Module 1: Pig Latin

Write a Pig script to load the transaction_data.csv dataset, filter rows where amount is greater than 300, and store the result in a new file.

hadoop fs -put Desktop/transaction_data.csv

$ pig

transactions = LOAD 'transaction_data.csv'   USING PigStorage(',')   AS (transaction_id:chararray, user_id:chararray, amount:float, status:chararray);

filtered_transactions = FILTER transactions BY amount > 300;

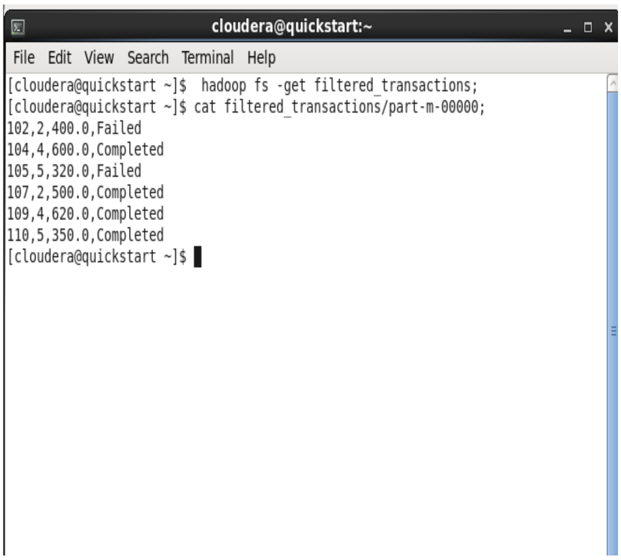
STORE filtered_transactions INTO 'filtered_transactions'

    USING PigStorage(',');

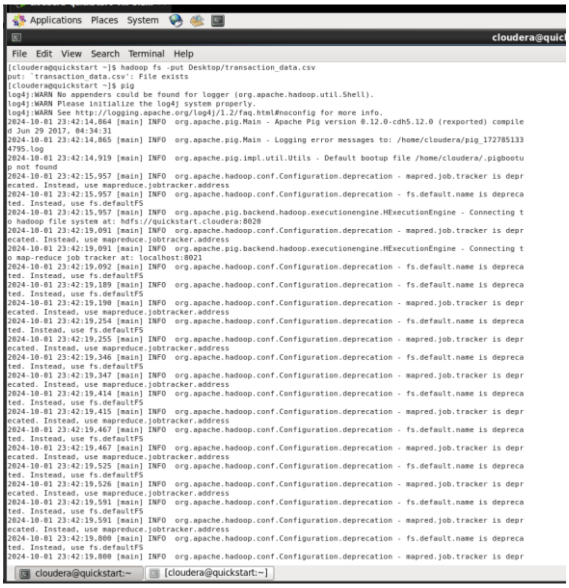
$ hadoop fs -get filtered_transactions

$ cat filtered_transactions/part-m-00000

OUTPUT

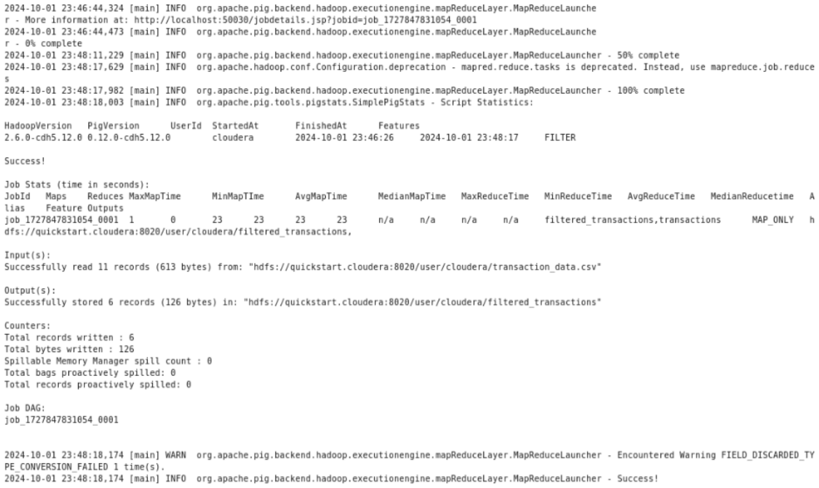


```
[cloudera@quickstart ~]$  hadoop fs -get filtered_transactions;
[cloudera@quickstart ~]$ cat filtered_transactions/part-m-00000;
102,2,400.0,Failed
104,4,600.0,Completed
105,5,320.0,Failed
107,2,500.0,Completed
109,4,620.0,Completed
110,5,350.0,Completed
[cloudera@quickstart ~]$ 
```



Applications Places System
cloudera@quic
File Edit View Search Terminal Help
cloudera@quickstart:~
[cloudera@quickstart:~]

```
[cloudera@quickstart ~]$ hadoop fs -put Desktop/transaction_data.csv
put: `transaction_data.csv': File exists
[cloudera@quickstart ~]$ pig
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
2024-10-01 23:42:14,864 [main] INFO  org.apache.pig.Main - Apache Pig version 0.12.0-cdh5.12.0 (rexported) compile
d Jun 29 2017, 04:34:31
2024-10-01 23:42:14,865 [main] INFO  org.apache.pig.Main - Logging error messages to: /home/cloudera/pig_172785133
4795.log
2024-10-01 23:42:14,919 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file /home/cloudera/.pigbootu
p not found
2024-10-01 23:42:15,957 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is depr
ecated. Instead, use mapreduce.jobtracker.address
2024-10-01 23:42:15,957 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depreca
ted. Instead, use fs.defaultFS
2024-10-01 23:42:15,957 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting t
o hadoop file system at: hdfs://quickstart.cloudera:8020
2024-10-01 23:42:19,091 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is depr
ecated. Instead, use mapreduce.jobtracker.address
2024-10-01 23:42:19,091 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting t
o map-reduce job tracker at: localhost:8021
2024-10-01 23:42:19,092 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depreca
ted. Instead, use fs.defaultFS
2024-10-01 23:42:19,189 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depreca
ted. Instead, use fs.defaultFS
2024-10-01 23:42:19,190 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is depr
ecated. Instead, use mapreduce.jobtracker.address
2024-10-01 23:42:19,254 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depreca
ted. Instead, use fs.defaultFS
2024-10-01 23:42:19,255 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is depr
ecated. Instead, use mapreduce.jobtracker.address
2024-10-01 23:42:19,346 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depreca
ted. Instead, use fs.defaultFS
2024-10-01 23:42:19,347 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is depr
ecated. Instead, use mapreduce.jobtracker.address
2024-10-01 23:42:19,414 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depreca
ted. Instead, use fs.defaultFS
2024-10-01 23:42:19,415 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is depr
ecated. Instead, use mapreduce.jobtracker.address
2024-10-01 23:42:19,467 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depreca
ted. Instead, use fs.defaultFS
2024-10-01 23:42:19,467 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is depr
ecated. Instead, use mapreduce.jobtracker.address
2024-10-01 23:42:19,525 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depreca
ted. Instead, use fs.defaultFS
2024-10-01 23:42:19,526 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is depr
ecated. Instead, use mapreduce.jobtracker.address
2024-10-01 23:42:19,591 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depreca
ted. Instead, use fs.defaultFS
2024-10-01 23:42:19,591 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is depr
ecated. Instead, use mapreduce.jobtracker.address
2024-10-01 23:42:19,800 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depreca
ted. Instead, use fs.defaultFS
2024-10-01 23:42:19,800 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is depr
```



```
2024-10-01 23:46:44,324 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLaunche
r - More information at: http://localhost:50030/jobdetails.jsp?jobid=job_1727847831054_0001
2024-10-01 23:46:44,473 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLaunche
r - 0% complete
2024-10-01 23:48:11,229 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 50% complete
2024-10-01 23:48:17,629 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.reduce.tasks is deprecated. Instead, use mapreduce.job.reduce
s
2024-10-01 23:48:17,982 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 100% complete
2024-10-01 23:48:18,003 [main] INFO  org.apache.pig.tools.pigstats.SimplePigStats - Script Statistics:

HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
2.6.0-cdh5.12.0 0.12.0-cdh5.12.0        cloudera        2024-10-01 23:46:26     2024-10-01 23:48:17     FILTER

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTIme      AvgMapTime      MedianMapTime   MaxReduceTime   MinReduceTime   AvgReduceTime   MedianReducetime   A
lias    Feature Outputs
job_1727847831054_0001  1       0       23      23      23      23      n/a     n/a     n/a     n/a     filtered_transactions,transactions      MAP_ONLY   h
dfs://quickstart.cloudera:8020/user/cloudera/filtered_transactions,

Input(s):
Successfully read 11 records (613 bytes) from: "hdfs://quickstart.cloudera:8020/user/cloudera/transaction_data.csv"

Output(s):
Successfully stored 6 records (126 bytes) in: "hdfs://quickstart.cloudera:8020/user/cloudera/filtered_transactions"

Counters:
Total records written : 6
Total bytes written : 126
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1727847831054_0001


2024-10-01 23:48:18,174 [main] WARN  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Encountered Warning FIELD_DISCARDED_TY
PE_CONVERSION_FAILED 1 time(s).
2024-10-01 23:48:18,174 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
```

