Ingesting Data From RDS To HBase Table

Step 1: First we need to login to EMR ssh. We need to login as root user. For root login we use following command:

sudo -i

```
[hadoop@ip-172-31-31-253 ~]$ sudo -i

EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M::::::M         M::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M::::M   M::::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M::::M M::::M M::::M M::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M::::M  M::::M::::M  M::::M   R::::::::::::RR
  E:::::EEEEEEEEEE   M::::M   M::::::M   M::::M   R:::RRRRRR::::R
  E::::E             M::::M    M::::M    M::::M   R:::R      R::::R
  E::::E       EEEEE M::::M     MMM     M::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M::::M             M::::M   R:::R      R::::R
E::::::::::::::::::E M::::M             M::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMM             MMMMMM RRRRRRR      RRRRRR
```

Step 2: Now we create an HBase table where the data from RDS will be ingested. Here we have created a table named yellow_taxi_data_hbase:

1: hbase shell

2: create 'yellow_taxi_data_hbase', 'info'

3: exit

This creates an HBase table 'yellow_taxi_data_hbase' with a column family info

Step 3: Next, use the Sqoop import command to ingest data from your MySQL RDS instance to HBase.

sqoop import \

  --connect jdbc:mysql://database-1.c7uy42424v94.us-east-1.rds.amazonaws.com:3306/assigment \

  --username admin \

  --password '#RSPrp97' \

  --table yellow_taxi_data \

  --hbase-table yellow_taxi_data_hbase \

```
  --column-family info \
  --hbase-create-table \
  --split-by tpep_pickup_datetime \
  --fields-terminated-by ',' \
  --lines-terminated-by '\n' \
  -m 4
```

Explation:

--connect: MySQL JDBC connection string to your RDS instance.

--username: MySQL username for RDS.

--password: MySQL password for RDS.

--table: The table in RDS from which data is imported.

--hbase-table: The name of the HBase table where data will be imported.

--column-family: The column family in HBase.

--hbase-create-table: This option ensures the HBase table is created if it doesn't exist.

--split-by: A column to split the data.

--fields-terminated-by: Specifies how fields are separated in the input file.

--lines-terminated-by: Specifies the line termination character.

Step 4: Now to check whether the data is loaded to hbase or not we will use given below commands:

1: list

```
hbase(main):001:0> list
TABLE
yellow_taxi_data
yellow_taxi_data_hbase
2 row(s) in 1.1780 seconds
```

