**SPARK WORKOUTS**

DataFrame API is created by reading File, running a query in top of tables.

DataFrame consists of row and columns similar to relational database management system.

# Create a dataframe for empdata.txt

df=spark.read.option("header",True).option("inferSchema",True).format("csv").option("delimiter",",").option("mode","DROPMALFORMED").load("file:/home/hduser/empdata.txt")

#Check the type

type(df)

# View the data in DataFrame

df.show()

# Create a DataFrame for custs file

custdf=spark.read.option("header",True).option("inferSchema",True).format("csv").load("file:/home/hduser/custs")

#To see the records use show() function. It prints by default 20 records

custdf.show()

#Set the truncate=False

custdf.show(100,False)

# To rename the column use withColumnRenamed

df2=custdf.withColumnRenamed("first\_name","fname").withColumnRenamed("last\_name","lname")

df2.show()

# To add a new column use withColumn

#import the needed functions

from pyspark.sql.functions import lit, col, udf, concat,desc

df3=df2.withColumn("TypeOfData",lit("CustInfo"))

df3.show()

df4=df3.withColumn("fullname",concat("fname",lit(" "),"lname")).drop("fname","lname")

#To view the spark schema use printSchema

df4.printSchema

#View the DataFrame data ordering by age in descending order

df4.orderBy(desc("age")).show(100)

#DSL (Domain Specific Language) – To see the selected columns

df5=df4.select("custid","fullname","age","profession","TypeOfData")

df5.show()

---------------------------------------------------------------------------------------------------------

df5.select("fullname","profession","age").where("age>70").show()

df5.select("\*").where("profession='Pilot' and age>70").show()

df5.sort(desc("age")).show()

#Cache and persist is used to store the DataFrame in memory which needs to be used by more than once. It avoid recreating the dataframe again from the behinning. It is one of the optimization technique in Spark

df5.cache()

df5.unpersist()

df5.persist(pyspark.StorageLevel.MEMORY\_AND\_DISK)

df5.unpersist()

/--------------

DISK\_ONLY = StorageLevel(True, False, False, False, 1)

DISK\_ONLY\_2 = StorageLevel(True, False, False, False, 2)

MEMORY\_AND\_DISK = StorageLevel(True, True, False, False, 1)

MEMORY\_AND\_DISK\_2 = StorageLevel(True, True, False, False, 2)

MEMORY\_AND\_DISK\_SER = StorageLevel(True, True, False, False, 1)

MEMORY\_AND\_DISK\_SER\_2 = StorageLevel(True, True, False, False, 2)

MEMORY\_ONLY = StorageLevel(False, True, False, False, 1)

MEMORY\_ONLY\_2 = StorageLevel(False, True, False, False, 2)

MEMORY\_ONLY\_SER = StorageLevel(False, True, False, False, 1)

MEMORY\_ONLY\_SER\_2 = StorageLevel(False, True, False, False, 2)

-------------------/

df5.count()

--- remove the record with null in any of the column

df6=df5.na.drop();

df6.count()

df6\_cid=df5.na.drop("any")

df6\_cid.count()

df6.count()

df6\_all=df5.na.drop("all")

df6\_all.count()

df\_cid=df5.na.drop(subset=["profession"])

df\_cid.count()

df5.select("\*").where("profession IS NULL").show(1000)

df5\_unique=df5.distinct()

df6=df5.na.fill("BIGDATA\_ENGG", "Profession")

df6.select("\*").where("profession=='BIGDATA\_ENGG'").show()

===================================-------------------------==============================

#Replace Function

df6.na.replace(["BIGDATA\_ENGG","Artist"],["DataEngineer","SUPER STAR"],"profession").where("profession IN ('DataEngineer','SUPER STAR')").show(10000)

#Spark windowing function

from pyspark.sql.window import Window

from pyspark.sql.functions import row\_number, rank, dense\_rank

win\_spec=Window.partitionBy("profession").orderBy("age")

df6.withColumn("row\_num", row\_number().over(win\_spec)).show(truncate=False)

df6.withColumn("row\_num", row\_number().over(win\_spec)).where("row\_num == 1").show(100,truncate=False)

win\_spec=Window.partitionBy("profession").orderBy(desc("age"))

df6.withColumn("row\_num", row\_number().over(win\_spec)).where("row\_num == 1").show(100,truncate=False)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

from pyspark.sql.functions import row\_number, rank, dense\_rank, min, max

df6.agg(max("age"), min("age")).show()

df6.write.csv("file:/home/hduser/02022023\_output")

df6.write.format("csv").save("file:/home/hduser/02022023\_output1")

#Spark Write operations

df7=df6.repartition(10)

df7.write.format("csv").save("file:/home/hduser/02022023\_outputs")

df7.coalesce(1).write.format("csv").save("file:/home/hduser/02022023\_outputs\_1")

spark.sql(""" SELECT \* FROM customer WHERE profession IN ('DataEngineer','SUPER STAR','Pilot') and age between 20 and 30""").show()

spark.sql(""" SELECT custid, fullname, age, profession, dense\_rank() over (partition by profession order by age desc) as rnk FROM customer""").show()

spark.sql(""" SELECT custid, fullname, age, profession,rnk FROM (

SELECT custid, fullname, age, profession, dense\_rank() over (partition by profession order by age desc) as rnk FROM customer) tbl

