# Case Study 2 :
# Customer Transaction

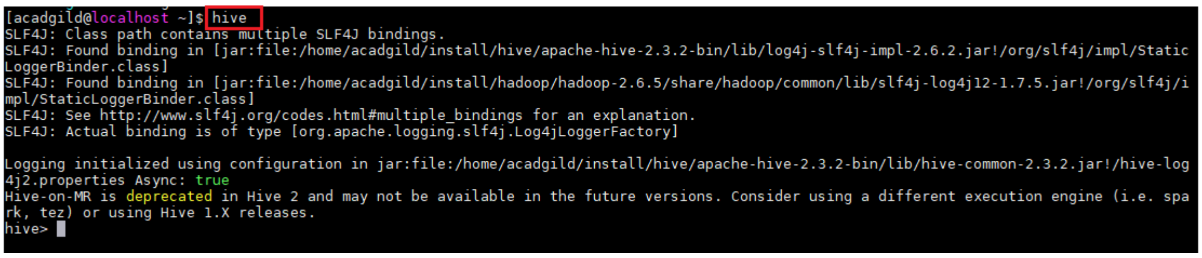
## <u>To-do</u>

**Case Study Description**

**Let us take up the CUSTOMER and TRANSACTIONS table we have created in the Let's Do Together section. Let us solve the following use cases using these tables :-**

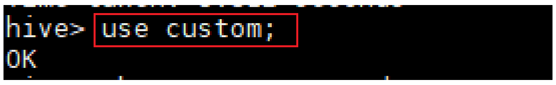
**1. Find out the number of transaction done by each customer (These should be take up in module 8 itself)**

We have started hive shell by using command : **hive**



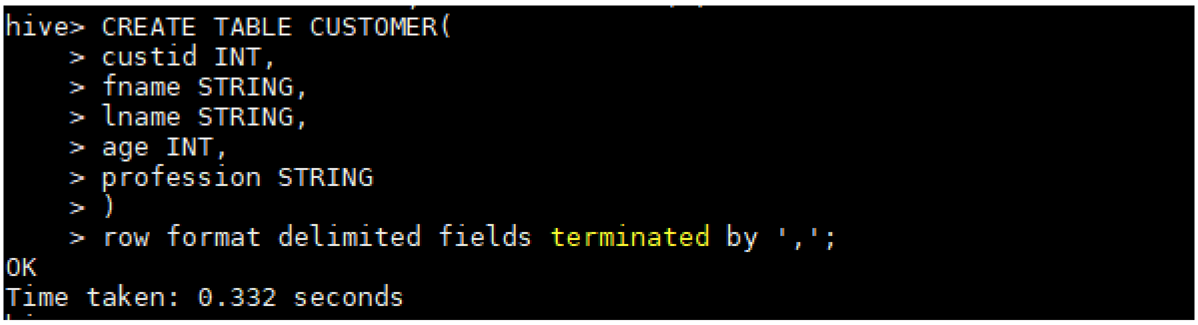
```
[acadgild@localhost ~]$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/acadgild/install/hive/apache-hive-2.3.2-bin/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/Static
LoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/i
mpl/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]

Logging initialized using configuration in jar:file:/home/acadgild/install/hive/apache-hive-2.3.2-bin/lib/hive-common-2.3.2.jar!/hive-log
4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spa
rk, tez) or using Hive 1.X releases.
hive>
```

Then we have used custom database to create tables by using command**:  use custom**



```
hive> use custom;
OK
```

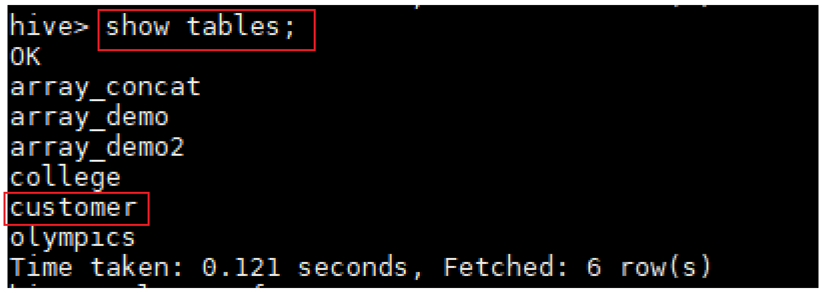
Then we have created table CUSTOMER  as shown below :



```
hive> CREATE TABLE CUSTOMER(
    > custid INT,
    > fname STRING,
    > lname STRING,
    > age INT,
    > profession STRING
    > )
    > row format delimited fields terminated by ',';
OK
Time taken: 0.332 seconds
```

# Case Study 2 :
# Customer Transaction

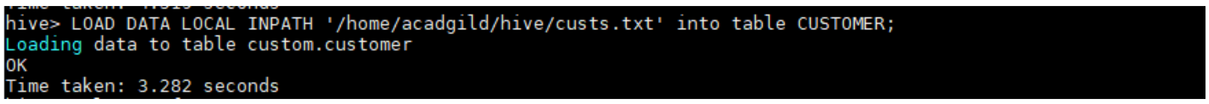
We could verify that CUSTOMER table has been created successfully by using command :
**SHOW TABLES**



```
hive> show tables;
OK
array_concat
array_demo
array_demo2
college
customer
olympics
Time taken: 0.121 seconds, Fetched: 6 row(s)
```

Then we have inserted data into customer table from local file custs.txt as shown below :

**LOAD DATA LOCAL INPATH '/home/acadgild/hive/custs.txt' into table CUSTOMER;**



```
hive> LOAD DATA LOCAL INPATH '/home/acadgild/hive/custs.txt' into table CUSTOMER;
Loading data to table custom.customer
OK
Time taken: 3.282 seconds
```

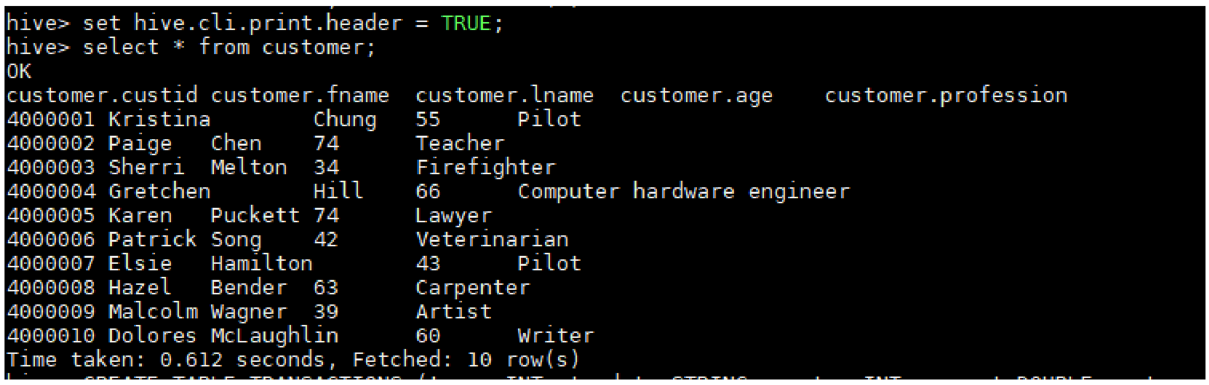
After inserting data into **customer** table, we are verifying that data is present in customer table by fetching rows by using query :

**select * from customer;**

You could see all 10 records in customer table.

