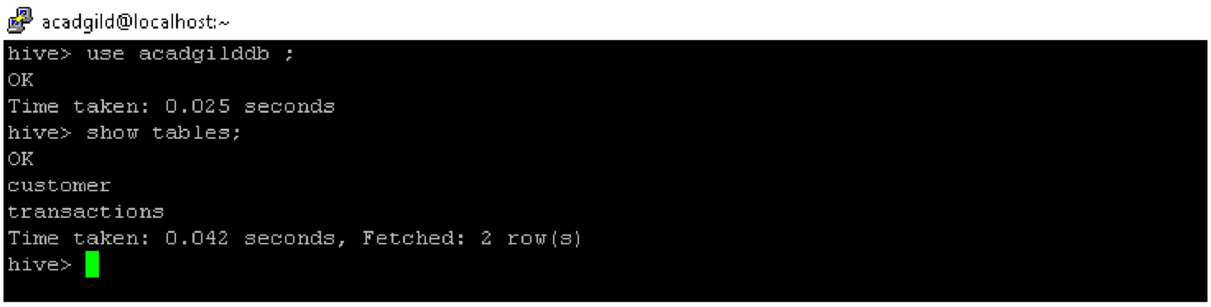
<u>CASE STUDY 1 **CUSTOMER_TRANSACTION**</u>

In this Case study, we have two tables **customer** and **transactions** in database **acadgilddb** as shown below in screenshot:

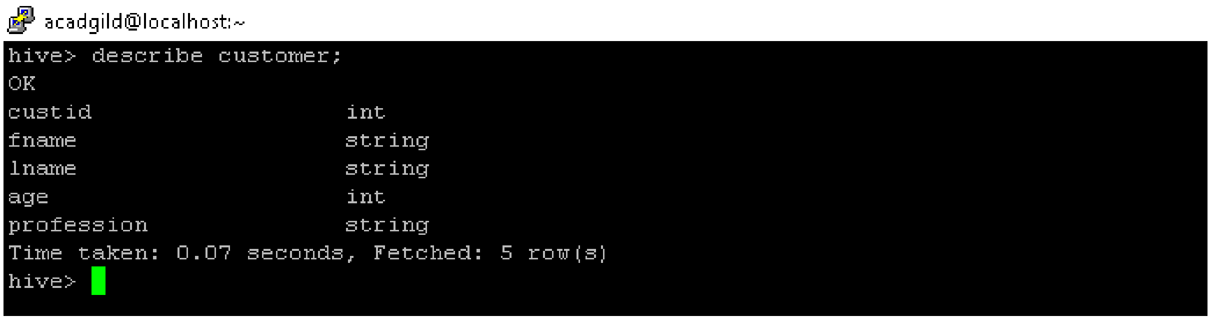


acadgild@localhost:~

```
hive> use acadgilddb ;
OK
Time taken: 0.025 seconds
hive> show tables;
OK
customer
transactions
Time taken: 0.042 seconds, Fetched: 2 row(s)
hive>
```

customer table have five columns consist of **customer ID, customer first name, customer last name, age** and **customer profession**.

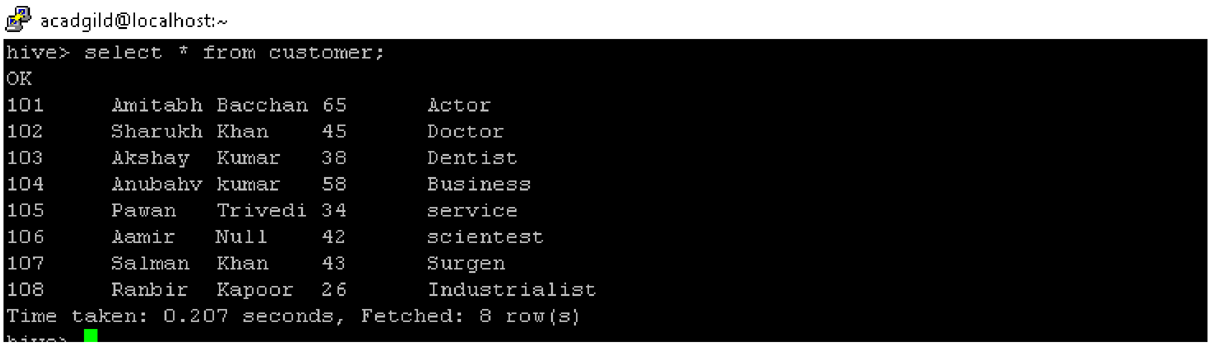
We can find customer schema by typing: **describe customer** as shown below:



acadgild@localhost:~

```
hive> describe customer;
OK
custid                  int
fname                   string
lname                   string
age                     int
profession              string
Time taken: 0.07 seconds, Fetched: 5 row(s)
hive>
```

<u>Below screen shot shows the data present in the **customer** table</u>

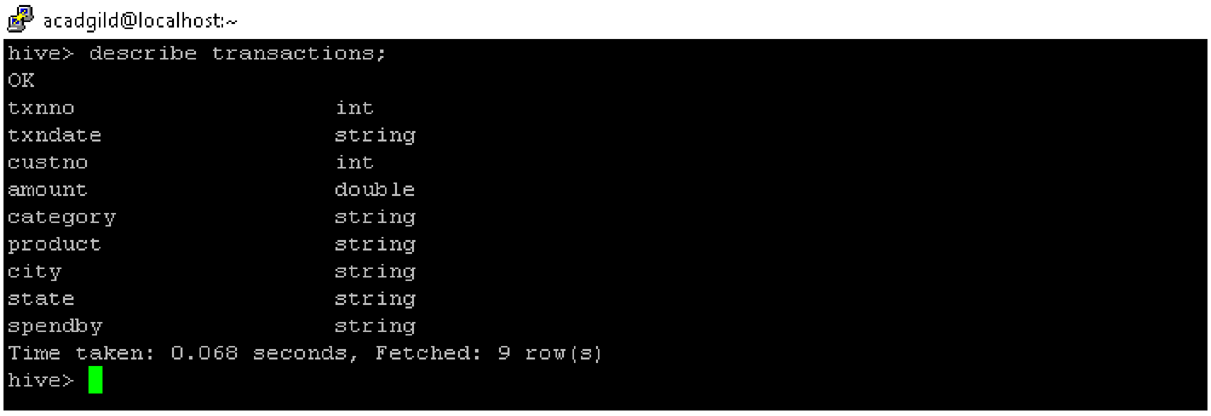


acadgild@localhost:~

```
hive> select * from customer;
OK
101     Amitabh Bacchan 65      Actor
102     Sharukh Khan    45      Doctor
103     Akshay  Kumar   38      Dentist
104     Anubahv kumar   58      Business
105     Pawan   Trivedi 34      service
106     Aamir   Null    42      scientest
107     Salman  Khan    43      Surgen
108     Ranbir  Kapoor  26      Industrialist
Time taken: 0.207 seconds, Fetched: 8 row(s)
```

transaction table have nine columns consist of transaction **number, transaction date, customer ID, amount,category,product detail,city,state,spendby details**.

