

Logic For First Submission

<Properly explain the code, list the steps to run the code provided by you and attach screenshots of code execution>

<Problem Statement>

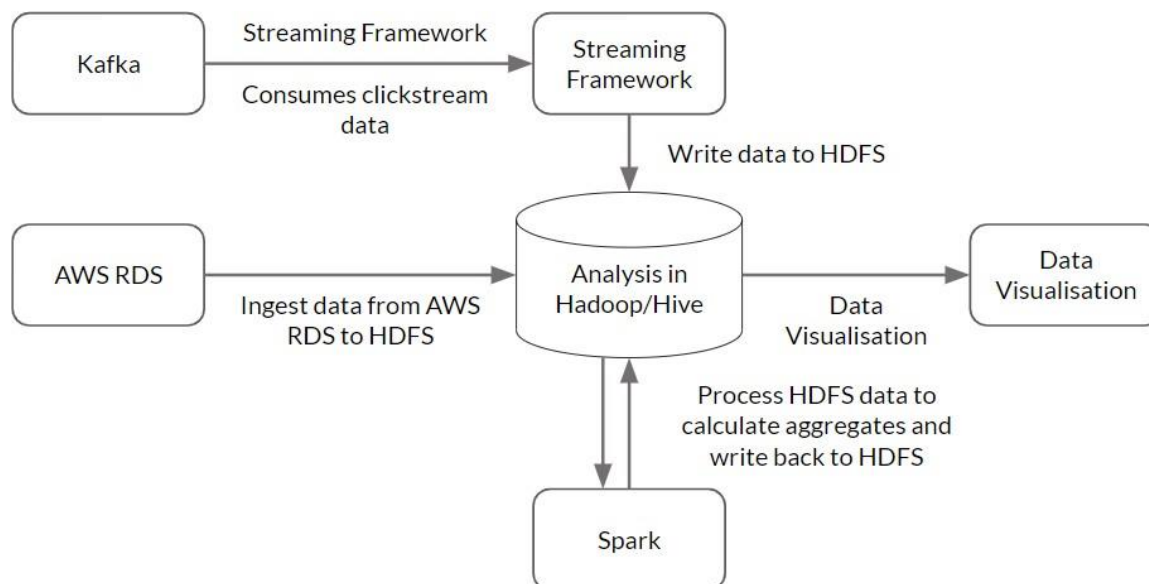
Suppose you are working at a mobility start-up company called 'YourOwnCabs' (YOC). Primarily, it is an online on-demand cab booking service. When you joined the company, it was doing around 100 rides per day. Owing to a successful business model and excellent service, the company's business is growing rapidly, and these numbers are breaking their own records every day. YOC's customer base and ride counts are increasing on a day-by-day basis.

It is highly important for business stakeholders to derive quick and on-demand insights regarding the numbers to decide the company's future strategy. Owing to the massive growth in business, it is getting tough for the company's management to obtain the business numbers frequently, as backend MySQL is not capable of catering to all types of queries owing to large volume data. The following statistics will help you understand the gravity of the situation:

Building a solution to cater to the following requirements:

- **Booking data analytics solution:** This is a feature in which the ride-booking data is stored in a way such that it is available for all types of analytics without hampering business. These analyses mostly include daily, weekly and monthly booking counts as well as booking counts by the mobile operating system, average booking amount, total tip amount, etc.
- **Clickstream data analytics solution:** Clickstream is the application browsing data that is generated on every action taken by the user on the app. This includes link click, button click, screen load, etc. This is relevant to know the user's intent and then take action to engage them more in the app according to their browsing behavior. Since this is very high-volume data (more than the bookings data), it needs to be architecture quite well and stored in a proper format for analysis.

Architecture:



Flow of data:

The two types of data are the clickstream data and the batch data. The clickstream data is captured in Kafka. A streaming framework should consume the data from Kafka and load the same to Hadoop. Once the clickstream data enters the Stream processing layer, it is synced to the HDFS directory to process further. For the batch data, which is the bookings data, the data is stored in the RDS and needs to be ingested to Hadoop.