Before that, we have set column header to TRUE so that we can have column headers along with output by using command :

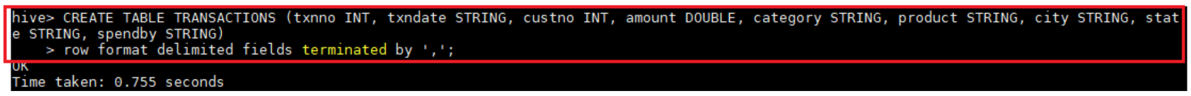
**set  hive.cli.print.header = TRUE;**



```
hive> set hive.cli.print.header = TRUE;
hive> select * from customer;
OK
customer.custid customer.fname  customer.lname  customer.age    customer.profession
4000001 Kristina        Chung   55      Pilot
4000002 Paige   Chen    74      Teacher
4000003 Sherri  Melton  34      Firefighter
4000004 Gretchen        Hill    66      Computer hardware engineer
4000005 Karen   Puckett 74      Lawyer
4000006 Patrick Song    42      Veterinarian
4000007 Elsie   Hamilton        43      Pilot
4000008 Hazel   Bender  63      Carpenter
4000009 Malcolm Wagner  39      Artist
4000010 Dolores McLaughlin      60      Writer
Time taken: 0.612 seconds, Fetched: 10 row(s)
```

# Case Study 2 :
# Customer Transaction

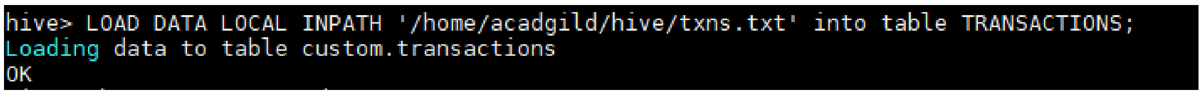
**Similarly we have created TRANSACTIONS table by using below query :**



```
hive> CREATE TABLE TRANSACTIONS (txnno INT, txndate STRING, custno INT, amount DOUBLE, category STRING, product STRING, city STRING, stat
e STRING, spendby STRING)
    > row format delimited fields terminated by ',';
OK
Time taken: 0.755 seconds
```

Then we have inserted data into **transactions** table from local file txns.txt as shown below :

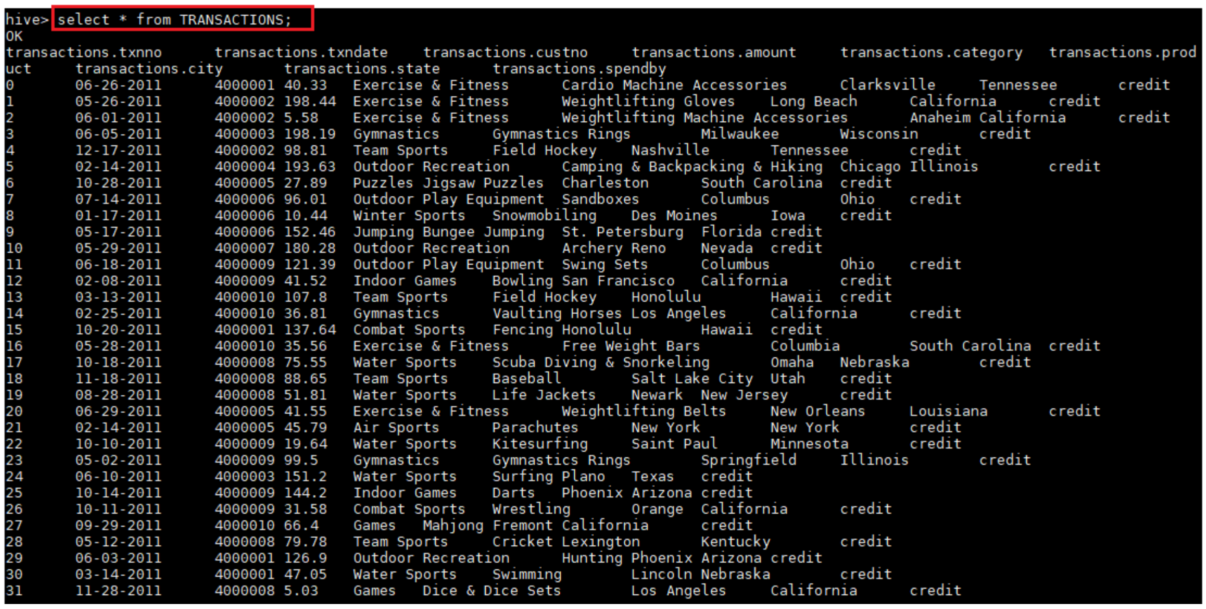
**LOAD DATA LOCAL INPATH '/home/acadgild/hive/txns.txt' into table TRANSACTIONS;**



```
hive> LOAD DATA LOCAL INPATH '/home/acadgild/hive/txns.txt' into table TRANSACTIONS;
Loading data to table custom.transactions
OK
```

After inserting data into **transactions** table, we are verifying that data is present in **transactions** table by fetching rows by using query :

**select * from transactions;**

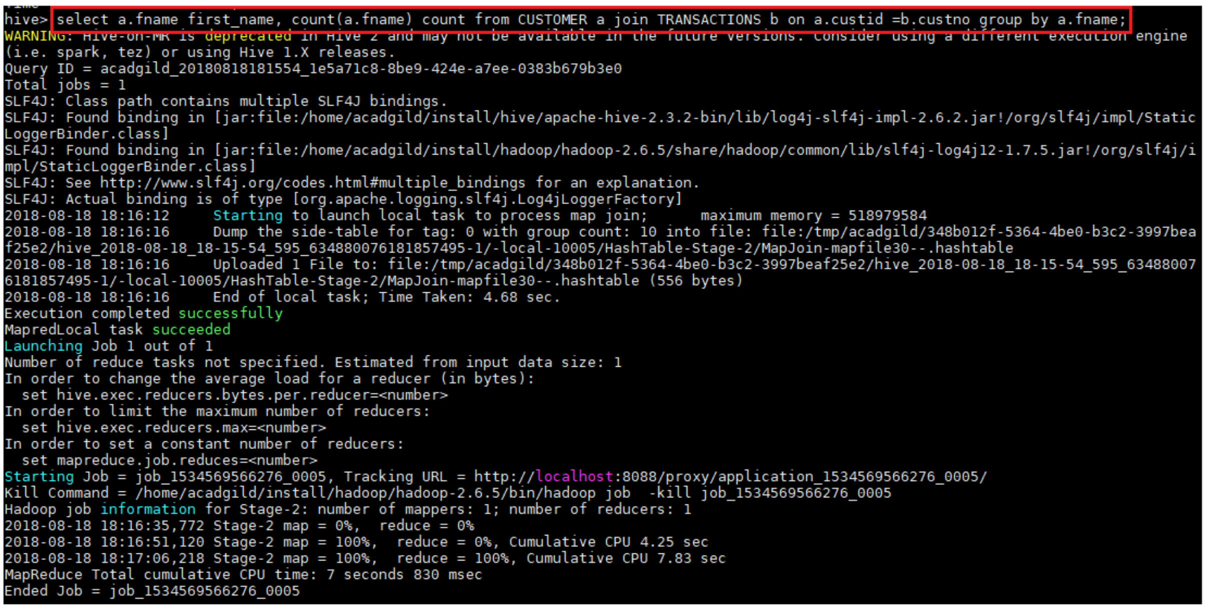


```
hive> select * from TRANSACTIONS;
OK
transactions.txnno      transactions.txndate    transactions.custno     transactions.amount     transactions.category   transactions.prod
uct     transactions.city       transactions.state      transactions.spendby
0       06-26-2011      4000001 40.33   Exercise & Fitness      Cardio Machine Accessories      Clarksville     Tennessee       credit
1       05-26-2011      4000002 198.44  Exercise & Fitness      Weightlifting Gloves    Long Beach      California      credit
2       06-01-2011      4000002 5.58    Exercise & Fitness      Weightlifting Machine Accessories       Anaheim California      credit
3       06-05-2011      4000003 198.19  Gymnastics      Gymnastics Rings        Milwaukee       Wisconsin       credit
4       12-17-2011      4000002 98.81   Team Sports     Field Hockey    Nashville       Tennessee       credit
5       02-14-2011      4000004 193.63  Outdoor Recreation      Camping & Backpacking & Hiking  Chicago Illinois        credit
6       10-28-2011      4000005 27.89   Puzzles Jigsaw Puzzles  Charleston      South Carolina  credit
7       07-14-2011      4000006 96.01   Outdoor Play Equipment  Sandboxes       Columbus        Ohio    credit
8       01-17-2011      4000006 10.44   Winter Sports   Snowmobiling    Des Moines      Iowa    credit
9       05-17-2011      4000006 152.46  Jumping Bungee Jumping  St. Petersburg  Florida credit
10      05-29-2011      4000007 180.28  Outdoor Recreation      Archery Reno    Nevada  credit
11      06-18-2011      4000009 121.39  Outdoor Play Equipment  Swing Sets      Columbus        Ohio    credit
12      02-08-2011      4000009 41.52   Indoor Games    Bowling San Francisco   California      credit
13      03-13-2011      4000010 107.8   Team Sports     Field Hockey    Honolulu        Hawaii  credit
14      02-25-2011      4000010 36.81   Gymnastics      Vaulting Horses Los Angeles     California      credit
15      10-20-2011      4000001 137.64  Combat Sports   Fencing Honolulu        Hawaii  credit
16      05-28-2011      4000010 35.56   Exercise & Fitness      Free Weight Bars        Columbia        South Carolina  credit
17      10-18-2011      4000008 75.55   Water Sports    Scuba Diving & Snorkeling       Omaha   Nebraska        credit
18      11-18-2011      4000008 88.65   Team Sports     Baseball        Salt Lake City  Utah    credit
19      08-28-2011      4000008 51.81   Water Sports    Life Jackets    Newark  New Jersey      credit
20      06-29-2011      4000005 41.55   Exercise & Fitness      Weightlifting Belts     New Orleans     Louisiana       credit
21      02-14-2011      4000005 45.79   Air Sports      Parachutes      New York        New York        credit
22      10-10-2011      4000009 19.64   Water Sports    Kitesurfing     Saint Paul      Minnesota       credit
23      05-02-2011      4000009 99.5    Gymnastics      Gymnastics Rings        Springfield     Illinois        credit
24      06-10-2011      4000003 151.2   Water Sports    Surfing Plano   Texas   credit
25      10-14-2011      4000009 144.2   Indoor Games    Darts   Phoenix Arizona credit
26      10-11-2011      4000009 31.58   Combat Sports   Wrestling       Orange  California      credit
27      09-29-2011      4000010 66.4    Games   Mahjong Fremont California      credit
28      05-12-2011      4000008 79.78   Team Sports     Cricket Lexington       Kentucky        credit
29      06-03-2011      4000001 126.9   Outdoor Recreation      Hunting Phoenix Arizona credit
30      03-14-2011      4000001 47.05   Water Sports    Swimming        Lincoln Nebraska        credit
31      11-28-2011      4000008 5.03    Games   Dice & Dice Sets        Los Angeles     California      credit
```

