Final Submission

Problem Statement:

YourOwnCabs (YOC), an on-demand cab booking start-up, is experiencing rapid growth with a rising number of daily rides and users. The existing MySQL-based backend is unable to handle large-scale, complex queries efficiently, making it difficult for business stakeholders to access timely, data-driven insights. Quick analysis of daily, weekly, and monthly bookings, booking patterns by mobile operating systems, average fares, and total tips is essential for strategic decision-making.

Additionally, YOC generates high-volume clickstream data from user interactions within the mobile app, which is critical for understanding user behaviour and improving engagement. This data requires an optimized, scalable storage and processing solution to support seamless analytics without affecting application performance.

The challenge is to design and implement a robust big data architecture that can efficiently manage both booking and clickstream data, ensuring fast, reliable, and scalable analytics to support the company's growth.

Proposed Solution:

To address the growing data and analytics needs of YourOwnCabs (YOC), a scalable big data pipeline is proposed to efficiently manage both booking and clickstream data.

Booking Data Analytics Solution

- Ingestion: Use Sqoop to import booking data from MySQL to HDFS in scheduled batches.
- Storage: Save data in Parquet format for optimized storage and fast querying.
- Processing: Use Apache Spark for data cleaning, aggregations (daily, weekly, monthly counts, OS-wise bookings, average fare, total tips), and reporting.
- Access: Create Hive external tables for easy querying and integration with BI tools.

Clickstream Data Analytics Solution

- Ingestion: Use Apache Kafka for real-time clickstream data ingestion.
- Storage: Store clickstream data in HDFS (Parquet format) for efficient handling.

- Processing: Apply Spark Structured Streaming for real-time parsing and user behavior tracking.
- Access: Expose data via Hive tables to support user analytics and engagement strategies.

This solution ensures fast, scalable, and reliable analytics without impacting business operations.

Data:

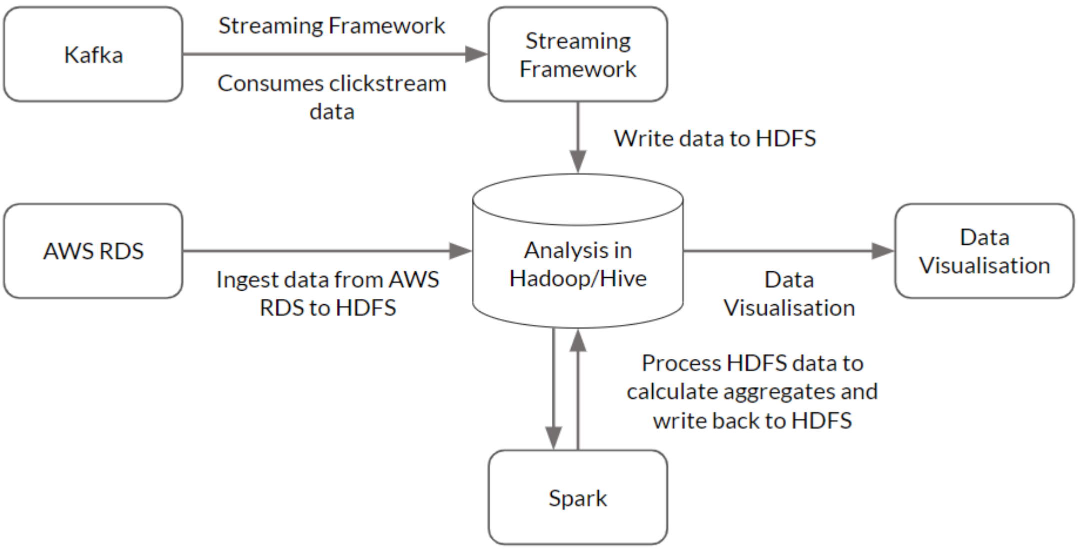
The following data will be used for this problem:

- **bookings** (The booking data is added to/updated in this table after a booking/ride is successfully completed.)
    - booking_id: Booking ID String
    - customer_id: Customer ID Number
    - driver_id: Driver ID Number
    - customer_app_version: Customer App Version String
    - customer_phone_os_version: Customer mobile operating system
    - pickup_lat: Pickup latitude
    - pickup_lon: Pickup longitude
    - drop_lat: Dropoff latitude
    - drop_lon: Dropoff longitude
    - pickup_timestamp: Timestamp at which customer was picked up
    - drop_timestamp: Timestamp at which customer was dropped at destination
    - trip_fare: Total amount of the trip
    - tip_amount: Tip amount given by customer to driver for this ride
    - currency_code: Currency Code String for the amount paid by customer
    - cab_color: Color of the cab
    - cab_registration_no: Registration number string of the vehicle
    - customer_rating_by_driver: Rating number given by driver to customer after ride
    - rating_by_customer: Rating number given by customer to driver after ride
    - passenger_count: Total count of passengers who boarded the cab for ride

- **clickstream** (All user's activity data such as click and page load):
    - customer_id: Customer ID Number
    - app_version: Customer App Version String
    - os_version: User mobile operating system
    - lat: Latitude

- lon: Longitude
- page_id: UUID of the page/screen browsed by a user
- button_id: UUID of the button clicked by a user
- is_button_click: Yes/No depending on whether a user clicked button
- is_page_view: Yes/No depending on whether a user loaded a new screen/page
- is_scroll_up: Yes/No depending on whether a user scrolled up on the current screen
- is_scroll_down: Yes/No depending on whether a user scrolled down on the current screen
- timestamp: Timestamp at which the user action is captured

Architecture and Approach



The solution involves handling two types of data: clickstream data and batch (booking) data.

Clickstream Data Flow**:** Clickstream data is captured in Kafka and consumed by a stream processing framework (like Spark Structured Streaming). The streaming data is continuously loaded into HDFS for storage and further processing.

Batch Data Flow: Booking data stored in AWS RDS is imported into HDFS using tools like Sqoop in batch mode.

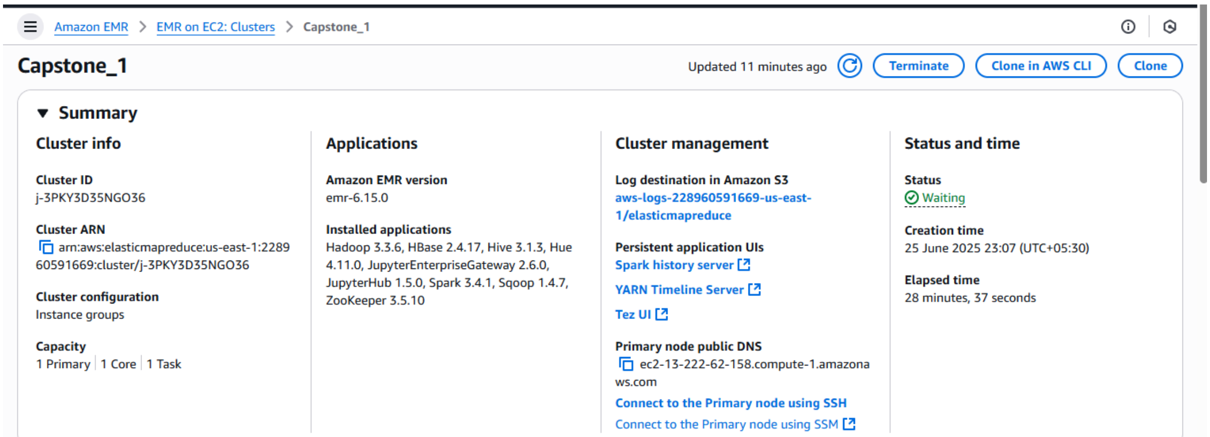
Processing and Aggregation: Both clickstream and batch data stored in HDFS are processed using Apache Spark to perform cleaning, transformations, and aggregations as needed. The processed data is written back to HDFS.

Data Access Layer: Processed and raw data in HDFS are made queryable via Hive tables, which serve as the final data layer for stakeholders. These Hive tables support efficient querying and are used for reporting and analytics.

Key Approach:

- Capture clickstream data from Kafka → Load to HDFS via stream processing.
- Ingest batch booking data from RDS → Load to HDFS via Sqoop.
- Process and aggregate data in Spark → Store results in Hive tables.
- Use Hive tables as the final query layer for business insights.

We will first create an EMR cluster with required application bundles like Hive, Sqoop, Hadoop, HBase, JupyterHub, Spark, etc.



Make sure to edit the Security groups properly.

Task 1: Write a job to consume clickstream data from Kafka and ingest to Hadoop.

