Task 1.1:

```
hive> create table olympics_table
    > (
    > athlete_name string,
    > age int,
    > country string,
    > year int,
    > closing_date string,
    > sport_name string,
    > no_of_gold_medals int,
    > no_of_silver_medals int,
    > no_of_bronze_medals int,
    > total_medals int
    > )
    > row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
    > with serdeproperties("separatorChar" = "\t")
    > stored as textfile;
OK
Time taken: 2.859 seconds
hive> load data local inpath '/home/acadgild/olympix_data.csv' overwrite into table olympics_table;
Loading data to table default.olympics_table
OK
Time taken: 3.689 seconds
```

```
hive> select * from olympics_table limit 10;
OK
Michael Phelps  23      United States   2008    08-24-08        Swimming        8       0       0       8       NUL
Michael Phelps  19      United States   2004    08-29-04        Swimming        6       0       2       8
Michael Phelps  27      United States   2012    08-12-12        Swimming        4       2       0       6
Natalie Coughlin        25      United States   2008    08-24-08        Swimming        1       2       3       1
Aleksey Nemov   24      Russia  2000    10-01-00        Gymnastics      2       1       3       6
Alicia Coutts   24      Australia       2012    08-12-12        Swimming        1       3       1       5
Missy Franklin  17      United States   2012    08-12-12        Swimming        4       0       1       5
Ryan Lochte     27      United States   2012    08-12-12        Swimming        2       2       1       5
Allison Schmitt 22      United States   2012    08-12-12        Swimming        3       1       1       5
Natalie Coughlin        21      United States   2004    08-29-04        Swimming        2       2       1       5
Time taken: 0.597 seconds, Fetched: 10 row(s)
hive>
```

1. Write a Hive program to find the number of medals won by each country in swimming.

```
hive> select country,sum(total_medals) from olympics_table where sport_name='Swimming' group by country;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execu
tion engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = acadgild_20180905085033_50272800-3570-4682-9d8f-47c4c3839438
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_1535993336114_0028, Tracking URL = http://localhost:8088/proxy/application_1535993336114_0028/
Kill Command = /home/acadgild/hadoop-2.7.2/bin/hadoop job  -kill job_1535993336114_0028
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-09-05 08:51:13,837 Stage-1 map = 0%,  reduce = 0%
2018-09-05 08:51:43,120 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 7.04 sec
2018-09-05 08:52:09,862 Stage-1 map = 100%,  reduce = 68%, Cumulative CPU 11.27 sec
2018-09-05 08:52:11,477 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 12.11 sec
MapReduce Total cumulative CPU time: 12 seconds 110 msec
Ended Job = job_1535993336114_0028
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 12.11 sec   HDFS Read: 529391 HDFS Write: 949 SUCCESS
Total MapReduce CPU Time Spent: 12 seconds 110 msec
OK
Argentina       1.0
Australia       163.0
Austria 3.0
Belarus 2.0
Brazil  8.0
```

```
OK
Argentina       1.0
Australia       163.0
Austria 3.0
Belarus 2.0
Brazil  8.0
Canada  5.0
China   35.0
Costa Rica      2.0
Croatia 1.0
Denmark 1.0
France  39.0
Germany 32.0
Great Britain   11.0
Hungary 9.0
Italy   16.0
Japan   43.0
Lithuania       1.0
Netherlands     46.0
Norway  2.0
Poland  3.0
Romania 6.0
Russia  20.0
Serbia  1.0
Slovakia        2.0
Slovenia        1.0
South Africa    11.0
South Korea     4.0
Spain   3.0
Sweden  9.0
Trinidad and Tobago     1.0
Tunisia 3.0
Ukraine 7.0
United States   254.0
Zimbabwe        7.0
Time taken: 100.71 seconds, Fetched: 34 row(s)
```

