Let's start our Practical Session 2 :

Task 1

Our first streaming using rate source

Initialization SparkSession

In [ ]:
```
import findspark
findspark.init()
```

In [ ]:
```
import pyspark
from pyspark.sql import SparkSession
spark=SparkSession.builder.getOrCreate()
```

Let’s first import the required libraries.

- Pyspark functions
- Pyspark Types

In [ ]:
```
from pyspark.sql.functions import *
from pyspark.sql.types import *
```

In [ ]:
```

```

Create streaming DataFrame

Let’s create our first Spark Streaming DataFrame using rate source. Here we have specified the format as rate and specified

rowsPerSecond = 1 to generate 1 row for each micro-batch and load the data into initDF streaming DataFrame.

In [ ]:
```
first_df = (spark
  .readStream
  .format("rate")
  .option("rowsPerSecond", 1)
  .load()
  )
```

Check if DataFrame is streaming or Not.

In [ ]:
```
print("Streaming Status : " ,first_df.isStreaming)
```

```
Streaming DataFrame :  True
```

Transformation

Perform transformation on initDF to generate another column result by just adding 1 to column value :

In [ ]:
```
first_df = first_df.withColumn("result", col("value") + lit(1))
```

Output

Use append output mode to output only newly generated data and format as console to print the result on the console.

In [ ]:
```
first_df.writeStream.outputMode("append").option("truncate", False).format("console").start().awaitTerminat
```

Our Second streaming using rate source

Create Streaming DataFrame

Create Streaming DataFrame using socket source. Also, check if DataFrame isStreaming.

In [ ]:
```
host = "128.0.0.2"
port = "8080"
```

In [ ]:
```
wordCount.writeStream.outputMode("complete").option("truncate", False).format("console").start().awaitTermi
```

In [ ]:
```
second_df = (spark.readStream.format("socket").option("host", host).option("port", port).load())
```

In [ ]:
```
print("Streaming Status : " ,second_df.isStreaming)
```

```
Streaming DataFrame :  True
```

Transformation: Word Count

In [ ]:
```
wordCount = second_df.select(explode(split(col("value"), " ")).alias("words")).groupBy("words").count()
```

Here we count words from a stream of data coming from this socket. Also, we check the schema of our streaming DataFrame.

In [ ]:
```
wordCount.writeStream.outputMode("complete").option("truncate", False).format("console").start().awaitTermi
```

Input Sources — File

With file input source, our application will wait for available data in the specified directory. We will use some of the stock data available here. For example, Apple stock data present in this file: AAPL_2006–01–01_to_2018–01–01.csv. We will take the data for a few years like 2015, 2016, and 2017 and manually save it to a different file like AAPL_2015.csv, AAPL_2016.csvand AAPL_2017.csv respectively. Similarly, we will create the sample data for Google, Amazon, and Microsoft as well. We will keep all the CSV files locally under data/stocks folder. Also, create another folder data/stream which we will use to simulate the streaming data.

Schema Our data contains the fields Date,Open,High,Low,Close,Adj Close,Volume and we will extract Name from the filename using a custom function. Here we define the schema and write a custom function to extract the stock ticker symbol.

In [ ]:
```
from pyspark.sql.types import *
```

In [ ]:
```
schema = StructType([StructField('Date', StringType(), True),
                     StructField('Open', FloatType(), True),
                     StructField('High', FloatType(), True),
                     StructField('Low', FloatType(), True),
                     StructField('Close', FloatType(), True),
                     StructField('Volume', IntegerType(), True),
                     StructField('Name', StringType(), True)])
```

In [ ]:
```
def obtain_FileName():
        file_name = reverse(split(input_file_name(), "/")).getItem(0)
        split(file_name, "_").getItem(0)
```

In [ ]:
```

```

Create Streaming DataFrame

Create a streaming DataFrame to read csv data from a specified directory "data/stream" and apply the above schema . We also specify the param maxFilesPerTrigger = 2 , which means our application will process a maximum of 2 CSV files in each batch. At the end, we create another column called Name using the function getFileName. That column contains stock ticker symbols like GOOGL, AMZN, AAPL etc.

In [ ]:
```
third_df = spark.readStream.option("maxFilesPerTrigger", 2).option("header", True).schema(schema).csv("Data
```

Transformation

Group the data based on stock Name, Year and find the maximum value of the HIGH column.

