Práctica Hive - Spark

1. Crear las tablas payments, passenger, distance, tolls y congestion en Hive:

```
CREATE TABLE payments (VendorID int, tpep_pickup_datetime date, payment_type int,
total_amount float)
COMMENT 'Payments'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

CREATE TABLE tolls (tpep_pickup_datetime date, passenger_count int, tolls_amount float,
total_amount float)
COMMENT 'Tolls'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

CREATE TABLE passenger (tpep_pickup_datetime date, passenger_count int, total_amount float)
COMMENT 'Passenger'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

CREATE TABLE congestion (tpep_pickup_datetime date, passenger_count int,
congestion_surcharge float, total_amount float )
COMMENT 'Congestion'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

CREATE TABLE distance (tpep_pickup_datetime date, passenger_count int, trip_distance float,
total_amount float )
COMMENT 'Distance'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

Para ver los resultados en la terminal se utiliza *use tripdb* y *show tables.*

```
hive> use tripdb;
OK
Time taken: 0.05 seconds
hive> show tables;
OK
congestion
distance
passenger
payments
tolls
Time taken: 0.089 seconds, Fetched: 5 row(s)
```

2. En Hive hacer un ‘describe’ de las tablas passengers y distance:

```
hive> DESCRIBE passenger;
OK
tpep_pickup_datetime    date
passenger_count         int
total_amount            float
Time taken: 0.096 seconds, Fetched: 3 row(s)
```

```
hive> DESCRIBE distance;
OK
tpep_pickup_datetime    date
passenger_count         int
trip_distance           float
total_amount            float
Time taken: 0.118 seconds, Fetched: 4 row(s)
```

3. Hacer ingest del file ‘Yellow_tripdata_2021-01.csv’.

```
hadoop@401bec58e4c6:~/landing$ hdfs dfs -ls /ingest
Found 2 items
-rw-r--r--   1 hadoop supergroup       5462 2024-04-19 23:34 /ingest/starwars.cs
v
-rw-r--r--   1 hadoop supergroup  125981363 2024-05-07 10:36 /ingest/yellow_trip
data_2021-01.csv
```

Para los isguientes ejercicios se debe usar Pyspark. Si se desea practicar más también se puede repetir los mismos en SQL.

4. Inserción de tablas mediante Pyspark con dataframes:

```
# Carga del csv alojado en hdfs en el dataframe df.
df = spark.read.option("header", "true").csv("/ingest/yellow_tripdata_2021-01.csv")
df.show(1)

# Para la tabla payments:
df_payments = df.select(df.VendorID.cast("int"), df.tpep_pickup_datetime.cast("date"),
df.payment_type.cast("int"), df.total_amount.cast("float"))
df_payments.write.insertInto("tripdb.payments")

# Para la tabla passenger:
df_passenger = df.select(df.tpep_pickup_datetime.cast("date"), df.passenger_count.cast("int"),
df.total_amout.cast("float"))
df_passenger.write.insertInto("tripdb.passenger")

# Para la tabla tolls
df_tolls = df.select(df.tpep_pickup_datetime.cast("date"), df.passenger_count.cast("int"),
df.tolls_amount.cast("float"), df.total_amount.cast("float"))
df_tolls.write.insertInto("tripdb.tolls")

# Para la tabla congestion:
df_congestion = df.select(df.tpep_pickup_datetime.cast("date"), df.passenger_count.cast("int"),
df.congestion_surcharge.cast("float"), df.total_amount.cast("float"))
df_congestion.write.insertInto("tripdb.congestion")

# Para la tabla distance:
df_distance = df.select(df.tpep_pickup_datetime.cast("date"), df.passenger_count.cast("int"),
df.trip_distance.cast("float"), df.total_amount.cast("float"))
df_distance.write.insertInto("tripdb.distance")
```

Resultados en Hive:

```
hive> select * from payments limit 10;
OK
1       2021-01-01      2       11.8
1       2021-01-01      2       4.3
1       2021-01-01      1       51.95
1       2021-01-01      1       36.35
2       2021-01-01      1       24.36
1       2021-01-01      1       14.15
1       2021-01-01      2       17.3
1       2021-01-01      2       21.8
1       2021-01-01      4       28.8
1       2021-01-01      1       18.95
```

```
hive> select * from passenger limit 10 ;
OK
2021-01-01      1       11.8
2021-01-01      1       4.3
2021-01-01      1       51.95
2021-01-01      0       36.35
2021-01-01      1       24.36
2021-01-01      1       14.15
2021-01-01      1       17.3
2021-01-01      1       21.8
2021-01-01      1       28.8
2021-01-01      2       18.95
```

```
hive> select * from tolls limit 10;
OK
2021-01-01      1       0.0     11.8
2021-01-01      1       0.0     4.3
2021-01-01      1       0.0     51.95
2021-01-01      0       0.0     36.35
2021-01-01      1       0.0     24.36
2021-01-01      1       0.0     14.15
2021-01-01      1       0.0     17.3
2021-01-01      1       0.0     21.8
2021-01-01      1       0.0     28.8
2021-01-01      2       0.0     18.95
```