Also, in cases wherein aggregates need to be prepared, data is read from HDFS, processed by a processing framework such as Spark and written back to HDFS to create a Hive table for the aggregated data. Once both the data are loaded in HDFS, this data is loaded into Hive tables to make it query-able. Hive tables serve as final consumption tables for end-user querying and are eventually consistent.

Approach:

1. Write a job to consume clickstream data from Kafka and ingest to Hadoop.
2. Write a script to ingest the relevant bookings data from AWS RDS to Hadoop.
3. Create aggregates for finding date-wise total bookings using the Spark script.
4. Create a Hive-managed table for clickstream data.
5. Create a Hive-managed table for bookings data.
6. Create a Hive-managed table for aggregated data in Step 3.
7. Load the data in the Hive tables created

Step 1:

1. Connect to the ec2 instance
2. Switch the user from ec2 to root using. (**sudo -i**)
3. Create a python file with code that can ingest real-time clickstream data from Kafka Server and save it to the local directory. (vi **spark_kafka_to_local.py**)
4. Run the command - **export SPARK_KAFKA_VERSION=0.10**
5. Submit the spark job using the python file with spark jar file (**spark2-submit --jars spark-sql-kafka-0-10_2.11-2.3.0.jar spark_kafka_to_local.py**)
6. Create another python file for cleaning the Kafka loaded data and structuring the data and saving it in csv file format. (vi **spark_local_flatten.py**)
7. Spark submit the python file with spark jar file (**spark2-submit --jars spark-sql-kafka-0-10_2.11-2.3.0.jar spark_local_flatten.py**)

<Steps to load the data into Hadoop>

In the above cleaning python file we have specified the default path for the creation of the folder where the Kafka data would be stored in the structured format and screenshot of the code and the folder is provided below.

```
df1.coalesce(1).write.format('com.databricks.spark.csv').mode('overwrite').save('user/root/clickstream_data_flatten', header = 'true')
```

<Screenshot of the data>

```
[root@ip-10-0-0-133 ~]# hadoop fs -ls /user/root/
Found 5 items
drwx----- - root supergroup          0 2022-02-13 13:26 /user/root/.Trash
drwxr-xr-x - root supergroup          0 2022-02-09 11:18 /user/root/.sparkStaging
drwx----- - root supergroup          0 2022-02-10 16:13 /user/root/.staging
drwxr-xr-x - root supergroup          0 2022-02-13 13:21 /user/root/clickstream_checkpoint
drwxr-xr-x - root supergroup          0 2022-02-13 13:21 /user/root/clickstream_data
[root@ip-10-0-0-133 ~]# hadoop fs -ls /user/root/clickstream_data
Found 2 items
drwxr-xr-x - root supergroup          0 2022-02-13 13:21 /user/root/clickstream_data/_spark_metadata
-rw-r--r--  3 root supergroup    1267605 2022-02-13 13:21 /user/root/clickstream_data/part-00000-74523766-d63c-49e5-8045-bd6a78a62d06-c000.json
```

```
root@ip-10-0-0-133:~
[root@ip-10-0-0-133 ~]# hadoop fs -cat /user/root/clickstream_data/part-00000-74523766-d63c-49e5-8045-bd6a78a62d06-c000.json | wc -l
3000
[root@ip-10-0-0-133 ~]#
```

```
Found 2 items
drwxr-xr-x - root supergroup          0 2022-02-13 13:51 /user/root/clickstream_data/_spark_metadata
-rw-r--r--  3 root supergroup    1267605 2022-02-13 13:51 /user/root/clickstream_data/part-00000-62b5e35a-b98c-4d44-ae6a-5774b3549b0e-c000.json
[root@ip-10-0-0-133 ~]# clear
[root@ip-10-0-0-133 ~]# vi spark_local_flatten.py
[root@ip-10-0-0-133 ~]# spark2-submit --jars spark-sql-kafka-0-10_2.11-2.3.0.jar spark_local_flatten.py
22/02/13 14:07:03 INFO spark.SparkContext: Running Spark version 2.3.0.cloudera2
22/02/13 14:07:03 INFO spark.SparkContext: Submitted application: Kafka To HDFS
```

```
22/02/13 14:07:12 INFO scheduler.DAGScheduler: ResultStage 1 (showString at NativeMethodAccessorImpl.java:0) finished in 0.290 s
22/02/13 14:07:12 INFO scheduler.DAGScheduler: Job 1 finished: showString at NativeMethodAccessorImpl.java:0, took 0.301916 s
```