2: scan 'yellow_taxi_data_hbase', { LIMIT => 10 }

```
=> ["yellow_taxi_data", "yellow_taxi_data_hbase"]
hbase(main):002:0> scan 'yellow_taxi_data_hbase', { LIMIT => 10 }
ROW                                                    COLUMN+CELL
 2017-01-01 00:00:00.0                                 column=info:Airport_fee, timestamp=1736098684316, value=0.0
 2017-01-01 00:00:00.0                                 column=info:DOLocationID, timestamp=1736098684316, value=234
 2017-01-01 00:00:00.0                                 column=info:PULocationID, timestamp=1736098684316, value=249
 2017-01-01 00:00:00.0                                 column=info:RatecodeID, timestamp=1736098684316, value=2
 2017-01-01 00:00:00.0                                 column=info:VendorID, timestamp=1736098684316, value=2
 2017-01-01 00:00:00.0                                 column=info:extra, timestamp=1736098684316, value=0.0
 2017-01-01 00:00:00.0                                 column=info:fare_amount, timestamp=1736098684316, value=52.0
 2017-01-01 00:00:00.0                                 column=info:improvement_surcharge, timestamp=1736098684316, value=0.3
 2017-01-01 00:00:00.0                                 column=info:mta_tax, timestamp=1736098684316, value=0.5
 2017-01-01 00:00:00.0                                 column=info:passenger_count, timestamp=1736098684316, value=1
 2017-01-01 00:00:00.0                                 column=info:payment_type, timestamp=1736098684316, value=2
 2017-01-01 00:00:00.0                                 column=info:store_and_fwd_flag, timestamp=1736098684316, value=N
 2017-01-01 00:00:00.0                                 column=info:tip_amount, timestamp=1736098684316, value=0.0
 2017-01-01 00:00:00.0                                 column=info:tolls_amount, timestamp=1736098684316, value=0.0
 2017-01-01 00:00:00.0                                 column=info:total_amount, timestamp=1736098684316, value=52.8
 2017-01-01 00:00:00.0                                 column=info:tpep_dropoff_datetime, timestamp=1736098684316, value=2017-01-01 00:00:00.0
 2017-01-01 00:00:00.0                                 column=info:trip_distance, timestamp=1736098684316, value=0.02
 2017-01-01 00:00:02.0                                 column=info:Airport_fee, timestamp=1736098684936, value=0.0
 2017-01-01 00:00:02.0                                 column=info:DOLocationID, timestamp=1736098684936, value=48
 2017-01-01 00:00:02.0                                 column=info:PULocationID, timestamp=1736098684936, value=48
 2017-01-01 00:00:02.0                                 column=info:RatecodeID, timestamp=1736098684936, value=1
 2017-01-01 00:00:02.0                                 column=info:VendorID, timestamp=1736098684936, value=1
 2017-01-01 00:00:02.0                                 column=info:extra, timestamp=1736098684936, value=0.5
 2017-01-01 00:00:02.0                                 column=info:fare_amount, timestamp=1736098684936, value=4.0
 2017-01-01 00:00:02.0                                 column=info:improvement_surcharge, timestamp=1736098684936, value=0.3
 2017-01-01 00:00:02.0                                 column=info:mta_tax, timestamp=1736098684936, value=0.5
 2017-01-01 00:00:02.0                                 column=info:passenger_count, timestamp=1736098684936, value=1
 2017-01-01 00:00:02.0                                 column=info:payment_type, timestamp=1736098684936, value=2
```

3: echo "count 'yellow_taxi_data_hbase'" | hbase shell > count_output.txt

tail -n 1 count_output.txt

```
[root@ip-172-31-31-253 ~]# echo "count 'yellow_taxi_data_hbase'" | hbase shell > count_output.t
login as: hadoop
Authenticating with public key "imported-openssh-key"
Last login: Sun Jan  5 18:06:56 2025

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
89 package(s) needed for security, out of 154 available
Run "sudo yum update" to apply all updates.
```

```
[root@ip-172-31-31-253 ~]# tail -n 1 count_output.txt
18880595
[root@ip-172-31-31-253 ~]#
```

We got same number of rows.

Inserting Data of yellow_tripdata_2017-03.csv and yellow_tripdata_2017-04.csv to HBase

Step1: First we need to download both the files using wget command.

1: wget https://nyc-tlc-upgrad.s3.amazonaws.com/yellow_tripdata_2017-03.csv

2: wget https://nyc-tlc-upgrad.s3.amazonaws.com/yellow_tripdata_2017-04.csv

Step2: We have created HBase table using python. Following are the commands to follow

1: vi create_python.py

```
import happybase


# Create connection

connection = happybase.Connection('localhost', port=9090, autoconnect=False)

# Open connection to perform operations

def open_connection():

    connection.open()

# Close the opened connection

def close_connection():

    connection.close()

# List all tables in HBase

def list_tables():

    print("Fetching all tables...")

    open_connection()

    tables = connection.tables()

    close_connection()

    print("All tables fetched.")
```

```
    return tables
# Create a table by passing name and column families as parameters
def create_table(name, column_families):
    print(f"Creating table {name}...")
    tables = list_tables()
    if name not in tables:
        open_connection()
        connection.create_table(name, column_families)
        close_connection()
        print(f"Table {name} created.")
    else:
        print(f"Table {name} already present.")
# Define column families for yellow_taxi_trips table
column_families = {
    'info': dict(max_versions=5),
}
# Create the table
create_table('yellow_taxi_trips', column_families)
```

