Big Data And Hadoop

Session 16 – Assignment 3

Problem Statement:

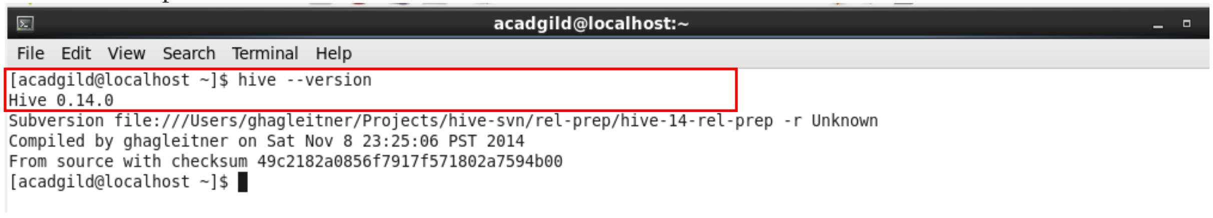
Link: https://acadgild.com/blog/transactions-in-hive/

Refer the above given link for transactions in Hive and implement the operations given in the blog using your own sample data set and send us the screenshot.

Solution:

1. **Row Level Transactions:**

Transactions are provided at the row-level in Hive 0.14. So we verify the version of hive installed using the **version** option as follows:

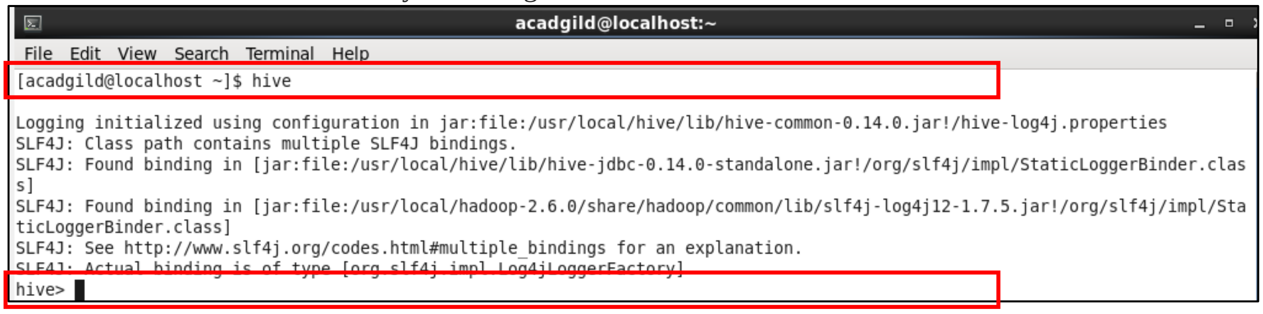


```
[acadgild@localhost ~]$ hive --version
Hive 0.14.0
Subversion file:///Users/ghagleitner/Projects/hive-svn/rel-prep/hive-14-rel-prep -r Unknown
Compiled by ghagleitner on Sat Nov 8 23:25:06 PST 2014
From source with checksum 49c2182a0856f7917f571802a7594b00
[acadgild@localhost ~]$
```

The different row-level transactions available in Hive 0.14 are as follows:

1. Insert
2. Delete
3. Update

We start the hive command line by executing the command hive as shown below:



```
[acadgild@localhost ~]$ hive

Logging initialized using configuration in jar:file:/usr/local/hive/lib/hive-common-0.14.0.jar!/hive-log4j.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hive/lib/hive-jdbc-0.14.0-standalone.jar!/org/slf4j/impl/StaticLoggerBinder.clas
s]
SLF4J: Found binding in [jar:file:/usr/local/hadoop-2.6.0/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/Sta
ticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
hive>
```

The above snapshot also shows that hive prompt has started. A pre-requisite to use hive is to start mysql server. This was done using the command sudo service mysqld start.

