SPARK DATAFRAME TRANSFORMATIONS

Reader API:

```
Val readDF=spark.read
.format("csv")
.option("header","true")
.schema("mySchema")
.option("mode","PERMISSIVE/MALFOLRMED/FAIL FAST")
.option("path", "input_path")
.load()
```

Writer API:

```
Val writeDF=spark.write
.format("csv")
.coalesce(4)
.partitionBy("country")
.bucketBy("order_id")
.mode("append/overwrite")
.option("maxRecordsPerFile",1000)
.option("path","output_path")
.save()
```

Convert Rdd to DataSet:

```
Val readFile=spark.sc.textFile("file path")   //rdd
Case class mySchema("col1:Integer, col2:String, col3:LongInteger")
Import spark.implicits.*
Val mapRdd=readFile.split(" ").map(x=>x(0).toInt,x(1).toString,x(2).toLong )
```

Val
readDF=**spark.createDataFrame(mapRdd).toDF("col1,col2,col3").as[mySchema]**

Convert List to DF:
Val rddList=sc.**parallelize**(myList) // convert rdd from list
//check if rdd data is in proper format before it can be converted into data frame
Val readDF=spark.createDataFrame(rddList).toDF("col1,col2,col3")

Spark SQL:

Val sqlDF=readDF.**createOrReplaceTempView**("orders_table")
Val sparkDF=**spark.sql**(""" select c1,c2,c3 from orders_table
order by c2 desc """)

Refer to a column:
readDF.**col**("orders_id") readDF.select("col(col1), col(col2)")
readDF.select("**$**order_id")
readDF.select("`order_id")

Write SQL functions:
readDF.**selectExpr**("select order_id, col1*col2 as totalAmount, concat(s1, s2) as combineResult")

Spark UDF:

1. Reader API to load data and convert into data frame with schema
2. Define the function
   Def custCountry(custId:Int)
   {if (custId> 200 "India" else "USA"}

3. Register the spark UDF

Spark.udf.register("customerCountry", custCountry(_:Int)) // "customerCountry" is UDF name

4. Add the column into spark table
   readDF.**withColumn**("customer_country", expr("customerCountry("customerId")) // new_column_name, function(old_column_name)

Transformations:

1. Date column converted into unix timestamp
   readDF.withColumn("new_format",**unix_timestamp**(col("order_date")).cast(DateType)

2. Date column in format 'dd-mm-yyyy'
   .withColumn("col_1",date_format("order_date", 'dd-mm-yyyy'))

3. Create new column with increasing unique values
   readDF.withColum("new_col", **monotonically_increasing_id**)

4. To drop duplicate values from table
   .**dropDuplicates**("order_id","purchase_id")

5. To drop a column
   .**drop**("col_name")

6. To rename a column
   .**withColumnRenamed**("old_col","new_col")

7. To deal with Null values
   .withColumn("col_name", **coalesce**("col_name",-1)

Join Condition:
joinCondition=" df1.col("stu_id") === df2.col("student_school_id")

```
joinType="leftOuter"
Val joinDF=df1.join(df2, joinCondition, joinType)
```

Aggregate Functions:

1. **Group aggregates** like sum, avg, median

```
readDF.select(a.*)
.agg(sum("amount").as"totalAmount"
.agg(avg("col1).as"avgCount"
.sum(expr(c1/c2).as"divideTerm"
```

2. **Window aggregates**

```
myWindow=Window
.partitionBy("order_id")
.orderBy("col2")
.rowsBetween(Window.unboundedPreceding, Window.currentRow)

Spark.sql("""select col1,col2,col3,sum(col3).over(myWindow) """)
readDF.withColumn("newSum", expr(sum("col1").over(myWindow) )
```

SPARK STREAMING

Reader API :

```
spark=SparkSession.builder
.appName(“word_count”)
.master(“local[*]”)
.config(spark.sql.shuffle.partitions,2)
.config(spark.streaming.stopGracefullyOnShutdown,true)
.getOrCreate()


Val readDF=spark.readStream
.format(“json”)
.option(“header”,”true”)
.option(“path”)
.option(“maxFilesPerTrigger”,1)
.option(“cleanSource”,”delete”)
.option(“cleanArchiveDir”,”archiveFolder”)
.load()
```

Writer API:

```
Val writeDF=resultDF.writeStream
.format(“ ”)
.outputModel(“append/complete/update”)
.option(“checkpointLocation”,”loc1”)
.trigger(Trigger.ProcessingTime,”15 seconds”)
.option(“path”,””)
```

.start()

Transformations:
-->It depends on the scenario given

Find frequency of word count:
Val wordCount=(readDF.explode(split(value,' ') as words))
.groupBy("words")
.count()

Tumbling window:
mySchema=List(StructType
(
structField("order_id","LongType"),
structField("order_date","TimestampType")
)
Val dataFrame=spark.createDataFrame(readDF).toDF("col1, col2, col3").as[mySchema]
Val windowDF=dataFrame.select("o.*")
.groupBy(**window(col("order_date"), "15 minutes"**) //partitionBy column and window size
.agg(sum(amount))
.alias("sum function")

Sliding window:
Val windowDF=dataFrame.select("o.*")
.groupBy(**window(col("order_date"), "15 minutes", "5 minutes"**) //partitionBy column ,window size, sliding window
.agg(sum(amount))
.alias("sum function")

Late arriving records:
Val windowDF=dataFrame.select("o.*")
.withWatermark("order_date", "30 minutes")
.groupBy(window(col("order_date"), "15 minutes") //partitionBy column and window size
.agg(sum(amount))

.alias("sum function")

Join Condition:

joinCondition=" df1.col("stu_id) === df2.col("student_school_id")
joinType="leftOuter"
Val joinDF=df1.join(df2, joinCondition, joinType)