



Assignment

Week16: Apache Spark - Streaming
Part-2

IMPORTANT

Self-assessment enables students to develop:

1. A sense of responsibility for their own learning and the ability & desire to continue learning,
2. Self-knowledge & capacity to assess their own performance critically & accurately, and
3. An understanding of how to apply their knowledge and abilities in different contexts.

All assignments are for self-assessment. Solutions will be released on every subsequent week. Once the solution is out, evaluate yourself.

No discussions/queries allowed on assignment questions in slack channel.

Note: You can raise your doubts in the subsequent week once the solution is released

TRENDYTECH 9108179578

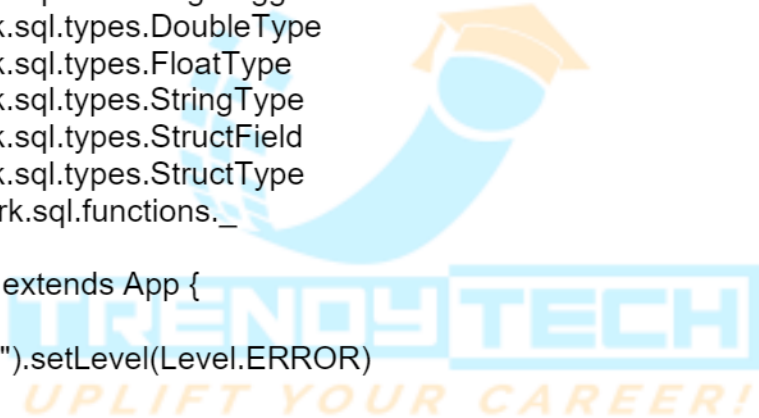
Solution1:

```
import org.apache.log4j.Level
import org.apache.log4j.Logger
import org.apache.spark.sql.Session
import org.apache.spark.sql.streaming.Trigger
import org.apache.spark.sql.types.DoubleType
import org.apache.spark.sql.types.FloatType
import org.apache.spark.sql.types.StringType
import org.apache.spark.sql.types.StructField
import org.apache.spark.sql.types.StructType
//import org.apache.spark.sql.functions._

object W16_Problem_1 extends App {

  Logger.getLogger("org").setLevel(Level.ERROR)

  val spark = SparkSession.builder()
    .master("local[2]")
    .appName("APP1")
    .config("spark.sql.shuffle.partitions", 3)
    .config("spark.streaming.stopGracefullyOnShutdown", "true")
```



TRENDYTECH 9108179578

```
.getOrCreate()
```

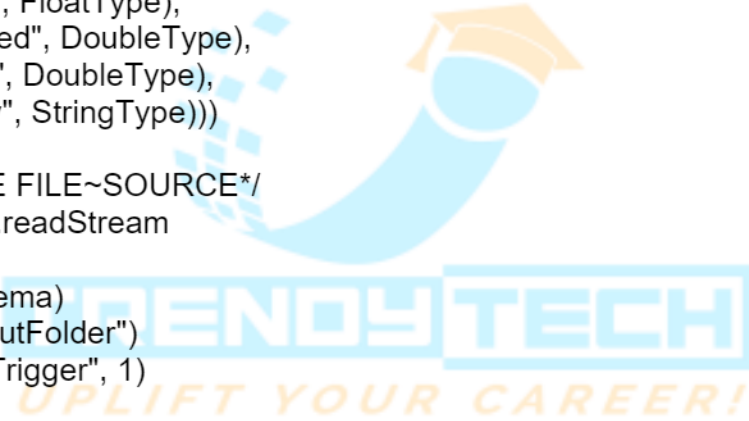
```
val weatherSchema = StructType(List(  
  StructField("DateTime", StringType),  
  StructField("Temperature", DoubleType),  
  StructField("Humidity", FloatType),  
  StructField("WindSpeed", DoubleType),  
  StructField("Pressure", DoubleType),  
  StructField("Summary", StringType)))
```

```
/**1. READ FROM THE FILE~SOURCE*/
```

```
val weatherDF = spark.readStream  
  .format("csv")  
  .schema(weatherSchema)  
  .option("path", "myInputFolder")  
  .option("maxFilesPerTrigger", 1)  
  .load()
```

```
weatherDF.printSchema()
```

```
weatherDF.createOrReplaceTempView("weatherHistory")
```