We need to write above code in create_table.py file

```
[root@ip-172-31-24-194 ~]# python -c 'import happybase'
[root@ip-172-31-24-194 ~]# vi create_table.py
```

root@ip-172-31-24-194:~

```python
import happybase

# Create connection
connection = happybase.Connection('localhost', port=9090, autoconnect=False)

# Open connection to perform operations
def open_connection():
    connection.open()

# Close the opened connection
def close_connection():
    connection.close()

# List all tables in HBase
def list_tables():
    print("Fetching all tables...")
    open_connection()
    tables = connection.tables()
    close_connection()
    print("All tables fetched.")
    return tables

# Create a table by passing name and column families as parameters
def create_table(name, column_families):
    print(f"Creating table {name}...")
    tables = list_tables()
    if name not in tables:
        open_connection()
        connection.create_table(name, column_families)
        close_connection()
        print(f"Table {name} created.")
    else:
        print(f"Table {name} already present.")

# Define column families for yellow_taxi_trips table
column_families = {
    'info': dict(max_versions=5),
}

# Create the table
create_table('yellow_taxi_trips', column_families)
```

In the above python code open_connection() and close_connection() functions are used to open and close the connection. The list_table() function is use to fetch the list of existing tables. Whereas, create_table() function is use to create the table. It takes table name and column family as arguments. In create_table function we call the list_table() function and store the list of existing table in tables variable. In if condition we check that the table already exists or not. If not then we create the table otherwise we return table already exists.

2: Now to create hbase table we need to run the create_table.py file. The commnad is “python create_table.py”

```
[root@ip-172-31-24-194 ~]# python create_table.py
Creating table yellow_taxi_trips...
Fetching all tables...
All tables fetched.
Table yellow_taxi_trips created.
[root@ip-172-31-24-194 ~]#
```

3: To check whether the table is created or not we can either go to hbase shell and run list command or use following python code:

```
[root@ip-172-31-24-194 ~]# vi list_tables.py
```

root@ip-172-31-24-194:~

```
# Listing Table
import happybase

print("connecting to HBase")
con= happybase.Connection('localhost')

con.open()
print("Connected")

print("Listing tables......")
print(con.tables())

print("Closing the connection")
con.close()

~
```

```
[root@ip-172-31-24-194 ~]# python list_tables.py
connecting to HBase
Connected
Listing tables......
[b'yellow_taxi_trips']
Closing the connection
```

Step 4: Now we can upload the data to the table. To upload data I used below commands and python code.

1: Create a python file named batch_ingest.py using vi and put the below code into it.

