

Loading files: (after loading files , data will be in Data frames)

```
# TSV/CSV (Structued Data)
```

```
#get the schema from the data (Use inferSchema)
```

```
#User define Schema(use it based on schema else use InferSchema)
```

```
#JSON/PARQUET( SEMI STRUCTURED DATA)
```

```
#Write the dataset into file
```

```
df=spark.read.json(<"filepath.json">) # for reading json
```

```
df=spark.read.csv(<"filepath".csv>) # for reading csv file
```

```
df=spark.read.parquet(<"filepath".parquet>) # for reading parquet file
```

```
# for reading tsv file (Tab seperated file) use CSV and define seperator using options and heade is used to display header
```

```
#infer schema converst the schema as per the data automatically-Kind of learning and siplayig the schema
```

```
df=spark.read.option("sep","\t").options("header",True).option("inferSchema",True).csv(<"filepath">)
```

```
#General method to laod any file
```

```
df=Spark.read.format(<format of file>).load(<filepath>)
```

```
df=(spark.read.  
      csv("/FileStore/tables/BroadcastLogs_2018_Q3_M8_sample.csv",  
           sep="|",  
           header=True,  
           inferSchema=True  
      )  
)
```

```
# Creating User define schema and using it while Loading the data
```

```
(This is not only used to define the Data type but also control the number of columns to be in DF)
```

```
# Format is
```

```
StructType([
    StructFiled("Field Name1",Datatype,Nullable)
    StructFiled("Field Name2",Datatype,Nullable)
    StructFiled("Field Name3",Datatype,Nullable)
])
```

```
#Example:
```

```
from pyspark.sql.types import StructType,StructFiled,StringType,IntgerType
```

```
UserdefinedSchema=StructType([
```

```
    StructFiled("TimeStamp",StringType(),True)
    StructFiled("Site",StringType(),True)
    StructFiled("Requestid",IntgerType(),True)
])
```

```
#Using user defined schema while loading the file
```

```
df=spark.read.option("sep","\t").option("header",True).schema(UserdefinedSchema).csv("filename.tsv")
```

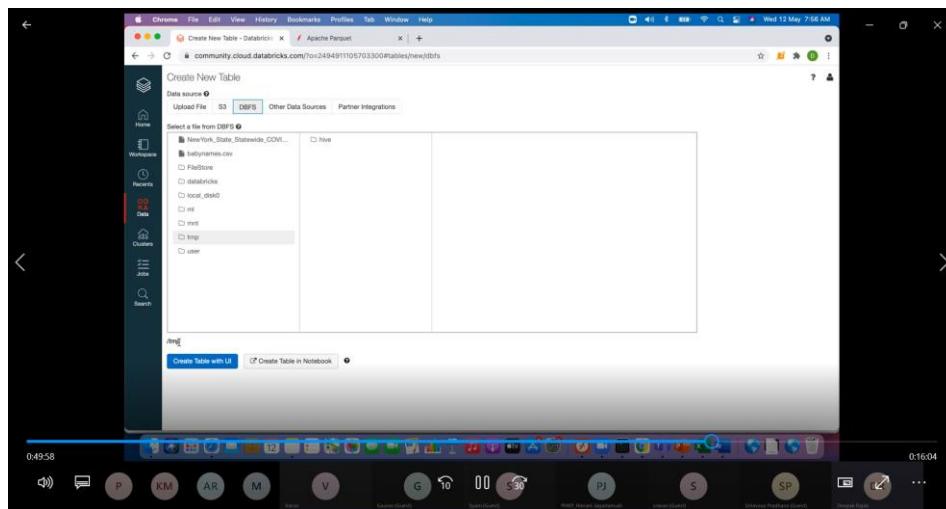
```
df=spark.read.option("inferSchema",True).schema(UserdefinedSchema).parquet("filename.parquet")
```

#DDL Formatting for creating schema

```
ddlschema="string, request integer"
```

```
df=spark.read.schema(ddlschema).parquet(filename.parquet)
```

Writing dataset to file:



```

1 json_source_path = "/mnt/training/zips.json"
2
3 zipsDF = spark.read.option("inferSchema", True).json(json_source_path)

# (1) Spark Job
# (2) zipDF.parquet(database.DataFrame = [Int string, city: string ... 3 more fields])

Comment took 1.35 seconds -- by deepak@juliebeavis.com at 12/05/2021, 07:32:14 on My New Cluster
Cell 28
1 target_path = "/temp/zips"
2
3 zipsDF.write.

```

Shift-enter to run

#default file format is parquet

Target_path="/temp/zips"

df.write.save(Target_path) is same as **df.write.parquet(Target_path)** as the default is parquet.

df.write.format(<file format>).mode("overwrite").save(Target_path)

#Reas and write the files in one go:

Spark.read.option("inferSchema",True).json(json_source_path).write.mode("overwrite").save(Target_path).

Note:if no mode is applied while writing then it will be error mode by default

OutputModes:

overwrite , append , error , ignore

Deleting files

dbutils.fs.rm(<filepath>)

```
1 dbutils.fs.rm("/tmp/df1.parquet", True)
```

Day 4

#Read the source(file/kafka/Mango DB) -Extraction
#Transformation (Applying business Logic)
#Loading (Store into file/DB/Streamimg(kafka))
#Transformation(select,filter,join,groupby ,drop) & Action(saving the data,displaying,counting,first n display)
#Transformations are Lazy
#Actions are Eager

Agenda:

#How can we RENAME a COLUMN?
#How can we change the datatype of a column?
#How can we change the value of the existing column?
#How can we derive new column from an existing?

#How can we RENAME a COLUMN?

```
Sourcepath="/mnt/training/wikipedia/pageviews/pageviews_by_second.tsv"  
df=(spark  
    .read  
    .option("sep","\t")  
    .option("header",True)  
    .csv(sourcePath)  
)  
  
## timestamp -→ recordedTime is a Transformation and will not display and results  
## site->Column2  
  
newdf=df.withColumnRenamed("timestamp","recordedTime").withColumnRenamed("site","Column2") #chaning the columns  
  
display(newdf) -→is an action will display the resultst
```

#To change all the column

```
anotherdf=df.toDF("c1","c2","c3")  
display(anotherdf)
```

#How can we change the datatype of a column?

```
#requests(int)→String  
  
from pyspark.sql.functions import col  
  
newdf=df.withColumn("requests",col("requests").cast("string"))  
  
display(newdf)
```

#How can we change the value of the existing column?

```
from pyspark.sql.functions import col  
  
newdf=df.withColumn("requests",col("requests")*2) #Mutiplying requests column by 2
```

#How can we derive new column from an existing?

```
#creating a new column newcolumnname and assigning it a value which is twice the value of requests column
```

```
from pyspark.sql.functions import col,lit
```

```
newdf=df.withColumn("newcolumnname",col("requests")*2)
```

#creating multiple columns

```
newdf=df.withColumn("newcolumnname",col("requests")*2).withColumn("country",lit("USA"))
```

Note: `lit` function is used for supplying Constant Values

#Reading from source -file /table/Streaming (kafka)

```
#range-id  
  
#createDataframe -Value  
  
#range(start(optional),number of records,increament(optional),number of partitions(optional))
```

```
df=spark.range(10) -by default column name is id
```

```
display(df)
```

▶ (3) Spark Jobs

	id
1	0
2	1
3	2
4	3
5	4
6	5

Showing all 10 rows.

```
df=spark.range(1,10,2)
```

```
display(df)
```

▶ (3) Spark Jobs

	id
1	1
2	3
3	5
4	7
5	9

Showing all 5 rows.

createDataFrame

```
#createDataFram()
```

```
from pyspark.sql.types import integerType
```

```
data=[1,2,3,4]
```

```
df=spark.createDataFrame(data,integerType()) #Default column name is value
```

UNION:

```
myschema=StructType([
```

```
    StructFiled("id",integerType(),True),
    StructFiled("name",StringType(),True)
])
```

```

My_list=['deepak','leena','ankit','madhu']

Data=[[0,'deepak'],[1,'leena'],[2,'ankit'],[3,'madhu']]

#id should be generated -used enumerate

df=spark.createDataFrame(enumerate(my_list),schema=myschema)

or

df=spark.createDataFrame(Data,schema=myschema)

#create 2 records with nulls

nulldf=spark.createDataFrame([[None,'priya'], [5,None]],schema=myschema)

nulldf.show() #in first record id is null and in second record name is null

appended_df=df.union(nulldf)

appended_df.show()

+---+---+
| id| name|
+---+---+
| null|priya|
| 5| null|
+---+---+
+---+---+
| id| name|
+---+---+
| 0|deepak|
| 1| leena|
| 2| ankit|
| 3| madhu|
| null| priya|
| 5| null|
+---+---+

```

FILTERS:

```

from pyspark.sql.functions import col

missing_name_df=appended_df.filter(col("name").isNull()) #retrieves only rows with missing names

print("-----missing names-----")
display(missing_name_df)

missing_id_df=appended_df.filter(col("id").isNull()) #retrieves only rows with missing id

```

```
print("-----missing id-----")
display(missing_id_df)
```

-----Missing Name-----	
id	name
5	null
-----Missing Id-----	
id	name
null	priya

COUNT AND DISTINCT COUNT:

df

	date	delay	distance	origin	destination
1	1011245	6	602	ABE	ATL
2	1020600	-8	369	ABE	DTW
3	1021245	-2	602	ABE	ATL
4	1020605	-4	602	ABE	ATL
5	1031245	-4	602	ABE	ATL
6	1030605	0	602	ABE	ATL

Showing the first 1000 rows.

#Count total flights leaving from a particular source

```
newdf=(df
       .groupby("origin")
       .agg({'destination' : 'count'})
       .withColumnRenamed(count(destination),"totalNoofFlights")
       )
```

```

· (2) Spark Jobs
  ↘ csvDF: pyspark.sql.dataframe.DataFrame
    date: integer
    delay: integer
    distance: integer
    origin: string
    destination: string
  ↘ newDF: pyspark.sql.dataframe.DataFrame
    origin: string
    totalNoofFlights: long

```

display(newdf)

	origin	totalNoofFlights
1	GEG	2044
2	BUR	5079
3	GRB	1109
4	GTF	425
5	GRR	2585
6	EUG	1271

```

-----Total Count-----
+-----+-----+
|origin|totalCounts|
+-----+-----+
|  GEG|     2044|
|  BUR|     5079|
|  GRB|     1109|
|  GTF|      425|
|  GRR|     2585|
+-----+-----+
only showing top 5 rows

```

#Count only distinct destination

```

from pyspark.sql.functions import countDistinct

uniquedf=(df
    .groupby("origin")
    .agg(countDistinct('destination').alias('uniquedest'))
)
Uniquedf.show()

```

-----Total Count Distinct-----	
origin uniquedest	
MSY 35	
GEG 9	
BUR 11	

FILTER AND WHERE WORK THE SAME WAY:

WMD 4.1

FILTER vs WHERE

```

1 from pyspark.sql.functions import col
2
3 parquetDF = spark.read.parquet("/mnt/training/City-Data.parquet") # Create DF via Parquet File
4
5 print("-----Printing first 1 records of Parquet DF-----")
6 parquetDF.show(1)
7
8 filteredDF = parquetDF.filter(col("state") == "New York") # filter() records from state = New York
9 print("-----Printing first 1 records of Filtered DF-----\n")
10 filteredDF.show(1)
11
12 filter_by_where_df = parquetDF.where((col("state") == "California") & (col("city") == "Los Angeles"))
13
14 print("-----Printing first 1 records of Filtered DF by where()-----\n")
15 filter_by_where_df.show() # The where() clause is equivalent to filter()

```

ORDERBY OR SORTBY.BOTH ARE SAME

Usage:

`df.sort("state","city").show()`

`df.sort("state","city").ascn().show() #by default descending`

`df.orderby("state","city",ascending=0).show(4) #ascending =1 means ascending`

GROUP BY AND ORDER BY:

LMO 4.5

GROUPBY ORDERBY

```
1 parquetDF = spark.read.parquet("/mnt/training/City-Data.parquet") # Create DF via Parquet File
2 print("-----Printing first 5 records of Parquet DF-----")
3 parquetDF.show(5)
4
5 # Apply GroupBy and Agg Operations and Orderby - state desc
6
7 groupedDF = (parquetDF
8     .groupBy("state")
9     .agg({"city": "count"}) # count(city)
10    .withColumnRenamed("count(city)", "NoOfCities")
11    .orderBy("state", ascending = 0)
12 ) # option 0 for descending
13
14 print("-----Printing first 5 records of Grouped / Aggregated / ordered DF-----")
15
16 groupedDF.write.mode("overwrite").save("/tmp/citydf/grouped.parquet")
```

AGGREGATION ON MULTIPLE COLUMNS

AGG on MULTIPLE COLUMNS

```
1 parquetDF = spark.read.parquet("/mnt/training/City-Data.parquet") # Create DF via Parquet FILE
2
3 # groupBy and aggregate on single column
4
5 (parquetDF
6     .groupBy("state")
7     .min("population2010")
8     .withColumnRenamed("min(population2010)", "minPopulation")
9     .orderBy("state", ascending = 0)
10    .show(4)
11 )
12
13 # groupBy and aggregate on multiple columns
14
15 from pyspark.sql.functions import min, max, avg, sum, mean, count
16
17 (parquetDF
18     .groupBy("state") [
19         .agg(min("population2010").alias("min_population")) \
20             ,max("population2010").alias("max_population") \
21             ,sum("estPopulation2016").alias("est_population") \
22             ,count("city").alias("total_no_cities") \
23             ,mean("estPopulation2016").alias("mean_estPopulation2016")) \
24
25     .orderBy("state", ascending = 1).show(5)
26 ) # ascending True
```

DISTINCT:

```
distinctdf=df.distinct()
```

DROPPDuplicates:

```
distinctdf=df.dropDuplicates()
```

SELECT:

```
from pyspark.sql.functions import col  
nDF=df.select("origin","destination").filter(col("origin")=="ABE")--Narrow Dependency/Transformation-
```

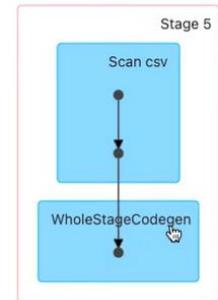
```
▼ nDF: pyspark.sql.dataframe.DataFrame  
  origin: string  
  destination: string
```

```
display(nDF)
```

```
1 display(nDF)
```

```
▼ (3) Spark Jobs  
  ▷ Job 3 View (Stages: 1/1)  
  ▷ Job 4 View (Stages: 1/1)  
  ▷ Job 5 View (Stages: 1/1)
```

	origin	destination
1	ABE	ATL
2	ABE	DTW
3	ABE	ATL
4	ABE	ATL
5	ABE	ATL
6	ABE	ATL



```
wdf=df.groupBy("origin").agg({"destination":"count"})--WIDE TRANSFORMATION-SHUFFLING  
HAPPENS
```

```
▼ wDF: pyspark.sql.dataframe.DataFrame  
  origin: string  
  count(destination): long
```

```
Command took 0.20 seconds -- by deepak.rajak@exusia.com at 17/05/2021, 07:18:37 on My New Cluster
```

REPARTITION AND COALESCE:

Part1 part2 part3 part4

11 12 13 14

Source == 4

Df.repartition(2) - try to evenly distribute the data

New partitions

Part1 = part1(5) + part2(6) + part3(5) + part4(8)

Part2 = part1(6) + part2(6) + part3(8) + part4(6)

Df.coalesce(2) - Even distribution is not guaranteed (1 PARTITION)

Part1 = part1+part3 (15)

Part2 = part2+part4 (30)

df.rdd.getNumPartitions() #to get the number of partitions in Data frame

df.repartition(2) -Distributes the Data evenly

df.coalesce(2) -Even distribution is not guaranteed -so used when you need one partition

from pyspark.sql import Row

from pyspark.sql.functions import col,split

df_row=Row("id","name","city") #header

row1=df_row(1,"VAMSHIDHAR,Reddy,kanamantahreddy","T-Hyderabad")

row2=df_row(2,"SANDEEP,Reddy,E","GA-ATLANTA")

row3=df_row(3,"DHARMEDER,,A","GA-ATLANTA")

row4=df_row(4,"SRAVAN,Reddy,G","TX-FRISCO")

row5=df_row(5,"VARUN,Reddy,kanamantahreddy","KW-KUWAIT")

df_row_seq=[row1,row2,row3,row4,row5]

df=spark.createDataFrame(df_row_seq)

display(df)

```
▶ (3) Spark Jobs
└─ df: pyspark.sql.dataframe.DataFrame
    └─ id: long
        name: string
        city: string
```

	id	name	city
1	1	deepak,kumar,rajak	T-Hyderabad
2	2	virendra,singh,solanki	MP-Bhoapl
3	3	sarvesh,kumar,mishra	MH-Pune
4	4	vasavi,,devi	AP-Vizag
5	5	leena.,rajak	CG-Bilaspur

SPLITTING

#Split name

Splitteddf=(df

```
.withColumn("firstname",split(col("name"),",")[0]
.withColumn("Middlename",split(col("name"),",")[1]
.withColumn("lastname",split(col("name"),",")[2]
.drop(col("name"))
)
```

display(splittdf)

	id	city	firstname	middlename	lastname
1	1	T-Hyderabad	deepak	kumar	rajak
2	2	MP-Bhoapl	virendra	singh	solanki
3	3	MH-Pune	sarvesh	kumar	mishra
4	4	AP-Vizag	vasavi		devi
5	5	CG-Bilaspur	leena		rajak

Showing all 5 rows.

IMPUTING NULL OR MISSING DATA

Imputing Null or Missing Data

```
1 # Creating a dataframe
2 corruptDF = spark.createDataFrame([(11, "Deepak", 5), \
3                                     (12, "Leena", None), \
4                                     (13, "Deepa", 7), \
5                                     (14, None, 9), \
6                                     (15, "Preety", None)], \
7                                     ["id", "name", "bdy"])
8 # Handling NULLs
9 drop_nulls = corruptDF.na.drop() # Dropping NULL Values ( Only 2 Rows will be left)
10
11 drop_nulls.show()
12
13 fill_nulls_with_value = corruptDF.na.fill({"name" : "Unknown", "bdy": 1}) # ( Filling NULL Values)
14
15 fill_nulls_with_value.show() # name we are filling as "Unknown", bday we are filling as "1"
```

```
▶ (6) Spark Jobs
▶ └── corruptDF: pyspark.sql.dataframe.DataFrame = [id: long, name: string ... 1 more fields]
▶ └── drop_nulls: pyspark.sql.dataframe.DataFrame = [id: long, name: string ... 1 more fields]
▶ └── fill_nulls_with_value: pyspark.sql.dataframe.DataFrame = [id: long, name: string ... 1 more fields]
+-----+
| id | name|bdy|
+---+---+---+
| 11|Deepak| 5|
| 13| Deepa| 7|
+---+---+---+ ↗
+-----+
| id| name|bdy|
+---+---+---+
| 11| Deepak| 5|
| 12| Leena| 1|
| 13| Deepa| 7|
| 14|Unknown| 9|
| 15| Preety| 1|
+---+---+---+
```

SPARK TABLES: 4 types

```
1 # Managed Table
2
3 # External Table
4
5 # Temporary View / Table
6
7 # Global Temp View / Table
```

Managed Table - → Spark will control the Data + Metadata (Drop)

External Table - → Spark controls the metadata & user controls the Data (drop)

`%sql` → denotes you are writing SQL code below

Create database sparktraining

The screenshot shows a database management interface. At the top, there is a command-line window labeled "Cmd 9" containing the following SQL commands:

```
1 %sql
2 create database sparktraining
```

Below the command line is a status message "OK".

The main area is titled "Data" and contains three tabs: "Databases", "Tables", and "Permissions".

- Databases Tab:** Shows a list of databases: default, sparkdb, and sparktraining. The "sparkdb" database is currently selected.
- Tables Tab:** Shows a message "No tables".
- Permissions Tab:** Shows a lock icon and the word "Permissions".

A "Create Table" button is located at the top right of the Data section.

For using Database

`%sql`

`use sparkdb`

`df.write.saveAsTable("my_managed_table")`

Creating Data frame from DB

`query=""" select * from my_managed_table """`

`df=spark.sql(query)`

or

`df=sql(query)`

The screenshot shows a Jupyter Notebook cell with the following content:

```
1 query = """ select * from my_managed_table """
2
3 query_df = spark.sql(query)
```

Below the code, the resulting DataFrame is displayed:

```
query_df: pyspark.sql.dataframe.DataFrame
  id: long
  name: string
  city: string
```

`%sql`

```
select * from my_managed_table
```

▶ (3) Spark Jobs

	id	name	city
1	2	virendra,singh,solanki	MP-Bhopal
2	1	deepak,kumar,rajak	T-Hyderabad
3	3	sarvesh,kumar,mishra	MH-Pune
4	5	leena.,rajak	CG-Bilaspur
5	4	vasavi.,devi	AP-Vizag

Showing all 5 rows.

Other way:

```
tabledf=spark.read.table("my_managed_table")
```

or

```
tabledf= table("my_managed_table")
```

```
1 tableDF = spark.read.table("my_managed_table")  
2  
3 |  
  
▼ tableDF: pyspark.sql.dataframe.DataFrame  
  Schema  Details  History  
  
  id: long  
  name: string  
  city: string
```

%sql

```
describe detail my_managed_table
```

```
1 %sql  
2 describe detail my_managed_table
```

format	id	name	description	location	createdAt
delta	3c338774-d572-412e-8799-0489c7baccd8	sparkdb.my_managed_table	null	dbfs:/user/hive/warehouse/sparkdb.db/my_managed_table	2021-05-18T02:06:37.41

Showing all 1 rows.

```
describe extended detail my_managed_table
```

```
1 %sql  
2 describe extended my_managed_table
```

	col_name	data_type	comment
1	id	bigint	
2	name	string	
3	city	string	
4			
5	# Partitioning		
6	Not partitioned		

Showing all 13 rows.

	col_name	data_type	comment
8	# Detailed Table Information		
9	Name	sparkdb.my_managed_table	
10	Location	dbfs:/user/hive/warehouse/sparkdb.db/my_managed_table	
11	Provider	delta	
12	Type	MANAGED	
13	Table Properties	[delta.minReaderVersion=1,delta.minWriterVersion=2]	

Showing all 13 rows.

`spark.catalog.listTables()`

```
1 spark.catalog.listTables()  
  
▶ (1) Spark Jobs  
Out[28]: [Table(name='my_managed_table', database='sparkdb', description=None, tableType='MANAGED', isTemporary=False)]  
Command took 0.64 seconds -- by deepak.rajak@exusia.com at 18/05/2021, 08:02:18 on My New Cluster
```

Writing the data to table-EXTERNAL TABLE

```
df.write.option("path","/tmp/mypath_ext").saveAsTable("MY_EXTERNAL_TABLE")
```

```
1 df.write.option("path", "/tmp/mypath_ext").saveAsTable("MY_EXTERNAL_TABLE")
```

» (4) Spark Jobs

Command took 8.74 seconds -- by deepak.rajkumar@exusia.com at 18/05/2021, 08:05:17 on My New Cluster

md 46

```
1 %sql  
2 describe extended my_external_table
```

	col_name	data_type	comment
1	id	bigint	
2	name	string	
3	city	string	
4			
5	# Partitioning		
6	Not partitioned		

Showing all 13 rows.



Command took 0.26 seconds -- by deepak.rajkumar@exusia.com at 18/05/2021, 08:05:38 on My New Cluster

	col_name	data_type	comment
8	# Detailed Table Information		
9	Name	sparkdb.my_external_table	
10	Location	dbfs:/tmp/mypath_ext	
11	Provider	delta	
12	Type	EXTERNAL	
13	Table Properties	{delta.minReaderVersion=1,delta.minWriterVersion=2}	

Showing all 13 rows.

LIST THE TABLES:

```
1 spark.catalog.listTables()
```

» (2) Spark Jobs

Out[30]: [Table(name='my_external_table', database='sparkdb', description=None, tableType='EXTERNAL', isTemporary=False), Table(name='my_managed_table', database='sparkdb', description=None, tableType='MANAGED', isTemporary=False)]

Command took 0.01 seconds -- by deepak.rajkumar@exusia.com at 18/05/2021, 08:05:41 on My New Cluster

#Temporary View

```
1 df.cre|  
  f createGlobalTempView(name) function  
  f createOrReplaceGlobalTempView(name) function  
Cmd 48  f createOrReplaceTempView(name) function  
  f createTempView(name) function  
1 dbutils.ls(/tmp/mypath_ext/)
```

```
df.createOrReplaceTempView('temptable')
```

```
spark.catalog.listTables()
```

```

1 | spark.catalog.listTables()

> (3) Spark Jobs
Out[32]: [Table(name='my_external_table', database='sparkdb', description=None, tableType='EXTERNAL', isTemporary=False),
Table(name='my_managed_table', database='sparkdb', description=None, tableType='MANAGED', isTemporary=False),
Table(name='temptable', database=None, description=None, tableType='TEMPORARY', isTemporary=True)]

```

%sql

Select * from temptable

```

1 | %sql
2 | select * from temptable

```

> (3) Spark Jobs

	id	name	city
1	1	deepak,kumar.rajak	T-Hyderabad
2	2	virendra,singh,solanki	MP-Bhoapl
3	3	sarvesh,kumar,mishra	MH-Pune
4	4	vasavi,,devi	AP-Vizag
5	5	leena,,rajak	CG-Bilaspur

Showing all 5 rows.

Note that temp table is not available in spark

The screenshot shows the Databricks Data browser. On the left, there's a sidebar with 'Databases' dropdown set to 'sparkdb'. Below it are 'Filter Databases' and two entries: 'default' and 'sparkdb'. On the right, under 'Tables', there's a 'Create Table' button. Below it is a 'Tables' section with a 'Filter Tables' dropdown. The 'my_managed_table' is listed in the dropdown menu.

#Global tables

df.createOrReplaceGlobalTempView("gtemptable")

spark.catalog.listTables()

```

1 | spark.catalog.listTables()

> (3) Spark Jobs
Out[34]: [Table(name='my_external_table', database='sparkdb', description=None, tableType='EXTERNAL', isTemporary=False),
Table(name='my_managed_table', database='sparkdb', description=None, tableType='MANAGED', isTemporary=False),
Table(name='temptable', database=None, description=None, tableType='TEMPORARY', isTemporary=True)]
Command took 0.49 seconds -- by deepak.rajak@exusia.com at 18/05/2021, 08:11:53 on My New Cluster

```

%sql

select * from gtemptable—will give error

```

1 %sql
2 select * from gtemptable

```

④ Error in SQL statement: AnalysisException: Table or view not found: gtemptable; line 1 pos 14;
 'Project [*]
 +- 'UnresolvedRelation [gtemptable], [], false
Command took 0.07 seconds -- by deepak.rajak@exusia.com at 18/05/2021, 08:12:10 on My New Cluster

Note: GLOBAL TEMP TABLES WILL BE STORED UNDER DB global_temp

select * from global_temp.gtemptable

```

1 %sql
2 select * from global_temp.gtemptable

```

▶ (3) Spark Jobs

	id	name	city
1	1	deepak,kumar,rajak	T-Hyderabad
2	2	virendra,singh,solanki	MP-Bhoapl
3	3	sarvesh,kumar,mishra	MH-Pune
4	4	vasavi,,devi	AP-Vizag
5	5	leena,,rajak	CG-Bilaspur

Showing all 5 rows.

```

1 spark.catalog.listTables("global_temp")

```

▶ (2) Spark Jobs

Out[35]: [Table(name='gtemptable', database='global_temp', description=None, tableType='TEMPORARY', isTemporary=True),
Table(name='temptable', database=None, description=None, tableType='TEMPORARY', isTemporary=True)]

Command took 0.36 seconds -- by deepak.rajak@exusia.com at 18/05/2021, 08:13:13 on My New Cluster

Note: we can access global temp tables from any where until cluster is running where as temp tables cannot be accessed in another jupiter book

Configure storage account key:

Spark.conf.set("fs.azure.account.key.<storage-account-name>.dfs.core.windows.net",<storage-account-key>)

Spark.conf.set("fs.azure.createRemoteFileSystemDuringInitialization","true")

Configure storage account key

```

1 spark.conf.set("fs.azure.account.key.<storage-account-name>.dfs.core.windows.net", <storage-account-key>)
2
3 spark.conf.set("fs.azure.createRemoteFileSystemDuringInitialization", "true")
4
5 dbutils.fs.ls("abfss://<file_system>@<storage-account-name>.dfs.core.windows.net/")
6
7 spark.conf.set("fs.azure.createRemoteFileSystemDuringInitialization", "false")
8

```

```

Cmd 26
1 #storage-account-name - storagedatabricks19
2 #storage-account-key - Bm49nF+HVC/NRlTuHbhuiY2y8yh@Db1PCEs7F1EcQRlXL3XK9u7HKA5kBqajZkbjuI0uxgB0fOZ/ojTeaq1Nw==

Cmd 27
1 spark.conf.set("fs.azure.account.key.storagedatabricks19.dfs.core.windows.net",
  "Bm49nF+HVC/NRlTuHbhuiY2y8yh@Db1PCEs7F1EcQRlXL3XK9u7HKA5kBqajZkbjuI0uxgB0fOZ/ojTeaq1Nw==")

Command took 0.44 seconds -- by deepak.rajak@exusia.com at 19/05/2021, 08:06:17 on My New Cluster
Cmd 28
1 dbutils.fs.ls("abfss://first@storagedatabricks19.dfs.core.windows.net/")

Out[2]: [FileInfo(path='abfss://first@storagedatabricks19.dfs.core.windows.net/source/', name='source/', size=0),
 FileInfo(path='abfss://first@storagedatabricks19.dfs.core.windows.net/target/', name='target/', size=0)]

Command took 1.76 seconds -- by deepak.rajak@exusia.com at 19/05/2021, 08:08:18 on My New Cluster

```

Reading file from azure storage

```

1 # Reading from ADLS
2 csvDF = (spark
3   .read
4   .option("sep", "\t")
5   .option("header", True)
6   .option("inferSchema", True)
7   .csv("abfss://first@storagedatabricks19.dfs.core.windows.net/source/pageviews_by_second.tsv")
8 )
9