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
26564820	3.2.35	Android	16.4454865	99.902065	de545711-3914-445... fcbaf68aa-1231-11e...		No	Yes	No	Yes	null
31906387	2.4.7	iOS	-64.813749	-133.527040	de545711-3914-445... a95dd57b-779f-49d...		No	No	Yes	Yes	null
25713677	3.4.12	Android	89.943435	127.313415	b328829e-17ae-11e... fcbaf68aa-1231-11e...		No	No	Yes	No	null
83474293	3.1.8	Android	-69.939070	-36.451670	e7bc5fb2-1231-11e... ele99492-17ae-11e...		Yes	No	Yes	No	null
63727807	2.2.9	iOS	64.082108	-81.822078	e7bc5fb2-1231-11e... fcbaf68aa-1231-11e...		No	Yes	Yes	Yes	null
73737907	4.3.19	Android	-18.850508	-116.358375	b328829e-17ae-11e... ele99492-17ae-11e...		No	Yes	No	Yes	null
36927433	3.2.26	iOS	-84.6857245	-146.507678	de545711-3914-445... a95dd57b-779f-49d...		Yes	Yes	No	Yes	null
12691783	3.3.11	Android	54.3852925	-37.411814	de545711-3914-445... ele99492-17ae-11e...		Yes	Yes	No	No	null
22635021	4.4.36	iOS	-31.805500	150.655650	e7bc5fb2-1231-11e... a95dd57b-779f-49d...		No	No	No	No	null
23593546	1.2.16	Android	8.8918475	-83.929878	de545711-3914-445... ele99492-17ae-11e...		Yes	No	Yes	No	null

only showing top 10 rows

```
[root@ip-10-0-0-133 ~]# hadoop fs -ls /user/root/user/root/clickstream_data_flatten/
Found 2 items
-rw-r--r-- 3 root supergroup 0 2022-02-13 14:07 /user/root/user/root/clickstream_data_flatten/ SUCCESS
-rw-r--r-- 3 root supergroup 397733 2022-02-13 14:07 /user/root/user/root/clickstream_data_flatten/part-00000-8371bd57-dd00-4a55-8933-05ad5732a984-c000.csv
[root@ip-10-0-0-133 ~]# hadoop fs -cat /user/root/user/root/clickstream_data_flatten/part-00000-8371bd57-dd00-4a55-8933-05ad5732a984-c000.csv | wc -l
wc: invalid option -- 'l'
Try 'wc --help' for more information.
cat: Unable to write to output stream.
[root@ip-10-0-0-133 ~]# hadoop fs -cat /user/root/user/root/clickstream_data_flatten/part-00000-8371bd57-dd00-4a55-8933-05ad5732a984-c000.csv | wc -l
3001
```

Step 2:

<Command to import data from AWS RDS to Hadoop>

```
sqoop import \
--connect jdbc:mysql://upgradtest.cyaieic9bmnf.us-east-1.rds.amazonaws.com/testdatabase \
--table bookings \
--username student
--password STUDENT123 \
--target-dir /user/root/bookings_data \
-m1
```

<Command to view the imported data>

```
hadoop fs -ls /user/root/bookings_data
hadoop fs -cat /user/root/bookings_data/part-m-00000 | wc -l
```