import happybase

import csv

```
# Create connection
connection = happybase.Connection('localhost', port=9090, autoconnect=False)
# Open connection to perform operations
def open_connection():
    connection.open()
# Close the opened connection
def close_connection():
    connection.close()
# Get the pointer to a table
def get_table(name):
    open_connection()
    table = connection.table(name)
    return table
# Batch insert data from CSV into HBase
def batch_insert_data(file_path):
    print(f"Starting batch insert for {file_path}")
    try:
        with open(file_path, 'r') as file:
            csv_reader = csv.DictReader(file)
            table = get_table('yellow_taxi_trips')  # Replace with your actual table
name
            with table.batch(batch_size=100) as b:  # Adjust batch size as needed
                for row in csv_reader:
                    # Ensure all required fields exist
```

```
            if all(field in row for field in ['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', 'airport_fee']):
                row_key = f"{row['VendorID']}:{row['tpep_pickup_datetime']}" # Unique key based on VendorID and pickup time
                # Prepare the data to be inserted
                row_data = {
                    'info:VendorID': row['VendorID'],
                    'info:tpep_pickup_datetime': row['tpep_pickup_datetime'],
                    'info:tpep_dropoff_datetime': row['tpep_dropoff_datetime'],
                    'info:passenger_count': row['passenger_count'],
                    'info:trip_distance': row['trip_distance'],
                    'info:RatecodeID': row['RatecodeID'],
                    'info:store_and_fwd_flag': row['store_and_fwd_flag'],
                    'info:PULocationID': row['PULocationID'],
                    'info:DOLocationID': row['DOLocationID'],
                    'info:payment_type': row['payment_type'],
                    'info:fare_amount': row['fare_amount'],
                    'info:extra': row['extra'],
                    'info:mta_tax': row['mta_tax'],
                    'info:tip_amount': row['tip_amount'],
                    'info:tolls_amount': row['tolls_amount'],
                    'info:improvement_surcharge': row['improvement_surcharge'],
                    'info:total_amount': row['total_amount'],
```

```
                    'info:congestion_surcharge': row['congestion_surcharge'],
                    'info:airport_fee': row['airport_fee']
                }
                # Insert data into batch
                b.put(row_key, row_data)
            else:
                print(f"Missing data for row: {row}")
    print(f"Batch insert done for {file_path}")
except Exception as e:
    print(f"Error inserting data from {file_path}: {e}")
# Insert data for both CSV files
batch_insert_data('yellow_tripdata_2017-03.csv')
batch_insert_data('yellow_tripdata_2017-04.csv')
# Close the connection
close_connection()
```

```
[root@ip-172-31-24-194 ~]# vi batch_ingest.py
```

```
root@ip-172-31-24-194:~
[root@ip-172-31-24-194 ~]# vi batch_ingest.py
            csv_reader = csv.DictReader(file)
            table = get_table('yellow_taxi_trips')  # Replace with your actual table name

            with table.batch(batch_size=100) as b:  # Adjust batch size as needed
                for row in csv_reader:
                    # Ensure all required fields exist
                    if all(field in row for field in ['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime', 'passenger_count', 'trip_distance', 'RatecodeID', 'store_and_fwd_flag', 'P
ULocationID', 'DOLocationID', 'payment_type', 'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge', 'total_amount', 'congestion_surcharge', 'airport_fee'
]):
                        row_key = f"{row['VendorID']}:{row['tpep_pickup_datetime']}"  # Unique key based on VendorID and pickup time
                        # Prepare the data to be inserted
                        row_data = {
                            'info:VendorID': row['VendorID'],
                            'info:tpep_pickup_datetime': row['tpep_pickup_datetime'],
                            'info:tpep_dropoff_datetime': row['tpep_dropoff_datetime'],
                            'info:passenger_count': row['passenger_count'],
                            'info:trip_distance': row['trip_distance'],
                            'info:RatecodeID': row['RatecodeID'],
                            'info:store_and_fwd_flag': row['store_and_fwd_flag'],
                            'info:PULocationID': row['PULocationID'],
                            'info:DOLocationID': row['DOLocationID'],
                            'info:payment_type': row['payment_type'],
                            'info:fare_amount': row['fare_amount'],
                            'info:extra': row['extra'],
                            'info:mta_tax': row['mta_tax'],
                            'info:tip_amount': row['tip_amount'],
                            'info:tolls_amount': row['tolls_amount'],
                            'info:improvement_surcharge': row['improvement_surcharge'],
                            'info:total_amount': row['total_amount'],
                            'info:congestion_surcharge': row['congestion_surcharge'],
                            'info:airport_fee': row['airport_fee']
                        }
                        # Insert data into batch
                        b.put(row_key, row_data)
                    else:
                        print(f"Missing data for row: {row}")

        print(f"Batch insert done for {file_path}")
    except Exception as e:
        print(f"Error inserting data from {file_path}: {e}")

# Insert data for both CSV files
batch_insert_data('yellow_tripdata_2017-03.csv')
batch_insert_data('yellow_tripdata_2017-04.csv')

# Close the connection
close_connection()
```

In the above python code we have used open_conncetion() and close_connection() function to open and close the connection. We have used get_table() function to connect to the table where the data is to be inserted. This function takes table name as argument to which we need to connect for inputting the data.