2. Write a Hive program to find the number of medals that India won year wise.

```
hive> select year,sum(total_medals) from olympics_table where country='India' group by year;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execu
tion engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = acadgild_20180905085454_8c3da847-1de2-4633-9af3-0d603d318ca9
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_1535993336114_0029, Tracking URL = http://localhost:8088/proxy/application_1535993336114_0029/
Kill Command = /home/acadgild/hadoop-2.7.2/bin/hadoop job  -kill job_1535993336114_0029
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-09-05 08:55:34,069 Stage-1 map = 0%,  reduce = 0%
2018-09-05 08:56:04,690 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 7.05 sec
2018-09-05 08:56:31,360 Stage-1 map = 100%,  reduce = 83%, Cumulative CPU 11.36 sec
2018-09-05 08:56:32,719 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 12.01 sec
MapReduce Total cumulative CPU time: 12 seconds 10 msec
Ended Job = job_1535993336114_0029
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 12.01 sec   HDFS Read: 529370 HDFS Write: 171 SUCCESS
Total MapReduce CPU Time Spent: 12 seconds 10 msec
OK
2000    1.0
2004    1.0
2008    3.0
2012    6.0
Time taken: 100.861 seconds, Fetched: 4 row(s)
hive>
```

3. Write a Hive Program to find the total number of medals each country won.

```
hive> select country,sum(total_medals) from olympics_table group by country;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execu
tion engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = acadgild_20180905085910_f1b5e7de-6728-485e-9ec3-12624a1388d4
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_1535993336114_0030, Tracking URL = http://localhost:8088/proxy/application_1535993336114_0030/
Kill Command = /home/acadgild/hadoop-2.7.2/bin/hadoop job  -kill job_1535993336114_0030
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-09-05 08:59:50,815 Stage-1 map = 0%,  reduce = 0%
2018-09-05 09:00:17,089 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.97 sec
2018-09-05 09:00:43,747 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 9.12 sec
MapReduce Total cumulative CPU time: 9 seconds 120 msec
Ended Job = job_1535993336114_0030
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 9.63 sec   HDFS Read: 528571 HDFS Write: 2982 SUCCESS
Total MapReduce CPU Time Spent: 9 seconds 630 msec
OK
2008    NULL
Afghanistan     2.0
Algeria 8.0
Argentina       141.0
Armenia 10.0
Australia       609.0
Austria 91.0
Azerbaijan      25.0
Bahamas 24.0
Bahrain 1.0
Barbados        1.0
Belarus 97.0
```

```
Barbados        1.0
Belarus 97.0
Belgium 18.0
Botswana        1.0
Brazil  221.0
Bulgaria        41.0
Cameroon        20.0
Canada  370.0
Chile   22.0
China   530.0
Chinese Taipei  20.0
Colombia        13.0
Costa Rica      2.0
Croatia 81.0
Cuba    188.0
Cyprus  1.0
Czech Republic  81.0
Denmark 89.0
Dominican Republic      5.0
Ecuador 1.0
Egypt   8.0
Eritrea 1.0
Estonia 18.0
Ethiopia        29.0
Finland 118.0
France  318.0
Gabon   1.0
Georgia 23.0
Germany 629.0
Great Britain   322.0
Greece  59.0
Grenada 1.0
Guatemala       1.0
Hong Kong       3.0
Hungary 145.0
Iceland 15.0
India   11.0
```

```
India   11.0
Indonesia       22.0
Iran    24.0
Ireland 9.0
Israel  4.0
Italy   331.0
Jamaica 80.0
Japan   282.0
Kazakhstan      42.0
Kenya   39.0
Kuwait  2.0
Kyrgyzstan      3.0
Latvia  17.0
Lithuania       30.0
Macedonia       1.0
Malaysia        3.0
Mauritius       1.0
Mexico  38.0
Moldova 5.0
Mongolia        10.0
Montenegro      14.0
Morocco 11.0
Mozambique      1.0
Netherlands     318.0
New Zealand     52.0
Nigeria 39.0
North Korea     21.0
Norway  192.0
Panama  1.0
Paraguay        17.0
Poland  80.0
Portugal        9.0
Puerto Rico     2.0
Qatar   3.0
Romania 123.0
Russia  768.0
Saudi Arabia    6.0
```

```
Portugal        9.0
Puerto Rico     2.0
Qatar   3.0
Romania 123.0
Russia  768.0
Saudi Arabia    6.0
Serbia  31.0
Serbia and Montenegro   38.0
Singapore       7.0
Slovakia        35.0
Slovenia        25.0
South Africa    25.0
South Korea     308.0
Spain   205.0
Sri Lanka       1.0
Sudan   1.0
Sweden  181.0
Switzerland     93.0
Syria   1.0
Tajikistan      3.0
Thailand        18.0
Togo    1.0
Trinidad and Tobago     19.0
Tunisia 4.0
Turkey  28.0
Uganda  1.0
Ukraine 143.0
United Arab Emirates    1.0
United States   1299.0
Uruguay 1.0
Uzbekistan      19.0
Venezuela       4.0
Vietnam 2.0
Zimbabwe        7.0
Time taken: 96.174 seconds, Fetched: 111 row(s)
hive>
```

