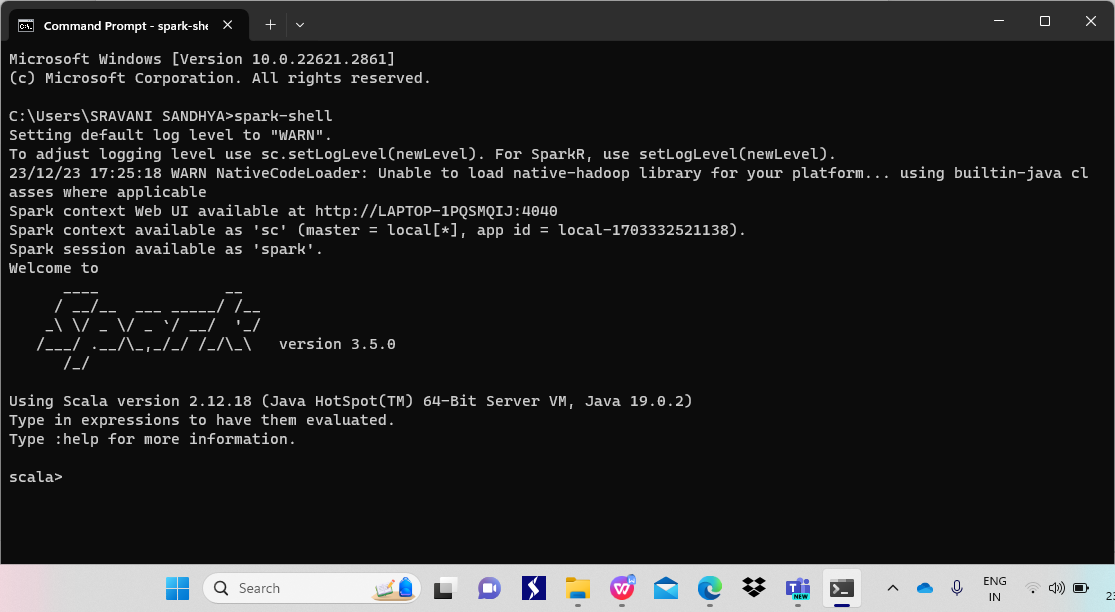
Mallu Sravana Sandhya Assessment-18 23/12/23

**Pyspark Commands**

Open command prompt and type spark-shell

It will starts the spark shell as shown in the below figure



Val data=sc.textFile(“fie\_name”)--to read the txt file from the local system

Sc.parallelize(var)--to parallelize the data stored in var

Var.count()--to count the items in RDD

Collect()--to gather all the RDD content at oneplace

Var.take(val)--to get specified number of values (val) from data stored in var

Counts.saveAsTextFile(“output”)--to save the recent output in text file

Val DFData=data.filter(line=>line.contains(“yes”))--it applies for existing RDD and generates new list of items

**Initializing Spark:**used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files.

from pyspark.sql import SparkSession

spark = SparkSession \

.builder \

.appName("Python Spark SQL basic example") \

.config("spark.some.config.option", "some-value") \

.getOrCreate()

**From RDD**

from pyspark.sql.types import \*

*Infer Schema*

**sc = spark.sparkContext**

It creates a spark context it can be created only one at a time if we want to create another we have to close the existing context

**lines = sc.textFile("people.txt")**

It reads the data from the people.txt file from local system

**parts = lines.map(lambda l: l.split(","))**

It transforms the data in the file to csv data

**people = parts.map(lambda p: Row(name=p[0],age=int(p[1])))**

It adds the data from parts to people in the form of rows

**peopledf = spark.createDataFrame(people)**

It create the dataframes for people data

*Specify Schema*

**people = parts.map(lambda p: Row(name=p[0],age=int(p[1].strip())))**

It adds the data from parts to people by splitting the rows of parts

**schemaString = "name age"**

Creates a string schema

fields = [StructField(field\_name, StringType(), True) for field\_name in schemaString.split()]

