DATABRICKS STREAM CASE STUDY

CASE STUDY

accenture technology

NEW YORK CITY TAXI DATA

accenturetechnology

Index

2 Background

- New York City has two types of taxies: yellow and green; they are widely recognizable symbols of the city.

- Taxis painted **yellow** (medallion taxis) can pick up passengers from anywhere in the five boroughs.

- Those painted apple **green** (street hail livery vehicles, commonly known as "boro taxis"), which began to appear in August 2013, are allowed to pick up passengers in Upper Manhattan, the Bronx, Brooklyn, Queens (excluding LaGuardia Airport and John F. Kennedy International Airport), and Staten Island.

- Both taxi types have the same fare structure.

- Taxicabs are operated by private companies and licensed by the ***New York City Taxi and Limousine Commission (TLC).***

- It also oversees over 40,000 other **for-hire vehicles (FHVs)**, including "black cars" like Uber, commuter vans, and ambulettes.

- All types of taxis are licensed by the ***TLC*** which oversees for-hire vehicles, taxis, commuter vans, and paratransit vehicles.

- **Accenture** is responsible for developing and maintaining the data and analytical systems for New York City taxi.

3 Challenges

- Things were smooth until the recent arrival of FHVs in the scene. Though the system was working well, the challenge started when Accenture started collecting, storing and processing the data for FHVs.

- FHVs are of multiple types:
    - Community cars, Black cars, Luxury limousines
    - High volume for-hire services which include app-based companies like Uber and Lyft. These dispatches are more than 10,000 trips per day

- The request for FHVs by passengers is accepted by bases, and then the bases dispatch the request to the cab drivers.

- There are more than 750 bases and 100000 FHVs, and all these different types of FHVs operate in different ways.

- The number of FHVs are much higher than yellow and green taxis, which led to exponential increase in data volumes.

- The schema for FHV data is a mix of various data formats like CSV, TSV and JSON formats. The sources of this data are quite disparate.

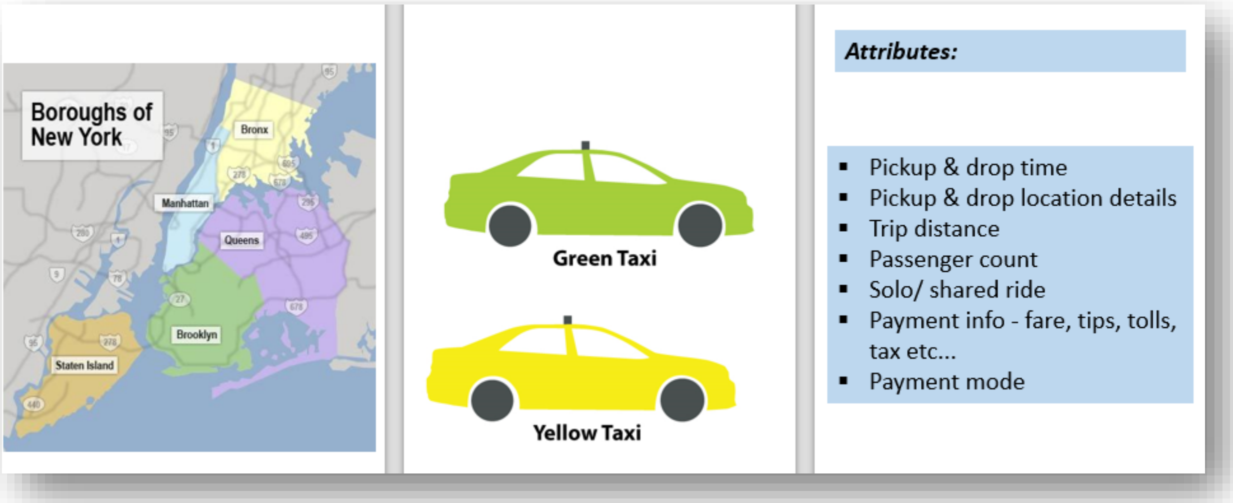
4 Business Need

- The requirement is to stage, process and store the data of all types of taxis irrespective of their sources and formats and transform it to the analytical needs.
- Finally build reports/visualizations which provide actionable insights.
- Most of this is needed in real time. There is an increase in demand for stream processing data as well due to higher trip rate.
- The requirement is to ingest and process the data at very high frequencies. Finally, none of the data should be discarded. In fact, it should be preserved for enabling use-cases related to regulatory compliance, passenger safety, insurance, targeted ads/promotions/offers etc.,
- TLC wants Accenture to build a common platform to store all data related to trips, cabs and passengers in order to analyze the data for better business insights like Revenue by taxi type, location, Total trips, max trips by regions etc.,

5 Proposed Solution

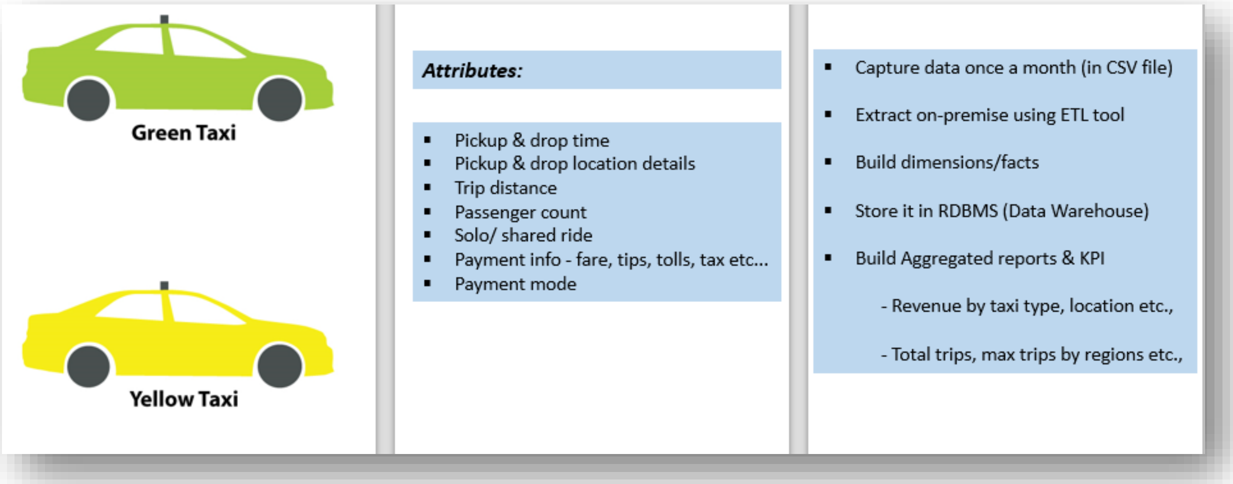
- Considering the Volume, Velocity and Veracity aspects of the data, Accenture has decided to build a Data Lake using Databricks Lake House Platform which is going to be a single store for raw intermittent and processed data.
- Accenture has analyzed various available options and delivered some PoC's. In a one-year contract with TLC, Accenture will build a Spark based Data Lake augmented by a Cloud based Data Lake on Azure.
- The solution will implement the latest features and concepts of Databricks Lake House Platform

6 Attributes and Sample Data



Attributes:

- Pickup & drop time
- Pickup & drop location details
- Trip distance
- Passenger count
- Solo/ shared ride
- Payment info - fare, tips, tolls, tax etc...
- Payment mode



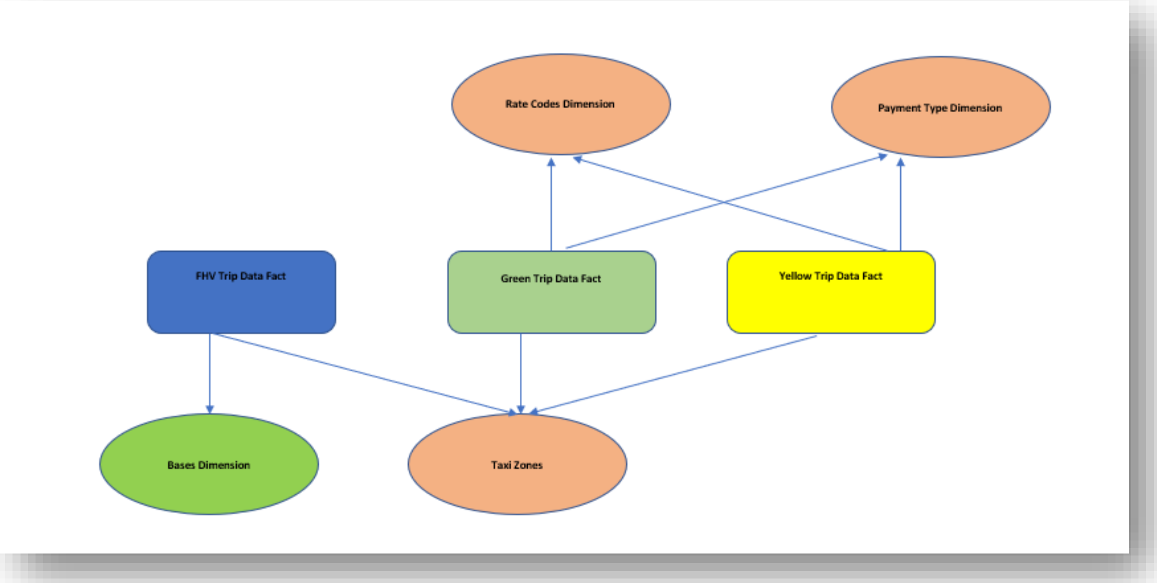
Attributes:

- Pickup & drop time
- Pickup & drop location details
- Trip distance
- Passenger count
- Solo/ shared ride
- Payment info - fare, tips, tolls, tax etc...
- Payment mode

- Capture data once a month (in CSV file)
- Extract on-premise using ETL tool
- Build dimensions/facts
- Store it in RDBMS (Data Warehouse)
- Build Aggregated reports & KPI

  - Revenue by taxi type, location etc.,

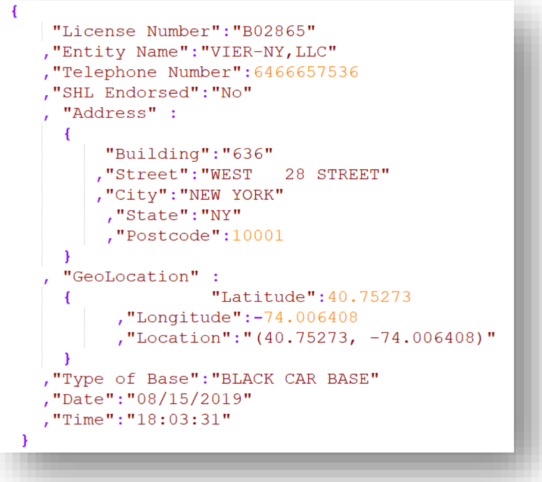
  - Total trips, max trips by regions etc.,



Sample Data: fhv_tripdata_2019-05.csv/ fhv_tripdata_2019-06.csv

dispatching_base_num	B00013
pickup_datetime	2019-06-01 00:51:33
dropoff_datetime	2019-06-01 01:20:07
PULocationID	83
DOLocationID	173
SR_Flag	Null

Sample Data: FhvBases.json



```
{
    "License Number":"B02865"
   ,"Entity Name":"VIER-NY,LLC"
   ,"Telephone Number":6466657536
   ,"SHL Endorsed":"No"
   , "Address" :
     {
        "Building":"636"
       ,"Street":"WEST   28 STREET"
       ,"City":"NEW YORK"
        ,"State":"NY"
        ,"Postcode":10001
     }
   , "GeoLocation" :
     {              "Latitude":40.75273
          ,"Longitude":-74.006408
          ,"Location":"(40.75273, -74.006408)"
     }
   ,"Type of Base":"BLACK CAR BASE"
   ,"Date":"08/15/2019"
   ,"Time":"18:03:31"
 }
```

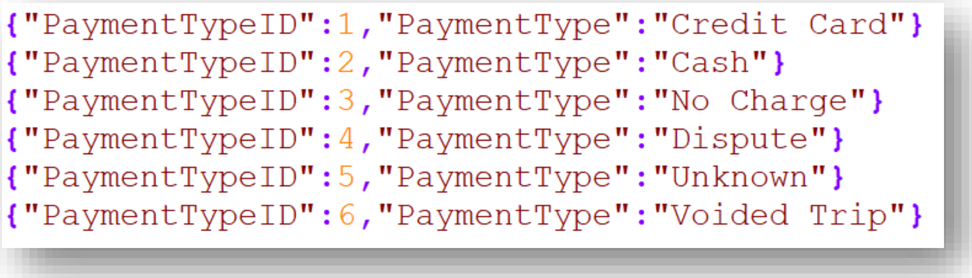
Sample Data: TaxiZones.csv

LocationID	1
Borough	EWR
Zone	Newark Airport
service_zone	EWR

Sample Data: green_tripdata_2019-05.csv/ green_tripdata_2019-06.csv

VendorID	1
lpep_pickup_datetime	2019-05-01 00:48:55
lpep_dropoff_datetime	2019-05-01 00:55:07
store_and_fwd_flag	N
RatecodeID	1
PULocationID	41
DOLocationID	42
passenger_count	1
trip_distance	1.50
fare_amount	7.5
extra	0
tip_amount	0.5
mta_tax	0
tolls_amount	0
ehail_fee	Null
improvement_surcharge	0.3
total_amount	8.3
payment_type	2
trip_type	1
congestion_surcharge	0

Sample Data: PaymentsType.json



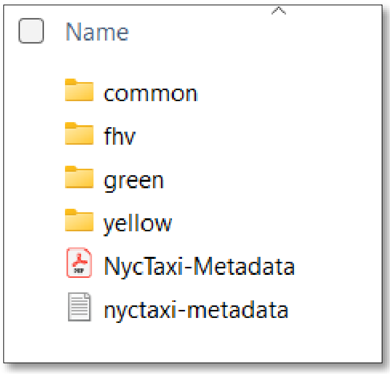
```
{"PaymentTypeID":1,"PaymentType":"Credit Card"}
{"PaymentTypeID":2,"PaymentType":"Cash"}
{"PaymentTypeID":3,"PaymentType":"No Charge"}
{"PaymentTypeID":4,"PaymentType":"Dispute"}
{"PaymentTypeID":5,"PaymentType":"Unknown"}
{"PaymentTypeID":6,"PaymentType":"Voided Trip"}
```

Sample Data: RateCodes.csv

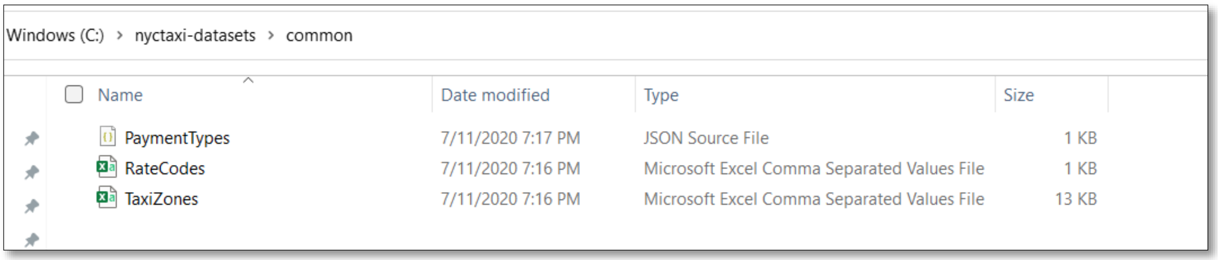
RateCodeID	**1**
RateCode	Standard Rate
IsApproved	Yes

Source Data Files

New York Taxi datasets

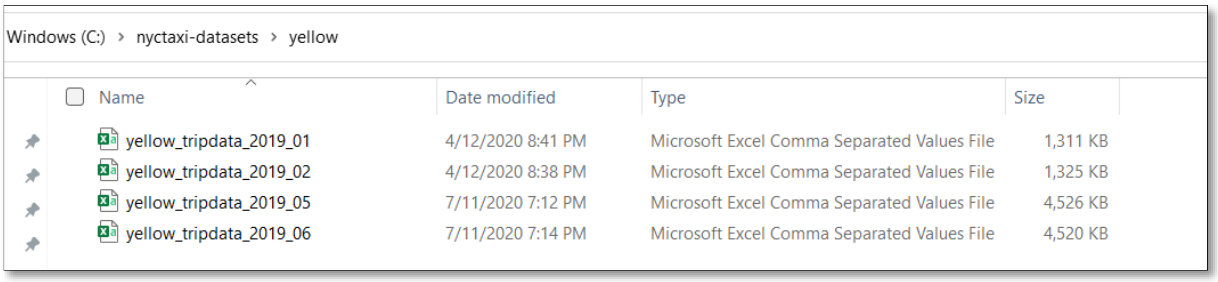


□ Name
common
fhv
green
yellow
NycTaxi-Metadata
nyctaxi-metadata



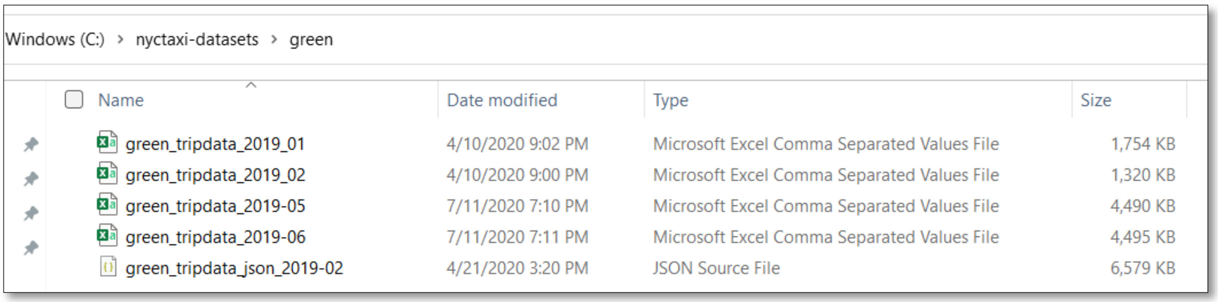
Windows (C:) › nyctaxi-datasets › common

□ Name	Date modified	Type	Size
PaymentTypes	7/11/2020 7:17 PM	JSON Source File	1 KB
RateCodes	7/11/2020 7:16 PM	Microsoft Excel Comma Separated Values File	1 KB
TaxiZones	7/11/2020 7:16 PM	Microsoft Excel Comma Separated Values File	13 KB



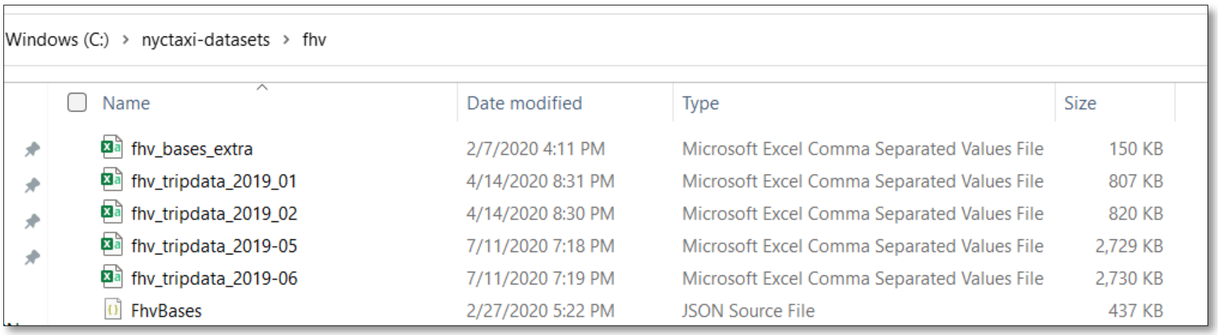
Windows (C:) › nyctaxi-datasets › yellow

□ Name	Date modified	Type	Size
yellow_tripdata_2019_01	4/12/2020 8:41 PM	Microsoft Excel Comma Separated Values File	1,311 KB
yellow_tripdata_2019_02	4/12/2020 8:38 PM	Microsoft Excel Comma Separated Values File	1,325 KB
yellow_tripdata_2019_05	7/11/2020 7:12 PM	Microsoft Excel Comma Separated Values File	4,526 KB
yellow_tripdata_2019_06	7/11/2020 7:14 PM	Microsoft Excel Comma Separated Values File	4,520 KB



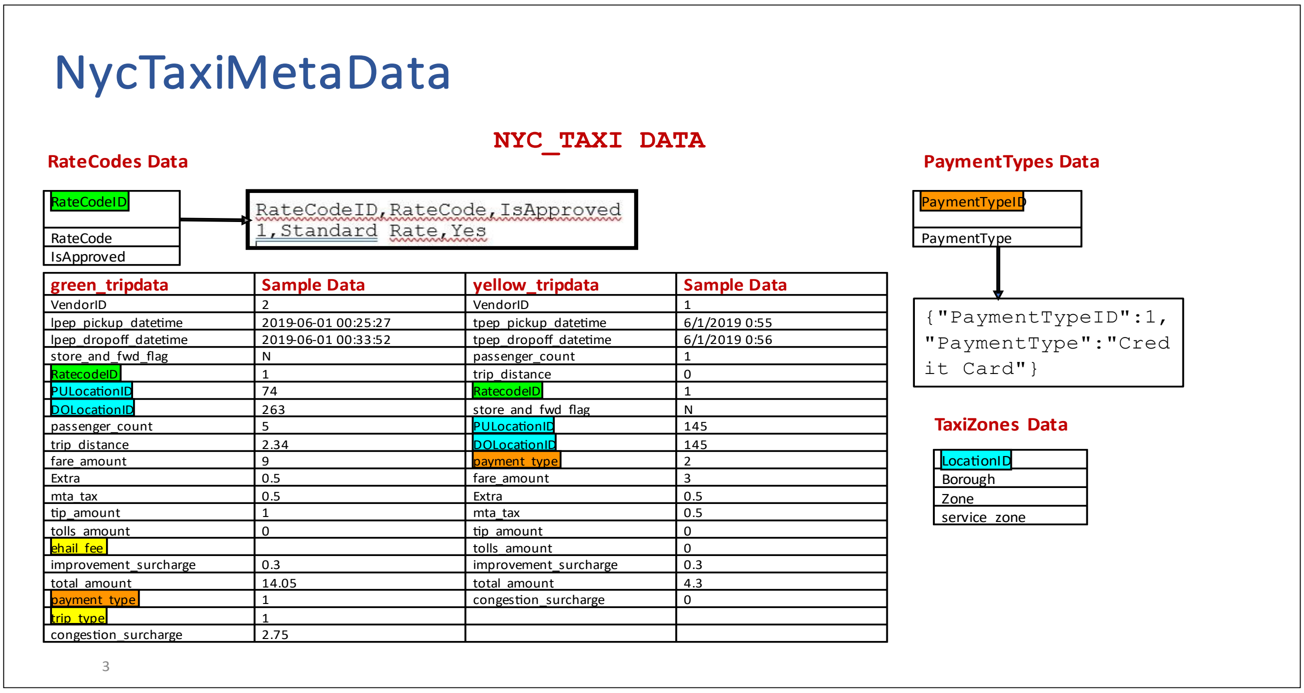
Windows (C:) › nyctaxi-datasets › green

□ Name	Date modified	Type	Size
green_tripdata_2019_01	4/10/2020 9:02 PM	Microsoft Excel Comma Separated Values File	1,754 KB
green_tripdata_2019_02	4/10/2020 9:00 PM	Microsoft Excel Comma Separated Values File	1,320 KB
green_tripdata_2019-05	7/11/2020 7:10 PM	Microsoft Excel Comma Separated Values File	4,490 KB
green_tripdata_2019-06	7/11/2020 7:11 PM	Microsoft Excel Comma Separated Values File	4,495 KB
green_tripdata_json_2019-02	4/21/2020 3:20 PM	JSON Source File	6,579 KB



Windows (C:) › nyctaxi-datasets › fhv

□ Name	Date modified	Type	Size
fhv_bases_extra	2/7/2020 4:11 PM	Microsoft Excel Comma Separated Values File	150 KB
fhv_tripdata_2019_01	4/14/2020 8:31 PM	Microsoft Excel Comma Separated Values File	807 KB
fhv_tripdata_2019_02	4/14/2020 8:30 PM	Microsoft Excel Comma Separated Values File	820 KB
fhv_tripdata_2019-05	7/11/2020 7:18 PM	Microsoft Excel Comma Separated Values File	2,729 KB
fhv_tripdata_2019-06	7/11/2020 7:19 PM	Microsoft Excel Comma Separated Values File	2,730 KB
FhvBases	2/27/2020 5:22 PM	JSON Source File	437 KB



NycTaxiMetaData

NYC_TAXI DATA

RateCodes Data

RateCodeID
RateCode
IsApproved

```
RateCodeID,RateCode,IsApproved
1,Standard Rate,Yes
```

PaymentTypes Data

PaymentTypeID
PaymentType

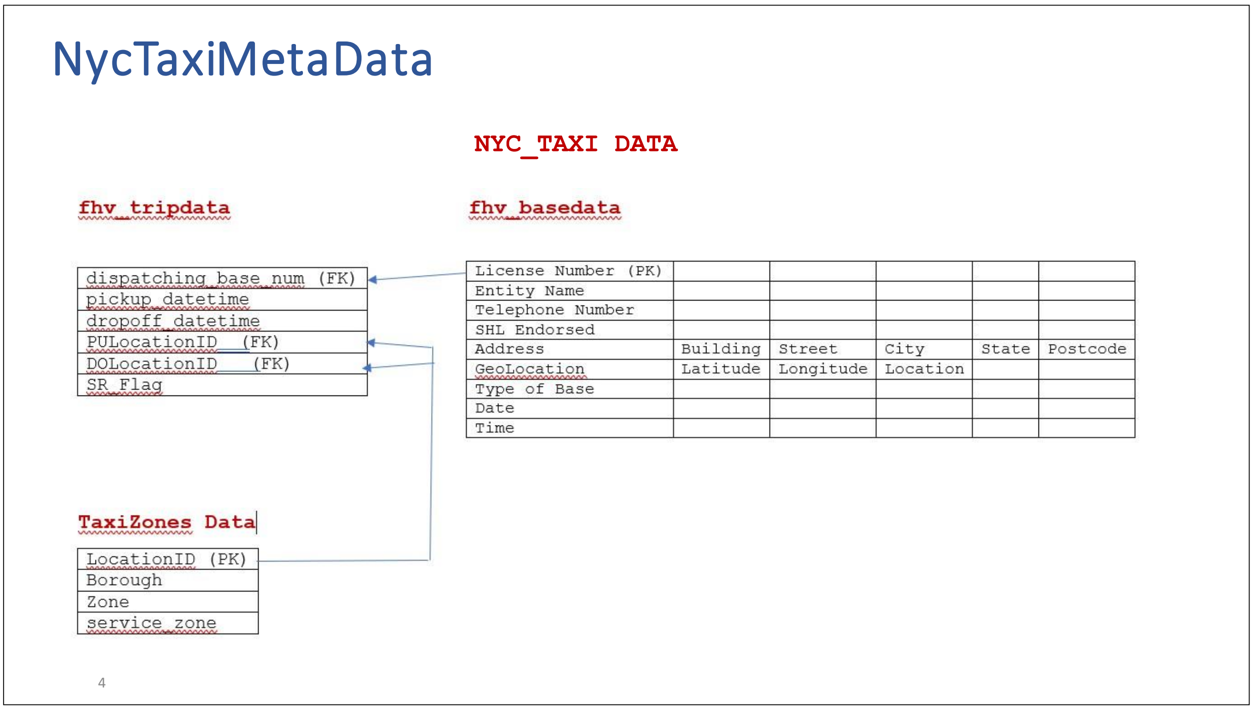
```
{"PaymentTypeID":1,
"PaymentType":"Cred
it Card"}
```

green_tripdata	Sample Data	yellow_tripdata	Sample Data
VendorID	2	VendorID	1
lpep_pickup_datetime	2019-06-01 00:25:27	tpep_pickup_datetime	6/1/2019 0:55
lpep_dropoff_datetime	2019-06-01 00:33:52	tpep_dropoff_datetime	6/1/2019 0:56
store_and_fwd_flag	N	passenger_count	1
RatecodeID	1	trip_distance	0
PULocationID	74	RatecodeID	1
DOLocationID	263	store_and_fwd_flag	N
passenger_count	5	PULocationID	145
trip_distance	2.34	DOLocationID	145
fare_amount	9	payment_type	2
Extra	0.5	fare_amount	3
mta_tax	0.5	Extra	0.5
tip_amount	1	mta_tax	0.5
tolls_amount	0	tip_amount	0
ehail_fee		tolls_amount	0
improvement_surcharge	0.3	improvement_surcharge	0.3
total_amount	14.05	total_amount	4.3
payment_type	1	congestion_surcharge	0
trip_type	1		
congestion_surcharge	2.75		