Then to find out the number of transaction done by each customer, we have used below query :
We have used JOIN with customer and transactions tables and grouped it by fname in customer.
Then we have fetched fname and count from this JOIN query.

**select a.fname first_name, count(a.fname) count from CUSTOMER a join TRANSACTIONS b on a.custid =b.custno group by a.fname;**

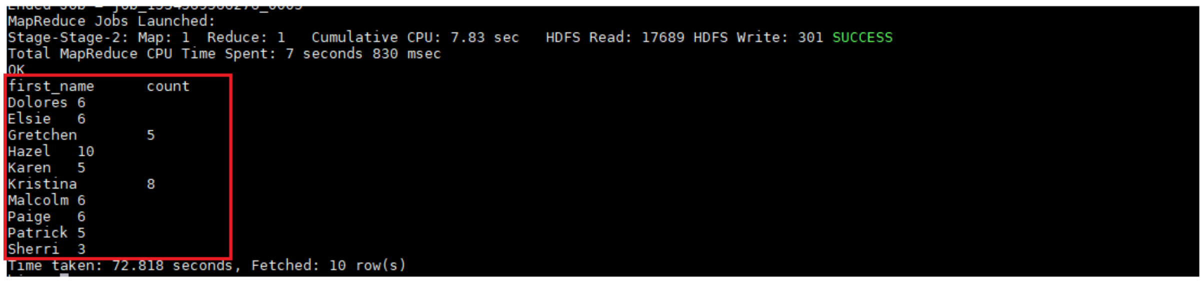
# Case Study 2 :
# Customer Transaction



```
hive> select a.fname first_name, count(a.fname) count from CUSTOMER a join TRANSACTIONS b on a.custid =b.custno group by a.fname;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine
(i.e. spark, tez) or using Hive 1.X releases.
Query ID = acadgild_20180818181554_1e5a71c8-8be9-424e-a7ee-0383b679b3e0
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/Static
LoggerBinder.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/i
mpl/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]
2018-08-18 18:16:12     Starting to launch local task to process map join;      maximum memory = 518979584
2018-08-18 18:16:16     Dump the side-table for tag: 0 with group count: 10 into file: file:/tmp/acadgild/348b012f-5364-4be0-b3c2-3997bea
f25e2/hive_2018-08-18_18-15-54_595_634880076181857495-1/-local-10005/HashTable-Stage-2/MapJoin-mapfile30--.hashtable
2018-08-18 18:16:16     Uploaded 1 File to: file:/tmp/acadgild/348b012f-5364-4be0-b3c2-3997beaf25e2/hive_2018-08-18_18-15-54_595_63488007
6181857495-1/-local-10005/HashTable-Stage-2/MapJoin-mapfile30--.hashtable (556 bytes)
2018-08-18 18:16:16     End of local task; Time Taken: 4.68 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_1534569566276_0005, Tracking URL = http://localhost:8088/proxy/application_1534569566276_0005/
Kill Command = /home/acadgild/install/hadoop/hadoop-2.6.5/bin/hadoop job  -kill job_1534569566276_0005
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2018-08-18 18:16:35,772 Stage-2 map = 0%,  reduce = 0%
2018-08-18 18:16:51,120 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 4.25 sec
2018-08-18 18:17:06,218 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 7.83 sec
MapReduce Total cumulative CPU time: 7 seconds 830 msec
Ended Job = job_1534569566276_0005
```

**So the Final output is :**

| **first_name** | **count** |
|---|---|
| Dolores | 6 |
| Elsie | 6 |
| Gretchen | 5 |
| Hazel | 10 |
| Karen | 5 |
| Kristina | 8 |
| Malcolm | 6 |
| Paige | 6 |
| Patrick | 5 |
| Sherri | 3 |



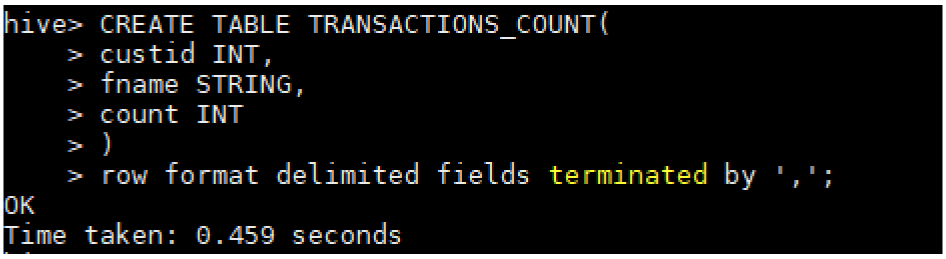
```
MapReduce Jobs Launched:
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 7.83 sec   HDFS Read: 17689 HDFS Write: 301 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 830 msec
OK
first_name      count
Dolores 6
Elsie   6
Gretchen        5
Hazel   10
Karen   5
Kristina        8
Malcolm 6
Paige   6
Patrick 5
Sherri  3
Time taken: 72.818 seconds, Fetched: 10 row(s)
```