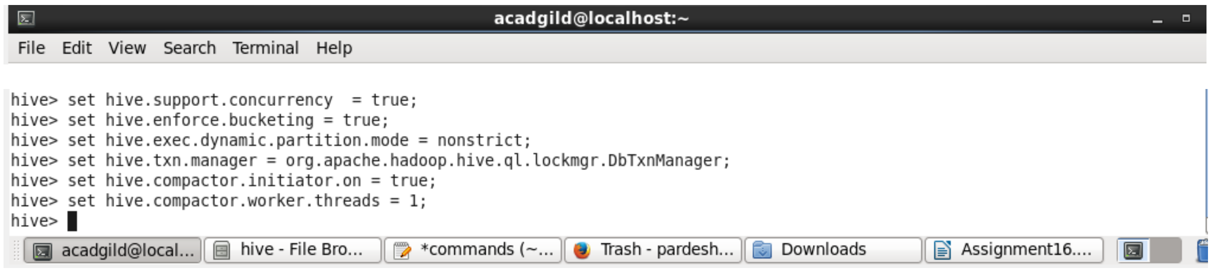
Before creating a Hive table that supports transactions, the transaction features present in Hive needs to be turned on, as by default they are turned off.

The below properties needs to be set appropriately in ***hive shell***, order-wise to work with transactions in Hive:

- Hive.support.concurrency = true;
- Hive.enforce.bucketing = true;
- Hive.exec.dynamic.partition.mode = nonstrict;
- Hive.txn.manager = org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;
- Hive.compactor.initiator.on = true;

- Hive.compactor.worker.threads = 1;

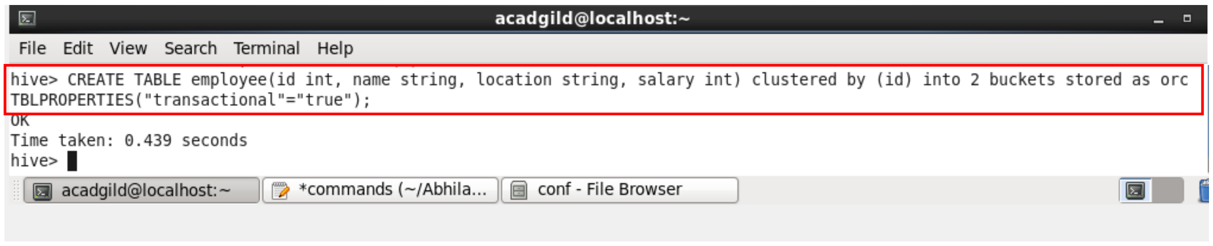
These properties as set as follows:



```
hive> set hive.support.concurrency  = true;
hive> set hive.enforce.bucketing = true;
hive> set hive.exec.dynamic.partition.mode = nonstrict;
hive> set hive.txn.manager = org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;
hive> set hive.compactor.initiator.on = true;
hive> set hive.compactor.worker.threads = 1;
hive>
```

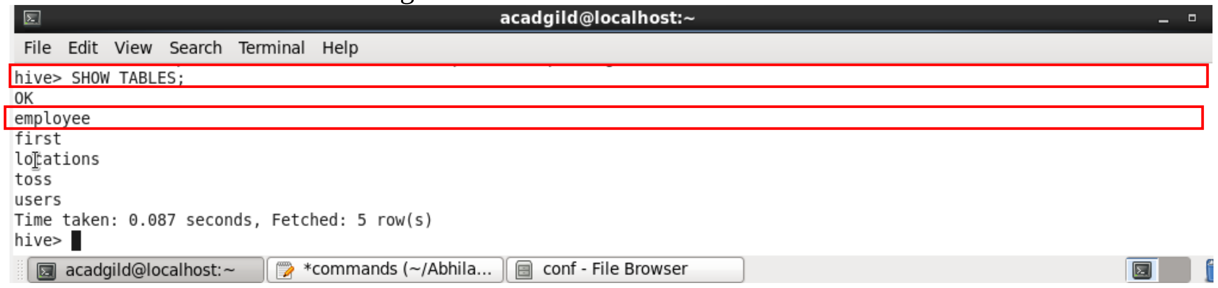
2. **Create Table that supports hive transactions:**
   We create a table with name ‘employee’ and the columns present in the table are *id, name, location, salary*. We are *bucketing* the table by ‘id’ and the table format is ‘*orc’,* also we are enabling the transactions in the table by specifying it inside
   the *TBLPROPERTIES* as *‘transactional’=’true’.*
   The query used is

CREATE TABLE employee(id int, name string, location string, salary int) clustered by (id) into 2 buckets stored as orc TBLPROPERTIES(“transactional”= “true”);