TaxiZones Data

LocationID
Borough
Zone
service_zone

3



NycTaxiMetaData

NYC_TAXI DATA

fhv_tripdata

dispatching_base_num (FK)
pickup_datetime
dropoff_datetime
PULocationID (FK)
DOLocationID (FK)
SR_Flag

fhv_basedata

License Number (PK)					
Entity Name					
Telephone Number					
SHL Endorsed					
Address	Building	Street	City	State	Postcode
GeoLocation	Latitude	Longitude	Location		
Type of Base					
Date					
Time					

TaxiZones Data

LocationID (PK)
Borough
Zone
service_zone

4

New York Taxi Data Sets Metadata

Green Trip Data Sets Metadata with Data Type (20 columns)

Column Name	*Data type*
VendorID	integer
lpep_pickup_datetime	timestamp
lpep_dropoff_datetime	timestamp
store_and_fwd_flag	string
RatecodeID	integer
PULocationID	integer
DOLocationID	integer
passenger_count	integer
trip_distance	double
fare_amount	double
extra	double
mta_tax	double
tip_amount	double
tolls_amount	double
ehail_fee	string
improvement_surcharge	double
total_amount	double
payment_type	integer
trip_type	integer
congestion_surcharge	double

Yellow Trip Data Sets Metadata with Data Type (18 columns)

Column Name	*Data type*
VendorID	integer
tpep_pickup_datetime	timestamp
tpep_dropoff_datetime	timestamp
passenger_count	integer
trip_distance	double
RatecodeID	integer
store_and_fwd_flag	string
PULocationID	integer
DOLocationID	integer
payment_type	integer
fare_amount	double
extra	double
mta_tax	double
tip_amount	double
tolls_amount	double
improvement_surcharge	double
total_amount	double
congestion_surcharge	double

Taxi Zones Data Set Metadata with Data Type (4 columns)

Column Name	Data type
LocationID	integer
Borough	string
Zone	string
service_zone	string

Payment Types Data Set Metadata with Data Type (2 columns)

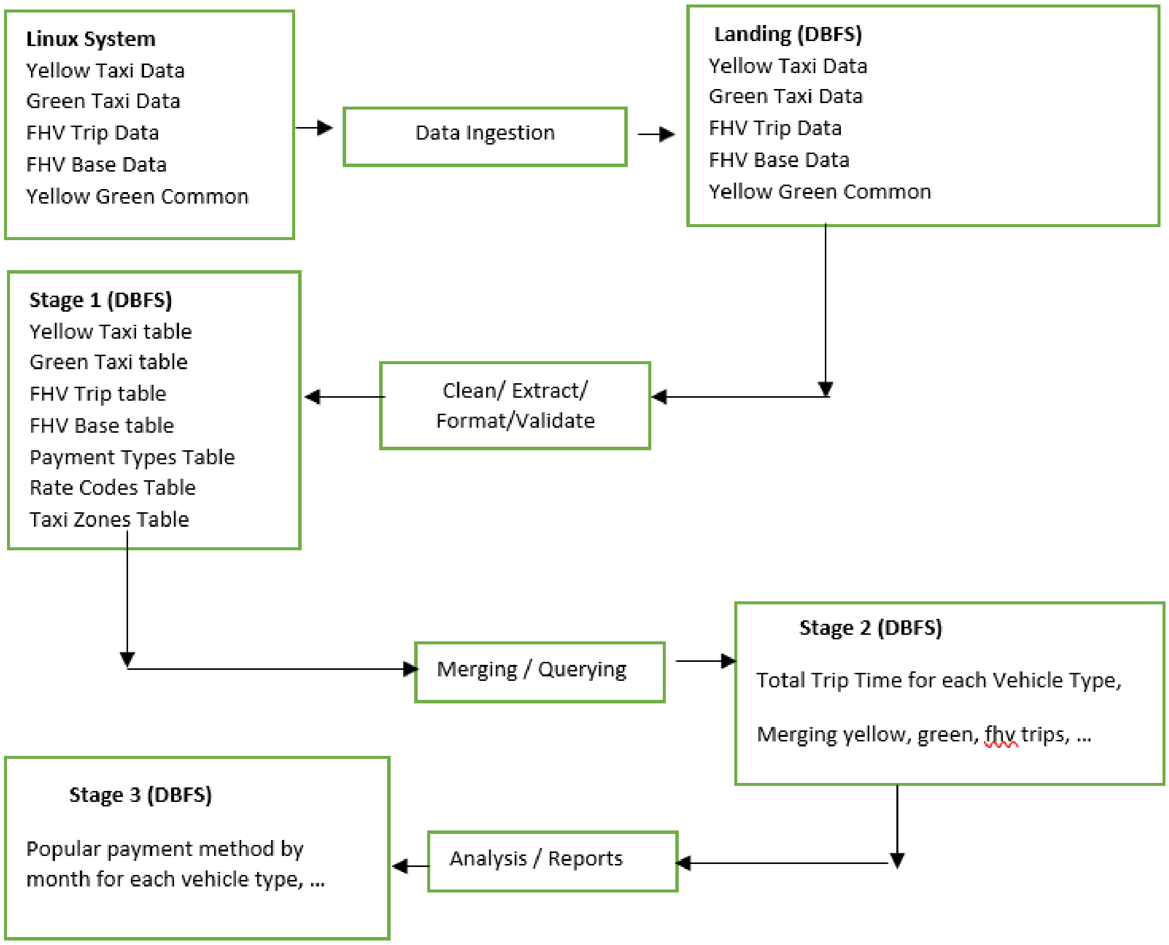
Column Name	Data type
PaymentTypeID	integer
PaymentType	string

Rate Codes Data Set Metadata with Data Type (3 columns)

Column Name	Data type
RateCodeID	integer
RateCode	string
IsApproved	string

New York Taxi Case Study Dataflow Diagram

Dataflow Diagram:



Path Convention for New York Taxi Data sets in DBFS

(For batch data processing)

`dbfs:/FileStore/tables/<CUSTOMIZED LOCATION>`

Landing location of Source Data in DBFS

dbfs:/FileStore/tables/nyctaxidata/landing/greendata/green_tripdata
dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata
dbfs:/FileStore/tables/nyctaxidata/landing/yellow_green_commondata/ratecodes/
dbfs:/FileStore/tables/nyctaxidata/landing/yellow_green_commondata/paymenttypes/
dbfs:/FileStore/tables/nyctaxidata/landing/yellow_green_commondata/taxizones/

dbfs:/FileStore/tables/nyctaxidata/landing/fhvdata/fhv_tripdata
dbfs:/FileStore/tables/nyctaxidata/landing/fhvdata/fhvbases
dbfs:/FileStore/tables/nyctaxidata/landing/fhvdata/fhvbases_csv

Stage1 Location of Data after ETL/Cleaning Operations from Landing Location:

dbfs:/FileStore/tables/nyctaxidata/stage1_spark/greendata/green_tripdata
dbfs:/FileStore/tables/nyctaxidata/stage1_spark/yellowdata/yellow_tripdata
dbfs:/FileStore/tables/nyctaxidata/stage1_spark/yellow_green_commondata/ratecodes
dbfs:/FileStore/tables/nyctaxidata/stage1_spark/yellow_green_commondata/paymenttypes
dbfs:/FileStore/tables/nyctaxidata/stage1_spark/yellow_green_commondata/taxizones

dbfs:/FileStore/tables/nyctaxidata/stage1_spark/fhvdata/fhv_tripdata
dbfs:/FileStore/tables/nyctaxidata/stage1_spark/fhvdata/fhv_bases
dbfs:/FileStore/tables/nyctaxidata/stage1_spark/fhvdata/taxizones

Merged Table location (Stage2)

dbfs:/FileStore/tables/nyctaxidata/stage2_spark/

Processed/Output Files locations (Stage3)

dbfs:/FileStore/tables/nyctaxidata/stage3_spark/

FILE LOCATION FOR STRUCTURED STREAMING

dbfs:/FileStore/tables/nyctaxidata/streaming/yellowdata/yellow_tripdata

COMMANDS TO WORK WITH FILES IN DBFS

https://docs.databricks.com/dbfs/index.html

https://docs.databricks.com/files/index.html

Command to check the Contents of file in DBFS

```
dbutils.fs.head("dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_05.csv ")
```

```
%fs head
dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_05.csv
```

Command to remove file from DBFS

```
dbutils.fs.rm("/FileStore/tables/your_table_name.csv")
```

Command to check DBFS root

# Default location for %fs is root

```
%fs ls /tmp/
%fs mkdirs /tmp/my_cloud_dir
%fs cp /tmp/test_dbfs.txt /tmp/file_b.txt
```

# Default location for dbutils.fs is root

```
dbutils.fs.ls ("/tmp/")
dbutils.fs.put("/tmp/my_new_file", "This is a file in cloud storage.")
```

# Default location for %sh is the local filesystem

```
%sh ls /dbfs/tmp/
```

# Default location for os commands is the local filesystem

```
import os
os.listdir('/dbfs/tmp')
```

# With `%fs` and `dbutils.fs`, you must use file:/ to read from local filesystem

----

```
%fs ls file:/tmp

%fs mkdirs file:/tmp/my_local_dir

dbutils.fs.ls ("file:/tmp/")

dbutils.fs.put("file:/tmp/my_new_file", "This is a file on the
local driver node.")
```

# %sh reads from the local filesystem by default

```
%sh ls /tmp
```

Distributed Data processing using Spark Core (RDD)

Working with yellowtrip data set

Sample Data: yellow_tripdata_2019-05.csv/ yellow_tripdata_2019-06.csv

VendorID	1	attribute[0]
tpep_pickup_datetime	2019-05-01 00:14:50	attribute[1]
tpep_dropoff_datetime	2019-05-01 00:16:48	attribute[2]
passenger_count	1	attribute[3]
trip_distance	.00	attribute[4]
RatecodeID	1	attribute[5]
store_and_fwd_flag	N	attribute[6]
PULocationID	145	attribute[7]
DOLocationID	145	attribute[8]
payment_type	2	attribute[9]
fare_amount	3	attribute[10]
extra	0.5	attribute[11]
mta_tax	0.5	attribute[12]
tip_amount	0	attribute[13]
tolls_amount	0	attribute[14]
improvement_surcharge	0.3	attribute[15]
total_amount	4.3	attribute[16]
congestion_surcharge	0	attribute[17]

Problem Statements:-

(Develop Notebook applications using Spark RDD API to generate reports as per the problem statements)

1. Ingest All Yellow Trip data Files into DBFS Landing location

   /FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata

2. Create RDD to store valid yellow trip records for June Month of Year 2019 (Filter conditions to be checked)
   a. tpep_pickup_datetime and tpep_dropoff_datetime should not be same
   b. passenger_count should not be zero
   c. trip_distance should not be zero
   d. fare_amount should not be zero
   e. total_amount should not be zero

Filtering Yellow Trip Data sets

Sample Data: yellow_tripdata_2019-05.csv/ yellow_tripdata_2019-06.csv

VendorID	1	attribute[0]
tpep_pickup_datetime	2019-05-01 00:14:50	attribute[1]
tpep_dropoff_datetime	2019-05-01 00:16:48	attribute[2]
passenger_count	1	attribute[3]
trip_distance	.00	attribute[4]
RatecodeID	1	attribute[5]
store_and_fwd_flag	N	attribute[6]
PULocationID	145	attribute[7]
DOLocationID	145	attribute[8]
payment_type	2	attribute[9]
fare_amount	3	attribute[10]
extra	0.5	attribute[11]
mta_tax	0.5	attribute[12]
tip_amount	0	attribute[13]
tolls_amount	0	attribute[14]
improvement_surcharge	0.3	attribute[15]
total_amount	4.3	attribute[16]
congestion_surcharge	0	attribute[17]

3. Create an application to get the total valid trip count of yellow taxi for each Vendor for June month
4. Create an application to find total trip cost for each yellow taxi Vendor for June month for all valid trips

5. Create an application to get the report of total passenger count for each yellow taxi Vendor for June month for all valid trips

6. Create an application to generate the report of total trip distance for each yellow taxi Vendor for June month for all valid trips

7. Create an application to generate the report of **total passenger count for each day for June month (Consider pickup_datetime)** for all valid trips

VendorID,**tpep_pickup_datetime**,tpep_dropoff_datetime,passenger_count,trip_distance,RatecodeID,store_and_fwd_flag,PULocationID,DOLocationID,payment_type,fare_amount,extra,mta_tax,tip_amount,tolls_amount,improvement_surcharge,total_amount,congestion_surcharge

1,**2019-01-01 00:46:40**,2019-01-01 00:53:20,1,1.50,1,N,151,239,1,7,0.5,0.5,1.65,0,0.3,9.95,

8. Create an application to generate the report of total passenger count for each Vendor and each day of June month (Consider pickup_datetime) for all valid trips

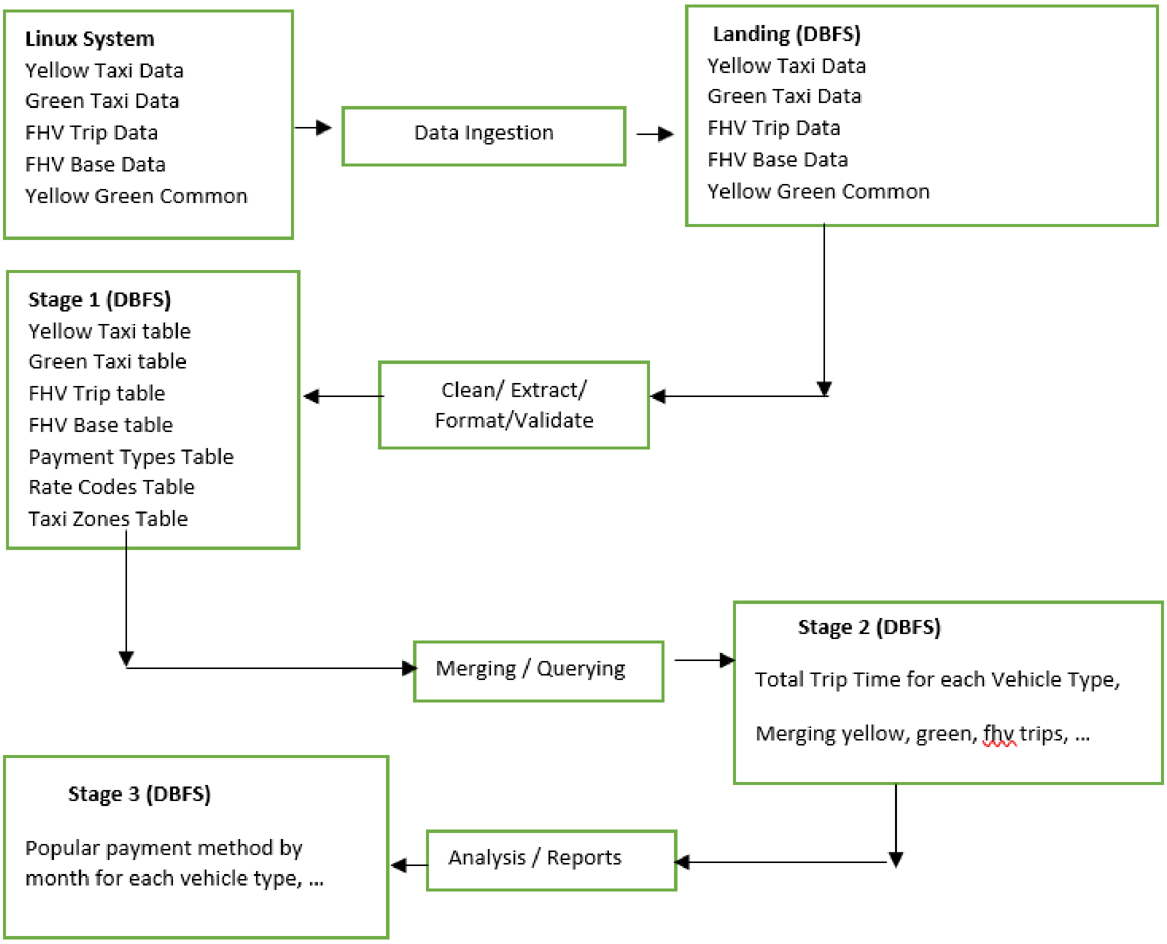
9. Save the previous results into DBFS location

```
dbfs:/FileStore/tables/nyctaxidata/stage3_spark/passcount_vendorday
```

10. Check the DBFS location

Distributed Data Processing Using Spark SQL

Dataflow Diagram:



Data Ingestion and Data Preparation

Create separate notebook for each problem statement

1. Ingest all New York Taxi datasets into DBFS Landing locations and check the locations

a. Yellow Trip data sets
b. Green Trip data sets
c. Taxi zones data sets
d. Payment types data sets
e. Rate codes data sets

2. ETL Processing with Yellow Trip Data Sets

a. Create a temporary view “**YellowTripRawDataView**” from Landing location

b. Filter the YellowTripRawDataView data as per the below filter conditions –

1. tpep_pickup_datetime and tpep_pickup_datetime should not be same
2. record will be considered on for year 2019 only
3. passenger_count should not be zero
4. trip_distance should not be zero
5. fare_amount should not be zero
6. total_amount should not be zero
7. PULocationID should not be null
8. DOLocationID should not be null

c. Create a Delta Table “**YellowTripDelta**” from the filtered view with the listed columns –

```
( "VendorID","PickupTime","DropTime","PassengerCount","TripDistance","RatecodeID",
  "PickupLocationId","DropLocationId","payment_type", "fare_amount", “extra",
   "mta_tax","tip_amount","tolls_amount", "improvement_surcharge", "total_amount",
    "congestion_surcharge")
 # Ignore the store_and_fwd_flag  column
```

3. ETL Processing with Green Trip Data Sets

a. Create a temporary view “**GreenTripRawDataView**” from Landing location

b. Filter the GreenTripRawDataView data as per the below filter conditions –

1. lpep_pickup_datetime and lpep_pickup_datetime should not be same
2. record will be considered on for year 2019
3. passenger_count should not be zero
4. trip_distance should not be zero
5. fare_amount should not be zero

6. total_amount should not be zero
7. PULocationID should not be null
8. DOLocationID should not be null

c. Create a Delta Table "**GreenTripDelta**" from the filtered view with the listed columns –

("VendorID","PickupTime","DropTime","PassengerCount","TripDistance","RatecodeID",
"PickupLocationId","DropLocationId","payment_type", "fare_amount", "extra",
"mta_tax","tip_amount","tolls_amount", "improvement_surcharge", "total_amount",
"congestion_surcharge", "ehail_fee", "trip_type")
# Ignore the store_and_fwd_flag column

4. Combine Yellow and Green Trip Data Sets

a. Create a Delta Table "**YellowGreenTripCombineDelta**" combining **YellowTripDelta** and **GreenTripDelta** tables with listed columns –

("VendorID","PickupTime","DropTime","PassengerCount","TripDistance","RatecodeID","PickupLocationId","DropLocationId","payment_type",
"fare_amount","extra","mta_tax","tip_amount","tolls_amount",
"improvement_surcharge","total_amount","congestion_surcharge","taxiType")

Where **taxiType** column value will be "Green" for all green trip records and "Yellow" for all yellow trip records.

5. Working with TaxiZones data sets

a. Create a temporary view "**TaxiZonesRawDataView**" from Landing location
b. Create a Delta Table "**TaxiZonesDelta**" from the **TaxiZonesRawDataView** view with all columns

6. Working with PaymentTypes data sets

a. Create a temporary view "**PaymentTypesRawDataView**" from Landing location
b. Create a Delta Table "**PaymentTypesDelta**" from the **PaymentTypesRawDataView** view with all columns

7. Working with RateCodes data sets

a. Create a temporary view "**RateCodesRawDataView**" from Landing location
b. Create a Delta Table "**RateCodesDelta**" from the **RateCodesRawDataView** view with all columns

Create Report on Yellow Green Combined Data sets

(Use Single Notebook for all reports)

1. Generate Report to get total trip time in hours for each taxi type

2. Generate Report to get taxi-type-wise total trip time in hours for each trip-month

3. Generate Report to get taxi-type-wise total number of passengers for each trip month

4. Generate Report to get taxi-type-wise total number of payments for each payment-type

5. Generate Report to get the total number of light trips for each taxi type (when passenger count for each trip is <=2)

6. Generate Report to get the total number of light trips for each taxi type month-wise (when passenger count for each trip is <=2) and save the results in a delta table "lightTripsTaxiTypeMonWise"

7. Generate Report to get the total number of fully-loaded trips for each taxi type (when passenger count for each trip is >=4)

8. Generate Report to get the total number of fully-loaded trips for each taxi type month-wise (when passenger count for each trip is >=4) and save the results in a delta table "loadedTripsTaxiTypeMonWise"

9. Generate Report to get the total number of midnight trips for each taxi type (when trips happen between 12AM to 4AM)

10. Generate Report to get the total number of midnight trips for each taxi type month-wise(when trips happen between 12AM to 4AM)

Create Report on Yellow Green Combined & Common Data sets

(Use Single Notebook for all reports)

1. Create a Report to get total trip time in hours for each taxi type

2. Create a Report to generate Passenger Count for Each Zone
   (Join **YellowGreenTripCombineDelta** and **TaxiZonesDelta** tables for creating report)
   Display and Save the output in table "**ZonePassCountTable**".

3. Create a database "**nyctaxi**" in Spark SQL

4. Create a Report to generate Total Trip Count Per Zone, TaxiType, TripMonth, VendorID . Display and save the output in table "**ZoneTaxiMonVendorTripCountTable**" under "**nyctaxi**" database.

5. Create a Report to generate Total Trip Time Per Borough,TaxiType,TripMonth,VendorID. Display and save the output in table "**BoroughTaxiMonVendorTripCountTable**" under "**nyctaxi**" database.

6. Create a Report to generate Total Travel Fare Per Service_Zone,TaxiType,TripMonth. Display and Save the output in table "**TotalFareSZoneTaxiMonth**" under "**nyctaxi**" database.

7. Generate report on Total Different Payment Method count for Each taxi type, each tripMonth, each VendorID.
   (Join **YellowGreenTripCombineDelta** and **PaymentTypesDelta** tables for creating report)
   Display and Save the output in table "**PaymentCountTaxiMonthVendor**" under "**nyctaxi**" database.

   Create a Report to generate Total Different Payment Method count for Each Zone, Each taxi type, each tripMonth,each VendorID. Display and Save the output in table "**PaymentCountZoneTaxiMonthVendor**" under "**nyctaxi**" database.

8. Create a report to generate Total Trip count for each vendor for each Zone where "Standard Rate" has been applied. Display and Save the output in table "**StandardRateTripCount**" under "**nyctaxi**" database.

9. Check "**nyctaxi**" database and display the records from each table.

Incremental Data Ingestion with Auto Loader

Incremental ETL is important since it allows us to deal solely with new data that has been encountered since the last ingestion.

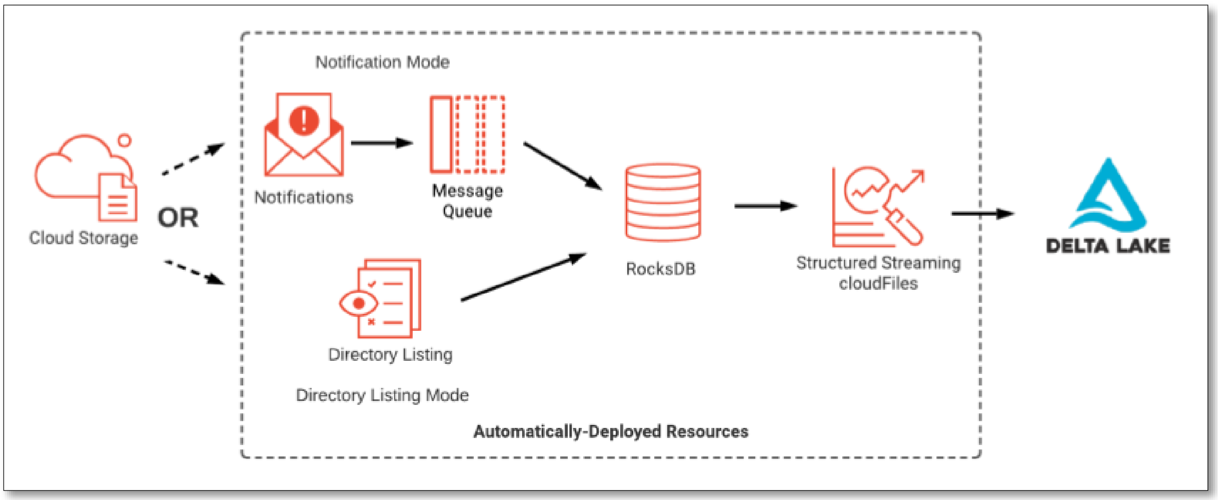
Reliably processing only the new data reduces redundant processing and helps enterprises reliably scale data pipelines.