# Case Study 2 :
# Customer Transaction

**2. Create a new table called TRANSACTIONS_COUNT. This table should have**

**3 fields - custid, fname and count. (Again to be done in module 8)**

We have created TRANSACTIONS_COUNT table with three columns : custid,fname and count by using below query :

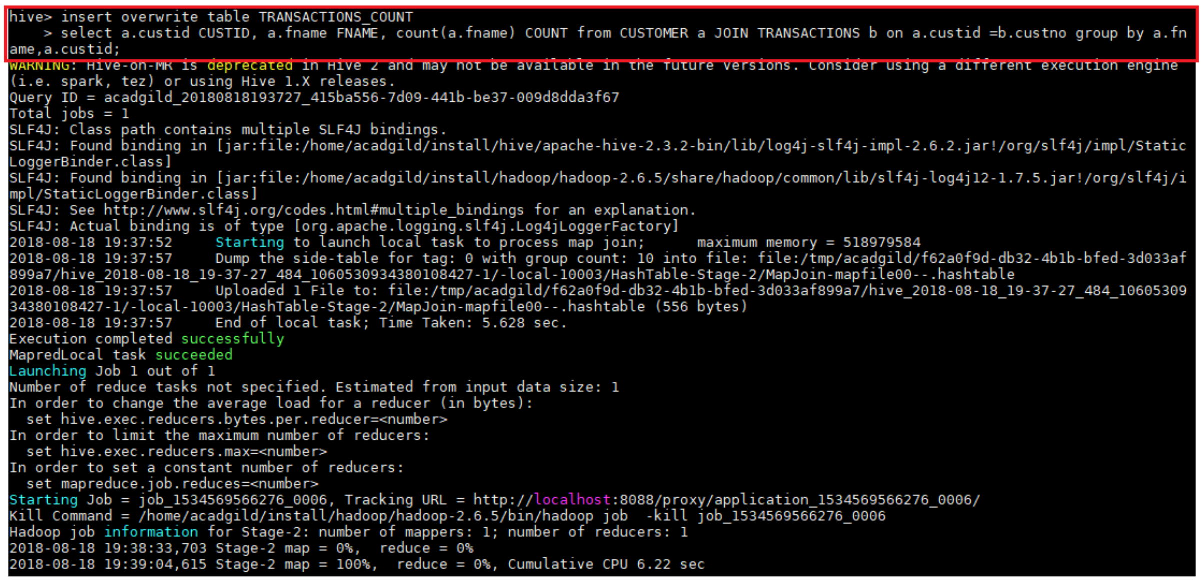


```
hive> CREATE TABLE TRANSACTIONS_COUNT(
    > custid INT,
    > fname STRING,
    > count INT
    > )
    > row format delimited fields terminated by ',';
OK
Time taken: 0.459 seconds
```

**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. (This has to be done in module 9).**

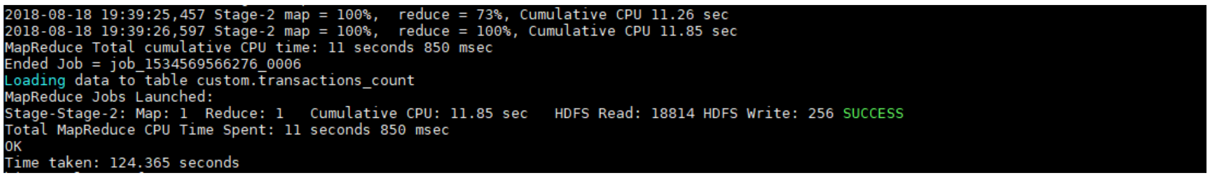
We have inserted data into TRANSACTIONS_COUNT table by using similar select query which we have used for query in question 1.

**insert overwrite table TRANSACTIONS_COUNT**
**select a.custid CUSTID, a.fname FNAME, count(a.fname) COUNT from CUSTOMER a JOIN TRANSACTIONS b on a.custid =b.custno group by a.fname,a.custid;**



```
hive> insert overwrite table TRANSACTIONS_COUNT
    > select a.custid CUSTID, a.fname FNAME, count(a.fname) COUNT from CUSTOMER a JOIN TRANSACTIONS b on a.custid =b.custno group by a.fn
ame,a.custid;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine
(i.e. spark, tez) or using Hive 1.X releases.
Query ID = acadgild_20180818193727_415ba556-7d09-441b-be37-009d8dda3f67
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/Static
LoggerBinder.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/i
mpl/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]
2018-08-18 19:37:52     Starting to launch local task to process map join;      maximum memory = 518979584
2018-08-18 19:37:57     Dump the side-table for tag: 0 with group count: 10 into file: file:/tmp/acadgild/f62a0f9d-db32-4b1b-bfed-3d033af
899a7/hive_2018-08-18_19-37-27_484_1060530934380108427-1/-local-10003/HashTable-Stage-2/MapJoin-mapfile00--.hashtable
2018-08-18 19:37:57     Uploaded 1 File to: file:/tmp/acadgild/f62a0f9d-db32-4b1b-bfed-3d033af899a7/hive_2018-08-18_19-37-27_484_10605309
34380108427-1/-local-10003/HashTable-Stage-2/MapJoin-mapfile00--.hashtable (556 bytes)
2018-08-18 19:37:57     End of local task; Time Taken: 5.628 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_1534569566276_0006, Tracking URL = http://localhost:8088/proxy/application_1534569566276_0006/
Kill Command = /home/acadgild/install/hadoop/hadoop-2.6.5/bin/hadoop job  -kill job_1534569566276_0006
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2018-08-18 19:38:33,703 Stage-2 map = 0%,  reduce = 0%
2018-08-18 19:39:04,615 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 6.22 sec
```

## Case Study 2 :
## Customer Transaction

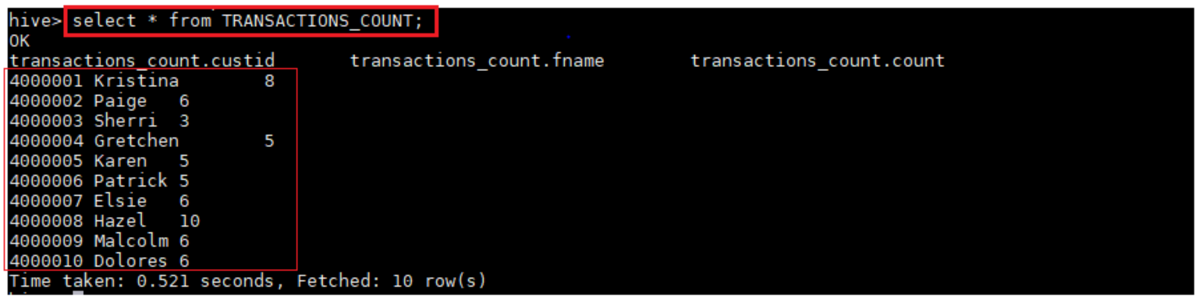


```
2018-08-18 19:39:25,457 Stage-2 map = 100%,  reduce = 73%, Cumulative CPU 11.26 sec
2018-08-18 19:39:26,597 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 11.85 sec
MapReduce Total cumulative CPU time: 11 seconds 850 msec
Ended Job = job_1534569566276_0006
Loading data to table custom.transactions_count
MapReduce Jobs Launched:
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 11.85 sec   HDFS Read: 18814 HDFS Write: 256 SUCCESS
Total MapReduce CPU Time Spent: 11 seconds 850 msec
OK
Time taken: 124.365 seconds
```