2.Use Pig to calculate the average amount from the transaction_data.csv grouped by status. Concatenate user_id and status into a new field in a Pig script.

transactions = LOAD 'transaction_data.csv' USING PigStorage(',')   AS (transaction_id:chararray, user_id:chararray, amount:float, status:chararray);

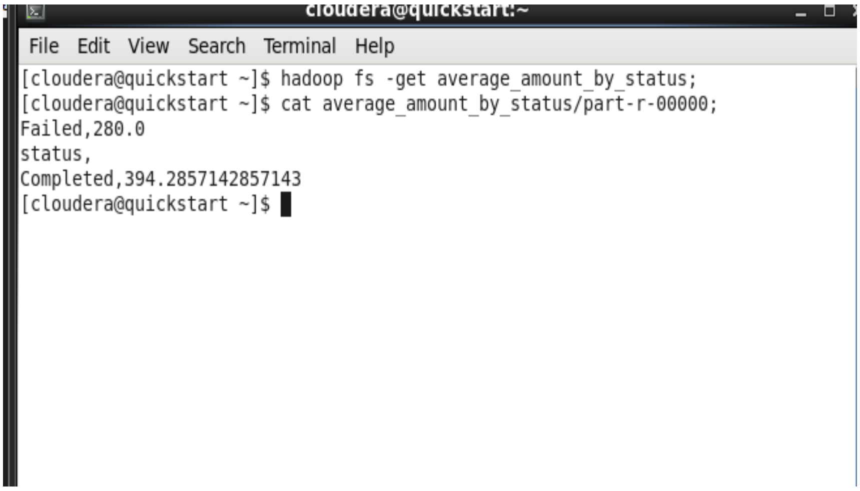
grouped_by_status = GROUP transactions BY status;

average_amount = FOREACH grouped_by_status GENERATE group AS status, AVG(transactions.amount) AS avg_amount;

STORE average_amount INTO 'average_amount_by_status' USING PigStorage(',');

$ hadoop fs -get average_amount_by_status

$ cat average_amount_by_status/part-r-00000



```
[cloudera@quickstart ~]$ hadoop fs -get average_amount_by_status;
[cloudera@quickstart ~]$ cat average_amount_by_status/part-r-00000;
Failed,280.0
status,
Completed,394.2857142857143
[cloudera@quickstart ~]$ 
```

3.Concatenate user_id and status into a new field in a Pig script.

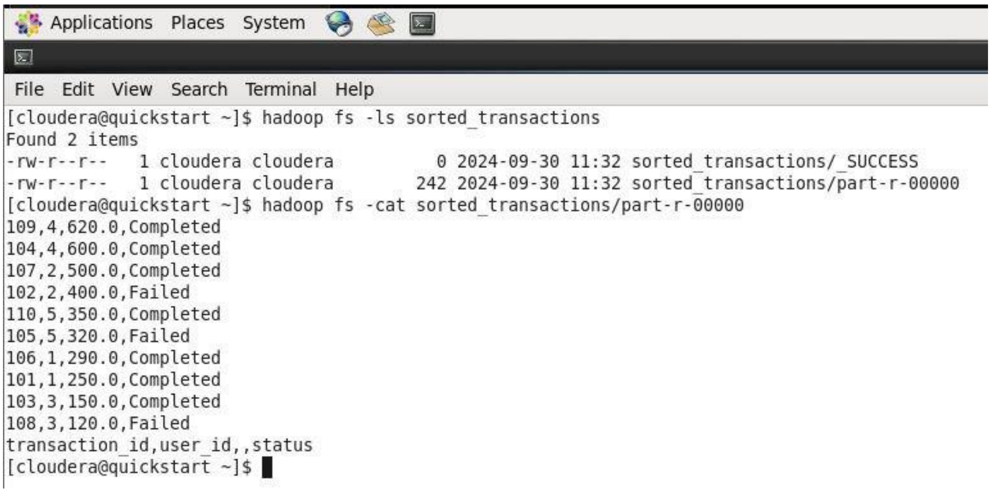
transactions_with_concat = FOREACH transactions GENERATE transaction_id, user_id, amount, status,

    CONCAT(user_id, status) AS user_status_concat;

STORE transactions_with_concat INTO 'transactions_with_concat'   USING PigStorage(',');

$ hadoop fs -get transactions_with_concat

$ cat transactions_with_concat/part-r-00000


Applications Places System
File Edit View Search Terminal Help

```
[cloudera@quickstart ~]$ hadoop fs -ls sorted_transactions
Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2024-09-30 11:32 sorted_transactions/_SUCCESS
-rw-r--r--   1 cloudera cloudera        242 2024-09-30 11:32 sorted_transactions/part-r-00000
[cloudera@quickstart ~]$ hadoop fs -cat sorted_transactions/part-r-00000
109,4,620.0,Completed
104,4,600.0,Completed
107,2,500.0,Completed
102,2,400.0,Failed
110,5,350.0,Completed
105,5,320.0,Failed
106,1,290.0,Completed
101,1,250.0,Completed
103,3,150.0,Completed
108,3,120.0,Failed
transaction_id,user_id,,status
[cloudera@quickstart ~]$ 
```

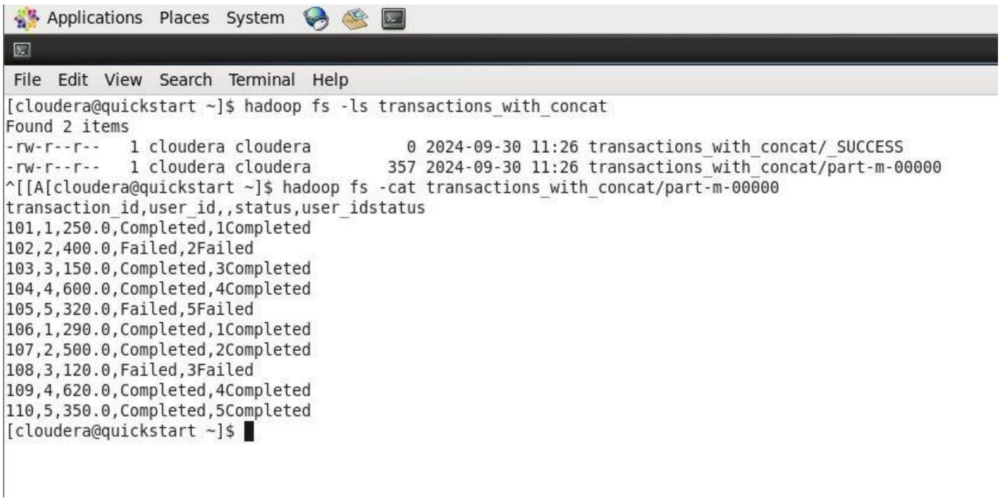
4.Sort the transaction_data.csv dataset by amount in descending order.

sorted_transactions = ORDER transactions BY amount DESC;

STORE sorted_transactions INTO 'sorted_transactions' USING PigStorage(',');