I. Creating directory path in HDFS for storing the processed clickstream data and creating a **checkpoint directory** in HDFS

```
[hadoop@ip-10-0-2-56 ~]$ hdfs dfs -mkdir -p /user/poushali/clickstream
[hadoop@ip-10-0-2-56 ~]$ hdfs dfs -mkdir -p /user/poushali/checkpoints/clickstream
```

```
hadoop@ip-10-0-2-56:~
[hadoop@ip-10-0-2-56 ~]$ hdfs dfs -test -d /user/poushali/checkpoints/clickstream && echo "Directory exists" || echo "Directory does not exist"
Directory exists
[hadoop@ip-10-0-2-56 ~]$
```

II. Writing spark_kafka_to_local.py file

```
[hadoop@ip-10-0-2-56 ~]$ nano spark_kafka_to_local.py
[hadoop@ip-10-0-2-56 ~]$ cat spark_kafka_to_local.py
import os
import sys

# Environment setup
os.environ["PYSPARK_PYTHON"] = "/opt/cloudera/parcels/Anaconda/bin/python"
os.environ["JAVA_HOME"] = "/usr/java/jdk1.8.0_161/jre"
os.environ["SPARK_HOME"] = "/opt/cloudera/parcels/SPARK2-2.3.0.cloudera2-1.cdh5.
13.3.p0.316101/lib/spark2/"
os.environ["PYLIB"] = os.environ["SPARK_HOME"] + "/python/lib"
sys.path.insert(0, os.environ["PYLIB"] + "/py4j-0.10.6-src.zip")
sys.path.insert(0, os.environ["PYLIB"] + "/pyspark.zip")

from pyspark.sql import SparkSession
from pyspark.sql.functions import *

# Create Spark session
spark = SparkSession.builder.appName("KafkaToHDFSBatch").getOrCreate()

# Read Kafka data (batch mode)
df_raw = spark.read \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "18.211.252.152:9092") \
    .option("subscribe", "de-capstone5") \
    .option("startingOffsets", "earliest") \
    .option("endingOffsets", "latest") \
    .load()

# Extract and cast value to string
df = df_raw.selectExpr("CAST(value AS STRING) as value_str")

# Write to HDFS in JSON format
df.write \
    .mode("overwrite") \
    .json("/user/poushali/clickstream_json")

spark.stop()

[hadoop@ip-10-0-2-56 ~]$
```

- This PySpark script is designed to read batch data from a Kafka topic and store it in HDFS in JSON format.
- It begins by setting up the required environment variables for Python, Java, and Spark to ensure that the PySpark libraries and the Spark engine are properly accessible. The script then creates a Spark session named "KafkaToHDFSBatch" which is essential to run Spark operations.
- Using this Spark session, the script connects to a Kafka broker located at 18.211.252.152:9092 and subscribes to the topic de-capstone5. It reads all available Kafka messages starting from the earliest to the latest (batch mode, not streaming).
- From the Kafka records, it selects and casts the value field (which holds the actual message content) to a string, preparing it for storage. Finally, the script writes this extracted data to HDFS in JSON format under the directory /user/poushali/clickstream_json. The write mode is set to overwrite, meaning that if the directory already exists, its contents will be replaced.
- The script completes by stopping the Spark session to release resources.

```
spark-submit \
  --master yarn \
```

--deploy-mode client \

--packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0 \spark_kafka_to_local.py

```
        [SUCCESSFUL ] org.apache.kafka#kafka-clients;2.8.1!kafka-clients.jar (216ms)
downloading https://repo1.maven.org/maven2/com/google/code/findbugs/jsr305/3.0.0/jsr305-3.0.0.jar ...
        [SUCCESSFUL ] com.google.code.findbugs#jsr305;3.0.0!jsr305.jar (5ms)
downloading https://repo1.maven.org/maven2/org/apache/commons/commons-pool2/2.11.1/commons-pool2-2.11.1.jar ...
        [SUCCESSFUL ] org.apache.commons#commons-pool2;2.11.1!commons-pool2.jar (9ms)
downloading https://repo1.maven.org/maven2/org/spark-project/spark/unused/1.0.0/unused-1.0.0.jar ...
        [SUCCESSFUL ] org.spark-project.spark#unused;1.0.0!unused.jar (3ms)
downloading https://repo1.maven.org/maven2/org/apache/hadoop/hadoop-client-runtime/3.3.2/hadoop-client-runtime-3.3.2.jar ...
        [SUCCESSFUL ] org.apache.hadoop#hadoop-client-runtime;3.3.2!hadoop-client-runtime.jar (519ms)
downloading https://repo1.maven.org/maven2/org/lz4/lz4-java/1.8.0/lz4-java-1.8.0.jar ...
        [SUCCESSFUL ] org.lz4#lz4-java;1.8.0!lz4-java.jar (14ms)
downloading https://repo1.maven.org/maven2/org/xerial/snappy/snappy-java/1.1.8.4/snappy-java-1.1.8.4.jar ...
        [SUCCESSFUL ] org.xerial.snappy#snappy-java;1.1.8.4!snappy-java.jar(bundle) (33ms)
downloading https://repo1.maven.org/maven2/org/slf4j/slf4j-api/1.7.32/slf4j-api-1.7.32.jar ...
        [SUCCESSFUL ] org.slf4j#slf4j-api;1.7.32!slf4j-api.jar (6ms)
downloading https://repo1.maven.org/maven2/org/apache/hadoop/hadoop-client-api/3.3.2/hadoop-client-api-3.3.2.jar ...
        [SUCCESSFUL ] org.apache.hadoop#hadoop-client-api;3.3.2!hadoop-client-api.jar (198ms)
downloading https://repo1.maven.org/maven2/commons-logging/commons-logging/1.1.3/commons-logging-1.1.3.jar ...
        [SUCCESSFUL ] commons-logging#commons-logging;1.1.3!commons-logging.jar (10ms)
:: resolution report :: resolve 2132ms :: artifacts dl 1084ms
        :: modules in use:
        com.google.code.findbugs#jsr305;3.0.0 from central in [default]
        commons-logging#commons-logging;1.1.3 from central in [default]
        org.apache.commons#commons-pool2;2.11.1 from central in [default]
        org.apache.hadoop#hadoop-client-api;3.3.2 from central in [default]
        org.apache.hadoop#hadoop-client-runtime;3.3.2 from central in [default]
        org.apache.kafka#kafka-clients;2.8.1 from central in [default]
        org.apache.spark#spark-sql-kafka-0-10_2.12;3.3.0 from central in [default]
        org.apache.spark#spark-token-provider-kafka-0-10_2.12;3.3.0 from central in [default]
        org.lz4#lz4-java;1.8.0 from central in [default]
        org.slf4j#slf4j-api;1.7.32 from central in [default]
        org.spark-project.spark#unused;1.0.0 from central in [default]
        org.xerial.snappy#snappy-java;1.1.8.4 from central in [default]
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   12  |   12  |   12  |   0   ||   12  |   12  |
        ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent-4170ddae-b017-4562-8bf4-28db1731ab8d
        confs: [default]
        12 artifacts copied, 0 already retrieved (56631kB/54ms)
```

Checking the loaded file:

hdfs dfs -cat /user/poushali/clickstream_json/part-00000-68bcef36-7bc5-4ec1-86c2-f38954624e13-c000.json | head -n 5

```
[hadoop@ip-10-0-2-56 ~]$ hdfs dfs -ls /user/poushali/clickstream_json
Found 2 items
-rw-r--r--   1 hadoop hdfsadmingroup          0 2025-06-25 18:00 /user/poushali/clickstream_json/_SUCCESS
-rw-r--r--   1 hadoop hdfsadmingroup 1077675702 2025-06-25 18:00 /user/poushali/clickstream_json/part-00000-68bcef36-7bc5-4ec1-86c2-f38954624e13-c000.json
[hadoop@ip-10-0-2-56 ~]$ ^C
[hadoop@ip-10-0-2-56 ~]$ hdfs dfs -cat /user/poushali/clickstream_json/part-00000-68bcef36-7bc5-4ec1-86c2-f38954624e13-c000.json | head -n 5
{"value_str":"{\"customer_id\": \"63817546\", \"app_version\": \"4.3.5\", \"OS_version\": \"Android\", \"lat\": \"-64.847451\", \"lon\": \"-100.129008\", \"page_id\": \
"de545711-3914-4450-8c11-b17b8dabb5e1\", \"button_id\": \"a95dd57b-779f-49db-819d-b6960483e554\", \"is_button_click\": \"No\", \"is_page_view\": \"Yes\", \"is_scroll_up
\": \"No\", \"is_scroll_down\": \"No\", \"timestamp\\n\": \"2020-01-21 16:57:11\\n\"}"}
{"value_str":"{\"customer_id\": \"79407165\", \"app_version\": \"4.3.25\", \"OS_version\": \"Android\", \"lat\": \"85.4985045\", \"lon\": \"152.362461\", \"page_id\": \
"e7bc5fb2-1231-11eb-adc1-0242ac120002\", \"button_id\": \"e1e99492-17ae-11eb-adc1-0242ac120002\", \"is_button_click\": \"Yes\", \"is_page_view\": \"Yes\", \"is_scroll_u
p\": \"Yes\", \"is_scroll_down\": \"Yes\", \"timestamp\\n\": \"2020-10-22 22:45:38\\n\"}"}
{"value_str":"{\"customer_id\": \"68513803\", \"app_version\": \"2.4.13\", \"OS_version\": \"iOS\", \"lat\": \"-66.981554\", \"lon\": \"113.405679\", \"page_id\": \"de5
45711-3914-4450-8c11-b17b8dabb5e1\", \"button_id\": \"e1e99492-17ae-11eb-adc1-0242ac120002\", \"is_button_click\": \"No\", \"is_page_view\": \"No\", \"is_scroll_up\": \
"No\", \"is_scroll_down\": \"Yes\", \"timestamp\\n\": \"2020-11-01 14:51:33\\n\"}"}
{"value_str":"{\"customer_id\": \"12949619\", \"app_version\": \"1.4.34\", \"OS_version\": \"iOS\", \"lat\": \"-68.7261955\", \"lon\": \"-88.368629\", \"page_id\": \"b3
28829e-17ae-11eb-adc1-0242ac120002\", \"button_id\": \"fcba68aa-1231-11eb-adc1-0242ac120002\", \"is_button_click\": \"No\", \"is_page_view\": \"Yes\", \"is_scroll_up\":
 \"Yes\", \"is_scroll_down\": \"No\", \"timestamp\\n\": \"2020-03-23 20:13:52\\n\"}"}
{"value_str":"{\"customer_id\": \"73600498\", \"app_version\": \"4.4.30\", \"OS_version\": \"Android\", \"lat\": \"-69.4296995\", \"lon\": \"22.086996\", \"page_id\": \
"de545711-3914-4450-8c11-b17b8dabb5e1\", \"button_id\": \"a95dd57b-779f-49db-819d-b6960483e554\", \"is_button_click\": \"No\", \"is_page_view\": \"Yes\", \"is_scroll_up
\": \"Yes\", \"is_scroll_down\": \"Yes\", \"timestamp\\n\": \"2020-03-23 23:11:16\\n\"}"}
cat: Unable to write to output stream.
[hadoop@ip-10-0-2-56 ~]$
```