```

Writing to azure location:

```
) csvDF.write.format("parquet").save("abfss://first@storagedatabricks19.dfs.core.windows.net/target/pageview.parquet")
```

Dr

mounting AWS S3 bucket - databricks1905 to dbfs location - /mnt/deepakS3_databricks19

```

1 ACCESS_KEY = "AKIAZ57C5I33LJZGZQ4F"
2
3 SECRET_KEY = "jEXxA9ni5Pkf3dIk4676bhzhU70FkLzI++qMuVSe"
4
5 ENCODED_SECRET_KEY = SECRET_KEY.replace("/", "%2F")
6
7 AWS_BUCKET_NAME = "databricks1905"
8
9 MOUNT_NAME = "deepak_s3"
10
11
12 # Mounting
13 dbutils.fs.mount(f"s3a://{ACCESS_KEY}:{ENCODED_SECRET_KEY}@{AWS_BUCKET_NAME}", f"/mnt/{MOUNT_NAME}")

```

Transformation

```
1 # test_date -> date ( col("column").cast("date") )
2 # Drop - cumulative_number_of_positives, cumulative_number_of_tests
3 # Add a column - State (Newyork State)
4 # Add a column - load_date ( current_date)
5
6 from pyspark.sql.functions import to_date, lit, current_date
7
8 loadDF = (newYorkCovidTestingDF
9     .withColumn("test_date", to_date("test_date")) # test_date -> date
10    .drop("cumulative_number_of_positives", "cumulative_number_of_tests") # Drop - cumulative_number_of_positives, cumulative_number_of_tests
11    .withColumn("state", lit("Newyork State")) # Add a column - State (Newyork State)
12    .withColumn("load_date", current_date()) # Add a column - load_date ( current_date)
13 )
14
15
16 display(loadDF)
```

Connecting to Database in Cloud server

Request connection string details from Admin

Load to SQL Database

```
1 jdbcUsername = "drajak" # dbuser
2 jdbcPassword = "Infy@123" # dbpass
3
4 driverClass = "com.microsoft.sqlserver.jdbc.SQLServerDriver"
5
6 jdbcHostname = "covid19server20.database.windows.net"
7 jdbcPort = 1433
8 jdbcDatabase = "covid19"
9
10 # Create the JDBC URL without passing in the user and password parameters.
11 jdbcUrl = "jdbc:sqlserver://{}:{};database={}".format(jdbcHostname, jdbcPort, jdbcDatabase)
12
13 # Create a Properties() object to hold the parameters.
14 connectionProperties = {
15     "user" : jdbcUsername,
16     "password" : jdbcPassword,
17     "driver" : driverClass
18 }
19
20 # Load in to Database - By Default single connection / 4
21
22 loadDF.repartition(8).write.jdbc(url=jdbcUrl, table="covid6april", mode="overwrite" , properties=connectionProperties)
23
```

Load to SQL Datatable

LOAD TO SQL Database

```
1 jdbcUsername = "drajak" # dbuser
2 jdbcPassword = "Infy@123" # dbpass
3
4 driverClass = "com.microsoft.sqlserver.jdbc.SQLServerDriver"
5
6 jdbcHostname = "covid19server20.database.windows.net"
7 jdbcPort = 1433
8 jdbcDatabase = "covid19"
9
10 # Create the JDBC URL without passing in the user and password parameters.
11 jdbcUrl = "jdbc:sqlserver://{}:{};database={}".format(jdbcHostname, jdbcPort, jdbcDatabase)
12
13 # Create a Properties() object to hold the parameters.
14 connectionProperties = {
15     "user" : jdbcUsername,
16     "password" : jdbcPassword,
17     "driver" : driverClass
18 }
19
20 # Load in to Database - By Default single connection / 4
21
22 loadDF.write.jdbc(url=jdbcUrl, table="covid19", mode="overwrite" , properties=connectionProperties)
23
24 print("Table loaded successfully!!!\n")
```

Read from SQL Database:

```
df=spark.read.jdbc(url=<jdbcurl>,table="SalesLT.Product", properties=connectionProperties)
```

```
display(df)
```

The screenshot shows the Databricks Data Preview interface. At the top, it displays the schema of the DataFrame:

```
+(1) Spark Jobs
+ productDF: pyspark.sql.dataframe.DataFrame
  + ProductID: integer
  + Name: string
  + ProductNumber: string
  + Color: string
  + StandardCost: decimal(19,4)
  + ListPrice: decimal(19,4)
  + Size: string
  + Weight: decimal(8,2)
  + ProductCategoryID: integer
  + ProductModelID: integer
  + SellStartDate: timestamp
  + SellEndDate: timestamp
  + DiscontinuedDate: timestamp
  + ThumbnailPhoto: binary
  + ThumbnailPhotoFileName: string
  + rowguid: string
  + ModifiedDate: timestamp
```

Below the schema, the table name is shown: "k's Teams Meeting-20210521_070212-Meeting Recording". The main area shows the first two rows of the data:

ProductID	Name	ProductNumber	Color	StandardCost	ListPrice	Size	Weight	ProductCategoryID	ProductModelID
200	HL Road Frame - Black, 56	FR-R02B-56	Black	1022.3700	1431.5000	58	1016.04	18	6
706	HL Road Frame - Red, 58	FR-R02R-58	Red	1059.3100	1431.5000	58	1016.04	18	6

Select only few columns

```
df=spark.read.jdbc(url=<jdbcurl>,table="SalesLT.Product",
properties=connectionProperties).select("ProductID","Name")
```

#Creating Input parameters in Databricks:

```
dbutils.widgets.text("Inputpath","/databricks-datasets/flights","EnterInputPath")
```

```

1 dbutils.widgets.removeAll()

Command took 0.05 seconds -- by deepak.rajk@exusia.com at 12/04/2021, 07:24:34 on My New Cluster

Cmd 2
1 # Creating Input parameters (widgets)
2 # These value will be passed during run time
3
4 dbutils.widgets.text("InputPath", "/databricks-datasets/flights", "EnterInputPath")
5
6 dbutils.widgets.text("InputFile", "departuredelays.csv", "EnterInputFileName")
7
8 dbutils.widgets.text("TargetTable", "emp", "SnowflakeTableName")
9
10 dbutils.widgets.text("TargetPath", "/tmp/target", "EnterOutputPath")
11
12 dbutils.widgets.text("LoadType", "Snowflake", "EnterLoadType")

Command took 0.04 seconds -- by deepak.rajk@exusia.com at 12/04/2021, 07:24:41 on My New Cluster

Cmd 3
Reading Values of Input Parameters

Cmd 4
1 # Getting the param values

```

#Reading Parameters

`Source_File_Path=getArgument("InputPath")`

Reading file format /extracting file format:

Method 1

`import pathlib`

`path=getArgument("InputPath")+"/"+getArgument("InputFile")`

`File_Extension=pathlib.Path(path).suffix`

Method 2

`File_Extension=path.split("/")[-1].split(".")[-1]` -Using python split function

Get the file based on file format:

```

Cmd 7
1 # Function to get the DataFrame based on the File Format
2
3 def getDF(path, File_Extension):
4     if File_Extension == '.csv':
5         inputDF = (spark.read.option("header",True).csv(path))
6     elif File_Extension == '.parquet':
7         inputDF = (spark.read.parquet(path))
8     elif File_Extension == '.json':
9         inputDF = (spark.read.json(path))
10    else:
11        print("File Format Not Supported")
12    return inputDF

```

```

1 # Calling the function and getting the Dataframe
2 df = getDF(path, File_Extension)

```

SNOWFLAKE-SAAS

REDSHIFT-FROM AWS

SYNAPSE-

BIGQuery

Are all MPP-How do they spereate storage and compute

Running Notebook using %run

#Creating and Renaming the columns using Select and Alias

```

df1=(df
      .select("Duration"
              .f.col("Duration").substr(1,2).cast("int").alias("dur_Hours"),
              .f.col("Duration").substr(1,2).cast("int").alias("dur_Minutes"),
              .f.col("Duration").substr(1,2).cast("int").alias("dur_Seconds"))

      .select("Duration"
              (f.col("dur_Hours")*60*60+
               f.col("dur_Minutes")*60+
               f.col("dur_Seconds"))).

```

```

    alias("Duration_in_sec")
)
)

```

Joins:

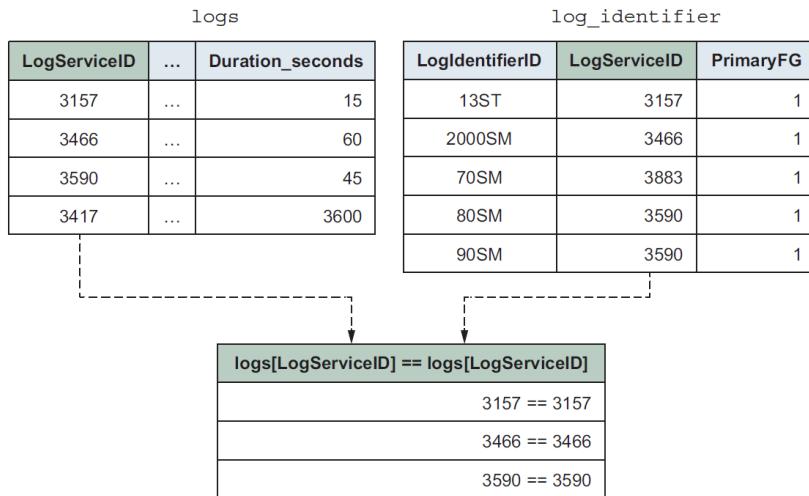
LEFT and RIGTH ARE TWO DIFFERENT DATA FRAMES

```

[LEFT].join(
[RIGHT],
on=[PREDICATES],
how=[METHOD]
)

```

e.g.



```
logs_Channels_Verbose=logs.join(log_identifier,"LogServiceID")
```

#or

```
logs_Channels_Verbose=logs.join(log_identifier,logs["LogServiceID"]==log_identifier["LogServiceID"])
```

Iterating over each Columns and Renaming the

For col in df.columns:

```
df=df.withColumnRenamed(col,col.replace(".", "").replace(" ", "_"))
```

or too change the type

For col in df.columns:

```
df=df.select(F.col(col).cast("int").alias(col))
```

or

For c in df.columns:

```
df=df.withColumn(c,col(c).cast("int"))
```

same as

```
df.columns =[col(c).cast("int").alias(c) for c in df.columns]
```

User Defined Function:

E.G

```
lower=udf(lambda s:s.lower(),
```

```
df=df.select(lower(df["firstname
```

Print columns:

```
df.columns <- gives the list of columns avialable
```

--

When-otherwise

```
F.when(
F.trim(F.col("ProgramClassCD")).isin(
["COM", "PRC", "PGI", "PRO", "PSA", "MAG", "LOC", "SPO", "MER", "SOL"]),
F.col("duration_seconds"),
).otherwise(0)
```

ExplODE AND Arrayzip

```
1 from pyspark.sql.functions import arrays_zip
2 df_arr = spark.createDataFrame([(([1, 2, 3], [2, 3, 4]))], ['vals1', 'vals2'])
3 display(df_arr)
```

▶ (3) Spark Jobs

	vals1	vals2
1	▶ [1, 2, 3]	▶ [2, 3, 4]

For converting the two lists into key pair use array zip

```
1 df_zip=df_arr.select(arrays_zip(F.col("vals1"),F.col("vals2")).alias("zipped"))
2 display(df_zip)
3
```

▶ (3) Spark Jobs

	zipped
1	▶ [{"vals1": 1, "vals2": 2}, {"vals1": 2, "vals2": 3}, {"vals1": 3, "vals2": 4}]

Showing all 1 rows.

For making it into multiple columns:

```
Cmd 10
1 df_final=df_zip.select(F.explode("zipped").alias("EXPLODED"))
2 display(df_final)
```

▶ (3) Spark Jobs

	EXPLODED
1	▶ {"vals1": 1, "vals2": 2}
2	▶ {"vals1": 2, "vals2": 3}

Showing all 3 rows.

*Reading JSON Files In Pyspark

```
shows = spark.read.json("./data/shows/shows-silicon-valley.json")
Reading Multiple Json files:-Make sure to have same schema
three_shows = spark.read.json("./data/shows/shows-*.*.json", multiLine=True)
```

Splitting array columns into multiple columns

Import pyspark.sql.functions as f

Consider below data frame as df

array_subset: DataFrame

schedule	genres
Silicon Valley	Comedy



Split genres column into multiple columns

```
df=df
    .select("schedule",f.col("genres")[0].asalias("genres_1"))
}
```

Or

```
df=df
    .select("schedule",f.col("genres").getItem(0).asalias("genres_1"))
}
```

Or

```
df=df
    .select("schedule",df.genres[0].asalias("genres_1"))
}
Or
df=df
    .select("schedule",df.genres.getItem[0].asalias("genres_1"))
}
```

Note: Do not used slicing operation like `df.genres[1:10]`.

Creating arrays from Columns

```
array_subset_repeated = array_subset  
select(  
  "name",  
  F.array("one", "two", "three").alias("Some_Genres"),  
  F.array_repeat("dot_and_index",  
  5).alias("Repeated_Genres"), # Duplicating the values five times within an array using array_repeat()  
)
```

Computing the size of array:

```
df=(df  
    .select ("name",f.size("Some_Genres"),  
            f.size("Repeated_Genres"))  
)  
o/p:
```

```
array_subset_repeated.select(  
  "name", F.size("Some_Genres"), F.size("Repeated_Genres")  
)  
show()  
  
# +-----+-----+-----+  
# | name|size(Some_Genres)|size(Repeated_Genres)|  
# +-----+-----+-----+  
# |Silicon Valley| 3| 5|  
# +-----+-----+-----+
```

Computing the
number of elements
into both arrays using
the size() function

Removing the Duplicates form Arrays-Using array distinct

```
df=df.select("name",f.array_distinct("Some_Genres"),  
f.array_distinct("Repeated_Genres"))
```

```
array_subset_repeated.select(  
    "name",  
    F.array_distinct("Some_Genres"),  
    F.array_distinct("Repeated_Genres"),  
)  
.show(1, False)
```

Removing duplicates into both arrays with the `array_distinct()` method. Since `Some_Genres` doesn't have any duplicates, the values within the array don't change.

```
# +-----+-----+  
# |name| array_distinct(Some_Genres) |array_distinct(Repeated_Genres)|  
# +-----+-----+  
# |Silicon Valley| [Comedy, Horror, Drama] | [Comedy] |  
# +-----+-----+
```

Finding common/matching values from Arrays-Using array intersect

```
df=df.select("name",f.array_intersect("Some_Genres"  
"Repeated_Genres"))
```

o/p

```
# +-----+-----+  
# | name | Genres |  
# +-----+-----+  
# | Silicon Valley | [Comedy] |  
# +-----+-----+
```

Finding position of an element in an array : using array position

```
df=df.select("Genres",f.array_position("Genres","Comedy"))
```

```
# +-----+-----+  
# | Genres | array_position(Genres, Comedy) |  
# +-----+-----+  
# | [Comedy] | 1 |  
# +-----+-----+
```

Creating Maps form Columns:

```
1 df_items=df_items.select(*df_items.columns)
2 df_items.show()
```

► (3) Spark Jobs

```
+----+-----+----+
| Items|Quantity|Price|
+----+-----+----+
| Banana|      2|  1.74|
| Apple|      4|  2.04|
| Carrot|      1|  1.09|
| Cake|      1|10.99|
+----+-----+----+
```

```
1 df1=df_items.select(f.array("Items","Quantity","Price").alias("values"),f.array([f.lit(c) for c in df_items.columns]).alias("keys")).show(10)
2 # df.show()
```

► (3) Spark Jobs

```
+-----+-----+
|     values|      keys|
+-----+-----+
|[Banana, 2, 1.74][Items, Quantity,...]
|[Apple, 4, 2.04][Items, Quantity,...]
|[Carrot, 1, 1.09][Items, Quantity,...]
|[Cake, 1, 10.99][Items, Quantity,...]
+-----+-----+
df2=df1.select(f.map_from_arrays("keys","values"))
df2.show()
```

```
1 df2=df1.select(f.map_from_arrays("keys","values"))
2 df2.show()
```

► (3) Spark Jobs

```
+-----+
|map_from_arrays(keys, values)|
+-----+
| {Items -> Banana,...}
| {Items -> Apple, ...}
| {Items -> Carrot,...}
| {Items -> Cake, Q...}
+-----+
```

Struct :

```
shows.select("schedule") : DataFrame
```

schedule	
days	time
<div style="border: 1px solid black; width: 50px; height: 50px;"></div>	HH:MM

The schedule column is a struct column, meaning that it contains multiple named fields as a value. We can think of a struct column as one containing a data frame.