$ hadoop fs -get sorted_transactions

$ cat sorted_transactions/part-r-00000


Applications Places System
File Edit View Search Terminal Help

```
[cloudera@quickstart ~]$ hadoop fs -ls transactions_with_concat
Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2024-09-30 11:26 transactions_with_concat/_SUCCESS
-rw-r--r--   1 cloudera cloudera        357 2024-09-30 11:26 transactions_with_concat/part-m-00000
^[[A[cloudera@quickstart ~]$ hadoop fs -cat transactions_with_concat/part-m-00000
transaction_id,user_id,,status,user_idstatus
101,1,250.0,Completed,1Completed
102,2,400.0,Failed,2Failed
103,3,150.0,Completed,3Completed
104,4,600.0,Completed,4Completed
105,5,320.0,Failed,5Failed
106,1,290.0,Completed,1Completed
107,2,500.0,Completed,2Completed
108,3,120.0,Failed,3Failed
109,4,620.0,Completed,4Completed
110,5,350.0,Completed,5Completed
[cloudera@quickstart ~]$ 
```

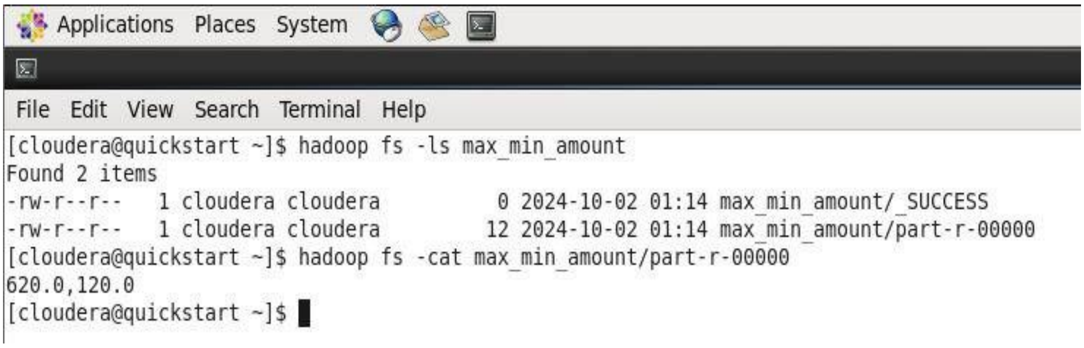
5.Use Pig to find the maximum and minimum amount values in the transaction_data.csv.

max_min_amount = FOREACH (GROUP transactions ALL) GENERATE MAX(transactions.amount) AS max_amount, MIN(transactions.amount) AS min_amount;

STORE max_min_amount INTO 'max_min_amount' USING PigStorage(',');

$ hadoop fs -get max_min_amount

$ cat max_min_amount/part-r-00000

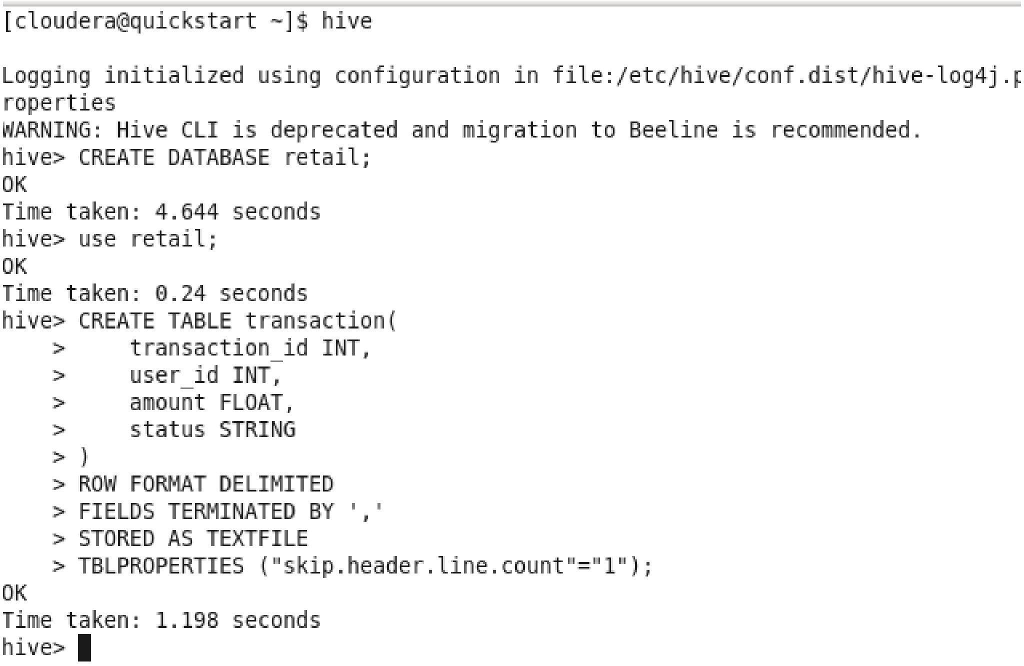

Applications Places System
File Edit View Search Terminal Help

```
[cloudera@quickstart ~]$ hadoop fs -ls max_min_amount
Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2024-10-02 01:14 max_min_amount/_SUCCESS
-rw-r--r--   1 cloudera cloudera         12 2024-10-02 01:14 max_min_amount/part-r-00000
[cloudera@quickstart ~]$ hadoop fs -cat max_min_amount/part-r-00000
620.0,120.0
[cloudera@quickstart ~]$ 
```

1. Write a HiveQL query to create a partitioned table on the status column from transaction_data.csv

  - hive;

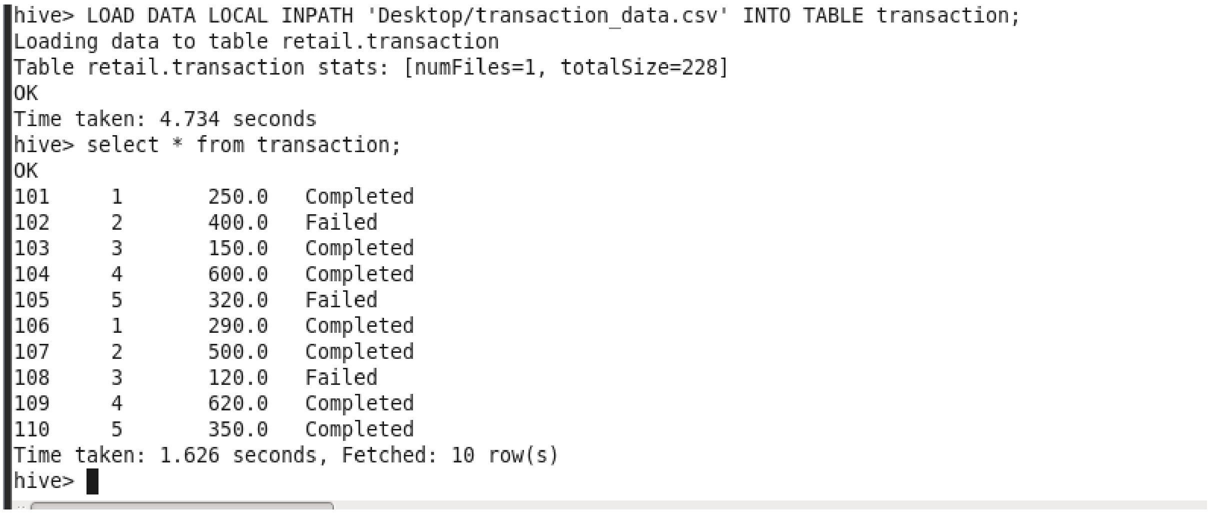
- CREATE DATABASE retail;
- USE retail;
- CREATE TABLE transaction(transaction_id INT,user_id INT, amount FLOAT,status STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE TBLPROPERTIES ("skip.header.line.count"="1");



```
[cloudera@quickstart ~]$ hive

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.p
roperties
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
hive> CREATE DATABASE retail;
OK
Time taken: 4.644 seconds
hive> use retail;
OK
Time taken: 0.24 seconds
hive> CREATE TABLE transaction(
    >     transaction_id INT,
    >     user_id INT,
    >     amount FLOAT,
    >     status STRING
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE
    > TBLPROPERTIES ("skip.header.line.count"="1");
OK
Time taken: 1.198 seconds
hive> 
```

