste mode and execute the following code. `:paste`

Note: After pasting following code in the scala terminal, Press `Ctrl` + `D` to run code.

```
val  extracted = casted
  .withColumn("year", year($"date"))
  .withColumn("month", month($"date"))
  .withColumn("dayOfYear", dayofyear($"date"))
  .withColumn("quarter", quarter($"date"))
  .withColumn("weekOfYear", weekofyear($"date"))
```

Output

We have used the year, month and dayofyear functions to extract the extract the individual attributes from the date column. We have also used the quarter function to get which quarter the date is from and weekofyear function to get the week of which the date belongs to.

`extracted.show()`

Step 4: Let us now use the arithmetic functions to manipulate the date.

Enter into the paste mode and execute the following code. `:paste`

Note: After pasting following code in the scala terminal, Press `Ctrl` + `D` to run code.

```
val  arithmetic = casted
  .withColumn("ageInDays", datediff(current_date(), $"date"))
  .withColumn("addedDays", date_add($"date", 25))
  .withColumn("subtrDays", date_sub($"date", 16))
  .withColumn("addedMonths", add_months($"date", 4))
  .withColumn("lastDay", last_day($"date"))
  .withColumn("nextDay", next_day($"date", "tuesday"))
  .withColumn("monthsBetween", months_between(current_date(), $"date", true))
```

`arithmetic.show()`

- The datediff function is used to calculate the date difference between two dates. Here we have used the current_date method to get the present date and get the difference from the date in date column.
- The date_add and date_sub functions are used to add and subtract the number of days from the date in date column. The function takes the date column and the number of days as arguments.

- The add_months function is used to add number of months to the date in date column. The function takes the date column and the number of months as arguments.
- The last_day and next_day functions are used to get the last day of the month and next day of the month for the day of the week for the date in date column respectively. The next_day function takes date column and the day of week as arguments.
- The months_between function is used to get the number of months between two days. We have used the present date using current_date function and date column as the arguments.

Step 5: Next, let us use the timestamp functions. Since we have only created a date type in the previous dataset, let us create a timestamp type instead of date type. First, let us create the dataset and rename the columns as shown below.

Enter into the paste mode and execute the following code. `:paste`

Note: After pasting following code in the scala terminal, Press `Ctrl` + `D` to run code.

```
val  timeStamp = spark.createDataset(Seq(
  (1, "Ernesto", "2015-09-24 00:01:12"),
  (2, "Lee", "1985-05-16 03:04:15"),
  (3, "John", "2012-07-16 06:07:18"),
  (4, "Doe", "1914-08-02 09:10:20")
))
```

Enter into the paste mode and execute the following code. `:paste`

Note: After pasting following code in the scala terminal, Press `Ctrl` + `D` to run code.

```
val  timeStampDS = timeStamp
  .withColumnRenamed("_1", "id")
  .withColumnRenamed("_2", "name")
  .withColumnRenamed("_3", "timeStamp")
```

Let us print the schema so that we can compare it with the timestamp type in the next step.

```
timeStampDS.printSchema()
```

Step 6: Let us now convert the timestamp which is of String type to timestamp type.

```
val castedTimeStamp = timeStampDS.select($"id", $"name",
$"timeStamp".cast("timestamp")).cache()
```

Let us now print the schema and the dataset to check the casting.

```
castedTimeStamp.printSchema()
castedTimeStamp.show()
```

As you can see, we have successfully casted the timestamp column from String type to timestamp type.

Step 7: Let us now extract the attributes from timestamp column as we did for the date column couple of steps ago.

Enter into the paste mode and execute the following code. `:paste`

Note: After pasting following code in the scala terminal, Press `Ctrl` + `D` to run code.

```
val  extractedTs = timeStampDS
  .withColumn("second", second($"timeStamp"))
  .withColumn("minute", minute($"timeStamp"))
  .withColumn("hour", hour($"timeStamp"))
```

The output is shown when we use the show method.

```
extractedTs.show()
```

Step 8: Finally, let us use couple of conversion functions to convert the dates into different formats.

Enter into the paste mode and execute the following code. `:paste`

Note: After pasting following code in the scala terminal, Press `Ctrl` + `D` to run code.

```
val  conversions = timeStampDS
    .withColumn("unixTime", unix_timestamp($"timeStamp"))
    .withColumn("fromUnix", from_unixtime($"unixTime"))
```

- The unix_timestamp function is used to convert the timestamp to unix timestamp.
- The from_unixtime function is used to cconvert the unix time which we obtained above.

The output is shown when we use the show method.

```
conversions.show()
```

Task is complete!