After inserting data into TRANSACTIONS_COUNT table, we are verifying that data is present in TRANSACTIONS_COUNT table by fetching rows by using select query :

**select * from TRANSACTIONS_COUNT;**

You could see all 10 records in customer table.



```
hive> select * from TRANSACTIONS_COUNT;
OK
transactions_count.custid       transactions_count.fname        transactions_count.count
4000001 Kristina        8
4000002 Paige   6
4000003 Sherri  3
4000004 Gretchen        5
4000005 Karen   5
4000006 Patrick 5
4000007 Elsie   6
4000008 Hazel   10
4000009 Malcolm 6
4000010 Dolores 6
Time taken: 0.521 seconds, Fetched: 10 row(s)
```

**4. Now lets make the TRANSACTIONS_COUNT table Hbase complaint. In the**

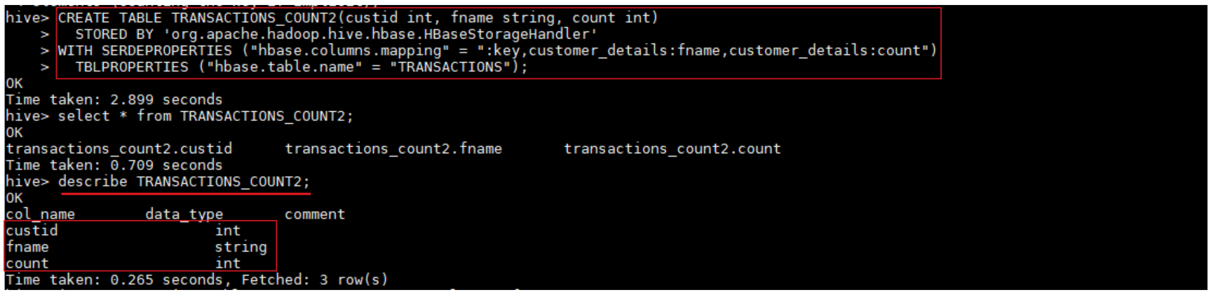
**sense, use Ser Des And Storate handler features of hive to change the**

**TRANSACTIONS_COUNT table to be able to create a TRANSACTIONS table**

**in Hbase. (This has to be done in module 10)**

So we have created a table TRANSACTIONS_COUNT2 similar to TRANSACTIONS_COUNT as HBase compliant as shown below :

**CREATE TABLE TRANSACTIONS_COUNT2 (custid int, fname string,count int)**
  **STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'**
**WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,customer_details:fname,customer_details:count")**
  **TBLPROPERTIES ("hbase.table.name" = "TRANSACTIONS");**



```
hive> CREATE TABLE TRANSACTIONS_COUNT2(custid int, fname string, count int)
    >   STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
    > WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,customer_details:fname,customer_details:count")
    >   TBLPROPERTIES ("hbase.table.name" = "TRANSACTIONS");
OK
Time taken: 2.899 seconds
hive> select * from TRANSACTIONS_COUNT2;
OK
transactions_count2.custid      transactions_count2.fname       transactions_count2.count
Time taken: 0.709 seconds
hive> describe TRANSACTIONS_COUNT2;
OK
col_name        data_type       comment
custid                  int
fname                   string
count                   int
Time taken: 0.265 seconds, Fetched: 3 row(s)
```

You could see that we have all three columns : custid, fname,count present in TRANSACTIONS_COUNT2 table as in TRANSACTIONS_COUNT.

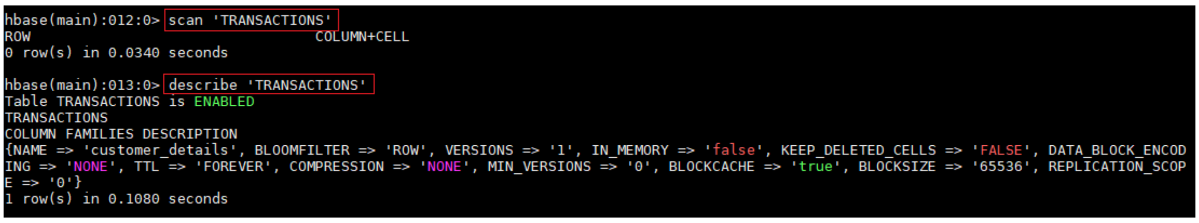
Here we have used **STORED BY** to make this Hive table as HBase compliant.

By using **TBLPROPERTIES**, we could create table TRANSCATIONS in HBase and by using **SERDEPROPERTIES**, we could map columns of TRANSACTIONS_COUNT2 table in Hive with columns of TRANSACTIONS in HBase.

Then we have connected to Hbase shell and verified that TRANSACTIONS table in HBase has been created by using commands :

**scan 'TRANSACTIONS'** which shows that table has been created successfully and

**describe 'TRANSACTIONS'** which shows column family is customer_details and other details.



```
hbase(main):012:0> scan 'TRANSACTIONS'
ROW                              COLUMN+CELL
0 row(s) in 0.0340 seconds

hbase(main):013:0> describe 'TRANSACTIONS'
Table TRANSACTIONS is ENABLED
TRANSACTIONS
COLUMN FAMILIES DESCRIPTION
{NAME => 'customer_details', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCOD
ING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOP
E => '0'}
1 row(s) in 0.1080 seconds
```

**5. Now insert the data in TRANSACTIONS_COUNT table using the query in step**

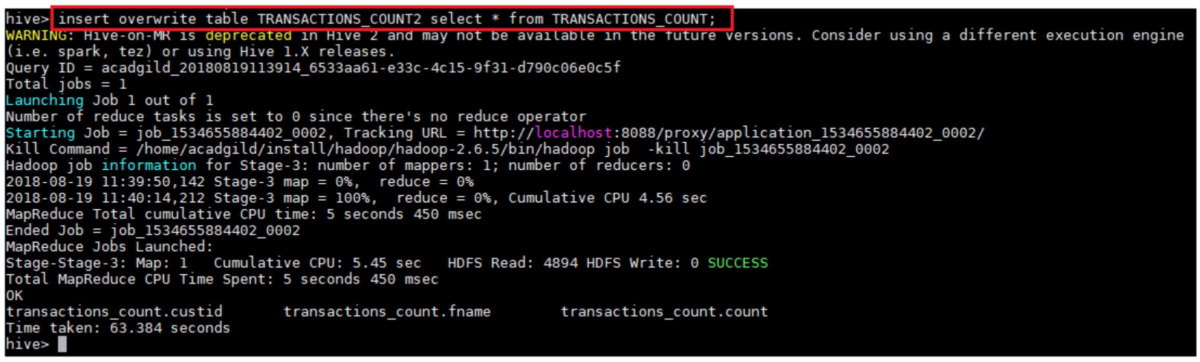
**3 again, this should populate the Hbase TRANSACTIONS table automatically**

**(This has to be done in module 10)**

Then to insert data into HBase table TRANSACTIONS, we have loaded data into TRANSACTIONS_COUNT2 from TRANSACTIONS_COUNT as