In [ ]:
```
fourth_df = third_df.groupBy(col("Name"), year(col("Date")).alias("Year")).agg(max("High").alias("Max"))
```

Can you build same transformation by Sql?

In [ ]:
```
third_df.createOrReplaceTempView("stockView")
fourth_df = spark.sql("""select year(Date) as Year, Name, max(High) as Max from stockView group by Name, Ye
```

Output

Print the contents of streaming DatFrame to console using update mode.

Work with any dataset from data file

In [ ]:
```
fourth_df.writeStream.outputMode("update").option("truncate", False).option("numRows", 3).format("console")
```

Task 2

Some aggregations transformation

Select "Name","Date", "Open", "High", "Low" and group by Name and Date and get the avg and use WithColumn to get current_timestamp

In [ ]:
```
fifth_df=third_df.select("Name","Date", "Open", "High", "Low").groupBy('Name','Date').avg().withColumn('cur
```

Checkpoints

In [ ]:
```
fifth_df.writeStream.outputMode("complete").option("truncate", False).format("console").start().awaitTermin
```

Task 3

One-time micro-batch

With a once trigger, our query will execute a single micro-batch. It will process all available data and then stop the application. This trigger is useful when you would like to spin-up a cluster periodically, Execute our streaming application with the once trigger.

In [ ]:
```
output_df = third_df.groupBy("Name", year("Date").as("Year"))
          .agg(max("High").as("Max"))
```

Users can create GraphFrames from vertex and edge DataFrames.

- Vertex DataFrame: A vertex DataFrame should contain a special column named “id” which specifies unique IDs for each vertex in the graph.
- Edge DataFrame: An edge DataFrame should contain two special columns: “src” (source vertex ID of edge) and “dst” (destination vertex ID of edge).

id	name	age
a	Alice	34
b	Bob	36
c	Charlie	30
d	David	29
e	Esther	32
f	Fanny	36

src	dst	relationship
a	e	friend
f	b	follow
c	e	friend
a	b	friend
b	c	follow
c	b	follow
f	c	follow
e	f	follow
e	d	friend
d	a	friend

In [ ]:
```
1 vertex = spark.createDataFrame([("a", "Alice", 34),("b", "Bob", 36),("c", "Charlie", 30),("d", "David", 29)
```

In [ ]:
```
Edge = spark.createDataFrame([  ("a", "e", "friend"),
  ("f", "b", "follow"),
  ("c", "e", "friend"),
  ("a", "b", "friend"),
  ("b", "c", "follow"),
  ("c", "b", "follow"),
  ("f", "c", "follow"),
  ("e", "f", "follow"),
  ("e", "d", "friend"),
  ("d", "a", "friend")],
  ["src", "dst", "relationship"])
```

In [ ]:
```
from graphframes import *
```

Create a GraphFrame from vertex and edge DataFrames

In [ ]:
```
graph = GraphFrame(vertex,Edge)
```

Take a look at the DataFrames

Get vertices, edges, and check the number of edges of each vertex

In [ ]:
```
1 vertex.show()
```

```
+---+-------+---+
| id|   name|age|
+---+-------+---+
|  a|  Alice| 34|
|  b|    Bob| 36|
|  c|Charlie| 30|
|  d|  David| 29|
|  e| Esther| 32|
|  f|  Fanny| 36|
+---+-------+---+
```

In [ ]:
```
1 Edge.show()
```

```
+---+---+------------+
|src|dst|relationship|
+---+---+------------+
|  a|  e|      friend|
|  f|  b|      follow|
|  c|  e|      friend|
|  a|  b|      friend|
|  b|  c|      follow|
|  c|  b|      follow|
|  f|  c|      follow|
|  e|  f|      follow|
|  e|  d|      friend|
|  d|  a|      friend|
+---+---+------------+
```

In [ ]:
```
1 graph.vertices.count()
```

Out[20]: 6

In [ ]:
```
1 graph.edges.count()
```

Out[22]: 10

Create UDF Functions

convert your graph by mapping a function over the edges DataFrame that deletes the row if src ≥ dst return "Delete" else "Keep"