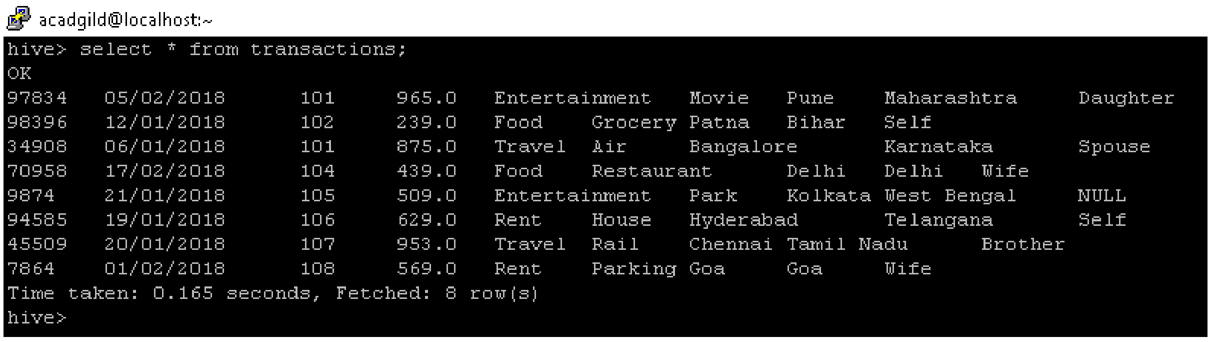
We will find this detail about table by: **"describe transaction"** as shown below.



acadgild@localhost:~

```
hive> describe transactions;
OK
txnno                   int
txndate                 string
custno                  int
amount                  double
category                string
product                 string
city                    string
state                   string
spendby                 string
Time taken: 0.068 seconds, Fetched: 9 row(s)
hive>
```

Below screen shot shows the **transactions** table data.



acadgild@localhost:~

```
hive> select * from transactions;
OK
97834   05/02/2018      101     965.0   Entertainment   Movie   Pune    Maharashtra     Daughter
98396   12/01/2018      102     239.0   Food    Grocery Patna   Bihar   Self
34908   06/01/2018      101     875.0   Travel  Air     Bangalore       Karnataka       Spouse
70958   17/02/2018      104     439.0   Food    Restaurant      Delhi   Delhi   Wife
9874    21/01/2018      105     509.0   Entertainment   Park    Kolkata West Bengal     NULL
94585   19/01/2018      106     629.0   Rent    House   Hyderabad       Telangana       Self
45509   20/01/2018      107     953.0   Travel  Rail    Chennai Tamil Nadu      Brother
7864    01/02/2018      108     569.0   Rent    Parking Goa     Goa     Wife
Time taken: 0.165 seconds, Fetched: 8 row(s)
hive>
```

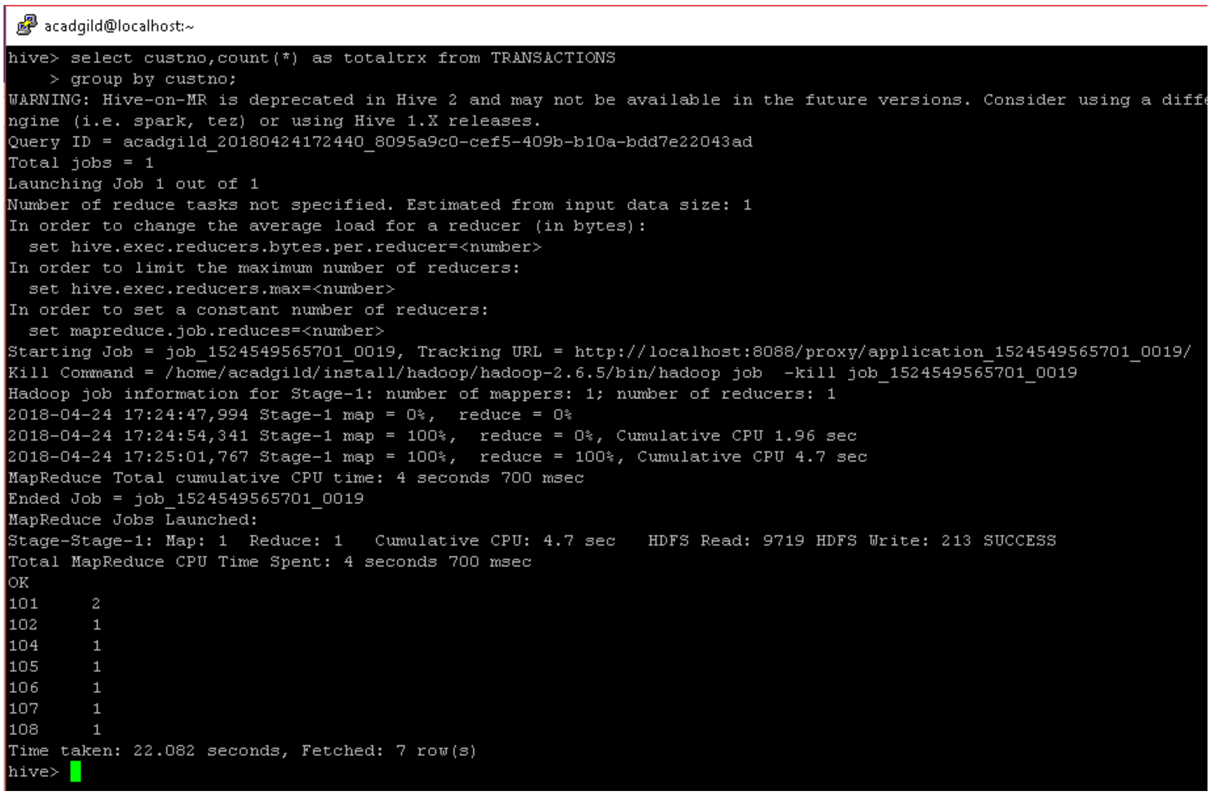
Let us solve the following use cases using these tables:-

1. Find out the number of transaction done by each customer.

To find the number of transaction done by each customer we have to use following query to get the result:-

SELECT custno, **COUNT**(*) as totaltrx **FROM** transactions

GROUP BY custno;


acadgild@localhost:~

```
hive> select custno,count(*) as totaltrx from TRANSACTIONS
    > group by custno;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a diff
ngine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = acadgild_20180424172440_8095a9c0-cef5-409b-b10a-bdd7e22043ad
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_1524549565701_0019, Tracking URL = http://localhost:8088/proxy/application_1524549565701_0019/
Kill Command = /home/acadgild/install/hadoop/hadoop-2.6.5/bin/hadoop job  -kill job_1524549565701_0019
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-04-24 17:24:47,994 Stage-1 map = 0%,  reduce = 0%
2018-04-24 17:24:54,341 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.96 sec
2018-04-24 17:25:01,767 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.7 sec
MapReduce Total cumulative CPU time: 4 seconds 700 msec
Ended Job = job_1524549565701_0019
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.7 sec   HDFS Read: 9719 HDFS Write: 213 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 700 msec
OK
101     2
102     1
104     1
105     1
106     1
107     1
108     1
Time taken: 22.082 seconds, Fetched: 7 row(s)
hive>
```

Above screen shot shows the output with number of transaction and customer ID.

