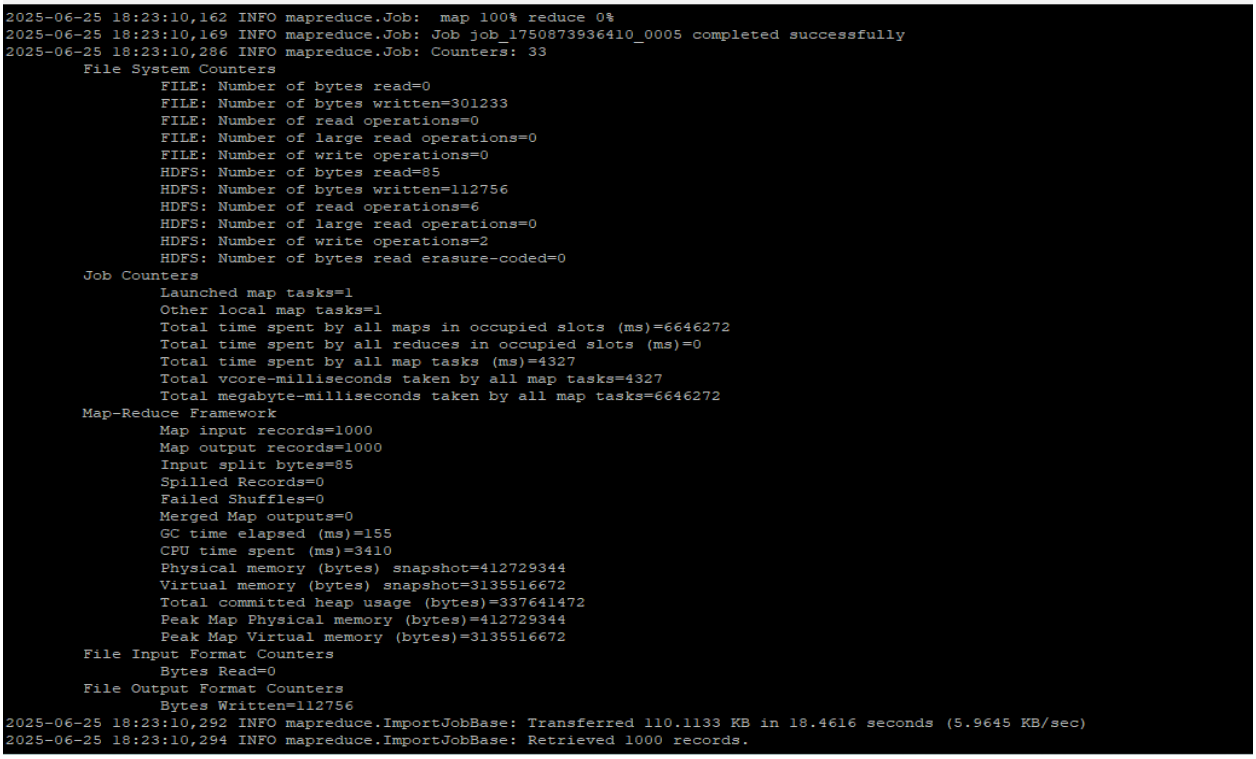
Load data from AWS RDS to Hadoop

1. Command to import data from AWS RDS to Hadoop

```
sqoop import \
> --connect jdbc:mysql://upgraddetest.cyaielc9bmnf.us-east-1.rds.amazonaws.com/testdatabase \
> --username student \
> --password STUDENT123 \
> --table bookings \
> --target-dir /user/poushali/rds_import/bookings \
> --as-parquetfile \
> --num-mappers 1
```

- Data from the bookings table is imported using this Sqoop command into a MySQL database located at upgraddetest.cyaielc9bmnf.us-east-1.rds.amazonaws.com (database: testdatabase).
- It connects with the password STUDENT123 and the username student. Parquet format, which is effective for processing and storing, is used to store the imported data in HDFS in the directory /user/poushali/rds_import/bookings.
- For smaller datasets or situations where the source database should be subjected to the least amount of pressure possible, the --num-mappers 1 option guarantees that the import operates as a single parallel operation.



```
2025-06-25 18:23:10,162 INFO mapreduce.Job:  map 100% reduce 0%
2025-06-25 18:23:10,169 INFO mapreduce.Job: Job job_1750873936410_0005 completed successfully
2025-06-25 18:23:10,286 INFO mapreduce.Job: Counters: 33
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=301233
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=85
                HDFS: Number of bytes written=112756
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=1
                Other local map tasks=1
                Total time spent by all maps in occupied slots (ms)=6646272
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=4327
                Total vcore-milliseconds taken by all map tasks=4327
                Total megabyte-milliseconds taken by all map tasks=6646272
        Map-Reduce Framework
                Map input records=1000
                Map output records=1000
                Input split bytes=85
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=155
                CPU time spent (ms)=3410
                Physical memory (bytes) snapshot=412729344
                Virtual memory (bytes) snapshot=3135516672
                Total committed heap usage (bytes)=337641472
                Peak Map Physical memory (bytes)=412729344
                Peak Map Virtual memory (bytes)=3135516672
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=112756
2025-06-25 18:23:10,292 INFO mapreduce.ImportJobBase: Transferred 110.1133 KB in 18.4616 seconds (5.9645 KB/sec)
2025-06-25 18:23:10,294 INFO mapreduce.ImportJobBase: Retrieved 1000 records.
```