The first step for any successful data lakehouse implementation is ingesting into a Delta Lake table from cloud storage.

Historically, ingesting files from a data lake into a database has been a complicated process.

Databricks Auto Loader provides an easy-to-use mechanism for incrementally and efficiently processing new data files as they arrive in cloud file storage.

In this tutorial, you'll see Auto Loader in action.



Learning Objectives

By the end of this lesson, you should be able to:

- Execute Auto Loader code to incrementally ingest data from cloud storage to Delta Lake
- Describe what happens when a new file arrives in a directory configured for Auto Loader
- Query a table fed by a streaming Auto Loader query

Dataset Used

This demo uses **yellow trip** datasets from NycTaxi datasets which is in CSV format.

Column List

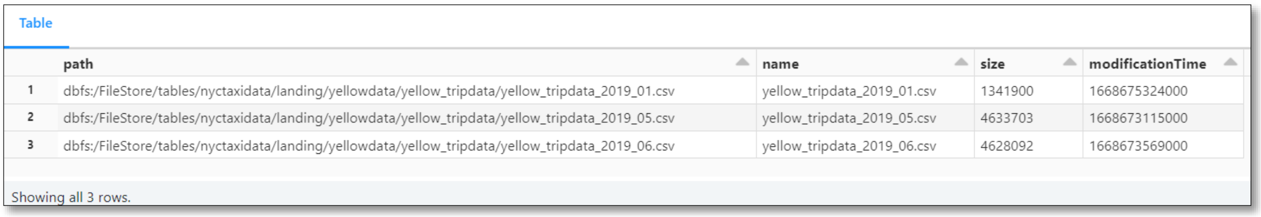
```
VendorID,tpep_pickup_datetime,tpep_dropoff_datetime,passenger_count,trip_distance,RatecodeID,store_and_fwd_flag,PULocationID,DOLocationID,payment_type,fare_amount,extra,mta_tax,tip_amount,tolls_amount,improvement_surcharge,total_amount,congestion_surcharge
```

Getting Started

Note :- All commands to be issued in Databricks Notebook

Run the following cell to check **Yellow** trip data files

```
%fs ls dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata
```

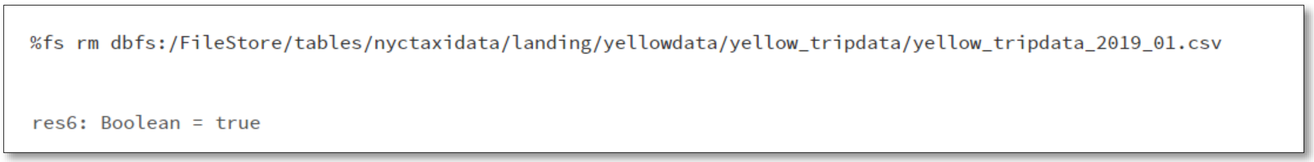


	path	name	size	modificationTime
1	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv	yellow_tripdata_2019_01.csv	1341900	1668675324000
2	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_05.csv	yellow_tripdata_2019_05.csv	4633703	1668673115000
3	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_06.csv	yellow_tripdata_2019_06.csv	4628092	1668673569000

Showing all 3 rows.

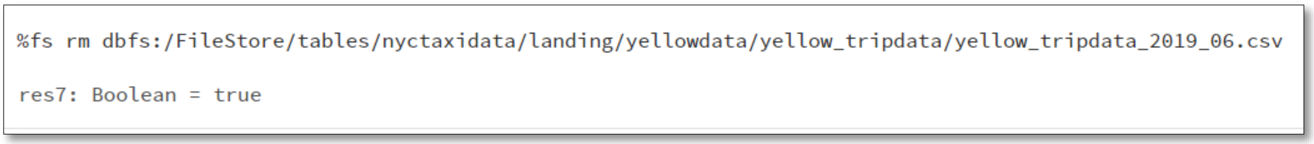
Delete all the files from the location

```
%fs rm dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv
```



```
%fs rm dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv

res6: Boolean = true
```

```
%fs rm dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_06.csv
```



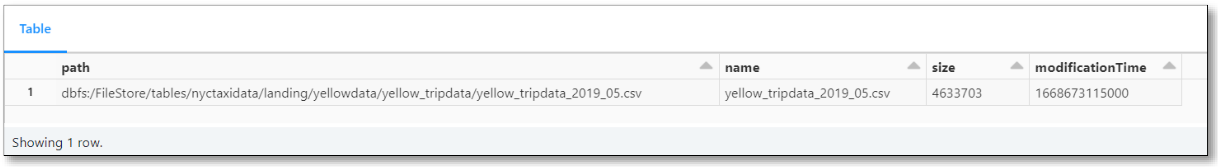
```
%fs rm dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_06.csv

res7: Boolean = true
```

Load yellow_tripdata_2019_05.csv File in DBFS as per the Path Convention

dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata

Check the file in DBFS

```
%fs ls dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata
```



	path	name	size	modificationTime
1	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_05.csv	yellow_tripdata_2019_05.csv	4633703	1668673115000

Showing 1 row.

Using Auto Loader

In the cell below, a function is defined to demonstrate using Databricks Auto Loader with the PySpark API. This code includes both a **Structured Streaming read and write**.

Note that when using Auto Loader with **automatic schema inference and evolution**, the 4 arguments shown here should allow ingestion of most datasets.

These arguments are explained below.

argument	what it is	how it's used
`data_source`	The directory of the source data	Auto Loader will detect new files as they arrive in this location and queue them for ingestion; passed to the **`.load()`** method
`source_format`	The format of the source data	While the format for all Auto Loader queries will be **`cloudFiles`**, the format of the source data should always be specified for the **`cloudFiles.format`** option
`table_name`	The name of the target table	Spark Structured Streaming supports writing directly to Delta Lake tables by passing a table name as a string to the **`.table()`** method. Note that you can either append to an existing table or create a new table
`checkpoint_directory`	The location for storing metadata about the stream	This argument is passed to the **`checkpointLocation`** and **`cloudFiles.schemaLocation`** options. Checkpoints keep track of streaming progress, while the schema location tracks updates to the fields in the source dataset

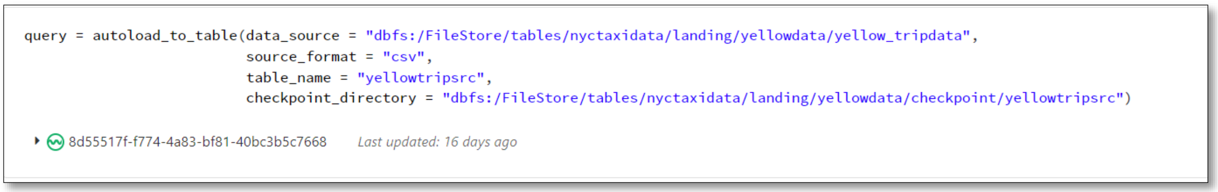
The code below has been streamlined to demonstrate Auto Loader functionality.

```
def autoload_to_table(data_source, source_format, table_name, checkpoint_directory):
    query = (spark.readStream
             .format("cloudFiles")
             .option("cloudFiles.format", source_format)
             .option("cloudFiles.schemaLocation", checkpoint_directory)
             .load(data_source)
             .writeStream
             .option("checkpointLocation", checkpoint_directory)
             .option("mergeSchema", "true")
             .table(table_name))
    return query
```

In the following cell, we use the previously defined function to begin an Auto Loader stream.

Here, we're reading from a source directory of CSV files.

```
query = autoload_to_table(data_source =
"dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata",
                          source_format = "csv",
                          table_name = "yellowtripsrc",
                          checkpoint_directory =
"dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/checkpoint/yellowtripsrc")
```



```
query = autoload_to_table(data_source = "dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata",
                          source_format = "csv",
                          table_name = "yellowtripsrc",
                          checkpoint_directory = "dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/checkpoint/yellowtripsrc")
```

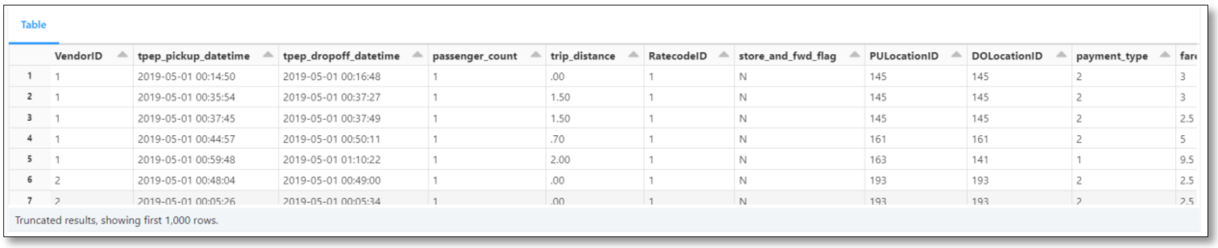
▸ 8d55517f-f774-4a83-bf81-40bc3b5c7668 *Last updated: 16 days ago*

Because Auto Loader uses Spark Structured Streaming to load data incrementally, the code above **doesn't appear to finish executing**.

We can think of this as a **continuously active query**. This means that as soon as new data arrives in our data source, it will be processed through our logic and loaded into our target table.

Query the Target Table Data

```
%sql
SELECT * FROM yellowtripsrc
```

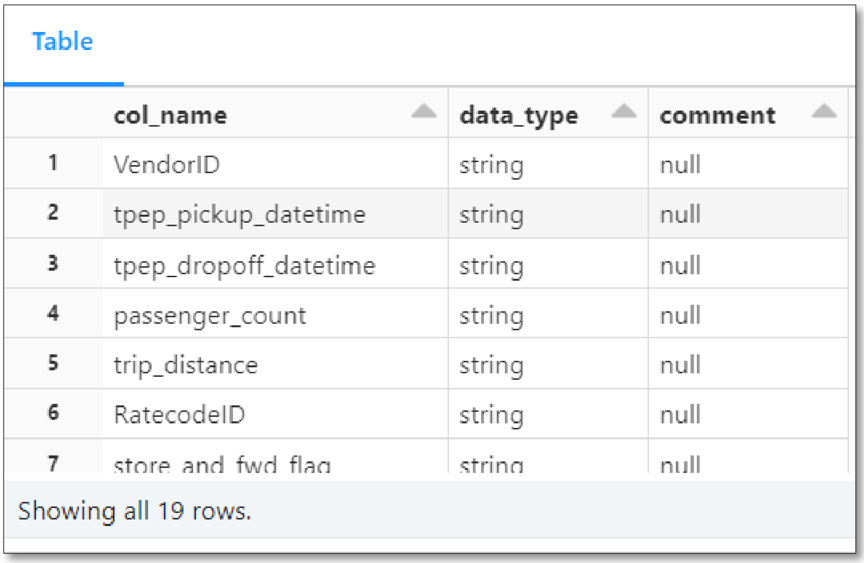


Table

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment_type	far
1	1	2019-05-01 00:14:50	2019-05-01 00:16:48	1	.00	1	N	145	145	2	3
2	1	2019-05-01 00:35:54	2019-05-01 00:37:27	1	1.50	1	N	145	145	2	3
3	1	2019-05-01 00:37:45	2019-05-01 00:37:49	1	1.50	1	N	145	145	2	2.5
4	1	2019-05-01 00:44:57	2019-05-01 00:50:11	1	.70	1	N	161	161	2	5
5	1	2019-05-01 00:59:48	2019-05-01 01:10:22	1	2.00	1	N	163	141	1	9.5
6	2	2019-05-01 00:48:04	2019-05-01 00:49:00	1	.00	1	N	193	193	2	2.5
7	2	2019-05-01 00:05:26	2019-05-01 00:05:34	1	.00	1	N	193	193	2	2.5

Truncated results, showing first 1,000 rows.

```
%sql
DESCRIBE TABLE yellowtripsrc
```



Table

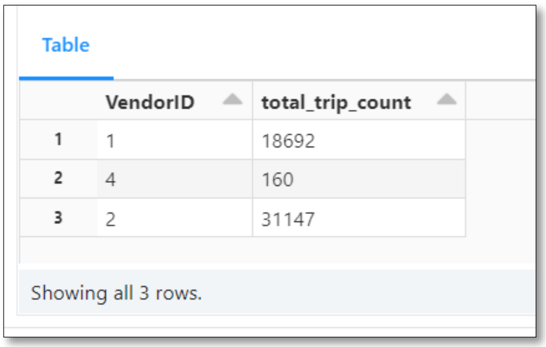
	col_name	data_type	comment
1	VendorID	string	null
2	tpep_pickup_datetime	string	null
3	tpep_dropoff_datetime	string	null
4	passenger_count	string	null
5	trip_distance	string	null
6	RatecodeID	string	null
7	store_and_fwd_flag	string	null

Showing all 19 rows.

Use the cell below to define a temporary view that summarizes the recordings in our target table.

We'll use this view below to demonstrate how new data is automatically ingested with Auto Loader.

```
%sql
CREATE OR REPLACE TEMP VIEW vendor_trip_count AS
  SELECT VendorID, count(*) total_trip_count
  FROM yellowtripsrc
  GROUP BY VendorID;

SELECT * FROM vendor_trip_count
```



Table

	VendorID	total_trip_count
1	1	18692
2	4	160
3	2	31147

Showing all 3 rows.

Land New Data

Load another Yellow trip data file **yellow_tripdata_2019_06.csv** in the path

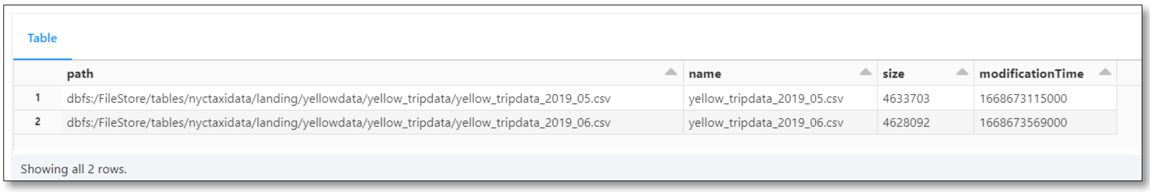
dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata

As mentioned previously, Auto Loader is configured to incrementally process files from a directory in cloud object storage into a Delta Lake table.

We have configured and are currently executing a query to process CSV files from the location specified by `source_path` into a table named `target_table`.

Let's review the contents of the `source_path` directory.
List the contents of the `source_path` again using the cell below. You should see an additional CSV file .

```
%fs ls dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata
```

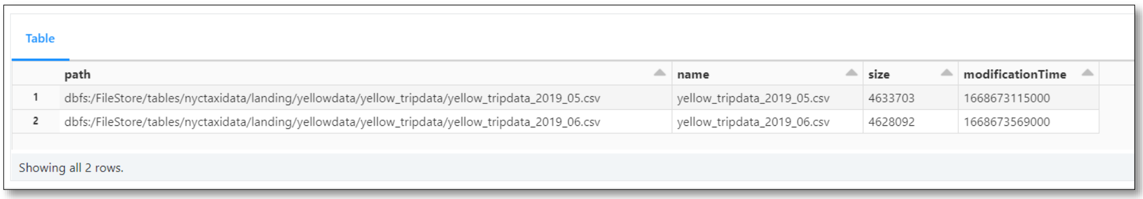


Table

	path	name	size	modificationTime
1	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_05.csv	yellow_tripdata_2019_05.csv	4633703	1668673115000
2	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_06.csv	yellow_tripdata_2019_06.csv	4628092	1668673569000

Showing all 2 rows.

```
files =
dbutils.fs.ls(f"dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata
/yellow_tripdata")

display(files)
```



Table

	path	name	size	modificationTime
1	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_05.csv	yellow_tripdata_2019_05.csv	4633703	1668673115000
2	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_06.csv	yellow_tripdata_2019_06.csv	4628092	1668673569000

Showing all 2 rows.

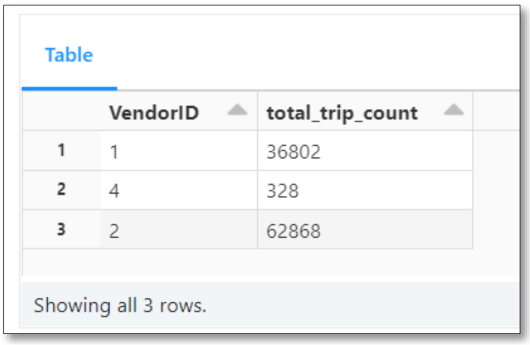
Tracking Ingestion Progress

Historically, many systems have been configured to either reprocess all records in a source directory to calculate current results or require data engineers to implement custom logic to identify new data that's arrived since the last time a table was updated.

With Auto Loader, your table has already been updated.

Run the query below to confirm that new data has been ingested.

```
%sql
SELECT * FROM vendor_trip_count
```



Table

	VendorID	total_trip_count
1	1	36802
2	4	328
3	2	62868

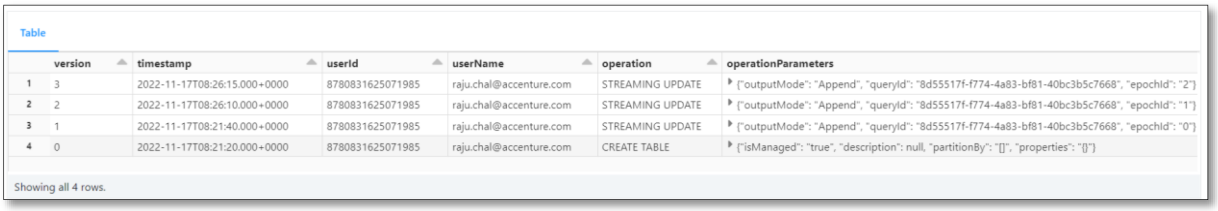
Showing all 3 rows.

The Auto Loader query we configured earlier automatically detects and processes records from the source directory into the target table.

There is a slight delay as records are ingested, but an Auto Loader query executing with default streaming configuration should update results in near real time.

The query below shows the table history.
A new table version should be indicated for each `STREAMING UPDATE`.
These update events coincide with new batches of data arriving at the source.

```
%sql
DESCRIBE HISTORY yellowtripsrc
```



Table

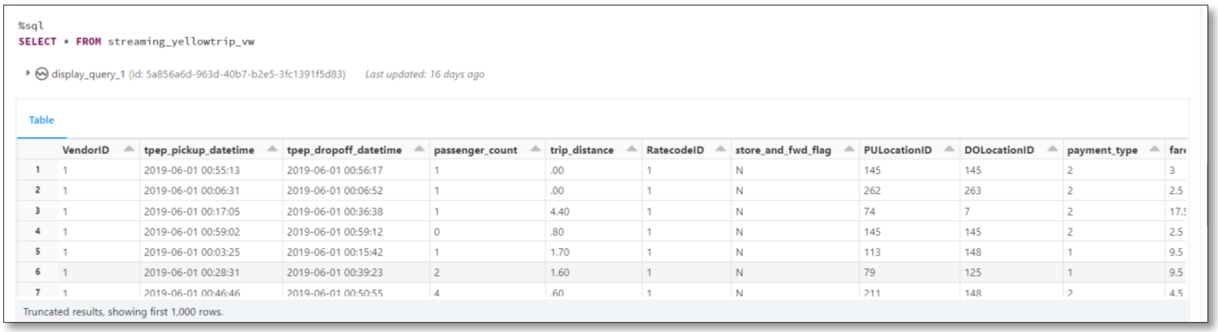
	version	timestamp	userId	userName	operation	operationParameters
1	3	2022-11-17T08:26:15.000+0000	8780831625071985	raju.chal@accenture.com	STREAMING UPDATE	▸ {"outputMode": "Append", "queryId": "8d55517f-f774-4a83-bf81-40bc3b5c7668", "epochId": "2"}
2	2	2022-11-17T08:26:10.000+0000	8780831625071985	raju.chal@accenture.com	STREAMING UPDATE	▸ {"outputMode": "Append", "queryId": "8d55517f-f774-4a83-bf81-40bc3b5c7668", "epochId": "1"}
3	1	2022-11-17T08:21:40.000+0000	8780831625071985	raju.chal@accenture.com	STREAMING UPDATE	▸ {"outputMode": "Append", "queryId": "8d55517f-f774-4a83-bf81-40bc3b5c7668", "epochId": "0"}
4	0	2022-11-17T08:21:20.000+0000	8780831625071985	raju.chal@accenture.com	CREATE TABLE	▸ {"isManaged": "true", "description": null, "partitionBy": "[]", "properties": "{}"}

Showing all 4 rows.

Create a Streaming View

```
(spark.readStream
  .table("yellowtripsrc")
  .createOrReplaceTempView("streaming_yellowtrip_vw"))
```

```
%sql
SELECT * FROM streaming_yellowtrip_vw
```



```
%sql
SELECT * FROM streaming_yellowtrip_vw
```

display_query_1 (id: 5a856a6d-963d-40b7-b2e5-3fc1391f5d83) *Last updated: 16 days ago*

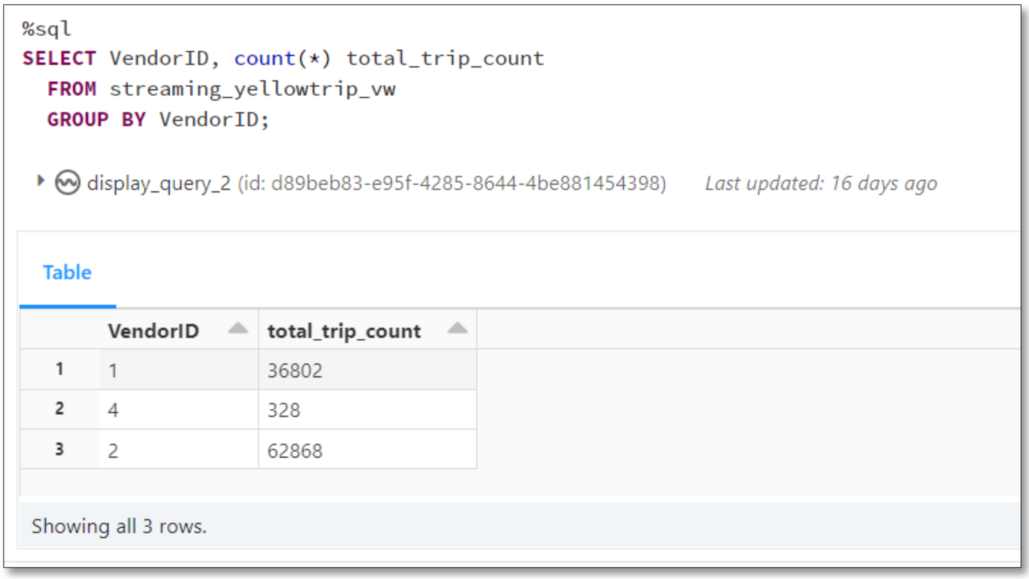
	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment_type	far
1	1	2019-06-01 00:55:13	2019-06-01 00:56:17	1	.00	1	N	145	145	2	3
2	1	2019-06-01 00:06:31	2019-06-01 00:06:52	1	.00	1	N	262	263	2	2.5
3	1	2019-06-01 00:17:05	2019-06-01 00:36:38	1	4.40	1	N	74	7	2	17.5
4	1	2019-06-01 00:59:02	2019-06-01 00:59:12	0	.80	1	N	145	145	2	2.5
5	1	2019-06-01 00:03:25	2019-06-01 00:15:42	1	1.70	1	N	113	148	1	9.5
6	1	2019-06-01 00:28:31	2019-06-01 00:39:23	2	1.60	1	N	79	125	1	9.5
7	1	2019-06-01 00:46:46	2019-06-01 00:50:55	4	.60	1	N	211	148	2	4.5

Truncated results, showing first 1,000 rows.

Cancel the streaming Execution

Find the Total Trip Count for Each Vendor ID

```
%sql
SELECT VendorID, count(*) total_trip_count
 FROM streaming_yellowtrip_vw
 GROUP BY VendorID;
```



```
%sql
SELECT VendorID, count(*) total_trip_count
  FROM streaming_yellowtrip_vw
  GROUP BY VendorID;
```

display_query_2 (id: d89beb83-e95f-4285-8644-4be881454398) *Last updated: 16 days ago*

	VendorID	total_trip_count
1	1	36802
2	4	328
3	2	62868

Showing all 3 rows.

Create another Temp View to store the Total Trip Count for Each Vendor ID

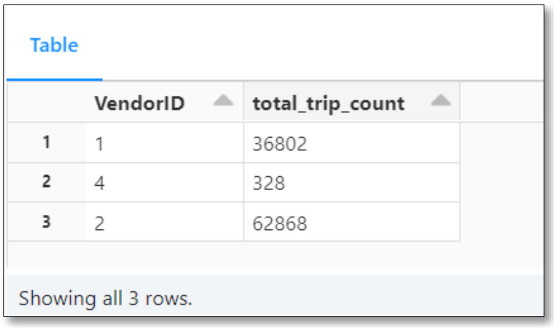
```
%sql
CREATE OR REPLACE TEMP VIEW vendor_trip_count_vw AS
 SELECT VendorID, count(*) total_trip_count
 FROM streaming_yellowtrip_vw
 GROUP BY VendorID;
```

Write the TempView Records in another Table

```
(spark.table("vendor_trip_count_vw")
   .writeStream
   .option("checkpointLocation",
f"dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/checkpoint/vendor_trip_count_table")
   .outputMode("complete")
   .trigger(availableNow=True)
   .table("vendor_trip_count_table")
   .awaitTermination()
# This optional method blocks execution of the next cell until the incremental batch write has
succeeded
)
```

Check the records from the Table

```
%sql
SELECT *
FROM vendor_trip_count_table;
```



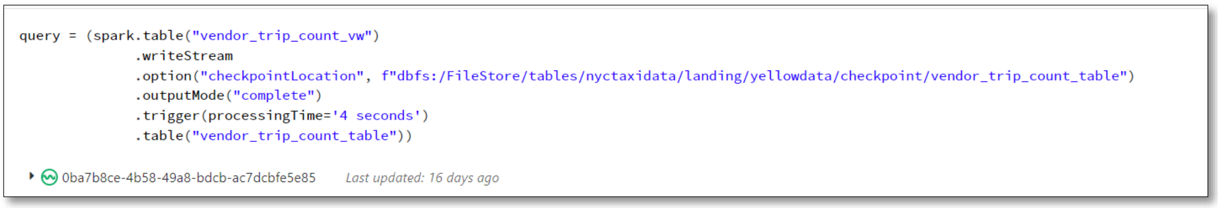
Table

	VendorID	total_trip_count
1	1	36802
2	4	328
3	2	62868

Showing all 3 rows.