2. Insert data into a Hive table from the transaction_data.csv dataset.

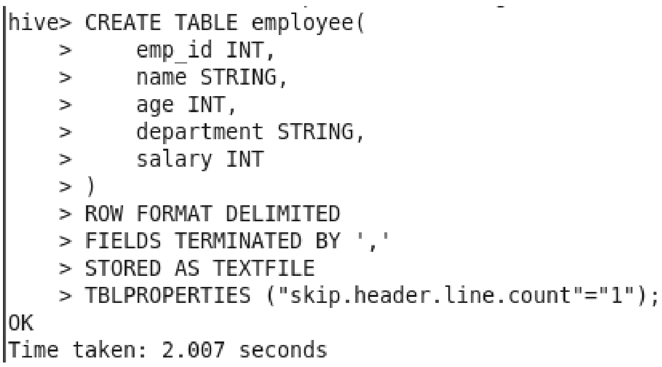
LOAD DATA LOCAL INPATH 'Desktop/transaction_data.csv' INTO TABLE transaction;



```
hive> LOAD DATA LOCAL INPATH 'Desktop/transaction_data.csv' INTO TABLE transaction;
Loading data to table retail.transaction
Table retail.transaction stats: [numFiles=1, totalSize=228]
OK
Time taken: 4.734 seconds
hive> select * from transaction;
OK
101     1       250.0   Completed
102     2       400.0   Failed
103     3       150.0   Completed
104     4       600.0   Completed
105     5       320.0   Failed
106     1       290.0   Completed
107     2       500.0   Completed
108     3       120.0   Failed
109     4       620.0   Completed
110     5       350.0   Completed
Time taken: 1.626 seconds, Fetched: 10 row(s)
hive> 
```

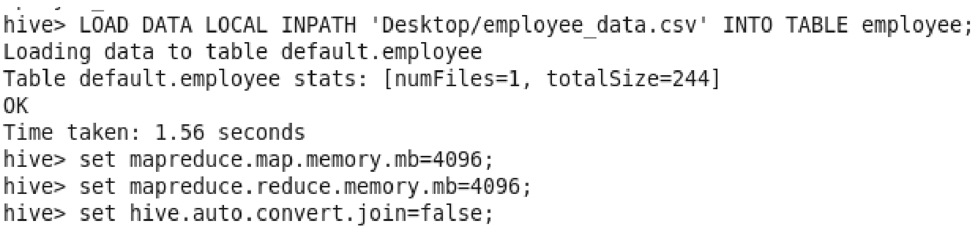
3. Perform a left outer join between the employee and transaction tables in Hive.

- CREATE TABLE employee(emp_id INT,name STRING,age INT, department STRING, salary INT) ROW FORMAT DELIMITEDTERMINATED BY ','STORED AS TEXTFILE TBLPROPERTIES ("skip.header.line.count"="1");



```
hive> CREATE TABLE employee(
    >     emp_id INT,
    >     name STRING,
    >     age INT,
    >     department STRING,
    >     salary INT
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE
    > TBLPROPERTIES ("skip.header.line.count"="1");
OK
Time taken: 2.007 seconds
```

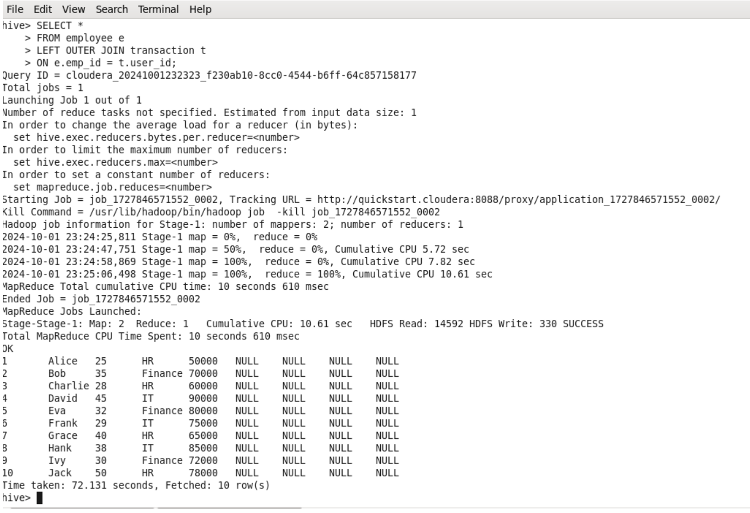
- LOAD DATA LOCAL INPATH 'Desktop/employee_data.csv' INTO TABLE employee;



```
hive> LOAD DATA LOCAL INPATH 'Desktop/employee_data.csv' INTO TABLE employee;
Loading data to table default.employee
Table default.employee stats: [numFiles=1, totalSize=244]
OK
Time taken: 1.56 seconds
hive> set mapreduce.map.memory.mb=4096;
hive> set mapreduce.reduce.memory.mb=4096;
hive> set hive.auto.convert.join=false;
```

- set mapreduce.map.memory.mb=4096;
- set mapreduce.reduce.memory.mb=4096;
- set hive.auto.convert.join=false;

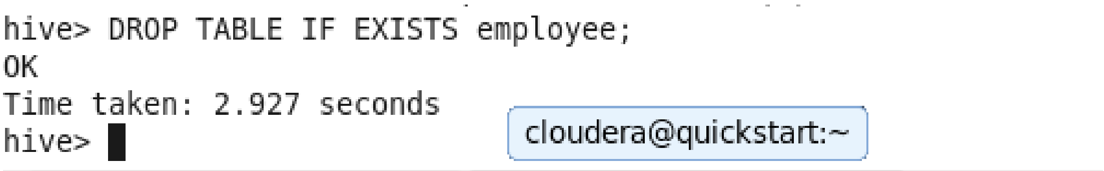
- SELECT * FROM employee e LEFT OUTER JOIN transaction t ON e.emp_id = t.user_id;



```
hive> SELECT *
    > FROM employee e
    > LEFT OUTER JOIN transaction t
    > ON e.emp_id = t.user_id;
Query ID = cloudera_20241001232323_f230ab10-8cc0-4544-b6ff-64c857158177
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>
Starting Job = job_1727846571552_0002, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1727846571552_0002/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1727846571552_0002
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1
2024-10-01 23:24:25,811 Stage-1 map = 0%,  reduce = 0%
2024-10-01 23:24:47,751 Stage-1 map = 50%,  reduce = 0%, Cumulative CPU 5.72 sec
2024-10-01 23:24:58,869 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 7.82 sec
2024-10-01 23:25:06,498 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 10.61 sec
MapReduce Total cumulative CPU time: 10 seconds 610 msec
Ended Job = job_1727846571552_0002
MapReduce Jobs Launched:
Stage-Stage-1: Map: 2  Reduce: 1   Cumulative CPU: 10.61 sec   HDFS Read: 14592 HDFS Write: 330 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 610 msec
OK
1       Alice   25      HR      50000   NULL    NULL    NULL    NULL
2       Bob     35      Finance 70000   NULL    NULL    NULL    NULL
3       Charlie 28      HR      60000   NULL    NULL    NULL    NULL
4       David   45      IT      90000   NULL    NULL    NULL    NULL
5       Eva     32      Finance 80000   NULL    NULL    NULL    NULL
6       Frank   29      IT      75000   NULL    NULL    NULL    NULL
7       Grace   40      HR      65000   NULL    NULL    NULL    NULL
8       Hank    38      IT      85000   NULL    NULL    NULL    NULL
9       Ivy     30      Finance 72000   NULL    NULL    NULL    NULL
10      Jack    50      HR      78000   NULL    NULL    NULL    NULL
Time taken: 72.131 seconds, Fetched: 10 row(s)
hive>
```