III. Writing spark_kafka_to_local.py file

```
[hadoop@ip-10-0-2-56 ~]$ nano spark_local_flatten.py
[hadoop@ip-10-0-2-56 ~]$ cat spark_local_flatten.py
import os
import sys
os.environ["PYSPARK_PYTHON"] = "/opt/cloudera/parcels/Anaconda/bin/python"
os.environ["JAVA_HOME"] = "/usr/java/jdk1.8.0_161/jre"
os.environ["SPARK_HOME"] = "/opt/cloudera/parcels/SPARK2-2.3.0.cloudera2-1.cdh5.13.3.p0.316101/lib/spark2/"
os.environ["PYLIB"] = os.environ["SPARK_HOME"] + "/python/lib"
sys.path.insert(0, os.environ["PYLIB"] + "/py4j-0.10.6-src.zip")
sys.path.insert(0, os.environ["PYLIB"] + "/pyspark.zip")

from pyspark.sql import SparkSession
from pyspark.sql.functions import get_json_object

# Initialize Spark session
spark = SparkSession.builder.appName("Kafka-JSON-Flatten").getOrCreate()

# Read raw Kafka JSON output from HDFS
df = spark.read.json("/user/poushali/clickstream_json/part-*.json")

# Flatten the nested JSON structure using get_json_object
df = df.select(
    get_json_object(df['value_str'], "$.customer_id").alias("customer_id"),
    get_json_object(df['value_str'], "$.app_version").alias("app_version"),
    get_json_object(df['value_str'], "$.OS_version").alias("OS_version"),
    get_json_object(df['value_str'], "$.lat").alias("lat"),
    get_json_object(df['value_str'], "$.lon").alias("lon"),
    get_json_object(df['value_str'], "$.page_id").alias("page_id"),
    get_json_object(df['value_str'], "$.button_id").alias("button_id"),
    get_json_object(df['value_str'], "$.is_button_click").alias("is_button_click"),
    get_json_object(df['value_str'], "$.is_page_view").alias("is_page_view"),
    get_json_object(df['value_str'], "$.is_scroll_up").alias("is_scroll_up"),
    get_json_object(df['value_str'], "$.is_scroll_down").alias("is_scroll_down"),
    get_json_object(df['value_str'], "$.timestamp").alias("timestamp")
)

# Write the flattened DataFrame to HDFS in CSV format
df.coalesce(1).write.format("csv").mode("overwrite").option("header", "true").save("/user/poushali/clickstream_flattened")

[hadoop@ip-10-0-2-56 ~]$
```

- The HDFS-stored Kafka JSON output is processed by this script. It starts by adding the appropriate PySpark and Py4J libraries to the system path and configuring the environment variables for Python, Java, and Spark.
- After that, a Spark session is started to manage the data processing. In order to extract important fields like customer ID, app version, OS version, position coordinates, interaction flags, and timestamps, the script reads raw JSON files from the designated HDFS directory and uses the get_json_object function to flatten the hierarchical JSON structure.
- Lastly, the result is consolidated into a single file for simpler analysis by writing the flattened data back to HDFS in CSV format with headers.

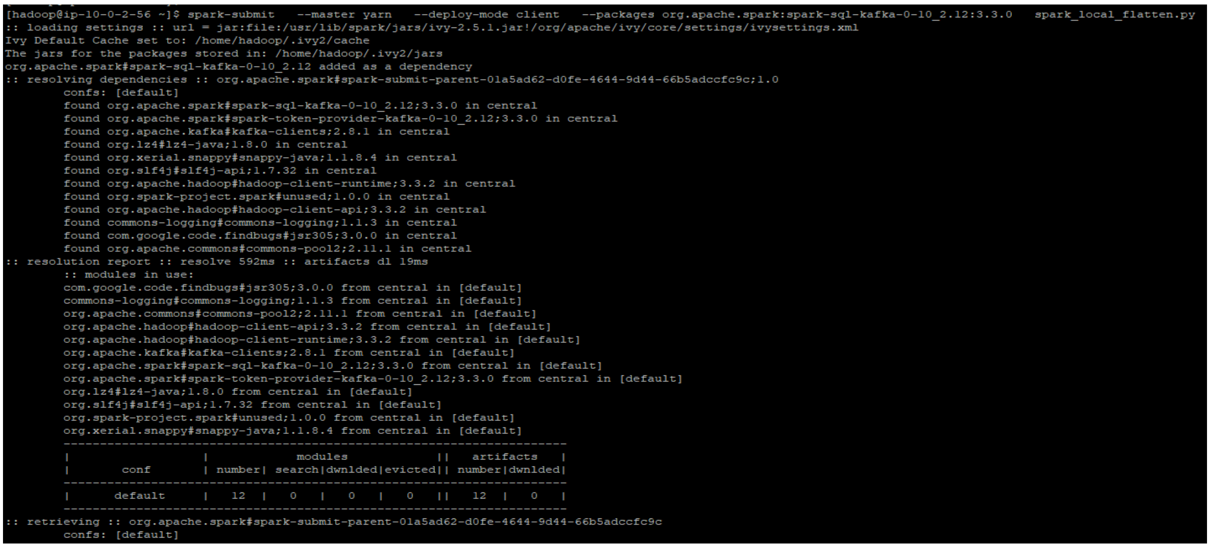
Spark – submit command:

spark-submit \

--master yarn \

--deploy-mode client \

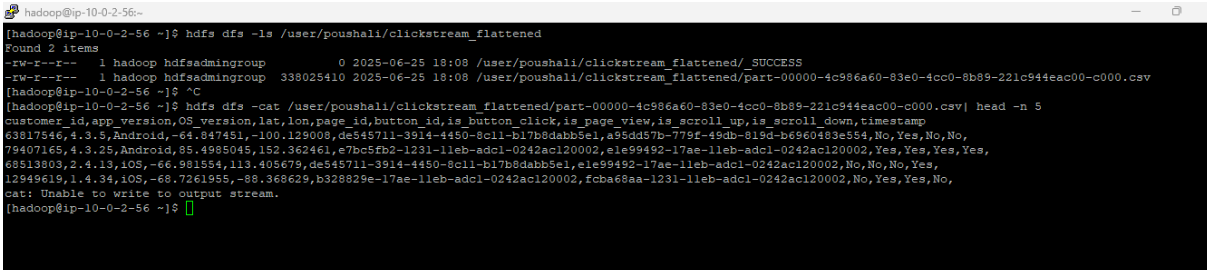
--packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0 \ spark_local_flatten.py



```
[hadoop@ip-10-0-2-56 ~]$ spark-submit   --master yarn   --deploy-mode client   --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0   spark_local_flatten.py
:: loading settings :: url = jar:file:/usr/lib/spark/jars/ivy-2.5.1.jar!/org/apache/ivy/core/settings/ivysettings.xml
Ivy Default Cache set to: /home/hadoop/.ivy2/cache
The jars for the packages stored in: /home/hadoop/.ivy2/jars
org.apache.spark#spark-sql-kafka-0-10_2.12 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-01a5ad62-d0fe-4644-9d44-66b5adccfc9c;1.0
        confs: [default]
        found org.apache.spark#spark-sql-kafka-0-10_2.12;3.3.0 in central
        found org.apache.spark#spark-token-provider-kafka-0-10_2.12;3.3.0 in central
        found org.apache.kafka#kafka-clients;2.8.1 in central
        found org.lz4#lz4-java;1.8.0 in central
        found org.xerial.snappy#snappy-java;1.1.8.4 in central
        found org.slf4j#slf4j-api;1.7.32 in central
        found org.apache.hadoop#hadoop-client-runtime;3.3.2 in central
        found org.spark-project.spark#unused;1.0.0 in central
        found org.apache.hadoop#hadoop-client-api;3.3.2 in central
        found commons-logging#commons-logging;1.1.3 in central
        found com.google.code.findbugs#jsr305;3.0.0 in central
        found org.apache.commons#commons-pool2;2.11.1 in central
:: resolution report :: resolve 592ms :: artifacts dl 19ms
        :: modules in use:
        com.google.code.findbugs#jsr305;3.0.0 from central in [default]
        commons-logging#commons-logging;1.1.3 from central in [default]
        org.apache.commons#commons-pool2;2.11.1 from central in [default]
        org.apache.hadoop#hadoop-client-api;3.3.2 from central in [default]
        org.apache.hadoop#hadoop-client-runtime;3.3.2 from central in [default]
        org.apache.kafka#kafka-clients;2.8.1 from central in [default]
        org.apache.spark#spark-sql-kafka-0-10_2.12;3.3.0 from central in [default]
        org.apache.spark#spark-token-provider-kafka-0-10_2.12;3.3.0 from central in [default]
        org.lz4#lz4-java;1.8.0 from central in [default]
        org.slf4j#slf4j-api;1.7.32 from central in [default]
        org.spark-project.spark#unused;1.0.0 from central in [default]
        org.xerial.snappy#snappy-java;1.1.8.4 from central in [default]
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   12  |   0   |   0   |   0   ||   12  |   0   |
        ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent-01a5ad62-d0fe-4644-9d44-66b5adccfc9c
        confs: [default]
```