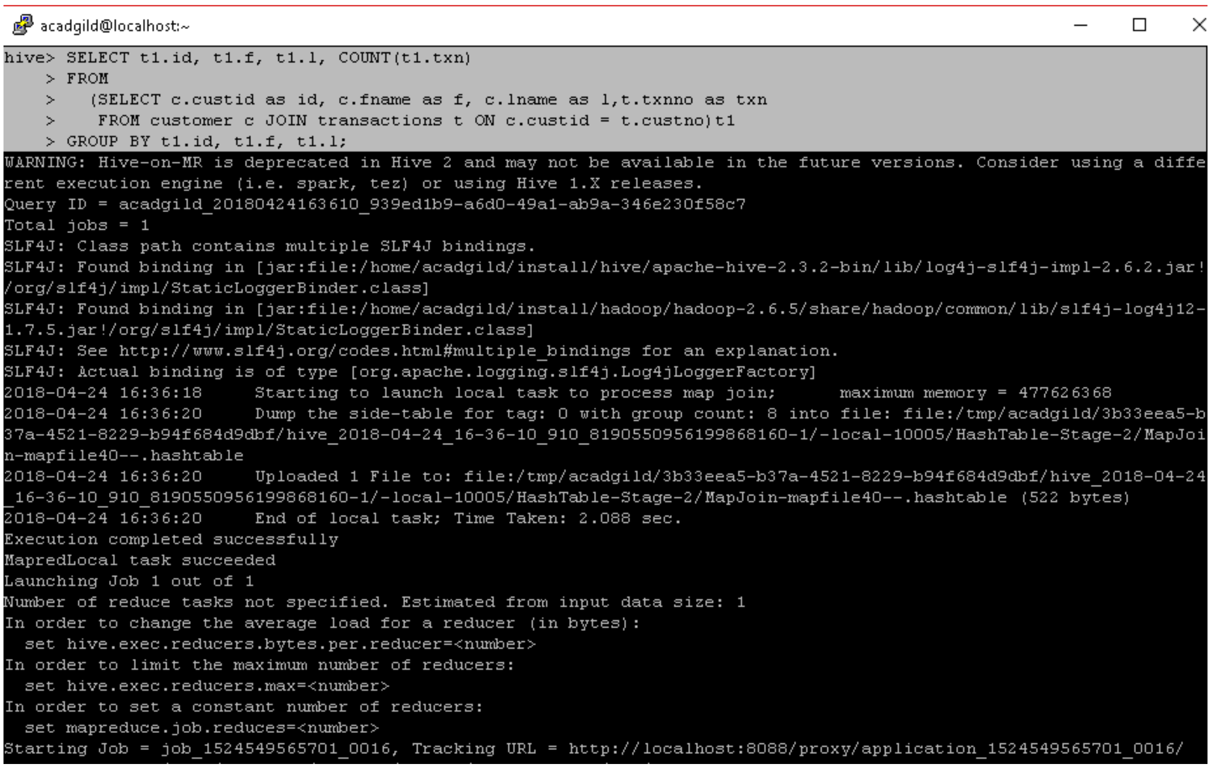
We can also find the number of transaction done by each customer by getting the name of the customer by using join query as shown below

SELECT t1.id, t1.f, t1.l, **COUNT** (t1.txn) **FROM**

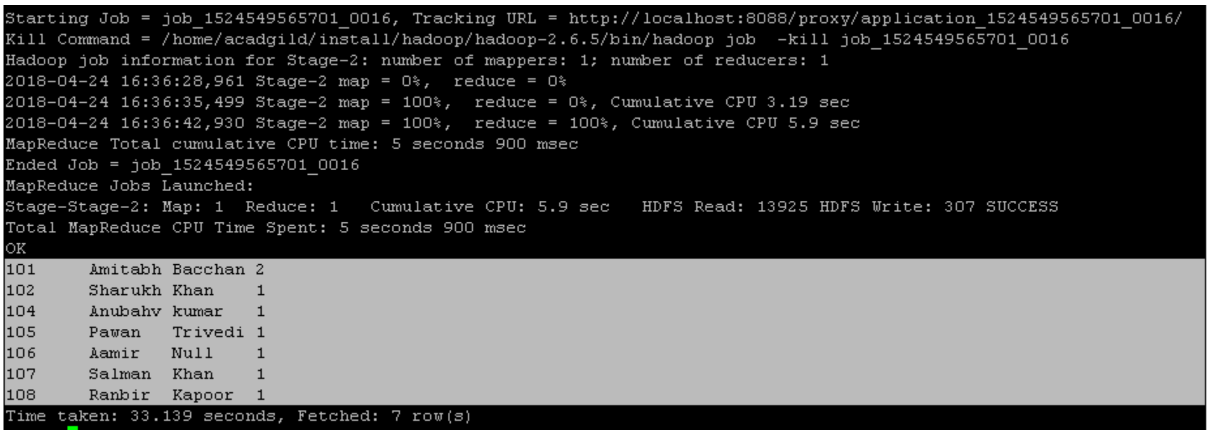
(SELECT c.custid as id, c.fname as f, c.lname as l, t.txnno as txn

FROM customer c **JOIN** transactions t ON c.custid = t.custno) t1

GROUP BY t1.id, t1.f, t1.l**;**



```
hive> SELECT t1.id, t1.f, t1.l, COUNT(t1.txn)
    > FROM
    >   (SELECT c.custid as id, c.fname as f, c.lname as l,t.txnno as txn
    >    FROM customer c JOIN transactions t ON c.custid = t.custno)t1
    > GROUP BY t1.id, t1.f, t1.l;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a diffe
rent execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = acadgild_20180424163610_939ed1b9-a6d0-49a1-ab9a-346e230f58c7
Total jobs = 1
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/acadgild/install/hive/apache-hive-2.3.2-bin/lib/log4j-slf4j-impl-2.6.2.jar!
/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/common/lib/slf4j-log4j12-
1.7.5.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.apache.logging.slf4j.Log4jLoggerFactory]
2018-04-24 16:36:18     Starting to launch local task to process map join;      maximum memory = 477626368
2018-04-24 16:36:20     Dump the side-table for tag: 0 with group count: 8 into file: file:/tmp/acadgild/3b33eea5-b
37a-4521-8229-b94f684d9dbf/hive_2018-04-24_16-36-10_910_8190550956199868160-1/-local-10005/HashTable-Stage-2/MapJoi
n-mapfile40--.hashtable
2018-04-24 16:36:20     Uploaded 1 File to: file:/tmp/acadgild/3b33eea5-b37a-4521-8229-b94f684d9dbf/hive_2018-04-24
_16-36-10_910_8190550956199868160-1/-local-10005/HashTable-Stage-2/MapJoin-mapfile40--.hashtable (522 bytes)
2018-04-24 16:36:20     End of local task; Time Taken: 2.088 sec.
Execution completed successfully
MapredLocal task succeeded
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_1524549565701_0016, Tracking URL = http://localhost:8088/proxy/application_1524549565701_0016/
```



```
Starting Job = job_1524549565701_0016, Tracking URL = http://localhost:8088/proxy/application_1524549565701_0016/
Kill Command = /home/acadgild/install/hadoop/hadoop-2.6.5/bin/hadoop job  -kill job_1524549565701_0016
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2018-04-24 16:36:28,961 Stage-2 map = 0%,  reduce = 0%
2018-04-24 16:36:35,499 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 3.19 sec
2018-04-24 16:36:42,930 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 5.9 sec
MapReduce Total cumulative CPU time: 5 seconds 900 msec
Ended Job = job_1524549565701_0016
MapReduce Jobs Launched:
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 5.9 sec   HDFS Read: 13925 HDFS Write: 307 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 900 msec
OK
101     Amitabh Bacchan 2
102     Sharukh Khan    1
104     Anubahv kumar   1
105     Pawan   Trivedi 1
106     Aamir   Null    1
107     Salman  Khan    1
108     Ranbir  Kapoor  1
Time taken: 33.139 seconds, Fetched: 7 row(s)
```

In above screen-shot we are able to see the output containing transaction done by each customer with their **first name** and **last name** and **customer ID**.

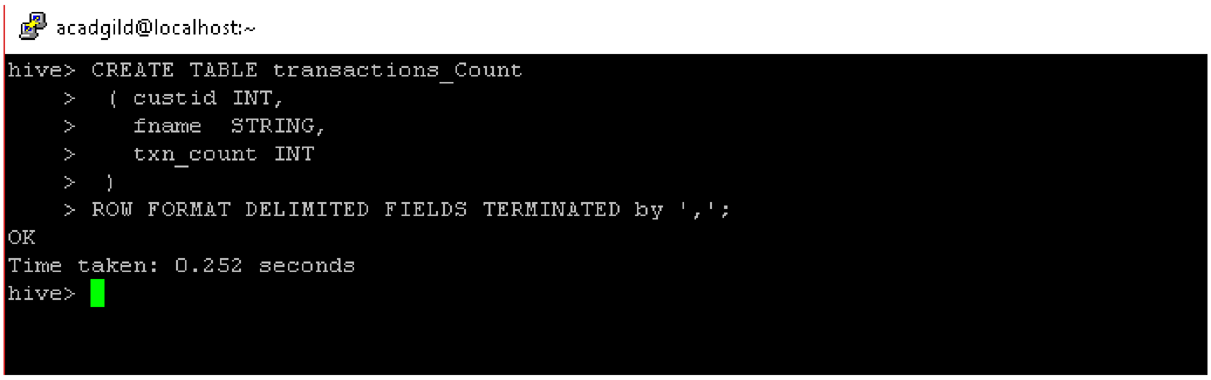
2. Create a new table called TRANSACTIONS_COUNT. This table should have three fields - custid, fname and count.