Write the TempView Records in another Table at some time interval

```
query = (spark.table("vendor_trip_count_vw")
            .writeStream
            .option("checkpointLocation",
f"dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/checkpoint/vendor_trip_count_table")
            .outputMode("complete")
            .trigger(processingTime='4 seconds')
            .table("vendor_trip_count_table"))
```

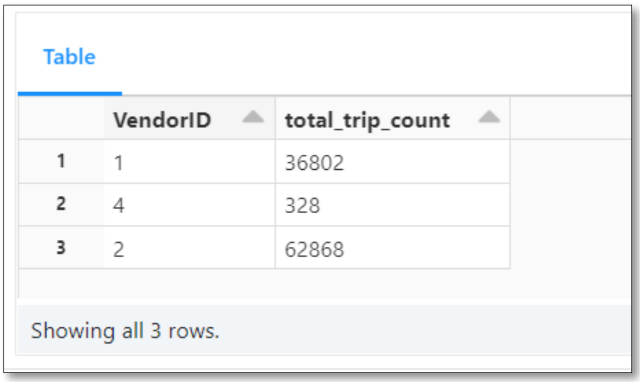


```
query = (spark.table("vendor_trip_count_vw")
            .writeStream
            .option("checkpointLocation", f"dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/checkpoint/vendor_trip_count_table")
            .outputMode("complete")
            .trigger(processingTime='4 seconds')
            .table("vendor_trip_count_table"))
```

0ba7b8ce-4b58-49a8-bdcb-ac7dcbfe5e85 *Last updated: 16 days ago*

Check the records from the Table

```
%sql
SELECT *
FROM vendor_trip_count_table
```



	VendorID	total_trip_count
1	1	36802
2	4	328
3	2	62868

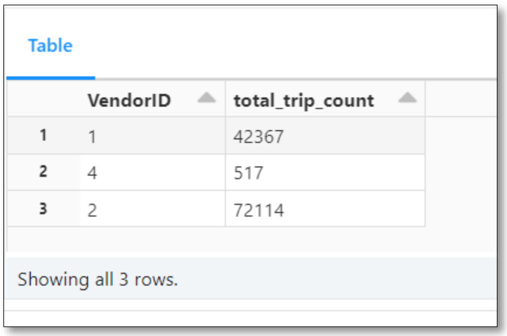
Showing all 3 rows.

Load Another New Yellow Trip Data File in DBFS

Load another Yellow trip data file **yellow_tripdata_2019_01.csv** **in the path**

dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata

```
%sql
SELECT *
FROM vendor_trip_count_table
```



	VendorID	total_trip_count
1	1	42367
2	4	517
3	2	72114

Showing all 3 rows.

Do it Yourself :-

Implement Auto Loader application as per previous activities for Green Trip datasets to generate Total Passenger count for each Green Trip Vendor ID

Incremental Multi-Hop in the Lakehouse

Now that we have a better understanding of how to work with incremental data processing by combining Structured Streaming APIs and Spark SQL, we can explore the tight integration between **Structured Streaming and Delta Lake**.

Learning Objectives

By the end of this lesson, you should be able to:

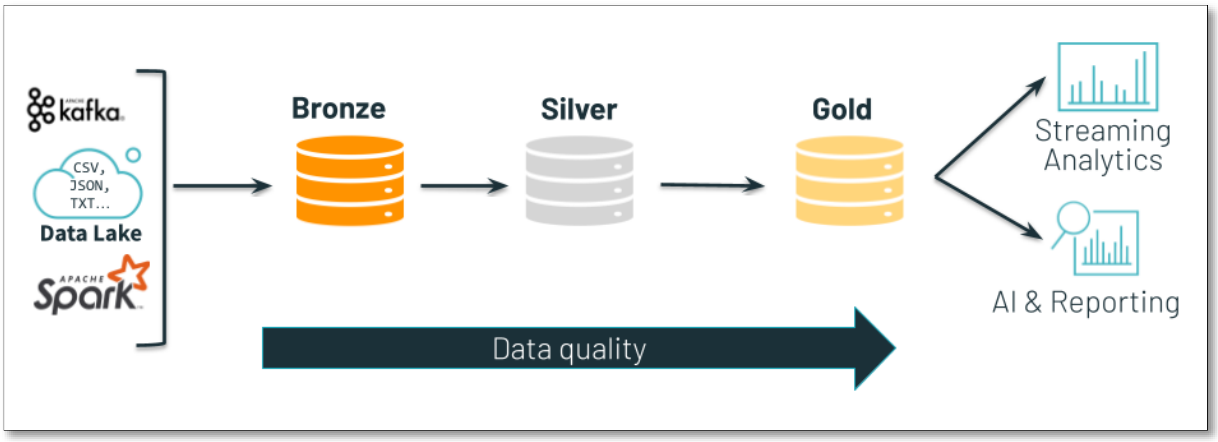
- Describe Bronze, Silver, and Gold tables
- Create a Delta Lake multi-hop pipeline

Incremental Updates in the Lakehouse

Delta Lake allows users to easily combine streaming and batch workloads in a unified multi-hop pipeline.

Each stage of the pipeline represents a state of our data valuable to driving core use cases within the business.

Because all data and metadata lives in object storage in the cloud, multiple users and applications can access data in near-real time, allowing analysts to access the freshest data as it's being processed.



- **Bronze** tables contain raw data ingested from various sources (JSON files, RDBMS data, IoT data, to name a few examples).
- **Silver** tables provide a more refined view of our data. We can join fields from various bronze tables to enrich streaming records, or update account statuses based on recent activity.

- **Gold** tables provide business level aggregates often used for reporting and dashboarding. This would include aggregations such as daily active website users, weekly sales per store, or gross revenue per quarter by department.

The end outputs are actionable insights, dashboards and reports of business metrics.

By considering our business logic at all steps of the ETL pipeline, we can ensure that storage and compute costs are optimized by reducing unnecessary duplication of data and limiting ad hoc querying against full historic data.

Each stage can be configured as a batch or streaming job, and ACID transactions ensure that we succeed or fail completely.

Datasets Used

Use **yellow trip** datasets and **TaxiZones** datasets from NycTaxi datasets which is in CSV format.

The schema of our two datasets is represented below. Note that we will be manipulating these schema during various steps.

Column List of yellow trip datasets

```
VendorID,tpep_pickup_datetime,tpep_dropoff_datetime,passenger_count,trip_distance,RatecodeID,store_and_fwd_flag,PULocationID,DOLocationID,payment_type,fare_amount,extra,mta_tax,tip_amount,tolls_amount,improvement_surcharge,total_amount,congestion_surcharge

1,2019-06-01 00:55:13,2019-06-01 00:56:17,1,.00,1,N,145,145,2,3,0.5,0.5,0,0,0.3,4.3,0
```

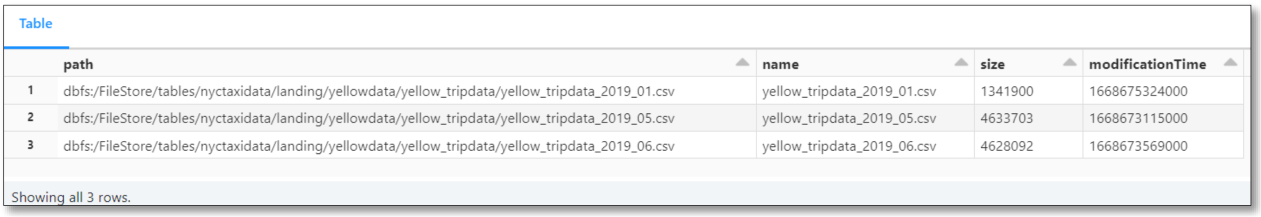
Column List of Taxizones datasets

```
LocationID,Borough,Zone,service_zone

1,"EWR","Newark Airport","EWR"
```

Getting Started

Run the following cell to check **Yellow** trip data files

%fs ls dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata

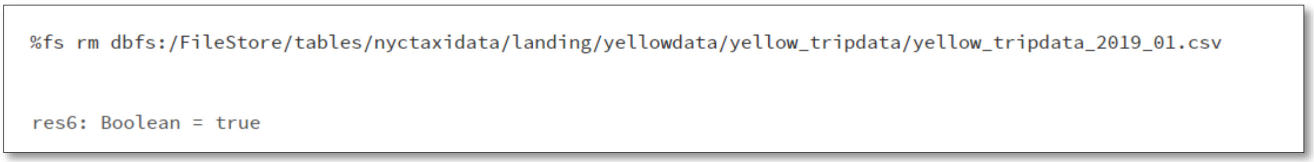


	path	name	size	modificationTime
1	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv	yellow_tripdata_2019_01.csv	1341900	1668675324000
2	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_05.csv	yellow_tripdata_2019_05.csv	4633703	1668673115000
3	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_06.csv	yellow_tripdata_2019_06.csv	4628092	1668673569000

Showing all 3 rows.

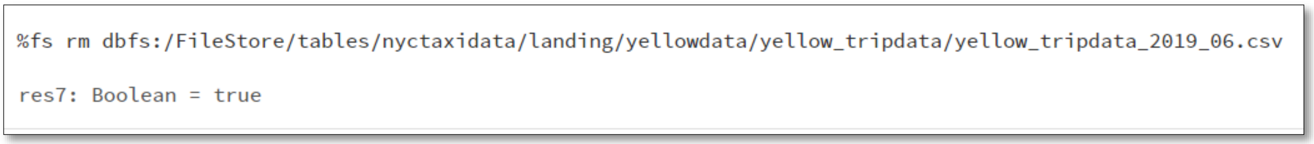
Delete two files from the location

%fs rm dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv



```
%fs rm dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv


res6: Boolean = true
```

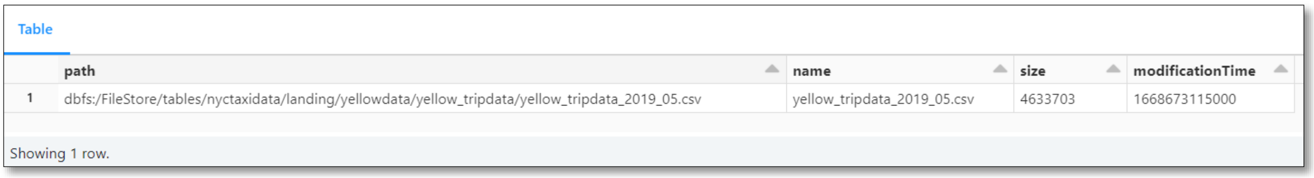
%fs rm dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_06.csv



```
%fs rm dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_06.csv

res7: Boolean = true
```

Check the location again

%fs ls dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata



	path	name	size	modificationTime
1	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_05.csv	yellow_tripdata_2019_05.csv	4633703	1668673115000

Showing 1 row.

Data Simulator

Databricks **Auto Loader** can automatically process files as they land in your cloud object stores.

To simulate this process, we will upload multiple Yellow trip data files in the same directory

dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata

Bronze Table: Ingesting Raw JSON Recordings

Below, we configure a read on a raw CSV source using Auto Loader with schema inference.

Note that while you need to use the Spark DataFrame API to set up an incremental read, once configured you can immediately register a temp view to leverage Spark SQL for streaming transformations on your data.

NOTE:
For a CSV data source, Auto Loader will default to inferring each column as a **string**.

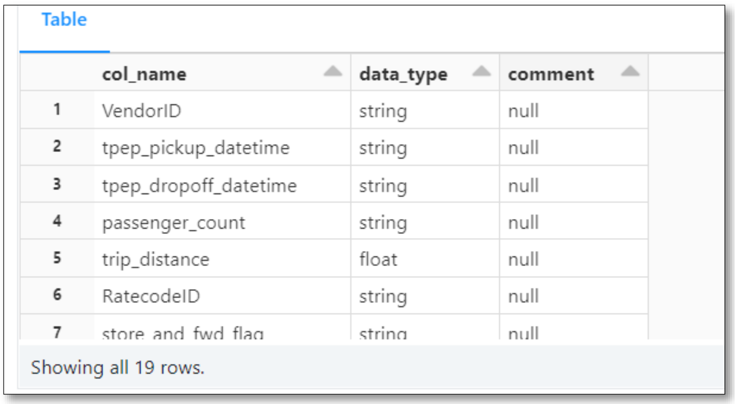
Here, we demonstrate specifying the data type for the trip_distance column using the `cloudFiles.schemaHints` option.

Note that specifying improper types for a field will result in null values.

```
(spark.readStream
   .format("cloudFiles")
   .option("cloudFiles.format", "csv")
   .option("cloudFiles.schemaHints", "trip_distance float")
   .option("cloudFiles.schemaLocation",
f"dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/checkpoint/yellow_bronze")
   .load("dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata")
   .createOrReplaceTempView("yellowtrip_raw_temp"))
```

Check the Schema of yellowtrip_raw_temp view

```
%sql
describe yellowtrip_raw_temp
```



Table

	col_name	data_type	comment
1	VendorID	string	null
2	tpep_pickup_datetime	string	null
3	tpep_dropoff_datetime	string	null
4	passenger_count	string	null
5	trip_distance	float	null
6	RatecodeID	string	null
7	store_and_fwd_flag	string	null

Showing all 19 rows.

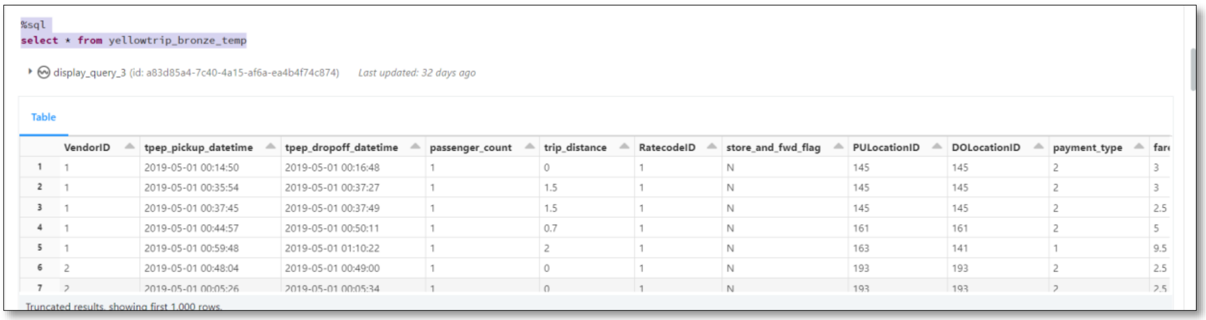
Enrich our RAW data sets

Here, we'll enrich our raw data with additional metadata describing the source file and the time it was ingested.

This additional metadata can be ignored during downstream processing while providing useful information for troubleshooting errors if corrupt data is encountered.

```
%sql
CREATE OR REPLACE TEMPORARY VIEW yellowtrip_bronze_temp AS (
 SELECT *, current_timestamp() receipt_time, input_file_name() source_file
 FROM yellowtrip_raw_temp
)
```

```
%sql
select * from yellowtrip_bronze_temp
```



```
%sql
select * from yellowtrip_bronze_temp
```

display_query_3 (id: a83d85a4-7c40-4a15-af6a-ea4b4f74c874) *Last updated: 32 days ago*

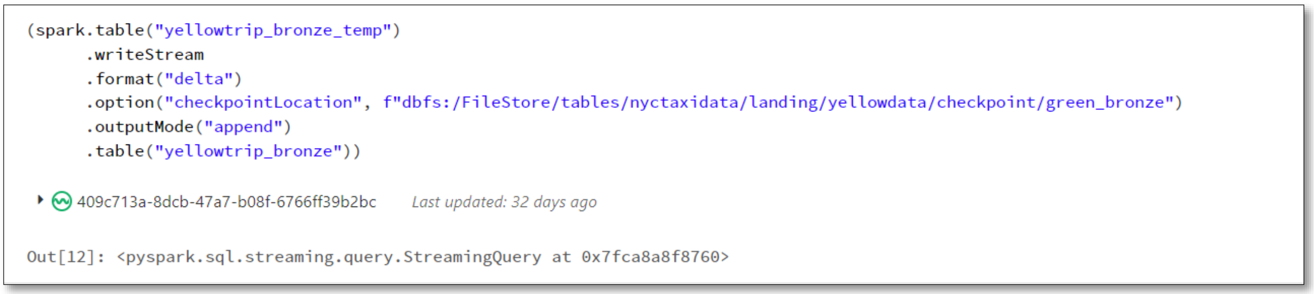
Table

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment_type	fare
1	1	2019-05-01 00:14:50	2019-05-01 00:16:48	1	0	1	N	145	145	2	3
2	1	2019-05-01 00:35:54	2019-05-01 00:37:27	1	1.5	1	N	145	145	2	3
3	1	2019-05-01 00:37:45	2019-05-01 00:37:49	1	1.5	1	N	145	145	2	2.5
4	1	2019-05-01 00:44:57	2019-05-01 00:50:11	1	0.7	1	N	161	161	2	5
5	1	2019-05-01 00:59:48	2019-05-01 01:10:22	1	2	1	N	163	141	1	9.5
6	2	2019-05-01 00:48:04	2019-05-01 00:49:00	1	0	1	N	193	193	2	2.5
7	2	2019-05-01 00:05:26	2019-05-01 00:05:34	1	0	1	N	193	193	2	2.5

Truncated results, showing first 1,000 rows.

The code below passes our enriched raw data back to PySpark API to process an incremental write to a Delta Lake table.

```
(spark.table("yellowtrip_bronze_temp")
    .writeStream
    .format("delta")
    .option("checkpointLocation",
f"dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/checkpoint/green_bronze")
    .outputMode("append")
    .table("yellowtrip_bronze"))
```



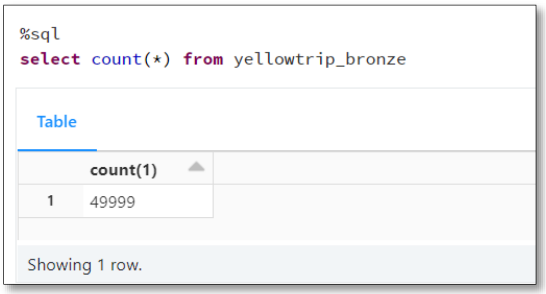
```
(spark.table("yellowtrip_bronze_temp")
      .writeStream
      .format("delta")
      .option("checkpointLocation", f"dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/checkpoint/green_bronze")
      .outputMode("append")
      .table("yellowtrip_bronze"))
```

409c713a-8dcb-47a7-b08f-6766ff39b2bc *Last updated: 32 days ago*

```
Out[12]: <pyspark.sql.streaming.query.StreamingQuery at 0x7fca8a8f8760>
```

Check this stream will be continuously running

```
%sql
select count(*) from yellowtrip_bronze
```



```
%sql
select count(*) from yellowtrip_bronze
```

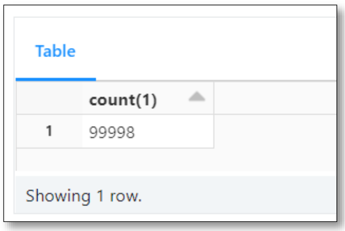
	count(1)
1	49999

Showing 1 row.

Upload another Yellow trip data files in the same directory

dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata
and you'll see the changes immediately detected by the streaming query you've written.

```
%fs ls dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata
%sql
select count(*) from yellowtrip_bronze
```



	count(1)
1	99998

Showing 1 row.

Load Static Lookup Table

The ACID guarantees that Delta Lake brings to your data are managed at the table level, ensuring that only fully successfully commits are reflected in your tables.

If you choose to merge these data with other data sources, be aware of how those sources version data and what sort of consistency guarantees they have.

In this simplified demo, we are loading a static CSV file (**TaxiZones** data file) to add TaxiZones data to our **YellowTrip** data sets.

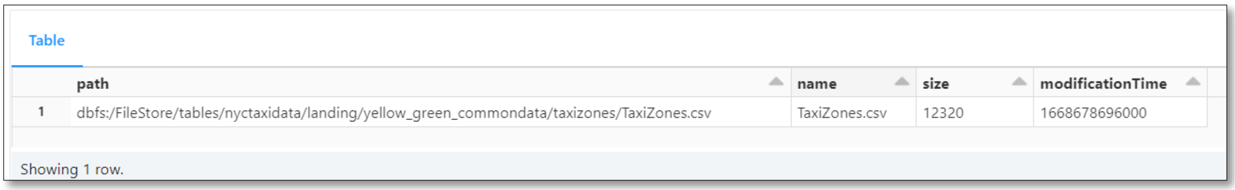
In production, we could use Databricks' Auto Loader feature to keep an up-to-date view of these data in our Delta Lake.

Ingest TaxiZones data sets into DBFS in this location

/FileStore/tables/nyctaxidata/landing/yellow_green_commondata/taxizones/

Check the file

```
%fs ls /FileStore/tables/nyctaxidata/landing/yellow_green_commondata/taxizones/
```



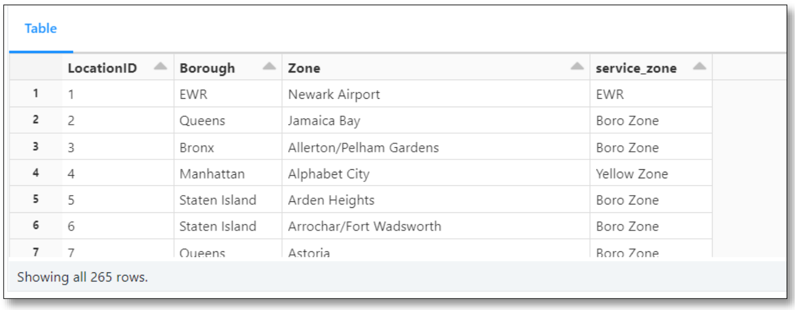
	path	name	size	modificationTime
1	dbfs:/FileStore/tables/nyctaxidata/landing/yellow_green_commondata/taxizones/TaxiZones.csv	TaxiZones.csv	12320	1668678696000

Showing 1 row.

Create View from TaxiZones data set

```
(spark.read
    .format("csv")
    .option("header", True)
    .option("inferSchema", True)
    .load(f"dbfs:/FileStore/tables/nyctaxidata/landing/yellow_green_commondata/taxizones/TaxiZones.csv")
    .createOrReplaceTempView("taxizones_view"))
```

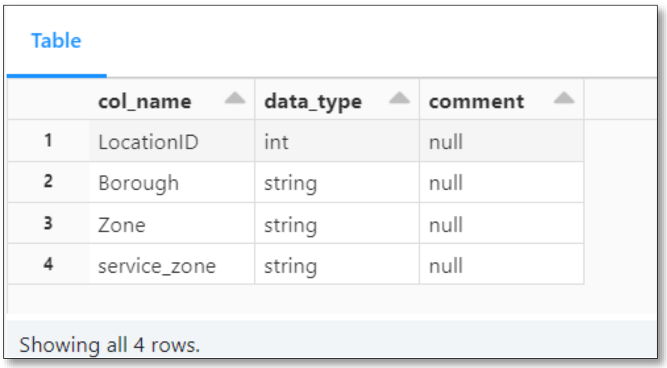
```
%sql
SELECT * FROM taxizones_view
```



	LocationID	Borough	Zone	service_zone
1	1	EWR	Newark Airport	EWR
2	2	Queens	Jamaica Bay	Boro Zone
3	3	Bronx	Allerton/Pelham Gardens	Boro Zone
4	4	Manhattan	Alphabet City	Yellow Zone
5	5	Staten Island	Arden Heights	Boro Zone
6	6	Staten Island	Arrochar/Fort Wadsworth	Boro Zone
7	7	Queens	Astoria	Boro Zone

Showing all 265 rows.

```
%sql
describe taxizones_view
```



	col_name	data_type	comment
1	LocationID	int	null
2	Borough	string	null
3	Zone	string	null
4	service_zone	string	null

Showing all 4 rows.

Silver Table: Enriched Recording Data

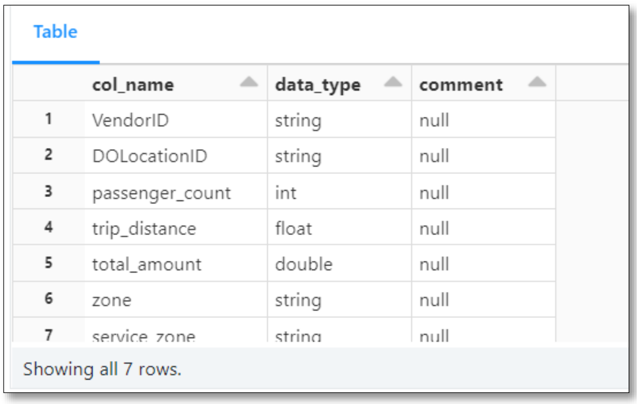
As a second hop in our silver level, we will do the follow enrichments and checks:

- Our **YellowTrip** data will be joined with the **TaxiZones** to add Zone names
- tpep_pickup_datetime!= tpep_dropoff_datetime
- passenger_count !=0
- trip_distance!=0.0
- fare_amount !=0.0
- total_amount!=0.0

```
(spark.readStream
 .table("yellowtrip_bronze")
 .createOrReplaceTempView("yellowtrip_bronze_tmp"))
```

```
%sql
CREATE OR REPLACE TEMPORARY VIEW yellowtrip_taxizones_view AS (
  SELECT y.VendorID,y.DOLocationID,cast(y.passenger_count as
int),y.trip_distance,cast(y.total_amount as double),t.zone,t.service_zone
  FROM yellowtrip_bronze_tmp y
  INNER JOIN taxizones_view t
  ON y.DOLocationID = t.LocationID
  WHERE tpep_pickup_datetime!= tpep_dropoff_datetime and
      cast(y.passenger_count as int)!=0 and
      trip_distance!=0.0 and
      cast(y.total_amount as double)!=0.0)
```

```
%sql
describe yellowtrip_taxizones_view
```

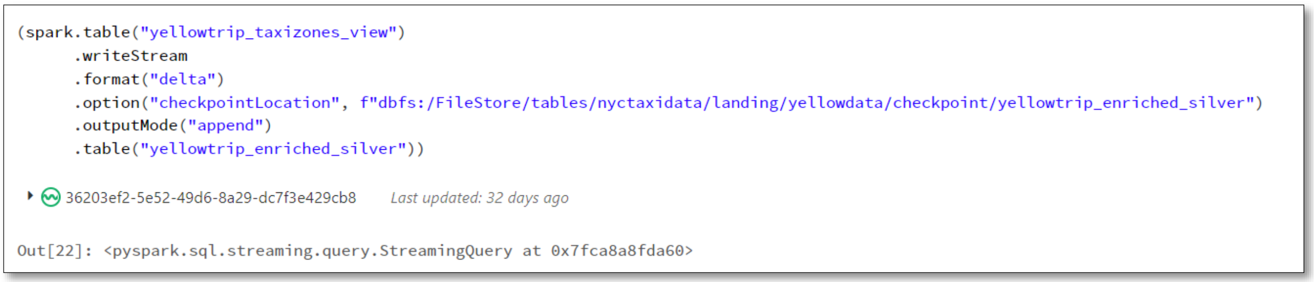


Table

	col_name	data_type	comment
1	VendorID	string	null
2	DOLocationID	string	null
3	passenger_count	int	null
4	trip_distance	float	null
5	total_amount	double	null
6	zone	string	null
7	service_zone	string	null

Showing all 7 rows.

```
(spark.table("yellowtrip_taxizones_view")
    .writeStream
    .format("delta")
    .option("checkpointLocation",
f"dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/checkpoint/yellowtrip_enriched_silver")
    .outputMode("append")
    .table("yellowtrip_enriched_silver"))
```



```
(spark.table("yellowtrip_taxizones_view")
      .writeStream
      .format("delta")
      .option("checkpointLocation", f"dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/checkpoint/yellowtrip_enriched_silver")
      .outputMode("append")
      .table("yellowtrip_enriched_silver"))
```

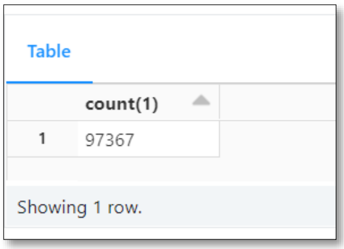
▸ 36203ef2-5e52-49d6-8a29-dc7f3e429cb8   *Last updated: 32 days ago*

```
Out[22]: <pyspark.sql.streaming.query.StreamingQuery at 0x7fca8a8fda60>
```

Check this stream will be continuously running

Check the records count of yellowtrip_enriched_silver delta table

```
%sql
select count(*) from yellowtrip_enriched_silver
```



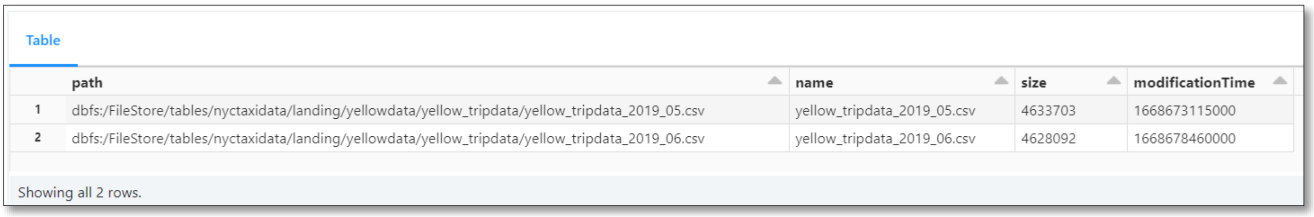
Table

	count(1)
1	97367

Showing 1 row.