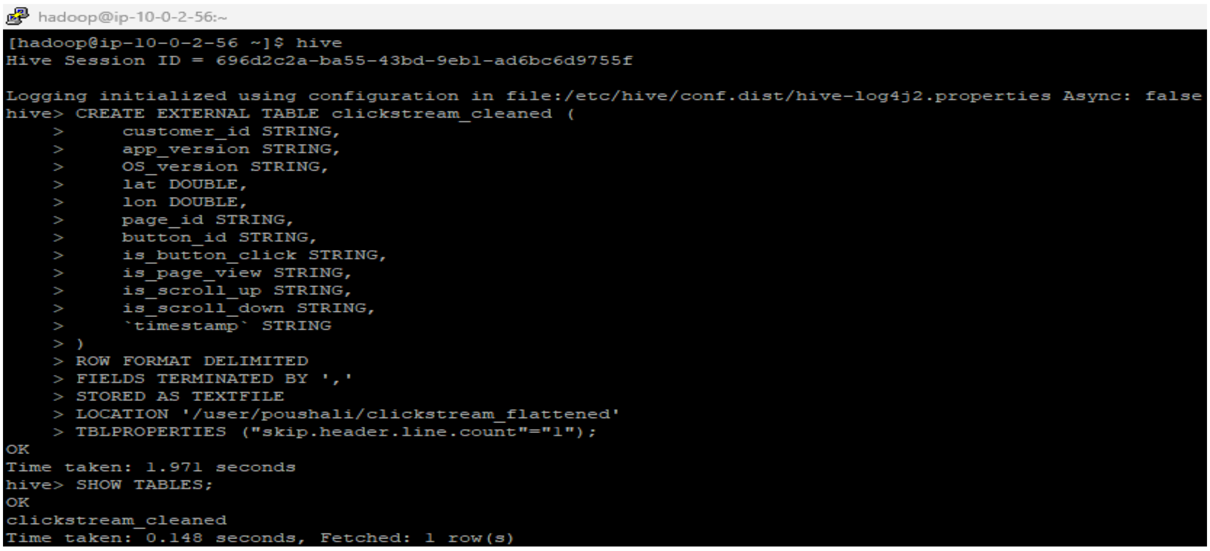
Reading the data:

hdfs dfs -cat /user/poushali/clickstream_flattened/part-00000-4c986a60-83e0-4cc0-8b89-221c944eac00-c000.csv| head -n 5



```
hadoop@ip-10-0-2-56:~
[hadoop@ip-10-0-2-56 ~]$ hdfs dfs -ls /user/poushali/clickstream_flattened
Found 2 items
-rw-r--r--   1 hadoop hdfsadmingroup          0 2025-06-25 18:08 /user/poushali/clickstream_flattened/_SUCCESS
-rw-r--r--   1 hadoop hdfsadmingroup  338025410 2025-06-25 18:08 /user/poushali/clickstream_flattened/part-00000-4c986a60-83e0-4cc0-8b89-221c944eac00-c000.csv
[hadoop@ip-10-0-2-56 ~]$ ^C
[hadoop@ip-10-0-2-56 ~]$ hdfs dfs -cat /user/poushali/clickstream_flattened/part-00000-4c986a60-83e0-4cc0-8b89-221c944eac00-c000.csv| head -n 5
customer_id,app_version,OS_version,lat,lon,page_id,button_id,is_button_click,is_page_view,is_scroll_up,is_scroll_down,timestamp
63817546,4.3.5,Android,-64.847451,-100.129008,de545711-3914-4450-8c11-b17b8dabb5e1,a95dd57b-779f-49db-819d-b6960483e554,No,Yes,No,No,
79407165,4.3.25,Android,85.4985045,152.362461,e7bc5fb2-1231-11eb-adc1-0242ac120002,e1e99492-17ae-11eb-adc1-0242ac120002,Yes,Yes,Yes,Yes,
68513803,2.4.13,iOS,-66.981554,113.405679,de545711-3914-4450-8c11-b17b8dabb5e1,e1e99492-17ae-11eb-adc1-0242ac120002,No,No,No,Yes,
12949619,1.4.34,iOS,-68.7261955,-88.368629,b328829e-17ae-11eb-adc1-0242ac120002,fcba68aa-1231-11eb-adc1-0242ac120002,No,Yes,Yes,No,
cat: Unable to write to output stream.
[hadoop@ip-10-0-2-56 ~]$
```

IV. Using Hive to create tables for cleaned clickstream data



```
hadoop@ip-10-0-2-56:~
[hadoop@ip-10-0-2-56 ~]$ hive
Hive Session ID = 696d2c2a-ba55-43bd-9eb1-ad6bc6d9755f

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j2.properties Async: false
hive> CREATE EXTERNAL TABLE clickstream_cleaned (
    >     customer_id STRING,
    >     app_version STRING,
    >     OS_version STRING,
    >     lat DOUBLE,
    >     lon DOUBLE,
    >     page_id STRING,
    >     button_id STRING,
    >     is_button_click STRING,
    >     is_page_view STRING,
    >     is_scroll_up STRING,
    >     is_scroll_down STRING,
    >     `timestamp` STRING
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE
    > LOCATION '/user/poushali/clickstream_flattened'
    > TBLPROPERTIES ("skip.header.line.count"="1");
OK
Time taken: 1.971 seconds
hive> SHOW TABLES;
OK
clickstream_cleaned
Time taken: 0.148 seconds, Fetched: 1 row(s)
```

- Data stored outside of Hive's internal storage system can be managed and queried using the clickstream_cleaned external table created by the supplied code. The EXTERNAL TABLE keyword guarantees that the actual data files in the specified HDFS location will not be deleted even if the table itself is dropped.
- The table schema captures various aspects of user activity, including customer ID, application and operating system versions, geographic coordinates (latitude and longitude), identifiers for pages and buttons, user interaction flags (such as button clicks, page views, and scrolling actions), and a timestamp indicating the event's occurrence time.
- The data is stored in CSV format (specified via ROW FORMAT DELIMITED and FIELDS TERMINATED BY ',') in the HDFS directory /user/poushali/clickstream_flattened, as declared in the LOCATION clause.

V. Checking the Hive Data

We used Hive queries to confirm that the cleaned clickstream data was successfully ingested into Hadoop. The data stored in HDFS was queried through the external table clickstream_cleaned, and the output shows that all expected fields have been populated correctly.

```
hive> SHOW TABLES;
OK
clickstream_cleaned
Time taken: 0.402 seconds, Fetched: 1 row(s)
hive> DESCRIBE clickstream_cleaned;
OK
customer_id             string
app_version             string
os_version              string
lat                     double
lon                     double
page_id                 string
button_id               string
is_button_click         string
is_page_view            string
is_scroll_up            string
is_scroll_down          string
timestamp               string
Time taken: 0.087 seconds, Fetched: 12 row(s)
```