The batch_insert_data() is the core function that reads CSV files and inserts data into the HBase table in batches. We use with open() to open the file in read mode. csv.DictReader reads the CSV file and converts each row into a dictionary, where the keys are the column names. In this function we call the get_table() function to get reference to the required table. table.batch function helps to optimize the performance. Then, using if all() condition we ensure that all the rows have required values otherwise, the row is skipped. The row key is a unique identifier created by concatenating VendorID and tpep_pickup_datetime. row_data={.....} here we prepare row data. We Constructs a dictionary where keys follow the HBase column family and qualifier format (column_family:qualifier). To add row to batch we use b.put(row_key, row_data). We use except Exception to catch and log occur any error that occur during the batch insert.

batch_insert_data('yellow_tripdata_2017-03.csv')
batch_insert_data('yellow_tripdata_2017-04.csv')

This calls the batch_insert_data function for two CSV files, yellow_tripdata_2017-03.csv and yellow_tripdata_2017-04.csv.

2: Then we need to run the batch_ingest.py file to insert data into hbase. We use “python batch_ingest.py” command.

```
[root@ip-172-31-24-194 ~]# python batch_ingest.py
Starting batch insert for yellow_tripdata_2017-03.csv
```

After this command is executed.

Step 5: To check whether data is loaded or not we can use below commands.

1: hbase shell

2: scan 'yellow_taxi_trips', {LIMIT => 10}

```
hbase(main):003:0> scan 'yellow_taxi_trips', {LIMIT => 10}
ROW                                                    COLUMN+CELL
 1:2017-03-01 00:00:00                                 column=info:DOLocationID, timestamp=1736249062617, value=181
 1:2017-03-01 00:00:00                                 column=info:PULocationID, timestamp=1736249062617, value=142
 1:2017-03-01 00:00:00                                 column=info:RatecodeID, timestamp=1736249062617, value=1
 1:2017-03-01 00:00:00                                 column=info:VendorID, timestamp=1736249062617, value=1
 1:2017-03-01 00:00:00                                 column=info:airport_fee, timestamp=1736249062617, value=
 1:2017-03-01 00:00:00                                 column=info:congestion_surcharge, timestamp=1736249062617, value=
 1:2017-03-01 00:00:00                                 column=info:extra, timestamp=1736249062617, value=0.5
 1:2017-03-01 00:00:00                                 column=info:fare_amount, timestamp=1736249062617, value=30.0
 1:2017-03-01 00:00:00                                 column=info:improvement_surcharge, timestamp=1736249062617, value=0.3
 1:2017-03-01 00:00:00                                 column=info:mta_tax, timestamp=1736249062617, value=0.5
 1:2017-03-01 00:00:00                                 column=info:passenger_count, timestamp=1736249062617, value=1
 1:2017-03-01 00:00:00                                 column=info:payment_type, timestamp=1736249062617, value=1
 1:2017-03-01 00:00:00                                 column=info:store_and_fwd_flag, timestamp=1736249062617, value=N
 1:2017-03-01 00:00:00                                 column=info:tip_amount, timestamp=1736249062617, value=7.8
 1:2017-03-01 00:00:00                                 column=info:tolls_amount, timestamp=1736249062617, value=0.0
 1:2017-03-01 00:00:00                                 column=info:total_amount, timestamp=1736249062617, value=39.1
 1:2017-03-01 00:00:00                                 column=info:tpep_dropoff_datetime, timestamp=1736249062617, value=2017-03-01 00:34:27
 1:2017-03-01 00:00:00                                 column=info:tpep_pickup_datetime, timestamp=1736249062617, value=2017-03-01 00:00:00
 1:2017-03-01 00:00:00                                 column=info:trip_distance, timestamp=1736249062617, value=8.7
 1:2017-03-01 00:00:01                                 column=info:DOLocationID, timestamp=1736249059792, value=116
 1:2017-03-01 00:00:01                                 column=info:PULocationID, timestamp=1736249059792, value=45
 1:2017-03-01 00:00:01                                 column=info:RatecodeID, timestamp=1736249059792, value=1
 1:2017-03-01 00:00:01                                 column=info:VendorID, timestamp=1736249059792, value=1
 1:2017-03-01 00:00:01                                 column=info:airport_fee, timestamp=1736249059792, value=
 1:2017-03-01 00:00:01                                 column=info:congestion_surcharge, timestamp=1736249059792, value=
```