4. Drop the employee table in Hive if it exists.

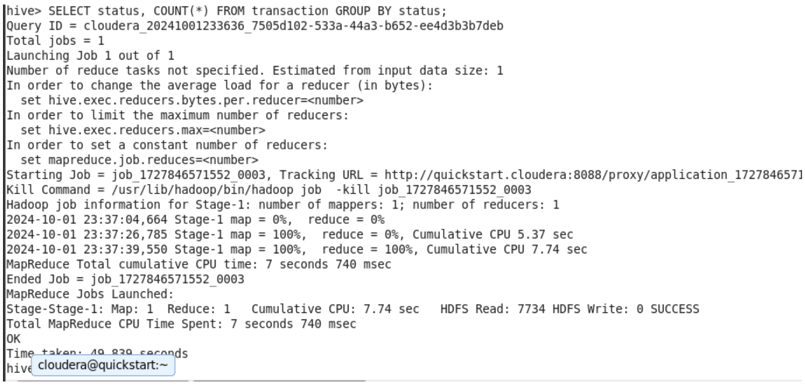
DROP TABLE IF EXISTS employee;



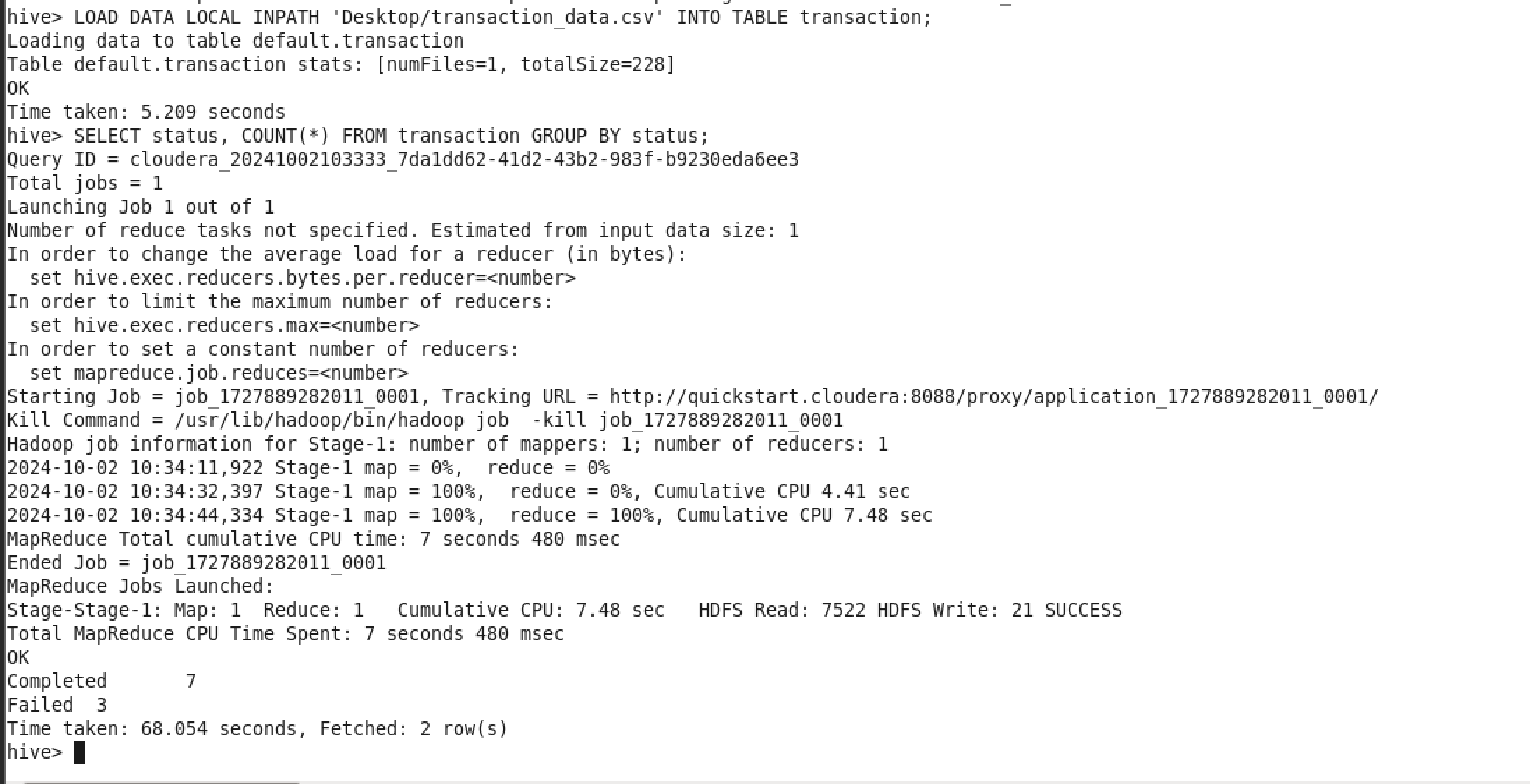
```
hive> DROP TABLE IF EXISTS employee;
OK
Time taken: 2.927 seconds
hive>
```

5. Group the transaction table by status and calculate the total count of records for each group.

SELECT status, COUNT(*) FROM transaction GROUP BY status;



```
hive> SELECT status, COUNT(*) FROM transaction GROUP BY status;
Query ID = cloudera_20241001233636_7505d102-533a-44a3-b652-ee4d3b3b7deb
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>
Starting Job = job_1727846571552_0003, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1727846571
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1727846571552_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2024-10-01 23:37:04,664 Stage-1 map = 0%,  reduce = 0%
2024-10-01 23:37:26,785 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.37 sec
2024-10-01 23:37:39,550 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.74 sec
MapReduce Total cumulative CPU time: 7 seconds 740 msec
Ended Job = job_1727846571552_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.74 sec   HDFS Read: 7734 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 740 msec
OK
Time taken: 49.839 seconds
hive
```

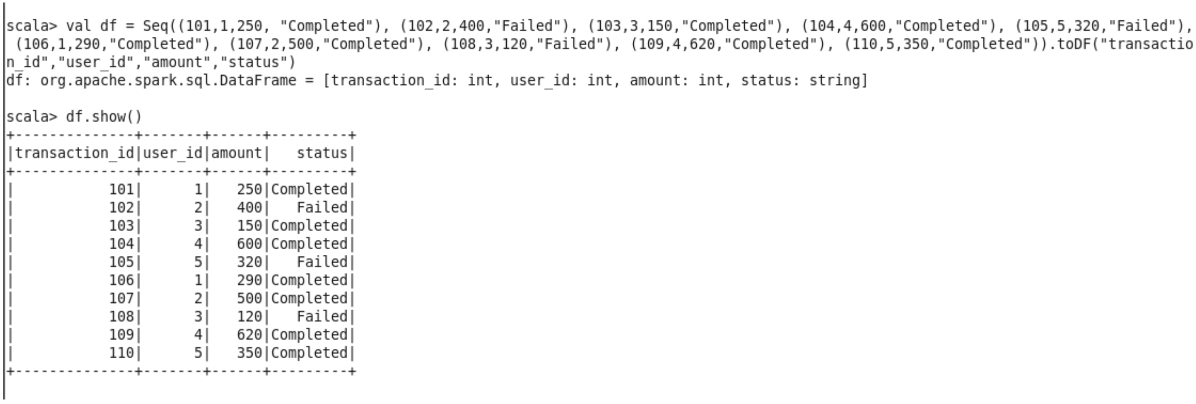


```
hive> LOAD DATA LOCAL INPATH 'Desktop/transaction_data.csv' INTO TABLE transaction;
Loading data to table default.transaction
Table default.transaction stats: [numFiles=1, totalSize=228]
OK
Time taken: 5.209 seconds
hive> SELECT status, COUNT(*) FROM transaction GROUP BY status;
Query ID = cloudera_20241002103333_7da1dd62-41d2-43b2-983f-b9230eda6ee3
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>
Starting Job = job_1727889282011_0001, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1727889282011_0001/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1727889282011_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2024-10-02 10:34:11,922 Stage-1 map = 0%,  reduce = 0%
2024-10-02 10:34:32,397 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.41 sec
2024-10-02 10:34:44,334 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.48 sec
MapReduce Total cumulative CPU time: 7 seconds 480 msec
Ended Job = job_1727889282011_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.48 sec   HDFS Read: 7522 HDFS Write: 21 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 480 msec
OK
Completed       7
Failed  3
Time taken: 68.054 seconds, Fetched: 2 row(s)
hive> 
```