TRENDYTECH 9108179578

```
val completedweatherDF = spark.sql("select * from weatherHistory where WindSpeed < 11.0000 AND  
Summary = 'Partly Cloudy'")
```

```
/**3. WRITE TO THE SINK*/
```

```
val weatherQuery = completedweatherDF.writeStream  
  .format("csv")  
  // .format("json")  
  .outputMode("append")  
  .option("path", "myOutputFolder")  
  .option("checkpointLocation", "checkpoint-location1")  
  .trigger(Trigger.ProcessingTime("30 seconds"))  
  .start()
```

```
weatherQuery.awaitTermination()  
scala.io.StdIn.readLine()
```


```
}
```



TRENDYTECH 9108179578

Solution 2:

```
import org.apache.log4j.Level
import org.apache.log4j.Logger
import org.apache.spark.sql.Session
import org.apache.spark.sql.functions.avg
import org.apache.spark.sql.functions.col
import org.apache.spark.sql.functions.from_json
import org.apache.spark.sql.functions.window
import org.apache.spark.sql.streaming.Trigger
import org.apache.spark.sql.types.DoubleType
import org.apache.spark.sql.types.FloatType
import org.apache.spark.sql.types.StringType
import org.apache.spark.sql.types.StructField
import org.apache.spark.sql.types.StructType
import org.apache.spark.sql.types.TimestampType
//import org.apache.spark.sql.functions._
```



```
object W16_Problem_2 extends App {
```

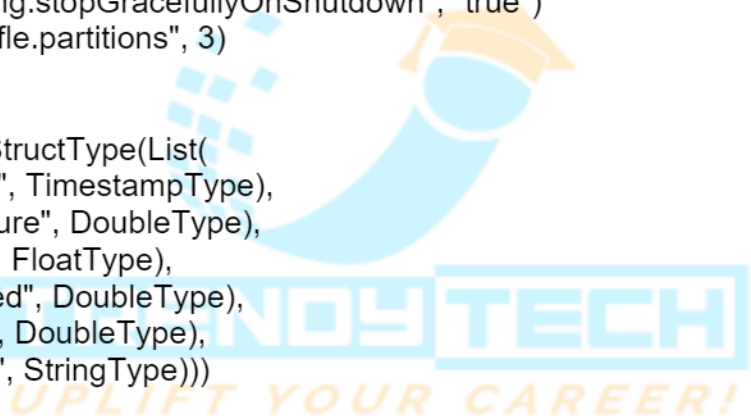
```
  Logger.getLogger("org").setLevel(Level.ERROR)
```

TRENDYTECH 9108179578

```
/**CREATING A SPARK~SESSION*/  
val spark = SparkSession.builder()  
  .master("local[2]")  
  .appName("App2 ")  
  .config("spark.streaming.stopGracefullyOnShutdown", "true")  
  .config("spark.sql.shuffle.partitions", 3)  
  .getOrCreate()
```

```
val weatherSchema = StructType(List(  
  StructField("DateTime", TimestampType),  
  StructField("Temperature", DoubleType),  
  StructField("Humidity", FloatType),  
  StructField("WindSpeed", DoubleType),  
  StructField("Pressure", DoubleType),  
  StructField("Summary", StringType)))
```

```
//1. READ FROM THE STREAM  
val weatherSocketDF = spark.readStream  
  .format("socket")  
  .option("host", "localhost")  
  .option("port", "7817")
```



TRENDYTECH 9108179578

```
.load()

/**processing */
val valueDF = weatherSocketDF.select(from_json(col("value"), weatherSchema).alias("Value"))

val refinedweatherSocketDF = valueDF.select("Value.*")

//refinedweatherSocketDF.printSchema()

val windowAggWeatherSocketDF = refinedweatherSocketDF
  .withWatermark("DateTime", "30 minute") //<-----"WATERMARK" MENTION HERE
  .groupBy(window(col("DateTime"), "15 minute", "5 minute"))
  .agg(avg("Temperature")
    .alias("totalTemperature"))

val outputWeatherSocketDF = windowAggWeatherSocketDF.select("window.start", "window.end",
"totalTemperature")

outputWeatherSocketDF.printSchema()

/**write to the sink console */
val WeatherSocketQuery = outputWeatherSocketDF.writeStream
```

TRENDYTECH 9108179578


```
.format("console") // .format("csv")  
.outputMode("update") // .outputMode("append")  
.option("checkpointLocation", "checkpoint-Location3")  
.trigger(Trigger.ProcessingTime("15 seconds"))  
.start()
```

```
WeatherSocketQuery.awaitTermination()
```

```
}
```



TRENDYTECH 9108179578



5 Star Google Rated
Big Data Course

LEARN FROM THE EXPERT



9108179578

Call for more details