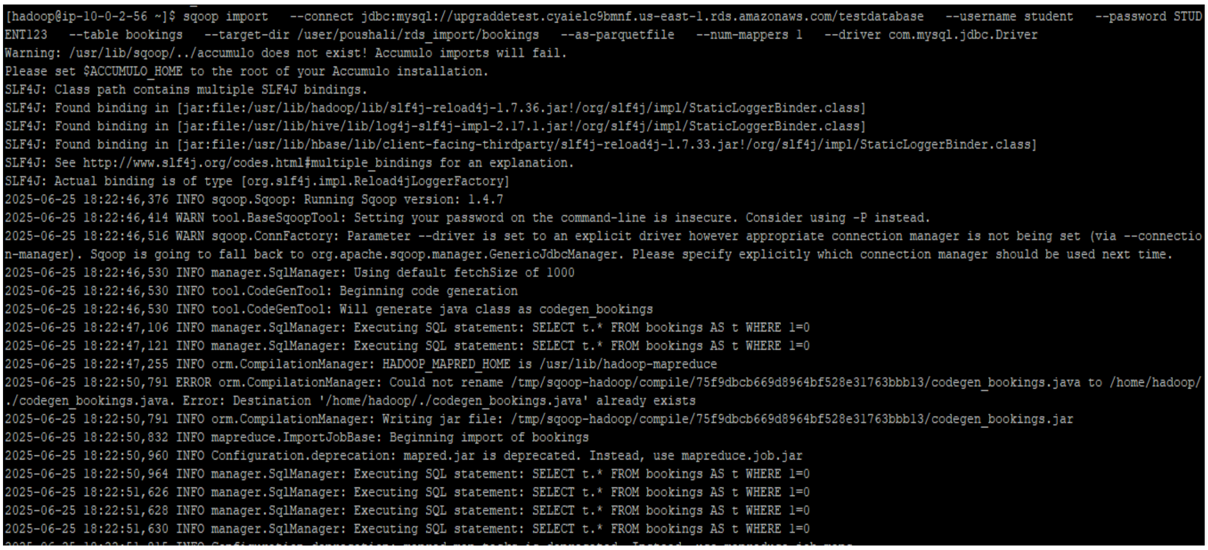
```
hive> SELECT * FROM clickstream_cleaned LIMIT 10;
OK
63817546        4.3.5   Android -64.847451      -100.129008     de545711-3914-4450-8c11-b17b8dabb5e1    a95dd57b-779f-49db-819d-b6960483e554    No      Yes     No      N
o
79407165        4.3.25  Android 85.4985045      152.362461      e7bc5fb2-1231-11eb-adc1-0242ac120002    e1e99492-17ae-11eb-adc1-0242ac120002    Yes     Yes     Yes     Y
es
68513803        2.4.13  iOS     -66.981554      113.405679      de545711-3914-4450-8c11-b17b8dabb5e1    e1e99492-17ae-11eb-adc1-0242ac120002    No      No      No      Y
es
12949619        1.4.34  iOS     -68.7261955     -88.368629      b328829e-17ae-11eb-adc1-0242ac120002    fcba68aa-1231-11eb-adc1-0242ac120002    No      Yes     Yes     N
o
73600498        4.4.30  Android -69.4296995     22.086996       de545711-3914-4450-8c11-b17b8dabb5e1    a95dd57b-779f-49db-819d-b6960483e554    No      Yes     Yes     Y
es
20001536        3.3.14  iOS     -32.799864      91.165241       de545711-3914-4450-8c11-b17b8dabb5e1    e1e99492-17ae-11eb-adc1-0242ac120002    No      Yes     No      N
o
45299375        1.4.22  iOS     24.030285       91.460414       e7bc5fb2-1231-11eb-adc1-0242ac120002    a95dd57b-779f-49db-819d-b6960483e554    Yes     Yes     Yes     Y
es
59008919        4.2.33  Android -72.712776      132.619151      e7bc5fb2-1231-11eb-adc1-0242ac120002    fcba68aa-1231-11eb-adc1-0242ac120002    Yes     Yes     Yes     Y
es
61809494        2.3.18  Android 82.4520345      -162.501195     de545711-3914-4450-8c11-b17b8dabb5e1    a95dd57b-779f-49db-819d-b6960483e554    Yes     No      No      N
o
49573886        3.4.4   iOS     31.839155       153.782118      b328829e-17ae-11eb-adc1-0242ac120002    fcba68aa-1231-11eb-adc1-0242ac120002    Yes     No      No      Y
es
Time taken: 2.484 seconds, Fetched: 10 row(s)
hive>
```

Task 2: Write a script to ingest the relevant bookings data from AWS RDS to Hadoop.

I. Command to import data from 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.



```
[hadoop@ip-10-0-2-56 ~]$ sqoop import   --connect jdbc:mysql://upgraddetest.cyaielc9bmnf.us-east-1.rds.amazonaws.com/testdatabase   --username student   --password STUD
ENT123   --table bookings   --target-dir /user/poushali/rds_import/bookings   --as-parquetfile   --num-mappers 1   --driver com.mysql.jdbc.Driver
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hadoop/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/hive/lib/log4j-slf4j-impl-2.17.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/hbase/lib/client-facing-thirdparty/slf4j-reload4j-1.7.33.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Reload4jLoggerFactory]
2025-06-25 18:22:46,376 INFO sqoop.Sqoop: Running Sqoop version: 1.4.7
2025-06-25 18:22:46,414 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
2025-06-25 18:22:46,516 WARN sqoop.ConnFactory: Parameter --driver is set to an explicit driver however appropriate connection manager is not being set (via --connectio
n-manager). Sqoop is going to fall back to org.apache.sqoop.manager.GenericJdbcManager. Please specify explicitly which connection manager should be used next time.
2025-06-25 18:22:46,530 INFO manager.SqlManager: Using default fetchSize of 1000
2025-06-25 18:22:46,530 INFO tool.CodeGenTool: Beginning code generation
2025-06-25 18:22:46,530 INFO tool.CodeGenTool: Will generate java class as codegen_bookings
2025-06-25 18:22:47,106 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM bookings AS t WHERE 1=0
2025-06-25 18:22:47,121 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM bookings AS t WHERE 1=0
2025-06-25 18:22:47,255 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/lib/hadoop-mapreduce
2025-06-25 18:22:50,791 ERROR orm.CompilationManager: Could not rename /tmp/sqoop-hadoop/compile/75f9dbcb669d8964bf528e31763bbb13/codegen_bookings.java to /home/hadoop/
./codegen_bookings.java. Error: Destination '/home/hadoop/./codegen_bookings.java' already exists
2025-06-25 18:22:50,791 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-hadoop/compile/75f9dbcb669d8964bf528e31763bbb13/codegen_bookings.jar
2025-06-25 18:22:50,832 INFO mapreduce.ImportJobBase: Beginning import of bookings
2025-06-25 18:22:50,960 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
2025-06-25 18:22:50,964 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM bookings AS t WHERE 1=0
2025-06-25 18:22:51,626 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM bookings AS t WHERE 1=0
2025-06-25 18:22:51,628 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM bookings AS t WHERE 1=0
2025-06-25 18:22:51,630 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM bookings AS t WHERE 1=0
```

```
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.
```

II. 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.

III. Creating Hive table for RDS data

```
[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>
```

- 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.

```
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>
```

IV. Checking imported data


```
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>
```


Task 03 Create aggregates for finding date-wise total bookings using the Spark script.

I. Running datewise_bookings_aggregates_spark.py

```
[hadoop@ip-10-0-2-56 ~]$ nano datewise_bookings_aggregates_spark.py
[hadoop@ip-10-0-2-56 ~]$ cat datewise_bookings_aggregates_spark.py
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_unixtime, to_date, col

# Initialize Spark session
spark = SparkSession.builder.appName("DateWiseBookings").getOrCreate()

# Read parquet data
df = spark.read.parquet("hdfs:///user/poushali/rds_import/bookings")

# Extract date from timestamp and aggregate
df_with_date = df.withColumn("booking_date", to_date(from_unixtime(col("pickup_timestamp") / 1000)))
agg_df = df_with_date.groupBy("booking_date").count().withColumnRenamed("count", "total_bookings")

# Save it to HDFS directly
agg_df.coalesce(1).write.csv("hdfs:///user/poushali/datewise_bookings_output", header=True, mode="overwrite")
[hadoop@ip-10-0-2-56 ~]$
```

- To begin, the script creates a Spark session called "DateWiseBookings" and reads Parquet data from HDFS, which is accessible at hdfs:///user/poushali/rds_import/bookings.
- After that, it uses the from_unixtime and to_date methods to transform the pickup_timestamp field, which is in Unix time (milliseconds), into a date that can be read by humans.
- These changed dates are stored in a new column called booking_date. The groupBy and count methods are then used to aggregate the data and get the total number of bookings for each date.
- In the directory hdfs:///user/poushali/datewise_bookings_output, the generated DataFrame—which includes booking dates and the total number of bookings associated with them—is saved back to HDFS as a CSV file.
- The header=True option adds column headings to the CSV file, and the coalesce(1) function guarantees that the output is recorded to a single CSV file.
- Any existing files in the output directory will be replaced if the mode="overwrite" option is used.

```
spark-submit \
  --master yarn \
  --deploy-mode client \
  --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0 \
  datewise_bookings_aggregates_spark.py
```

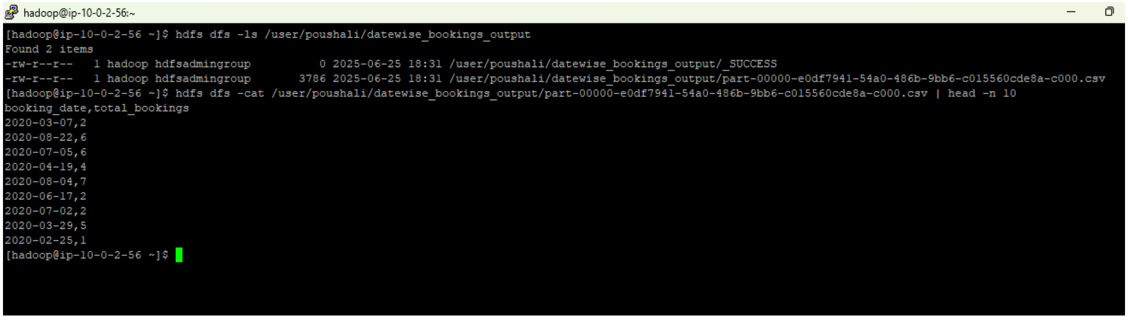
II. Command to move the csv file to HDFS

```
agg_df.coalesce(1).write.csv("hdfs:///user/poushali/datewise_bookings_output", header=True, mode="overwrite")
```

- agg_df: This is your starting point—a collection of data that already summarizes bookings by date.
- .coalesce(1): This is a crucial step for managing output. In a distributed system like Spark, data is often split into many parts (partitions). Without coalesce(1), Spark would typically create a separate CSV file for each of these partitions. By using .coalesce(1), you're telling Spark to combine all those partitions into a single one, ensuring that your output is just one consolidated CSV file rather than many smaller ones.
- .write.csv(...): This command initiates the process of saving your data in the popular CSV (Comma Separated Values) format.
- "hdfs:///user/poushali/datewise_bookings_output": This specifies the exact location where your CSV file will be stored. It's an HDFS path, meaning it's going onto a Hadoop Distributed File System.