Module 3: Spark

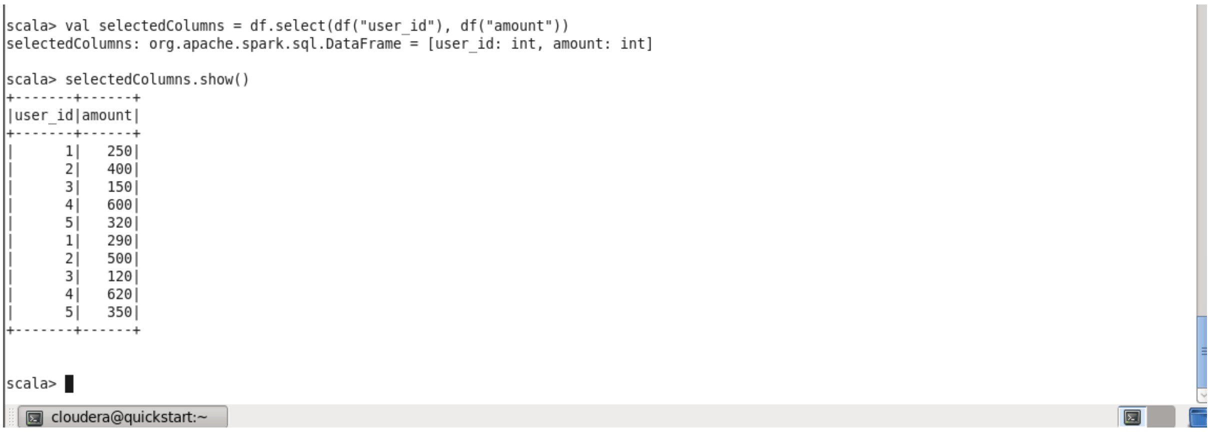
1. **Select the user_id and amount columns from the transaction_data.csv in Spark.**

   Step 1: **Create the transaction data frame.**
   val df = Seq((101,1,250, "Completed"), (102,2,400,"Failed"), (103,3,150,"Completed"), (104,4,600,"Completed"), (105,5,320,"Failed"), (106,1,290,"Completed"), (107,2,500,"Completed"), (108,3,120,"Failed"), (109,4,620,"Completed"), (110,5,350,"Completed")).toDF("transaction_id","user_id","amount","status")



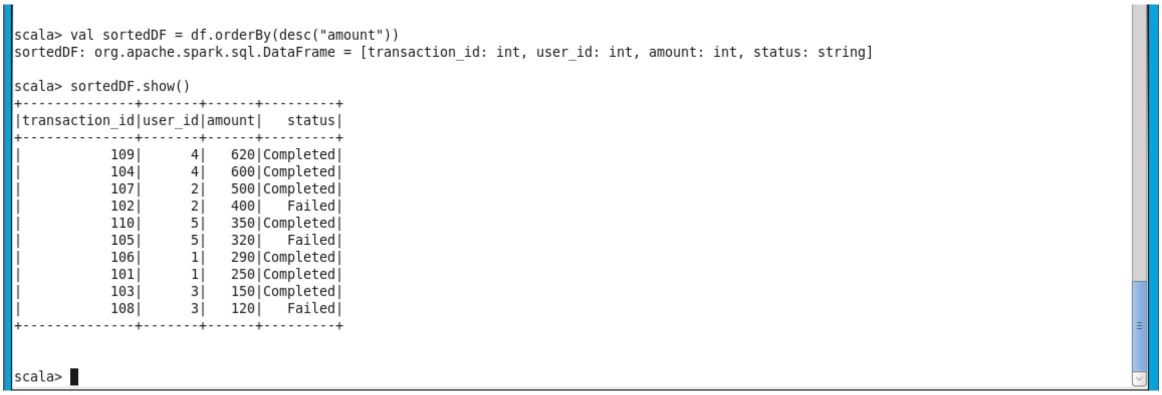
```
scala> val df = Seq((101,1,250, "Completed"), (102,2,400,"Failed"), (103,3,150,"Completed"), (104,4,600,"Completed"), (105,5,320,"Failed"),
 (106,1,290,"Completed"), (107,2,500,"Completed"), (108,3,120,"Failed"), (109,4,620,"Completed"), (110,5,350,"Completed")).toDF("transactio
n_id","user_id","amount","status")
df: org.apache.spark.sql.DataFrame = [transaction_id: int, user_id: int, amount: int, status: string]

scala> df.show()
+--------------+-------+------+---------+
|transaction_id|user_id|amount|   status|
+--------------+-------+------+---------+
|           101|      1|   250|Completed|
|           102|      2|   400|   Failed|
|           103|      3|   150|Completed|
|           104|      4|   600|Completed|
|           105|      5|   320|   Failed|
|           106|      1|   290|Completed|
|           107|      2|   500|Completed|
|           108|      3|   120|   Failed|
|           109|      4|   620|Completed|
|           110|      5|   350|Completed|
+--------------+-------+------+---------+
```

   Step 2: Select only the specified columns and display it.
   val selectedColumns = df.select(df(“user_id”), df(“amount”))
   selectedColumns.show()



```
scala> val selectedColumns = df.select(df("user_id"), df("amount"))
selectedColumns: org.apache.spark.sql.DataFrame = [user_id: int, amount: int]

scala> selectedColumns.show()
+-------+------+
|user_id|amount|
+-------+------+
|      1|   250|
|      2|   400|
|      3|   150|
|      4|   600|
|      5|   320|
|      1|   290|
|      2|   500|
|      3|   120|
|      4|   620|
|      5|   350|
+-------+------+


scala> 
```

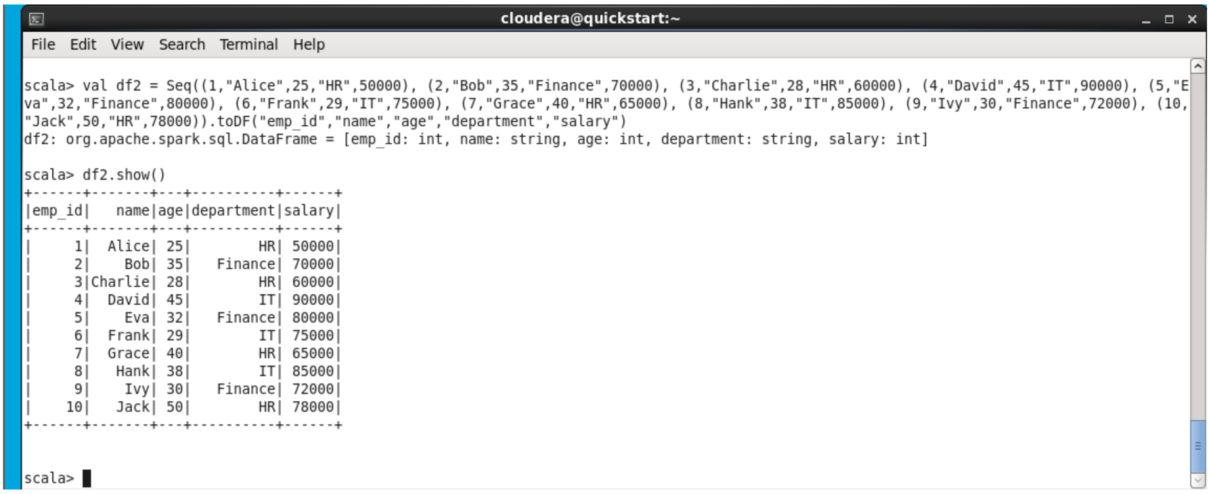
2. **Sort the DataFrame by amount in descending order.**
   Step 1: Sort the data frame that you created in the previous question.
   val sortedDF = df.orderBy(desc("amount"))
   Step 2: Display.
   sortedDF.show()



```
scala> val sortedDF = df.orderBy(desc("amount"))
sortedDF: org.apache.spark.sql.DataFrame = [transaction_id: int, user_id: int, amount: int, status: string]

scala> sortedDF.show()
+--------------+-------+------+---------+
|transaction_id|user_id|amount|   status|
+--------------+-------+------+---------+
|           109|      4|   620|Completed|
|           104|      4|   600|Completed|
|           107|      2|   500|Completed|
|           102|      2|   400|   Failed|
|           110|      5|   350|Completed|
|           105|      5|   320|   Failed|
|           106|      1|   290|Completed|
|           101|      1|   250|Completed|
|           103|      3|   150|Completed|
|           108|      3|   120|   Failed|
+--------------+-------+------+---------+


scala> 
```