WHERE rnk<=5""").show(10000)

spark.sql("""SELECT profession, count(1) FROM customer GROUP BY 1 ORDER BY 2 DESC """).show(100)

df7.coalesce(1).write.format("parquet").save("file:/home/hduser/02022023\_output\_parquets")

#Create a DataFrame on top of ebayjson file

ebay\_df=spark.read.format("json").load("file:/home/hduser/sparkdata/ebayjson.json")

ebay\_df.count()

ebay\_df.show()

df1=ebay\_df.select("\*").where("bidder is not null")

ebay\_df.count()

df1.count()

df1.show()

ebay\_df1=spark.read.format("json").option("mode","PERMISSIVE").load("file:/home/hduser/sparkdata/ebayjson.json")

ebay\_df1=spark.read.format("json").option("mode","FAILFAST").load("file:/home/hduser/sparkdata/ebayjson.json")

ebay\_df1=spark.read.format("json").option("mode","DROPMALFORMED").load("file:/home/hduser/sparkdata/ebayjson.json")

df1.show()

from pyspark.sql.functions import col,dense\_rank,rank,row\_number

from pyspark.sql.window import Window

window\_spec=Window.partitionBy("bidder").orderBy(col('price').desc())

ebay\_df1.withColumn("rnk",dense\_rank().over(window\_spec)).show()

ebay\_df1.groupBy("bidder").max("price").show()

ebay\_df1.createOrReplaceTempView("ebay\_data")

ebay\_df2=spark.sql("SELECT bidder, sum(price) as tot\_price FROM ebay\_data group by 1")

ebay\_df2.createOrReplaceTempView("ebay\_tbl")

# Create a Schema

from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DoubleType

schema=StructType([

StructField("auctionid",IntegerType(),False),

StructField("bid",DoubleType(),False),

StructField("bidtime",DoubleType(),False),

StructField("bidder",StringType(),False),

StructField("bidderrate",IntegerType(),False),

StructField("openbid",IntegerType(),False),

StructField("price",DoubleType(),False),

StructField("item",StringType(),False),

StructField("daystolive",IntegerType(),False)

])

df=spark.read.format("csv").option("delimiter","~").schema(schema).load("file:/home/hduser/sparkdata/auctiondata")

df.show()

#Spark UDF (User Defined Funtion):

UDF:

def category(bid\_value):

if bid\_value >= 100:

return "High Value"

return "Less Value"

spark.udf.register("myfunc",category)

df.createOrReplaceTempView("table1")

spark.sql("SELECT \* from table1").show(2)

spark.sql("""SELECT bidder,bid,myfunc(bid) as category FROM table1""").show()

#Spark JOINS

JOINS:

broadcast join

sort merge join

shuffle hash join

spark.conf.get("spark.sql.autoBroadcastJoinThreshold")

spark.sql("SELECT \* FROM ebay\_data t1 JOIN ebay\_data t2 ON t1.auctionid=t2.auctionid").show(

spark.sql("SELECT \* FROM ebay\_data t1 JOIN ebay\_data t2 ON t1.auctionid=t2.auctionid").explain()

spark.conf.set("spark.sql.autoBroadcastJoinThreshold",20971520)

spark.conf.get("spark.sql.autoBroadcastJoinThreshold")

spark.conf.set("spark.sql.autoBroadcastJoinThreshold",-1)

spark.sql("SELECT \* FROM ebay\_data t1 JOIN ebay\_data t2 ON t1.auctionid=t2.auctionid").explain()

spark.conf.get("spark.sql.join.preferSortMergeJoin")

spark.conf.set("spark.sql.join.preferSortMergeJoin","false")

================================================

**SPARK OPTIMIZATION:**

1. Use KyroSerializer

2. Choose the right API

(RDD, DF)

3.Cache, Persist

4.Broadcost and Accumulator Variable

city=["Chennai","Hyderabad","Banglore","Mumbai","Pune","Delhi"]

city=spark.sparkContext.broadcast(["Chennai","Hyderabad","Banglore","Mumbai","Pune","Delhi"])

user\_count=spark.sparkContext.accumulator(0)

5. Join ( \*\*\*\*\*broadcast join, soft-merge and shuffle hash join)

6.user coalesce rather using repartition to descrease the partition count.

7. spark.sql.shuffule.partition

spark.conf.get("spark.sql.shuffle.partitions")

spark.conf.set("spark.sql.shuffle.partitions",1000)

spark.conf.get("spark.sql.shuffle.partitions")

8.Predicate Pushdown and projective pushdown

SELECT custid, amount FROM (

SELECT \* FROM TABLE )

WHERE city='Chennai'

SELECT custid, amount FROM (

SELECT \* FROM TABLE WHERE city='Chennai')

SELECT custid, amount FROM (

SELECT custid,amount FROM TABLE WHERE city='Chennai')

SELECT count(\*) FROM TABLE

SELECT count(1) FROM TABLE