- header=True: This option is for readability. It ensures that when you open your CSV file, the first row will clearly display the names of your columns, making it easy to understand what each column represents.
- mode="overwrite": This is an important safety and convenience feature. If there are already files in the specified HDFS location, this option tells Spark to delete them and replace them with your new output. If you didn't include this (or set it to "error"), Spark would stop and give you an error if it found existing files, preventing accidental overwrites.

III. Checking the generated file using command
hdfs dfs -cat user/poushali/datewise_bookings_output/part-00000-e0df7941-54a0-486b-9bb6-c015560cde8a-c000.csv | head -n 5



Task 04 Creating Hive Tables

I. Hive table for clickstream data:

```
CREATE EXTERNAL TABLE clickstream_cleaned (
    customer_id STRING,
    app_version STRING,
    OS_version STRING,
    lat DOUBLE,
    lon DOUBLE,
    page_id STRING,
    button_id STRING,
    is_button_click STRING,
    is_page_view STRING,
    is_scroll_up STRING,
    is_scroll_down STRING,
    `timestamp` STRING
```

```
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY ','
 STORED AS TEXTFILE
 LOCATION '/user/poushali/clickstream_flattened';
```

II. Hive table for Booking data:

```
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';

III. Hive table for aggregated data:

```
hive> CREATE TABLE datewise_booking_aggregates (
    >   booking_date DATE,
    >   total_bookings INT
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 0.234 seconds
```

- The datewise_booking_aggregates table is created by the supplied Hive query and is intended to hold daily booking summaries.
- Booking_date, which records the date of each booking activity, and total_bookings, which stores the total number of bookings for that particular date, make up its two columns.
- Using ROW FORMAT DELIMITED and fields separated by commas (FIELDS TERMINATED BY ','), the table is set up to handle data in a CSV format. \
- Since the table data is saved as a TEXTFILE, it will be in plain text format. Simple, legible datasets are usually stored in this configuration, which is also appropriate for loading outputs such as the CSV file produced by the PySpark aggregation.

For clarity and consistency, we renamed the previous tables:

ALTER TABLE clickstream_cleaned RENAME TO clickstream_data;

ALTER TABLE rds_bookings RENAME TO bookings_data;

```
[hadoop@ip-10-0-2-56 ~]$ hive
Hive Session ID = 0ce052ed-13ac-4063-aca4-adc765764a0d

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j2.properties Async: false
hive> show tables;
OK
clickstream_cleaned
rds_bookings
Time taken: 0.788 seconds, Fetched: 2 row(s)
hive> ALTER TABLE clickstream_cleaned RENAME TO clickstream_data;
OK
Time taken: 0.109 seconds
hive> ALTER TABLE rds_bookings RENAME TO bookings_data;
OK
Time taken: 0.093 seconds
hive> show tables;
OK
bookings_data
clickstream_data
Time taken: 0.028 seconds, Fetched: 2 row(s)
hive>
```

IV. Command to load the data into Hive tables:

Only datewise_booking_aggregates requires a data load from HDFS.

Command

LOAD DATA INPATH '/user/poushali/datewise_bookings_output/part-00000-80d4e987-cbeb-44b9-b304-703ae2d9adcb-c000.csv'

INTO TABLE datewise_booking_aggregates;

V. Checking all the 3 tables:

```
hive> show tables;
OK
bookings_data
clickstream_data
datewise_booking_aggregates
Time taken: 0.035 seconds, Fetched: 3 row(s)
```

SELECT * FROM clickstream_data LIMIT 10;

```
hive> SELECT * FROM clickstream_data LIMIT 10;
OK
customer_id     app_version     OS_version      NULL    NULL    page_id button_id       is_button_click is_page_view    is_scroll_up    is_scroll_do
wn      timestamp
98215369        3.1.7   Android 80.203577       -68.44555       e7bc5fb2-1231-11eb-adc1-0242ac120002    fcba68aa-1231-11eb-adc1-0242ac120002    No N
o       Yes     No
22684722        3.1.27  iOS     43.877177       -51.940886      e7bc5fb2-1231-11eb-adc1-0242ac120002    fcba68aa-1231-11eb-adc1-0242ac120002    YesY
es      Yes     No
48680451        1.4.39  Android 59.04954        119.355631      b328829e-17ae-11eb-adc1-0242ac120002    fcba68aa-1231-11eb-adc1-0242ac120002    YesY
es      No      Yes
38371811        3.2.24  Android -83.155814      -164.111381     b328829e-17ae-11eb-adc1-0242ac120002    fcba68aa-1231-11eb-adc1-0242ac120002    No N
o       No      No
69281860        3.1.1   iOS     79.6680345      43.156676       b328829e-17ae-11eb-adc1-0242ac120002    fcba68aa-1231-11eb-adc1-0242ac120002    YesN
o       Yes     Yes
47178276        4.2.1   iOS     46.8875865      -64.982925      de545711-3914-4450-8c11-b17b8dabb5e1    fcba68aa-1231-11eb-adc1-0242ac120002    YesN
o       Yes     Yes
91573295        2.3.20  iOS     -47.019646      67.117437       b328829e-17ae-11eb-adc1-0242ac120002    a95dd57b-779f-49db-819d-b6960483e554    No N
o       Yes     Yes
94375933        3.1.10  Android -51.38097       105.899543      de545711-3914-4450-8c11-b17b8dabb5e1    e1e99492-17ae-11eb-adc1-0242ac120002    No Y
es      No      Yes
21864180        4.4.28  iOS     -57.2782655     132.991072      de545711-3914-4450-8c11-b17b8dabb5e1    fcba68aa-1231-11eb-adc1-0242ac120002    YesY
es      Yes     Yes
Time taken: 1.67 seconds, Fetched: 10 row(s)
```

SELECT * FROM bookings_data LIMIT 10;

```
hive> SELECT * FROM bookings_data LIMIT 10;
OK
BK8968087150    51811359        15055660        2.2.14  Android -49.4319655     103.917851      -58.8043875     146.477367      1592940790000   1591
434130000       534     83      INR     black   054-38-4479     4       3       3
BK629851904     31663218        60872180        3.4.1   iOS     -83.5408405     175.80085       86.20705        128.367238      1590236524000   1596
999776000       126     67      INR     lime    796-39-6801     3       2       4
BK1797410350    86869399        94276051        4.1.36  iOS     -67.8930645     55.234128       -51.1079        -31.07475       1589897672000   1598
207919000       297     63      INR     olive   748-73-1579     1       3       3
BK5788246325    58230837        45457227        2.4.27  Android 13.707887       113.499943      54.3812915      -18.437751      1585013415000   1589
887005000       932     32      INR     white   558-80-6346     3       2       2
BK8342703255    84232510        86494681        4.1.34  Android -6.091461       -114.649789     22.8449505      70.137827       1596481852000   1585
038340000       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   1588
222467000       907     53      INR     purple  102-10-5639     3       2       3
BK4529355854    60071878        78022360        2.1.9   iOS     1.215274        -56.014903      35.152876       104.324905      1577929720000   1581
827335000       547     17      INR     teal    866-83-4349     2       3       4
BK9720088219    14327312        94427067        3.1.2   Android -55.4822225     173.362256      65.0121265      51.390751       1586531467000   1579
555062000       259     33      INR     maroon  572-73-6526     3       3       2
BK7157532607    46407210        43160003        1.3.4   Android 46.005843       -16.826146      7.6126015       -156.428577     1591682191000   1584
582796000       787     21      INR     olive   667-23-5880     2       2       3
BK5014871433    65861573        64708618        1.3.28  iOS     -29.565326      64.843709       84.068109       -49.820835      1597437822000   1591
177199000       586     5       INR     fuchsia 255-52-5654     5       5       1
Time taken: 0.063 seconds, Fetched: 10 row(s)
```

SELECT * FROM datewise_booking_aggregates LIMIT 10;

```
hive> SELECT * FROM datewise_booking_aggregates LIMIT 10;
OK
NULL    NULL
2020-03-07      2
2020-08-22      6
2020-07-05      6
2020-04-19      4
2020-08-04      7
2020-06-17      2
2020-07-02      2
2020-03-29      5
2020-02-25      1
Time taken: 0.082 seconds, Fetched: 10 row(s)
```

Counts of each table:

```
hive> SELECT COUNT(*) AS total_records FROM bookings_data;
Query ID = hadoop_20250625193407_8a192c84-f4d2-4428-81a1-9f5ccf3b315f
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1750873936410_0017)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 4.81 s
----------------------------------------------------------------------------------------------
OK
total_records
1000
Time taken: 5.128 seconds, Fetched: 1 row(s)
hive>
```

```
hive> SELECT COUNT(*) AS total_records FROM datewise_booking_aggregates;
Query ID = hadoop_20250625193716_ec058ceb-7053-42e2-8645-ae36b039e381
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1750873936410_0017)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 4.98 s
----------------------------------------------------------------------------------------------
OK
total_records
290
Time taken: 5.331 seconds, Fetched: 1 row(s)
hive>
```

Task 5: Calculate the total number of different drivers for each customer.

Query:

```
SELECT
    customer_id,
    COUNT(DISTINCT driver_id) AS Distinct_no_of_drivers
FROM    bookings_data
GROUP BY    customer_id
ORDER BY    customer_id;
```

- This SQL query examines the bookings_data table's customer-driver interactions. It determines how many distinct drivers are connected to every customer. The query makes sure that each customer's records are aggregated independently by classifying the data according to customer_id. The number of different drivers each client has come across is then determined using the COUNT(DISTINCT driver_id) function.
- Lastly, a clear and structured representation of the number of drivers who served each client is provided by sorting the results by customer_id in ascending order. Understanding patterns of driver distribution and client involvement can be aided by this kind of information.