RDD N UDFs

```
sc = spark.sparkContext  
  
collection = [1, "two", 3.0, ("four", 4), {"five": 5}]  
  
collection_rdd = sc.parallelize(collection)  
  
print(collection_rdd)  
  
def add_one(value):  
    value=value+1  
  
map:Used for transformation .applies transformation function on entire data  
  
collection_rdd = collection_rdd.map(add_one)  
  
Print(collection_rdd)
```

Filters-Filter will return Truthy Values:

```
collection_rdd = collection_rdd.filter(
```

```

lambda elem: isinstance(elem, (float, int))
)

```

Reduce : Similar to group by And aggaregare in DF

```

from operator import add
collection_rdd = sc.parallelize([4, 7, 9, 1, 3])
print(collection_rdd.reduce(add))

```

UDF Definition

Userdefined Name=udf(**userdefinedfunction,returntype**)

```

reduce_fraction = F.udf(py_reduce_fraction, SparkFrac)

```

Reading Corrupt data:

```

BY DEFAULT READ MODE IS PERMISSIVE

data='""{"a":1,"b":2,"c":3}|{"a":1,"b":2,"c":3}|{"a":1,"b":3}""".split("|")
rdd_data=sc.parallelize(data)
curreptdf=spark.read.json(rdd_data)
display(curreptdf)

```

```

1 data='""{"a":1,"b":2,"c":3}|{"a":1,"b":2,"c":3}| {"a":1,"b":3}"".split("|")
2 rdd_data=sc.parallelize(data)
3 print(rdddf.collect())
4 curreptdf=spark.read.json(rdd_data)
5 display(curreptdf)

```

► (5) Spark Jobs

```

[ '{"a":1,"b":2,"c":3}', '{"a":1,"b":2,"c":3}', '{"a":1,"b":3}']

```

	_corrupt_record	a	b	c
1	null	1	2	3
2	null	1	2	3
3	{"a":1,"b":3}	null	null	null

Types of reading files

DROPMALFORMED:

```

1 data=''''{"a":1,"b":2,"c":3}|{"a":1,"b":2,"c":3}|{"a":1,"b":2,"c":3}'''.split("|")
2 rdd_data=sc.parallelize(data)
3 currentDF=spark.read.option("mode","DROPMALFORMED").json(rdd_data)
4 display(currentDF)

```

▶ (4) Spark Jobs

	a	b	c
1	1	2	3
2	1	2	3

FAILFAST

```

1 data=''''{"a":1,"b":2,"c":3}|{"a":1,"b":2,"c":3}|{"a":1,"b":2,"c":3}'''.split("|")
2 rdd_data=sc.parallelize(data)
3 currentDF=spark.read.option("mode","FAILFAST").json(rdd_data)
4 display(currentDF)

```

▶ (1) Spark Jobs

Borg.apache.spark.SparkException: Job aborted due to stage failure: Task 7 in stage 20.0 failed 1 times, most recent failure: Lost task 7.0 in stage 20.0 (TID 103) (ip-10-172-166-243.us-west-2.compute.internal executor driver): org.apache.spark.SparkException: Malformed records are detected in schema inference. Parse Mode: FAILFAST.

Command took 0.85 seconds -- by vkanaparthireddy10@tu-edu at 3/20/2022 9:31:07 AM on Doms

Read Streaming data:

```

1 # Define the schema using a DDL-formatted string.
2 dataSchema = "Recorded_At timestamp, Device string, Index long, Model string, User string, _corrupt_record String, gt string, x double, y double, z double"

```

Command took 0.04 seconds -- by deepak.rajak@xusia.com at 13/04/2021, 07:33:01 on Exusia

d 8

```

1 # Near Real time ( A little lag between receiving the data & processing the data )
2 # continuous stream of data == microbatches1 + microbatch2 + 3 + 4..... .
3 # processing rate > input rate
4 # Why inferSchema cant be used ?

```

6 9

```

1 dataPath = "dbfs:/mnt/training/definitive-guide/data/activity-data-stream.json"
2
3 initialDF = (spark
4   .readStream
5   .option("maxFilesPerTrigger", 1)      # Returns DataStreamReader
6   .schema(dataSchema)                 # Force processing of only 1 file per trigger
7   .json(dataPath)                    # Required for all streaming DataFrames
8   )

```

Streaming Query:

Code 18

```
1 streamingQuery = (streamingDF          # Start with our "streaming" DataFrame
2   .writeStream           # Get the DataStreamWriter
3   .queryName(myStreamName) # Name the query
4   .trigger(processingTime="3 seconds") # Configure for a 3-second micro-batch
5   .format("parquet")        # Specify the sink type, a Parquet file
6   .option("checkpointLocation", checkpointPath) # Specify the location of checkpoint files & W-A logs
7   .outputMode("append")     # Write only new data to the "file"
8   .start(outputPathDir)    # Start the job, writing to the specified directory
9 )
```

Reading from KAFKA

Cmd 3

```
1 from pyspark.sql.functions import col
2 spark.conf.set("spark.sql.shuffle.partitions", sc.defaultParallelism)
3
4 kafkaServer = "server1.databricks.training:9092" # US (Oregon)
5 # kafkaServer = "server2.databricks.training:9092" # Singapore
6
7 editsDF = (spark.readStream
8   .format("kafka")
9   .option("kafka.bootstrap.servers", kafkaServer)
10  .option("subscribe", "en")
11  .option("startingOffsets", "earliest")
12  .option("maxOffsetsPerTrigger", 1000)
13  .load())
14  .select(col("value").cast("STRING"))
15 )
```

Cmd 5

```
1 myStreamName = "myStream"
2 display(editsDF, streamName = myStreamName)
```

(1) Spark Jobs

⌚ myStream (id: 0a9c007b-bd0d-45cd-8b35-b142df766a2) Last updated: 10 seconds ago

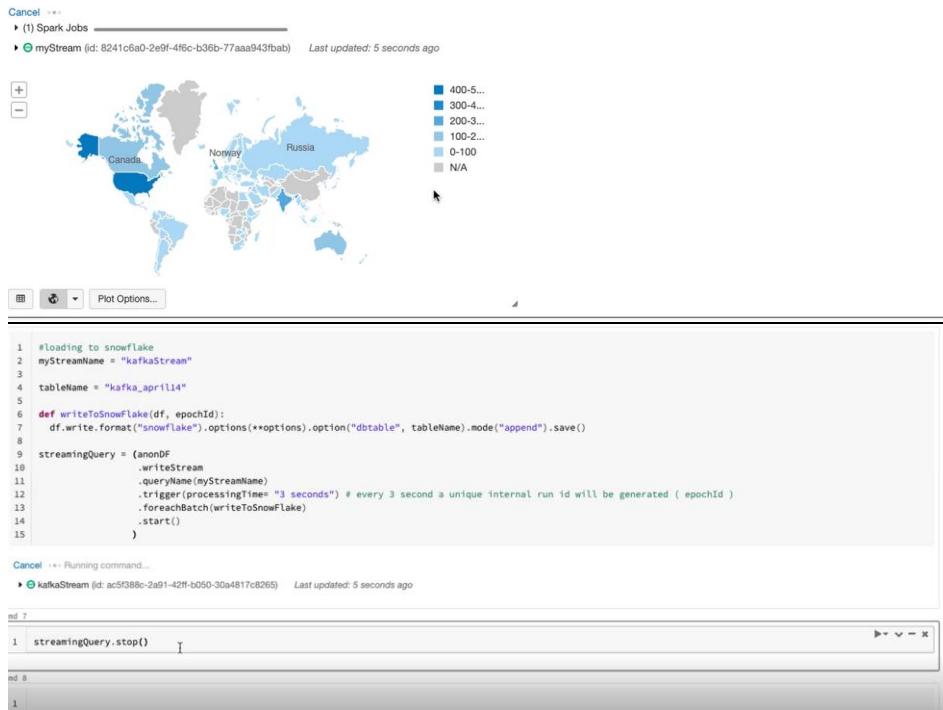
value

```
{"isRobot":false,"channel":"en.wikipedia","timestamp":"2021-04-11T06:47:35.337Z","url":"","isIngested":false,"page":"Special:Log/abusefilter","wikipedia":"en","wikipediaURL":"http://en.wikipedia.org","comment":"2409-4072:83:7C77:A3AD:3D64:81F9:BFE3 triggered [[Special:AbuseFilter/1113|filter 1113]], performing the action 'edit' on [[u003002|User:Brahminiu003010]]. Actions taken: none"}, [{"@special:abuseLog/29004069[details]"}, {"userURL":"http://en.wikipedia.org/wiki/User:2409-4072:83:7C77:A3AD:3D64:81F9:BFE3","pageURL":"http://en.wikipedia.org/wiki/Special:Log/abusefilter","delta":null,"flag":true,"isNewPage":false,"isAnonymous":true,"geocoding":true,"countryCode":null,"city":null,"latitude":null,"longitude":null,"stateProvince":null,"countryCode3":null,"user":2409-4072:83:7C77:A3AD:3D64:81F9:BFE3,"namespace":special"}], [{"@special:abuseLog/29004069[details]"}, {"userURL":"http://en.wikipedia.org/wiki/User:2409-4072:83:7C77:A3AD:3D64:81F9:BFE3","pageURL":"http://en.wikipedia.org/wiki/Special:Log/abusefilter","delta":null,"flag":true,"isNewPage":false,"isAnonymous":true,"geocoding":true,"countryCode":null,"city":null,"latitude":null,"longitude":null,"stateProvince":null,"countryCode3":null,"user":2409-4072:83:7C77:A3AD:3D64:81F9:BFE3,"namespace":special}]]
```

Live streaming

myStreamName = "myStream"

```
1 myStreamName = "myStream"
2
3
4 mappedDF = (anonDF
5   .groupby("geocoding.countryCode3") # Aggregate by country (code)
6   .count() # Produce a count of each aggregate
7 )
8
9 display(mappedDF, streamName = myStreamName)
```



Creating Delta Lake

SAVE THE DATA as delta file:

Create a delta lake table

Rad for the table as as when needed.

Any data that come in to the file location needs to be appended so that they automatically reflect in delta lake table

Cmd 3

```
1 dbutils.fs.ls("/tmp/")
```

Cmd 4

```
1 dbutils.fs.rm("/tmp/", True)
```

Cmd 5

```
1 inputPath = "/mnt/training/online_retail/data-001/data.csv"
2
3 parquetDataPath = "/tmp" + "/customer-data/"
4
5 deltaDataPath = "/tmp" + "/customer-data-delta/"
```

Cmd 6

```
1 inputSchema = "InvoiceNo STRING, StockCode STRING, Description STRING, Quantity INT, InvoiceDate STRING, UnitPrice DOUBLE, CustomerID INT, Country STRING"
2
3 rawDF = (spark.read
4   .option("header", "true")
5   .schema(inputSchema)
6   .csv(inputPath)
7 )
```

Cancel Running command... 

```
1 # write using Databricks Delta format
2 (rawDF.write
3   .mode("overwrite")
4   .format("delta")
5   .partitionBy("Country")
6   .save(deltaDataPath) )
```

Cancel Running command... 

```

My New Cluster | File | View | Edit | Permissions | Run All | Clear |
3 (spark.read
4   .option("header", "true")
5   .schema(inputSchema)
6   .csv(inputPath)
7   .write
8   .mode("overwrite")
9   .format("delta")
10  .partitionBy("Country")
11  .save(deltaDataPath) )

```

▶ (4) Spark Jobs

Command took 14.76 seconds -- by deepak.rajk@exusia.com at 15/04/2021, 07:40:16 on My New Cluster

Cmd 28

```

1 %sql
2 drop table customer_delta_data

```

⊕ Error in SQL statement: AnalysisException: Table or view not found: customer_delta_data;

Command took 7.33 seconds -- by deepak.rajk@exusia.com at 15/04/2021, 07:40:27 on My New Cluster

Cmd 29

```

1 spark.sql("""
2   CREATE TABLE IF NOT EXISTS customer_delta_data
3     USING delta
4     OPTIONS (path = '{}')
5 """.format(deltaDataPath))

```

SCD : Slowly Changing Dimensions

180
Day Deleted 3

Day 1 = 100 records
7:50 AM Day 2 = 12...

Delta Lake ||| Not...
Yesterday Delta / del...

Output mode -
Tuesday Overwrite

Technical demons...
Monday 1.Azure Dat...

10 Files - S3/Azure...
Monday 3 Parquet

dapi37742f03cd5...
Friday No additional...

15 April 2021 at 7:50 AM

Day 1 = 100 records

Day 2 = 12 records (3 records came as update + 9 are new records - inserts)

=====> 109 (SCD Type 1 Table)

Slowly changing dimension

Empid	dep	address	phone			
1	HR	NJ	201	N	2018 - 2020	
1	HR	NJ	301	Y	2020 - NULL	

Cmd 7

Day1 - Load

```

1 (miniDataDF
2   .write
3   .mode("overwrite")
4   .format("delta")
5   .save(deltaMinDataPath)
6 )
7
8 spark.sql("""
9   CREATE TABLE IF NOT EXISTS customer_data_delta_mini
10  USING DELTA
11  LOCATION '{}'
12 """.format(deltaMinDataPath))

```

▶ (4) Spark Jobs
Out [3]: DataFrame[]
Command took 11.93 seconds -- by deepak.rajk@exusia.com at 18/03/2021, 07:56:38 on Exusia

```

1 sqlCmd = "SELECT * FROM customer_data_delta_mini WHERE CustomerID=20993"
2
3 display(spark.sql(sqlCmd))

```

▶ (1) Spark Jobs

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
536371	32129	EverGlow Single	228	1/1/18 9:01	33.85	20993	Sierra Leone

Showing all 1 rows.