2. Command to view the imported data

We use PySpark to read the imported Parquet file and inspect the schema, we did this through nano function:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("CheckSchema").getOrCreate()

df = spark.read.parquet("hdfs:///user/poushali/rds_import/bookings")
df.printSchema()
df.show(5)
```

- A Spark session with the application name "CheckSchema" is initialised by this PySpark script.
- It retrieves a Parquet file with data imported from MySQL using Sqoop from the HDFS path hdfs:///user/poushali/rds_import/bookings.
- To check if the schema was correctly interpreted, the script uses printSchema() to show the DataFrame's structure, including column names and the corresponding data types.
- It also employs show(5) to quickly examine the dataset by displaying the top five records.
- Lastly, the spark-submit check_schema.py command can be used to run this script on a Spark cluster, sending the job to Spark for processing.

3. Creating Hive table

After we identified the correct column names and data types now we created a Hive external table:

```
CREATE EXTERNAL TABLE IF NOT EXISTS rds_bookings (
   booking_id STRING,
   customer_id BIGINT,
   driver_id BIGINT,
   customer_app_version STRING,
   customer_phone_os_version STRING,
   pickup_lat DOUBLE,
   pickup_lon DOUBLE,
   drop_lat DOUBLE,
   drop_lon DOUBLE,
   pickup_timestamp BIGINT,
   drop_timestamp BIGINT,
   trip_fare INT,
   tip_amount INT,
   currency_code STRING,
```

```
    cab_color STRING,
    cab_registration_no STRING,
    customer_rating_by_driver INT,
    rating_by_customer INT,
    passenger_count INT
)
STORED AS PARQUET
LOCATION '/user/poushali/rds_import/bookings';
```

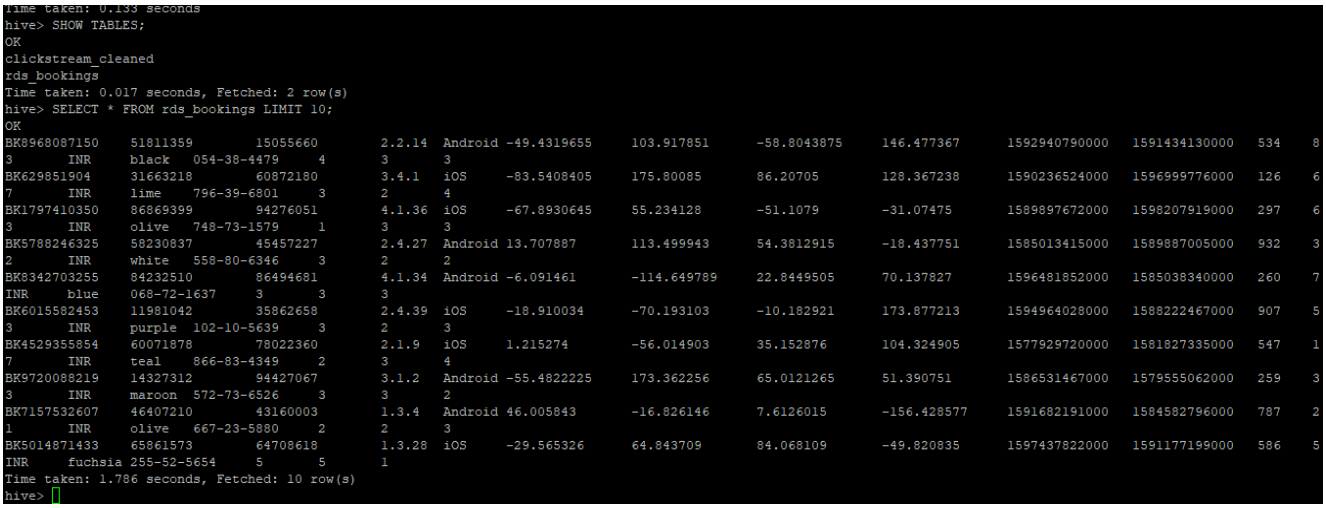
- If an external Hive table called rds_bookings does not already exist, the script builds one.
- Time stamps, trip fare, tip amount, currency code, booking ID, customer and driver IDs, app and phone OS versions, pickup and drop coordinates, booking details, customer ratings, and passenger count are just a few of the properties that can be included in this table for taxi booking records.
- Types such as STRING, BIGINT, DOUBLE, and INT are used to allocate data types according to the characteristics of each field.
- The data is already present in HDFS at the designated location: /user/poushali/rds_import/bookings. The table is saved in Parquet format, which is effective for both querying and storing data. Because it is an external table, the underlying data will remain intact even if the table is deleted.