4. Write a Hive program to find the number of gold medals each country won.

```
hive> select country,sum(no_of_gold_medals) from olympics_table group by country;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execu
tion engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = acadgild_20180905090609_0d47ba31-a71e-4acf-985d-670b978d92e4
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_1535993336114_0031, Tracking URL = http://localhost:8088/proxy/application_1535993336114_0031/
Kill Command = /home/acadgild/hadoop-2.7.2/bin/hadoop job  -kill job_1535993336114_0031
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-09-05 09:06:49,976 Stage-1 map = 0%,  reduce = 0%
2018-09-05 09:07:15,858 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.66 sec
2018-09-05 09:07:43,184 Stage-1 map = 100%,  reduce = 67%, Cumulative CPU 8.77 sec
2018-09-05 09:07:44,864 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 9.83 sec
MapReduce Total cumulative CPU time: 9 seconds 830 msec
Ended Job = job_1535993336114_0031
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 9.83 sec   HDFS Read: 528589 HDFS Write: 2944 SUCCESS
Total MapReduce CPU Time Spent: 9 seconds 830 msec
OK
2008    0.0
Afghanistan     0.0
Algeria 2.0
Argentina       49.0
Armenia 0.0
Australia       163.0
Austria 36.0
Azerbaijan      6.0
Bahamas 11.0
Bahrain 0.0
```

```
Bahrain 0.0
Barbados        0.0
Belarus 17.0
Belgium 2.0
Botswana        0.0
Brazil  46.0
Bulgaria        8.0
Cameroon        20.0
Canada  168.0
Chile   3.0
China   234.0
Chinese Taipei  2.0
Colombia        2.0
Costa Rica      0.0
Croatia 35.0
Cuba    57.0
Cyprus  0.0
Czech Republic  14.0
Denmark 46.0
Dominican Republic      3.0
Ecuador 0.0
Egypt   1.0
Eritrea 0.0
Estonia 6.0
Ethiopia        13.0
Finland 11.0
France  108.0
Gabon   0.0
Georgia 6.0
Germany 223.0
Great Britain   124.0
Greece  12.0
Grenada 1.0
Guatemala       0.0
Hong Kong       0.0
Hungary 77.0
Iceland 0.0
```

```
India   1.0
Indonesia       5.0
Iran    10.0
Ireland 1.0
Israel  1.0
Italy   86.0
Jamaica 24.0
Japan   57.0
Kazakhstan      13.0
Kenya   11.0
Kuwait  0.0
Kyrgyzstan      0.0
Latvia  3.0
Lithuania       5.0
Macedonia       0.0
Malaysia        0.0
Mauritius       0.0
Mexico  19.0
Moldova 0.0
Mongolia        2.0
Montenegro      0.0
Morocco 2.0
Mozambique      1.0
Netherlands     101.0
New Zealand     18.0
Nigeria 6.0
North Korea     6.0
Norway  97.0
Panama  1.0
Paraguay        0.0
Poland  20.0
Portugal        1.0
Puerto Rico     0.0
Qatar   0.0
Romania 57.0
Russia  234.0
Saudi Arabia    0.0
```

```
Qatar   0.0
Romania 57.0
Russia  234.0
Saudi Arabia    0.0
Serbia  1.0
Serbia and Montenegro   11.0
Singapore       0.0
Slovakia        10.0
Slovenia        5.0
South Africa    10.0
South Korea     110.0
Spain   19.0
Sri Lanka       0.0
Sudan   0.0
Sweden  57.0
Switzerland     21.0
Syria   0.0
Tajikistan      0.0
Thailand        6.0
Togo    0.0
Trinidad and Tobago     1.0
Tunisia 2.0
Turkey  9.0
Uganda  1.0
Ukraine 31.0
United Arab Emirates    1.0
United States   544.0
Uruguay 0.0
Uzbekistan      5.0
Venezuela       1.0
Vietnam 0.0
Zimbabwe        2.0
Time taken: 97.649 seconds, Fetched: 111 row(s)
```

