**PySpark Basics** Cheat Sheet

**groupBy**

Group by age, count the members in the groups

> empDF.groupBy("age")\

.count().show()

**Duplicate Values Queries**

Show all entries in emp\_name and age, add 1 to the entries of age

Show all entries where age >28

Show emp\_name and 0 or 1 depending on age >32

Show emp\_name if in the given options Show emp\_nane, and emp\_id is TRUE if emp\_name is like Sam

Show emp\_id, and TRUE if

emp\_name starts with Sm

Show last names ending in hn

Return substrings of emp\_name

Show age: values are TRUE if between

32 and 40

Show all entries in emp\_name column

Show all entries in emp\_name, age

and type

> from pyspark.sql import functions as F

**Select**

> empDF.select("emp\_name").show()

> empDF.select("emp\_name","age") \

.show()

> empDF.select("emp\_name",

"age",explode("phoneNumber") \

.alias("contactInfo")) \

.select("contactInfo.type", "emp\_name",

"age").show()

>empDF.select(empDF["emp\_name"],df["age"]+)

.show()

>empDF.select(empDF ['age'] >24).show()

**When**

> empDF.select("emp\_name",

F.when(df.age > 32, 1) \

.otherwise(0)) \.show()

>empDF[empDF.emp\_name.isin("Sam","John")]

.collect()

**Like**

>df.select("emp\_id",df.emp\_name.like("Sam") \ .show()

**Startswith - Endswith**

> df.select("emp\_name",df. emp\_id \

.startswith("Sm")).show()

> df.select(df.emp\_name.endswith("hn")) \

.show()

**Substring**

> df.select(df.emp\_name.substr(1, 3) \

.alias("name")) \

.collect()

**Between**

> df.select(df.age.between(32, 40)) \

.show()

> empDF = empDF.dropDuplicates()

for Data Engineering



**Filter**

Filter entries of age, only keep those records of which the values are >28

> empDF.filter(df["age"]>28).show()



**PySpark**

PySpark is a Python API for Apache Spark.

**Sort**

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

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

> empDF.orderBy(["age","city"],ascending=[0,1])\

.collect()

**Initializing SparkSession**

A SparkSession can be used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read different format files.

> from pyspark.sql import SparkSession

> spark = SparkSession \

.builder \

.appName('PySpark Practice') \

.master('local[2]') \

.getOrCreate()

**Fill & Fillna**

> df.na.fill(value=0).show()

> df.na.fill(value=0,subset=["city"]).show()

> df.fillna("unknown",["city"]) \   
 .fillna("",["age"]).show()

**Creating DataFrames**

**Creating DataFrame**

> empData=[(101,'Sam', 32),

(201,'John', 40),

(301,'David', 28)

]

> empSchema= ['emp\_id', 'emp\_name', 'age']

> empDF= spark.createDataFrame(data=empData, schema= empSchema)

**Creating DataFrame from Custom Schema**

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

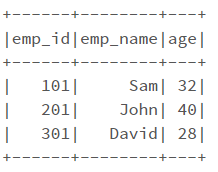
> empSchema= StructType([StructField('emp\_id',IntegerType(), True),

StructField('emp\_name', StringType(), True),

StructField('age', IntegerType(), True)

])

> empDF.show(truncate= False)



**Creating DataFrame by Reading Files**

**InferSchema option**

> empDF= spark.read.format('csv').options(header= 'True', InferSchema= 'True', sep= ',').load('./emp.csv')

**Different File Formats: csv, text, json, parquet, avro, orc can be read with different options**

**Custom Schema option**

> empDF= spark.read.format('csv').options(header= 'True', sep= ',').schema(empSchema).load('./emp.csv')

**Mode() option**

**Different mode options while reading file are**

**‘PERMISSIVE’ – read all records from file,**

**‘DROPMALFORMED’ - delete bad records & don’t read them**

**‘FAILFAST’ – raise error(SparkException) if there are bad records in file.**

**by default mode =’** **PERMISSIVE’**

> empDF= spark.read.format("csv").options(mode= "PERMISSIVE", header= "true", sep= ',').schema(empSchema).load("./emp.csv")

**Inspect Data**

> df.show() Display first 20 rows n truncate

column value to 20 characters of DF

> df.head() Return first ‘n’ row

> df.first() Return first row

> df.take(2) Return the first ‘n’ rows

> df.schema Return the schema of DF

> df.dtype Return DF column names and data types

> df.columns Returns column names as list

**Date timestamp**

**date\_format() – convert date from one format to another**

> df.select(current\_date().alias("current\_date"), \

date\_format(current\_timestamp(),"yyyy MM dd hh:mm:ss")).alias("yyyy MM dd").show()

**unix\_timestamp() – convert Timestamp into Unix Epoch Time**

> df.select(date\_format(current\_timestamp(),"yyyy MM dd hh:mm:ss")) \

.withColumn('unix\_epoch\_time', unix\_timestamp('current\_date')).show()

eg – Timestamp : 2022 08 21 09:11:48

into Unix Epoch Time : 1661040000

**from\_unixtime()- convert Unix Epoch Time into Timestamp**

**Add, Update & Remove Columns**

**To add new column to DataFrame**

> empDF = empDF.withColumn('city',lit('Mumbai'))

**To rename column name of DataFrame**

> empDF = empDF.withColumnRenamed('age', 'emp\_age')

**To drop column of DataFrame**

> empDF= empDF.drop('city')

**JOINS**

> joinDF= empDF.join(deptDF, 'dept\_id', 'FULLOUTER')

**Different Types of Joins are INNER, FULLOUTER, RIGHTOUTER, LEFTOUTER.**

**Removing Columns**

**Write Files from DataFrame & Save**

>empDF.write.format('parquet').mode('overwrite').save(path= './Output/Employee.parquet')

>empDF.write.mode('overwrite').csv(path='./Output/Employee.txt', header='True', sep= '\t')

**Mode options are append, overwrite, error, ignore.**

**Files can be write into different formats such as csv, text, json, parquet.**

**REGEX\_REPLACE():** Replace one value with other value in column

from pyspark.sql.functions import regexp\_replace

> deptDF.withColumn('dept\_name', regexp\_replace('dept\_name', 'HR', 'Human Resource'))

**TRIM()** : trim space from left/right/ both

from pyspark.sql.functions import ltrim, rtrim, trim

> deptDF.withColumn('dept\_name', ltrim('dept\_name'))



*Created by Monali Sahare*

*and Suraj Kumar*

*~ Diggibyte Technologies Pvt. Ltd.~*

*www.diggibyte.com*