Check the file location

%fs ls dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata



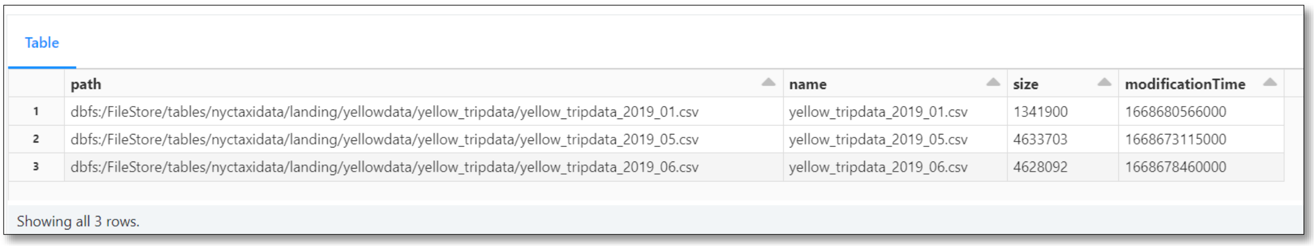
	path	name	size	modificationTime
1	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_05.csv	yellow_tripdata_2019_05.csv	4633703	1668673115000
2	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_06.csv	yellow_tripdata_2019_06.csv	4628092	1668678460000

Showing all 2 rows.

Ingest another Yellow Trip data files into DBFS in the same location dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata

Check the file location

%fs ls dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata

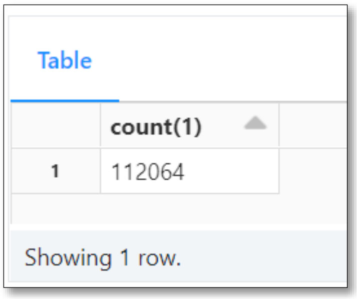


	path	name	size	modificationTime
1	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv	yellow_tripdata_2019_01.csv	1341900	1668680566000
2	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_05.csv	yellow_tripdata_2019_05.csv	4633703	1668673115000
3	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_06.csv	yellow_tripdata_2019_06.csv	4628092	1668678460000

Showing all 3 rows.

Re-check the records count of yellowtrip_enriched_silver delta table

```
%sql
SELECT COUNT(*) FROM yellowtrip_enriched_silver
```



	count(1)
1	112064

Showing 1 row.

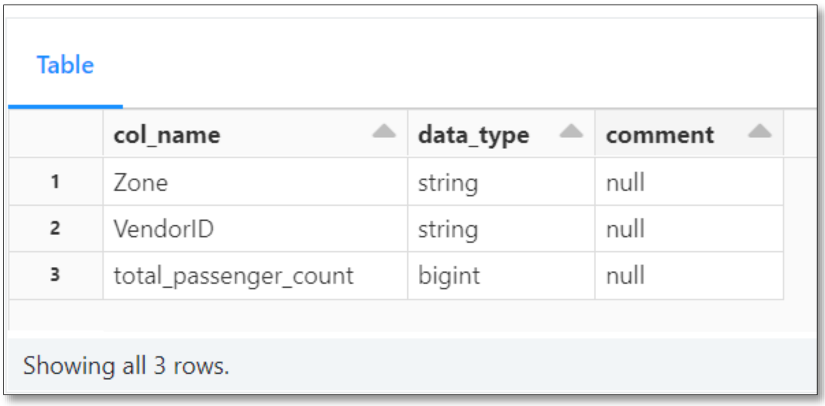
Gold Table: Total Count of Passengers for each Zone, each vendorid

Here we read a stream of data from yellowtrip_enriched_silver and write another stream to create an aggregate gold table of **Total Count of Passengers for each Zone, each vendorid.**

```
(spark.readStream
 .table("yellowtrip_enriched_silver")
 .createOrReplaceTempView("yellowtrip_enriched_silver_temp"))
```

```
%sql
CREATE OR REPLACE TEMP VIEW zone_passenger_count_view AS (
 SELECT Zone, VendorID, sum(passenger_count) as total_passenger_count
 FROM yellowtrip_enriched_silver_temp
 GROUP BY Zone, VendorID)
```

```
%sql
describe zone_passenger_count_view
```



	col_name	data_type	comment
1	Zone	string	null
2	VendorID	string	null
3	total_passenger_count	bigint	null

Showing all 3 rows.

Note that we're using `.trigger(availableNow=True)` below. This provides us the ability to continue to use the strengths of Structured Streaming while triggering this job one-time to process all available data in micro-batches.

To recap, these strengths include:

- exactly once end-to-end fault tolerant processing
- automatic detection of changes in upstream data sources

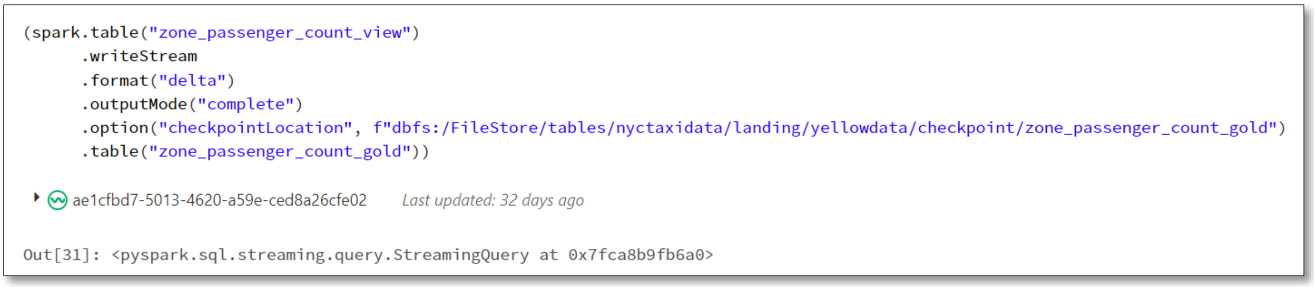
If we know the approximate rate at which our data grows, we can appropriately size the cluster we schedule for this job to ensure fast, cost-effective processing.

The customer will be able to evaluate how much updating this final aggregate view of their data costs and make informed decisions about how frequently this operation needs to be run.

Downstream processes subscribing to this table do not need to re-run any expensive aggregations.

Rather, files just need to be de-serialized and then queries based on included fields can quickly be pushed down against this already-aggregated source.

```
(spark.table("zone_passenger_count_view")
    .writeStream
    .format("delta")
    .outputMode("complete")
    .option("checkpointLocation", f"dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/checkpoint/zone_passenger_count_gold")
    .table("zone_passenger_count_gold"))
```



```
(spark.table("zone_passenger_count_view")
      .writeStream
      .format("delta")
      .outputMode("complete")
      .option("checkpointLocation", f"dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/checkpoint/zone_passenger_count_gold")
      .table("zone_passenger_count_gold"))
```

ae1cfbd7-5013-4620-a59e-ced8a26cfe02   *Last updated: 32 days ago*

```
Out[31]: <pyspark.sql.streaming.query.StreamingQuery at 0x7fca8b9fb6a0>
```

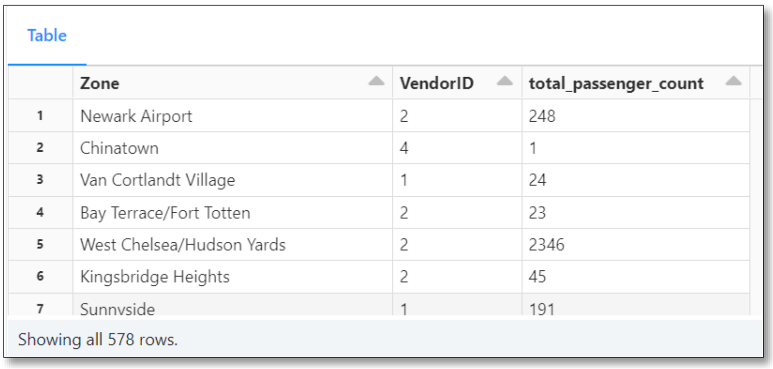
Check this cell command will be <u>continuously</u> running

Important Considerations for complete Output with Delta

When using `complete` output mode, we rewrite the entire state of our table each time our logic runs. While this is ideal for calculating aggregates, we **cannot** read a stream from this directory, as Structured Streaming assumes data is only being appended in the upstream logic.

The gold Delta table we have just registered will perform a static read of the current state of the data each time we run the following query.

```
%sql
select * from zone_passenger_count_gold
```

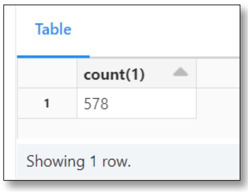


Table

	Zone	VendorID	total_passenger_count
1	Newark Airport	2	248
2	Chinatown	4	1
3	Van Cortlandt Village	1	24
4	Bay Terrace/Fort Totten	2	23
5	West Chelsea/Hudson Yards	2	2346
6	Kingsbridge Heights	2	45
7	Sunnyside	1	191

Showing all 578 rows.

```
%sql
select count(*) from zone_passenger_count_gold
```



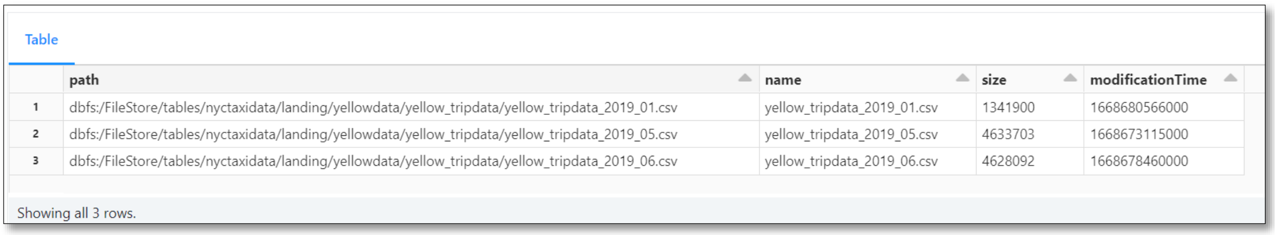
Table

	count(1)
1	578

Showing 1 row.

Check the File Location

```
%fs ls dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata
```



Table

	path	name	size	modificationTime
1	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv	yellow_tripdata_2019_01.csv	1341900	1668680566000
2	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_05.csv	yellow_tripdata_2019_05.csv	4633703	1668673115000
3	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_06.csv	yellow_tripdata_2019_06.csv	4628092	1668678460000

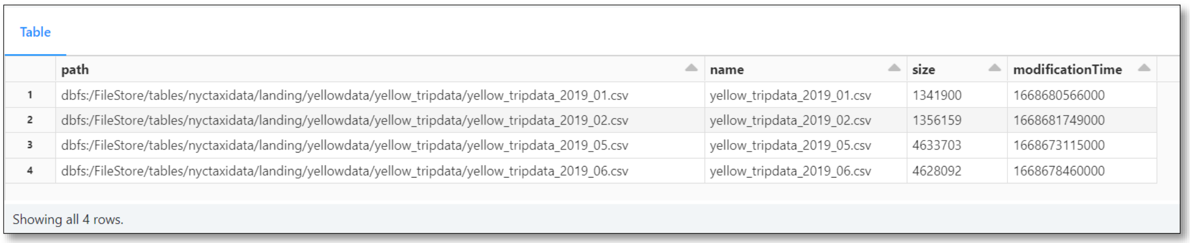
Showing all 3 rows.

Load another YellowTrip data file into DBFS

dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata

Re-Check the File Location

```
%fs ls dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata
```



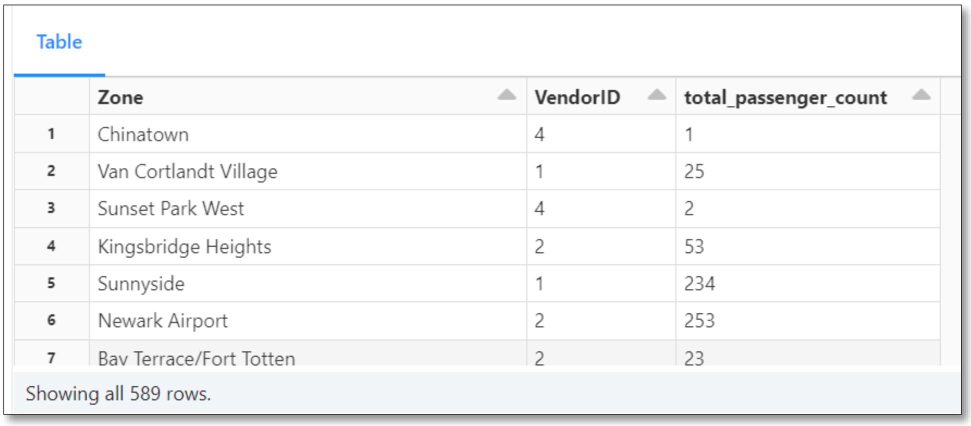
Table

	path	name	size	modificationTime
1	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv	yellow_tripdata_2019_01.csv	1341900	1668680566000
2	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_02.csv	yellow_tripdata_2019_02.csv	1356159	1668681749000
3	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_05.csv	yellow_tripdata_2019_05.csv	4633703	1668673115000
4	dbfs:/FileStore/tables/nyctaxidata/landing/yellowdata/yellow_tripdata/yellow_tripdata_2019_06.csv	yellow_tripdata_2019_06.csv	4628092	1668678460000

Showing all 4 rows.

Check the Delta Table

```
%sql
select * from zone_passenger_count_gold
```



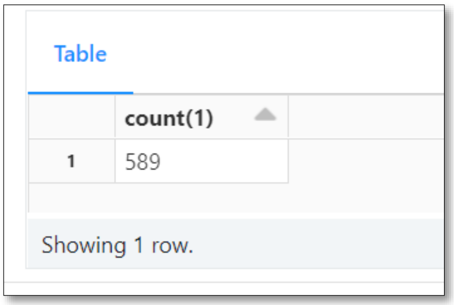
Table

	Zone	VendorID	total_passenger_count
1	Chinatown	4	1
2	Van Cortlandt Village	1	25
3	Sunset Park West	4	2
4	Kingsbridge Heights	2	53
5	Sunnyside	1	234
6	Newark Airport	2	253
7	Bay Terrace/Fort Totten	2	23

Showing all 589 rows.

Check the count from Delta Table after data ingestion

```
%sql
select count(*) from zone_passenger_count_gold
```



Table

	count(1)
1	589

Showing 1 row.

Do it Yourself :-

Generate multi-hop data pipeline using Green Trip datasets and TaxiZones dataset to generate

1. Total Passenger count for each zone and each vendor
2. Total Trip distance for each service_zone and each vendor
3. Total Fare amount for each borough and each vendor

Challenge Yourself

Implementing Delta Live Table

Lab: Migrating SQL Notebooks to Delta Live Tables

Learning Objectives

By the end of this lab, you should be able to:

- **Convert existing data pipelines to Delta Live Tables**

Datasets Used

This demo uses New York taxi data sets.
The schema of our two datasets is represented below.
Note that we will be manipulating these schemas during various steps.

Yellow Trip Data Sets Metadata with Data Type (18 columns)

The main dataset uses records from Yellow trip datasets in the CSV format.

Column Name	*Data type*
VendorID	integer
tpep_pickup_datetime	timestamp
tpep_dropoff_datetime	timestamp
passenger_count	integer
trip_distance	double
RatecodeID	integer
store_and_fwd_flag	string
PULocationID	integer
DOLocationID	integer
payment_type	integer
fare_amount	double
extra	double
mta_tax	double
tip_amount	double
tolls_amount	double
improvement_surcharge	double
total_amount	double
congestion_surcharge	double

These data will later be joined with a static table of Taxi Zones information stored in an external system to identify Zone, Borough and Service_Zone by name.

Taxi Zones Data Set Metadata with Data Type (4 columns)

Column Name	Data type
LocationID	integer
Borough	string
Zone	string
service_zone	string

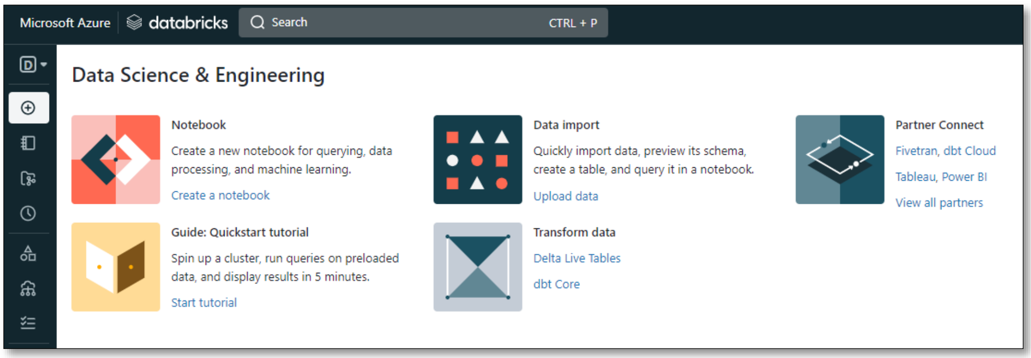
Getting Started

Land Initial Data

Seed the landing zone with more data before proceeding.

You will re-run this command to land additional data later.

Click on “Upload data” under “Data import” menu -



Data Science & Engineering

Notebook

Create a new notebook for querying, data processing, and machine learning.

Data import

Quickly import data, preview its schema, create a table, and query it in a notebook.

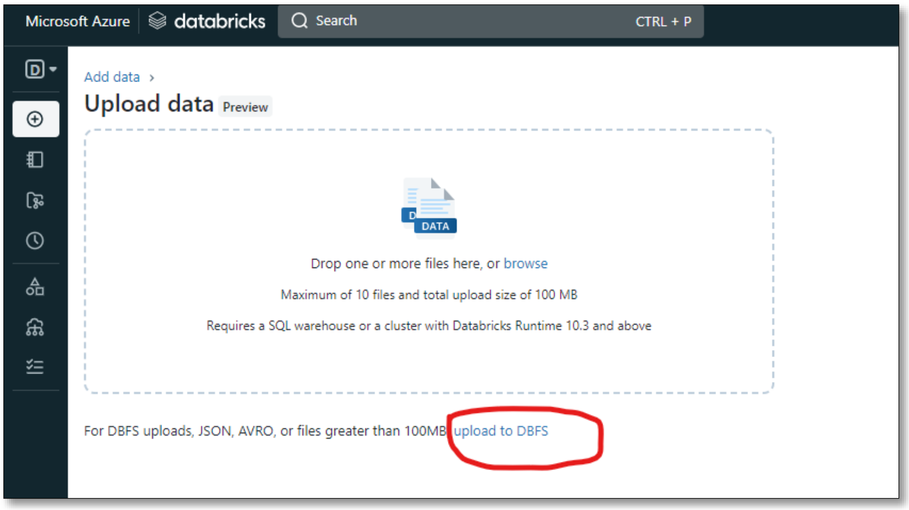
Partner Connect

Guide: Quickstart tutorial

Spin up a cluster, run queries on preloaded data, and display results in 5 minutes.

Transform data

Upload data file



Upload data Preview

Drop one or more files here, or browse

Maximum of 10 files and total upload size of 100 MB

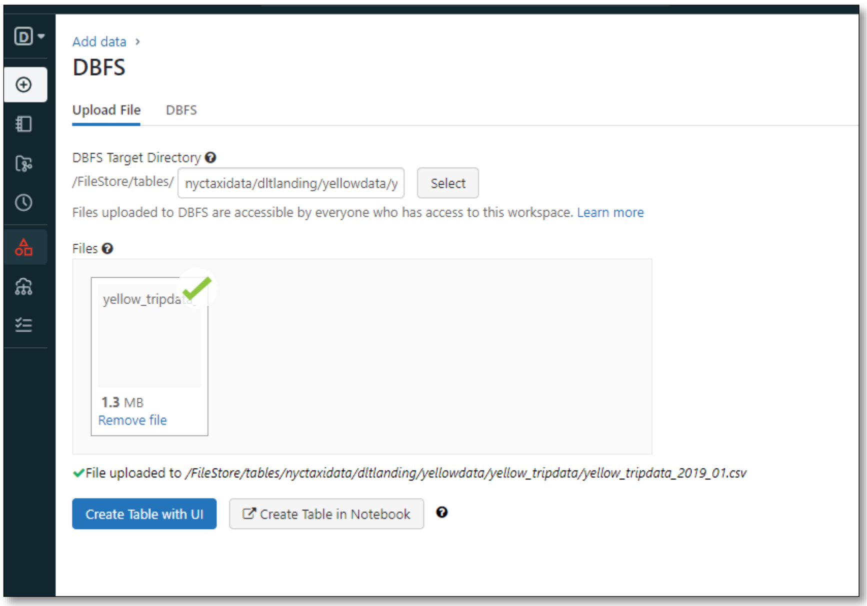
Requires a SQL warehouse or a cluster with Databricks Runtime 10.3 and above

For DBFS uploads, JSON, AVRO, or files greater than 100MB upload to DBFS

Upload Yellow Trip datasets

Landing location of Source Data in DBFS for Delta Live Table

dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata



Add data ›

DBFS

DBFS Target Directory

/FileStore/tables/ nyctaxidata/dltlanding/yellowdata/y

Files uploaded to DBFS are accessible by everyone who has access to this workspace. Learn more

Files

yellow_tripda

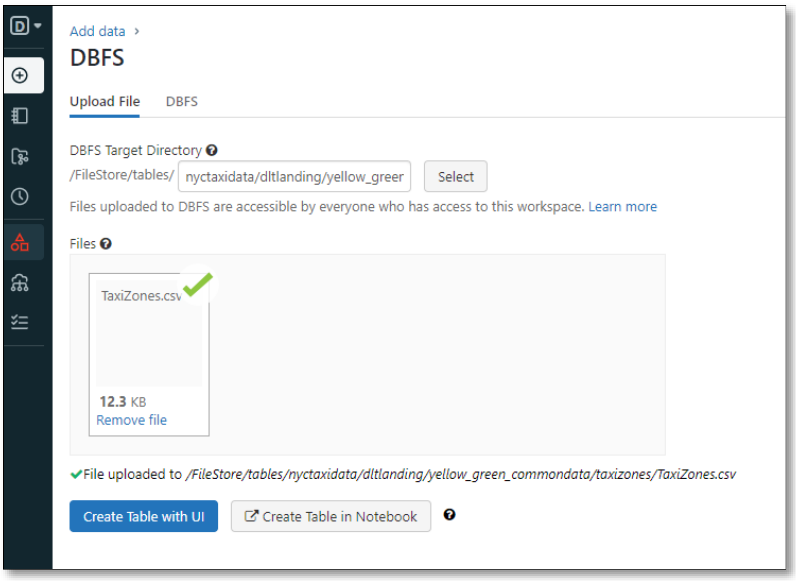
1.3 MB

✓File uploaded to */FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv*

Upload Taxi Zones datasets

Refresh the page
Landing location of Source Data in DBFS for Delta Live Table

dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellow_green_commondata/taxizones/



Add data ›

DBFS

DBFS Target Directory

/FileStore/tables/ nyctaxidata/dltlanding/yellow_greer

Files uploaded to DBFS are accessible by everyone who has access to this workspace. Learn more

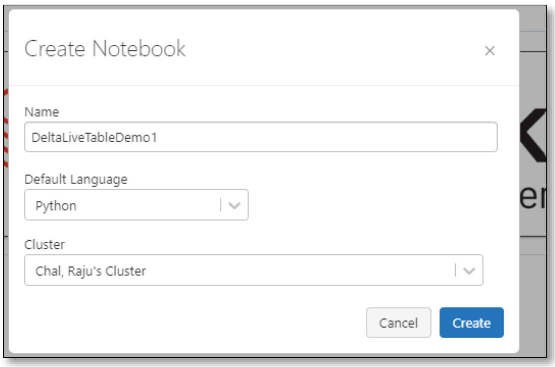
Files

TaxiZones.csv

12.3 KB

✓File uploaded to */FileStore/tables/nyctaxidata/dltlanding/yellow_green_commondata/taxizones/TaxiZones.csv*

Create a New Note Book


Create Notebook

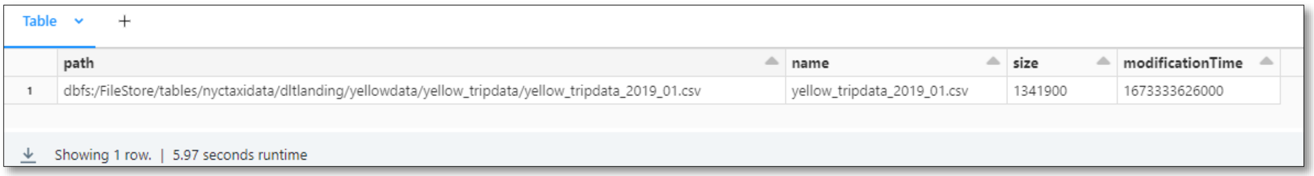
Name: DeltaLiveTableDemo1

Default Language: Python

Cluster: Chal, Raju's Cluster

Execute the following cell to check the files that has been uploaded into DBFS.