<Screenshot of the data>

```
[root@ip-10-0-0-133 ~]# clear
[root@ip-10-0-0-133 ~]# sqoop import \
> --connect jdbc:mysql://upgradtest.cyaic9bmnf.us-east-1.rds.amazonaws.com/testdatabase \
> --table bookings \
> --username student --password STUDENT123 \
> --target-dir /user/root/bookings_data \
> -m1 \
>
Warning: /opt/cloudera/parcels/CDH-5.15.1-1.cdh5.15.1.p0.4/bin/../lib/sqoop/./accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
22/02/13 14:18:20 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.15.1
22/02/13 14:18:20 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
22/02/13 14:18:21 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
22/02/13 14:18:21 INFO tool.CodeGenTool: Beginning code generation
22/02/13 14:18:21 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM 'bookings' AS t LIMIT 1
22/02/13 14:18:21 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM 'bookings' AS t LIMIT 1
22/02/13 14:18:21 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /opt/cloudera/parcels/CDH/lib/hadoop-mapreduce
Note: /tmp/sqoop-root/compile/5bbaab85996c9daae14b22df95fa2568/bookings.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
22/02/13 14:18:21 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-root/compile/5bbaab85996c9daae14b22df95fa2568/bookings.jar
Dyces written: 10000
22/02/13 14:18:49 INFO mapreduce.ImportJobBase: Transferred 161.7949 KB in 24.0578 seconds (6.7253 KB/sec)
22/02/13 14:18:49 INFO mapreduce.ImportJobBase: Retrieved 1000 records.
[root@ip-10-0-0-133 ~]# hadoop fs -ls /user/root/bookings_data
Found 2 items
-rw-r--r-- 3 root supergroup 0 2022-02-13 14:18 /user/root/bookings_data/_SUCCESS
-rw-r--r-- 3 root supergroup 165678 2022-02-13 14:18 /user/root/bookings_data/part-m-00000
[root@ip-10-0-0-133 ~]# hadoop fs -cat /user/root/bookings_data/part-m-00000 | wc -l
1000
```

Step 3:

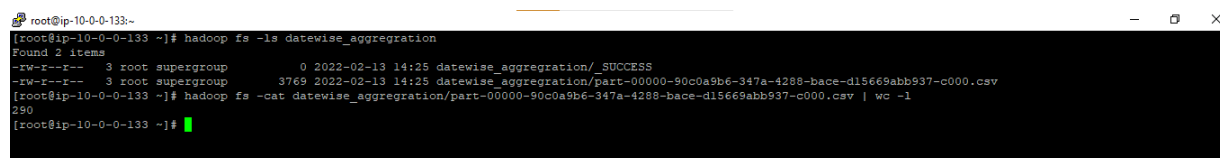
<Command to run the python file>

1. Connect to the ec2 instance
2. Switch the user from ec2 to root using. (**sudo -i**)
3. Create a python file with code that can ingest real-time clickstream data from Kafka Server and save it to the local directory. (**vi datewise_bookings_aggregates_spark.py**)
4. Submit the spark job using the python file with spark jar file (**spark2-submit --jars spark-sql-kafka-0-10_2.11-2.3.0.jar datewise_bookings_aggregates_spark.py**)

<Command to move the csv file to HDFS>

We've specified the path (**/user/root/datewise_aggregation**) for direct saving of the formatted csv and easy retrieval process.

<Screenshot of the file in HDFS>



```
root@ip-10-0-0-133-~
[root@ip-10-0-0-133 ~]# hadoop fs -ls datewise_aggregation
Found 2 items
-rw-r--r-- 3 root supergroup 0 2022-02-13 14:25 datewise_aggregation/_SUCCESS
-rw-r--r-- 3 root supergroup 3769 2022-02-13 14:25 datewise_aggregation/part-00000-90c0a9b6-347a-4288-bace-d15669abb937-c000.csv
[root@ip-10-0-0-133 ~]# hadoop fs -cat datewise_aggregation/part-00000-90c0a9b6-347a-4288-bace-d15669abb937-c000.csv | wc -l
290
[root@ip-10-0-0-133 ~]#
```

Step 4 - 7:

<Command to create the Hive tables>

Managed table for clickstream_data

1. create database if not exists cab_bookings ;
2. use cab_bookings ;
3. create table if not exists clickstream_data (customer_id int ,app_version string, os_version string,lat string ,lon string ,page_id string,button_id string , is_button_click varchar(3) ,is_page_view varchar(3) ,is_scroll_up varchar(3) ,is_scroll_down varchar(3)) row format delimited fields terminated by "," ;

Managed table for bookings data

4. create table if not exists booking_data (booking_id string ,customer_id int ,driver_id int , customer_app_version string, customer_phone_os_version string , pickup_lat double , pickup_lon double, drop_lat double, drop_lon double, pickup_timestamp timestamp , drop_timestamp timestamp ,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) row format delimited fields terminated by "," ;