Cmd 11

```

1 %sql drop table customer_data_delta_to_upsert

```

② Error in SQL statement: AnalysisException: Table or view not found: customer_data_delta_to_upsert;
Command took 3.66 seconds -- by deepak.rajk@exusia.com at 15/04/2021, 07:54:15 on My New Cluster

Cmd 12

```

1 from pyspark.sql.functions import lit, col
2
3 customerSpecificDF = (miniDataDF
4   .filter("CustomerID=20993")
5   .withColumn("StockCode", lit(99999))
6 )
7
8 spark.sql("DROP TABLE IF EXISTS customer_data_delta_to_upsert")
9
10 customerSpecificDF.write.saveAsTable("customer_data_delta_to_upsert")
11

```

Cmd 14

Day 2

```

1 %sql
2 select * from customer_data_delta_to_upsert
  
```

► (1) Spark Jobs

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1 536371	99999	EverGlow Single	228	1/1/18 9:01	33.85	20993	Sierra Leone

Showing all 1 rows.

Command took 0.91 seconds -- by deepak.rajk@exusia.com at 15/04/2021, 07:55:40 on My New Cluster

Cmd 15

```

1 %sql MERGE INTO customer_data_delta_mini USING customer_data_delta_to_upsert ON customer_data_delta_mini.CustomerID = customer_data_delta_to_upsert.CustomerID
2 WHEN MATCHED THEN
3 UPDATE
4   SET
5     customer_data_delta_mini.StockCode = customer_data_delta_to_upsert.StockCode
6   WHEN NOT MATCHED THEN
7     INSERT
8     (
9       InvoiceNo,
10      StockCode,
11      Description,
12      Quantity,
13      InvoiceDate,
14      UnitPrice,
15      CustomerID,
16      Country
17    )
18   VALUES
19   (
20     customer_data_delta_to_upsert.InvoiceNo,
21     customer_data_delta_to_upsert.StockCode,
22     customer_data_delta_to_upsert.Description,
23     customer_data_delta_to_upsert.Quantity,
24     customer_data_delta_to_upsert.InvoiceDate,
25     customer_data_delta_to_upsert.UnitPrice,
26     customer_data_delta_to_upsert.CustomerID,
27     customer_data_delta_to_upsert.Country
28   )
  
```

Cmd 18

```

1 # MERGE Operation
2 spark.sql(""" MERGE INTO del_delay as d
3   USING merge_table as m
4   on d.origin = m.origin
5   WHEN MATCHED THEN
6     UPDATE SET | |
7   WHEN NOT MATCHED
8     THEN INSERT * """)
  
```

Using Python code to merge:

```

1 # Create the 5 records ( We will be updating the column - name)
2 newIncrementalData = spark.range(5).withColumn("name", lit("Neha"))
3
4 # Create DeltaTable instances using the path of the Delta table
5 deltaTable = DeltaTable.forPath(spark, "/tmp/ids")
6
7 # Execute Update + Insert ( Upsert = merge )
8 (deltaTable
9   .alias("oldData")
10  .merge(newIncrementalData.alias("newData"), "oldData.id = newData.id")
11  .whenMatchedUpdate(set = { "name": col("newData.name") })
12  .whenNotMatchedInsert(values = { "id": col("newData.id") , "name": col("newData.name") })
13  .execute()
14 )  

15
16 # Display the records to check if the records are Merged
17
18
19 # Display the records to check if the records are Merged

```

sqlCmd = "SELECT * FROM customer_data_delta_mini WHERE CustomerID=20993"
display(spark.sql(sqlCmd))

(1) Spark Jobs

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
536371	99999	EverGlow Single	228	1/1/18 9:01	33.85	20993	Sierra Leone

Showing all 1 rows.

Command took 3.15 seconds -- by deepak.rajk@exusia.com at 15/04/2021, 08:03:09 on My New Cluster

Merging schema:

Day 1 with 5 columns

```

9 df.write.format("delta").save(delta_path)

10 (5) Spark Jobs
11 df: pyspark.sql.DataFrame = [date: string, delay: string ... 3 more fields]
12 Command took 47.29 seconds -- by deepak.rajk@exusia.com at 16/04/2021, 07:12:13 on My New Cluster
13 Cmd 5

1 # Day1
2 delta_path = "/tmp/departure"
3
4 data_file = "/databricks-datasets/flights/departuredelays.csv"
5
6 spark.read.format("delta").load(delta_path).show(1) #We already have DELTA LAKE file
7

(1) Spark Jobs
+-----+-----+-----+
| date|delay|distance|origin|destination|
+-----+-----+-----+
|01011245| 6| 602| ABE| ATL|
+-----+-----+-----+
only showing top 1 row

Command took 6.34 seconds -- by deepak.rajk@exusia.com at 16/04/2021, 07:12:35 on My New Cluster
14 Cmd 6

1 %sql
2 select count(*) from delta.`/tmp/departure`  

3
4 Cancel Command submitted to cancel the execution

```

Day2 if the data comes in with additional column :

```

1 # Day 2
2
3 from pyspark.sql.functions import lit
4
5 newDF = spark.read.format("delta").load(delta_path).limit(5).withColumn("country", lit("USA"))
6
7 display(newDF)

▶ (1) Spark Jobs
▼ newDF: pyspark.sql.DataFrame
  date: string
  delay: string
  distance: string
  origin: string
  destination: string
  country: string

```

	date	delay	distance	origin	destination	country
1	01011245	6	602	ABE	ATL	USA
2	01020600	-8	369	ABE	DTW	USA
3	01021245	-2	602	ABE	ATL	USA
4	01020605	-4	602	ABE	ATL	USA
5	01031245	-4	602	ABE	ATL	USA

Showing all 5 rows.

Error occurred as data mismatched:

```

> Cnd 8
1 # Append
2 newDF.write.format("delta").mode("append").save(delta_path)

AnalysisException: A schema mismatch detected when writing to the Delta table (Table ID: 6ad87511-49d9-4dac-9394-e7f5a5ad7d8f).
  1 # Append
  ----> 2 newDF.write.format("delta").mode("append").save(delta_path)

  /databricks/spark/python/pyspark/sql/readwriter.py in save(self, path, format, mode, partitionBy, **options)
    828         self._jwrite.save()
    829     else:
--> 830         self._jwrite.save(path)
    831
    832     @since(1,4)

  /databricks/spark/python/lib/py4j-0.10.9-src.zip/py4j/java_gateway.py in __call__(self, *args)
    1303     answer = self.gateway_client.send_command(command)
    1304     return_value = get_return_value(
-> 1305         answer, self.gateway_client, self.target_id, self.name)
    1306
    1307     for temp_arg in temp_args:

  /databricks/spark/python/pyspark/sql/utils.py in deco(*a, **kw)
    131         # Hide where the exception came from that shows a non-Pythonic
    132         # JVM exception message.
--> 133         raise_fromconverted()


```

To avoid this use option("mergeSchema", "True")

```

Cnd 9
1 newDF.write.format("delta").option("mergeSchema", True).mode("append").save(delta_path)

▶ (4) Spark Jobs
Command took 7.11 seconds --- by deepak.rajk@exusia.com at 16/04/2021, 07:21:13 on My New Cluster

```

History table-Every delta table has an history table associated.

```

Command took 4.23 seconds --- by deepak.rajk@exusia.com at 16/04/2021, 07:24:04 on My New Cluster
Cnd 16
1 !sql
2 describe history delta./tmp/departure


```

```

1 %sql
2 describe history delta.`/tmp/departure`

▶ (1) Spark Jobs


| version | timestamp                    | userId           | userName                | operation | operationParameters                      | job  | notebook                |
|---------|------------------------------|------------------|-------------------------|-----------|------------------------------------------|------|-------------------------|
| 0       | 2021-04-16T01:53:54.000+0000 | 6370931699180690 | deepak.rajak@exusia.com | WRITE     | {"mode": "Overwrite", "partitionBy": ""} | null | {"notebookId": "17889"} |


Showing all 1 rows.

```

As the transaction increases , the version increases:

```

1 %sql
2 describe history delta.`/tmp/departure`

▶ (1) Spark Jobs


| version | timestamp                    | userId           | userName                | operation | operationParameters                                                         | job  | notebook |
|---------|------------------------------|------------------|-------------------------|-----------|-----------------------------------------------------------------------------|------|----------|
| 2       | 2021-04-16T01:58:29.000+0000 | 6370931699180690 | deepak.rajak@exusia.com | WRITE     | {"mode": "Append", "partitionBy": "[]"}<br>null<br>{"notebookId": "17889"}  | null |          |
| 1       | 2021-04-16T01:57:04.000+0000 | 6370931699180690 | deepak.rajak@exusia.com | WRITE     | {"mode": "Append", "partitionBy": "[]"}<br>null<br>{"notebookId": "17889"}  | null |          |
| 0       | 2021-04-16T01:53:54.000+0000 | 6370931699180690 | deepak.rajak@exusia.com | WRITE     | {"mode": "Overwrite", "partitionBy": ""}<br>null<br>{"notebookId": "17889"} | null |          |


Showing all 3 rows.

```

Get the data of required version using option("versionAsof:<versionnumber>) this includes the data as of that version

```
cmd 11
1 # delta lake ( 1391578 ) + ( 1391578 ) + ( 1391578 )
2 df.write.mode("append").format("delta").save("/tmp/departure")
▶ ▶ ▶

▶ (4) Spark Jobs
Command took 13.29 seconds -- by deepak.rajk@exusia.com at 16/04/2021, 07:28:19 on My New Cluster
Cmd 18
1 # To get the FIRST verson
2
3 df_version0 = spark.read.format("delta").option("versionAsOf", 0).load("/tmp/departure") #versionAsOf = 0 / timestampASOF
    ▾
▶ ▾ df_version0: pyspark.sql.dataframe.DataFrame = [date: integer, delay: integer ... 3 more fields]
Command took 0.78 seconds -- by deepak.rajk@exusia.com at 16/04/2021, 07:31:13 on My New Cluster
Cmd 19
```

If we are using parquet file instead of Delta lake,

```
Cmd 25
1 %%sql
2 msck repair table customer_data
▶ ▶ ▶

▶ (1) Spark Jobs
OK
Command took 5.44 seconds -- by deepak.rajk@exusia.com at 15/04/2021, 07:26:19 on My New Cluster
```

Write data to Snow flake:

```

5 # snowflake connection options
6 options = dict(sfUrl="https://exusiaipartner.us-east-1.snowflakecomputing.com",
7   sfUser=user,
8   sfPassword=password,
9   sfDatabase="TEST_DB",
10  sfSchema="PUBLIC",
11  sfWarehouse="LOAD_WH")
12

13 loadType = getArgument("LoadType")
14
15 if loadType == "Snowflake":
16     tableName = getArgument("TargetTable")
17     df.write.format("snowflake").options(**options).option("dbtable", tableName).mode("overwrite").save()
18     print(tableName, "table loaded successfully in Snowflake!!!")
19
20 elif loadType == "File":
21     targetPath = getArgument("TargetPath") + "/out_" + getArgument("InputFile")
22     # write the dataset to DBFS
23     df.write.mode("overwrite").parquet(targetPath)
24     print("file loaded successfully in DBFS!!!")
25
26 else:
27     print("load not supported")

```

The screenshot shows the Snowflake Web UI interface. At the top, there's a navigation bar with links like 'News', 'Latest News', 'Breaking News', 'News Headlines', 'Live News', 'Today News', 'Chin-News/English'. Below the navigation is a search bar and a 'Warehouses' tab selected. On the left, there's a sidebar with 'Create', 'Continue', 'Resume', and 'Run' buttons. The main area has two sections: 'Create Warehouse' on the left and a list of existing warehouses on the right.

Create Warehouse Dialog:

- Name:** [Input field]
- Size:** Medium (4 credits / hour) [Select dropdown]
- Maximum Clusters:** 10 [Input field]
- Minimum Clusters:** 1 [Input field]
- Scaling Policy:** Standard [Select dropdown]
- Auto Suspend:** 10 minutes [Input field]
- Comment:** [Input field]
- Show SQL:** [Button] [Cancel] [Finish]

Warehouses List:

ID	Resumed On	Owner	Comments
1	4/8/2021, 9:28:55 P...	SYSADMIN	Test WH
2	3/3/2021, 5:47:23 P...	SYSADMIN	
3	2/26/2021, 12:37:51...	SYSADMIN	
4	3/2/2021, 10:47:14 A...	SYSADMIN	Test wh
5	3/2/2021, 9:53:41...	SYSADMIN	
6	2/13/2021, 4:59:34...	SYSADMIN	
7	2/27/2021, 11:45:35...	SYSADMIN	
8	2/28/2021, 4:07:32...	SYSADMIN	
9	2/28/2021, 7:24:19...	SYSADMIN	
10	3/8/2021, 9:42:52 P...	SYSADMIN	Small Ter
11	2/16/2021, 9:03:19...	SYSADMIN	Warehouse
12	3/4/2021, 10:11:35...	SYSADMIN	
13	2/24/2021, 3:36:38...	SYSADMIN	Small Ter
14	3/8/2021, 4:12:11 PM...	SYSADMIN	
15	2/25/2021, 5:32:22...	SYSADMIN	Small Ter

The image shows two side-by-side screenshots of the Snowflake interface, specifically the 'Workbooks' tab. Both windows have the title 'Databricks - Databricks_conformation'. The left window shows a query editor with the following SQL code:

```

1 select * from "CITIBIKE","PUBLIC","TRIPS_2"
2 truncate table "CITIBIKE","PUBLIC","TRIP_2"
3
4 select * from "CITIBIKE","PUBLIC","HEALTH"
5
6 select * from "DEPARTUREDELAYS","PUBLIC","DEPARTUREDELAYS_COPY"
7
8 select * from "DEVELOPMENT","PUBLIC","ORDERS_COPY"
9 select * from "DEVELOPMENT","PUBLIC","ORDERS_AWS"
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30

```

The right window shows a similar query editor with the same SQL code. Both windows show a sidebar with database objects like SOCIAL MEDIA, FLUGOOGATES, EMON, DEPARTUREDELAYS, and EMP. The bottom status bar indicates 'Mon 12 Apr 7:48 AM'.

```

columns = train_data.columns
stages = []
for i in train_data.dtypes:
    if(i[1]=='string'):
        columns.remove(i[0])
        stages.append(StringIndexer(inputCol=i[0], outputCol=i[0]+"_IX"))

```

Auto load:

```
=====
checkpoint_path = '/tmp/delta/population_data/_checkpoints'
write_path = '/tmp/delta/population_data'

# Set up the stream to begin reading incoming files from the
# upload_path location.
df = spark.readStream.format('cloudFiles') \
    .option('cloudFiles.format', 'csv') \
    .option('header', 'true') \
    .schema('city string, year int, population long') \
    .load(upload_path)

# Start the stream.
# Use the checkpoint_path location to keep a record of all files that
# have already been uploaded to the upload_path location.
# For those that have been uploaded since the last check,
# write the newly-uploaded files' data to the write_path location.
df.writeStream.format('delta') \
    .option('checkpointLocation', checkpoint_path) \
    .start(write_path)
$
```

SnowFlake CONNECTION:

```
options=dict(sfurl="https://sg39612.us-east-2.aws.snowflakecomputing.com",
            sfUser="kvr",
            sfPassword="Sravanthi@1",
            sfDatabase="TEST",
            sfSchema="TEST")
```

Writing to Table:

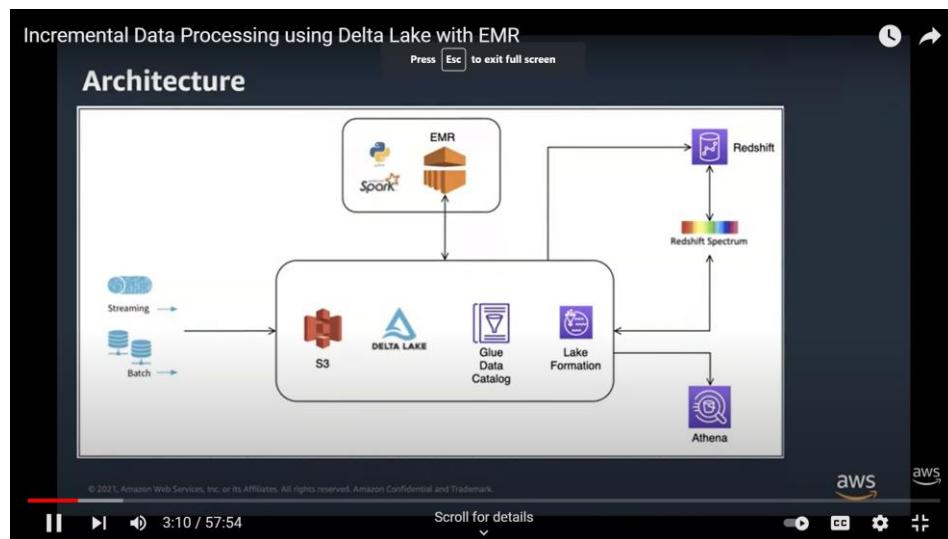
```
df.write.format("snowflake").options(**options).option("dbtable","sparktable").mode("overwrite").save()
```

Reading from Table

```
df1=spark.read.format("snowflake").options(**options).option("dbtable","sparktable").load()
```

DELTA Lake using EMR

[Ref: \(60\) Incremental Data Processing using Delta Lake with EMR - YouTube](#)



Configuration for Delta Lake:

Incremental Data Processing Using Delta Lake with EMR

```
In [1]: from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Perform Delta Demo").getOrCreate()

Starting Spark application
ID          YARN Application ID  Kind  State  Spark UI  Driver log  Current session?
18  application_1615402599665.0020  pyspark  idle  Link      ✓
SparkSession available as 'spark'.
```

Adding Delta Jar package and configurations to the notebook

```
In [2]: !configure -f
{
  "conf": {
    "spark.jars.packages": "io.delta:delta-core_2.12:0.8.0",
    "spark.sql.extensions": "io.delta.sql.DeltaSparkSessionExtension",
    "spark.sql.catalog.spark_catalog": "org.apache.spark.sql.delta.catalog.DeltaCatalog"
  }
}
Starting Spark application
ID          YARN Application ID  Kind  State  Spark UI  Driver log  Current session?
19  application_1615402599665.0021  pyspark  idle  Link      ✓
SparkSession available as 'spark'.
Current session config: {'conf': {'spark.jars.packages': 'io.delta:delta-core_2.12:0.8.0', 'spark.sql.extensions': 'io.delta.sql.DeltaSparkSessionExtension', 'spark.sql.catalog.spark_catalog': 'org.apache.spark.sql.delta.catalog.DeltaCatalog'}}
```

Incremental Data Processing using Delta Lake with EMR

```
In [1]: from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Perform Delta Demo").getOrCreate()

Starting Spark application
ID          YARN Application ID  Kind  State  Spark UI  Driver log  Current session?
19  application_1615402599665.0021  pyspark  idle  Link      ✓
SparkSession available as 'spark'.
```

```
In [2]: !configure -f
{
  "conf": {
    "spark.jars.packages": "io.delta:delta-core_2.12:0.8.0",
    "spark.sql.extensions": "io.delta.sql.DeltaSparkSessionExtension",
    "spark.sql.catalog.spark_catalog": "org.apache.spark.sql.delta.catalog.DeltaCatalog"
  }
}
Starting Spark application
ID          YARN Application ID  Kind  State  Spark UI  Driver log  Current session?
19  application_1615402599665.0021  pyspark  idle  Link      ✓
SparkSession available as 'spark'.
Current session config: {'conf': {'spark.jars.packages': 'io.delta:delta-core_2.12:0.8.0', 'spark.sql.extensions': 'io.delta.sql.DeltaSparkSessionExtension', 'spark.sql.catalog.spark_catalog': 'org.apache.spark.sql.delta.catalog.DeltaCatalog'}}
```

Defining Methods to generate Json Data

```
In [4]: from datetime import datetime
def get_json_data(timestamp, count, dest):
    time_stamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    data = [{"trip_id": i, "timestamp": time_stamp, "route_id": chr(65 + (i * 10)), "destination": dest[i%10]} for i in range(count)]
    return data

# Creates the DataFrame
def create_json_df(spark, data):
    sc = spark.sparkContext
    return spark.read.json(sc.parallelize(data))
```

Creating Spark dataframe from trip data

To view tables:

```
Creating a Spark table in Glue Data Catalog with same data

In [10]: deltaTable = "demo_delta_db" + "." + "delta_spark_table"

In [11]: df1.write.format("delta").partitionBy("route_id").mode("overwrite").option("path", deltaSparkPath).saveAsTable(deltaTable)

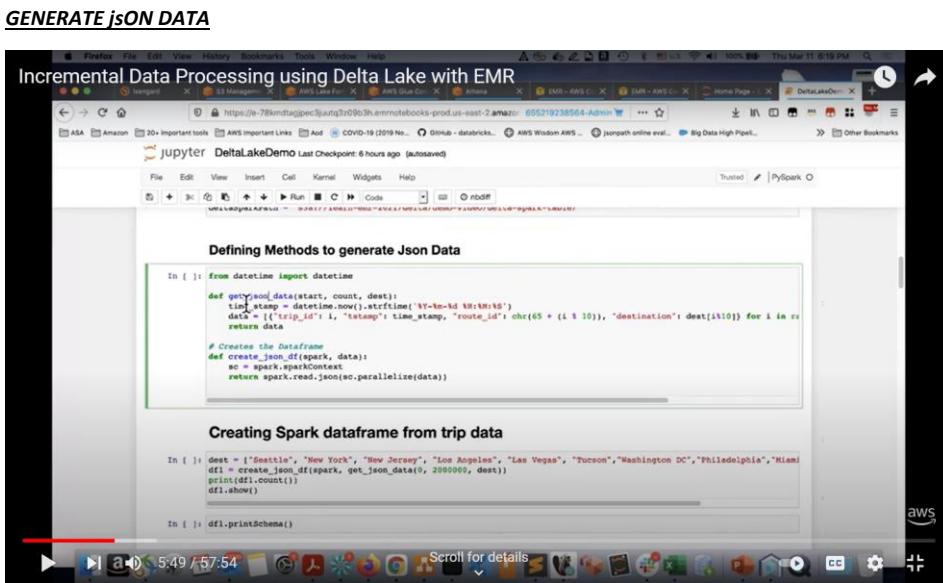
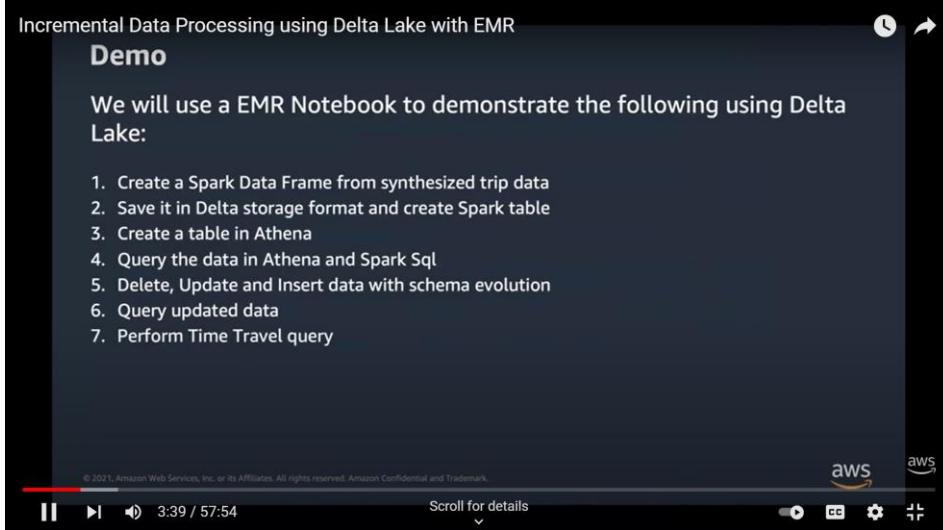
In [12]: spark.sql("use demo_delta_db")
spark.sql("show tables").show(20)

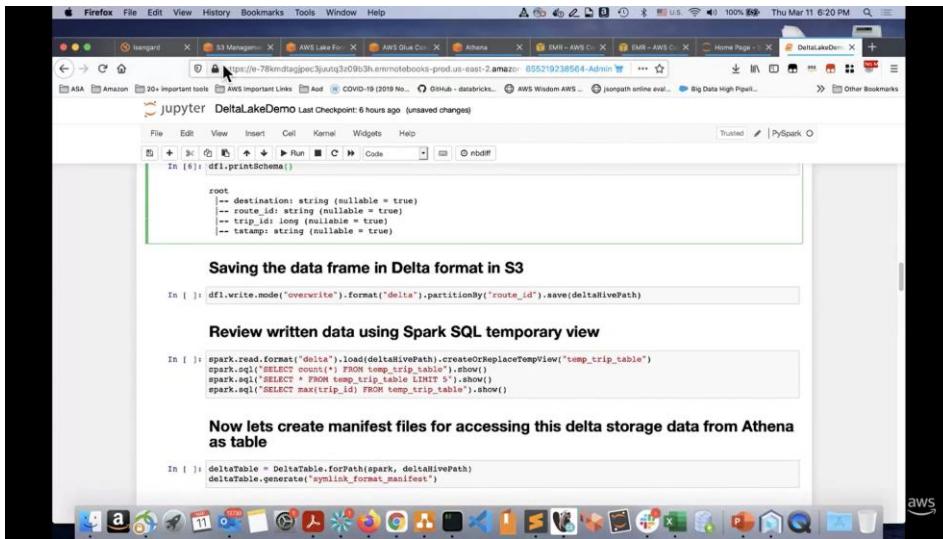
+-----+-----+
| database | tableName |
+-----+-----+
| demo_delta_db | delta_spark_table | false |
| demo_delta_db | temp_trip_table | true |
+-----+-----+


In [13]: spark.sql("show create table delta_spark_table").show(200,truncate=False)

In [14]: %%spark -c spark -o delta_trip_op
delta_trip_op = spark.sql("select count(*) from demo_delta_db.delta_spark_table")
delta_trip_op.show()
```

Inserting Data to Delta Lake:





```
In [1]: df1.printSchema()
```

```
root
 |-- destination: string (nullable = true)
 |-- route_id: string (nullable = true)
 |-- trip_id: long (nullable = true)
 |-- timestamp: string (nullable = true)
```

```
Saving the data frame in Delta format in S3
```

```
In [ ]: df1.write.mode("overwrite").format("delta").partitionBy("route_id").save(deltaHivePath)
```

```
Review written data using Spark SQL temporary view
```

```
In [ ]: spark.read.format("delta").load(deltaHivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT count(*) FROM temp_trip_table").show()
spark.sql("SELECT * FROM temp_trip_table LIMIT 5").show()
spark.sql("SELECT max(trip_id) FROM temp_trip_table").show()
```

```
Now lets create manifest files for accessing this delta storage data from Athena as table
```

```
In [ ]: deltaTable = DeltaTable.forPath(spark, deltaHivePath)
deltaTable.generate("symlink_format_manifest")
```

Data saved in s3

The image shows two side-by-side screenshots of a Firefox browser window. Both windows are displaying the AWS S3 Management console.

Top Window: The URL is <https://s3.console.aws.amazon.com/s3/buckets/learn-emr-2021?region=us-east-1>. The page title is "Objects (11)". The table lists 11 objects under the "route_id" folder, all of which are sub-folders named "route_id-A", "route_id-B", "route_id-C", "route_id-D", "route_id-E", "route_id-F", "route_id-G", "route_id-H", "route_id-I", and "route_id-J".

Name	Type	Last modified	Size	Storage class
_delta_log/	Folder	-	-	-
route_id-A/	Folder	-	-	-
route_id-B/	Folder	-	-	-
route_id-C/	Folder	-	-	-
route_id-D/	Folder	-	-	-
route_id-E/	Folder	-	-	-
route_id-F/	Folder	-	-	-
route_id-G/	Folder	-	-	-
route_id-H/	Folder	-	-	-
route_id-I/	Folder	-	-	-
route_id-J/	Folder	-	-	-

Bottom Window: The URL is <https://s3.console.aws.amazon.com/s3/buckets/learn-emr-2021/learn-emr-2021/delta/demo-video/>. The page title is "delta-hive-table". The table lists 2 objects under the "delta-hive-table" folder: "_delta_log/" and "route_id-A/".