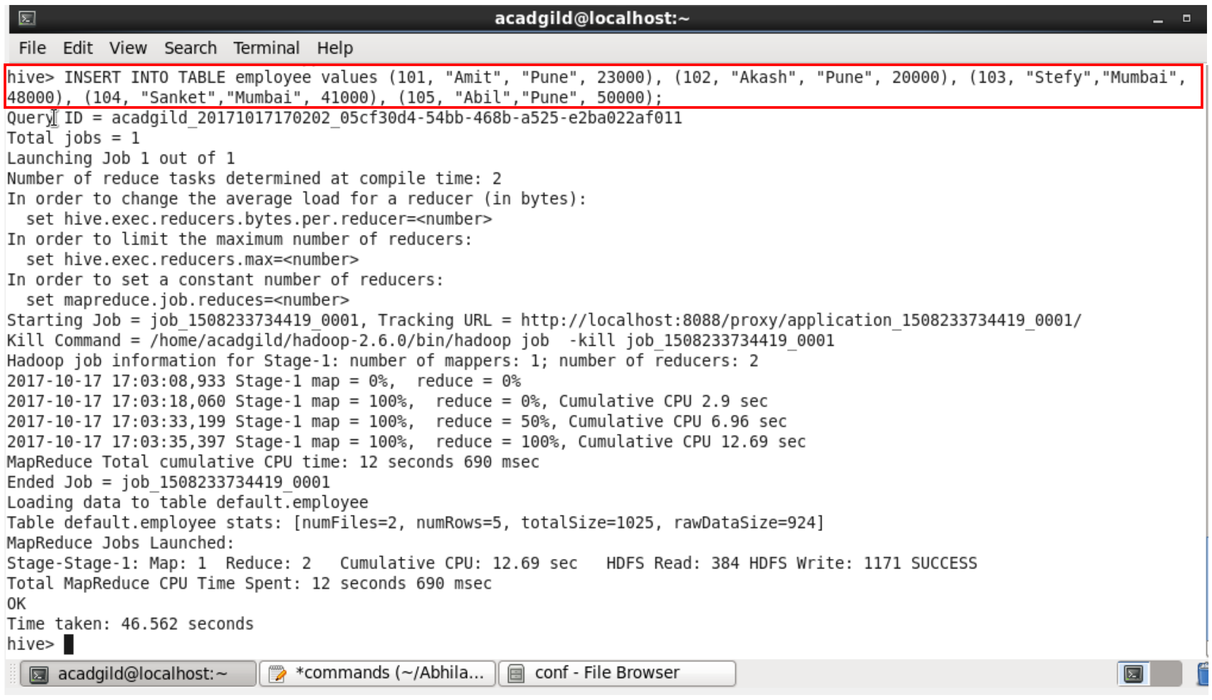
```
hive> CREATE TABLE employee(id int, name string, location string, salary int) clustered by (id) into 2 buckets stored as orc
TBLPROPERTIES("transactional"="true");
OK
Time taken: 0.439 seconds
hive>
```

We can see the table created using the command **SHOW TABLES:**



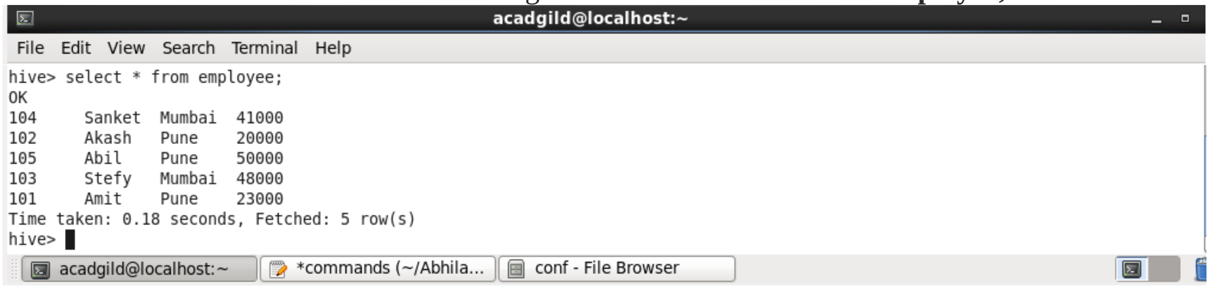
```
hive> SHOW TABLES;
OK
employee
first
locations
toss
users
Time taken: 0.087 seconds, Fetched: 5 row(s)
hive>
```