Managed table for aggregated

5. create table if not exists datewise_data (date string , count int) row format delimited fields terminated by "," ;

<Command to load the data into Hive tables>

6. load data inpath '/user/root/user/root/clickstream_data_flatten/part-00000-8371bd57-dd00-4a55-8933-05ad5732a984-c000.csv' into table clickstream_data ;
7. load data inpath '/user/root/bookings_data_csv/part-00000-a38e960c-5cd8-4c9c-a134-62b4bbb3a6ac-c000.csv' into table booking_data ;
8. load data inpath '/user/root/datewise_aggregation/part-00000-90c0a9b6-347a-4288-bace-d15669abb937-c000.csv' into table datewise_data ;

<screenshot from HIVE>

```
hive> create database if not exists cab_bookings ;
OK
Time taken: 0.058 seconds
hive> show databases;
OK
cab_bookings
default
Time taken: 0.012 seconds, Fetched: 2 row(s)
hive>
```

```
hive> show tables;
OK
booking_data
clickstream_data
datewise_data
Time taken: 0.018 seconds, Fetched: 3 row(s)
hive>
```

```
hive> load data inpath '/user/root/user/root/clickstream_data_flatten/part-00000-8371bd57-dd00-4a55-8933-05ad5732a984-c000.csv' into table clickstream_data ;
Loading data to table cab_bookings.clickstream_data
Table cab_bookings.clickstream_data stats: [numFiles=1, totalSize=397733]
OK
Time taken: 0.469 seconds
```

```
hive> load data inpath '/user/root/bookings_data_csv/part-00000-a38e960c-5cd8-4c9c-a134-62b4bbb3a6ac-c000.csv' into table booking_data ;
Loading data to table cab_bookings.booking_data
Table cab_bookings.booking_data stats: [numFiles=1, totalSize=182961]
OK
Time taken: 0.262 seconds
hive> load data inpath '/user/root/datewise_aggregation/part-00000-90c0a9b6-347a-4288-bace-d15669abb937-c000.csv' into table datewise_data ;
Loading data to table cab_bookings.datewise_data
Table cab_bookings.datewise_data stats: [numFiles=1, totalSize=3769]
OK
Time taken: 0.28 seconds
```

Logic For Final Submission

<Explain the queries, list them and attach screenshots after successful execution of queries>

Queries

<Hive Query for Task 5>

In this query we are calculating the total number of different drivers for each customer.

```
select customer_id ,count( DISTINCT driver_id) as driver_count from booking_data group by customer_id order by customer_id asc;
```

<Screenshot after executing Query>



The screenshot shows a Hive query execution interface. The query is: `select customer_id ,count(DISTINCT driver_id) as driver_count from booking_data group by customer_id order by customer_id asc;`. The results are displayed in a table with two columns: `customer_id` and `driver_count`. The results show 7 rows, each with a unique customer_id and a driver_count of 1.

customer_id	driver_count
10022393	1
10058402	1
10339567	1
10435129	1
10555335	1
10592274	1
10614890	1

<Hive Query for Task 6>

In this query we are calculating the total rides taken by each customer.

```
select customer_id ,count( DISTINCT booking_id) as total_rides from booking_data group by customer_id order by customer_id asc;
```

<Screenshot after executing Query>


```

4 select customer_id ,count( DISTINCT booking_id) as total_rides from booking_data group by customer_id
5 order by customer_id asc;
6

```

customer_id	total_rides
1 10022393	1
2 10058402	1
3 10339567	1
4 10435129	1
5 10555335	1
6 10592274	1
7 10614890	1
8 10678994	1
9 11264797	1
10 11353346	1
11 11418437	1

<Hive Query for Task 7>

In this query we are finding the total visits made by each customer on the booking page and the total 'Book Now' button presses which is further used to calculate the conversion ratio of customers. Conversion ratio can be calculated as Total 'Book Now' Button Press/Total Visits made by customer on the booking page.

```

select count(b.button_id)/count(a.booking_id) from booking_data a full outer join
clickstream_data b on a.customer_id = b.customer_id;