**insert overwrite table transactions_count2 select * from transactions_count;**

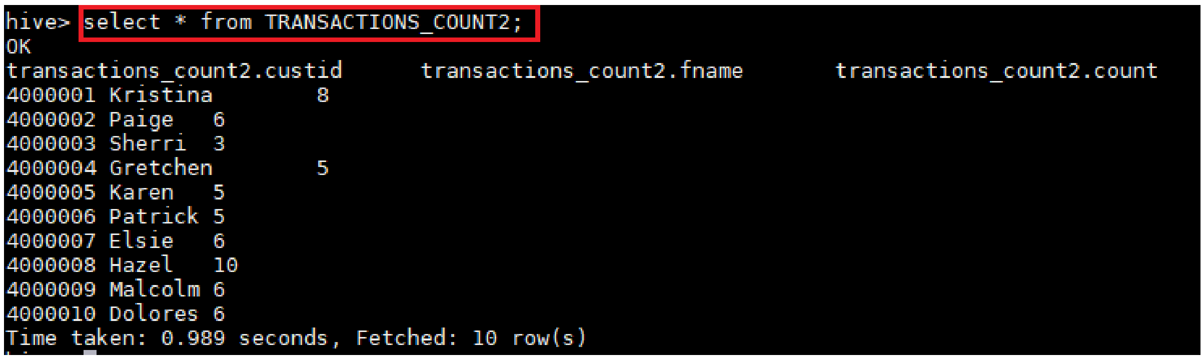
# Case Study 2 :
# Customer Transaction



```
hive> insert overwrite table TRANSACTIONS_COUNT2 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 different execution engine
(i.e. spark, tez) or using Hive 1.X releases.
Query ID = acadgild_20180819113914_6533aa61-e33c-4c15-9f31-d790c06e0c5f
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_1534655884402_0002, Tracking URL = http://localhost:8088/proxy/application_1534655884402_0002/
Kill Command = /home/acadgild/install/hadoop/hadoop-2.6.5/bin/hadoop job  -kill job_1534655884402_0002
Hadoop job information for Stage-3: number of mappers: 1; number of reducers: 0
2018-08-19 11:39:50,142 Stage-3 map = 0%,  reduce = 0%
2018-08-19 11:40:14,212 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 4.56 sec
MapReduce Total cumulative CPU time: 5 seconds 450 msec
Ended Job = job_1534655884402_0002
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1   Cumulative CPU: 5.45 sec   HDFS Read: 4894 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 450 msec
OK
transactions_count.custid       transactions_count.fname        transactions_count.count
Time taken: 63.384 seconds
hive>
```

We could verify that data has been inserted successfully into table TRANSCATIONS_COUNT2 by using below select query :

**select * from TRANSACTIONS_COUNT2;**

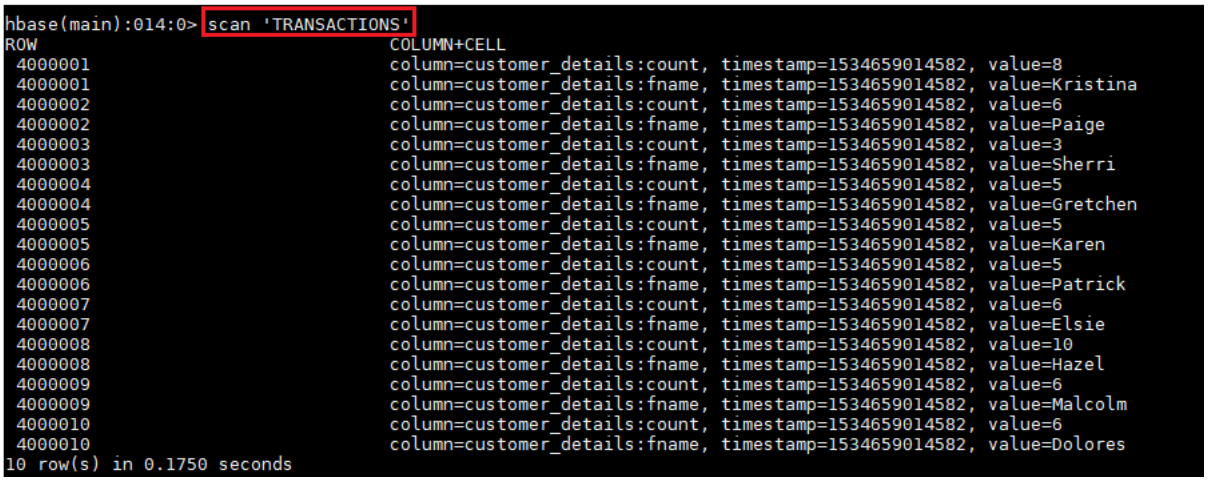


```
hive> select * from TRANSACTIONS_COUNT2;
OK
transactions_count2.custid      transactions_count2.fname       transactions_count2.count
4000001 Kristina        8
4000002 Paige   6
4000003 Sherri  3
4000004 Gretchen        5
4000005 Karen   5
4000006 Patrick 5
4000007 Elsie   6
4000008 Hazel   10
4000009 Malcolm 6
4000010 Dolores 6
Time taken: 0.989 seconds, Fetched: 10 row(s)
```

**After Inserting data into TRANSACTIONS_COUNT2,** we have verified that same data has been inserted into TRANSACTIONS table in HBase by using  command :

**scan 'TRANSACTIONS'**

We could see that all 10 rows have been inserted into **TRANSACTIONS** table successfully.



```
hbase(main):014:0> scan 'TRANSACTIONS'
ROW                             COLUMN+CELL
 4000001                        column=customer_details:count, timestamp=1534659014582, value=8
 4000001                        column=customer_details:fname, timestamp=1534659014582, value=Kristina
 4000002                        column=customer_details:count, timestamp=1534659014582, value=6
 4000002                        column=customer_details:fname, timestamp=1534659014582, value=Paige
 4000003                        column=customer_details:count, timestamp=1534659014582, value=3
 4000003                        column=customer_details:fname, timestamp=1534659014582, value=Sherri
 4000004                        column=customer_details:count, timestamp=1534659014582, value=5
 4000004                        column=customer_details:fname, timestamp=1534659014582, value=Gretchen
 4000005                        column=customer_details:count, timestamp=1534659014582, value=5
 4000005                        column=customer_details:fname, timestamp=1534659014582, value=Karen
 4000006                        column=customer_details:count, timestamp=1534659014582, value=5
 4000006                        column=customer_details:fname, timestamp=1534659014582, value=Patrick
 4000007                        column=customer_details:count, timestamp=1534659014582, value=6
 4000007                        column=customer_details:fname, timestamp=1534659014582, value=Elsie
 4000008                        column=customer_details:count, timestamp=1534659014582, value=10
 4000008                        column=customer_details:fname, timestamp=1534659014582, value=Hazel
 4000009                        column=customer_details:count, timestamp=1534659014582, value=6
 4000009                        column=customer_details:fname, timestamp=1534659014582, value=Malcolm
 4000010                        column=customer_details:count, timestamp=1534659014582, value=6
 4000010                        column=customer_details:fname, timestamp=1534659014582, value=Dolores
10 row(s) in 0.1750 seconds
```

## Case Study 2 :
## Customer Transaction

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

**the TRANSACTIONS table data from java level.**

We have written two JAVA API codes :

1. **ScanTableAdvanced.java** : This will scan entire HBase table TRANSACTIONS.
2. **GetOperations.java :** This will fetch data in TRANSACTIONS table for a particular rowkey.