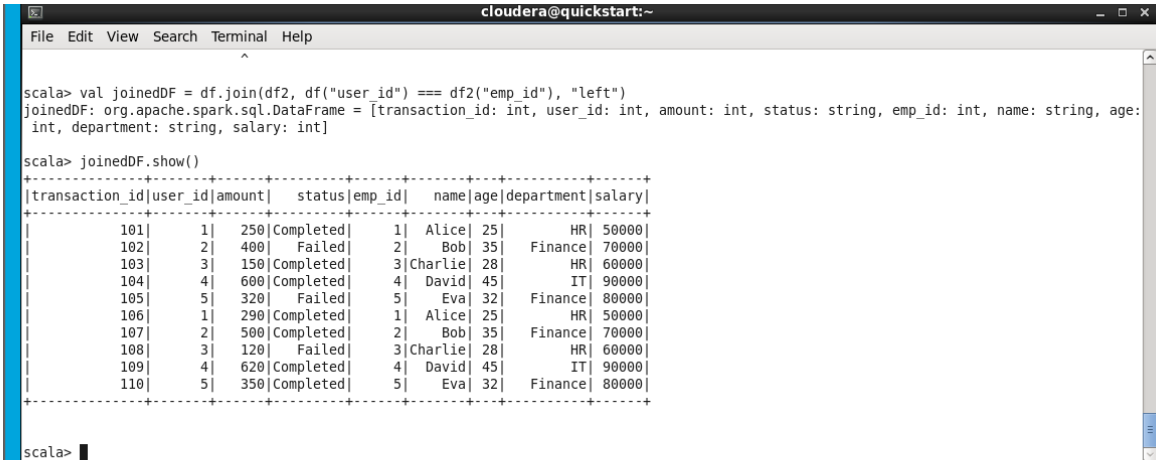
3. **Perform a left join between the transaction and employee DataFrames based on user_id and emp_id.**

Step 1: Create the employee data frame.
val df2 = Seq((1,"Alice",25,"HR",50000), (2,"Bob",35,"Finance",70000), (3,"Charlie",28,"HR",60000), (4,"David",45,"IT",90000), (5,"Eva",32,"Finance",80000), (6,"Frank",29,"IT",75000), (7,"Grace",40,"HR",65000), (8,"Hank",38,"IT",85000), (9,"Ivy",30,"Finance",72000), (10,"Jack",50,"HR",78000)).toDF("emp_id","name","age","department","salary")



```
scala> val df2 = Seq((1,"Alice",25,"HR",50000), (2,"Bob",35,"Finance",70000), (3,"Charlie",28,"HR",60000), (4,"David",45,"IT",90000), (5,"E
va",32,"Finance",80000), (6,"Frank",29,"IT",75000), (7,"Grace",40,"HR",65000), (8,"Hank",38,"IT",85000), (9,"Ivy",30,"Finance",72000), (10,
"Jack",50,"HR",78000)).toDF("emp_id","name","age","department","salary")
df2: org.apache.spark.sql.DataFrame = [emp_id: int, name: string, age: int, department: string, salary: int]

scala> df2.show()
+------+-------+---+----------+------+
|emp_id|   name|age|department|salary|
+------+-------+---+----------+------+
|     1|  Alice| 25|        HR| 50000|
|     2|    Bob| 35|   Finance| 70000|
|     3|Charlie| 28|        HR| 60000|
|     4|  David| 45|        IT| 90000|
|     5|    Eva| 32|   Finance| 80000|
|     6|  Frank| 29|        IT| 75000|
|     7|  Grace| 40|        HR| 65000|
|     8|   Hank| 38|        IT| 85000|
|     9|    Ivy| 30|   Finance| 72000|
|    10|   Jack| 50|        HR| 78000|
+------+-------+---+----------+------+


scala> 
```

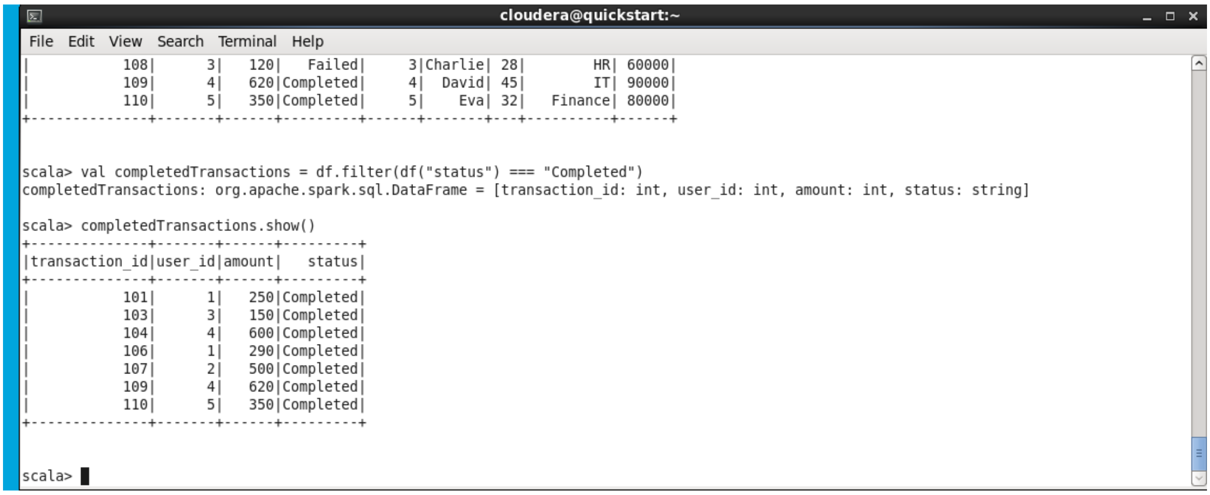
Step 2: Join the two dataframes using user_id from df and emp_if from df2 and then display it.
val joinedDF = df.join(df("user_id") === df2("emp_id"), "left")
joinedDF.show()



```
                            ^

scala> val joinedDF = df.join(df2, df("user_id") === df2("emp_id"), "left")
joinedDF: org.apache.spark.sql.DataFrame = [transaction_id: int, user_id: int, amount: int, status: string, emp_id: int, name: string, age:
 int, department: string, salary: int]

scala> joinedDF.show()
+--------------+-------+------+---------+------+-------+---+----------+------+
|transaction_id|user_id|amount|   status|emp_id|   name|age|department|salary|
+--------------+-------+------+---------+------+-------+---+----------+------+
|           101|      1|   250|Completed|     1|  Alice| 25|        HR| 50000|
|           102|      2|   400|   Failed|     2|    Bob| 35|   Finance| 70000|
|           103|      3|   150|Completed|     3|Charlie| 28|        HR| 60000|
|           104|      4|   600|Completed|     4|  David| 45|        IT| 90000|
|           105|      5|   320|   Failed|     5|    Eva| 32|   Finance| 80000|
|           106|      1|   290|Completed|     1|  Alice| 25|        HR| 50000|
|           107|      2|   500|Completed|     2|    Bob| 35|   Finance| 70000|
|           108|      3|   120|   Failed|     3|Charlie| 28|        HR| 60000|
|           109|      4|   620|Completed|     4|  David| 45|        IT| 90000|
|           110|      5|   350|Completed|     5|    Eva| 32|   Finance| 80000|
+--------------+-------+------+---------+------+-------+---+----------+------+


scala> 
```

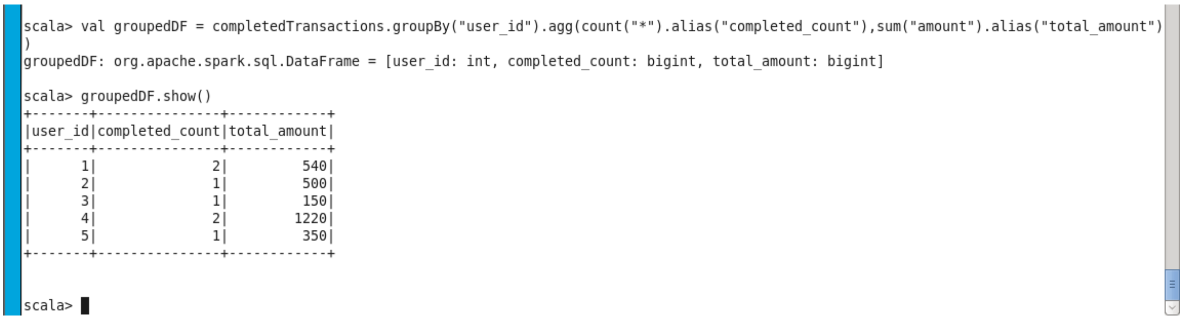
4. Filter rows in the transaction DataFrame where status is Completed and group by user_id.