```
hadoop@ip-10-0-2-56:~
hive> SET hive.cli.print.header=true;
hive> SELECT
    >     customer_id,
    >     COUNT(DISTINCT driver_id) AS Distinct_no_of_drivers
    > FROM
    >     bookings_data
    > GROUP BY
    >     customer_id
    > ORDER BY
    >     customer_id;
Query ID = hadoop_20250625190454_497935d6-3596-4227-a0cc-4ea53751bca1
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1750873936410_0011)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      2          2        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 5.67 s
----------------------------------------------------------------------------------------------
OK
customer_id     distinct_no_of_drivers
10022393        1
10058402        1
10339567        1
10435129        1
10555335        1
10592274        1
10614890        1
10678994        1
11264797        1
11353346        1
11418437        1
11438890        1
11454977        1
11479815        1
11518953        1
11580321        1
11596512        1
```

```
hadoop@ip-10-0-2-56:~
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 5.67 s
----------------------------------------------------------------------------------------------
OK
customer_id     distinct_no_of_drivers
10022393        1
10058402        1
10339567        1
10435129        1
10555335        1
10592274        1
10614890        1
10678994        1
11264797        1
11353346        1
11418437        1
11438890        1
11454977        1
11479815        1
11518953        1
11580321        1
11596512        1
11608791        1
11655671        1
11757536        1
11764909        1
11860278        1
11981042        1
12106105        1
12142182        1
12312603        1
12334699        1
12367832        1
12856708        1
12885363        1
12913608        1
12914577        1
12966909        1
13015449        1
13229062        1
13262795        1
13356177        1
13387493        1
```

The following Output matches validation document.

Task 6: Calculate the total rides taken by each customer.

Query:

```
SELECT
    customer_id,
    COUNT(booking_id) AS NUMBER_OF_TOTAL_RIDES
FROM    bookings_data
GROUP BY    customer_id
ORDER BY    customer_id;
```

- Using the bookings_data table, this SQL query determines how many rides each client has taken overall. After choosing the customer_id, it counts how many booking_id entries there are linked to each customer.
- The query organises all reservations under each distinct customer by utilising the GROUP BY clause on customer_id. A summary of the number of rides each customer has completed is provided by the outcome.
- It is simple to examine the total number of rides for every client in an orderly fashion because the final output is sorted by customer_id.

hadoop@ip-10-0-2-56:~

```
99943608        1
99947969        1
Time taken: 6.025 seconds, Fetched: 1000 row(s)
hive> SELECT
    >     customer_id,
    >     COUNT(booking_id) AS NUMBER_OF_TOTAL_RIDES
    > FROM
    >     bookings_data
    > GROUP BY
    >     customer_id
    > ORDER BY
    >     customer_id;
Query ID = hadoop_20250625190602_178f0243-77fa-49a9-a22e-aa60aa8949ca
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1750873936410_0011)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 ..........   container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ......   container     SUCCEEDED      2          2        0        0       0       0
Reducer 3 ......   container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 6.11 s
----------------------------------------------------------------------------------------------
OK
customer_id     number_of_total_rides
10022393        1
10058402        1
10339567        1
10435129        1
10555335        1
10592274        1
10614890        1
10678994        1
11264797        1
11353346        1
11418437        1
11438890        1
11454977        1
11479815        1
```

hadoop@ip-10-0-2-56:~

```
Reducer 2 ......   container     SUCCEEDED      2          2        0        0       0       0
Reducer 3 ......   container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 6.11 s
----------------------------------------------------------------------------------------------
OK
customer_id     number_of_total_rides
10022393        1
10058402        1
10339567        1
10435129        1
10555335        1
10592274        1
10614890        1
10678994        1
11264797        1
11353346        1
11418437        1
11438890        1
11454977        1
11479815        1
11518953        1
11580321        1
11596512        1
11608791        1
11655671        1
11757536        1
11764909        1
11860278        1
11981042        1
12106105        1
12142182        1
12312603        1
12334699        1
12367832        1
12856708        1
12885363        1
12913608        1
12914577        1
12966909        1
13015449        1
13229062        1
13262795        1
```

The above Output matched the validation document.

Task 7: Find the total visits made by each customer on the booking page and the total 'Book Now' button presses. This can show the conversion ratio.

The booking page id is 'e7bc5fb2-1231-11eb-adc1-0242ac120002'.

The Book Now button id is 'fcba68aa-1231-11eb-adc1-0242ac120002'. You also need to calculate the conversion ratio as part of this task. Conversion ratio can be calculated as Total 'Book Now' Button Press/Total Visits made by customer on the booking page.

Query:

```
SELECT
  COUNT(CASE
    WHEN page_id = 'e7bc5fb2-1231-11eb-adc1-0242ac120002'
    AND is_page_view = 'Yes' THEN 1
  END) AS total_page_visits,

  COUNT(CASE
    WHEN button_id = 'fcba68aa-1231-11eb-adc1-0242ac120002'
    AND is_button_click = 'Yes' THEN 1
  END) AS total_button_pressed,

  ROUND(
    COUNT(CASE
      WHEN button_id = 'fcba68aa-1231-11eb-adc1-0242ac120002'
      AND is_button_click = 'Yes' THEN 1
    END) * 1.0 /
    COUNT(CASE
      WHEN page_id = 'e7bc5fb2-1231-11eb-adc1-0242ac120002'
      AND is_page_view = 'Yes' THEN 1
    END), 4
```

) AS conversion_ratio

FROM clickstream_data;

- The purpose of this SQL query is to assess user interaction effectiveness and engagement on a particular webpage.
- It uses the clickstream_data table to compute three important metrics. It first counts the records where the given page_id was viewed and tagged as 'Yes' in the is_page_view field in order to calculate the overall number of visits to a specific page.
- Second, it counts the number of times a particular button on that page was clicked; this is verified by the is_button_click field being set to 'Yes' and is recognised by a unique button_id.
- Lastly, by dividing the total number of button clicks by the total number of page visits and rounding the result to four decimal places, the query determines the conversion ratio.
- This conversion ratio offers important insights into user behaviour and the performance of the page design by showing how well the page drives user activities.

```
hadoop@ip-10-0-2-56:~
99943608        1
99947969        1
Time taken: 6.485 seconds, Fetched: 1000 row(s)
hive> SELECT
    >     COUNT(CASE
    >         WHEN page_id = 'e7bc5fb2-1231-11eb-adc1-0242ac120002'
    >         AND is_page_view = 'Yes' THEN 1
    >     END) AS total_page_visits,
    >
    >     COUNT(CASE
    >         WHEN button_id = 'fcba68aa-1231-11eb-adc1-0242ac120002'
    >         AND is_button_click = 'Yes' THEN 1
    >     END) AS total_button_pressed,
    >
    >     ROUND(
    >         COUNT(CASE
    >             WHEN button_id = 'fcba68aa-1231-11eb-adc1-0242ac120002'
    >             AND is_button_click = 'Yes' THEN 1
    >         END) * 1.0 /
    >         COUNT(CASE
    >             WHEN page_id = 'e7bc5fb2-1231-11eb-adc1-0242ac120002'
    >             AND is_page_view = 'Yes' THEN 1
    >         END), 4
    >     ) AS conversion_ratio
    > FROM clickstream_data;
Query ID = hadoop_20250625190919_1e2dd756-5549-4a65-a8a3-dfd0eebd4336
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1750873936410_0011)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED     10         10        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 12.83 s
----------------------------------------------------------------------------------------------
OK
total_page_visits       total_button_pressed    conversion_ratio
436010  422418  0.9688
Time taken: 13.416 seconds, Fetched: 1 row(s)
hive>
```

We got the same conversion ratio of 0.9688 as we have in validation document.

Task 8: Calculate the count of all trips done on black cabs.

Query:

```
WITH black_cabs AS (
   SELECT *
   FROM bookings_data
   WHERE cab_color = 'black'
)
SELECT  COUNT(booking_id) AS TOTAL_TRIPS_BY_BLACK_CABS
FROM black_cabs;
```

- Using the bookings_data table, this SQL query determines the total number of trips made by black taxis.
- A Common Table Expression (CTE) called black_cabs is initially created, filtering and storing any records with the cab_color set to 'black'. The query then counts the total number of booking_id entries—which indicate the total number of trips made in black taxis—using this filtered dataset.
- The total number of trips made by black taxis is the only value displayed in the final result. This method assists in separating particular data subsets for targeted investigation.