To solve above problem we have to use below query to create the table

CREATE TABLE transactions_Count

(custid **INT,** fname **STRING,** txn_count **INT)**

ROW FORMAT DELIMITED FIELDS TERMINATED by ',' ;



acadgild@localhost:~

```
hive> CREATE TABLE transactions_Count
    >  ( custid INT,
    >    fname  STRING,
    >    txn_count INT
    >  )
    > ROW FORMAT DELIMITED FIELDS TERMINATED by ',';
OK
Time taken: 0.252 seconds
hive>
```

3. Now write a hive query in such a way that the query populates the data   obtained in Step 1 above and populate the table in step 2 above.

To solve above problem we have to use **insert** query to insert data obtained from the problem number 2 into

Transactions_count

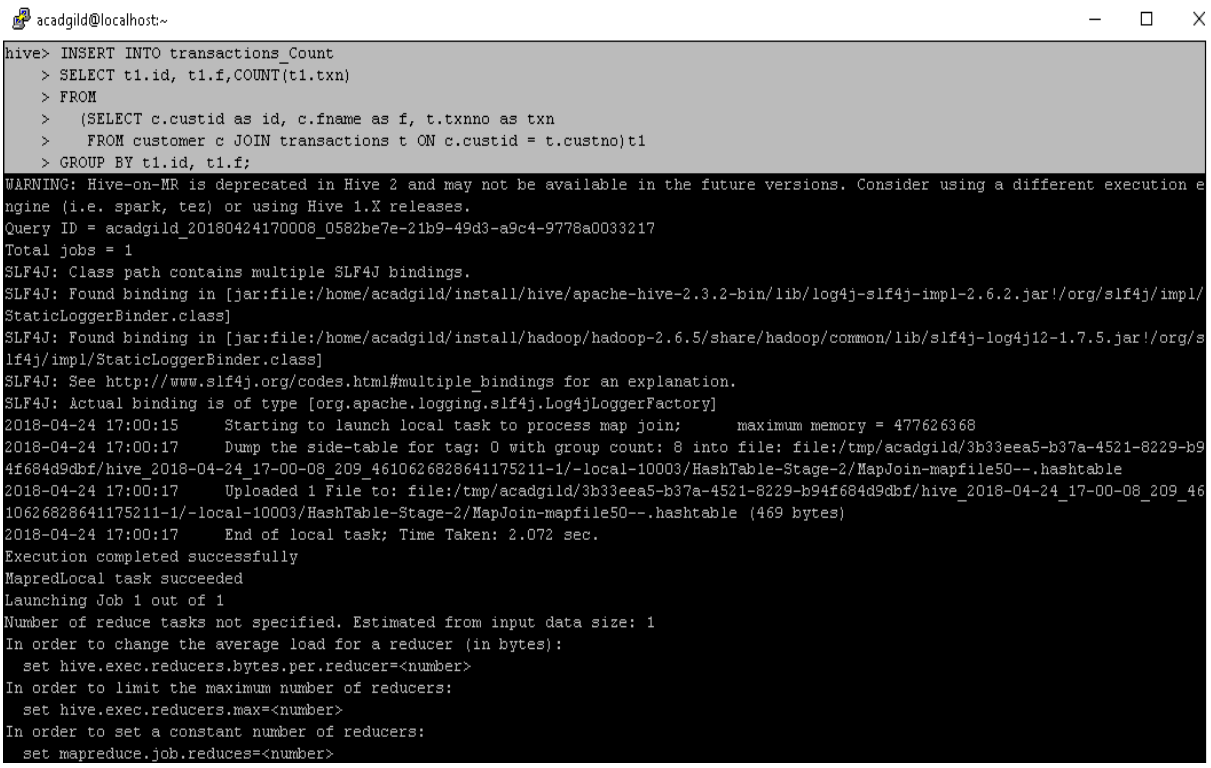
INSERT INTO transactions_Count

SELECT t1.id, t1.f,**COUNT**(t1.txn) **FROM**

(SELECT c.custid as id, c.fname as f, t.txnno as txn

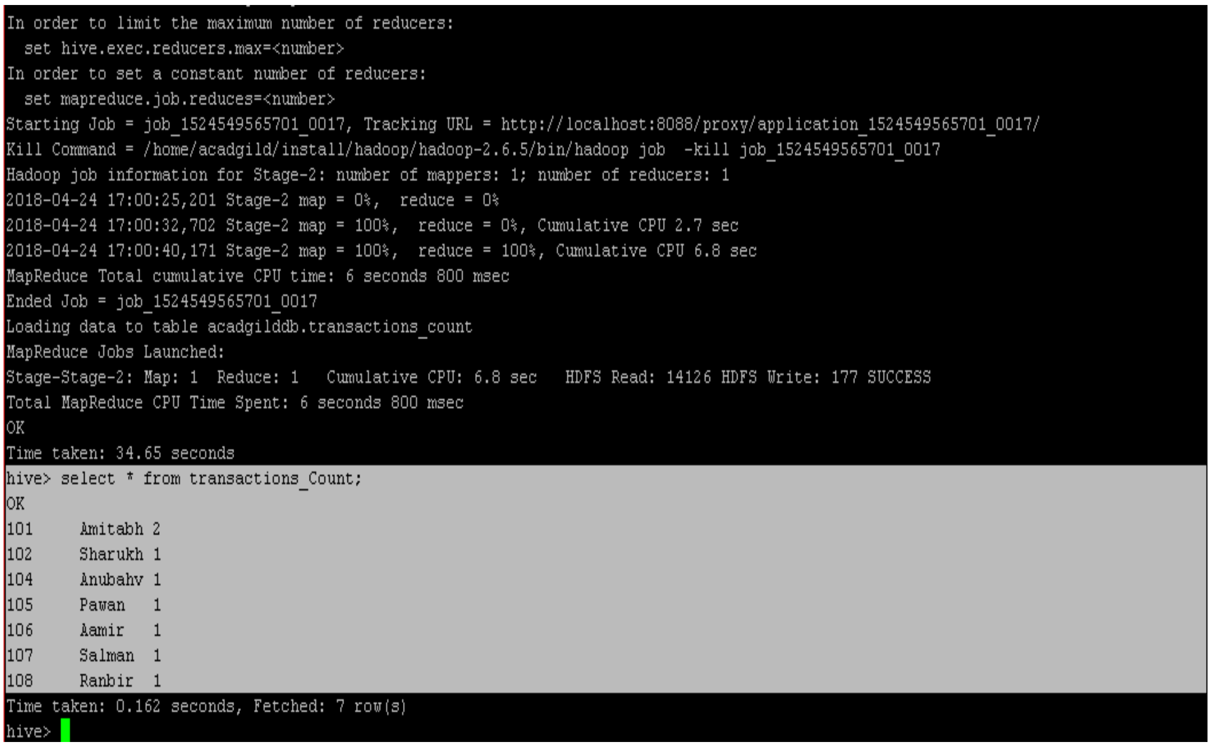
FROM customer c **JOIN** transactions t **ON** c.custid = t.custno)t1