```

<Screenshot after executing Query>

```

2 select customer_id ,count( DISTINCT booking_id) as total_rides from booking_data group by customer_id
3 order by customer_id asc;
4 select customer_id ,count( DISTINCT booking_id) as total_rides from booking_data group by customer_id
5 order by customer_id asc;
6
7 select count(b.button_id)/count(a.booking_id) from booking_data a full outer join
8 clickstream_data b on a.customer_id = b.customer_id;
9

```

_c0
1 3.001

<Hive Query for Task 8>

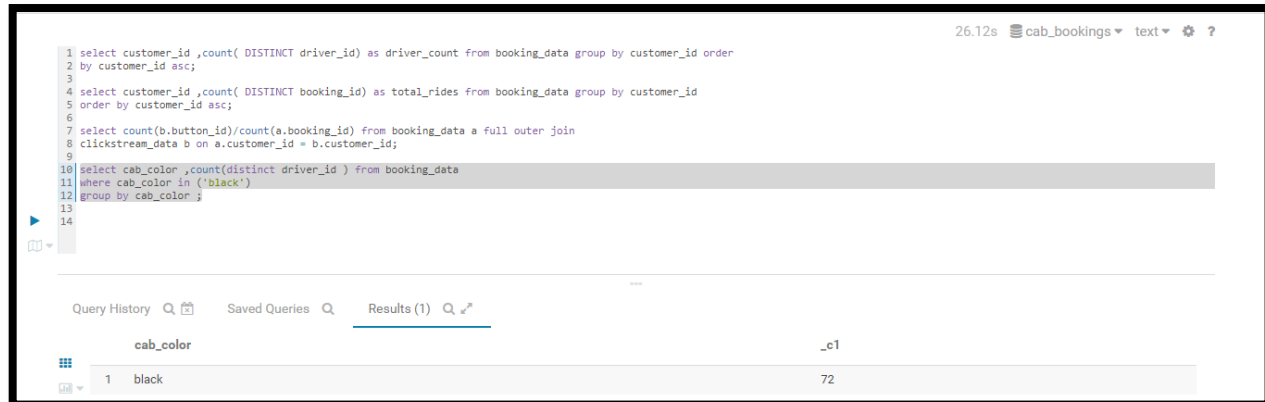
In this query we are calculating the count of all trips done on black cabs.

```

select cab_color ,count(distinct driver_id ) from booking_data
where cab_color in ('black')
group by cab_color ;

```

<Screenshot after executing Query>



```

1 select customer_id ,count( DISTINCT driver_id) as driver_count from booking_data group by customer_id order
2 by customer_id asc;
3
4 select customer_id ,count( DISTINCT booking_id) as total_rides from booking_data group by customer_id
5 order by customer_id asc;
6
7 select count(b.button_id)/count(a.booking_id) from booking_data a full outer join
8 clickstream_data b on a.customer_id = b.customer_id;
9
10 select cab_color ,count(distinct driver_id ) from booking_data
11 where cab_color in ('black')
12 group by cab_color ;
13
14

```

Query History | Saved Queries | Results (1)

