SparkSQL Cheatsheet

Table of Contents

[Array Queries 3](#_Toc21586574)

[Return the size of an array 3](#_Toc21586575)

[Case Statements 3](#_Toc21586576)

[Change a pyspark.sql.dataframe.DataFrame Data Type (CAST) 3](#_Toc21586577)

[Constants 3](#_Toc21586578)

[pi() 3](#_Toc21586579)

[DataFrame (pyspark.sql.dataframe.DataFrame) 3](#_Toc21586580)

[Change a DataFrame’s Column Names 3](#_Toc21586581)

[Create a DataFrame from Scratch 5](#_Toc21586582)

[Execute SQL on a DataFrame 5](#_Toc21586583)

[Explode a List inside a DataFrame column 6](#_Toc21586584)

[Select Distinct Rows of a pySpark DataFrame 6](#_Toc21586585)

[Distance on Earth Surface Using Haversine Equation 7](#_Toc21586586)

[Invoke a pySpark UDF from SQL 7](#_Toc21586587)

[Joins 8](#_Toc21586588)

[Left Anti Join 8](#_Toc21586589)

[Left Outer Join 8](#_Toc21586590)

[LEAD and LAG 10](#_Toc21586591)

[Loading a Schema while Correcting org.apache.parquet.io.ParquetDecodingException 11](#_Toc21586592)

[Read a Parquet File into a DataFrame 11](#_Toc21586593)

[Rounding Numerical Values 11](#_Toc21586594)

[Store DataFrame as Temp Table for Later Querying 12](#_Toc21586595)

[Select an Array Element 12](#_Toc21586596)

[Parameterized Queries 12](#_Toc21586597)

[Passing a List or Tuple in a Parameterized Query 12](#_Toc21586598)

[Parquet 13](#_Toc21586599)

[Explode a list or array data field 13](#_Toc21586600)

[Statistics 15](#_Toc21586601)

[Average 15](#_Toc21586602)

[Median 15](#_Toc21586603)

[Strings 15](#_Toc21586604)

[Concatenate strings 16](#_Toc21586605)

[Convert a numeric field to a string 16](#_Toc21586606)

[Convert a String to UPPER or lower case 17](#_Toc21586607)

[Match Part of a String, Using Wildcards 17](#_Toc21586608)

[Substring 17](#_Toc21586609)

[Time Intervals 19](#_Toc21586610)

[Timestamps 20](#_Toc21586611)

[Convert a timestamp in UTC to a timestamp in a Local Timezone 20](#_Toc21586612)

[Convert string in a local timezone to a UTC timestamp 20](#_Toc21586613)

[Convert string to timestamp 20](#_Toc21586614)

[Convert ISO 8601 Standard DateTime to unix Timestamp 21](#_Toc21586615)

[Convert Unix Timestamp to Formatted Date 21](#_Toc21586616)

[Convert Unix Epoch to a pySpark Timestamp 22](#_Toc21586617)

[Convert Unix Epoch Timestamp to ISO Standard Date and Time String 22](#_Toc21586618)

[Extract a Date Part from a Unix Epoch 23](#_Toc21586619)

[Windowing Functions (like Group By ... Having) 24](#_Toc21586620)

# Array Queries

## Return the size of an array

SIZE(smfPayloadData.geolocation.detectedPois)

# Case Statements

shipment\_modes\_df = sqlContext.sql(''' SELECT poiFromId, poiToId,

truckCount / (truckCount + intermodalCount + unknownCount) as truckPct

FROM

(SELECT poiFromId, poiToId,

SUM(CASE WHEN mode = 'truck' THEN 1 ELSE 0 END) AS truckCount,

SUM(CASE WHEN mode = 'intermodal' THEN 1 ELSE 0 END) AS intermodalCount,

SUM(CASE WHEN mode = 'unknown' THEN 1 ELSE 0 END) AS unknownCount

FROM

(SELECT shipmentId, poiFromId, poiToId,

explode(modes) as mode

FROM shipment\_sums\_table

WHERE legCount = 1)

GROUP BY poiFromId, poiToId

ORDER BY poiFromId, poiToId

)

order by truckPct

''')

# Change a pyspark.sql.dataframe.DataFrame Data Type (CAST)

od\_pairs\_df = spark.sql('''SELECT DISTINCT a.prev\_poi, a.poi,

CONCAT(CAST(a.prev\_poi AS STRING), '-', CAST(a.poi as STRING)) as od\_pair

FROM leg\_2017\_transits a

LEFT OUTER JOIN pois\_data b ON a.prev\_poi = b.id

LIMIT 10''')

od\_pairs\_df.show(10)

Output:

+--------+-----+----------+

|prev\_poi| poi| od\_pair|

+--------+-----+----------+

| 4823| 5271| 4823-5271|

| 4823| 5003| 4823-5003|

...

