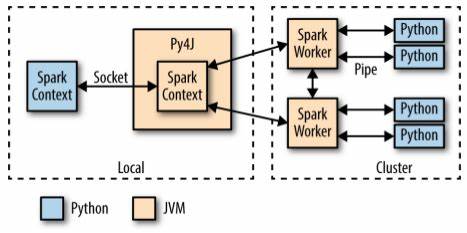
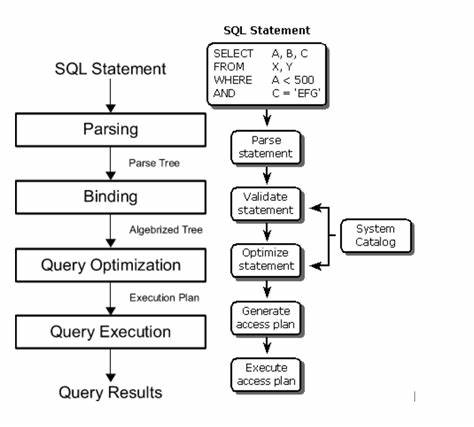
PySpark With & Without Sql

PYSPARK ARCHITECTURE



SQL ENGINE



|  |  |
| --- | --- |
| spark.Sql | Without spark.sql |
| ASsuming table is formed using df.createOrReplaceTempView[“table”] | ASsuming import pyspark.sql.functions AS f |

Selection

* SQL spark.sql(SELECT col\_1 FROM table”)
* Pyspark df.select(f.col(“col\_1”))

Alias

* SQL spark.sql(SELECT col\_1 AS “new” FROM table”)
* Pyspark df.select(f.col(“col\_1”).alias(“new”)

Describe

* SQL spark.sql(“DESCRIBE table”)
* Pyspark df.printSchema( )

Select Distinct

* SQL spark.sql(“SELECT distinct col\_1 FROM table”)
* Pyspark df.select(f.col(“col\_1”)).distinct( )

Limit n

* SQL spark.sql(“SELECT col\_1 FROM table limit n”)
* Pyspark df.select(f.col(“col\_1”)).limit( n )

Ascending Sort

* SQL spark.sql(“SELECT col\_1 FROM table order by col\_1”)
* Pyspark df.orderBy(“col\_1”).

Descending Sort

* SQL spark.sql(“SELECT col\_1 FROM table order by col\_1 desc”)
* Pyspark df.orderBy(“col\_1”,AScending=false)

Filter

* SQL spark.sql(“SELECT col\_1 FROM table WHERE col\_1>5”)
* Pyspark df.filter(f.col(“col\_1”)>f.lit( 5 ))

New Column with constant

* SQL spark.sql(“SELECT col\_1,1 AS “new” FROM table”)
* Pyspark df.withColumn(“new, f.lit(1))

Group By with Aggregation

* SQL

spark.sql(“SELECT col\_1,sum(col\_1),count(col\_1) FROM table group by col\_1”)

* Pyspark

df.groupBy(“col\_1).agg(f.sum(col\_1),f.count(col\_1))

Union

* SQL

spark.sql(“SELECT col\_1 FROM left\_table UNION SELECT col\_1 FROM right\_table”)

* Pyspark

left\_df.select(f.col(“col\_1”)).union(right\_df.select(f.col(“col\_1))).distinct( ).

Union All

* SQL

spark.sql(“SELECT col\_1 FROM left\_table UNION ALL SELECT col\_1 FROM right\_table”)

* Pyspark

left\_df.select(f.col(“col\_1”)).union(right\_df.select(f.col(“col\_1)))

Inner Join

* SQL

spark.sql(“SELECT \* FROM left\_table l join right\_table r ON l.col\_1=r.col\_2”)

* Pyspark

left\_df.join(right\_df,left\_df.left1==right\_df.right1”).show( )

Outer Join

* SQL

spark.sql(“SELECT \* FROM left\_table FULL OUTER JOIN right\_table ON left\_table.left1=right\_table.right1”)

* Pyspark

left\_df.join(right\_df,left\_df.left1==right\_df.right1,how=”outer”).show( )

Left Join

* SQL

spark.sql(“SELECT \* FROM left\_table LEFT JOIN right\_table ON left\_table.left1=right\_table.right1”)

* Pyspark

left\_df.join(right\_df,left\_df.left1==right\_df.right1,how=”left”).show( )

Cross Join

* SQL

spark.sql(“SELECT \* FROM left\_table CROSS JOIN right\_table ON left\_table.left1=right\_table.right1”)

* Pyspark

left\_df.crossJoin(right\_df).show( )

Case When

* SQL

spark.sql(“SELECT \*, CASE WHEN col\_1 >5 THEN “Yes” ELSE “No” end FROM table”)

* Pyspark

df.withColumn(“col\_4”,f.when(df.col\_1>5 ,”Yes”).otherwise(“No”))\.

show( )

Power

* SQL

spark.sql(“SELECT POWER(f.col(“col\_1”),5) AS newMeASure

FROM table”)

* Pyspark

Df.selectExpr(f.col(“col1”),Power(f.col(“col1”,5))).show( )

Round & Bround

* SQL

spark.sql(“SELECT ROUND( 2.5 ), BROUND( 2.5 ) “)

* Pyspark

df.select(f.round(lit(“2.5”),f.bround( lit( 2.5 ) ).show( )

Regular Expression

* SQL

spark.sql(“SELECT regexp\_replace( Description,'BLACK|WHITE|RED|GREEN|BLUE', 'COLOR') AS color\_clean, Description FROM table”)