3. **Inserting data into hive table:**
   The command is used to insert row wise data into the Hive table is
   INSERT INTO TABLE employee values (101, “Amit”, “Pune”, 23000), (102, “Akash”, “ Pune”, 20000), (103, “Stefy”, “ Mumbai”, 48000), (104, “Sanket”,” Mumbai”, 41000), (1 05, “Abil”, “Pune”, 50000);
   Here, each row is seperated by ‘*( )’ brackets.*



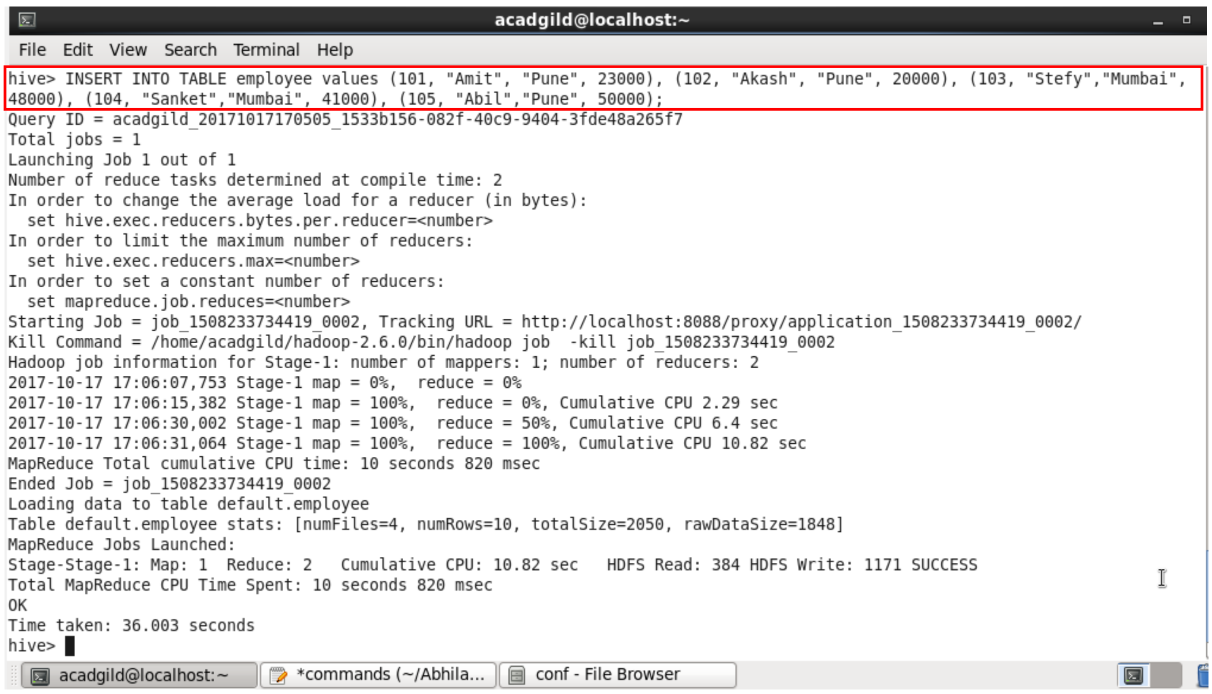
```
hive> INSERT INTO TABLE employee values (101, "Amit", "Pune", 23000), (102, "Akash", "Pune", 20000), (103, "Stefy","Mumbai",
48000), (104, "Sanket","Mumbai", 41000), (105, "Abil","Pune", 50000);
Query ID = acadgild_20171017170202_05cf30d4-54bb-468b-a525-e2ba022af011
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 2
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_1508233734419_0001, Tracking URL = http://localhost:8088/proxy/application_1508233734419_0001/
Kill Command = /home/acadgild/hadoop-2.6.0/bin/hadoop job  -kill job_1508233734419_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 2
2017-10-17 17:03:08,933 Stage-1 map = 0%,  reduce = 0%
2017-10-17 17:03:18,060 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.9 sec
2017-10-17 17:03:33,199 Stage-1 map = 100%,  reduce = 50%, Cumulative CPU 6.96 sec
2017-10-17 17:03:35,397 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 12.69 sec
MapReduce Total cumulative CPU time: 12 seconds 690 msec
Ended Job = job_1508233734419_0001
Loading data to table default.employee
Table default.employee stats: [numFiles=2, numRows=5, totalSize=1025, rawDataSize=924]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 2   Cumulative CPU: 12.69 sec   HDFS Read: 384 HDFS Write: 1171 SUCCESS
Total MapReduce CPU Time Spent: 12 seconds 690 msec
OK
Time taken: 46.562 seconds
hive>
```