```
hive> select * from congestion limit 10;
OK
2021-01-01      1       2.5     11.8
2021-01-01      1       0.0     4.3
2021-01-01      1       0.0     51.95
2021-01-01      0       0.0     36.35
2021-01-01      1       2.5     24.36
2021-01-01      1       2.5     14.15
2021-01-01      1       0.0     17.3
2021-01-01      1       2.5     21.8
2021-01-01      1       0.0     28.8
2021-01-01      2       2.5     18.95
```

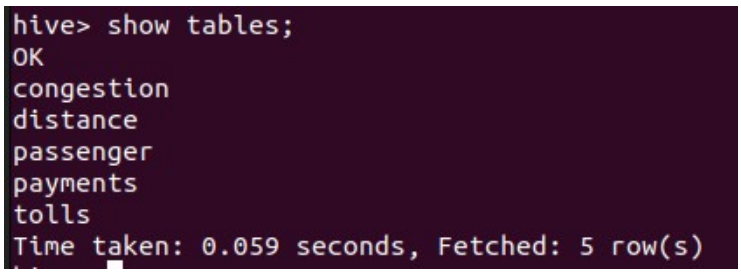
```
hive> select * from distance limit 10;
OK
2021-01-01      1       2.1     11.8
2021-01-01      1       0.2     4.3
2021-01-01      1       14.7    51.95
2021-01-01      0       10.6    36.35
2021-01-01      1       4.94    24.36
2021-01-01      1       1.6     14.15
2021-01-01      1       4.1     17.3
2021-01-01      1       5.7     21.8
2021-01-01      1       9.1     28.8
2021-01-01      2       2.7     18.95
```

5. Inserción de tablas mediante Pyspark con SQL:

Dropeamos la database

```
hive> drop database tripdb cascade;
OK
Time taken: 1.157 seconds
hive> show databases
    > ;
OK
default
Time taken: 0.07 seconds, Fetched: 1 row(s)
```

Ahora se crea nuevamente con los comandos de Hive que se escribieron al principio.



Código para crear las tablas con SQL:

```
# Carga inicial del documento en hdfs a un dataframe llamado df
df = spark.read.option("header", "true").csv("/ingest/yellow_tripdata_2021-01.csv")
df.createOrReplaceTempView("vs_df")

# Carga de la tabla payments
df_payments = spark.sql("select cast(VendorID as int), cast(tpep_pickup_datetime as date),
cast(payment_type as int),  cast(total_amount as float) from vs_df")
df_payments.createOrReplaceTempView("vs_payments_load")
spark.sql("insert into tripdb.payments select * from vs_payments_load")

# Carga de la tabla passenger

df_passengers = spark.sql("select cast(tpep_pickup_datetime as date), cast(passenger_count as int),
cast(total_amount as float) from vs_df")
df_passengers.createOrReplaceTempView("vs_passengers_load")
spark.sql("insert into tripdb.passenger select * from vs_passengers_load")

# Carga de la tabla tolls

df_tolls = spark.sql("select cast(tpep_pickup_datetime as date), cast(passenger_count as int),
cast(tolls_amount as float),  cast(total_amount as float) from vs_df")
df_tolls.createOrReplaceTempView("vs_tolls_load")
spark.sql("insert into tripdb.tolls select * from vs_tolls_load")

# Carga de la tabla congestion

df_congestion = spark.sql("select cast(tpep_pickup_datetime as date), cast(passenger_count as int),
cast(congestion_surcharge as float), cast(total_amount as float) from vs_df")
df_congestion.createOrReplaceTempView("vs_congestion_load")
spark.sql("insert into tripdb.congestion select * from vs_congestion_load")

# Carga de la tabla distance

df_distance = spark.sql("select cast(tpep_pickup_datetime as date), cast(passenger_count as int),
cast(trip_distance as float), cast(total_amount as float) from vs_df")
df_distance.createOrReplaceTempView("vs_distance_load")
spark.sql("insert into tripdb.distance select* from vs_distance_load")
```

Los resultados en Hive son los mismos que llenando las tablas sin sentencias SQL:

```
hive> select * from payments limit 10;
OK
1       2021-01-01      2       11.8
1       2021-01-01      2       4.3
1       2021-01-01      1       51.95
1       2021-01-01      1       36.35
2       2021-01-01      1       24.36
1       2021-01-01      1       14.15
1       2021-01-01      2       17.3
1       2021-01-01      2       21.8
1       2021-01-01      4       28.8
1       2021-01-01      1       18.95
```

```
hive> select * from passenger limit 10;
OK
2021-01-01      1       11.8
2021-01-01      1       4.3
2021-01-01      1       51.95
2021-01-01      0       36.35
2021-01-01      1       24.36
2021-01-01      1       14.15
2021-01-01      1       17.3
2021-01-01      1       21.8
2021-01-01      1       28.8
2021-01-01      2       18.95
```

```
hive> select * from tolls limit 10;
OK
2021-01-01      1       0.0     11.8
2021-01-01      1       0.0     4.3
2021-01-01      1       0.0     51.95
2021-01-01      0       0.0     36.35
2021-01-01      1       0.0     24.36
2021-01-01      1       0.0     14.15
2021-01-01      1       0.0     17.3
2021-01-01      1       0.0     21.8
2021-01-01      1       0.0     28.8
2021-01-01      2       0.0     18.95
```

```
hive> select * from congestion limit 10;
OK
2021-01-01      1       2.5     11.8
2021-01-01      1       0.0     4.3
2021-01-01      1       0.0     51.95
2021-01-01      0       0.0     36.35
2021-01-01      1       2.5     24.36
2021-01-01      1       2.5     14.15
2021-01-01      1       0.0     17.3
2021-01-01      1       2.5     21.8
2021-01-01      1       0.0     28.8
2021-01-01      2       2.5     18.95
```

```
hive> select * from distance limit 10;
OK
2021-01-01      1       2.1     11.8
2021-01-01      1       0.2     4.3
2021-01-01      1       14.7    51.95
2021-01-01      0       10.6    36.35
2021-01-01      1       4.94    24.36
2021-01-01      1       1.6     14.15
2021-01-01      1       4.1     17.3
2021-01-01      1       5.7     21.8
2021-01-01      1       9.1     28.8
2021-01-01      2       2.7     18.95
```