GROUP BY t1.id, t1.f**;**



acadgild@localhost:~

```
hive> INSERT INTO transactions_Count
    > SELECT t1.id, t1.f,COUNT(t1.txn)
    > FROM
    >   (SELECT c.custid as id, c.fname as f, t.txnno as txn
    >    FROM customer c JOIN transactions t ON c.custid = t.custno)t1
    > GROUP BY t1.id, t1.f;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution e
ngine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = acadgild_20180424170008_0582be7e-21b9-49d3-a9c4-9778a0033217
Total jobs = 1
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/acadgild/install/hive/apache-hive-2.3.2-bin/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/
StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/s
lf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
2018-04-24 17:00:15     Starting to launch local task to process map join;      maximum memory = 477626368
2018-04-24 17:00:17     Dump the side-table for tag: 0 with group count: 8 into file: file:/tmp/acadgild/3b33eea5-b37a-4521-8229-b9
4f684d9dbf/hive_2018-04-24_17-00-08_209_4610626828641175211-1/-local-10003/HashTable-Stage-2/MapJoin-mapfile50--.hashtable
2018-04-24 17:00:17     Uploaded 1 File to: file:/tmp/acadgild/3b33eea5-b37a-4521-8229-b94f684d9dbf/hive_2018-04-24_17-00-08_209_46
10626828641175211-1/-local-10003/HashTable-Stage-2/MapJoin-mapfile50--.hashtable (469 bytes)
2018-04-24 17:00:17     End of local task; Time Taken: 2.072 sec.
Execution completed successfully
MapredLocal task succeeded
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>
```



```
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_1524549565701_0017, Tracking URL = http://localhost:8088/proxy/application_1524549565701_0017/
Kill Command = /home/acadgild/install/hadoop/hadoop-2.6.5/bin/hadoop job  -kill job_1524549565701_0017
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2018-04-24 17:00:25,201 Stage-2 map = 0%,  reduce = 0%
2018-04-24 17:00:32,702 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 2.7 sec
2018-04-24 17:00:40,171 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 6.8 sec
MapReduce Total cumulative CPU time: 6 seconds 800 msec
Ended Job = job_1524549565701_0017
Loading data to table acadgilddb.transactions_count
MapReduce Jobs Launched:
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 6.8 sec   HDFS Read: 14126 HDFS Write: 177 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 800 msec
OK
Time taken: 34.65 seconds
hive> select * from transactions_Count;
OK
101     Amitabh 2
102     Sharukh 1
104     Anubahv 1
105     Pawan   1
106     Aamir   1
107     Salman  1
108     Ranbir  1
Time taken: 0.162 seconds, Fetched: 7 row(s)
hive>
```

Above screen, shot shows that data obtained from query in case1 has successfully inserted in table transactions_count.

Query “SELECT *** FROM** transactions_Count“ shows the result.

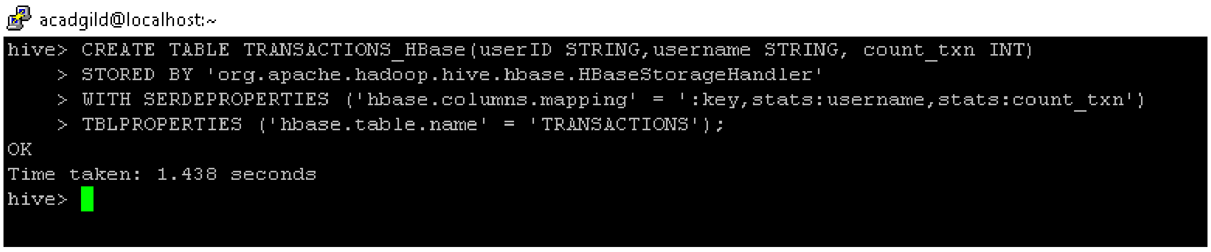
4. Now lets make the TRANSACTIONS_COUNT table Hbase complaint. In the sense, use SerDes and Storage handler features of hive to change the TRANSACTIONS_COUNT table to be able to create a TRANSACTIONS table in Hbase.

CREATE TABLE TRANSACTIONS_HBase**(**userID **STRING,**username **STRING,** count_txn **STRING)**

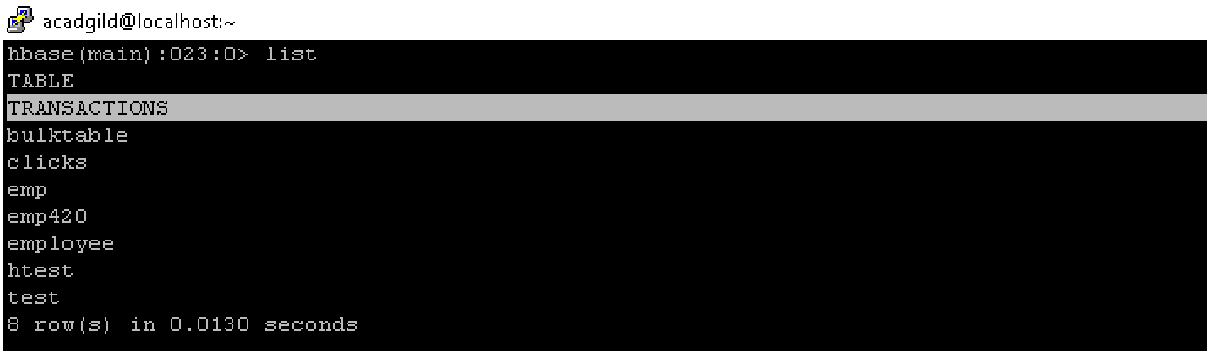
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler**'**

WITH SERDEPROPERTIES ('hbase.columns.mapping' = ': key,stats:username,stats:count_txn')

TBLPROPERTIES ('hbase.table.name' = 'TRANSACTIONS');

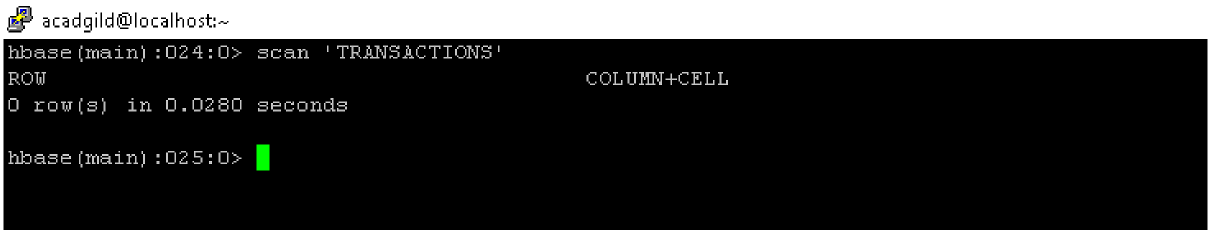


```
hive> CREATE TABLE TRANSACTIONS_HBase(userID STRING,username STRING, count_txn INT)
    > STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
    > WITH SERDEPROPERTIES ('hbase.columns.mapping' = ':key,stats:username,stats:count_txn')
    > TBLPROPERTIES ('hbase.table.name' = 'TRANSACTIONS');
OK
Time taken: 1.438 seconds
hive>
```



acadgild@localhost:~

```
hbase(main):023:0> list
TABLE
TRANSACTIONS
bulktable
clicks
emp
emp420
employee
htest
test
8 row(s) in 0.0130 seconds
```



acadgild@localhost:~

```
hbase(main):024:0> scan 'TRANSACTIONS'
ROW                                          COLUMN+CELL
0 row(s) in 0.0280 seconds

hbase(main):025:0>
```