Hence our data is successfully loaded. To further conform this, I have done following things.

```
hbase(main):004:0> get 'yellow_taxi_trips', '1:2017-03-01 00:00:00'
COLUMN                                                 CELL
 info:DOLocationID                                     timestamp=1736249062617, value=181
 info:PULocationID                                     timestamp=1736249062617, value=142
 info:RatecodeID                                       timestamp=1736249062617, value=1
 info:VendorID                                         timestamp=1736249062617, value=1
 info:airport_fee                                      timestamp=1736249062617, value=
 info:congestion_surcharge                             timestamp=1736249062617, value=
 info:extra                                            timestamp=1736249062617, value=0.5
 info:fare_amount                                      timestamp=1736249062617, value=30.0
 info:improvement_surcharge                            timestamp=1736249062617, value=0.3
 info:mta_tax                                          timestamp=1736249062617, value=0.5
 info:passenger_count                                  timestamp=1736249062617, value=1
 info:payment_type                                     timestamp=1736249062617, value=1
 info:store_and_fwd_flag                               timestamp=1736249062617, value=N
 info:tip_amount                                       timestamp=1736249062617, value=7.8
 info:tolls_amount                                     timestamp=1736249062617, value=0.0
 info:total_amount                                     timestamp=1736249062617, value=39.1
 info:tpep_dropoff_datetime                            timestamp=1736249062617, value=2017-03-01 00:34:27
 info:tpep_pickup_datetime                             timestamp=1736249062617, value=2017-03-01 00:00:00
 info:trip_distance                                    timestamp=1736249062617, value=8.7
1 row(s) in 0.1110 seconds
```

```
hbase(main):005:0> scan 'yellow_taxi_trips', {STARTROW => '1:2017-03-01 00:00:00', STOPROW => '1:2017-03-01 01:00:00', LIMIT => 10}
ROW                                                    COLUMN+CELL
 1:2017-03-01 00:00:00                                 column=info:DOLocationID, timestamp=1736249062617, value=181
 1:2017-03-01 00:00:00                                 column=info:PULocationID, timestamp=1736249062617, value=142
 1:2017-03-01 00:00:00                                 column=info:RatecodeID, timestamp=1736249062617, value=1
 1:2017-03-01 00:00:00                                 column=info:VendorID, timestamp=1736249062617, value=1
 1:2017-03-01 00:00:00                                 column=info:airport_fee, timestamp=1736249062617, value=
 1:2017-03-01 00:00:00                                 column=info:congestion_surcharge, timestamp=1736249062617, value=
 1:2017-03-01 00:00:00                                 column=info:extra, timestamp=1736249062617, value=0.5
 1:2017-03-01 00:00:00                                 column=info:fare_amount, timestamp=1736249062617, value=30.0
 1:2017-03-01 00:00:00                                 column=info:improvement_surcharge, timestamp=1736249062617, value=0.3
 1:2017-03-01 00:00:00                                 column=info:mta_tax, timestamp=1736249062617, value=0.5
 1:2017-03-01 00:00:00                                 column=info:passenger_count, timestamp=1736249062617, value=1
 1:2017-03-01 00:00:00                                 column=info:payment_type, timestamp=1736249062617, value=1
 1:2017-03-01 00:00:00                                 column=info:store_and_fwd_flag, timestamp=1736249062617, value=N
 1:2017-03-01 00:00:00                                 column=info:tip_amount, timestamp=1736249062617, value=7.8
 1:2017-03-01 00:00:00                                 column=info:tolls_amount, timestamp=1736249062617, value=0.0
 1:2017-03-01 00:00:00                                 column=info:total_amount, timestamp=1736249062617, value=39.1
```