Task 1.2:

ConcatWs.java

```
1  package udf;
2
3  import org.apache.hadoop.hive.ql.exec.UDF;
4  import java.util.ArrayList;
5
6  public class ConcatWs extends UDF {
7      public String evaluate(String ch, ArrayList<String> input
8          if (input == null) {
9              return null;
10         }
11         return String.join(ch, input);
12     }
13 }
14
```

Then I have created a jar file from the above program with the name **ConcatWS.jar**

Now I have created the emp_skills.txt file which contains the following information:

```
File  Edit  View  Search  Terminal  Help
1       pradeep hadoop,pig,hive,sqoop,hbase
2       naresh  hadoop,pig,hive,hbase
3       Raj     hadoop,scala,spark
~
~
~
"emp skills.txt" 3L, 94C                                    3,1           All
```

Now I have created a table to store the details of employee with the following command:

- Create table emp
- (
- eIdint,
- eName String,
- eSkillsarray<String>
-)

Row format delimited
Fields terminated by '\t'
Collection items terminated by ',';

Then I have loaded the data into the emp table with the following command:

- LOAD DATA LOCAL INPATH '/home/acadgild/Pradeep/assignment5/emp_skills.txt'
- Into table emp;

Then I have verified whether the data is loaded as per expectations with the following command:

- Select * from emp;

```
hive> create table emp
    > (
    > eId int,
    > eName String,
    > eskills String
    > )
    > row format delimited
    > fields terminated by '\t'
    > collection items terminated by ',';
OK
Time taken: 3.336 seconds
hive> LOAD DATA LOCAL INPATH '/home/acadgild/Pradeep/assignment5/emp_skills.txt'

    > into table emp;
Loading data to table assignment5_db.emp
OK
Time taken: 4.069 seconds
hive> select * from emp;
OK
1       pradeep hadoop,pig,hive,sqoop,hbase
2       naresh  hadoop,pig,hive,hbase
3       Raj     hadoop,scala,spark
Time taken: 1.183 seconds, Fetched: 3 row(s)
hive>
```

Then I have registered this jar to existing hive session with the following command:

- ADD JAR ConcatWS.jar;