The contents of the table can be viewed using the command **select * from employee;**



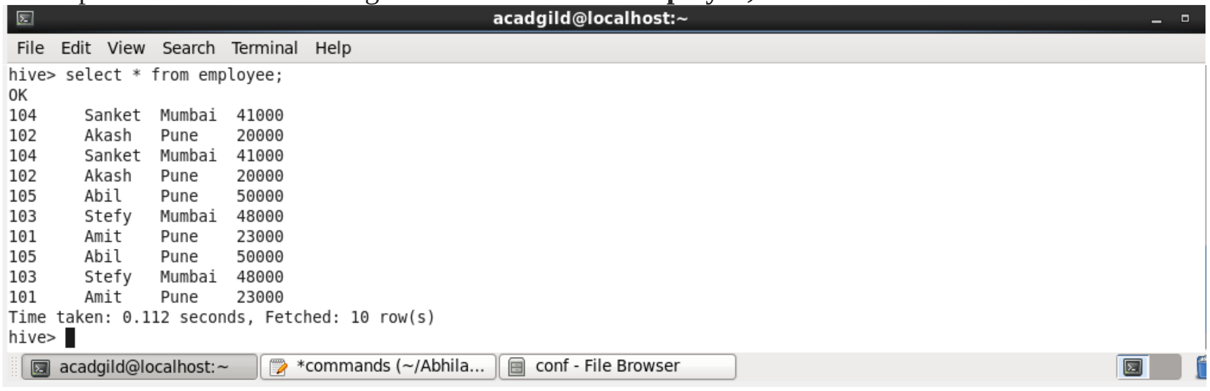
```
hive> select * from employee;
OK
104     Sanket  Mumbai  41000
102     Akash   Pune    20000
105     Abil    Pune    50000
103     Stefy   Mumbai  48000
101     Amit    Pune    23000
Time taken: 0.18 seconds, Fetched: 5 row(s)
hive>
```

Now if we try to re-insert the same data again, it will be appended to the previous data as shown below:



```
hive> INSERT INTO TABLE employee values (101, "Amit", "Pune", 23000), (102, "Akash", "Pune", 20000), (103, "Stefy","Mumbai",
48000), (104, "Sanket","Mumbai", 41000), (105, "Abil","Pune", 50000);
Query ID = acadgild_20171017170505_1533b156-082f-40c9-9404-3fde48a265f7
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 2
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_1508233734419_0002, Tracking URL = http://localhost:8088/proxy/application_1508233734419_0002/
Kill Command = /home/acadgild/hadoop-2.6.0/bin/hadoop job  -kill job_1508233734419_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 2
2017-10-17 17:06:07,753 Stage-1 map = 0%,  reduce = 0%
2017-10-17 17:06:15,382 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.29 sec
2017-10-17 17:06:30,002 Stage-1 map = 100%,  reduce = 50%, Cumulative CPU 6.4 sec
2017-10-17 17:06:31,064 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 10.82 sec
MapReduce Total cumulative CPU time: 10 seconds 820 msec
Ended Job = job_1508233734419_0002
Loading data to table default.employee
Table default.employee stats: [numFiles=4, numRows=10, totalSize=2050, rawDataSize=1848]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 2   Cumulative CPU: 10.82 sec   HDFS Read: 384 HDFS Write: 1171 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 820 msec
OK
Time taken: 36.003 seconds
hive>
```