**schema = StructType(fields)**

Returns type of fields data

**Inspect Data**

**df.dtypes-**Return df column names and data types

**df.show()**-Return first n rows

**df.head()-**Display the content of df

**df.first()--** Return the first row

**df.take(2)**- Return the first n rows

**df.schema**-Return the schema of df

**df.describe().show()**-Compute summary statistics

**df.columns**-Return the columns of df

**df.count()**-Count the number of rows

**df.distinct().count()**-count the number of distinct rows

**df.printSchema()-**print schema of df

**df.explain()-**print logical and physical plans

**df = df.dropDuplicates()** -removes the duplicate data from df and the reamining to df

**SQL Queries in pyspark**

**from pyspark.sql import functions as F**

**Select df.select("firstName").show()**

It shows all the data present in firstname column **df.select("firstName","lastName").show()**

It shows all entries of firstname and lastname

**>>> df.select("firstName","age",explode("phoneNumber") .alias("contactInfo")) .select("contactInfo.type", "firstName","age") .show()**

It shows all entries of firstname and age and type

**>>> df.select(df["firstName"],df["age"]+ 1).show()**

It shows firstname and age increased by 1

**>>> df.select(df['age'] > 24).show()**

It shows all entries where age greater than 24

**>>> df.select("firstName",F.when(df.age >30,1) .otherwise(0)) .show()**

It shows firstname and 1 if age greater than 30 otherwise 0

**>>> df[df.firstName.isin("Jane","Boris")].collect()**

It shows all firstname if it is in given options

**Like**

**>>> df.select("firstName",df.lastName.like("Smith")) .show()**

Shows firstname whose last name is smith

**Startswith - Endswith**

**>>> df.select("firstName",df.lastName .startswith("Sm")) .show()**

It shows firstname whose last name starts with “sm”

**>>> df.select(df.lastName.endswith("th")).show()**

It shows all entries whose last name ends with “th”

**Substring**

**>>> df.select(df.firstName.substr(1, 3) .alias("name")).collect()**

It selects first 3 letters of firstname as substring under name column

**Between**

**>>> df.select(df.age.between(22, 24)) .show()**

It shows all entries whose age is between 22 and 24

**Adding Columns:**

we use below syntax to add new column to the table

Here we added city, postalcode,state,street address, number to the address table

df = df.withColumn('city',df.address.city)\

.withColumn('postalCode',df.address.postalCode) \

.withColumn('state',df.address.state) \

.withColumn('streetAddress',df.address.streetAddress) \

.withColumn('telePhoneNumber',explode(df.phoneNumber.number)) \

.withColumn('telePhoneType',explode(df.phoneNumber.type))

**Updating columns**

We use following syntax to update columns here we are updated telePhoneNumber column to phone Number

df = df.withColumnRenamed('telePhoneNumber', 'phoneNumber')

**Removing columns**

We have two different syntax to drop the column from table

Here we are removing column named phoneNumber from adddress

df = df.drop("address", "phoneNumber")

df = df.drop(df.address).drop(df.phoneNumber)

**Group By**

The below query shows the number of entries in the table group by age

df.groupBy("age").count().show()

**Filter**

It shows the records whose age is greater than 24

df.filter(df["age"]>24).show()

**Sort**

>>>peopledf.sort(peopledf.age.desc()).collect()

It sorts the age of people dataframes in descending order

>>> df.sort("age", ascending=False).collect()

>>> df.orderBy(["age","city"],ascending=[0,1]).collect()

It sorts the age in descending order and city in alphabetical order 0,1 in above syntax represents 0 for false and 1 for true

**Missing and Replacing**

>>df.na.fill(50).show()

It returns new df by relacing all null values with 50

>.df.na.drop().show()

It returns new dataframes by removing all the null values

>df.na.replace(10,20).show()

It returns an other df by replacing 10 with 20

DataStructures

>rdd1=df.rdd

It converts dataframes into resilent distrubuted dataset

>df.toJSON().first()

Converts df into a RDD of string

>df.toPandas()

Returns contents of df as pandas data structure

**Broadcast**

Inorder to join two tables in pyspark we use broadcast

from pyspark.sql.functions import broadcast cases=cases.join(broadcast(regions),[‘province’,’city’],how=’left’)

To limit the values to show the output we use limit() function and convert into pandas

Cases.limit(10).toPandas()

It returns only 10 rows

**Write and Save Files**

>df.select("firstName", "city").write.save("nameAndCity.parquet")

It writes the data into the file named nameAndCity.perquet and save into it

>df.select("firstName","age").write.save("namesAndAges.json",format="json")

It saves the file in json format

**Stoping Spark Session**

We use below command to stop the spark session

Spark.stop()