1. Below is the java code of **ScanTableAdvanced** :

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
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 ScanTableAdvanced {
	public static void main(String[] args) throws IOException,
InterruptedException {
		Configuration conf = HBaseConfiguration.create();

		System.out.println("Creating HTable instance to 'TRANSACTIONS'...");
		HTable table = new HTable(conf, "TRANSACTIONS");

		System.out.println("Creating scan object to scan TRANSACTIONS
table...");
		Scan scan = new Scan();

		System.out.println("Scanner Caching at table level: " +
table.getScannerCaching());
		scan.setCaching(1);
		scan.setBatch(2);

		System.out.println("Scanner Caching at scan object level: " +
scan.getCaching());

		System.out.println("Getting a result scanner object...");
		ResultScanner rs = table.getScanner(scan);

		for (Result r : rs) {
			System.out.println("Result: " + r);
		}

		System.out.println("Closing Scanner instance...");
		rs.close();
	}
}
```

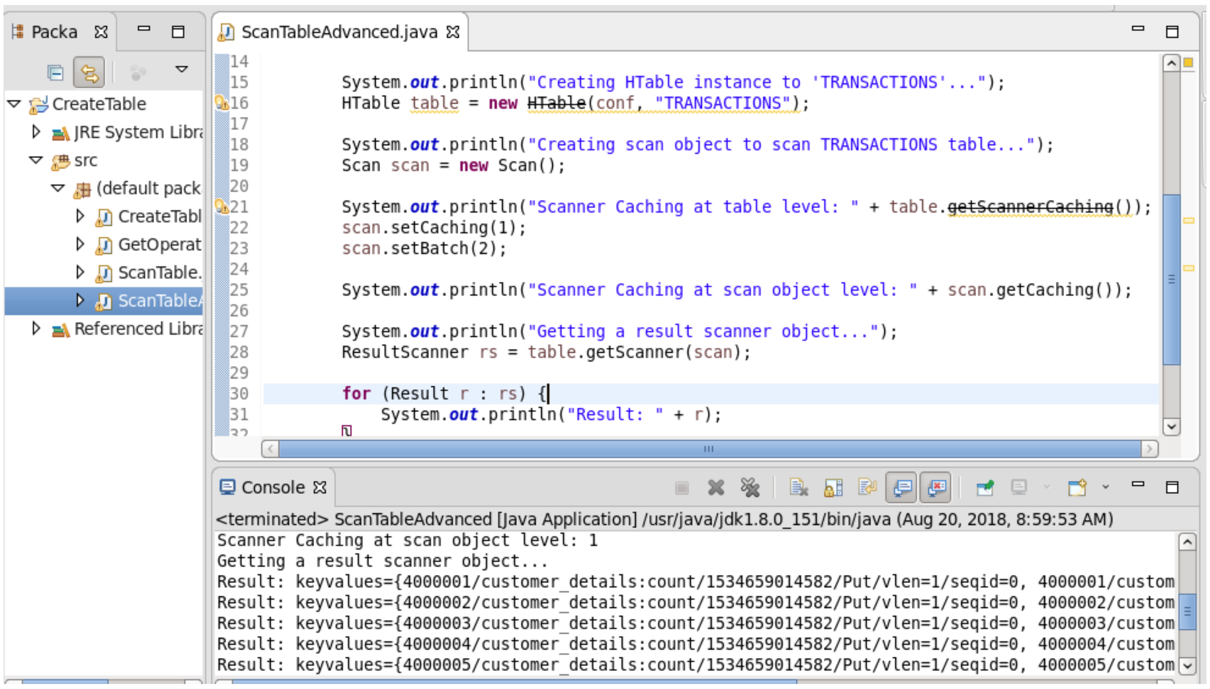
**Case Study 2 :**
**Customer Transaction**

**Below is the output at Console :**

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/acadgild/install/spark/spark-
2.2.1-bin-hadoop2.7/jars/slf4j-log4j12-
1.7.16.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/acadgild/install/hbase/hbase-
1.2.6/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.slf4j.impl.Log4jLoggerFactory]
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.
Creating HTable instance to 'TRANSACTIONS'...
Creating scan object to scan TRANSACTIONS table...
Scanner Caching at table level: 2147483647
Scanner Caching at scan object level: 1
Getting a result scanner object...
Result:
keyvalues={4000001/customer_details:count/1534659014582/Put/vlen=1/seqi
d=0, 4000001/customer_details:fname/1534659014582/Put/vlen=8/seqid=0}
Result:
keyvalues={4000002/customer_details:count/1534659014582/Put/vlen=1/seqi
d=0, 4000002/customer_details:fname/1534659014582/Put/vlen=5/seqid=0}
Result:
keyvalues={4000003/customer_details:count/1534659014582/Put/vlen=1/seqi
d=0, 4000003/customer_details:fname/1534659014582/Put/vlen=6/seqid=0}
Result:
keyvalues={4000004/customer_details:count/1534659014582/Put/vlen=1/seqi
d=0, 4000004/customer_details:fname/1534659014582/Put/vlen=8/seqid=0}
Result:
keyvalues={4000005/customer_details:count/1534659014582/Put/vlen=1/seqi
d=0, 4000005/customer_details:fname/1534659014582/Put/vlen=5/seqid=0}
Result:
keyvalues={4000006/customer_details:count/1534659014582/Put/vlen=1/seqi
d=0, 4000006/customer_details:fname/1534659014582/Put/vlen=7/seqid=0}
Result:
keyvalues={4000007/customer_details:count/1534659014582/Put/vlen=1/seqi
d=0, 4000007/customer_details:fname/1534659014582/Put/vlen=5/seqid=0}
Result:
keyvalues={4000008/customer_details:count/1534659014582/Put/vlen=2/seqi
d=0, 4000008/customer_details:fname/1534659014582/Put/vlen=5/seqid=0}
Result:
keyvalues={4000009/customer_details:count/1534659014582/Put/vlen=1/seqi
d=0, 4000009/customer_details:fname/1534659014582/Put/vlen=7/seqid=0}
Result:
keyvalues={4000010/customer_details:count/1534659014582/Put/vlen=1/seqi
d=0, 4000010/customer_details:fname/1534659014582/Put/vlen=7/seqid=0}
Closing Scanner instance...
```

# Case Study 2 :
# Customer Transaction

**Below is the screenshot of ScanTableAdvanced.java code and it's console :**



```java

        System.out.println("Creating HTable instance to 'TRANSACTIONS'...");
        HTable table = new ~~HTable~~(conf, "TRANSACTIONS");

        System.out.println("Creating scan object to scan TRANSACTIONS table...");
        Scan scan = new Scan();

        System.out.println("Scanner Caching at table level: " + table.~~getScannerCaching~~());
        scan.setCaching(1);
        scan.setBatch(2);

        System.out.println("Scanner Caching at scan object level: " + scan.getCaching());

        System.out.println("Getting a result scanner object...");
        ResultScanner rs = table.getScanner(scan);

        for (Result r : rs) {
            System.out.println("Result: " + r);
```

```
<terminated> ScanTableAdvanced [Java Application] /usr/java/jdk1.8.0_151/bin/java (Aug 20, 2018, 8:59:53 AM)
Scanner Caching at scan object level: 1
Getting a result scanner object...
Result: keyvalues={4000001/customer_details:count/1534659014582/Put/vlen=1/seqid=0, 4000001/custom
Result: keyvalues={4000002/customer_details:count/1534659014582/Put/vlen=1/seqid=0, 4000002/custom
Result: keyvalues={4000003/customer_details:count/1534659014582/Put/vlen=1/seqid=0, 4000003/custom
Result: keyvalues={4000004/customer_details:count/1534659014582/Put/vlen=1/seqid=0, 4000004/custom
Result: keyvalues={4000005/customer_details:count/1534659014582/Put/vlen=1/seqid=0, 4000005/custom
```

This above java code will scan entire TRANSACTIONS table. Hence you could see in the output that all 10 rowkeys from 4000001 to 4000010 with their respective count and fname column values are present.