The repeated records seen using **SELECT * FROM employee;**

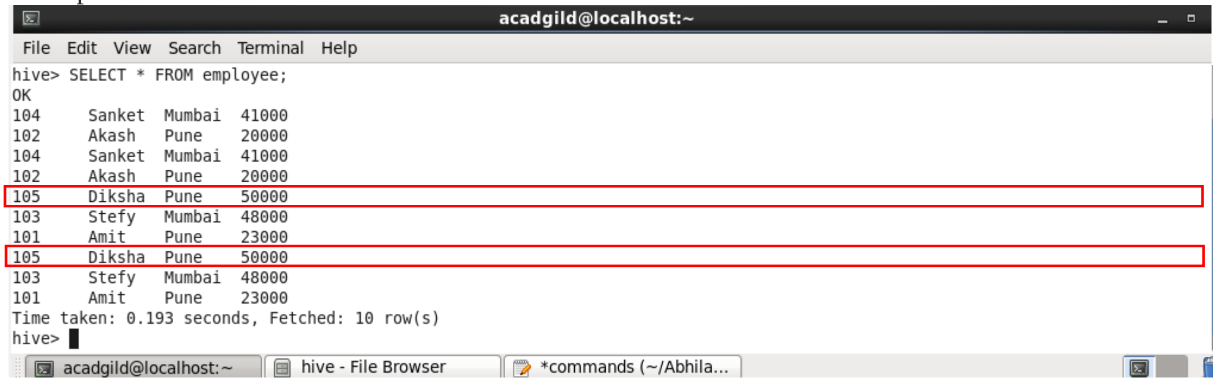


```
hive> select * from employee;
OK
104     Sanket  Mumbai  41000
102     Akash   Pune    20000
104     Sanket  Mumbai  41000
102     Akash   Pune    20000
105     Abil    Pune    50000
103     Stefy   Mumbai  48000
101     Amit    Pune    23000
105     Abil    Pune    50000
103     Stefy   Mumbai  48000
101     Amit    Pune    23000
Time taken: 0.112 seconds, Fetched: 10 row(s)
hive>
```

4. **Updating the data in the hive table:**
   We can update the records using the update command as follows:
   UPDATE employee SET name = ‘Diksha’ WHERE id =’105’;

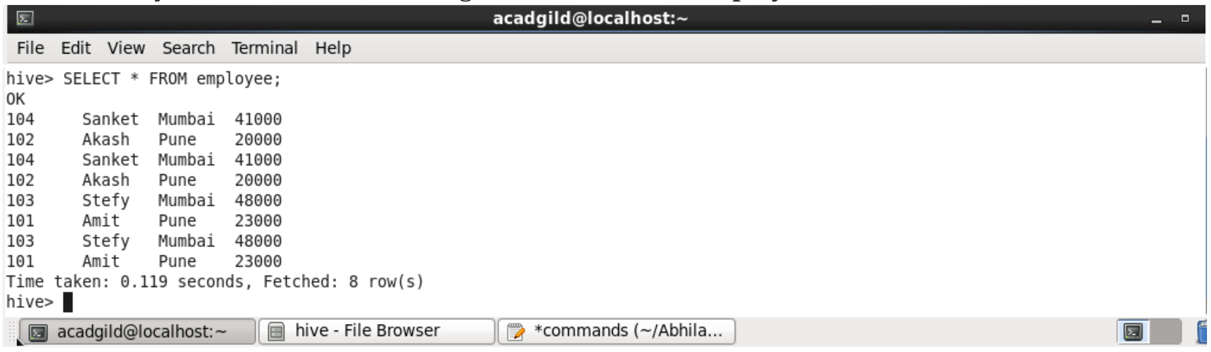
   If we try to update the values of column used for bucketing, it throws an exception, this means that the Update command is not supported on the columns that are bucketed.

The updated data can be seen as follows:



```
hive> SELECT * FROM employee;
OK
104     Sanket  Mumbai  41000
102     Akash   Pune    20000
104     Sanket  Mumbai  41000
102     Akash   Pune    20000
105     Diksha  Pune    50000
103     Stefy   Mumbai  48000
101     Amit    Pune    23000
105     Diksha  Pune    50000
103     Stefy   Mumbai  48000
101     Amit    Pune    23000
Time taken: 0.193 seconds, Fetched: 10 row(s)
hive>
```

5. **Deleting row from hive table:**
   The command used to delete the row is as follows
   DELETE from employee where id = 105;
   This command will delete the record which satisfies the condition **id = 105.**
   We can verify the deletion of data using **SELECT * from employee.**



```
hive> SELECT * FROM employee;
OK
104     Sanket  Mumbai  41000
102     Akash   Pune    20000
104     Sanket  Mumbai  41000
102     Akash   Pune    20000
103     Stefy   Mumbai  48000
101     Amit    Pune    23000
103     Stefy   Mumbai  48000
101     Amit    Pune    23000
Time taken: 0.119 seconds, Fetched: 8 row(s)
hive>
```

From the output, we observe that the records with id 105 have been eliminated.