# Constants

## pi()

# DataFrame (pyspark.sql.dataframe.DataFrame)

## Change a DataFrame’s Column Names

from pyspark.sql.functions import explode

df1.show()

+---+---+------------+

| a|col| c|

+---+---+------------+

| 1| 1| [7, 8, 9]|

| 1| 2| [7, 8, 9]|

| 1| 3| [7, 8, 9]|

| 2| 4|[10, 11, 12]|

| 2| 5|[10, 11, 12]|

| 2| 6|[10, 11, 12]|

+---+---+------------+

df2 = df1.selectExpr("a as alph", "col as exploded", "c as age")

df2.show()

Output:

+----+--------+------------+

|alph|exploded| age|

+----+--------+------------+

| 1| 1| [7, 8, 9]|

| 1| 2| [7, 8, 9]|

| 1| 3| [7, 8, 9]|

| 2| 4|[10, 11, 12]|

| 2| 5|[10, 11, 12]|

| 2| 6|[10, 11, 12]|

+----+--------+------------+

## Create a DataFrame from Scratch

df = sqlContext.createDataFrame([Row(a=1, b=[1,2,3],c=[7,8,9]), Row(a=2, b=[4,5,6],c=[10,11,12])])

df.show()

Output:

+---+---------+------------+

| a| b| c|

+---+---------+------------+

| 1|[1, 2, 3]| [7, 8, 9]|

| 2|[4, 5, 6]|[10, 11, 12]|

+---+---------+------------+

Note: This is tyape pyspark.sql.dataframe.DataFrame

## Execute SQL on a DataFrame

sqlContext.registerDataFrameAsTable(data, "myTable")

df2 = sqlContext.sql("SELECT Name AS name, askdaosdka as age from myTable")

df2.show()

# Output

#+-------+---+

#| name|age|

#+-------+---+

#|Alberto| 2|

#| Dakota| 2|

#+-------+---+

## Explode a List inside a DataFrame column

df = sqlContext.createDataFrame([Row(a=1, b=[1,2,3],c=[7,8,9]), Row(a=2, b=[4,5,6],c=[10,11,12])])

df.show()

Output:

+---+---------+------------+

| a| b| c|

+---+---------+------------+

| 1|[1, 2, 3]| [7, 8, 9]|

| 2|[4, 5, 6]|[10, 11, 12]|

+---+---------+------------+

df1 = df.select(df.a, explode(df.b), df.c)

df.show()

Output:

+---+---+------------+

| a|col| c|

+---+---+------------+

| 1| 1| [7, 8, 9]|

| 1| 2| [7, 8, 9]|

| 1| 3| [7, 8, 9]|

| 2| 4|[10, 11, 12]|

| 2| 5|[10, 11, 12]|

| 2| 6|[10, 11, 12]|

+---+---+------------+

## Select Distinct Rows of a pySpark DataFrame

port\_vessel\_mooring\_df.select('\*').**distinct()**.orderBy(['imo', 'poi', 'mooring\_start\_ts', 'mooring\_end\_ts', 'timestamp']).show()

# Distance on Earth Surface Using Haversine Equation

od\_pairs\_train\_df = spark.sql(

'''SELECT DISTINCT a.prev\_poi as leg\_orig\_poi, a.poi as leg\_dest\_poi,

CONCAT(CAST(a.prev\_poi AS STRING), '-', CAST(a.poi as STRING)) as od\_pair,

a.duration\_mean\_hrs, a.traversal\_count, a.duration\_median,

b.centroid\_lat as leg\_orig\_lat, b.centroid\_lng as leg\_orig\_lon, b.radius\_km as leg\_orig\_radius\_km,

c.centroid\_lat as leg\_dest\_lat, c.centroid\_lng as leg\_dest\_lon, c.radius\_km as leg\_dest\_radius\_km,

2 \* 6371 \* asin(sqrt(pow((sin(radians((c.centroid\_lat - b.centroid\_lat) / 2))), 2) +

cos(radians(b.centroid\_lat)) \* cos(radians(c.centroid\_lat)) \*

pow((sin(radians((c.centroid\_lng - b.centroid\_lng) / 2))), 2))) as dist\_km

FROM leg\_2017\_transits a

LEFT OUTER JOIN pois\_data b ON a.prev\_poi = b.id

LEFT OUTER JOIN pois\_data c ON a.poi = c.id

''')

Output:

+------------+------------+-----------+-----------------+---------------+------------------+------------+------------+------------------+------------+------------+------------------+------------------+

|leg\_orig\_poi|leg\_dest\_poi| od\_pair|duration\_mean\_hrs|traversal\_count| duration\_median|leg\_orig\_lat|leg\_orig\_lon|leg\_orig\_radius\_km|leg\_dest\_lat|leg\_dest\_lon|leg\_dest\_radius\_km| dist\_km|