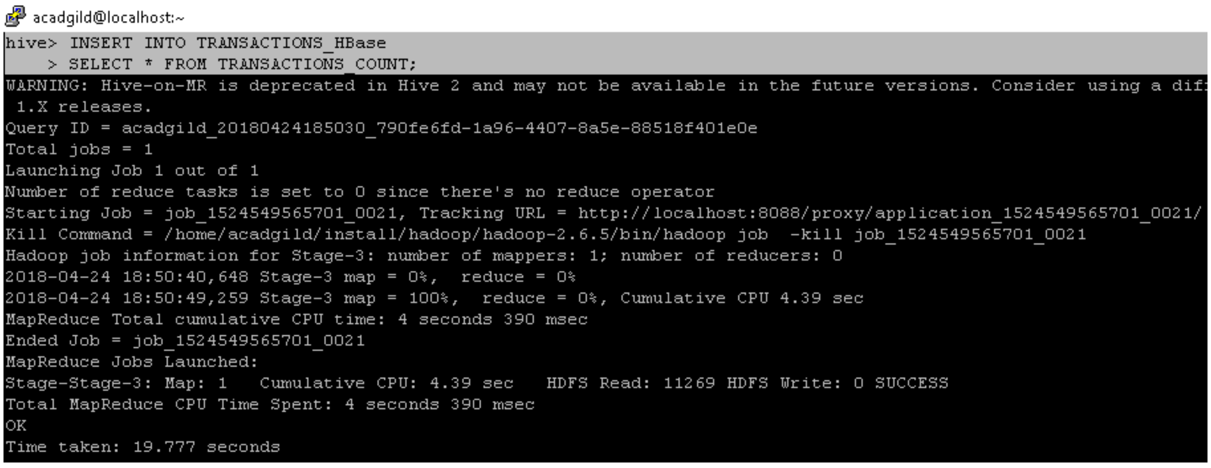
5. Now insert the data in TRANSACTIONS_Hbase table using the query in step-3 again, this should populate the Hbase TRANSACTIONS table automatically.

To solve above problem we use insert query to transfer data from TRANSACTIONS_COUNT into TRANSACTIONS_HBASE.

INSERT INTO TRANSACTIONS_HBase

SELECT * FROM TRANSACTIONS_COUNT;

<u>Below screen shot shows the TRANSACTIONS_COUNT data is successfully inserted into TRANSACTIONS_HBase</u>



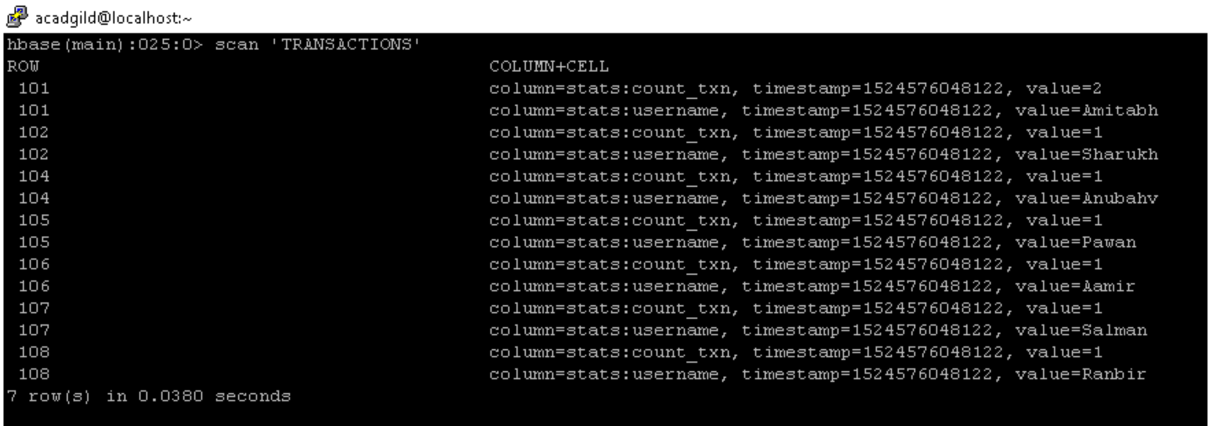
acadgild@localhost:~

```
hive> INSERT INTO TRANSACTIONS_HBase
    > SELECT * FROM TRANSACTIONS_COUNT;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a dif
 1.X releases.
Query ID = acadgild_20180424185030_790fe6fd-1a96-4407-8a5e-88518f401e0e
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1524549565701_0021, Tracking URL = http://localhost:8088/proxy/application_1524549565701_0021/
Kill Command = /home/acadgild/install/hadoop/hadoop-2.6.5/bin/hadoop job  -kill job_1524549565701_0021
Hadoop job information for Stage-3: number of mappers: 1; number of reducers: 0
2018-04-24 18:50:40,648 Stage-3 map = 0%,  reduce = 0%
2018-04-24 18:50:49,259 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 4.39 sec
MapReduce Total cumulative CPU time: 4 seconds 390 msec
Ended Job = job_1524549565701_0021
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1   Cumulative CPU: 4.39 sec   HDFS Read: 11269 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 390 msec
OK
Time taken: 19.777 seconds
```

scan **'**TRANSACTIONS'

Above command will show the content of the table.

Below screen shot shows the content of the TRANSACTIONS table by using scan command



```
acadgild@localhost:~
hbase(main):025:0> scan 'TRANSACTIONS'
ROW                                        COLUMN+CELL
 101                                       column=stats:count_txn, timestamp=1524576048122, value=2
 101                                       column=stats:username, timestamp=1524576048122, value=Amitabh
 102                                       column=stats:count_txn, timestamp=1524576048122, value=1
 102                                       column=stats:username, timestamp=1524576048122, value=Sharukh
 104                                       column=stats:count_txn, timestamp=1524576048122, value=1
 104                                       column=stats:username, timestamp=1524576048122, value=Anubahv
 105                                       column=stats:count_txn, timestamp=1524576048122, value=1
 105                                       column=stats:username, timestamp=1524576048122, value=Pawan
 106                                       column=stats:count_txn, timestamp=1524576048122, value=1
 106                                       column=stats:username, timestamp=1524576048122, value=Aamir
 107                                       column=stats:count_txn, timestamp=1524576048122, value=1
 107                                       column=stats:username, timestamp=1524576048122, value=Salman
 108                                       column=stats:count_txn, timestamp=1524576048122, value=1
 108                                       column=stats:username, timestamp=1524576048122, value=Ranbir
7 row(s) in 0.0380 seconds
```