```
[hadoop@ip-10-0-2-56 ~]$ hive
Hive Session ID = 646f6269-3ecd-40da-ae61-d91a6a9348d1

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j2.properties Async: false
hive> SHOW TABLES;
OK
clickstream_cleaned
Time taken: 0.761 seconds, Fetched: 1 row(s)
hive> CREATE EXTERNAL TABLE IF NOT EXISTS rds_bookings (
    >     booking_id STRING,
    >     customer_id BIGINT,
    >     driver_id BIGINT,
    >     customer_app_version STRING,
    >     customer_phone_os_version STRING,
    >     pickup_lat DOUBLE,
    >     pickup_lon DOUBLE,
    >     drop_lat DOUBLE,
    >     drop_lon DOUBLE,
    >     pickup_timestamp BIGINT,
    >     drop_timestamp BIGINT,
    >     trip_fare INT,
    >     tip_amount INT,
    >     currency_code STRING,
    >     cab_color STRING,
    >     cab_registration_no STRING,
    >     customer_rating_by_driver INT,
    >     rating_by_customer INT,
    >     passenger_count INT
    > )
    > STORED AS PARQUET
    > LOCATION '/user/poushali/rds_import/bookings';
OK
Time taken: 0.133 seconds
hive>
```

4. Checking imported data

```
SELECT * FROM rds_bookings LIMIT 10;
```



```
Time taken: 0.133 seconds
hive> SHOW TABLES;
OK
clickstream_cleaned
rds_bookings
Time taken: 0.017 seconds, Fetched: 2 row(s)
hive> SELECT * FROM rds_bookings LIMIT 10;
OK
BK8968087150    51811359        15055660        2.2.14  Android -49.4319655     103.917851      -58.8043875     146.477367      1592940790000   1591434130000   534     8
3       INR     black   054-38-4479     4       3       3
BK629851904     31663218        60872180        3.4.1   iOS     -83.5408405     175.80085       86.20705        128.367238      1590236524000   1596999776000   126     6
7       INR     lime    796-39-6801     3       2       4
BK1797410350    86869399        94276051        4.1.36  iOS     -67.8930645     55.234128       -51.1079        -31.07475       1589897672000   1598207919000   297     6
3       INR     olive   748-73-1579     1       3       3
BK5788246325    58230837        45457227        2.4.27  Android 13.707887       113.499943      54.3812915      -18.437751      1585013415000   1589887005000   932     3
2       INR     white   558-80-6346     3       2       2
BK8342703255    84232510        86494681        4.1.34  Android -6.091461       -114.649789     22.8449505      70.137827       1596481852000   1585038340000   260     7
INR     blue    068-72-1637     3       3       3
BK6015582453    11981042        35862658        2.4.39  iOS     -18.910034      -70.193103      -10.182921      173.877213      1594964028000   1588222467000   907     5
3       INR     purple  102-10-5639     3       2       3
BK4529355854    60071878        78022360        2.1.9   iOS     1.215274        -56.014903      35.152876       104.324905      1577929720000   1581827335000   547     1
7       INR     teal    866-83-4349     2       3       4
BK9720088219    14327312        94427067        3.1.2   Android -55.4822225     173.362256      65.0121265      51.390751       1586531467000   1579555062000   259     3
3       INR     maroon  572-73-6526     3       3       2
BK7157532607    46407210        43160003        1.3.4   Android 46.005843       -16.826146      7.6126015       -156.428577     1591682191000   1584582796000   787     2
1       INR     olive   667-23-5880     2       2       3
BK5014871433    65861573        64708618        1.3.28  iOS     -29.565326      64.843709       84.068109       -49.820835      1597437822000   1591177199000   586     5
INR     fuchsia 255-52-5654     5       5       1
Time taken: 1.786 seconds, Fetched: 10 row(s)
hive>
```