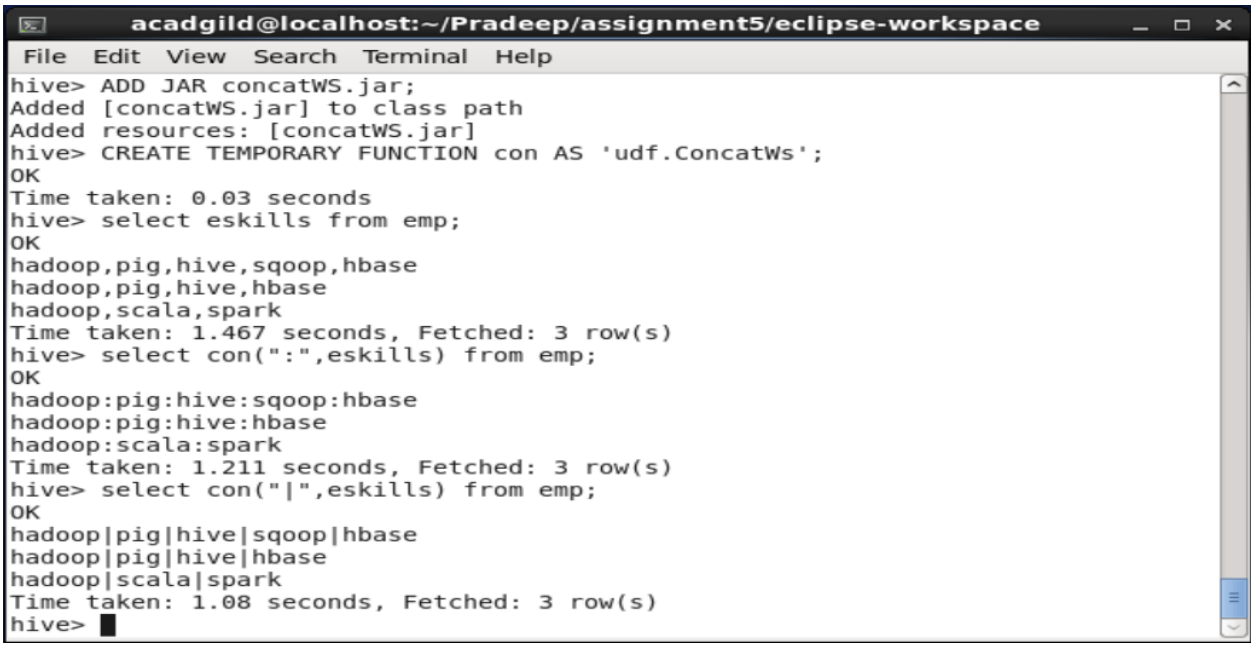
Then I have created a temporary function with the following command:

- CREATE TEMPORARY FUNCTION con AS ‘udf.ConcatWs’;

```
acadgild@localhost:~/Pradeep/assignment5/eclipse-workspace
File  Edit  View  Search  Terminal  Help
hive> ADD JAR concatWS.jar;
Added [concatWS.jar] to class path
Added resources: [concatWS.jar]
hive> CREATE TEMPORARY FUNCTION con AS 'udf.ConcatWs';
OK
Time taken: 0.03 seconds
hive>
```

Then I have executed the following command to work with udf function:

- Select eskills from emp;
- Select con(“:”,eskills) from emp;
- Select con(“|”,eskills) from emp;



As you can see in the above screen shot I am able to add the different separators for the eskills string array that I am passing to the Hiveudf function.

Task 1.3:

Turning on transaction properties in hive

```
hive> set hive.support.concurrency=true;
hive> set hive.enforce.bucketing=true;
Query returned non-zero code: 1, cause: hive configuration hive.enforce.bucketing does not exists.
hive> set hive.exec.dynamic.partition.mode=true;
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>
```