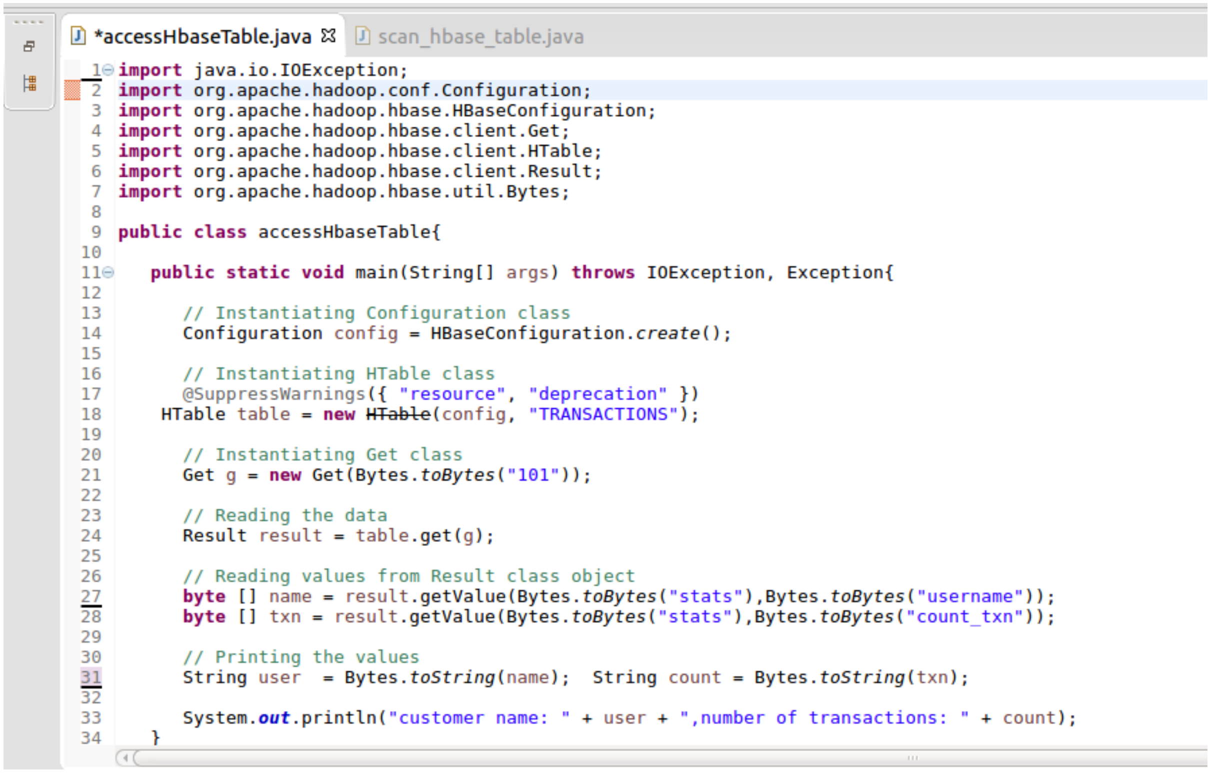
6. Now from the Hbase level, write the Hbase java API code to access and scan the TRANSACTIONS table data from java level.

To solve above problem two-java program coded in the eclipse platform to scan and access the Transaction table.

Program to access the hbase table

```
import java.io.IOException;


import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;

import org.apache.hadoop.hbase.client.Get;

import org.apache.hadoop.hbase.client.HTable;

import org.apache.hadoop.hbase.client.Result;

import org.apache.hadoop.hbase.util.Bytes;


public class accessHbaseTable{

  public static void main(String[] args) throws IOException, Exception{

      // Instantiating Configuration class

      Configuration config = HBaseConfiguration.create();
```

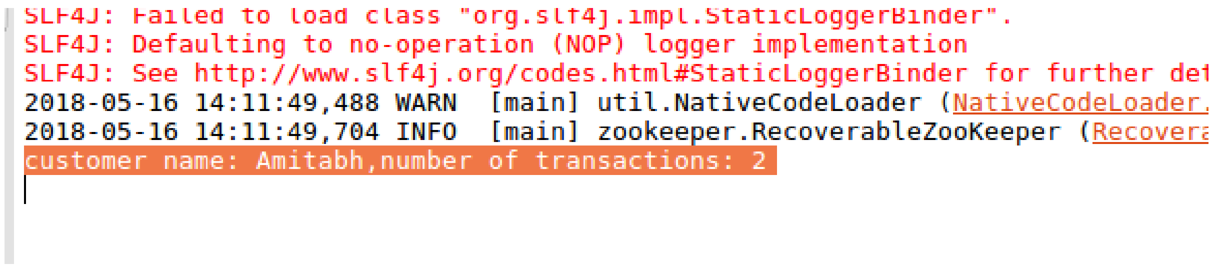
```
 // Instantiating HTable class
    @SuppressWarnings({ "resource", "deprecation" })
            HTable table = new HTable(config, "TRANSACTIONS");
// Instantiating Get class
    Get g = new Get(Bytes.toBytes("101"));
// Reading the data
    Result result = table.get(g);

// Reading values from Result class object
    byte [] name = result.getValue(Bytes.toBytes("stats"),Bytes.toBytes("username"));
    byte [] txn = result.getValue(Bytes.toBytes("stats"),Bytes.toBytes("count_txn"));

    // Printing the values
    String user  = Bytes.toString(name);
    String count = Bytes.toString(txn);
    System.out.println("customer name: " + user + ",number of transactions: " + count);
  }
}
```



*accessHbaseTable.java ⊠   scan_hbase_table.java

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.util.Bytes;

public class accessHbaseTable{

    public static void main(String[] args) throws IOException, Exception{

        // Instantiating Configuration class
        Configuration config = HBaseConfiguration.create();

        // Instantiating HTable class
        @SuppressWarnings({ "resource", "deprecation" })
      HTable table = new ~~HTable~~(config, "TRANSACTIONS");

        // Instantiating Get class
        Get g = new Get(Bytes.toBytes("101"));

        // Reading the data
        Result result = table.get(g);

        // Reading values from Result class object
        byte [] name = result.getValue(Bytes.toBytes("stats"),Bytes.toBytes("username"));
        byte [] txn = result.getValue(Bytes.toBytes("stats"),Bytes.toBytes("count_txn"));

        // Printing the values
        String user  = Bytes.toString(name);  String count = Bytes.toString(txn);

        System.out.println("customer name: " + user + ",number of transactions: " + count);
    }
```

Output: Access table program shows the value of row key 101

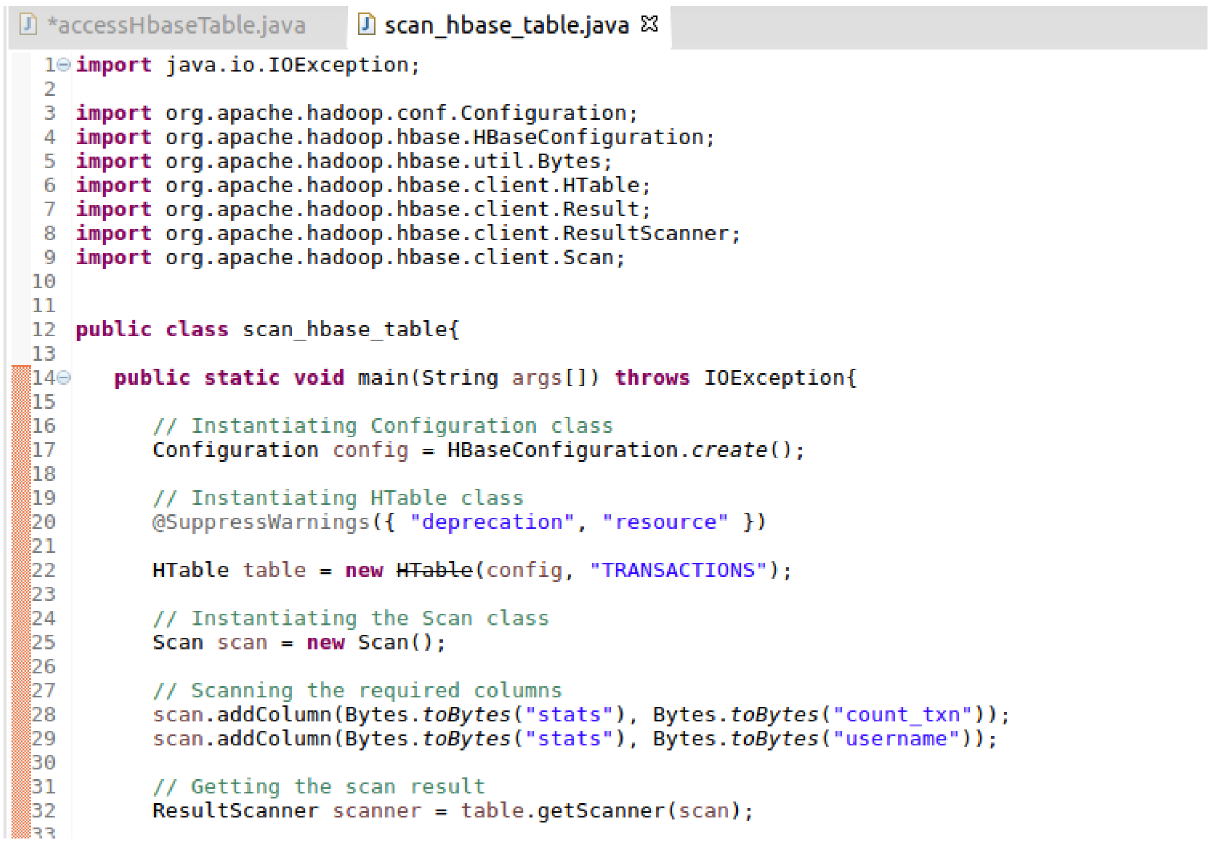


