HIVE

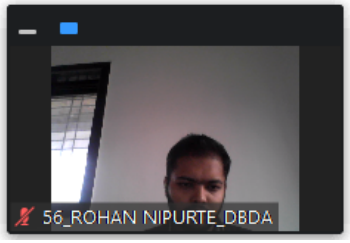
Please find the customer data set

1) **Write a program to find the count of customers for each profession**

```
use training_432538;
```

```
create table customer2(cust_id int, firstname string,lastname
string,age int, profession string)
   >
   > row format delimited
   >
   > fields terminated by ','
   >
   > stored as textfile;
```

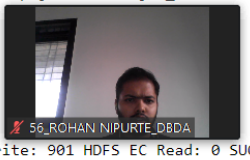
```
hive> create table customer1(cust_id int, firstname string,age int, profession string)
    >
    > row format delimited
    >
    > fields terminated by ','
    >
    > stored as textfile;
OK
Time taken: 0.416 seconds
hive>
```




```
load data local inpath 'custs1.txt' overwrite into table
customer2;
```

```
select count(cust_id) from customer2 group by profession;
```

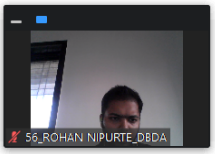
```
hive> select count(cust_id) from customer2 group by profession;
Query ID = bigcdac432538_20221214093059_c9b57260-be7c-4683-98cb-2fe1142592d8
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
22/12/14 09:31:01 INFO client.RMProxy: Connecting to ResourceManager at ip-10-1-1-204.ap-south-1.compute.internal/10.1.1.204:8032
22/12/14 09:31:01 INFO client.RMProxy: Connecting to ResourceManager at ip-10-1-1-204.ap-south-1.compute.internal/10.1.1.204:8032
Starting Job = job_1663041244711_22713, Tracking URL = http://ip-10-1-1-204.ap-south-1.compute.internal:6066/proxy/application_16630
Kill Command = /opt/cloudera/parcels/CDH-6.2.1-1.cdh6.2.1.p0.1425774/lib/hadoop/bin/hadoop job  -kill job_1663041244711_22713
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-12-14 09:32:35,167 Stage-1 map = 0%,  reduce = 0%
2022-12-14 09:32:50,535 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.09 sec
2022-12-14 09:33:22,045 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 9.55 sec
MapReduce Total cumulative CPU time: 9 seconds 550 msec
Ended Job = job_1663041244711_22713
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 9.55 sec   HDFS Read: 400711 HDFS Write: 901 HDFS EC Read: 0 SUCCESS
Total MapReduce CPU Time Spent: 9 seconds 550 msec
OK
199
202
195
203
175
196
193
181
209
```

2) Write a program to find the top 10 products sales wise.

```
create table transaction1(txn_id int,txn_date string,cust_id
int,amount double, category string, product string,city
string,state string,spendb
y string)
    >
    > row format delimited
    >
    > fields terminated by ','
    >
    > stored as textfile;
```

```
hive> create table transaction1(txn_id int,txn_date string,cust_id int,amount double, category string, product string,city string,state string,spendb
y string)
    >
    > row format delimited
    >
    > fields terminated by ','
    >
    > stored as textfile;
OK
Time taken: 0.117 seconds
```




```
load data local inpath 'txns1.txt' overwrite into table
transaction1;
```

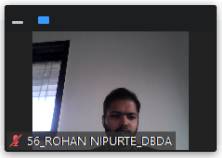
select product,sum(spendby) as tot from transaction1 group by product order by tot desc limit 10;

```
22/12/14 09:50:35 INFO client.RMProxy: Connecting to ResourceManager at ip-10-1-1-204.ap-south-1.compute.internal/10.1.1.2
Starting Job = job_1663041244711_22826, Tracking URL = http://ip-10-1-1-204.ap-south-1.compute.internal:6066/proxy/applica
Kill Command = /opt/cloudera/parcels/CDH-6.2.1-1.cdh6.2.1.p0.1425774/lib/hadoop/bin/hadoop job  -kill job_1663041244711_22
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2022-12-14 09:50:55,828 Stage-2 map = 0%,  reduce = 0%
2022-12-14 09:51:32,498 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 2.51 sec
2022-12-14 09:52:15,012 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 5.64 sec
MapReduce Total cumulative CPU time: 5 seconds 640 msec
Ended Job = job_1663041244711_22826
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 9.25 sec   HDFS Read: 4426749 HDFS Write: 4865 HDFS EC Read: 0 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 5.64 sec   HDFS Read: 10528 HDFS Write: 374 HDFS EC Read: 0 SUCCESS
Total MapReduce CPU Time Spent: 14 seconds 890 msec
OK
Beach Volleyball        0.0
Basketball      0.0
Yoga & Pilates  0.0
Ballet Bars     0.0
Balance Beams   0.0
Badminton       0.0
Archery 0.0
Air Suits       0.0
Wrestling       0.0
Abdominal Equipment     0.0
Time taken: 275.77 seconds, Fetched: 10 row(s)
```

3) Write a program to create partiioned table on category

```
create table salesPartitioned(txn_id int,txn_date
string,cust_id int,amount double, product string,city
string,state string,spendby string)
    >
    > partitioned by (category string)
    >
    > row format delimited
    >
    > fields terminated by ','
    >
    > stored as textfile;
```

```
hive> create table salesPartitioned(txn_id int,txn_date string,cust_id int,amount double, product string,city string,state string,spendby string)
    >
    > partitioned by (category string)
    >
    > row format delimited
    >
    > fields terminated by ','
    >
    > stored as textfile;
OK
Time taken: 0.099 seconds
```



SPARK

1) What was the highest number of people travelled in which year?