Creating table and inserting records into the table

```
hive> create table college
    > (
    > clg_id int,
    > clg_name string,
    > clg_loc string
    > )
    > clustered by(clg_id) into 5 buckets stored as orc tblproperties('transactional'='true');
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask. AlreadyExistsException(message:Table coll
ege already exists)
hive> desc college;
OK
clg_id                  int
clg_name                string
clg_loc                 string
Time taken: 0.211 seconds, Fetched: 3 row(s)
hive> select * from college;
OK
Time taken: 2.886 seconds
```

```
hive> insert into table college values(1,'nec','nlr'),(2,'vit','vlr'),(3,'srm','chen'),(4,'lpu','del'),(5,'stanford','uk'),(6
,'JNTUA','atp'),(7,'cambridge','us');
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execu
tion engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = acadgild_20180911135343_170dd124-d0ff-439d-bdbc-a167c4dc0aba
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 5
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_1536650846118_0001, Tracking URL = http://localhost:8088/proxy/application_1536650846118_0001/
Kill Command = /home/acadgild/hadoop-2.7.2/bin/hadoop job  -kill job_1536650846118_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 5
2018-09-11 13:54:10,193 Stage-1 map = 0%,  reduce = 0%
2018-09-11 13:54:22,271 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.69 sec
2018-09-11 13:55:07,740 Stage-1 map = 100%,  reduce = 13%, Cumulative CPU 3.82 sec
2018-09-11 13:55:09,283 Stage-1 map = 100%,  reduce = 27%, Cumulative CPU 4.73 sec
2018-09-11 13:55:10,929 Stage-1 map = 100%,  reduce = 40%, Cumulative CPU 5.6 sec
2018-09-11 13:55:17,247 Stage-1 map = 100%,  reduce = 67%, Cumulative CPU 8.43 sec
2018-09-11 13:55:23,360 Stage-1 map = 100%,  reduce = 73%, Cumulative CPU 11.76 sec
2018-09-11 13:55:28,177 Stage-1 map = 100%,  reduce = 80%, Cumulative CPU 14.29 sec
2018-09-11 13:55:29,731 Stage-1 map = 100%,  reduce = 87%, Cumulative CPU 16.71 sec
2018-09-11 13:55:33,908 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 21.34 sec
MapReduce Total cumulative CPU time: 21 seconds 340 msec
Ended Job = job_1536650846118_0001
Loading data to table default.college
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 5   Cumulative CPU: 21.34 sec   HDFS Read: 26127 HDFS Write: 3998 SUCCESS
Total MapReduce CPU Time Spent: 21 seconds 340 msec
OK
Time taken: 114.278 seconds
```

```
hive> select * from college;
OK
5       stanford        uk
6       JNTUA   atp
1       nec     nlr
7       cambridge       us
2       vit     vlr
3       srm     chen
4       lpu     del
Time taken: 0.359 seconds, Fetched: 7 row(s)
```