Step 1: Filter the rows where the status is completed.



```
|           108|      3|   120|   Failed|     3|Charlie| 28|        HR| 60000|
|           109|      4|   620|Completed|     4|  David| 45|        IT| 90000|
|           110|      5|   350|Completed|     5|    Eva| 32|   Finance| 80000|
+--------------+-------+------+---------+------+-------+---+----------+------+


scala> val completedTransactions = df.filter(df("status") === "Completed")
completedTransactions: org.apache.spark.sql.DataFrame = [transaction_id: int, user_id: int, amount: int, status: string]

scala> completedTransactions.show()
+--------------+-------+------+---------+
|transaction_id|user_id|amount|   status|
+--------------+-------+------+---------+
|           101|      1|   250|Completed|
|           103|      3|   150|Completed|
|           104|      4|   600|Completed|
|           106|      1|   290|Completed|
|           107|      2|   500|Completed|
|           109|      4|   620|Completed|
|           110|      5|   350|Completed|
+--------------+-------+------+---------+


scala> 
```

Step 2: Then from the filtered rows, group the rows based on user_id and show the user_id, count (i.e., number of entries for a particular user) and total amount.



```
scala> val groupedDF = completedTransactions.groupBy("user_id").agg(count("*").alias("completed_count"),sum("amount").alias("total_amount")
)
groupedDF: org.apache.spark.sql.DataFrame = [user_id: int, completed_count: bigint, total_amount: bigint]

scala> groupedDF.show()
+-------+---------------+------------+
|user_id|completed_count|total_amount|
+-------+---------------+------------+
|      1|              2|         540|
|      2|              1|         500|
|      3|              1|         150|
|      4|              2|        1220|
|      5|              1|         350|
+-------+---------------+------------+


scala> 
```

5. **Cache the employee DataFrame and use it in multiple queries to improve performance.**

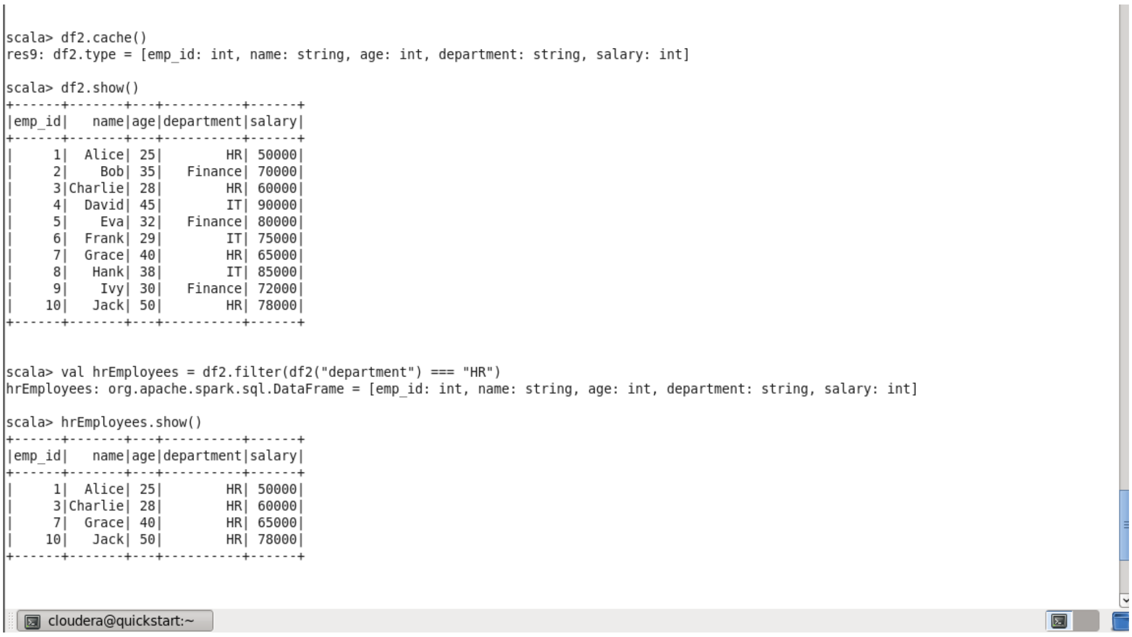
Step 1: Cache the employee data frame that was created before.

df2.cache()

Step 2: Perform multiple queries with that data frame. Here I have:

Displayed the data: df2.show()

Filtered the data to display only the rows where department is HR: val hrEmployees = df2.filter(df2("department") === "HR")

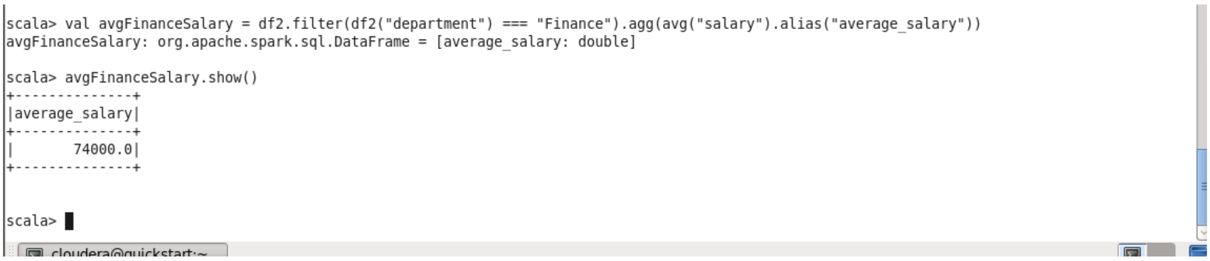


```
scala> df2.cache()
res9: df2.type = [emp_id: int, name: string, age: int, department: string, salary: int]

scala> df2.show()
+------+-------+---+----------+------+
|emp_id|   name|age|department|salary|
+------+-------+---+----------+------+
|     1|  Alice| 25|        HR| 50000|
|     2|    Bob| 35|   Finance| 70000|
|     3|Charlie| 28|        HR| 60000|
|     4|  David| 45|        IT| 90000|
|     5|    Eva| 32|   Finance| 80000|
|     6|  Frank| 29|        IT| 75000|
|     7|  Grace| 40|        HR| 65000|
|     8|   Hank| 38|        IT| 85000|
|     9|    Ivy| 30|   Finance| 72000|
|    10|   Jack| 50|        HR| 78000|
+------+-------+---+----------+------+


scala> val hrEmployees = df2.filter(df2("department") === "HR")
hrEmployees: org.apache.spark.sql.DataFrame = [emp_id: int, name: string, age: int, department: string, salary: int]

scala> hrEmployees.show()
+------+-------+---+----------+------+
|emp_id|   name|age|department|salary|
+------+-------+---+----------+------+
|     1|  Alice| 25|        HR| 50000|
|     3|Charlie| 28|        HR| 60000|
|     7|  Grace| 40|        HR| 65000|
|    10|   Jack| 50|        HR| 78000|
+------+-------+---+----------+------+
```

Computed the average salary of the people working in the Finance Department:

val avgFinanceSalary = df2.filter(df2("department") === "Finance").agg(avg("salary").alias("average_salary"))

avgFinanceSalary.show()



```
scala> val avgFinanceSalary = df2.filter(df2("department") === "Finance").agg(avg("salary").alias("average_salary"))
avgFinanceSalary: org.apache.spark.sql.DataFrame = [average_salary: double]

scala> avgFinanceSalary.show()
+--------------+
|average_salary|
+--------------+
|       74000.0|
+--------------+


scala> 
```