Name	Type	Last modified	Size	Storage class
_delta_log/	Folder	-	-	-
route_id-A/	Folder	-	-	-

[Read the delta files and Review:](#)

Incremental Data Processing using Delta Lake with EMR

Saving the data frame in Delta format in S3

```
In [7]: df1.write.mode("overwrite").format("delta").partitionBy("route_id").save(deltaHivePath)
```

Review written data using Spark SQL temporary view

```
In [1]: spark.read.format("delta").load(deltaHivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT * FROM temp_trip_table LIMIT 5").show()
spark.sql("SELECT max(trip_id) FROM temp_trip_table").show()
```

Now lets create manifest files for accessing this delta storage data from Athena as table

```
In [1]: deltaTable = DeltaTable.forPath(spark, deltaHivePath)
deltaTable.generate("symlink_format_manifest")
```

Creating a Spark table in Glue Data Catalog with same data

```
In [1]: deltaTable = "demo_delta_db" + "." + "delta_spark_table"
In [1]: df1.write.format("delta").partitionBy("route_id").mode("overwrite").option("path", deltaSparkPath).saveAsTable(deltaTable)
```

Creating A Delta table in Glue:

Now lets create manifest files for accessing this delta storage data from Athena as table

```
In [9]: deltaTable = DeltaTable.forPath(spark, deltaHivePath)
deltaTable.generate("symlink_format_manifest")
```

Creating a Spark table in Glue Data Catalog with same data

```
In [1]: deltaTable = "demo_delta_db" + "." + "delta_spark_table"
In [1]: df1.write.format("delta").partitionBy("route_id").mode("overwrite").option("path", deltaSparkPath).saveAsTable(deltaTable)
In [1]: spark.sql("use demo_delta_db")
spark.sql("show tables").show(20)
In [1]: spark.sql("show create table delta_spark_table").show(200, truncate=False)
In [1]: %%spark --spark -o delta_trip_op
delta_trip_op = spark.sql("select count(*) from demo_delta_db.delta_spark_table")
delta_trip_op.show()
delta_trip_op = spark.sql("select max(trip_id) from demo_delta_db.delta_spark_table")
delta_trip_op.show()
delta_trip_op = spark.sql("select destination, route_id, trip_id, timestamp from demo_delta_db.delta_spark_table limit 10")
delta_trip_op.show()
delta_trip_op = spark.sql("select destination, route_id, trip_id, timestamp from demo_delta_db.delta_spark_table order by route_id")
delta_trip_op.show()
```

Inserting Data to delta lake table:

```
This point forward only Athena table will be used for data processing

Let's INSERT some new rows to the Athena table with schema evolution

In [1]: from datetime import datetime
        def get_json_data_update(start, count, dest, origin):
            time_stamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            data = [{"trip_id": i, "timestamp": time_stamp, "route_id": chr(65 + (i % 10)), "destination": dest[i%10], "origin": origin} for i in range(start, start+count)]
            return data

        # Creates the Dataframe
        def create_json_df_update(spark, data):
            sc = spark.sparkContext
            return spark.read.json(sc.parallelize(data))

In [1]: insert_dest = ["Syracuse", "Syracuse", "Syracuse"]
insert_origin = ["Seattle", "New York", "New Jersey", "Los Angeles", "Las Vegas", "Tucson", "Washington DC", "Philadelphia", "Chicago", "Milwaukee", "Minneapolis", "St. Paul", "Honolulu", "Honolulu", "Honolulu", "Honolulu", "Honolulu", "Honolulu", "Honolulu"]

tripUpdates = create_json_df_update(spark, get_json_data_update(2000000, 20, insert_dest, insert_origin))
tripUpdates.count()
tripUpdates.show()

In [1]: tripUpdates.printSchema()

In [ ]: deltaHivePath = "s3://learn-env-2021/delta/dmn-video/delta-hive-table/"
tripUpdates.write.format("delta").mode("append").option("mergeSchema", "true").partitionBy("route_id").save(deltaHivePath)
```

Sechma Evaluation:

Add new column and insert the data to same database: (20 NEW RECORDS)

This point forward only Athena table will be used for data processing

Lets INSERT some new rows to the Athena table with schema evolution

```
In [1]: from datetime import datetime

def get_json_data_update(start, count, dest, origin):
    time_stamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    trip_id = i + 1
    timestamp = time_stamp, "route_id": chr(65 + (i * 10)), "destination": dest[i*10], "origin": origin
    return data

# Creates the DataFrame
def create_json_df(spark, data):
    sc = spark.sparkContext
    return spark.read.json(sc.parallelize(data))

In [2]: insert_dest = ["Syracuse", "Syracuse", "Syracuse", "Syracuse", "Syracuse", "Syracuse", "Syracuse", "Syracuse"]
insert_origin = ["Seattle", "New York", "New Jersey", "Los Angeles", "Las Vegas", "Tucson", "Washington DC", "Philadelphia"]

tripUpdates = create_json_df(spark, get_json_data_update(2000000, 20, insert_dest, insert_origin))
tripUpdates.count()
tripUpdates.show()

In [3]: tripUpdates.printSchema()

In [4]: deltaHivePath = "s3://learn-emr-2021/delta/demo-video/delta-hive-table/"
tripUpdates.write.format("delta").mode("append").option("mergedSchema", "true").partitionBy("route_id").save(deltaHivePath)
```

Query the updated Table

Firefox File Edit View History Bookmarks Tools Window Help

Incremental Data Processing using Delta Lake with EMR

jupyter DeltaLakeDemo Last Checkpoint: 5 hours ago (unaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted PySpark

```
In [19]: deltaLivePath = "s3://learn-emr-2021/delta/demovideo/delta-hive-table/"  
tripUpdates.write.format("delta").mode("append").option("mergeSchema", "true").partitionBy("route_id").save(deltaLivePath)
```

Now lets query the updated data using Spark SQL

```
In [20]: spark.read.format("delta").load(deltaLivePath).createOrReplaceTempView("temp_trip_table")  
spark.sql("SELECT count(*) FROM temp_trip_table").show()
```

Spark Job Progress

```
+-----+  
| count(1)|  
+-----+  
| 2000028 |  
+-----+
```

```
In [ ]: spark.sql("SELECT * FROM temp_trip_table where trip_id > 1999996 order by trip_id").show()
```

Now Updating the Manifest files for Athena

```
In [ ]: deltaLivePath = "s3://learn-emr-2021/delta/demovideo/delta-hive-table/"  
deltaTable = DeltaTable.forPath(spark, deltaLivePath)
```

27.21 57:54

Scroll for details

Firefox File Edit View History Bookmarks Tools Window Help

https://e-78kmfdtagjpe3juq3c09t3h.emrnotebooks-prod.us-east-2.amazonaws.com/655219238564-Admin

jupyter DeltaLakeDemo Last Checkpoint: 6 hours ago (unsaved changes)

In [21]: `spark.sql("SELECT * FROM temp_trip_table where trip_id > 1999996 order by trip_id").show()`

+---+
| destination|route_id|trip_id| timestamp| origin|
+---+
Philadelphia	H	1999997	2021-03-11 23:39:50	null
Miami	I	1999998	2021-03-11 23:39:50	null
San Francisco	J	1999999	2021-03-11 23:39:50	null
Philadelphia	A	2000000	2021-03-11 23:37:47	Seattle
Syracuse	B	2000001	2021-03-11 23:37:47	New York
Syracuse	C	2000002	2021-03-11 23:37:47	New Jersey
Syracuse	D	2000003	2021-03-11 23:37:47	Los Angeles
Syracuse	E	2000004	2021-03-11 23:37:47	Las Vegas
Syracuse	F	2000005	2021-03-11 23:37:47	Tucson
Syracuse	G	2000006	2021-03-11 23:37:47	Washington DC
Philadelphia	H	2000007	2021-03-11 23:37:47	Philadelphia
Syracuse	I	2000008	2021-03-11 23:37:47	Philadelphia
Syracuse	J	2000009	2021-03-11 23:37:47	San Francisco
Syracuse	A	2000010	2021-03-11 23:37:47	Seattle
Syracuse	B	2000011	2021-03-11 23:37:47	New York
Syracuse	C	2000012	2021-03-11 23:37:47	New Jersey
Syracuse	D	2000013	2021-03-11 23:37:47	Los Angeles
Syracuse	E	2000014	2021-03-11 23:37:47	Las Vegas
Syracuse	F	2000015	2021-03-11 23:37:47	Tucson
Syracuse	G	2000016	2021-03-11 23:37:47	Washington DC
+---+
only showing top 20 rows

PySpark

aws

Firefox File Edit View History Bookmarks Tools Window Help

https://e-78kmfdtagjpe3juq3c09t3h.emrnotebooks-prod.us-east-2.amazonaws.com/655219238564-Admin

jupyter DeltaLakeDemo Last Checkpoint: 6 hours ago (unsaved changes)

In [21]: `spark.sql("SELECT * FROM temp_trip_table").show()`

+---+
| destination|route_id|trip_id| timestamp| origin|
+---+
Syracuse	A	2000000	2021-03-11 23:37:47	Las Vegas
Syracuse	B	2000001	2021-03-11 23:37:47	Tucson
Syracuse	C	2000002	2021-03-11 23:37:47	Washington DC
Syracuse	D	2000003	2021-03-11 23:37:47	Philadelphia
Syracuse	E	2000004	2021-03-11 23:37:47	Miami
Syracuse	F	2000005	2021-03-11 23:37:47	San Francisco
Syracuse	G	2000006	2021-03-11 23:37:47	Seattle
Syracuse	H	2000007	2021-03-11 23:37:47	New York
Syracuse	I	2000008	2021-03-11 23:37:47	New Jersey
Syracuse	J	2000009	2021-03-11 23:37:47	Los Angeles
Syracuse	A	2000010	2021-03-11 23:37:47	Las Vegas
Syracuse	B	2000011	2021-03-11 23:37:47	New York
Syracuse	C	2000012	2021-03-11 23:37:47	New Jersey
Syracuse	D	2000013	2021-03-11 23:37:47	Los Angeles
Syracuse	E	2000014	2021-03-11 23:37:47	Las Vegas
Syracuse	F	2000015	2021-03-11 23:37:47	Tucson
Syracuse	G	2000016	2021-03-11 23:37:47	Washington DC
+---+
only showing top 20 rows

In [22]: `spark.sql("CREATE OR REPLACE TEMP VIEW temp_trip_table AS SELECT * FROM temp_trip_table WHERE trip_id > 1999996 ORDER BY trip_id")`

In [23]: `spark.sql("SELECT COUNT(*) AS Count_Syracuse FROM temp_trip_table WHERE destination = 'Syracuse'").show()`

In [24]: `spark.sql("SELECT COUNT(*) AS Count_Philadelphia FROM temp_trip_table WHERE destination = 'Philadelphia'").show()`

PySpark

aws

Update Data IN Deltalake:

Now lets UPDATE some data

```
In [ ]: deltaTable = DeltaTable.forPath(spark, deltaHivePath)
In [ ]: spark.read.format("delta").load(deltaHivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT count(*) as Count_Syracuse FROM temp_trip_table where destination = 'Syracuse'").show()
spark.sql("SELECT count(*) as Count_Philadelphia FROM temp_trip_table where destination = 'Philadelphia'").show()

In [ ]: deltaTable = DeltaTable.forPath(spark, deltaHivePath)
deltaTable.update("destination = 'Syracuse'","destination = 'Philadelphia'")

deltaTable = DeltaTable.forPath(spark, deltaHivePath)
deltaTable.generate("symlink_format_manifest")

In [ ]: spark.read.format("delta").load(deltaHivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT count(*) as Count_Syracuse FROM temp_trip_table where destination = 'Syracuse'").show()
spark.sql("SELECT count(*) as Count_Philadelphia FROM temp_trip_table where destination = 'Philadelphia'").show()
```

Now lets DELETE some data

```
In [ ]: spark.read.format("delta").load(deltaHivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT count(*) as Count_total FROM temp_trip_table").show()
spark.sql("SELECT count(*) as Count_NJ FROM temp_trip_table where destination = 'New Jersey'").show()

In [ ]: deltaTable = DeltaTable.forPath(spark, deltaHivePath)
deltaTable.delete("destination = 'New Jersey'")

In [ ]: spark.read.format("delta").load(deltaHivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT count(*) as Count_Syracuse FROM temp_trip_table where destination = 'Syracuse'").show()
spark.sql("SELECT count(*) as Count_Philadelphia FROM temp_trip_table where destination = 'Philadelphia'").show()
```

Now lets UPDATE some data

```
In [ ]: deltaTable = DeltaTable.forPath(spark, deltaHivePath)
In [ ]: spark.read.format("delta").load(deltaHivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT count(*) as Count_Syracuse FROM temp_trip_table where destination = 'Syracuse'").show()
spark.sql("SELECT count(*) as Count_Philadelphia FROM temp_trip_table where destination = 'Philadelphia'").show()

In [ ]: deltaTable = DeltaTable.forPath(spark, deltaHivePath)
deltaTable.update("destination = 'Syracuse'","destination = 'Philadelphia'")

deltaTable = DeltaTable.forPath(spark, deltaHivePath)
deltaTable.generate("symlink_format_manifest")

In [ ]: spark.read.format("delta").load(deltaHivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT count(*) as Count_Syracuse FROM temp_trip_table where destination = 'Syracuse'").show()
spark.sql("SELECT count(*) as Count_Philadelphia FROM temp_trip_table where destination = 'Philadelphia'").show()
```

Now lets DELETE some data

```
In [ ]: spark.read.format("delta").load(deltaHivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT count(*) as Count_total FROM temp_trip_table").show()
spark.sql("SELECT count(*) as Count_NJ FROM temp_trip_table where destination = 'New Jersey'").show()

In [ ]: deltaTable = DeltaTable.forPath(spark, deltaHivePath)
deltaTable.delete("destination = 'New Jersey'")

In [ ]: spark.read.format("delta").load(deltaHivePath).createOrReplaceTempView("temp_trip_table")
```

The image shows two side-by-side Jupyter Notebook interfaces running in Firefox. Both notebooks are titled "jupyter DeltaLakeDemo" and show code snippets for interacting with a Delta Lake table named "temp_trip_table".

Top Notebook (Left):

```
In [1]: deltaTable = DeltaTable.forPath(spark, deltaHivePath)
In [23]: spark.read.format("delta").load(deltaHivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT count(*) as Count_Syracuse FROM temp_trip_table where destination = 'Syracuse'").show()
spark.sql("SELECT count(*) as Count_Philadelphia FROM temp_trip_table where destination = 'Philadelphia'").show()

Now lets UPDATE some data
In [1]: deltaTable = DeltaTable.forPath(spark, deltaHivePath)
deltaTable.update("destination = 'Syracuse'", {"destination": "'Philadelphia'"})
deltaTable = DeltaTable.forPath(spark, deltaHivePath)
deltaTable.generate("symlink_format_manifest")

In [1]: spark.read.format("delta").load(deltaHivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT count(*) as Count_Syracuse FROM temp_trip_table where destination = 'Syracuse'").show()
spark.sql("SELECT count(*) as Count_Philadelphia FROM temp_trip_table where destination = 'Philadelphia'").show()
```

Bottom Notebook (Right):

```
In [1]: deltaTable = DeltaTable.forPath(spark, deltaHivePath)
deltaTable.update("destination = 'Syracuse'", {"destination": "'Philadelphia'"})
deltaTable = DeltaTable.forPath(spark, deltaHivePath)
deltaTable.generate("symlink_format_manifest")

In [1]: spark.read.format("delta").load(deltaHivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT count(*) as Count_Syracuse FROM temp_trip_table where destination = 'Syracuse'").show()
spark.sql("SELECT count(*) as Count_Philadelphia FROM temp_trip_table where destination = 'Philadelphia'").show()

Now lets DELETE some data
In [1]: spark.read.format("delta").load(deltaHivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT count(*) as Count_total FROM temp_trip_table").show()
spark.sql("SELECT count(*) as Count_NJ FROM temp_trip_table where destination = 'New Jersey'").show()

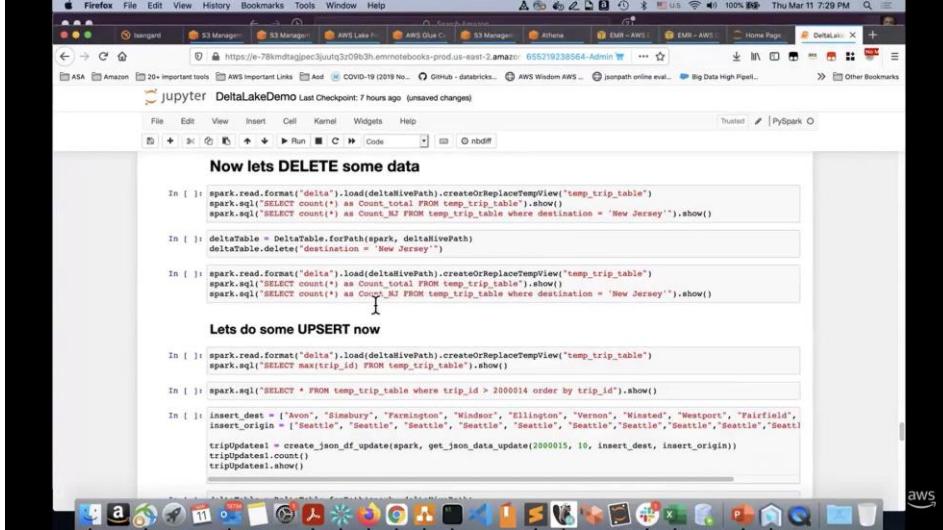
In [1]: deltaTable = DeltaTable.forPath(spark, deltaHivePath)
deltaTable.delete("destination = 'New Jersey'")

In [1]: spark.read.format("delta").load(deltaHivePath).createOrReplaceTempView("temp_trip_table")
```

df.update(<condition>,{columnnametoupdated:Newvalue})

```
df.update("destination='Syracuse'","destination"="Philadelphia")
```