	cab_color	_c1
1	black	72

<Hive Query for Task 9>

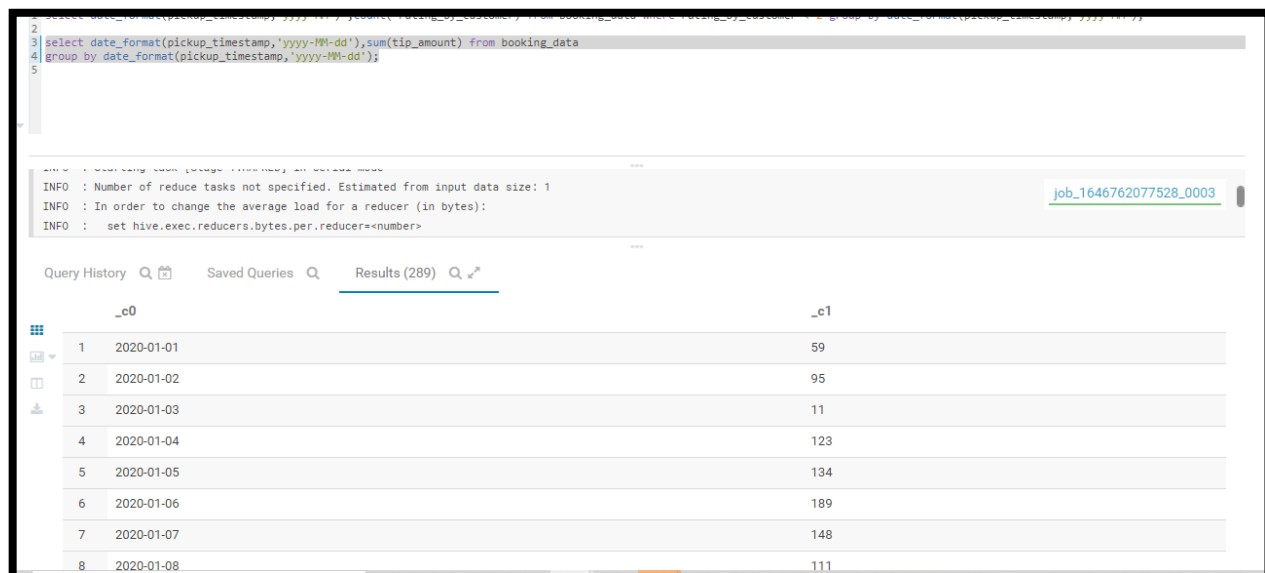
In this query we are calculating the total amount of tips given date wise to all drivers by customers.

```

select date_format(pickup_timestamp,'yyyy-MM-dd'),sum(tip_amount) from booking_data
group by date_format(pickup_timestamp,'yyyy-MM-dd');

```

<Screenshot after executing Query>



```

1 select date_format(pickup_timestamp,'yyyy-MM-dd'),sum(tip_amount) from booking_data
2 group by date_format(pickup_timestamp,'yyyy-MM-dd');
3
4
5

```

INFO : Number of reduce tasks not specified. Estimated from input data size: 1
 INFO : In order to change the average load for a reducer (in bytes):
 INFO : set hive.exec.reducers.bytes.per.reducer=<number>

Query History | Saved Queries | Results (289)

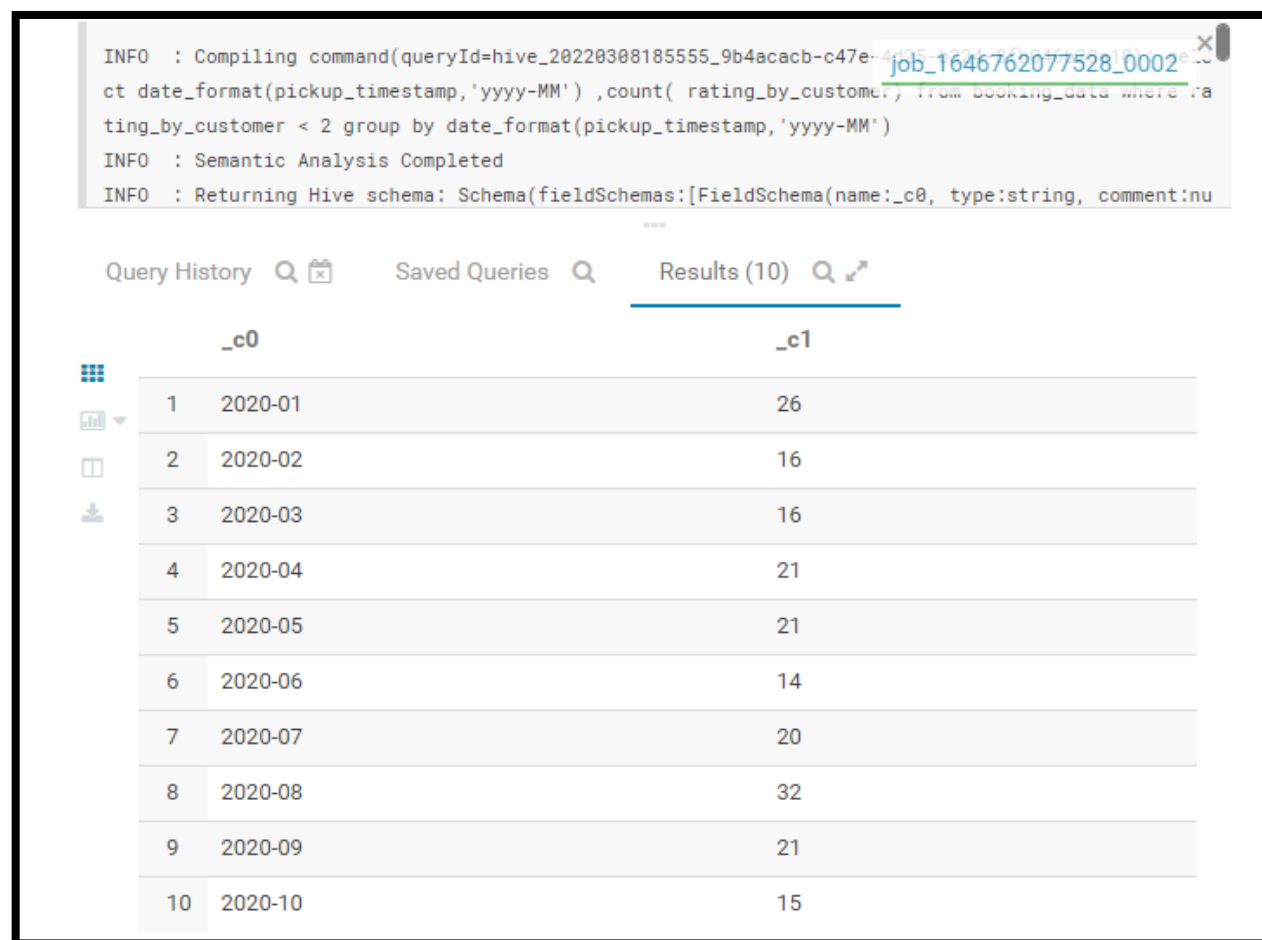
	_c0	_c1
1	2020-01-01	59
2	2020-01-02	95
3	2020-01-03	11
4	2020-01-04	123
5	2020-01-05	134
6	2020-01-06	189
7	2020-01-07	148
8	2020-01-08	111