+------------+------------+-----------+-----------------+---------------+------------------+------------+------------+------------------+------------+------------+------------------+------------------+

| 4877| 4894| 4877-4894| 9.9447220000| 2| 9.944721698760986| -26.9167| -48.6333| 6.335| -26.8948| -48.6551| 2.773| 3.256191534334778|

| 5564| 5556| 5564-5556| 166.6083330000| 1|166.60833740234375| 38.95| 118.5333| 38.208| 21.3667| 110.25| 3.872| 2108.692995101377|

| 9477| 9978| 9477-9978| 34.3502780000| 1|34.350276947021484| 53.5833| -0.7333| 3.117| 53.6| -0.7333| 3.1|1.8569552749641485|

| 10202| 9978| 10202-9978| 97.6855560000| 1| 97.68555450439453| 53.6| -0.75| 5.304| 53.6| -0.7333| 3.1| 1.101952329215023|

| 17161| 11434|17161-11434| 72.7575000000| 1| 72.75749969482422| 22.8083| 69.7008| 10.0| 15.4| 73.7833| 3.129| 928.5644530302043|

+------------+------------+-----------+-----------------+---------------+------------------+------------+------------+------------------+------------+------------+------------------+------------------+

# Invoke a pySpark UDF from SQL

from spark.sql.types import IntegerType()

# my python function example

def sum(effdate, trandate):

sum=effdate+trandate

return sum

spark.udf("sum", sum, IntegerType())

spark.sql("select sum(cm.effdate, cm.trandate)as totalsum, name from CMLEdG cm....").show()

# Joins

## Left Anti Join

Example: In this query, the traversal\_detail table has rows for many more vessels (IMOs) than does the vessel info table. So here, we are looking for all those vessels (IMOs) which we see in our traversal\_detail table, but for which we have no vessel info. At the time of this query, the traversal\_imos table had 23,658 distinct vessel IMOs, and 19,652 of them were missing from the vessel\_info table.

missing\_vessels\_df = sqlContext.sql(

'''

SELECT imo

FROM

(SELECT DISTINCT traversal\_detail.imo

FROM traversal\_detail

LEFT ANTI JOIN vessel\_info ON traversal\_detail.imo = vessel\_info.imo

)

'''

)

print missing\_vessels\_df.count()

missing\_vessels\_df.show()

Output:

19652

+-------+

| imo|

+-------+

|9066057|

|9673795|

|9518098|

|...

## Left Outer Join

llist = [('bob', '2015-01-13', 4), ('alice', '2015-04-23',10)]

left = spark.createDataFrame(llist, ['name','date','duration'])

right = spark.createDataFrame([('alice', 100),('bob', 23)],['name','upload'])

left.show()

right.show()

df\_015 = left.join(right, ['name'])

df\_015.show()

-or-

left.registerTempTable('left')

right.registerTempTable('right')

print "df\_017:"

df\_017 = spark.sql('''SELECT left.name, left.date, left.duration, right.upload

FROM left

LEFT OUTER JOIN right on left.name = right.name

''')

df\_017.show()

**Output:**

+-----+----------+--------+ “left”

| name| date|duration|

+-----+----------+--------+

| bob|2015-01-13| 4|

|alice|2015-04-23| 10|

+-----+----------+--------+

+-----+------+ “right”

| name|upload|

+-----+------+

|alice| 100|

| bob| 23|

+-----+------+

df\_017:

+-----+----------+--------+------+

| name| date|duration|upload|

+-----+----------+--------+------+

|alice|2015-04-23| 10| 100|

| bob|2015-01-13| 4| 23|

+-----+----------+--------+------+

test\_300\_2 = spark.sql('''SELECT a.imo, a.prev\_timestamp as leg\_start\_ts, a.timestamp as leg\_end\_ts,

a.prev\_poi as leg\_orig\_poi, a.poi as leg\_dest\_poi, a.duration as leg\_duration\_hrs,

b.smfPayloadData.timestamp.asOf as timestamp,

unix\_timestamp(b.smfPayloadData.timestamp.asOf, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") as unix\_ts,

b.smfPayloadData.geolocation.latitude,

b.smfPayloadData.geolocation.longitude, b.smfPayloadData.geolocation.speed,

c.centroid\_lat as dest\_lat, c.centroid\_lng as dest\_lon, c.radius\_km as dest\_poi\_radius\_km

FROM one\_leg\_transits\_test\_2017\_07 a

JOIN ais\_data\_2017\_07 b ON a.imo = b.smfPayloadData.paired.vesselId

AND unix\_timestamp(b.smfPayloadData.timestamp.asOf,

"yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") >= a.prev\_timestamp

AND unix\_timestamp(b.smfPayloadData.timestamp.asOf,

"yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") <= a.timestamp

JOIN pois\_data c ON a.poi = c.id

WHERE a.imo = '9189574'

ORDER BY unix\_ts

''')

test\_300\_2.show()

Another Example from <https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/2728434780191932/1483312212640900/6987336228780374/latest.html>

SELECT A.dep\_id,

A.employee\_id,

A.age,

B.max\_age

FROM employee A

LEFT OUTER JOIN (SELECT dep\_id,

MAX(age) max\_age

FROM employee B

GROUP BY dep\_id) B

ON B.dep\_id = A.dep\_id

ORDER BY 1,2

# LEAD and LAG

shipment\_lat\_lon\_df = sqlContext.sql('''