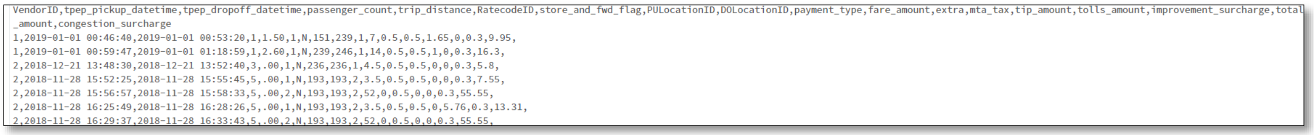
```
%fs ls dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata
```



	path	name	size	modificationTime
1	dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv	yellow_tripdata_2019_01.csv	1341900	1673333626000

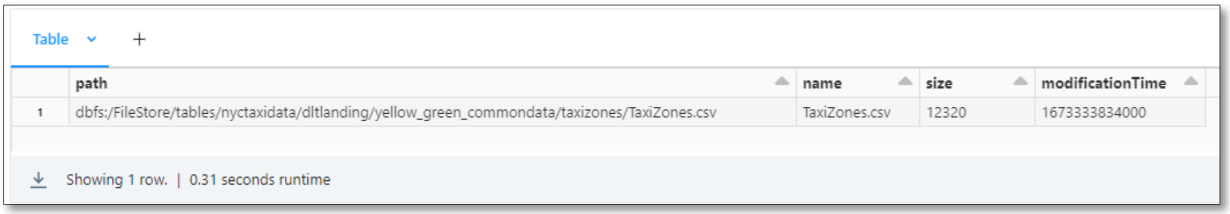
Showing 1 row. | 5.97 seconds runtime

```
%fs head dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv
```



```
VendorID,tpep_pickup_datetime,tpep_dropoff_datetime,passenger_count,trip_distance,RatecodeID,store_and_fwd_flag,PULocationID,DOLocationID,payment_type,fare_amount,extra,mta_tax,tip_amount,tolls_amount,improvement_surcharge,total
_amount,congestion_surcharge
1,2019-01-01 00:46:40,2019-01-01 00:53:20,1,1.50,1,N,151,239,1,7,0.5,0.5,1.65,0,0.3,9.95,
1,2019-01-01 00:59:47,2019-01-01 01:18:59,1,2.60,1,N,239,246,1,14,0.5,0.5,1,0,0.3,16.3,
2,2018-12-21 13:48:30,2018-12-21 13:52:40,3,.00,1,N,236,236,1,4.5,0.5,0.5,0,0,0.3,5.8,
2,2018-11-28 15:52:25,2018-11-28 15:55:45,5,.00,1,N,193,193,2,3.5,0.5,0.5,0,0,0.3,7.55,
2,2018-11-28 15:56:57,2018-11-28 15:58:33,5,.00,2,N,193,193,2,52,0,0.5,0,0,0.3,55.55,
2,2018-11-28 16:25:49,2018-11-28 16:28:26,5,.00,1,N,193,193,2,3.5,0.5,0.5,0,5.76,0.3,13.31,
2,2018-11-28 16:29:37,2018-11-28 16:33:43,5,.00,2,N,193,193,2,52,0,0.5,0,0,0.3,55.55,
```

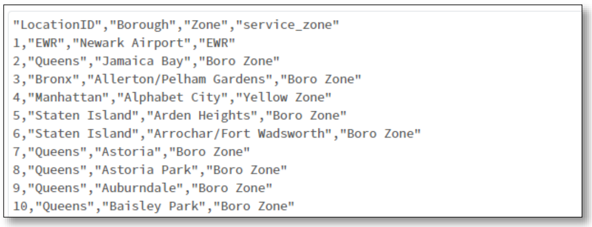
```
%fs ls dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellow_green_commondata/taxizones/
```



	path	name	size	modificationTime
1	dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellow_green_commondata/taxizones/TaxiZones.csv	TaxiZones.csv	12320	1673333834000

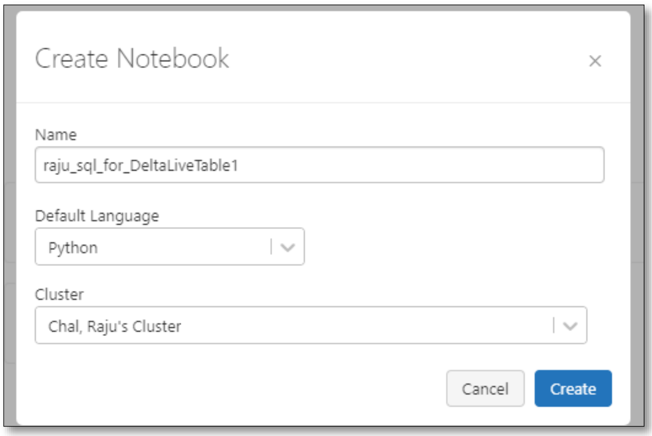
Showing 1 row. | 0.31 seconds runtime

```
%fs head dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellow_green_commondata/taxizones/TaxiZones.csv
```



```
"LocationID","Borough","Zone","service_zone"
1,"EWR","Newark Airport","EWR"
2,"Queens","Jamaica Bay","Boro Zone"
3,"Bronx","Allerton/Pelham Gardens","Boro Zone"
4,"Manhattan","Alphabet City","Yellow Zone"
5,"Staten Island","Arden Heights","Boro Zone"
6,"Staten Island","Arrochar/Fort Wadsworth","Boro Zone"
7,"Queens","Astoria","Boro Zone"
8,"Queens","Astoria Park","Boro Zone"
9,"Queens","Auburndale","Boro Zone"
10,"Queens","Baisley Park","Boro Zone"
```

Create another new notebook



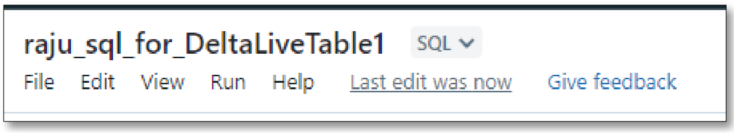
Create Notebook

Name: raju_sql_for_DeltaLiveTable1

Default Language: Python

Cluster: Chal, Raju's Cluster

Make Sure default language in SQL



raju_sql_for_DeltaLiveTable1 SQL

Lab: Migrating a SQL Pipeline to Delta Live Tables

This notebook will be completed by you to implement a DLT pipeline using SQL.

It is **not intended** to be executed interactively, but rather to be deployed as a pipeline once you have completed your changes.

To aid in completion of this Notebook, please refer to the DLT syntax documentation.

File Locations:-

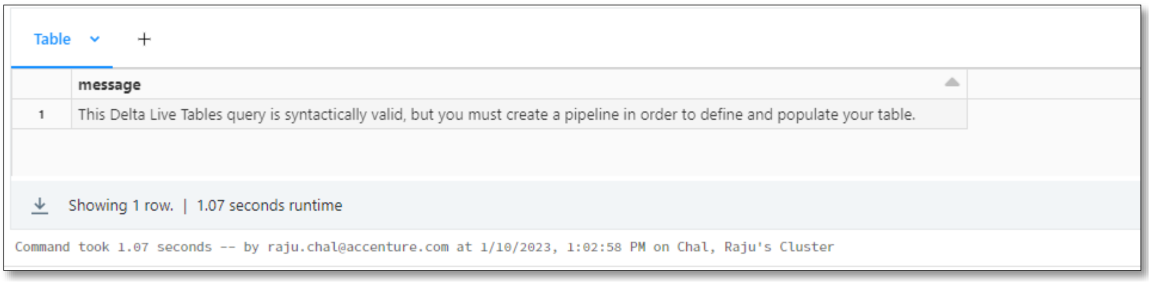
Declare Bronze Table

Declare a bronze table, `yellowTrip_bronze` that ingests CSV data incrementally (using Auto Loader) from the simulated cloud source.
The source location is already supplied as an argument; using this value is illustrated in the cell below.

As we did previously, include two additional columns:

- `receipt_time` that records a timestamp as returned by `current_timestamp()`
- `source_file` that is obtained by `input_file_name()`

```
CREATE OR REFRESH STREAMING LIVE TABLE yellowTrip_bronze
AS SELECT current_timestamp() receipt_time, input_file_name() source_file, *
FROM cloud_files("dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata", "csv",
map("cloudFiles.schemaHints", "passenger_count INTEGER,trip_distance DOUBLE, total_amount
DOUBLE","header", "true", "cloudFiles.inferColumnTypes", "true"))
```



	message
1	This Delta Live Tables query is syntactically valid, but you must create a pipeline in order to define and populate your table.

```
Showing 1 row. | 1.07 seconds runtime
Command took 1.07 seconds -- by raju.chal@accenture.com at 1/10/2023, 1:02:58 PM on Chal, Raju's Cluster
```

Taxi Zones File

Using a similar CTAS syntax, create a live **table** into the CSV data found in the *Taxi Zones* dataset.

To properly configure Auto Loader for this source, you will need to specify the following additional parameters:

option	value
header	true
cloudFiles.inferColumnTypes	true

Auto Loader configurations for CSV can be found here.

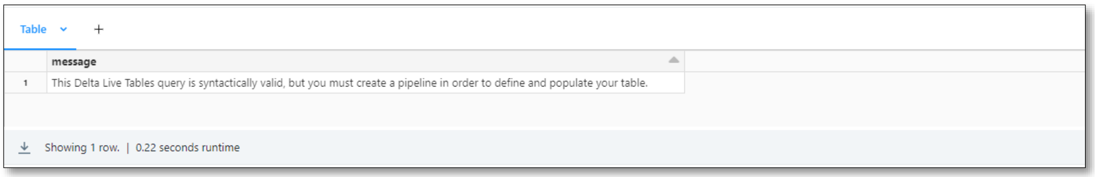
```
CREATE OR REFRESH STREAMING LIVE TABLE taxi_zones
AS SELECT * FROM cloud_files("${datasets_path}/yellow_green_commondata/taxizones", "csv",
map("header", "true", "cloudFiles.inferColumnTypes", "true"))
```

Declare Silver Tables

Our silver table, `yelloTrip_taxiZones_Silver`, will consist of the following fields:

`VendorID,tpep_pickup_datetime,tpep_dropoff_datetime,passenger_count,trip_distance,DOLocationID,total_amount,Borough,Zone,service_zone`

```
CREATE OR REFRESH STREAMING LIVE TABLE yelloTrip_taxiZones_Silver
(CONSTRAINT positive_passenger_count EXPECT (passenger_count > 0) ON VIOLATION DROP ROW,
CONSTRAINT positive_trip_distance EXPECT (trip_distance > 0.0) ON VIOLATION DROP ROW,
CONSTRAINT positive_total_amount EXPECT (total_amount > 0.0) ON VIOLATION DROP ROW,
CONSTRAINT pickup_drop_datetime_not_same EXPECT (pickup_datetime!=dropoff_datetime ) ON
VIOLATION DROP ROW
 )
AS SELECT
a.VendorID,
a.tpep_pickup_datetime pickup_datetime,
a.tpep_dropoff_datetime dropoff_datetime,
CAST(a.passenger_count AS INTEGER) passenger_count,
CAST(a.trip_distance AS DOUBLE) trip_distance,
a.DOLocationID,
CAST(a.total_amount AS DOUBLE) total_amount,
b.Borough,
b.Zone,
b.service_zone
FROM STREAM(live.yellowTrip_bronze) a
INNER JOIN STREAM(live.taxi_zones) b
ON a.DOLocationID = b.LocationID
```



	message
1	This Delta Live Tables query is syntactically valid, but you must create a pipeline in order to define and populate your table.

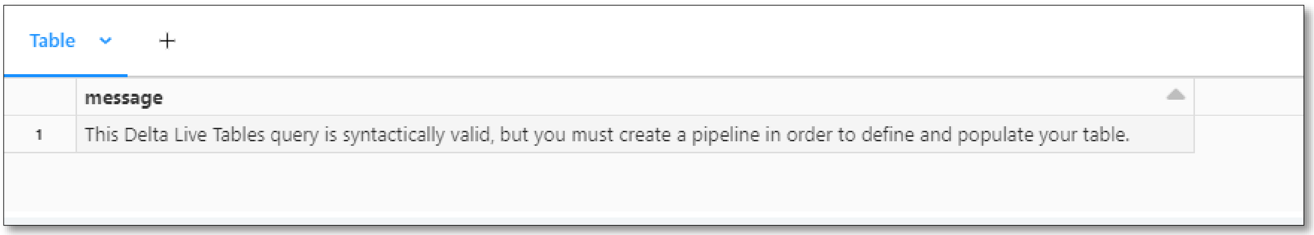
Showing 1 row. | 0.22 seconds runtime

Gold Table

Create a gold table, vendor_zone_passenger_count, that aggregates passenger_count by **`VendorID and Zone`** and delivers the following columns:

1

```
CREATE OR REFRESH LIVE TABLE vendor_zone_passenger_count
COMMENT "Total Passenger count for each yellow taxi vendor for each zone"
AS SELECT VendorID, Zone, sum(passenger_count) total_passenger_count
FROM live.yelloTrip_taxiZones_Silver
GROUP BY VendorID, Zone
```



	message
1	This Delta Live Tables query is syntactically valid, but you must create a pipeline in order to define and populate your table.

Create and Configure a Pipeline

Properties to Configure the Pipeline

Pipeline Name: DLT-Demo-rajuchal-10-jan-2023

Target: raju_chal_dlt_demo_10_jan_2023

Storage Location: dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage

Notebook Path: /Repos/raju.chal@accenture.com/data-engineering-with-databricks/08 - Delta Live Tables/DE 8.1 - DLT/DE 8.1.2 - SQL for Delta Live Tables

Datasets Path: dbfs:/FileStore/tables/nyctaxidata/dltlanding

Create and Configure a Pipeline using Web Interface

In this section you will create a pipeline using a notebook provided with the courseware. We'll explore the contents of the notebook in the following lesson.

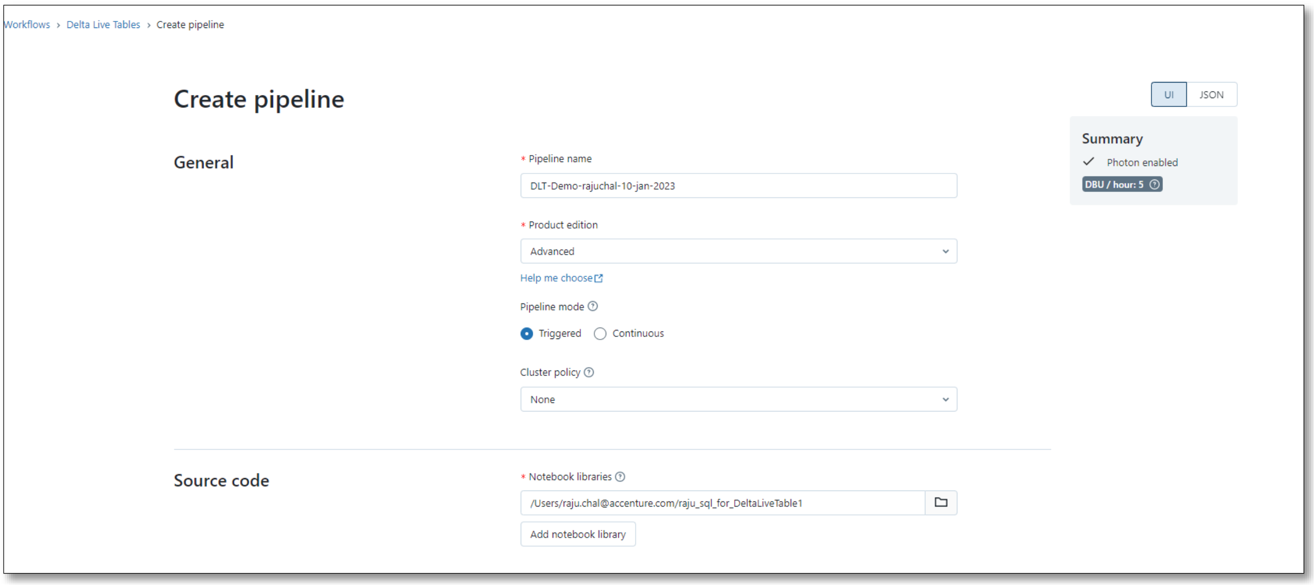
1. Click the **Workflows** button on the sidebar.
2. Select the **Delta Live Tables** tab.
3. Click **Create Pipeline**.
4. Leave **Product Edition** as **Advanced**.
5. Fill in a **Pipeline Name** - because these names must be unique, we suggest using the **Pipeline Name** provided in the cell above.

6. For **Notebook Libraries**, use the navigator to locate and select the notebook specified above.
    - Even though this document is a standard Databricks Notebook, the SQL syntax is specialized to DLT table declarations.
    - We will be exploring the syntax in the exercise that follows.
7. Towards the bottom of the page, there is a drop down titled **Advanced**. Click on that, then:
    - Click **Add configuration**, set the "key" to **spark.master** and the "value" to **local[*]**.
    - Click **Add configuration**, set the "key" to **datasets_path** and the "value" to the value provided in the cell above.
8. In the **Target** field, enter the database name provided in the cell above. This should follow the pattern `<name>_<hash>_dbacademy_dewd_dlt_demo_81`
    - This field is optional; if not specified, then tables will not be registered to a metastore, but will still be available in the DBFS. Refer to the documentation for more information on this option.
9. In the **Storage location** field, enter the path provided in the cell above.
    - This optional field allows the user to specify a location to store logs, tables, and other information related to pipeline execution.
    - If not specified, DLT will automatically generate a directory.
10. For **Pipeline Mode**, select **Triggered**.
    - This field specifies how the pipeline will be run.
    - **Triggered** pipelines run once and then shut down until the next manual or scheduled update.
    - **Continuous** pipelines run continuously, ingesting new data as it arrives. Choose the mode based on latency and cost requirements.
11. Uncheck the **Enable autoscaling** box.
12. Set the number of `workers` to `0` (zero).
    - Along with the **spark.master** config above, this will create a **Single Node** clusters.
13. Check the **Use Photon Acceleration** box.
14. For **Channel**, select **Current**
15. For **Policy**, select the value provided in the cell above.

The fields **Enable autoscaling**, **Min Workers** and **Max Workers** control the worker configuration for the underlying cluster processing the pipeline.

Notice the DBU estimate provided, similar to that provided when configuring interactive clusters.

Finally, click **Create**.



Create pipeline

Summary
✓ Photon enabled
DBU / hour: 5

General

* Pipeline name: DLT-Demo-rajuchal-10-jan-2023

* Product edition: Advanced

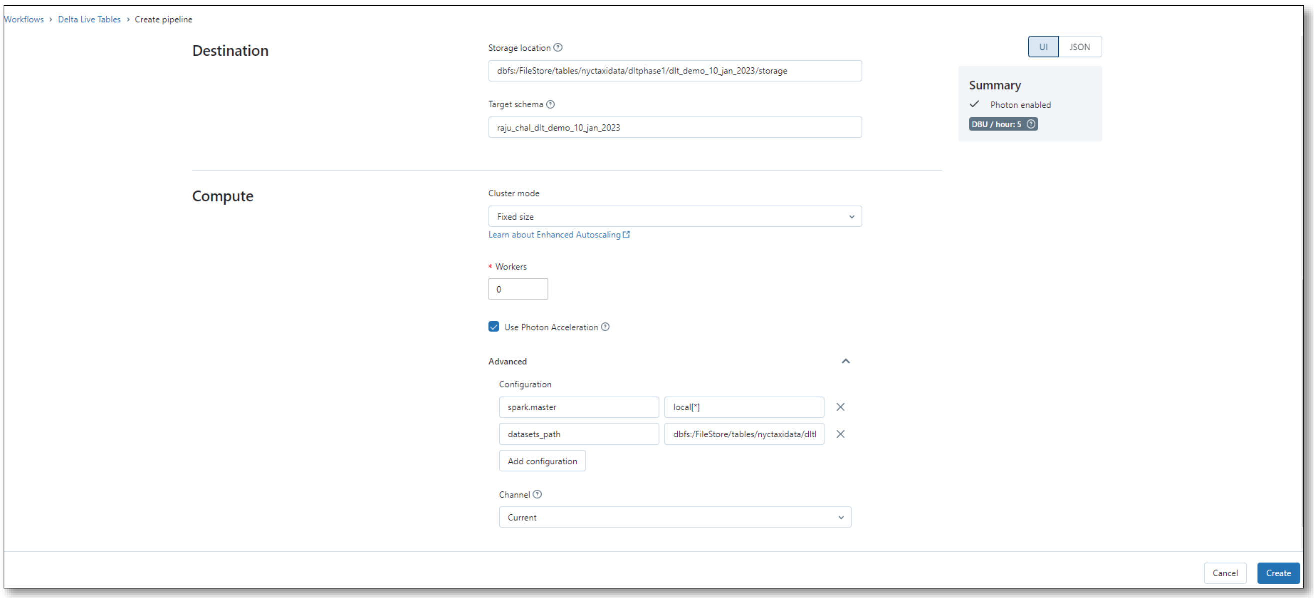
Help me choose

Pipeline mode: ● Triggered ○ Continuous

Cluster policy: None

Source code

* Notebook libraries: /Users/raju.chal@accenture.com/raju_sql_for_DeltaLiveTable1



Summary
✓ Photon enabled
DBU / hour: 5

Destination

Storage location: dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage

Target schema: raju_chal_dlt_demo_10_jan_2023

Compute

Cluster mode: Fixed size

Learn about Enhanced Autoscaling

* Workers: 0

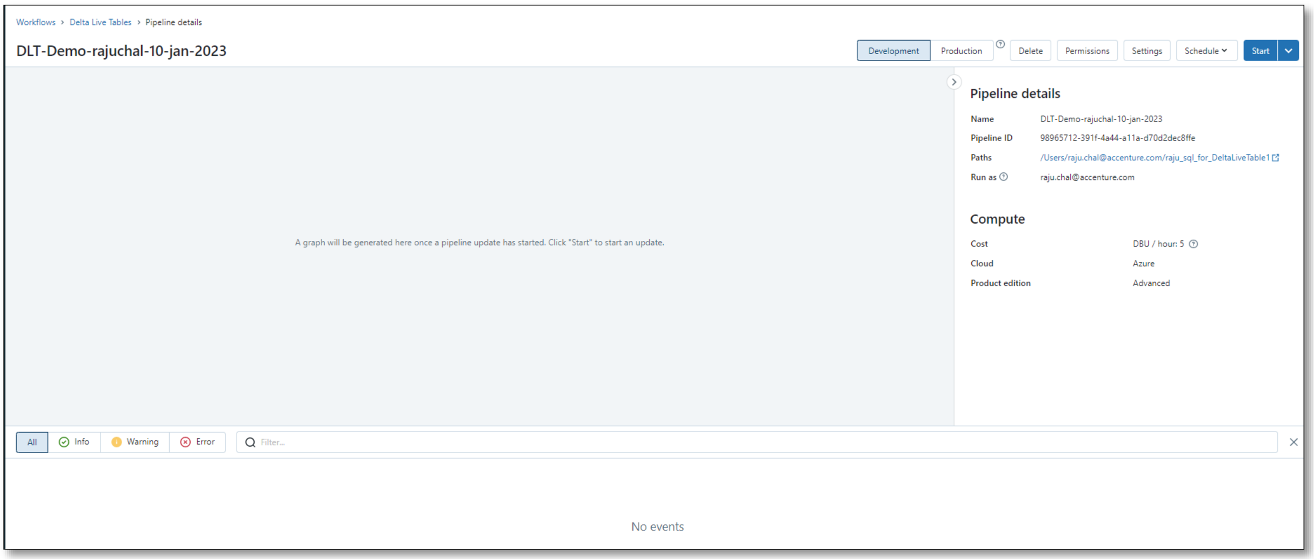
☑ Use Photon Acceleration

Advanced

Configuration

spark.master	local[*]
datasets_path	dbfs:/FileStore/tables/nyctaxidata/dltl

Channel: Current



DLT-Demo-rajuchal-10-jan-2023

A graph will be generated here once a pipeline update has started. Click "Start" to start an update.

Pipeline details

Name	DLT-Demo-rajuchal-10-jan-2023
Pipeline ID	98965712-391f-4a44-a11a-d70d2dec8ffe
Paths	/Users/raju.chal@accenture.com/raju_sql_for_DeltaLiveTable1
Run as	raju.chal@accenture.com

Compute

Cost	DBU / hour: 5
Cloud	Azure
Product edition	Advanced

No events

Open and Complete DLT Pipeline Notebook

Open the Notebook and, following the guidelines provided therein, fill in the cells where prompted to implement a multi-hop architecture similar to the one we worked with in the previous section.

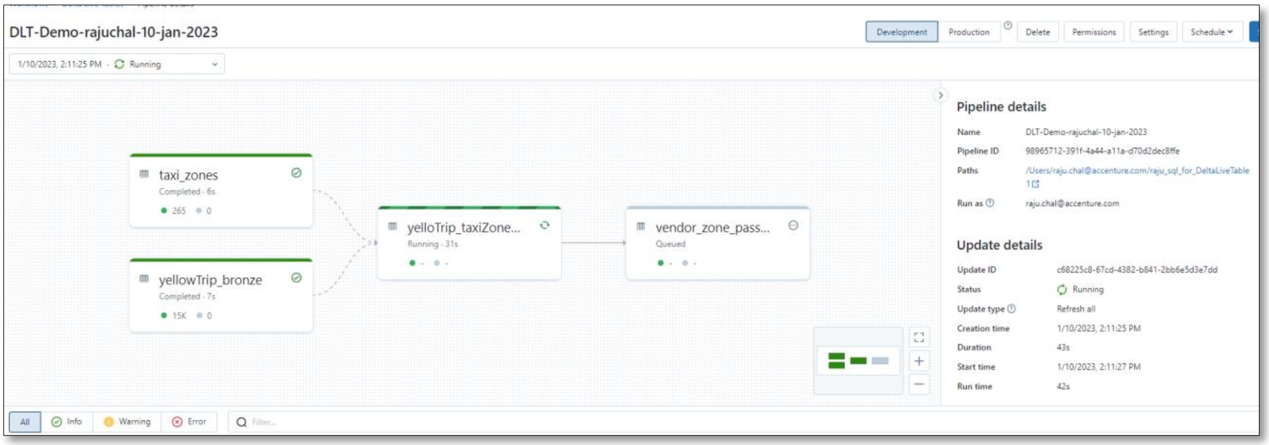
Run your Pipeline

Select **Development** mode, which accelerates the development lifecycle by reusing the same cluster across runs.
It will also turn off automatic retries when jobs fail.