Delete Records:



```

Now lets DELETE some data

In [1]: spark.read.format("delta").load(deltaLivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT count(*) as Count_total FROM temp_trip_table").show()
spark.sql("SELECT count(*) as Count_NJ FROM temp_trip_table where destination = 'New Jersey').show()

In [2]: deltaTable = DeltaTable.forPath(spark, deltaHivePath)
deltaTable.delete("destination = 'New Jersey')

In [3]: spark.read.format("delta").load(deltaLivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT count(*) as Count_total FROM temp_trip_table").show()
spark.sql("SELECT count(*) as Count_NJ FROM temp_trip_table where destination = 'New Jersey').show()

```

Lets do some UPSERT now

```

In [4]: spark.read.format("delta").load(deltaLivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT max(trip_id) FROM temp_trip_table").show()

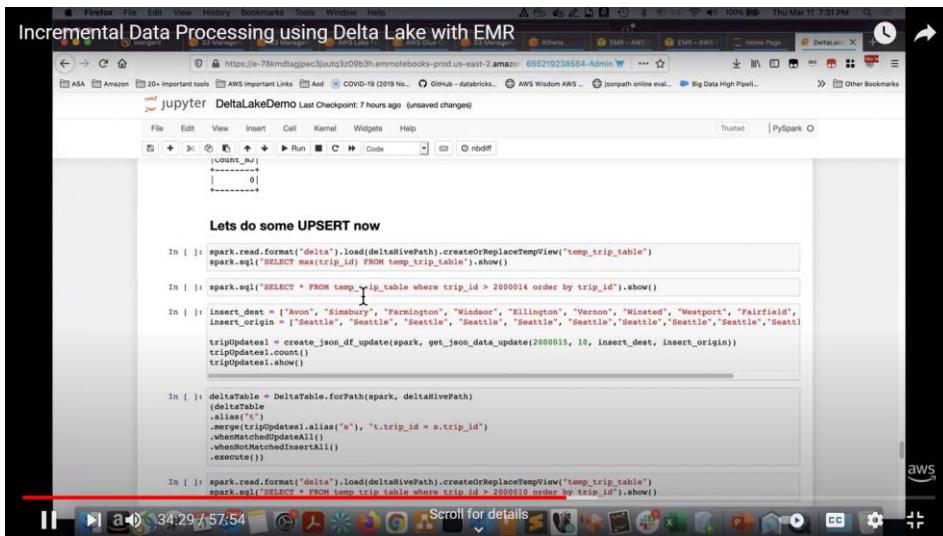
In [5]: insert_dest = ["Avon", "Simsbury", "Farmington", "Windsor", "Ellington", "Vernon", "Middletown", "Westport", "Fairfield", "West Hartford"]
insert_origin = ["Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle"]

tripUpdateisel = create_json_df_update(spark, get_json_data_update(2000015, 10, insert_dest, insert_origin))
tripUpdateisel.count()
tripUpdateisel.show()

```

`df.delete("destination='newjersy'")`

Upsert:



```

Incremental Data Processing using Delta Lake with EMR

In [1]: spark.read.format("delta").load(deltaLivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT max(trip_id) FROM temp_trip_table").show()

In [2]: insert_dest = ["Avon", "Simsbury", "Farmington", "Windsor", "Ellington", "Vernon", "Middletown", "Westport", "Fairfield", "West Hartford"]
insert_origin = ["Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle"]

tripUpdateisel = create_json_df_update(spark, get_json_data_update(2000015, 10, insert_dest, insert_origin))
tripUpdateisel.count()
tripUpdateisel.show()

In [3]: deltaTable = DeltaTable.forPath(spark, deltaHivePath)
(deltaTable
 .alias("t")
 .as(selectUpdateisel.alias("s"), "t.trip_id = s.trip_id")
 .whenNotMatchedUpdateAll()
 .whenNotMatchedInsertAll()
 .execute())

In [4]: spark.read.format("delta").load(deltaLivePath).createOrReplaceTempView("temp_trip_table")
spark.sql("SELECT * FROM temp_trip_table where trip_id > 2000010 order by trip_id").show()

```

Time travel:

```

In [31]: insert_deat = ["Avon", "Blaebury", "Farmington", "Windsor", "Ellington", "Vermon", "Winseted", "Westport", "Fairfield", "Vernon", "Seattle", "Winseted", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle"]
insert_origin = ["Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle", "Seattle"]

tripUpdateesi = create_json_df_update(spark, get_json_data_update(2000015, 10, insert_deat, insert_origin))
tripUpdateesi.count()
tripUpdateesi.show()

```

Spark Job Progress

destination	origin	route_id	trip_id	timestamp
Vernon	Seattle	F	2000015	2021-03-12 00:33:36
Winseted	Seattle	G	2000016	2021-03-12 00:33:36
Westport	Seattle	H	2000017	2021-03-12 00:33:36
Fairfield	Seattle	I	2000018	2021-03-12 00:33:36
Stamford	Seattle	J	2000019	2021-03-12 00:33:36
Avon	Seattle	A	2000020	2021-03-12 00:33:36
Silsbury	Seattle	B	2000021	2021-03-12 00:33:36
Farmington	Seattle	C	2000022	2021-03-12 00:33:36
Windsor	Seattle	D	2000023	2021-03-12 00:33:36

History table(timestampasof & versiosasof)

```

In [1]: df_v0 = (spark.read
            .format("delta")
            .option("timestampAsOf", "2021-03-11 17:04:46") # timestamp after table creation
            .load(deltaTablePath)
            .df_v0.take(5))

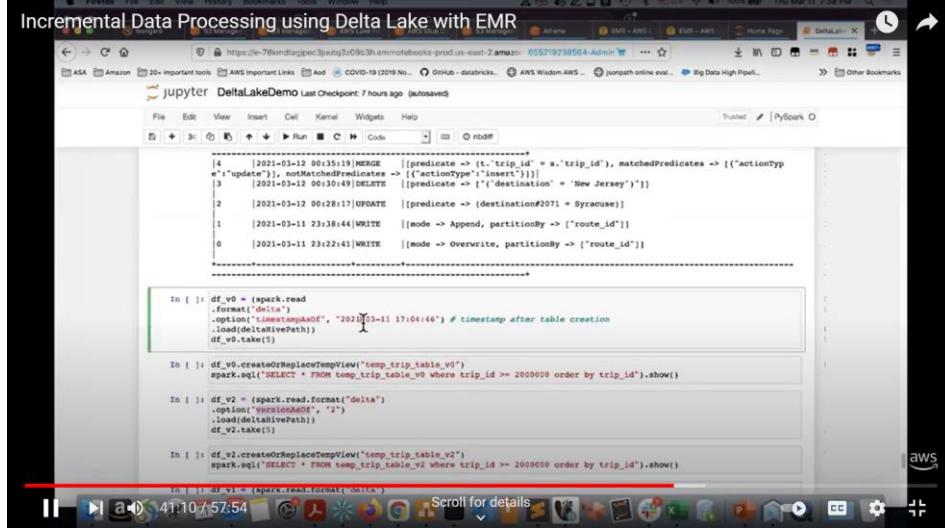
In [2]: df_v0.createOrReplaceTempView("temp_trip_table_v0")
spark.sql("SELECT * FROM temp_trip_table_v0 where trip_id >= 2000000 order by trip_id").show()

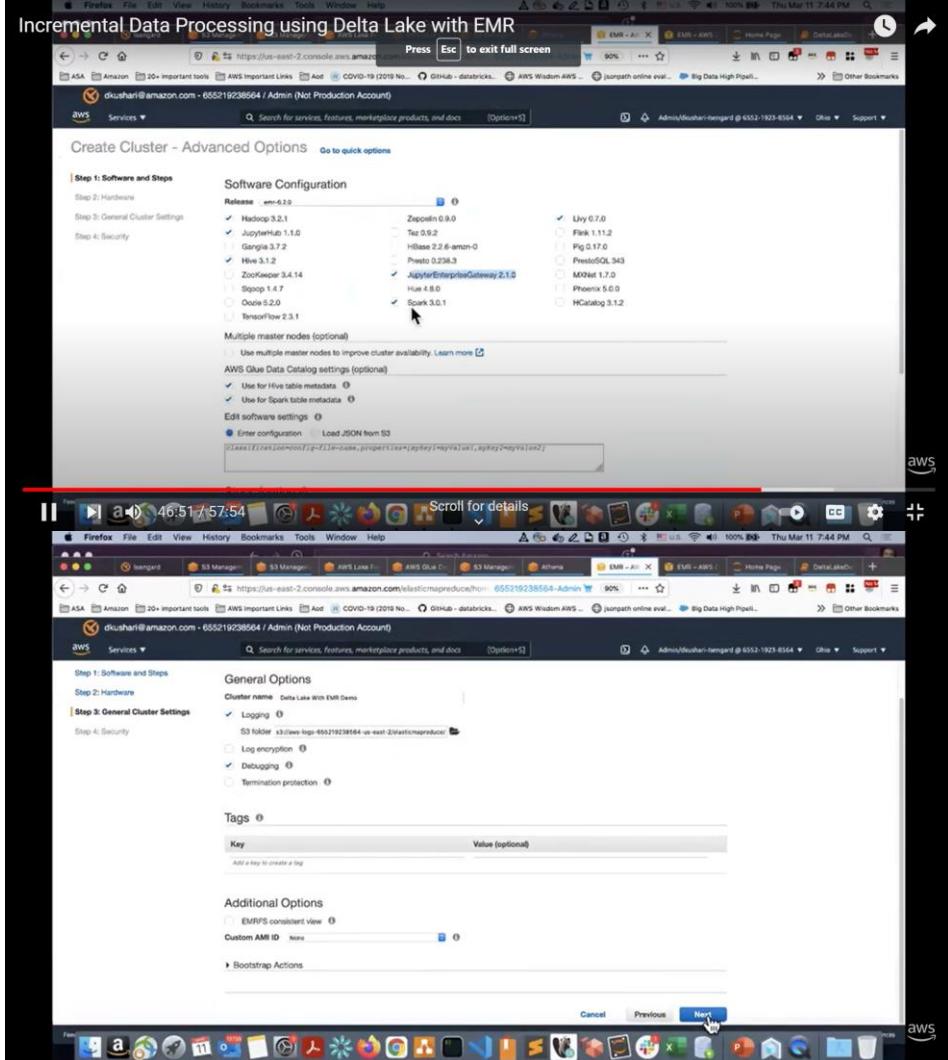
In [3]: df_v2 = (spark.read.format("delta")
            .option("versionAsOf", "2")
            .load(deltaTablePath)
            .df_v2.take(5))

In [4]: df_v2.createOrReplaceTempView("temp_trip_table_v2")
spark.sql("SELECT * FROM temp_trip_table_v2 where trip_id >= 2000000 order by trip_id").show()

In [5]: df_v1 = (spark.read.format("delta")
            .option("versionAsOf", "1")
            .load(deltaTablePath)
            .df_v1.take(5))

```





Create Note Book:

Firefox File Edit View History Bookmarks Tools Window Help

https://us-east-2.console.aws.amazon.com/elasticmapreduce/home?region=us-east-2 EMR - All EMR - AWS Home Page Default skin...

dushar@amazon.com - 655219238564 / Admin (Not Production Account)

AWS Services ▾ Search for services, features, marketplace products, and docs [Options+I]

Adm/dushar-hengard @ 655219238564 Groups Support ▾

Notebooks

Amazon EMR

EMR on EC2

- Clusters
- Notebooks**
- Git repositories
- Security configurations
- Block public access
- VPC subnets
- Events

EMR on EKS

Virtual clusters

Help What's new

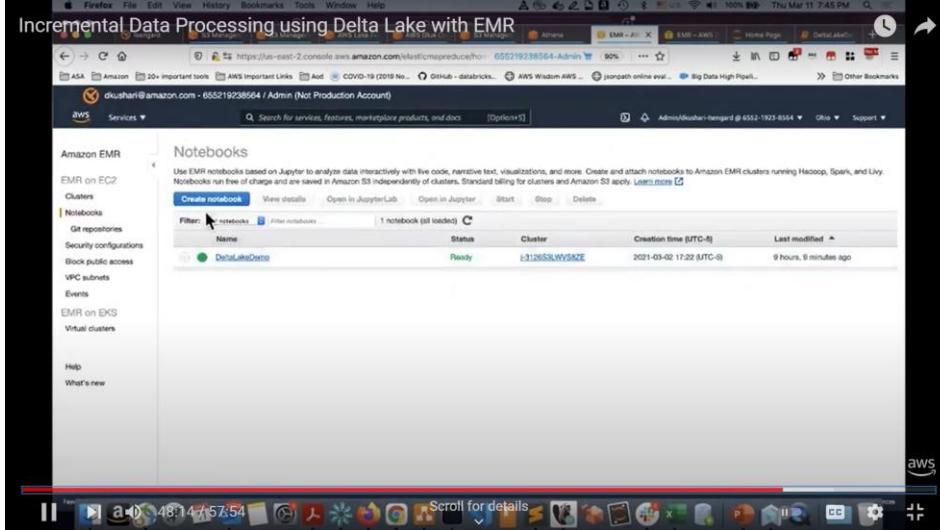
Create notebook View details Open in JupyterLab Open in Jupyter Start Stop Delete

Filter Notebooks Filter notebooks 1 notebook (all listed) C

Name	Status	Cluster	Creation time (UTC-8)	Last modified
DetalutDemo	Ready	EMR-12345678901234567890	2021-03-02 17:22 (UTC-8)	9 hours, 9 minutes ago

aws

Scroll for details



Firefox File Edit View History Bookmarks Tools Window Help

https://us-east-2.console.aws.amazon.com/elasticmapreduce/home?region=us-east-2 EMR - All EMR - AWS Home Page Default skin...

dushar@amazon.com - 655219238564 / Admin (Not Production Account)

AWS Services ▾ Search for services, features, marketplace products, and docs [Options+I]

Adm/dushar-hengard @ 655219238564 Groups Support ▾

Create notebook

Amazon EMR

EMR on EC2

- Clusters
- Notebooks**
- Git repositories
- Security configurations
- Block public access
- VPC subnets
- Events

EMR on EKS

Virtual clusters

Help What's new

Name and configure your notebook

Name your notebook, choose a cluster or create one, and customize configuration options if desired. Learn more

Notebook name* Notes may only contain alphanumeric characters, hyphens (-), underscores (_), and periods (.)

Description

Cluster* Choose an existing cluster Create a cluster

Security groups Use default security groups Choose security groups

AWS service role* EMR_Notebooks_DefaultRole

Notebook location* Choose an S3 location where files for this notebook are saved.

Use the default S3 location s3://us-east-1:9238564-us-east-2/notebooks/ Choose an existing S3 location in us-east-2

Git repository Link to a Git repository