SELECT smfPayloadData.paired.deviceId,

smfPayloadData.paired.deviceMake,

smfPayloadData.paired.shipmentId,

smfPayloadData.paired.shipper,

smfPayloadData.paired.vesselId,

smfPayloadData.geolocation.asOf,

smfPayloadData.geolocation.latitude,

smfPayloadData.geolocation.longitude

FROM smurf\_processed

WHERE smfPayloadData.geolocation.latitude is not null

AND smfPayloadData.paired.deviceId is not null

AND smfPayloadData.paired.shipmentId is not null

AND smfPayloadData.paired.vesselId is not null

ORDER BY vesselId, asOf

LIMIT 100

''')

shipment\_lat\_lon\_df.show(100)

shipment\_lat\_lon\_df.registerTempTable('tbl\_shimpment\_lat\_lon\_df')

Output:

+--------------+------------------+----------+-------+--------+--------------------+--------+---------+

| deviceId| deviceMake|shipmentId|shipper|vesselId| asOf|latitude|longitude|

+--------------+------------------+----------+-------+--------+--------------------+--------+---------+

|86737803528965|sensor-savi-locate|0009424460| SYT|US--3113|2018-04-10T18:11:...| 40.6161| -89.4637|

|86737803528965|sensor-savi-locate|0009424460| SYT|US--3113|2018-04-10T18:31:...| 40.6161| -89.4637|

|86737803531687|sensor-savi-locate|0009427810| SYT|US--3114|2018-05-02T12:04:...| 40.6161| -89.4637|

|86737803531687|sensor-savi-locate|0009428134| SYT|US--3114|2018-05-02T12:04:...| 40.6161| -89.4637|

|86737803531687|sensor-savi-locate|0009428804| SYT|US--3114|2018-05-02T12:04:...| 40.6161| -89.4637|

|86737803531687|sensor-savi-locate|0009429154| SYT|US--3114|2018-05-02T13:03:...| 40.6161| -89.4637|

|86737803531687|sensor-savi-locate|0009427810| SYT|US--3114|2018-05-02T13:03:...| 40.6161| -89.4637|

shipment\_lat\_lon\_with\_prior\_df = sqlContext.sql('''

SELECT a.\*,

LAG(asOf, 1) OVER (PARTITION BY vesselId ORDER BY asOf) AS prev\_asOf

FROM tbl\_shimpment\_lat\_lon\_df a

''')

Output:

+--------------+------------------+----------+-------+--------+--------------------+--------+---------+--------------------+

| deviceId| deviceMake|shipmentId|shipper|vesselId| asOf|latitude|longitude| last\_asOf|

+--------------+------------------+----------+-------+--------+--------------------+--------+---------+--------------------+

|86737803528965|sensor-savi-locate|0009424460| SYT|US--3113|2018-04-10T18:11:...| 40.6161| -89.4637| null|

|86737803528965|sensor-savi-locate|0009424460| SYT|US--3113|2018-04-10T18:31:...| 40.6161| -89.4637|2018-04-10T18:11:...|

|86737803531687|sensor-savi-locate|0009428804| SYT|US--3114|2018-05-02T12:04:...| 40.6161| -89.4637| null|

|86737803531687|sensor-savi-locate|0009427810| SYT|US--3114|2018-05-02T12:04:...| 40.6161| -89.4637|2018-05-02T12:04:...|

|86737803531687|sensor-savi-locate|0009428134| SYT|US--3114|2018-05-02T12:04:...| 40.6161| -89.4637|2018-05-02T12:04:...|

+--------------+------------------+----------+-------+--------+--------------------+--------+---------+--------------------+

# Loading a Schema while Correcting org.apache.parquet.io.ParquetDecodingException

This throws an Exception:

smf\_parquet = sqlContext.read.parquet( '/lambda/smf-parquet/POC/smurf-processed//quarter=2017-q1/\*',

'/lambda/smf-parquet/POC/smurf-processed//quarter=2017-q2/\*',

'/lambda/smf-parquet/POC/smurf-processed//quarter=2017-q3/\*')

smf\_parquet.registerTempTable('smurf\_processed')

test\_df = sqlContext.sql('''

SELECT \*

FROM smurf\_processed

LIMIT 10''')

test\_df.take(5)

but the following works:

merged\_df = spark.read.option("mergeSchema", "true").parquet( '/lambda/smf-parquet/POC/smurf-processed//quarter=2017-q1/\*',  
'/lambda/smf-parquet/POC/smurf-processed//quarter=2017-q2/\*')

# Read a Parquet File into a DataFrame

shipment\_summaries\_path = '/lambda/summaries-parquet/PG/shipment-summaries/\*'

shipment\_sums = sqlContext.read.parquet(shipment\_summaries\_path) # pyspark.sql.dataframe.DataFrame

# Rounding Numerical Values

port\_incoming\_vessel\_anchorings = sqlContext.sql(

'''

SELECT leg\_dest\_poi, long\_name, count(\*) as arrival\_count, sum(vessel\_anchored) as anchored\_count,

**CAST(100.0 \* sum(vessel\_anchored) / count(\*) AS DECIMAL(8,1)) as anchored\_pct**

Output:

+------------+--------------------+-------------+--------------+------------+

|leg\_dest\_poi| long\_name|arrival\_count|anchored\_count|anchored\_pct|

+------------+--------------------+-------------+--------------+------------+

| 4778| ZANDVLIET [BEZVT]| 777| 45| 5.8|

| 6012| FAMAGUSTA [CYFMG]| 51| 16| 31.4|

# Store DataFrame as Temp Table for Later Querying

shipment\_sums.createOrReplaceTempView("shipment\_sums\_table")

Query from the Temp Table:

test\_df = sqlContext.sql('''SELECT shipmentId FROM shipment\_sums\_table limit 1''')

test\_df.head()

Output:

Row(shipmentId=u'0306098015')

# Select an Array Element

df.printSchema()

## root

## |-- stuff: array (nullable = true)

## | |-- element: struct (containsNull = true)

## | | |-- a: long (nullable = true)

## | | |-- b: long (nullable = true)

## | | |-- c: long (nullable = true)

sqlContext.sql("SELECT stuff[0].a FROM df").show()

## +---+

## |\_c0|

## +---+

## | 1|

## +---+

# Parameterized Queries

q25 = 500

var2 = 50

Q1 = spark.sql("SELECT col1 from table where col2>{0} limit {1}".format(var2,q25))

**Note: Be sure to wrap character literals in quotes like:**

od\_pair\_train\_df = spark.sql('''

SELECT \*

FROM od\_pairs\_train opt

WHERE od\_pair = **'**{0}**'**

LIMIT 10

'''.format(od\_pair)

## Passing a List or Tuple in a Parameterized Query

Hint: Convert the List to a tuple

syngenta\_truck\_df = sqlContext.sql('''

SELECT smfPayloadData.paired.shipper,

smfPayloadData.paired.shipmentId,

smfPayloadData.paired.vesselId,

smfPayloadData.paired.vesselName,

smfPayloadData.timestamp.asOf,

smfPayloadData.geolocation.latitude,

smfPayloadData.geolocation.longitude,

smfPayloadData.geolocation.detectedPois,

smfPayloadData.computations.CurrentShipmentState.humanizedState.motion,

smfPayloadData.computations.CurrentShipmentState.humanizedState.currentLeg,

smfPayloadData.computations.CurrentShipmentState.humanizedState.totalLegs,

smfPayloadData.computations.CurrentShipmentState.humanizedState.lastSpName,

smfPayloadData.computations.CurrentShipmentState.humanizedState.nextSpName,

smfPayloadData.computations.CurrentShipmentState.detectedPoiState,

smfPayloadData.computations.CurrentShipmentState.departureTimesAtShippingPoint

FROM smurf\_processed

WHERE smfPayloadData.paired.shipmentId in {0}

ORDER BY smfPayloadData.timestamp.asOf

'''.format(tuple(shipment\_ids))

)

# Parquet

## Explode a list or array data field

shipment\_df = sqlContext.sql('''SELECT shipmentId, modes

FROM shipment\_sums\_table

where legCount = 2

limit 5

''')

shipment\_df.show()

Output:

+----------+--------------+

|shipmentId| modes|

+----------+--------------+

|0306080481|[truck, truck]|

|0306120361|[truck, truck]|

|0306120411|[truck, truck]|

|0306103320|[truck, truck]|

|0306078031|[truck, truck]|

+----------+--------------+

# What modes are available?

modes\_df = sqlContext.sql('''SELECT shipmentId,

explode(distinct modes)as mode

FROM shipment\_sums\_table

WHERE legCount = 2

limit 6

''')

modes\_df.show()

Output:

+----------+-----+

|shipmentId| mode|

+----------+-----+

|0306080481|truck|

|0306080481|truck|

|0306120361|truck|

|0306120411|truck|

+----------+-----+

**Exploding an Array stored in a Column**

Unexploded:

start\_moored = sqlContext.sql('''

SELECT timestamp, mmsi, pois

FROM (

SELECT \*

FROM pos\_data

WHERE moored = 1 and prev\_moored = 0 and prev2\_moored = 0 and prev3\_moored = 0 and prev4\_moored = 0 and prev5\_moored = 0 and prev6\_moored = 0 and prev7\_moored = 0

) a

'''

)

Output:

+----------+---------+--------------+

| timestamp| mmsi| pois|

+----------+---------+--------------+

|1493653149|212370000| [12223]|

|1493847898|212370000|[14952, 15249]|

|1493949978|212370000| [12223]|

|1494075847|212370000| [12275]|

|1494258522|212370000| [12223]|

Exploded:

start\_moored = sqlContext.sql('''

SELECT timestamp, mmsi, poi

FROM (

SELECT \*

FROM pos\_data

WHERE moored = 1 and prev\_moored = 0 and prev2\_moored = 0 and prev3\_moored = 0 and prev4\_moored = 0 and prev5\_moored = 0 and prev6\_moored = 0 and prev7\_moored = 0

) a

LATERAL VIEW OUTER EXPLODE(pois) zzz AS poi

'''

Output:

+----------+---------+-----+

| timestamp| mmsi| poi|

+----------+---------+-----+

|1493653149|212370000|12223|

|1493847898|212370000|14952| # These were

|1493847898|212370000|15249| # exploded

|1493949978|212370000|12223|

|1494075847|212370000|12275|

|1494258522|212370000|12223|

|1494419315|212370000|14952|

|1494419315|212370000|15249|

|1494621881|212370000|12275|

|1494913457|212370000|12223|

+----------+---------+-----+

# Statistics

## Average

outliers\_df = sqlContext.sql('''

SELECT prev\_poi, poi,

avg(duration)

FROM legSummaries

GROUP BY prev\_poi, poi

LIMIT 5

''')

outliers\_df.show(5)

Output:

+--------+-----+-------------+

|prev\_poi| poi|avg(duration)|

+--------+-----+-------------+

| 7361|11322|55.1361110000|

| 9991| 9469| 4.3386058879|

...

## Median

stats\_df = sqlContext.sql('''

SELECT imo, prev\_poi, poi,

avg(duration) as leg\_mean\_hrs,

**percentile\_approx(duration, 0.5) as median\_hrs**,

stddev(duration) as leg\_stddev\_hrs,

count(\*)

FROM legSummaries

WHERE imo is not null

AND poi != -1

GROUP BY imo, prev\_poi, poi

HAVING count(\*) > 1

AND median\_hrs >= 8.0

LIMIT 10

''')

stats\_df.registerTempTable('stats')

stats\_df.show(10)

Output:

+---------+--------+-----+-------------+----------+------------------+--------+

| imo|prev\_poi| poi| leg\_mean\_hrs|median\_hrs| leg\_stddev\_hrs|count(1)|

+---------+--------+-----+-------------+----------+------------------+--------+

|135206012| 5749| 5634|20.8236110000| 21.241389|0.5908273136611072| 2|

|402653322| 5863| 5862|36.6962036667| 46.911944|31.852472485088068| 3|

| 6507983| 6757|17133| 9.1079170000| 11.863056| 3.896354940023047| 2|

# Strings

## Concatenate strings

ais\_imo\_750\_df = spark.sql(

''' SELECT (concat(string(1234), '-', string(2345))) as test

'''

)

ais\_imo\_750\_df.show()

print ais\_imo\_750\_df.dtypes

Output:

+---------+

| test|

+---------+

|1234-2345|

+---------+

[('test', 'string')]

## Convert a numeric field to a string

ais\_imo\_750\_df = spark.sql(

''' SELECT (concat(string(1234), '-', string(2345))) as test

'''

)

ais\_imo\_750\_df.show()

print ais\_imo\_750\_df.dtypes

Output:

+---------+

| test|

+---------+

|1234-2345|

+---------+

[('test', 'string')]

test = sqlContext.sql('''

SELECT DISTINCT format\_string("%s", poiFromId) as originId,

format\_string("%s", poiToId) as destId,

format\_string("%s-%s", poiFromId, poiToId) as od\_pair

FROM shipment\_sums

LIMIT 5

''')

print type(test)

test.show(5)

Output:

+--------+------+-------+

|originId|destId|od\_pair|

+--------+------+-------+

| 39| 981| 39-981|

| 31| 1056|31-1056|

| 38| 547| 38-547|

| 33| 174| 33-174|

| 37| 40| 37-40|

+--------+------+-------+

## Convert a String to UPPER or lower case

SELECT \*

FROM

(SELECT \*, to\_utc\_timestamp(eta, 'UTC') as eta\_ts\_loc,

to\_utc\_timestamp(etd, 'UTC') as etd\_ts\_loc,

to\_utc\_timestamp(ctime, 'UTC') as ctime\_ts\_utc,

to\_utc\_timestamp(stime, 'UTC') as stime\_ts\_utc,

p.tz as port\_tz

FROM ls\_sched schd

JOIN pois p on **UPPER**(p.short\_name) = **UPPER**(schd.pname)

)

WHERE unix\_timestamp(ctime\_ts\_utc) - unix\_timestamp(stime\_ts\_utc) < (7 \* 24 \* 60 \* 60)

## Match Part of a String, Using Wildcards

pois\_data\_df = sqlContext.sql(

'''

SELECT id, poi\_type, short\_name, long\_name, centroid\_lat, centroid\_lng, radius\_km

FROM pois\_data

WHERE poi\_type = 'marinePort'

AND long\_name LIKE '**%OAKLAND%**’

LIMIT 50

'''

)

pois\_data\_df.show(50)

Output:

+-----+----------+----------+---------------+------------+------------+---------+

| id| poi\_type|short\_name| long\_name|centroid\_lat|centroid\_lng|radius\_km|

+-----+----------+----------+---------------+------------+------------+---------+

|15597|marinePort| OAKLAND|OAKLAND [USOAK]| 37.8167| -122.3333| 5.142|

+-----+----------+----------+---------------+------------+------------+---------+

## Substring

SUBSTR(smfPayloadData.timestamp.asOf,1,10)

output: “2016-03-01”