# Case Study 2 :
# Customer Transaction

2. Below is the java code of **getOperations** :

```java
import java.io.IOException;
import java.util.Map;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.KeyValue;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.hbase.client.Result;

public class GetOperations {

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

		Configuration conf = HBaseConfiguration.create();

		System.out.println("Creating HTable instance to 'TRANSACTIONS'...");
		HTable table = new HTable(conf, "TRANSACTIONS");

		System.out.println("Creating get object to retrieve the row with key 
p1...");
		Get get = new Get(Bytes.toBytes("4000008"));
		System.out.println("Default Time Range of get object: " + 
get.getTimeRange());
		System.out.println("Will Block Cache be scanned: " + 
get.getCacheBlocks());
		System.out.println("Default versions to fetch: " + 
get.getMaxVersions());

		System.out.println("Setting versions to fetch to be 3");
		get.setMaxVersions(3);
		System.out.println("Versions to fetch: " + get.getMaxVersions());

		Result result = table.get(get);
		System.out.println("Result fetched: " + result);

		System.out.println("Fetching the most recent givenName...");
		String givenName = 
Bytes.toString(result.getValue(Bytes.toBytes("customer_details"), 
Bytes.toBytes("fname")));
		System.out.println("Given Name retrieved: " + givenName);

		System.out.println("Scanning across all the values of column 
personal:givenName...");

		for (KeyValue kv : 
result.getColumn(Bytes.toBytes("customer_details"), Bytes.toBytes("fname"))) {
			System.out.println("Timestamp: " + kv.getTimestamp() + "\t" + 
"Value of giveName: " + Bytes.toString(kv.getValue()));
		}

		System.out.println("Scanning across all the values of key p1...");
```

```
            for (KeyValue kv : result.raw()) {
                  System.out.println("Row Key: " + Bytes.toString(kv.getRow()) +
", Column Family: " + Bytes.toString(kv.getFamily()) +
                              ", Column Name: " +
Bytes.toString(kv.getQualifier()) + ", Value: " + Bytes.toString(kv.getValue()) +
", Timestamp: " + kv.getTimestamp() );
            }

            Map resultMap = result.getMap();
            for (Object key : resultMap.keySet()) {
                  System.out.println("Key: " + Bytes.toString((byte[]) key));
            }

            System.out.println(resultMap.get(Bytes.toBytes("TRANSACTIONS")));
            System.out.println(resultMap.values());
            System.out.println(result.getNoVersionMap());

      }
}
```

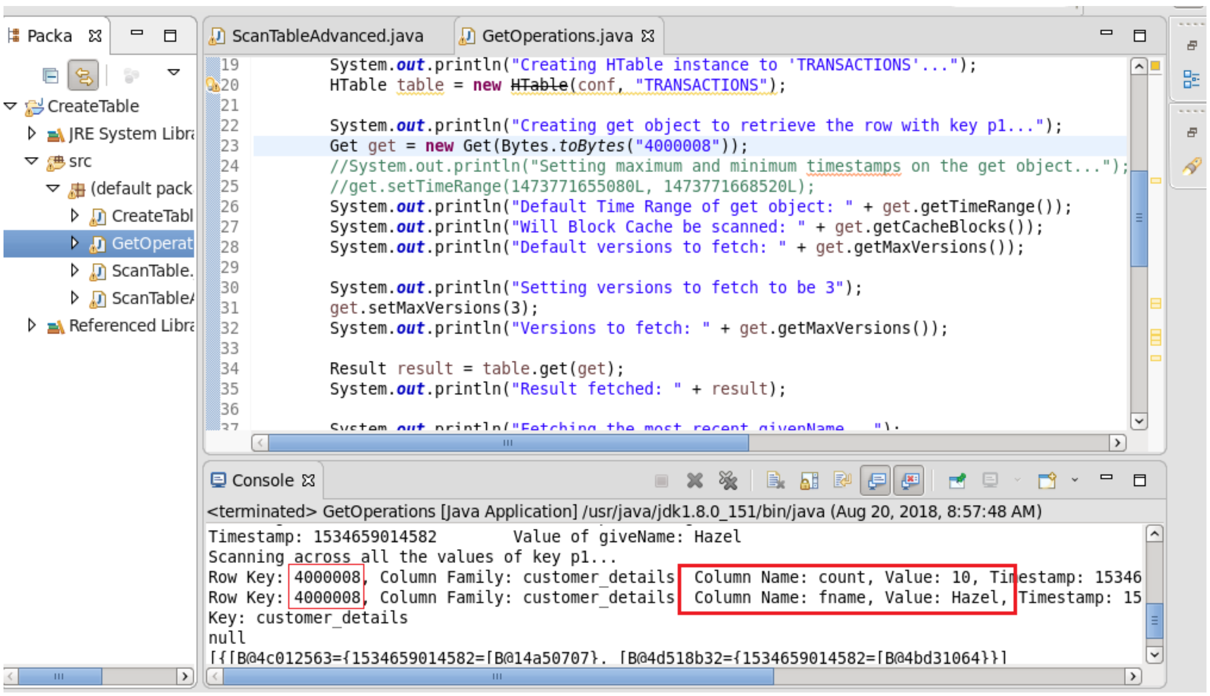
**Below is the screenshot of getOperations.java code and it's console :**

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/acadgild/install/spark/spark-
2.2.1-bin-hadoop2.7/jars/slf4j-log4j12-
1.7.16.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/acadgild/install/hbase/hbase-
1.2.6/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.slf4j.impl.Log4jLoggerFactory]
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.
Creating HTable instance to 'TRANSACTIONS'...
Creating get object to retrieve the row with key p1...
Default Time Range of get object: maxStamp=9223372036854775807,
minStamp=0
Will Block Cache be scanned: true
Default versions to fetch: 1
Setting versions to fetch to be 3
Versions to fetch: 3
Result fetched:
keyvalues={4000008/customer_details:count/1534659014582/Put/vlen=2/seqi
d=0, 4000008/customer_details:fname/1534659014582/Put/vlen=5/seqid=0}
Fetching the most recent givenName...
Given Name retrieved: Hazel
Scanning across all the values of column personal:givenName...
Timestamp: 1534659014582     Value of giveName: Hazel
```

# Case Study 2 :
# Customer Transaction

```
Scanning across all the values of key p1...
Row Key: 4000008, Column Family: customer_details, Column Name: count,
Value: 10, Timestamp: 1534659014582
Row Key: 4000008, Column Family: customer_details, Column Name: fname,
Value: Hazel, Timestamp: 1534659014582
Key: customer_details
null
[{[B@4c012563={1534659014582=[B@14a50707},
[B@4d518b32={1534659014582=[B@4bd31064}}]
{[B@e3c0e40={[B@4c012563=[B@14a50707, [B@4d518b32=[B@4bd31064}}
```

**Below is the screenshot of GetOperations.java code and it's console :**



```java
        System.out.println("Creating HTable instance to 'TRANSACTIONS'...");
        HTable table = new ~~HTable~~(conf, "TRANSACTIONS");

        System.out.println("Creating get object to retrieve the row with key p1...");
        Get get = new Get(Bytes.toBytes("4000008"));
        //System.out.println("Setting maximum and minimum timestamps on the get object...");
        //get.setTimeRange(1473771655080L, 1473771668520L);
        System.out.println("Default Time Range of get object: " + get.getTimeRange());
        System.out.println("Will Block Cache be scanned: " + get.getCacheBlocks());
        System.out.println("Default versions to fetch: " + get.getMaxVersions());

        System.out.println("Setting versions to fetch to be 3");
        get.setMaxVersions(3);
        System.out.println("Versions to fetch: " + get.getMaxVersions());

        Result result = table.get(get);
        System.out.println("Result fetched: " + result);
```

```
<terminated> GetOperations [Java Application] /usr/java/jdk1.8.0_151/bin/java (Aug 20, 2018, 8:57:48 AM)
Timestamp: 1534659014582        Value of giveName: Hazel
Scanning across all the values of key p1...
Row Key: 4000008, Column Family: customer_details  Column Name: count, Value: 10, Timestamp: 15346
Row Key: 4000008, Column Family: customer_details  Column Name: fname, Value: Hazel, Timestamp: 15
Key: customer_details
null
[{[B@4c012563={1534659014582=[B@14a50707}, [B@4d518b32={1534659014582=[B@4bd31064}}]
```

This above java code will fetch TRANSACTIONS table only having rowkey **4000008** as we have mentioned in our code to fetch data with rowkey as 4000008. Hence you could see in the output that count value is 10 and fname value is Hazel.