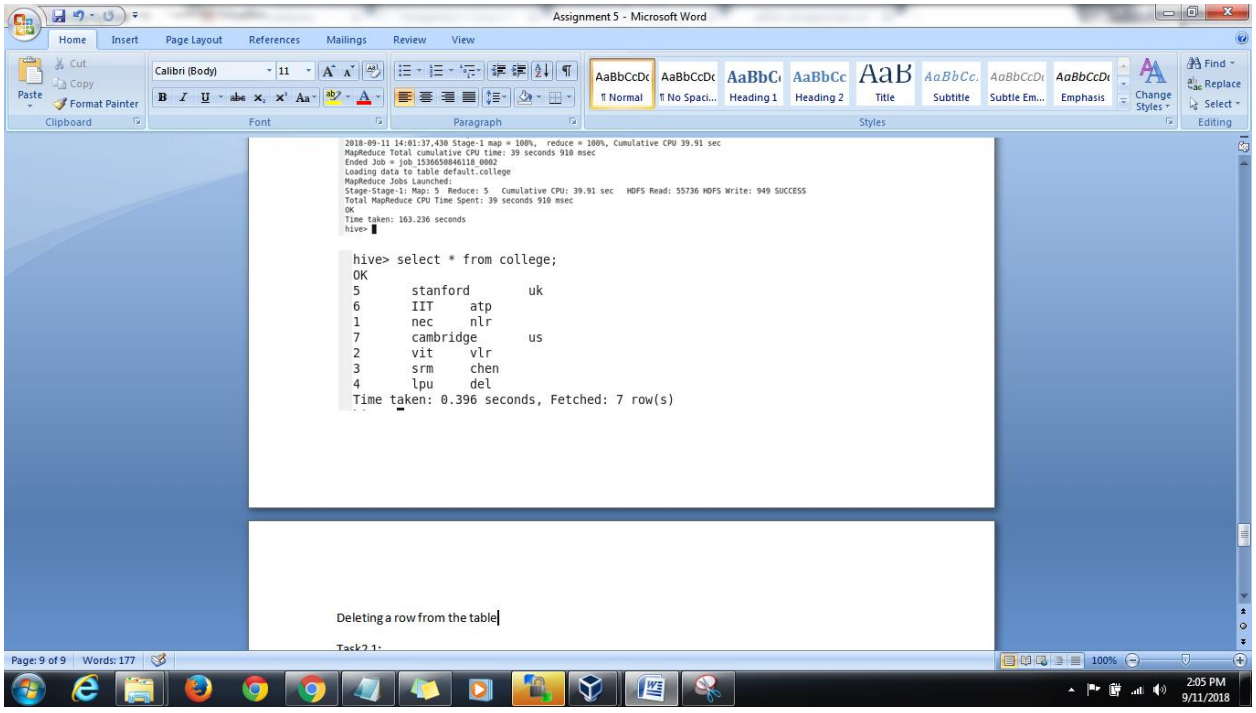
Updating data in table:

```
hive> update college set clg_id=8 where clg_id=7;
FAILED: SemanticException [Error 10302]: Updating values of bucketing columns is not supported.  Column clg_id.
hive>
```

```
hive> update college set clg_name='IIT' where clg_id=6;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execu
tion engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = acadgild_20180911135858_d707f81b-f917-497d-86b5-f1e55ecbded8
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 5
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_1536650846118_0002, Tracking URL = http://localhost:8088/proxy/application_1536650846118_0002/
Kill Command = /home/acadgild/hadoop-2.7.2/bin/hadoop job  -kill job_1536650846118_0002
Hadoop job information for Stage-1: number of mappers: 5; number of reducers: 5
2018-09-11 13:59:15,987 Stage-1 map = 0%,  reduce = 0%
2018-09-11 14:00:17,025 Stage-1 map = 0%,  reduce = 0%, Cumulative CPU 9.62 sec
2018-09-11 14:00:24,958 Stage-1 map = 20%,  reduce = 0%, Cumulative CPU 10.94 sec
2018-09-11 14:00:34,626 Stage-1 map = 60%,  reduce = 0%, Cumulative CPU 22.07 sec
2018-09-11 14:00:36,041 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 26.73 sec
2018-09-11 14:01:14,590 Stage-1 map = 100%,  reduce = 13%, Cumulative CPU 27.69 sec
2018-09-11 14:01:23,946 Stage-1 map = 100%,  reduce = 27%, Cumulative CPU 28.64 sec
2018-09-11 14:01:25,753 Stage-1 map = 100%,  reduce = 60%, Cumulative CPU 32.45 sec
2018-09-11 14:01:28,506 Stage-1 map = 100%,  reduce = 73%, Cumulative CPU 33.85 sec
2018-09-11 14:01:34,398 Stage-1 map = 100%,  reduce = 80%, Cumulative CPU 35.61 sec
2018-09-11 14:01:35,884 Stage-1 map = 100%,  reduce = 93%, Cumulative CPU 38.18 sec
2018-09-11 14:01:37,430 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 39.91 sec
MapReduce Total cumulative CPU time: 39 seconds 910 msec
Ended Job = job_1536650846118_0002
Loading data to table default.college
MapReduce Jobs Launched:
Stage-Stage-1: Map: 5  Reduce: 5   Cumulative CPU: 39.91 sec   HDFS Read: 55736 HDFS Write: 949 SUCCESS
Total MapReduce CPU Time Spent: 39 seconds 910 msec
OK
Time taken: 163.236 seconds
hive>
```

```
hive> select * from college;
OK
5       stanford        uk
6       IIT     atp
1       nec     nlr
7       cambridge       us
2       vit     vlr
3       srm     chen
4       lpu     del
Time taken: 0.396 seconds, Fetched: 7 row(s)
```

Deleting a row from the table:




```
hive> select * from college;
OK
6       IIT     atp
1       nec     nlr
7       cambridge       us
2       vit     vlr
3       srm     chen
4       lpu     del
Time taken: 0.393 seconds, Fetched: 6 row(s)
hive>
```

Task2.1:

1.What is NoSQL database?

NoSQL is an approach to database design that can accommodate a wide variety of data models, including key-value,document,columnar and graph formats. NoSQL,which stand for “not only SQL,” is an alternative to traditional relational databases in which data is placed in tables and data schema is carefully designed before the database is built. NoSQL databases are especially useful for working with large sets of distributed data.

Examples for NoSQL database is MongoDB,Cassandra,Hbase

2.How does data get stored in NoSQL database?