# Time Intervals

+--------------------+--------------------+--------------------+

| shipment\_id| port\_start\_ts| port\_end\_ts|

+--------------------+--------------------+--------------------+

| 27690030954076|2018-03-11T18:26:...|2018-03-13T08:14:...|

|27690031135535\_YM...|2018-02-20T18:55:...|2018-02-22T01:00:...|

|27690030954052\_YM...|2018-03-03T21:57:...|2018-03-05T08:20:...|

|27690030954069\_YM...|2018-03-05T20:49:...|2018-03-07T18:51:...|

|27690030699786\_YM...|2018-02-04T19:59:...|2018-02-06T09:26:...|

+--------------------+--------------------+--------------------+

timeFmt = "yyyy-MM-dd'T'HH:mm:ss.SSS"

timeDiff = (unix\_timestamp('port\_end\_ts', format=timeFmt)

- unix\_timestamp('port\_start\_ts', format=timeFmt))

df = df.withColumn("Duration", timeDiff)

df.show()

Output:

+--------------------+--------------------+--------------------+--------+

| shipment\_id| port\_start\_ts| port\_end\_ts|Duration|

+--------------------+--------------------+--------------------+--------+

| 27690030954076|2018-03-11T18:26:...|2018-03-13T08:14:...| 136058|

|27690031135535\_YM...|2018-02-20T18:55:...|2018-02-22T01:00:...| 108287|

|27690030954052\_YM...|2018-03-03T21:57:...|2018-03-05T08:20:...| 123786|

|27690030954069\_YM...|2018-03-05T20:49:...|2018-03-07T18:51:...| 165738|

|27690030699786\_YM...|2018-02-04T19:59:...|2018-02-06T09:26:...| 134809|

+--------------------+--------------------+--------------------+--------+

Another example:

test\_847 = sqlContext.sql('''SELECT unix\_timestamp('2019-04-15 12:00:00') –

unix\_timestamp('2019-04-15 13:00:00') as delta\_sec''')

test\_847.show()

Output:

+---------+

|delta\_sec|

+---------+

| -3600|

+---------+

# Timestamps

## Convert a timestamp in UTC to a timestamp in a Local Timezone

test\_754\_df = sqlContext.sql('''SELECT from\_utc\_timestamp('2016-03-01 00:00:00','UTC') as utct,

from\_utc\_timestamp('2016-03-01 00:00:00','Pacific/Honolulu') as local\_time''')

test\_754\_df.show(truncate=False)

test\_754\_df.dtypes

Output:

+---------------------+---------------------+

|utct |local\_time |

+---------------------+---------------------+

|2016-03-01 00:00:00.0|2016-02-29 14:00:00.0|

+---------------------+---------------------+

[('utct', 'timestamp'), ('local\_time', 'timestamp')]

## Convert string in a local timezone to a UTC timestamp

test\_541\_df = sqlContext.sql(

'''

SELECT to\_utc\_timestamp('2016-03-01 00:00:00', 'America/New\_York') as example\_timestamp\_utc

''')

test\_541\_df.show(truncate=False)

print test\_541\_df.dtypes

Output:

+---------------------+

|example\_timestamp\_utc|

+---------------------+

|2016-03-01 05:00:00.0|

+---------------------+

[('example\_timestamp\_utc', 'timestamp')]

## Convert string to timestamp

shipment\_carr = sqlContext.sql('''

SELECT smfPayloadData.paired.shipmentId,

smfPayloadData.paired.shipper,

smfPayloadData.paired.carrier,

smfPayloadData.timestamp.asOf as dest\_arr\_utc,

to\_utc\_timestamp('2016-03-01 00:00:00','UTC') as example\_timestamp

FROM mapped

where smfPayloadData.events.DeliveryLocationArrival is not null

and smfPayloadData.timestamp.asOf >= to\_utc\_timestamp('2016-03-01 00:00:00','UTC')

and smfPayloadData.timestamp.asOf < to\_utc\_timestamp('2016-03-02 00:00:00','UTC')

limit 1

''')

printResultsAsJson(shipment\_carr)

output:

{

**"example\_timestamp": "2016-03-01 00:00:00.0"**,

"carrier": "SCNN",

"shipper": "PG",

"dest\_arr\_utc": "2016-03-01T23:51:00.000Z",

"shipmentId": "Jackie will add"

}

Another way:

test\_847 = sqlContext.sql('''SELECT CAST('2019-04-15 12:00:00' as TIMESTAMP)''')

test\_847.show(truncate=False)

Output:

+--------------------------------------+

|CAST(2019-04-15 12:00:00 AS TIMESTAMP)|

+--------------------------------------+

|2019-04-15 12:00:00.0 |

+--------------------------------------+

## Convert ISO 8601 Standard DateTime to unix Timestamp

test\_ais\_data\_135206012\_df = spark.sql(

'''SELECT smfPayloadData.timestamp.asOf,

**unix\_timestamp(smfPayloadData.timestamp.asOf, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") as unix\_ts**,

outputs a unix\_ts like:

+--------------------+----------+---------+----------+------------------+--------+---------+-----+-------+------------+

| asOf| unix\_ts| imo|vesselName|vesselTelematicsId|latitude|longitude|speed|heading|detectedPois|

+--------------------+----------+---------+----------+------------------+--------+---------+-----+-------+------------+

|2017-07-01T00:08:...|**1498867725**|135206012| JINLUN| 4122

## Convert Unix Timestamp to Formatted Date

mooring\_exp\_df = sqlContext.sql('''

SELECT \*,

from\_unixtime(timestamp, 'YYYY-MM-dd') as timestamp\_dt

FROM start\_moored

ORDER BY mmsi, timestamp

LIMIT 10

'''

)

+----------+---------+-----+------------+

| timestamp| mmsi| poi|timestamp\_dt|

+----------+---------+-----+------------+

|1493942222|173313120|12052| 2017-05-04|

|1494196603|173313120|12043| 2017-05-07|

|1494472207|173313120|12052| 2017-05-11|

...