```
OK
total_page_visits       total_button_pressed    conversion_ratio
436010  422418  0.9688
Time taken: 13.416 seconds, Fetched: 1 row(s)
hive> WITH black_cabs AS (
    >     SELECT *
    >     FROM bookings_data
    >     WHERE cab_color = 'black'
    > )
    >
    > SELECT
    >     COUNT(booking_id) AS TOTAL_TRIPS_BY_BLACK_CABS
    > FROM black_cabs;
Query ID = hadoop_20250625191221_5e695501-ddf9-4e22-b0c0-6aec0ac87e42
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1750873936410_0011)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 5.73 s
----------------------------------------------------------------------------------------------
OK
total_trips_by_black_cabs
72
Time taken: 6.062 seconds, Fetched: 1 row(s)
hive>
```

We got 72 Black Cab trips same as Validation documentation.

Task 9: Calculate the total amount of tips given date wise to all drivers by customers.

Query:

```
WITH daily_tips AS (
   SELECT
           TO_DATE(from_unixtime(CAST(pickup_timestamp / 1000 AS BIGINT))) AS
TRIP_DATE,
      SUM(tip_amount) AS SUM_TIPS
   FROM
      bookings_data
   GROUP BY
      TO_DATE(from_unixtime(CAST(pickup_timestamp / 1000 AS BIGINT)))
)

SELECT
   TRIP_DATE,
   ROUND(SUM_TIPS, 0) AS TOTAL_TIP_AMOUNT
FROM
   daily_tips
ORDER BY
   TRIP_DATE;
```

- The bookings_data table's total daily tip amounts are determined using this SQL query.
- In order to transform each trip's timestamp from Unix time (in milliseconds) to a readable date format (TRIP_DATE), it first creates a Common Table Expression (CTE) named daily_tips. After that, the query groups the records by TRIP_DATE and adds up the tip amounts for each day.
- The query then rounds the total tips to the closest whole number for easier reading after choosing the travel dates and associated total tip amounts from the daily_tips CTE. Next, TRIP_DATE is used to arrange the results chronologically.
- This query offers insights into tipping trends over time by summarising the total tips collected by day.

```
hadoop@ip-10-0-2-56:~
Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j2.properties Async: fals
hive> SET hive.cli.print.header=true;
hive> WITH daily_tips AS (
    >     SELECT
    >         TO_DATE(from_unixtime(CAST(pickup_timestamp / 1000 AS BIGINT))) AS TRIP_DATE,
    >         SUM(tip_amount) AS SUM_TIPS
    >     FROM
    >         bookings_data
    >     GROUP BY
    >         TO_DATE(from_unixtime(CAST(pickup_timestamp / 1000 AS BIGINT)))
    > )
    >
    > SELECT
    >     TRIP_DATE,
    >     ROUND(SUM_TIPS, 0) AS TOTAL_TIP_AMOUNT
    > FROM
    >     daily_tips
    > ORDER BY
    >     TRIP_DATE;
Query ID = hadoop_20250625192402_f6aecefa-669e-49be-b31d-a89221b8270b
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1750873936410_0017)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      2          2        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 5.85 s
----------------------------------------------------------------------------------------------
OK
trip_date       total_tip_amount
2020-01-01      59
2020-01-02      95
2020-01-03      11
2020-01-04      123
2020-01-05      134
2020-01-06      189
2020-01-07      148
2020-01-08      111
```

```
hadoop@ip-10-0-2-56:~
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 5.85 s
----------------------------------------------------------------------------------------------
OK
trip_date       total_tip_amount
2020-01-01      59
2020-01-02      95
2020-01-03      11
2020-01-04      123
2020-01-05      134
2020-01-06      189
2020-01-07      148
2020-01-08      111
2020-01-09      48
2020-01-10      77
2020-01-11      81
2020-01-12      109
2020-01-14      142
2020-01-15      338
2020-01-16      155
2020-01-17      296
2020-01-18      240
2020-01-20      210
2020-01-21      5
2020-01-23      148
2020-01-24      472
2020-01-25      98
2020-01-26      209
2020-01-27      231
2020-01-28      567
2020-01-29      123
2020-01-30      112
2020-01-31      256
2020-02-01      317
2020-02-02      338
2020-02-03      191
2020-02-04      258
2020-02-05      212
2020-02-06      154
2020-02-07      91
2020-02-08      270
2020-02-09      266
2020-02-10      115
```

The following Output matches the validation documentation.

Task 10: Calculate the total count of all the bookings with ratings lower than 2 as given by customers in a particular month.

Query:

```
 WITH low_rated_trips AS (
   SELECT
      date_format(from_unixtime(CAST(pickup_timestamp / 1000 AS BIGINT)), 'yyyy-MM')
AS TRIP_MONTH,
      booking_id
   FROM
      bookings_data
   WHERE
      rating_by_customer < 2
)

SELECT
   TRIP_MONTH,
   COUNT(booking_id) AS NO_OF_BOOKINGS
FROM
   low_rated_trips
GROUP BY
   TRIP_MONTH
ORDER BY
   TRIP_MONTH;
```

- This SQL query is designed to analyze low-rated trips on a monthly basis from the bookings_data table.
- It starts by creating a Common Table Expression (CTE) named low_rated_trips, which filters trips where the rating_by_customer is less than 2, indicating poorly rated rides. For each of these trips, it extracts the trip month by converting the Unix timestamp (in

milliseconds) to a readable yyyy-MM format and selects the corresponding booking_id.

- In the main query, it groups these low-rated trips by TRIP_MONTH and counts the number of bookings for each month, providing the total number of low-rated trips per month. The results are ordered chronologically by month.
- This query helps track monthly trends in customer dissatisfaction, which can be valuable for identifying periods of poor service and potential areas for operational improvement.

```
hadoop@ip-10-0-2-56:~
Time taken: 167.521 seconds, Fetched: 289 row(s)
hive> WITH low_rated_trips AS (
    >     SELECT
    >         date_format(from_unixtime(CAST(pickup_timestamp / 1000 AS BIGINT)), 'yyyy-MM') AS TRIP_MONTH,
    >         booking_id
    >     FROM
    >         bookings_data
    >     WHERE
    >         rating_by_customer < 2
    > )
    >
    > SELECT
    >     TRIP_MONTH,
    >     COUNT(booking_id) AS NO_OF_BOOKINGS
    > FROM
    >     low_rated_trips
    > GROUP BY
    >     TRIP_MONTH
    > ORDER BY
    >     TRIP_MONTH;
Query ID = hadoop_20250625192812_2cab59ea-0c20-4be3-b37f-8c101440fba9
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1750873936410_0017)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      2          2        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 6.82 s
----------------------------------------------------------------------------------------------
OK
trip_month      no_of_bookings
2020-01 26
2020-02 16
2020-03 16
2020-04 21
2020-05 21
2020-06 14
2020-07 20
```

```
Query ID = hadoop_20250625192812_2cab59ea-0c20-4be3-b37f-8c101440fba9
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1750873936410_0017)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      2          2        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 6.82 s
----------------------------------------------------------------------------------------------
OK
trip_month      no_of_bookings
2020-01 26
2020-02 16
2020-03 16
2020-04 21
2020-05 21
2020-06 14
2020-07 20
2020-08 32
2020-09 21
2020-10 15
Time taken: 7.427 seconds, Fetched: 10 row(s)
hive>
```

The above output matches the Validation document.

Task 11: Calculate the count of total iOS users.

Query:

```
WITH ios_users AS (
    SELECT DISTINCT customer_id
    FROM clickstream_data
    WHERE OS_VERSION = 'iOS'
)

SELECT
    COUNT(*) AS TOTAL_IOS_USERS
FROM
    ios_users;
```

- The clickstream_data dataset is used in this SQL query to determine the total number of distinct iOS users.
- A Common Table Expression (CTE) named ios_users is first created, and it picks all unique customer_id values where the OS_VERSION is 'iOS'. Even if an iOS user appears in the dataset more than once, this step guarantees that they are only counted once.
- In the last step, the query returns TOTAL_IOS_USERS, which is just the total number of distinct iOS users found in the CTE. For platform-specific user analysis or marketing campaigns aimed at iOS consumers, this query offers a clear and precise count of individual iOS users.

In the Output below we got 2 counts extra than the output from validation document. Looking more into this we found that since we got a little more of data ingested we got those 2 counts extra.

```
0
Time taken: 14.395 seconds, Fetched: 1 row(s)
hive> WITH ios_users AS (
    >     SELECT DISTINCT customer_id
    >     FROM clickstream_data
    >     WHERE OS_VERSION = 'iOS'
    > )
    >
    > SELECT
    >     COUNT(*) AS TOTAL_IOS_USERS
    > FROM
    >     ios_users;
Query ID = hadoop_20250625193101_e38c79bb-9b7c-457a-82d7-287b650d2861
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1750873936410_0017)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      6          6        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      2          2        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 14.46 s
----------------------------------------------------------------------------------------------
OK
total_ios_users
1515
Time taken: 14.92 seconds, Fetched: 1 row(s)
hive>
```

All the tasks from 1 to 11 are now successfully performed.