* Pyspark

regex\_string = "BLACK|WHITE|RED|GREEN|BLUE"

df.select( regexp\_replace(col("Description"), regex\_string, "COLOR").aliAS("color\_clean"),f.col("Description")).show(2)

Drop Null

* SQL

spark.sql(“SELECT \* FROM table WHERE col\_1 is null”)

* Pyspark

df.na.drop("all", subset=["col\_1", "col\_2"])

Fill Null

* SQL

spark.sql(“SELECT coalesce(col\_1,value,null ) AS new FROM table”)

* Pyspark

df.na.fill("all", subset=["col\_1", "col\_2"])

Replace

* SQL

spark.sql(“SELECT REPLACE(col\_1,’value’,’new value’) FROM table”)

* Pyspark

df.na.replace([""], ["UNKNOWN"], "col\_1")

Split

* SQL

spark.sql(“SELECT col\_1, col\_2, exploded FROM (SELECT \*, split(col\_1, " ") AS splitted FROM table) LATERAL VIEW explode(splitted) AS exploded”)

* Pyspark

df.select(split(col("DescriptiON"), " ")).show(2)

Explode

* SQL

spark.sql(“SELECT split( col\_1, ’,’ ) FROM table”)

* Pyspark

df.withColumn("splitted", split(col("col\_1"), " "))\.

withColumn("exploded", explode(col("splitted")))\ .

select("col\_1", "col\_2", "exploded").show(2)

Map

* SQL

spark.sql(“SELECT map(col\_1, col\_2) AS complex\_map FROM table WHERE col\_1 IS NOT NULL”)

* Pyspark

df.select(create\_map(col("col\_1"),col("col\_2")).aliAS("complex\_map")) .show(2)

Group Concat

* SQL

spark.sql(“SELECT col\_1,GROUP\_CONCAT ( col\_2,’,’ ) FROM table GROUPBY col\_1”)

* Pyspark

df.agg(f.collect\_set("col\_1"), f.collect\_list("Country")).show( )

Date Format

* SQL

spark.sql(“SELECT col\_1,to\_char (date\_1,’mm-dd-yyy’ ) AS new FROM table”)

* Pyspark

df.select(f.col("col\_1"),date\_format(f.col("col\_1"),

“MM-dd-yyyy”).aliAS(“new”).show(2)

Row Number

* SQL

spark.sql(“SELECT \*,row\_number( ) over(partition by col\_1 order by col\_2)AS row\_number FROM table”)

* Pyspark

windowSpec=f.Window.partitionBy(“col\_1).orderBy(“col\_2”)

df.withColumn(“row\_number,row\_number(),windowSpec)\.

show( truncate=False )

Rank

* SQL

spark.sql(“SELECT \*,rank( ) over(partition by col\_1 order by col\_2)AS rank FROM table”)

* Pyspark

windowSpec=f.Window.partitionBy(“col\_1).orderBy(“col\_2”)

df.withColumn(“ rank ”,rank( ).over( windowSpec )).show( )

Dense Rank

* SQL

spark.sql(“SELECT \*,dense\_rank( ) over(partition by col\_1 order by col\_2)AS dense\_rank FROM table”)

* Pyspark

windowSpec=f.Window.partitionBy(“col\_1).orderBy(“col\_2”)

df.withColumn(“dense\_rank,dense\_rank(),windowSpec)\.

show( truncate=False )

Lag

* SQL

spark.sql(“SELECT \*,lag(col\_2) over(partition by col\_1 order by col\_2)AS previous FROM table”)

* Pyspark

windowSpec=f.Window.partitionBy(“col\_1).orderBy(“col\_2”)

df.withColumn("previous",lag("col\_2",2).over(windowSpec)).show( )

Lead

* SQL

spark.sql(“SELECT \*,lead (col\_2 ) over(partition by col\_1 order by col\_2)AS next FROM table”)

* Pyspark

windowSpec=f.Window.partitionBy(“col\_1).orderBy(“col\_2”)

df.withColumn("next",lead("salary",2).over(windowSpec)) .show( )

Percentile Rank

* SQL

spark.sql(“SELECT \*,percent\_rank() over(partition by col\_1 order by col\_2) AS percent\_rank FROM table”)

* Pyspark

windowSpec=f.Window.partitionBy(“col\_1).orderBy(“col\_2”)

df.withColumn(“percent\_rank”,percent\_rank (),windowSpec)\.

show( truncate=False )

Window Aggregate

* SQL

spark.sql(“

With cte AS ( SELECT \*,row\_number( ) over(partition by col\_1)AS row\_number FROM table )

SELECT \*,AVG( col\_2) over(partition by col\_1),SUM(col\_2 ) over(partition by col\_1), MAX (col\_2 ) over(partition by col\_1), MIN ( col\_2) over(partition by col\_1) FROM cte WHERE r=1 ) ”)

* Pyspark

windowSpecAgg = Window.partitionBy("department")

df.withColumn("row",row\_number().over(windowSpec)) \

.withColumn("avg", avg(col("salary")).over(windowSpecAgg)) \

.withColumn("sum", sum(col("salary")).over(windowSpecAgg)) \

.withColumn("min", min(col("salary")).over(windowSpecAgg)) \

.withColumn("max", max(col("salary")).over(windowSpecAgg)) \

.WHERE(col("row")==1).select("department","avg","sum","min","max") \

.show( )

E -- N -- D