Click **Start** to begin the first update to your table.

Delta Live Tables will automatically deploy all the necessary infrastructure and resolve the dependencies between all datasets.

NOTE: The first table update may take several minutes as relationships are resolved and infrastructure deploys.

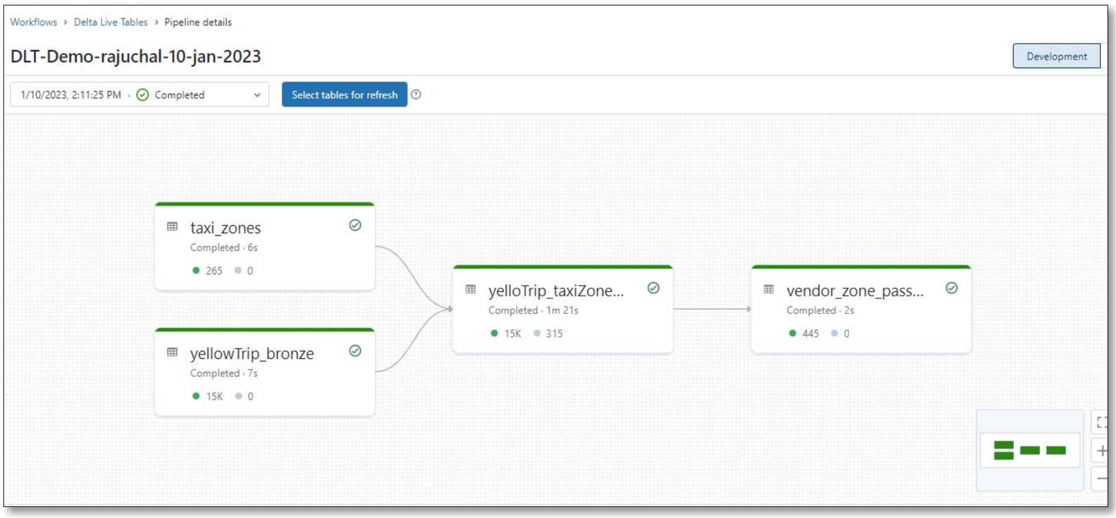


Pipeline details

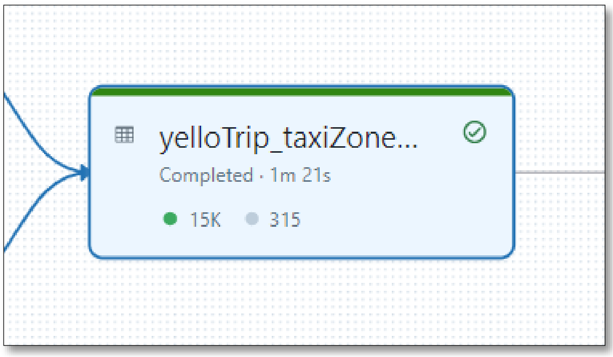
Name	DLT-Demo-rajuchal-10-jan-2023
Pipeline ID	98965712-391f-4a44-a11a-d70d2dec8ffe
Paths	/Users/raju.chal@accenture.com/raju_sql_for_DeltaLiveTable 1
Run as	raju.chal@accenture.com

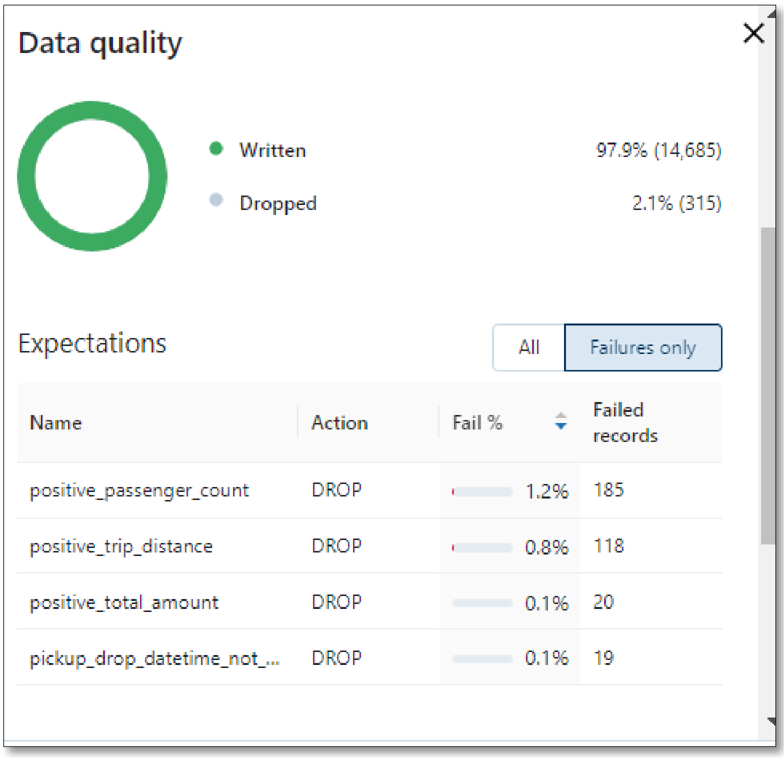
Update details

Update ID	c68225c8-67cd-4382-b841-2bb6e5d3e7dd
Status	Running
Update type	Refresh all
Creation time	1/10/2023, 2:11:25 PM
Duration	43s
Start time	1/10/2023, 2:11:27 PM
Run time	42s



Select yelloTrip_taxiZones_Silver table in data flow and check the Data Quality section





Data quality

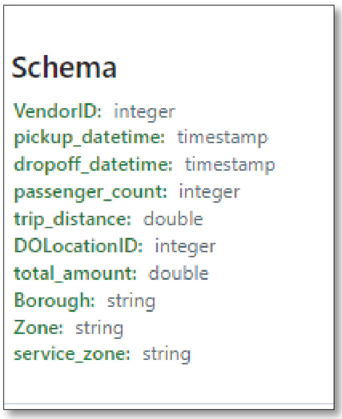
● Written 97.9% (14,685)

● Dropped 2.1% (315)

Expectations

All | Failures only

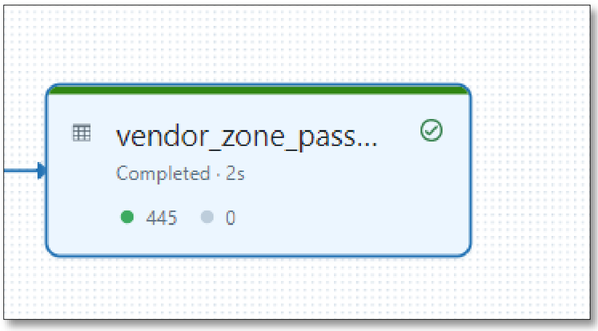
Name	Action	Fail %	Failed records
positive_passenger_count	DROP	1.2%	185
positive_trip_distance	DROP	0.8%	118
positive_total_amount	DROP	0.1%	20
pickup_drop_datetime_not_...	DROP	0.1%	19

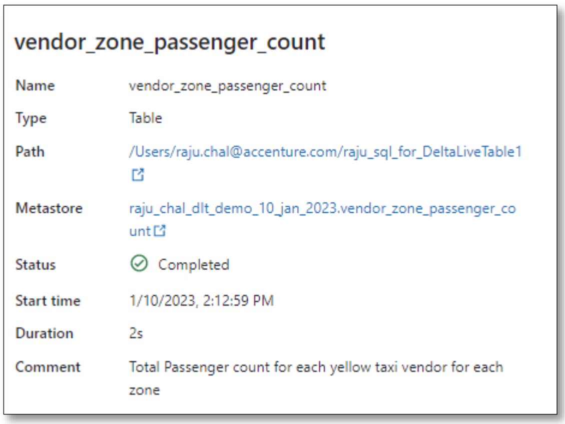


Schema

VendorID: integer
pickup_datetime: timestamp
dropoff_datetime: timestamp
passenger_count: integer
trip_distance: double
DOLocationID: integer
total_amount: double
Borough: string
Zone: string
service_zone: string

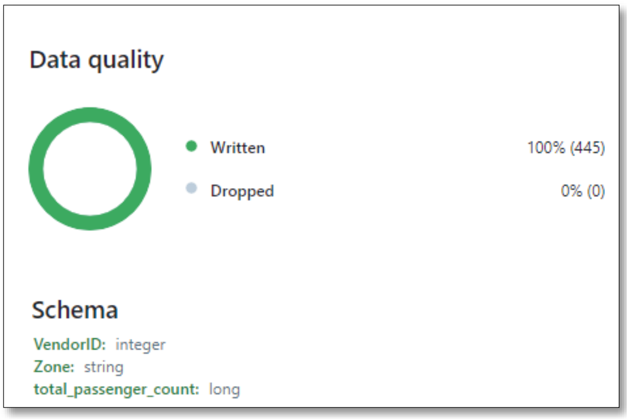
Select vendor_zone_passenger_count table in data flow and check the Data Quality section





vendor_zone_passenger_count

Name	vendor_zone_passenger_count
Type	Table
Path	/Users/raju.chal@accenture.com/raju_sql_for_DeltaLiveTable1
Metastore	raju_chal_dlt_demo_10_jan_2023.vendor_zone_passenger_count
Status	✓ Completed
Start time	1/10/2023, 2:12:59 PM
Duration	2s
Comment	Total Passenger count for each yellow taxi vendor for each zone



Data quality

● Written	100% (445)
● Dropped	0% (0)

Schema

VendorID: integer
Zone: string
total_passenger_count: long

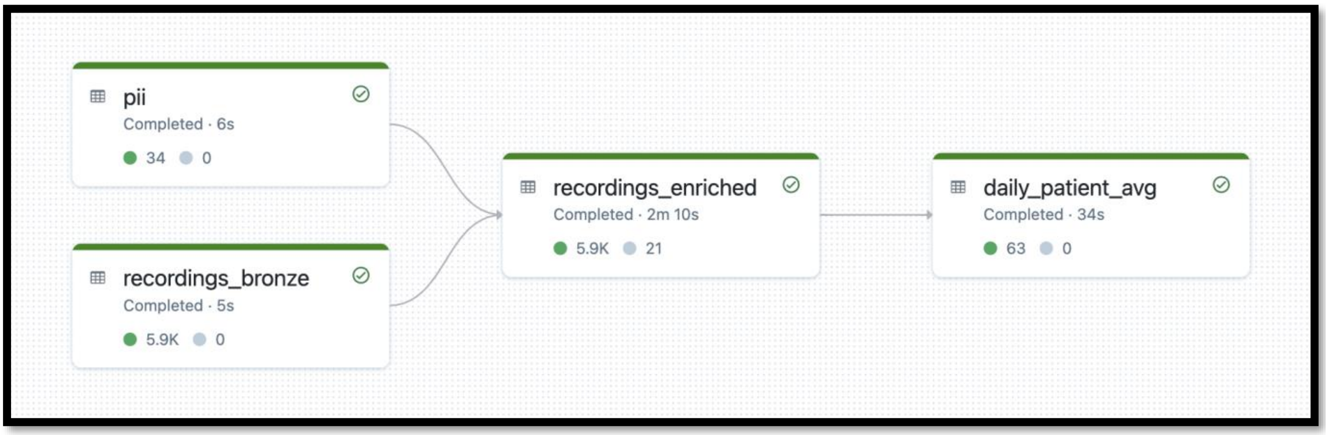
Open the DeltaLiveTableDemo1 note book that we have already created

Troubleshooting Code in Development Mode

Don't despair if your pipeline fails the first time. Delta Live Tables is in active development, and error messages are improving all the time.

Because relationships between tables are mapped as a DAG, error messages will often indicate that a dataset isn't found.

Let's consider our DAG below:



If the error message `Dataset not found: 'recordings_parsed'` is raised, there may be several culprits:

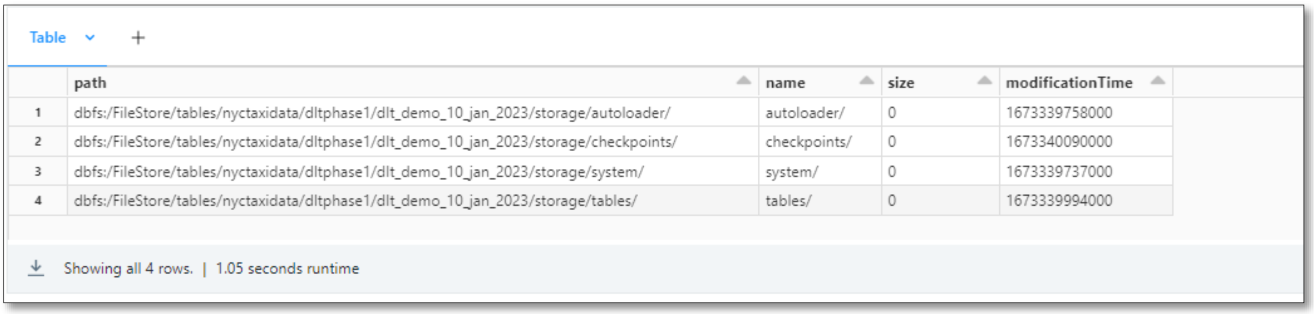
1. The logic defining `recordings_parsed` is invalid
2. There is an error reading from `recordings_bronze`
3. A typo exists in either `recordings_parsed` or `recordings_bronze`

The safest way to identify the culprit is to iteratively add table/view definitions back into your DAG starting from your initial ingestion tables. You can simply comment out later table/view definitions and uncomment these between runs.

Lab: Conclusion

Display Results

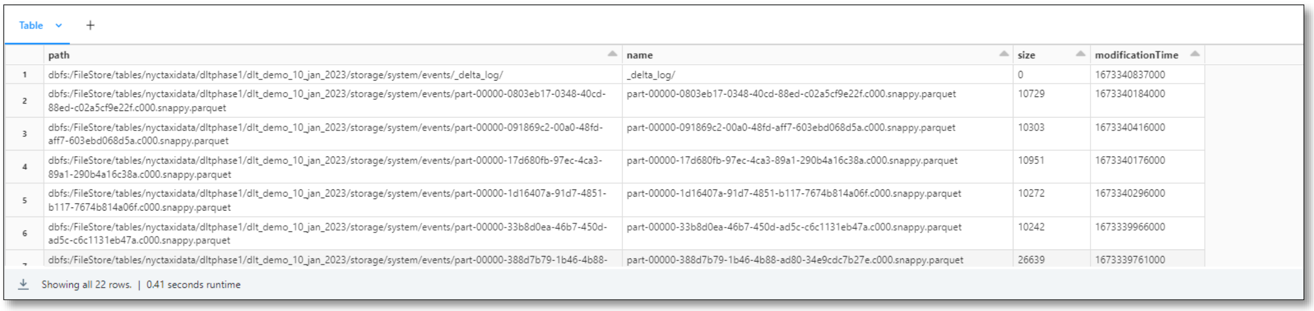
```
files = dbutils.fs.ls("dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage")

display(files)
```



	path	name	size	modificationTime
1	dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/autoloader/	autoloader/	0	1673339758000
2	dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/checkpoints/	checkpoints/	0	1673340090000
3	dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/system/	system/	0	1673339737000
4	dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/tables/	tables/	0	1673339994000

Showing all 4 rows. | 1.05 seconds runtime

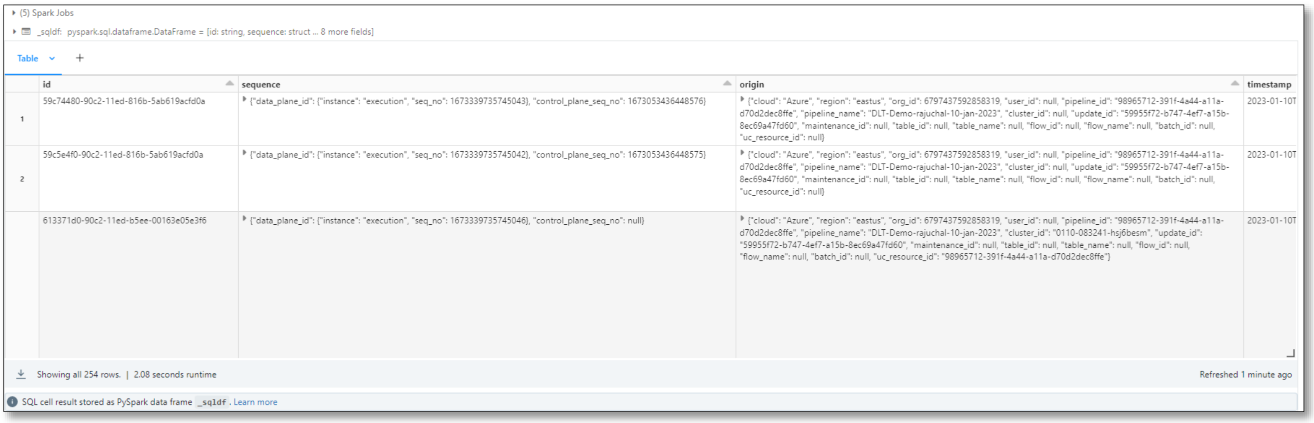
```
files =
dbutils.fs.ls("dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/system
/events")

display(files)
```



	path	name	size	modificationTime
1	dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/system/events/_delta_log/	_delta_log/	0	1673340837000
2	dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/system/events/part-00000-0803eb17-0348-40cd-88ed-c02a5cf9e22f.c000.snappy.parquet	part-00000-0803eb17-0348-40cd-88ed-c02a5cf9e22f.c000.snappy.parquet	10729	1673340184000
3	dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/system/events/part-00000-091869c2-00a0-48fd-aff7-603ebd068d5a.c000.snappy.parquet	part-00000-091869c2-00a0-48fd-aff7-603ebd068d5a.c000.snappy.parquet	10303	1673340416000
4	dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/system/events/part-00000-17d680fb-97ec-4ca3-89a1-290b4a16c38a.c000.snappy.parquet	part-00000-17d680fb-97ec-4ca3-89a1-290b4a16c38a.c000.snappy.parquet	10951	1673340176000
5	dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/system/events/part-00000-1d16407a-91d7-4851-b117-7674b814a06f.c000.snappy.parquet	part-00000-1d16407a-91d7-4851-b117-7674b814a06f.c000.snappy.parquet	10272	1673340296000
6	dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/system/events/part-00000-33b8d0ea-46b7-450d-ad5c-c6c1131eb47a.c000.snappy.parquet	part-00000-33b8d0ea-46b7-450d-ad5c-c6c1131eb47a.c000.snappy.parquet	10242	1673339966000
7	dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/system/events/part-00000-388d7b79-1b46-4b88-	part-00000-388d7b79-1b46-4b88-ad80-34e9cdc7b27e.c000.snappy.parquet	26639	1673339761000

Showing all 22 rows. | 0.41 seconds runtime

```
%sql

SELECT * FROM
delta.`dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/system/events`
```



(5) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [id: string, sequence: struct ... 8 more fields]

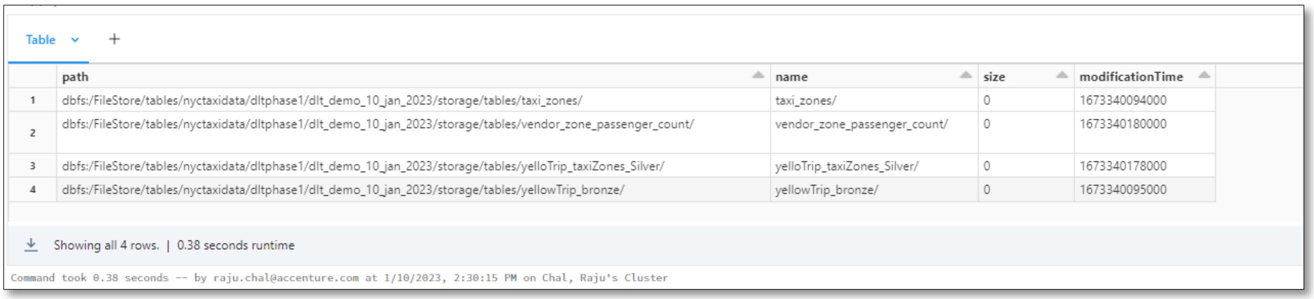
	id	sequence	origin	timestamp
1	59c74480-90c2-11ed-816b-5ab619acfd0a	{"data_plane_id": {"instance": "execution", "seq_no": 1673339735745043}, "control_plane_seq_no": 1673053436448576}	{"cloud": "Azure", "region": "eastus", "org_id": 6797437592858319, "user_id": null, "pipeline_id": "98965712-391f-4a44-a11a-d70d2dec8ffe", "pipeline_name": "DLT-Demo-rajuchal-10-jan-2023", "cluster_id": null, "update_id": "59955f72-b747-4ef7-a15b-8ec69a47fd60", "maintenance_id": null, "table_id": null, "table_name": null, "flow_id": null, "flow_name": null, "batch_id": null, "uc_resource_id": null}	2023-01-10T
2	59c5e4f0-90c2-11ed-816b-5ab619acfd0a	{"data_plane_id": {"instance": "execution", "seq_no": 1673339735745042}, "control_plane_seq_no": 1673053436448575}	{"cloud": "Azure", "region": "eastus", "org_id": 6797437592858319, "user_id": null, "pipeline_id": "98965712-391f-4a44-a11a-d70d2dec8ffe", "pipeline_name": "DLT-Demo-rajuchal-10-jan-2023", "cluster_id": null, "update_id": "59955f72-b747-4ef7-a15b-8ec69a47fd60", "maintenance_id": null, "table_id": null, "table_name": null, "flow_id": null, "flow_name": null, "batch_id": null, "uc_resource_id": null}	2023-01-10T
	613371d0-90c2-11ed-b5ee-00163e05e3f6	{"data_plane_id": {"instance": "execution", "seq_no": 1673339735745046}, "control_plane_seq_no": null}	{"cloud": "Azure", "region": "eastus", "org_id": 6797437592858319, "user_id": null, "pipeline_id": "98965712-391f-4a44-a11a-d70d2dec8ffe", "pipeline_name": "DLT-Demo-rajuchal-10-jan-2023", "cluster_id": "0110-083241-hsj6besm", "update_id": "59955f72-b747-4ef7-a15b-8ec69a47fd60", "maintenance_id": null, "table_id": null, "table_name": null, "flow_id": null, "flow_name": null, "batch_id": null, "uc_resource_id": "98965712-391f-4a44-a11a-d70d2dec8ffe"}	2023-01-10T

Showing all 254 rows. | 2.08 seconds runtime

Refreshed 1 minute ago

SQL cell result stored as PySpark data frame `_sqldf`. Learn more

```
files = dbutils.fs.ls("dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/tables/")

display(files)
```



	path	name	size	modificationTime
1	dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/tables/taxi_zones/	taxi_zones/	0	1673340094000
2	dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/tables/vendor_zone_passenger_count/	vendor_zone_passenger_count/	0	1673340180000
3	dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/tables/yelloTrip_taxiZones_Silver/	yelloTrip_taxiZones_Silver/	0	1673340178000
4	dbfs:/FileStore/tables/nyctaxidata/dltphase1/dlt_demo_10_jan_2023/storage/tables/yellowTrip_bronze/	yellowTrip_bronze/	0	1673340095000

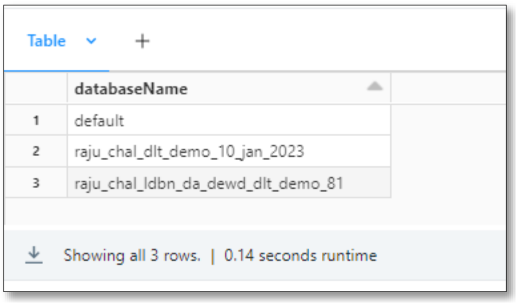
Showing all 4 rows. | 0.38 seconds runtime

```
Command took 0.38 seconds -- by raju.chal@accenture.com at 1/10/2023, 2:30:15 PM on Chal, Raju's Cluster
```

Assuming your pipeline runs successfully, display the contents of the gold table.

NOTE: Because we specified a value for **Target**, tables are published to the specified database. Without a **Target** specification, we would need to query the table based on its underlying location in DBFS (relative to the **Storage Location**).

