RDD and DataFrame Operations

Create DataFrame:

val emp = spark.read.format("csv").option("sep", ",").option("inferSchema", "true").option("header", "true").load("/home/nineleaps/spark/emp.txt")

val dept = spark.read.format("csv").option("sep", ",").option("inferSchema", "true").option("header", "true").load("/home/nineleaps/spark/dept.txt")

val sgrade = spark.read.format("csv").option("sep", ",").option("inferSchema", "true").option("header", "true").load("/home/nineleaps/spark/salsgrade.txt")

**Dataframe Operations:**

**1. Creating New column, Renaming New column, Changing DataType of column, selecting subset of column**

**1.1) Creating New column**

emp.withColumn("Ecopy", $"EMPNO").show()

**1.2) Renaming new column**

emp.withColumnRenamed("EMPNO", "Ecopy2").show()

**1.3) Changing DataType of the column**

emp.withColumn("MGR", $"MGR".cast("double")).printSchema()

**Functional way:**

import org.apache.spark.sql.Column

import org.apache.spark.sql.functions.udf

def toDouble(df: Column): Column = {

df.cast("double")

}

emp.withColumn("dd", toDouble(emp("MGR"))).show()

**1.4)** **selecting subset of column**

**Use of when:**

val emp1 = emp.withColumn("COMM", when($"COMM" === "NULL", 0).otherwise($"COMM"))

emp1.groupBy("DEPTNO").agg(sum(when($"COMM".isNull, $"SAL").otherwise($"SAL"+$"COMM"))).show()

**3. Counting null values, handling and filling null values**

**3.1 Counting null values:**

**pushing null values where value is “NULL”:**

val emp2 = emp.withColumn("COMM", when($"COMM" === "NULL", null).otherwise($"COMM"))

emp2.filter("COMM is null").count() //It will return integer value

**3.2 Filling null values:**

emp2.na.fill("0", Seq("COMM")).show()

**3.3 Droping null values:**

emp2.na.drop().show()

**4. Filter operation ( and , or) on single/multiple columns**

emp.filter( ($"SAL" === 950.0) || ($"JOB" === "MANAGER") ).show() //or

emp.filter( ($"SAL" === 950.0) && ($"JOB" === "MANAGER") ).show() //and

**5. Ranking on dataframe columns ( Window Functions)**

# 103. Find out least 5 earners of the company based on the department

emp.withColumn("Rank", dense\_rank.over(Window.partitionBy($"DEPTNO").orderBy($"SAL".desc))).show()

**6. Writing UDF's with**

Better use case eg will be attached to the repo

7. **Other Fuctionality like : lit(), head()**

emp.withColumn("$SAL", concat(**lit**("$"), $"SAL")).show()

val maxSal = emp.agg(max($"SAL")).**head**().getDouble(0)

emp.filter($"SAL" === maxSal).show()

**8. How to print Schema, show, truncate, save**

emp.printSchema()

emp.show()

emp.write.format("csv").save("/home/nineleaps/just")

9. **Joins and where to use them**

**Three Major types of join we are using, inner, left, right, self**

**Self:**

emp.as("e").join(emp.as("m"), $"e.MGR" === $"m.EMPNO").select("m.\*").distinct().show()

**Without Primary key:**

emp.as("e").join(sgrade.as("s"), $"e.SAL".between($"s.losal", $"s.hisal"), "inner").show

**Multiple Joins with filter:**

emp.as("e").join(sgrade.as("s"), $"e.SAL".between($"s.losal", $"s.hisal"), "inner").join(dept.as("d"), $"d.DEPTNO" === $"e.DEPTNO", "inner").filter($"DNAME" === "SALES").show()

10. **Dataframe reparation(), paritionby(), coalesce() functions**

**Repartition**(): Repartition will help us to increase the number of partition with full shuffle.

**Coalesce**(): coalesce with help us to decrease the number of partition with partial shuffle. It is always good to go with coalesce for reduce the no.of partitions

**partitionBy**(): it will take all the number 4 to starting indexes

**11. collect() ( Spark 1.6 and spark 2). What is disadvantage of collect())**

Collect (Action) - Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.

Collect will return all the values from the RDD so, incase our rdd is too big we will give the overload to the network. So we can use collect after filtering and our dataset is low in size.

So always prefer, take() and show().