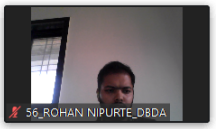
```
from pyspark.sql.types import
StructType,StringType,IntegerType,DoubleType,LongType


schema10=StructType().add("Year",StringType(),True).add("Quart
er",StringType(),True).add("Avg_rev_per_seat",DoubleType(),Tru
e).add("Booked_seats"
,IntegerType(),True)


df_schema10=spark.read.format("csv").option("header","True").s
chema(schema10).load("hdfs://nameservice1/user/bigcdac432538/t
raining/airlines.csv")

df_schema10.registerTempTable("airlines")
```

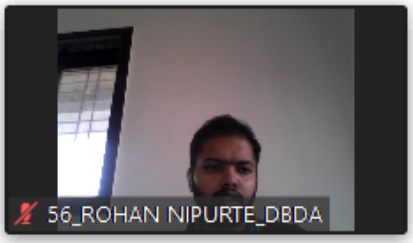
```
>>> df_schema10=spark.read.format("csv").option("header","True").schema(schema10).load("hdfs://nameservice1/user/bigcdac432538/training/airlines.csv"
)
>>> df_schema10.registerTempTable("airlines")
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
```




```
df_schema10.printSchema()
root
|-- Year: string (nullable = true)
|-- Quarter: string (nullable = true)
|-- Avg_rev_per_seat: double (nullable = true)
|-- Booked_seats: integer (nullable = true)

df_schema10.show()
```

```
>>> df_schema10.show()
+----+-------+----------------+------------+
|Year|Quarter|Avg_rev_per_seat|Booked_seats|
+----+-------+----------------+------------+
|1995|      1|           296.9|       46561|
|1995|      2|           296.8|       37443|
|1995|      3|          287.51|       34128|
|1995|      4|          287.78|       30388|
|1996|      1|          283.97|       47808|
|1996|      2|          275.78|       43020|
|1996|      3|          269.49|       38952|
|1996|      4|          278.33|       37443|
|1997|      1|           283.4|       35067|
|1997|      2|          289.44|       46565|
|1997|      3|          282.27|       38886|
|1997|      4|          293.51|       37454|
|1998|      1|          304.74|       31315|
|1998|      2|          300.97|       30852|
|1998|      3|          315.25|       38118|
|1998|      4|          316.18|       35393|
|1999|      1|          331.74|       47453|
|1999|      2|          329.34|       38243|
|1999|      3|          317.22|       33048|
|1999|      4|          317.93|       31256|
+----+-------+----------------+------------+
only showing top 20 rows
```



df_schema11=spark.sql("select year,sum(Avg_rev_per_seat * Booked_seats) as avgRevSeat from airlines group by year order by avgRevSeat desc limit 1")

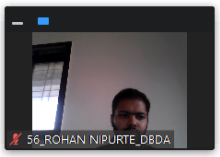
2) Identifying the highest revenue generation for which year

df_schema10=spark.sql("select Year,sum(Avg_rev_per_seat * Booked_seats) as avgRevSeat from airlines group by year order by avgRevSeat desc limit 1")

df_shema10.show()

```
>>> df_schema10=spark.sql("select Year,sum(Avg_rev_per_seat * Booked_seats) as avgRevSeat from airlines group by year order by avgRevSeat desc limit
1")
>>> df_schema10.show()
+----+-------------+
|Year|   avgRevSeat|
+----+-------------+
|2013|6.636320871E7|
+----+-------------+

>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
```

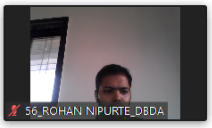


3) Identifying the highest revenue generation for which year and quarter (Common group)

df_schema10=spark.sql("select Year,quarter,sum(Avg_rev_per_seat * Booked_seats) as avgRevSeat from airlines group by year,quarter order by avgRev Seat desc limit 1")

```
>>> df_schema10=spark.sql("select Year,quarter,sum(Avg_rev_per_seat * Booked_seats) as avgRevSeat from airlines group by year,quarter order by avgRev
Seat desc limit 1")
>>> df_schema10.show()
+----+-------+-------------+
|Year|quarter|   avgRevSeat|
+----+-------+-------------+
|2014|      4|1.881940848E7|
+----+-------+-------------+

>>>
>>>
>>>
>>>
>>>
>>>
>>>
```



```
>>> df_schema10.show()
+----+-------+-------------+
|Year|quarter|   avgRevSeat|
+----+-------+-------------+
|2014|      4|1.881940848E7|
+----+-------+-------------+

>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
```

MapReduce