<Hive Query for Task 10>

In this query we are calculating the total count of all the bookings with ratings lower than 2 as given by customers in a particular month.

```
select date_format(pickup_timestamp,'yyyy-MM') ,count( rating_by_customer) from booking_data
where rating_by_customer < 2 group by date_format(pickup_timestamp,'yyyy-MM') ;
```

<Screenshot after executing Query>



The screenshot shows a Hive query execution interface. At the top, there is a text area containing the query: `select date_format(pickup_timestamp,'yyyy-MM') ,count(rating_by_customer) from booking_data where rating_by_customer < 2 group by date_format(pickup_timestamp,'yyyy-MM')`. Below the query, there are logs indicating the query was compiled, semantic analysis was completed, and the Hive schema was returned. The results are displayed in a table with two columns: `_c0` (date) and `_c1` (count).

	_c0	_c1
1	2020-01	26
2	2020-02	16
3	2020-03	16
4	2020-04	21
5	2020-05	21
6	2020-06	14
7	2020-07	20
8	2020-08	32
9	2020-09	21
10	2020-10	15

<Hive Query for Task 11>

In this query we are calculating the count of total iOS users.

```
select os_version ,count(distinct customer_id) from clickstream_data
where os_version in ('iOS')
group by os_version;
```

<Screenshot after executing Query>

27.12s
cab_bookings
text
?

```

1 select customer_id ,count( DISTINCT driver_id) as driver_count from booking_data group by customer_id order
2 by customer_id asc;
3
4 select customer_id ,count( DISTINCT booking_id) as total_rides from booking_data group by customer_id
5 order by customer_id asc;
6
7 select count(b.button_id)/count(a.booking_id) from booking_data a full outer join
8 clickstream_data b on a.customer_id = b.customer_id;
9
10 select cab_color ,count(distinct driver_id ) from booking_data
11 where cab_color in ('black')
12 group by cab_color ;
13
14 select os_version ,count(distinct customer_id) from clickstream_data
15 where os_version in ('IOS')
16 group by os_version;
17
18

```

Query History
Saved Queries
Results (1)

os_version		_c1
1	IOS	1515