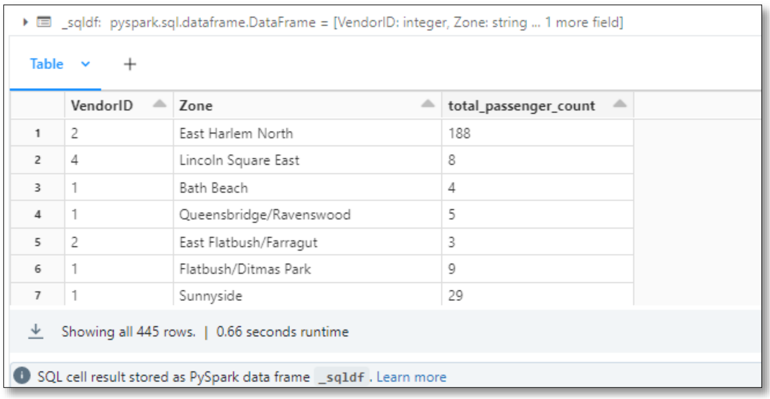
```
%sql

show databases
```



	databaseName
1	default
2	raju_chal_dlt_demo_10_jan_2023
3	raju_chal_ldbn_da_dewd_dlt_demo_81

Showing all 3 rows. | 0.14 seconds runtime

```
%sql

SELECT * FROM
raju_chal_dlt_demo_10_jan_2023.vendor_zone_passenger_count
```



_sqldf: pyspark.sql.dataframe.DataFrame = [VendorID: integer, Zone: string ... 1 more field]

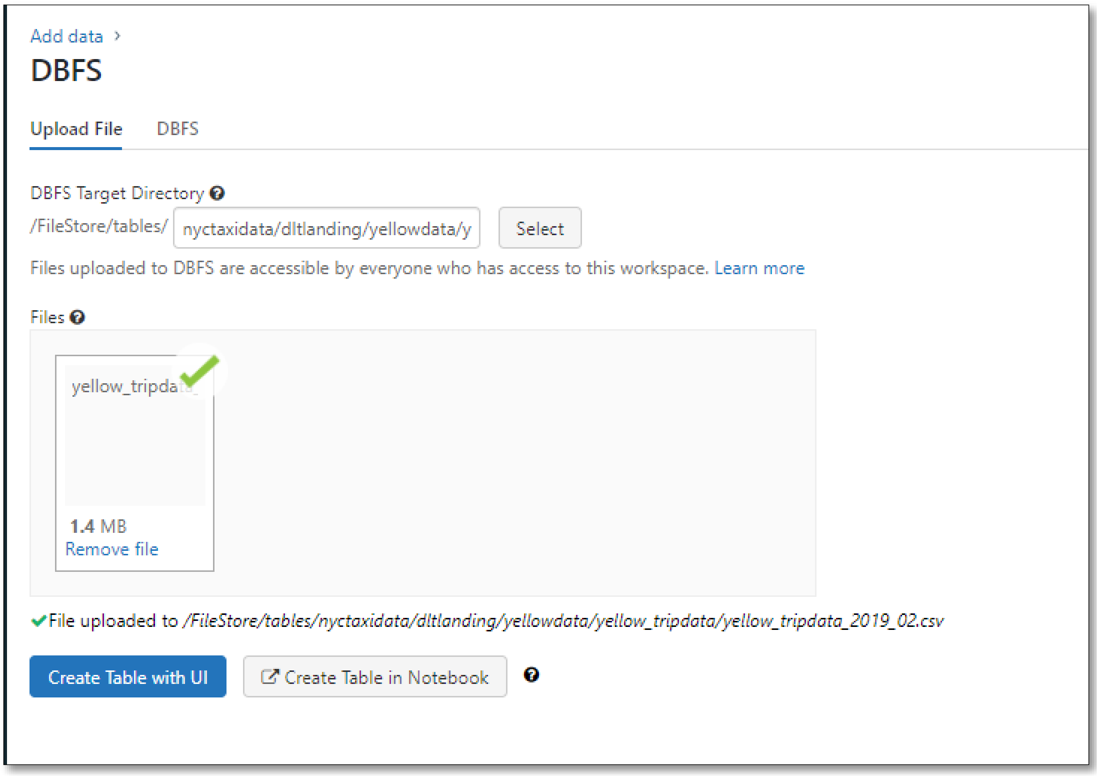
	VendorID	Zone	total_passenger_count
1	2	East Harlem North	188
2	4	Lincoln Square East	8
3	1	Bath Beach	4
4	1	Queensbridge/Ravenswood	5
5	2	East Flatbush/Farragut	3
6	1	Flatbush/Ditmas Park	9
7	1	Sunnyside	29

Showing all 445 rows. | 0.66 seconds runtime

SQL cell result stored as PySpark data frame _sqldf . Learn more

Upload another yellow trip data file into DBFS location

dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata


Add data ›

DBFS

DBFS Target Directory
/FileStore/tables/ nyctaxidata/dltlanding/yellowdata/y

Files uploaded to DBFS are accessible by everyone who has access to this workspace. Learn more

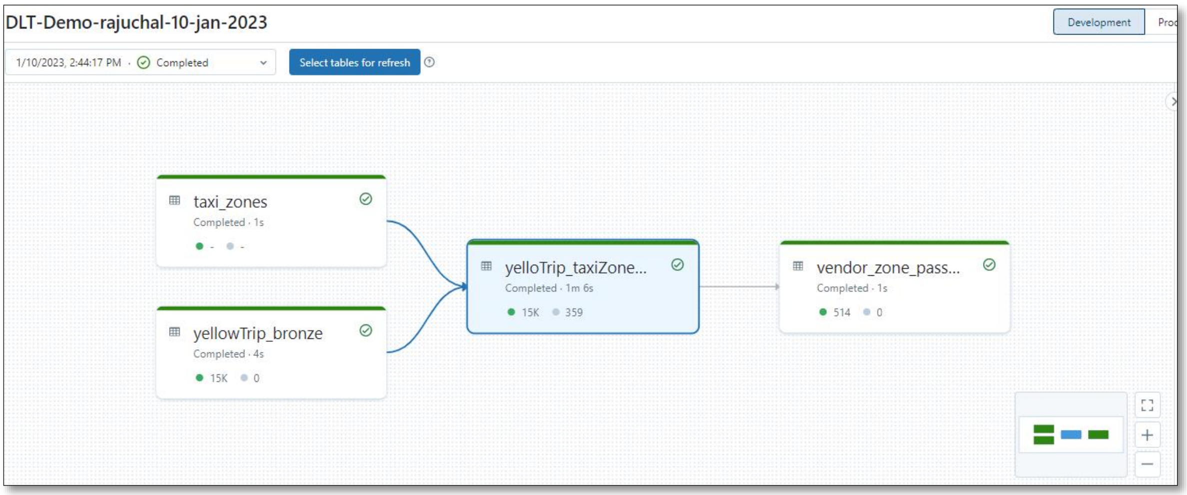
Files

yellow_tripda...
1.4 MB

✓File uploaded to */FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata/yellow_tripdata_2019_02.csv*

Feel free to run it a couple more times if desired.

Following this, run the pipeline again and view the results.


DLT-Demo-rajuchal-10-jan-2023

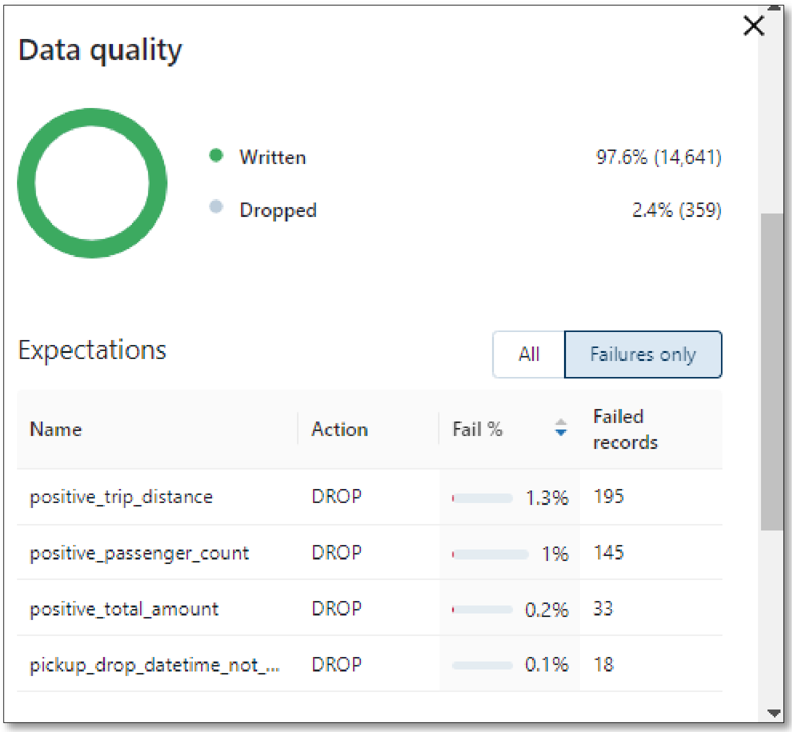
1/10/2023, 2:44:17 PM · Completed

taxi_zones
Completed · 1s
● - ● -

yellowTrip_bronze
Completed · 4s
● 15K ● 0

yelloTrip_taxiZone...
Completed · 1m 6s
● 15K ● 359

vendor_zone_pass...
Completed · 1s
● 514 ● 0



Data quality

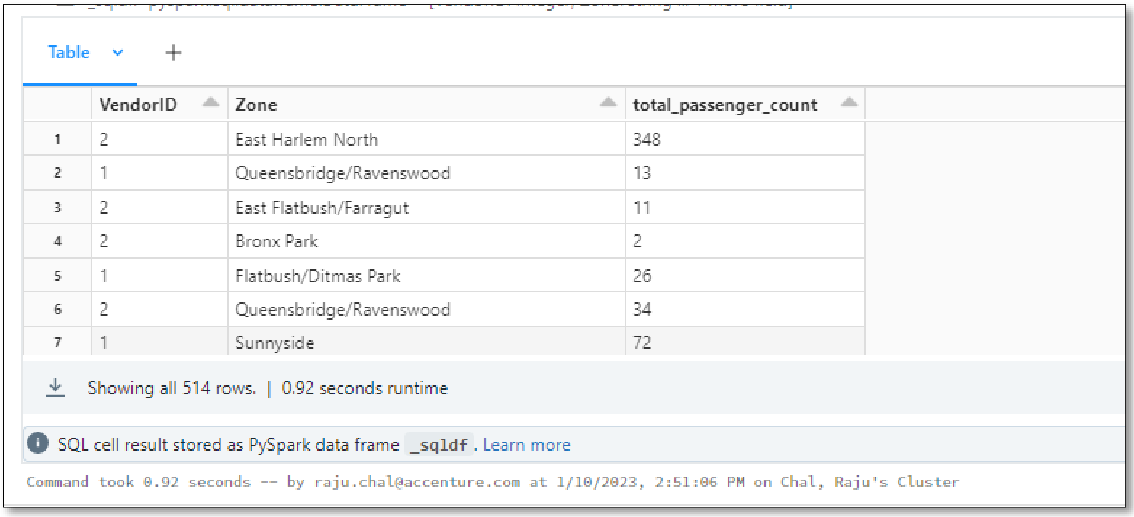
● Written	97.6% (14,641)
● Dropped	2.4% (359)

Expectations

Name	Action	Fail %	Failed records
positive_trip_distance	DROP	1.3%	195
positive_passenger_count	DROP	1%	145
positive_total_amount	DROP	0.2%	33
pickup_drop_datetime_not_...	DROP	0.1%	18

Feel free to re-run the cell above to gain an updated view of the **vendor_zone_passenger_count** table.

```
%sql

SELECT * FROM raju_chal_dlt_demo_10_jan_2023.vendor_zone_passenger_count
```



	VendorID	Zone	total_passenger_count
1	2	East Harlem North	348
2	1	Queensbridge/Ravenswood	13
3	2	East Flatbush/Farragut	11
4	2	Bronx Park	2
5	1	Flatbush/Ditmas Park	26
6	2	Queensbridge/Ravenswood	34
7	1	Sunnyside	72

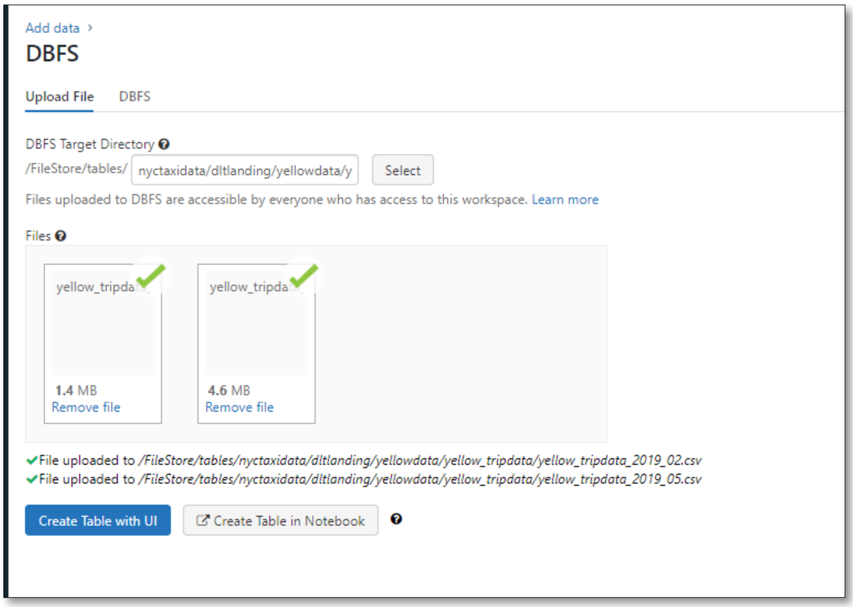
Showing all 514 rows. | 0.92 seconds runtime

SQL cell result stored as PySpark data frame `_sqldf`. Learn more

```
Command took 0.92 seconds -- by raju.chal@accenture.com at 1/10/2023, 2:51:06 PM on Chal, Raju's Cluster
```

Upload another yellow trip data file into DBFS location

dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata



Add data ›

DBFS

Upload File DBFS

DBFS Target Directory ❓

/FileStore/tables/ nyctaxidata/dltlanding/yellowdata/y Select

Files uploaded to DBFS are accessible by everyone who has access to this workspace. Learn more

Files ❓

yellow_tripda ✓ 1.4 MB Remove file

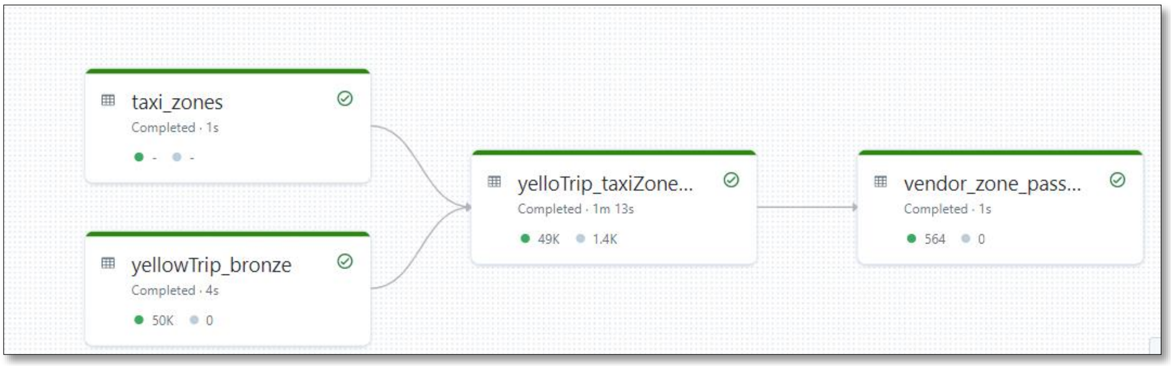
yellow_tripda ✓ 4.6 MB Remove file

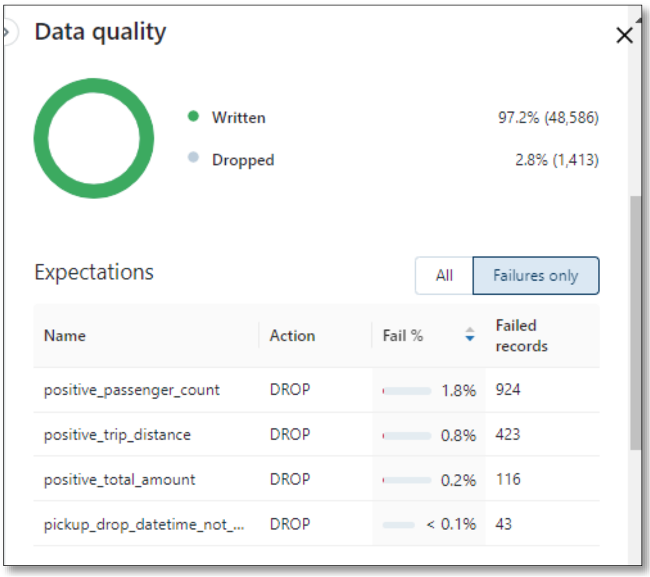
✓File uploaded to */FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata/yellow_tripdata_2019_02.csv*

✓File uploaded to */FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata/yellow_tripdata_2019_05.csv*

Create Table with UI | Create Table in Notebook ❓

Following this, run the pipeline again and view the results.





Data quality

● Written 97.2% (48,586)

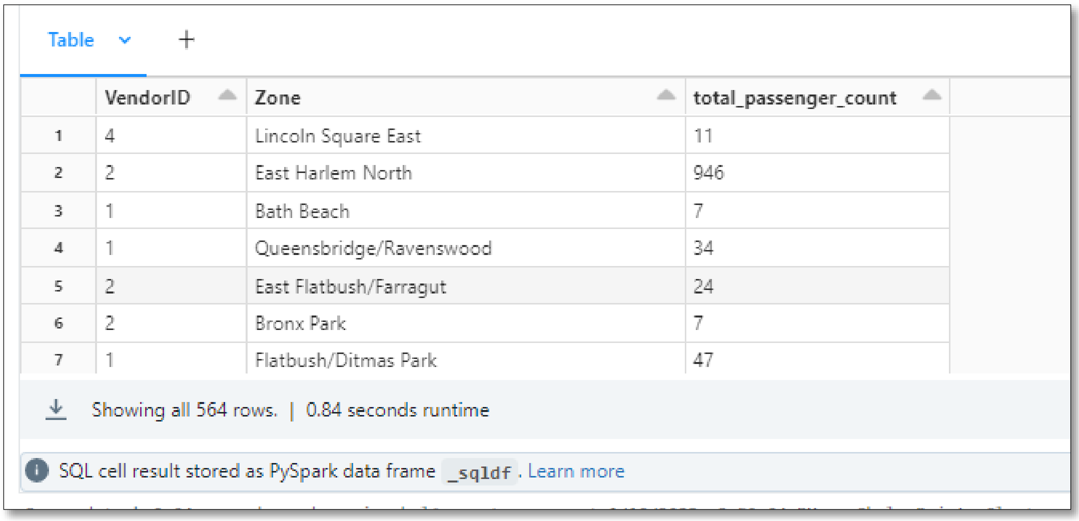
● Dropped 2.8% (1,413)

Expectations All | Failures only

Name	Action	Fail %	Failed records
positive_passenger_count	DROP	1.8%	924
positive_trip_distance	DROP	0.8%	423
positive_total_amount	DROP	0.2%	116
pickup_drop_datetime_not_...	DROP	< 0.1%	43

Check the updated view of the **vendor_zone_passenger_count** table.

```
%sql

SELECT * FROM raju_chal_dlt_demo_10_jan_2023.vendor_zone_passenger_count
```



	VendorID	Zone	total_passenger_count
1	4	Lincoln Square East	11
2	2	East Harlem North	946
3	1	Bath Beach	7
4	1	Queensbridge/Ravenswood	34
5	2	East Flatbush/Farragut	24
6	2	Bronx Park	7
7	1	Flatbush/Ditmas Park	47

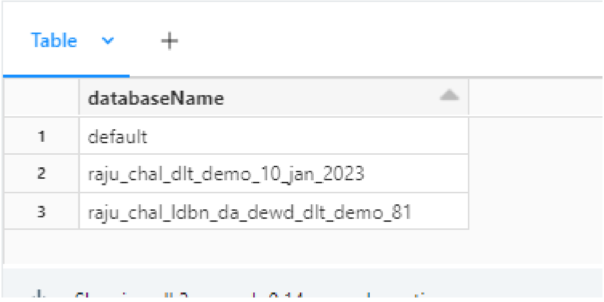
Showing all 564 rows. | 0.84 seconds runtime

SQL cell result stored as PySpark data frame `_sqldf`. Learn more

Summary

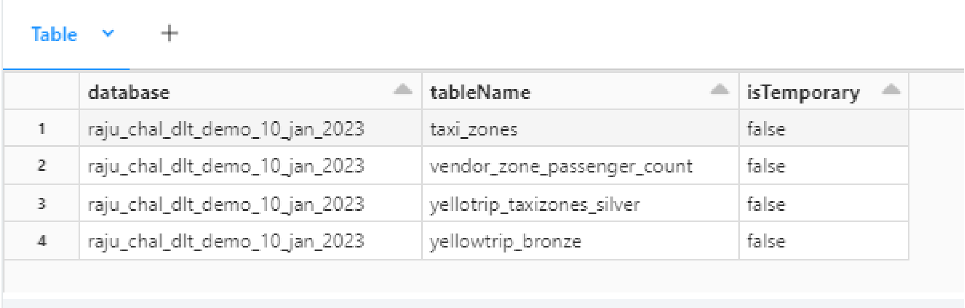
In this lab, you learned to convert an existing data pipeline to a Delta Live Tables SQL pipeline, and deployed that pipeline using the DLT UI.

```
%sql

show databases
```



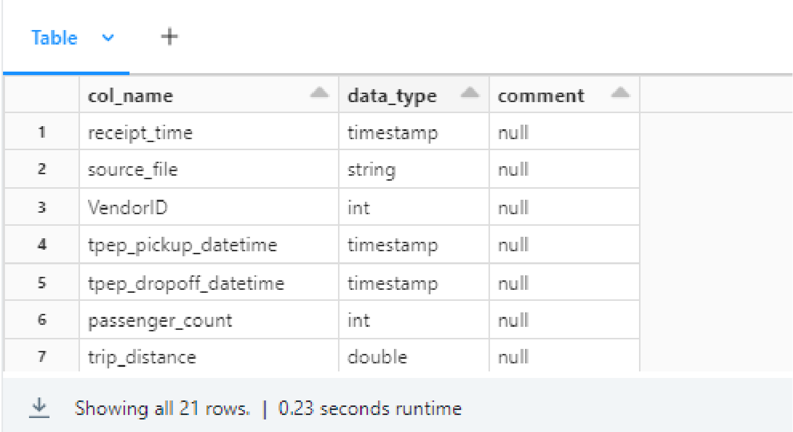
	databaseName
1	default
2	raju_chal_dlt_demo_10_jan_2023
3	raju_chal_ldbn_da_dewd_dlt_demo_81

```
%sql

use raju_chal_dlt_demo_10_jan_2023
```



	database	tableName	isTemporary
1	raju_chal_dlt_demo_10_jan_2023	taxi_zones	false
2	raju_chal_dlt_demo_10_jan_2023	vendor_zone_passenger_count	false
3	raju_chal_dlt_demo_10_jan_2023	yellotrip_taxizones_silver	false
4	raju_chal_dlt_demo_10_jan_2023	yellowtrip_bronze	false

```
%sql

DESCRIBE raju_chal_dlt_demo_10_jan_2023.yellowtrip_bronze
```

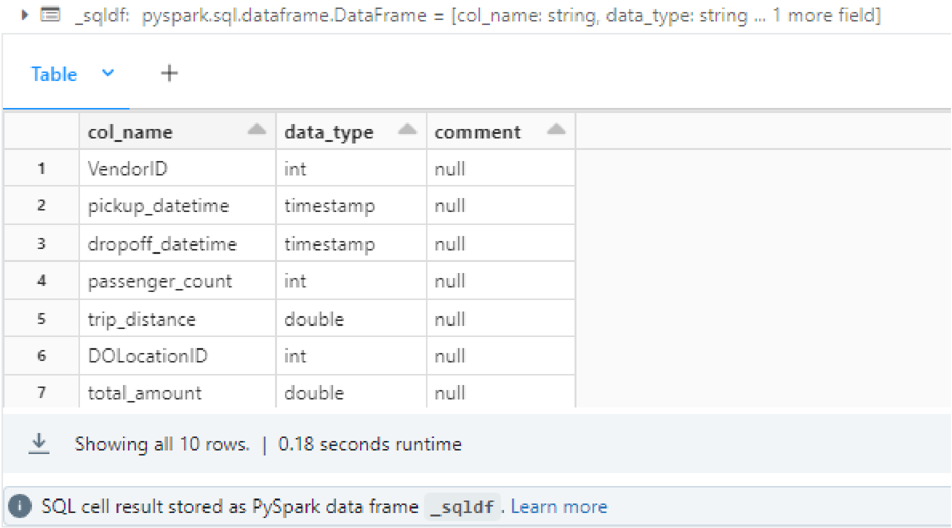


	col_name	data_type	comment
1	receipt_time	timestamp	null
2	source_file	string	null
3	VendorID	int	null
4	tpep_pickup_datetime	timestamp	null
5	tpep_dropoff_datetime	timestamp	null
6	passenger_count	int	null
7	trip_distance	double	null

Showing all 21 rows. | 0.23 seconds runtime

```
%sql
```

DESCRIBE raju_chal_dlt_demo_10_jan_2023.yellotrip_taxizones_silver



_sqldf: pyspark.sql.dataframe.DataFrame = [col_name: string, data_type: string ... 1 more field]

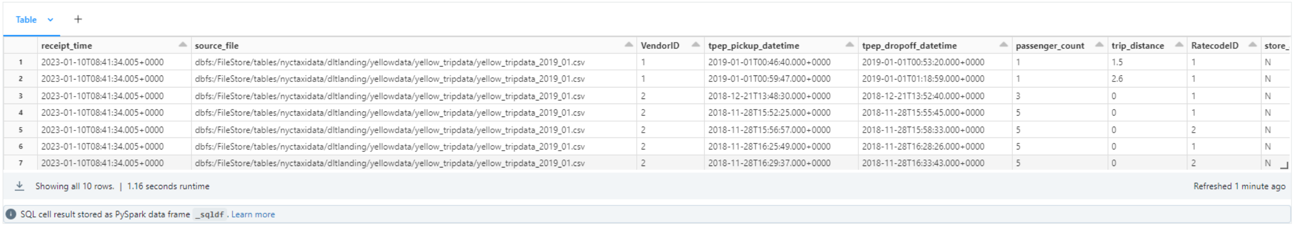
	col_name	data_type	comment
1	VendorID	int	null
2	pickup_datetime	timestamp	null
3	dropoff_datetime	timestamp	null
4	passenger_count	int	null
5	trip_distance	double	null
6	DOLocationID	int	null
7	total_amount	double	null

Showing all 10 rows. | 0.18 seconds runtime

SQL cell result stored as PySpark data frame `_sqldf`. Learn more

%sql

select * from raju_chal_dlt_demo_10_jan_2023.yellowtrip_bronze limit 10



	receipt_time	source_file	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_
1	2023-01-10T08:41:34.005+0000	dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv	1	2019-01-01T00:46:40.000+0000	2019-01-01T00:53:20.000+0000	1	1.5	1	N
2	2023-01-10T08:41:34.005+0000	dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv	1	2019-01-01T00:59:47.000+0000	2019-01-01T01:18:59.000+0000	1	2.6	1	N
3	2023-01-10T08:41:34.005+0000	dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv	2	2018-12-21T13:48:30.000+0000	2018-12-21T13:52:40.000+0000	3	0	1	N
4	2023-01-10T08:41:34.005+0000	dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv	2	2018-11-28T15:52:25.000+0000	2018-11-28T15:55:45.000+0000	5	0	1	N
5	2023-01-10T08:41:34.005+0000	dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv	2	2018-11-28T15:56:57.000+0000	2018-11-28T15:58:33.000+0000	5	0	2	N
6	2023-01-10T08:41:34.005+0000	dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv	2	2018-11-28T16:25:49.000+0000	2018-11-28T16:28:26.000+0000	5	0	1	N
7	2023-01-10T08:41:34.005+0000	dbfs:/FileStore/tables/nyctaxidata/dltlanding/yellowdata/yellow_tripdata/yellow_tripdata_2019_01.csv	2	2018-11-28T16:29:37.000+0000	2018-11-28T16:33:43.000+0000	5	0	2	N

Showing all 10 rows. | 1.16 seconds runtime

Refreshed 1 minute ago

SQL cell result stored as PySpark data frame `_sqldf`. Learn more