Data Storage is not based on a single data model. Most outstanding ones are key-value pair, graph, document, and columnar.

3.What is a column family in HBase?

A column family defines shared features to all columns that are created within them (think of it almost as a sub-table within your larger table). You will notice that HBase columns are composed of a combination of the column family and column qualifier (or column key): 'family:qualifier'

Where the qualifier can be an arbitrary array of bytes, the column family has to be composed of printable characters. Also, on HDFS, the column family is what is stored in human-readable format, example: '/hbase/table/region/<colfamX>'

Remember - all columns families must be created up-front whereas columns can be added on the fly. Hence understanding the design of your data access pattern is crucial when first building your HBase instance.

4.How many maximum number of columns can be added to HBase table?

It has been recommended to keep the number of column families under three. But there is no magic number like this. Why not two? Why not four? Technically, HBase can manage more than three of four column families. However, you need to understand how column families work to make the best use of them.

5.Why columns are not defined at the time of table creation in HBase?

An HBase table is made of column families which are the logical and physical grouping of columns. The columns in one family are stored separately from the columns in another family.

A single column family contains one or more columns, Column families must be defined at table creation time but columns can be added dynamically after table creation (if an insert statement states a column that does not exist for a column family it will create it).

6.How does data get managed in HBase?

The data in the HBASE is managed in such a way that,

- **WAL:** Write Ahead Log is a file on the distributed file system. The WAL is used to store new data that hasn't yet been persisted to permanent storage; it is used for recovery in the case of failure.
- **BlockCache:** is the read cache. It stores frequently read data in memory. Least Recently Used data is evicted when full.
- **MemStore:** is the write cache. It stores new data which has not yet been written to disk. It is sorted before writing to disk. There is one MemStore per column family per region.
- **Hfiles** store the rows as sorted KeyValues on disk.

All the above are sub components of Region server which manages the data in the HBASE.

7.What happens internally when new data gets inserted into HBase table?

- The client gets the Region server that hosts the META table from ZooKeeper.
- The client will query the .META. Server to get the region server corresponding to the row key it wants to access. The client caches this information along with the META table location.
- It will get the Row from the corresponding Region Server.

For future reads, the client uses the cache to retrieve the META location and previously read row keys. Over time, it does not need to query the META table, unless there is a miss because a region has moved; then it will re-query and update the cache.

Task 2.2:

Before working with HBase actual commands we need to start the Hadoop and HBase services with the following commands:

- Start-all.sh
- Start-hbase.sh

```
[acadgild@localhost ~]$ jps
3379 NodeManager
13635 Jps
12852
3269 ResourceManager
13269 HMaster
2951 DataNode
12473 Main
2831 NameNode
[acadgild@localhost ~]$
```

1. Create an HBase table named 'clicks' with a column family 'hits' such that it should be able to store last 5 values of qualifiers inside 'hits' column family.

Following is the command to create a table ‘clicks’ with column family ‘hits’ and having 5 versions:

- Create ‘clicks’, NAME=>‘hits’, VERSIONS=>5

```
hbase(main):005:0> create 'clicks',NAME=>'hits',VERSIONS=>5
0 row(s) in 0.5870 seconds

=> Hbase::Table - clicks
hbase(main):006:0>
```

2. Add few records in the table and update some of them. Use IP Address as row-key. Scan the table to view if all the previous versions are getting displayed.

To add records in the clicks table we use below put command and we use row ip address 192.168.48.12 as row key:

- Put ‘clicks’, ‘192.168.48.12’, ‘hits:noOfHits’, ‘12’

```
hbase(main):006:0> put 'clicks','192.168.1.11','hits:noOfHits',12
0 row(s) in 0.2260 seconds

hbase(main):007:0> scan clicks
NameError: undefined local variable or method `clicks' for #<Object:0x16015fc>

hbase(main):008:0> scan 'clicks'
ROW                             COLUMN+CELL
 192.168.1.11                   column=hits:noOfHits, timestamp=1536678070385, value=12
1 row(s) in 0.0890 seconds
```