In [ ]:
```
1 Edge.filter('src>=dst').show()
```

```
+---+---+------------+
|src|dst|relationship|
+---+---+------------+
|  f|  b|      follow|
|  c|  b|      follow|
|  f|  c|      follow|
|  e|  d|      friend|
|  d|  a|      friend|
+---+---+------------+
```

Filtering and connected components

Check vertices when "age" greater than 30 and check edges "relationship" equal "friend"

In [ ]:
```
1 graph2 = graph.filterVertices("age > 30").filterEdges("relationship = 'friend'").dropIsolatedVertices()
```

GraphFrames requires you to set a directory where it can save checkpoints. Create such a folder in your working directory

In [ ]:
```
1 sc=spark.sparkContext
```

In [ ]:
```
sc.setCheckpointDir(dirName="graphframes")

output = graph.connectedComponents()
output.select("id", "name","age").orderBy("id").show()
```

```
+---+-------+---+
| id|   name|age|
+---+-------+---+
|  a|  Alice| 34|
|  b|    Bob| 36|
|  c|Charlie| 30|
|  d|  David| 29|
|  e| Esther| 32|
|  f|  Fanny| 36|
+---+-------+---+
```

Then, the connected components can easily be computed with the connectedComponents-function.

In [ ]:
```
output.select("id", "name","age").orderBy("id").show()
```

```
+---+-------+---+
| id|   name|age|
+---+-------+---+
|  a|  Alice| 34|
|  b|    Bob| 36|
|  c|Charlie| 30|
|  d|  David| 29|
|  e| Esther| 32|
|  f|  Fanny| 36|
+---+-------+---+
```

Motif finding

Search for pairs of vertices a,b connected by edge e and pairs of vertices b,c connected by edge e2. It will return a DataFrame of all such structures in the graph

In [ ]:
```
1 graph.find("(a)-[e]->(b); (b)-[e2]->(c)").show()
```

```
+---------------+-------------+---------------+-------------+----------------+
|              a|            e|              b|           e2|               c|
+---------------+-------------+---------------+-------------+----------------+
|[c, Charlie, 30]|[c, e, friend]| [e, Esther, 32]|[e, f, follow]|  [f, Fanny, 36]|
|  [a, Alice, 34]|[a, e, friend]| [e, Esther, 32]|[e, f, follow]|  [f, Fanny, 36]|
|  [f, Fanny, 36]|[f, c, follow]|[c, Charlie, 30]|[c, e, friend]| [e, Esther, 32]|
|    [b, Bob, 36]|[b, c, follow]|[c, Charlie, 30]|[c, e, friend]| [e, Esther, 32]|
|  [d, David, 29]|[d, a, friend]|  [a, Alice, 34]|[a, e, friend]| [e, Esther, 32]|
|[c, Charlie, 30]|[c, e, friend]| [e, Esther, 32]|[e, d, friend]|  [d, David, 29]|
|  [a, Alice, 34]|[a, e, friend]| [e, Esther, 32]|[e, d, friend]|  [d, David, 29]|
| [e, Esther, 32]|[e, f, follow]|  [f, Fanny, 36]|[f, c, follow]|[c, Charlie, 30]|
|  [f, Fanny, 36]|[f, b, follow]|    [b, Bob, 36]|[b, c, follow]|[c, Charlie, 30]|
|[c, Charlie, 30]|[c, b, follow]|    [b, Bob, 36]|[b, c, follow]|[c, Charlie, 30]|
|  [a, Alice, 34]|[a, b, friend]|    [b, Bob, 36]|[b, c, follow]|[c, Charlie, 30]|
| [e, Esther, 32]|[e, f, follow]|  [f, Fanny, 36]|[f, b, follow]|    [b, Bob, 36]|
|  [f, Fanny, 36]|[f, c, follow]|[c, Charlie, 30]|[c, b, follow]|    [b, Bob, 36]|
|    [b, Bob, 36]|[b, c, follow]|[c, Charlie, 30]|[c, b, follow]|    [b, Bob, 36]|
|  [d, David, 29]|[d, a, friend]|  [a, Alice, 34]|[a, b, friend]|    [b, Bob, 36]|
| [e, Esther, 32]|[e, d, friend]|  [d, David, 29]|[d, a, friend]|  [a, Alice, 34]|
+---------------+-------------+---------------+-------------+----------------+
```

In [ ]:
```
1
```