```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further de
2018-05-16 14:11:49,488 WARN  [main] util.NativeCodeLoader (NativeCodeLoader.
2018-05-16 14:11:49,704 INFO  [main] zookeeper.RecoverableZooKeeper (Recovera
customer name: Amitabh,number of transactions: 2
```

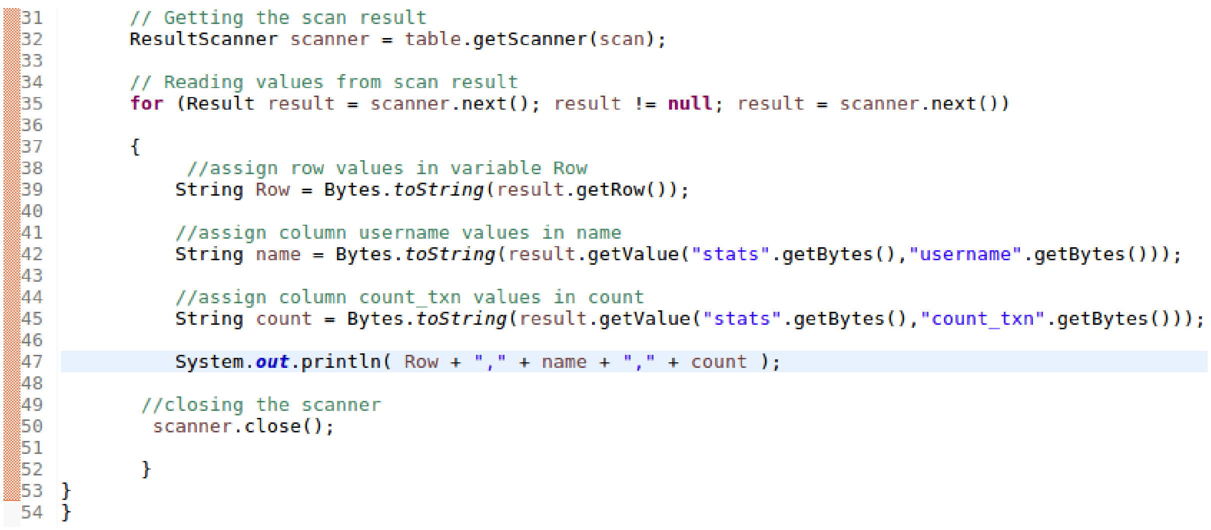
Program to scan the hbase TRANSACTIONS table:

```
import java.io.IOException;


import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;

import org.apache.hadoop.hbase.util.Bytes;

import org.apache.hadoop.hbase.client.HTable;

import org.apache.hadoop.hbase.client.Result;

import org.apache.hadoop.hbase.client.ResultScanner;

import org.apache.hadoop.hbase.client.Scan;



public class scan_hbase_table{


  public static void main(String args[]) throws IOException{


    // Instantiating Configuration class

    Configuration config = HBaseConfiguration.create();


    // Instantiating HTable class

    @SuppressWarnings({ "deprecation", "resource" })


    HTable table = new HTable(config, "TRANSACTIONS");


    // Instantiating the Scan class
```

```
      Scan scan = new Scan();


      // scanning the required columns
      scan.addColumn(Bytes.toBytes("stats"), Bytes.toBytes("count_txn"));
      scan.addColumn(Bytes.toBytes("stats"), Bytes.toBytes("username"));


      // Getting the scan result
      ResultScanner scanner = table.getScanner(scan);


      // Reading values from scan result
      for (Result result = scanner.next(); result != null; result = scanner.next())


      {
             //assign row values in variable Row
            String Row = Bytes.toString(result.getRow());


        //assign column username values in name
            String name = Bytes.toString(result.getValue("stats".getBytes(),"username".getBytes()));


            //assign column count_txn values in count
            String count = Bytes.toString(result.getValue("stats".getBytes(),"count_txn".getBytes()));


        System.out.println( Row + "," + name + "," + count );


       //closing the scanner
        scanner.close();


       }
   }}
```

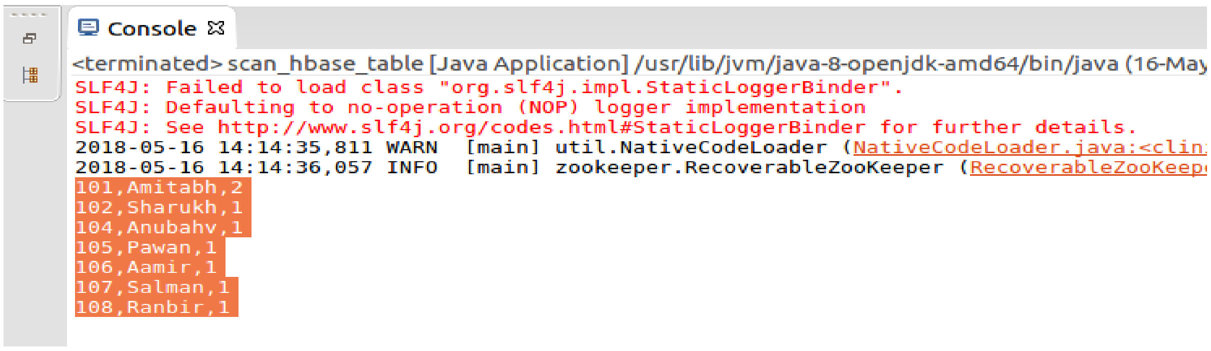


```java
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;


public class scan_hbase_table{

   public static void main(String args[]) throws IOException{

      // Instantiating Configuration class
      Configuration config = HBaseConfiguration.create();

      // Instantiating HTable class
      @SuppressWarnings({ "deprecation", "resource" })

      HTable table = new HTable(config, "TRANSACTIONS");

      // Instantiating the Scan class
      Scan scan = new Scan();

      // Scanning the required columns
      scan.addColumn(Bytes.toBytes("stats"), Bytes.toBytes("count_txn"));
      scan.addColumn(Bytes.toBytes("stats"), Bytes.toBytes("username"));

      // Getting the scan result
      ResultScanner scanner = table.getScanner(scan);
```



```java
      // Getting the scan result
      ResultScanner scanner = table.getScanner(scan);

      // Reading values from scan result
      for (Result result = scanner.next(); result != null; result = scanner.next())

      {
           //assign row values in variable Row
          String Row = Bytes.toString(result.getRow());

          //assign column username values in name
          String name = Bytes.toString(result.getValue("stats".getBytes(),"username".getBytes()));

          //assign column count_txn values in count
          String count = Bytes.toString(result.getValue("stats".getBytes(),"count_txn".getBytes()));

          System.out.println( Row + "," + name + "," + count );

       //closing the scanner
        scanner.close();

       }
}
}
```

Output: scan program shows the content of the TRANSACTIONS table



```
<terminated> scan_hbase_table [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (16-May
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
2018-05-16 14:14:35,811 WARN  [main] util.NativeCodeLoader (NativeCodeLoader.java:<clin
2018-05-16 14:14:36,057 INFO  [main] zookeeper.RecoverableZooKeeper (RecoverableZooKeep
101,Amitabh,2
102,Sharukh,1
104,Anubahv,1
105,Pawan,1
106,Aamir,1
107,Salman,1
108,Ranbir,1
```