## Convert Unix Epoch to a pySpark Timestamp

# Experiment

test\_055\_df = sqlContext.sql(

'''

SELECT 1514919810 as epoch,

from\_unixtime(1514919810, "y-MM-dd'T'hh:mm:ss'Z'") as origin\_date\_str,

to\_utc\_timestamp(from\_unixtime(1514919810, "y-MM-dd'T'hh:mm:ss'Z'"), 'UTC') as ts\_utc

'''

)

test\_055\_df.show(truncate=False)

print test\_055\_df.dtypes

Output:

+----------+--------------------+---------------------+

|epoch |origin\_date\_str |ts\_utc |

+----------+--------------------+---------------------+

|1514919810|2018-01-02T07:03:30Z|2018-01-02 07:03:30.0|

+----------+--------------------+---------------------+

[('epoch', 'int'), ('origin\_date\_str', 'string'), ('ts\_utc', 'timestamp')]

## Convert Unix Epoch Timestamp to ISO Standard Date and Time String

Note: This creates a string output, not a datetime

one\_leg\_transits\_test\_2017\_07\_df = spark.sql(

'''SELECT \*,

from\_unixtime(prev\_timestamp, "y-MM-dd'T'hh:mm:ss'Z'") as origin\_datetime

FROM one\_leg\_2017\_07

WHERE prev\_poi != 0

AND prev\_timestamp is not null

AND prev\_poi != poi

''')

one\_leg\_transits\_test\_2017\_07\_df.show(10)

Output:

+---------+-------+--------------+----------+--------+-----+-----------+------+---------+--------------------+

| mmsi| imo|prev\_timestamp| timestamp|prev\_poi| poi|prev\_arrive|arrive| duration| origin\_datetime|

+---------+-------+--------------+----------+--------+-----+-----------+------+---------+--------------------+

|212370000|9189574| 1498906282|1499099051| 12275|15643| 0| 1|53.546944|**2017-07-01T10:51:22Z**|

|212370000|9189574| 1499246131|1499453431| 15643|14952| 0| 1|57.583333|2017-07-05T09:15:31Z|

## Extract a Date Part from a Unix Epoch

test\_055\_df = sqlContext.sql(

'''

SELECT 1514919810 as epoch,

from\_unixtime(1514919810, "y-MM-dd'T'hh:mm:ss'Z'") as origin\_date\_str,

month(from\_unixtime(1514919810, "y-MM-dd'T'hh:mm:ss'Z'")) as month\_int,

to\_utc\_timestamp(from\_unixtime(1514919810, "y-MM-dd'T'hh:mm:ss'Z'"), 'UTC') as ts\_utc

'''

)

test\_055\_df.show(truncate=False)

print test\_055\_df.dtypes

Output:

+----------+--------------------+---------+---------------------+

|epoch |origin\_date\_str |month\_val|ts\_utc |

+----------+--------------------+---------+---------------------+

|1514919810|2018-01-02T07:03:30Z|1 |2018-01-02 07:03:30.0|

+----------+--------------------+---------+---------------------+

[('epoch', 'int'), ('origin\_date\_str', 'string'), ('month\_val', 'int'), ('ts\_utc', 'timestamp')]

For a list of the Date Part functions you can call, see:

<https://docs-snaplogic.atlassian.net/wiki/spaces/SD/pages/2458071/Date+Functions+and+Properties+Spark+SQL>

# Windowing Functions (like Group By ... Having)

These are used to compute min, max of groups in a table. (Similar to the Group By + Having).

<https://stackoverflow.com/questions/48829993/groupby-column-and-filter-rows-with-maximum-value-in-pyspark/48830780>

import pyspark.sql.functions as f

data = [

('a', 5),

('a', 8),

('a', 7),

('b', 1),

('b', 3)

]

df = sqlCtx.createDataFrame(data, ["A", "B"])

df.show()

#+---+---+

#| A| B|

#+---+---+

#| a| 5|

#| a| 8|

#| a| 7|

#| b| 1|

#| b| 3|

#+---+---+

df.registerTempTable('table')

q = "SELECT A, B FROM (SELECT \*, MAX(B) OVER (PARTITION BY A) AS maxB FROM table) M WHERE B = maxB"

sqlCtx.sql(q).show()

#+---+---+

#| A| B|

#+---+---+

#| b| 3|

#| a| 8|

#+---+---+