Session 9:

Advance Hive

Assignment 1

Creating the table and loading input dataset-

CREATE DATABASE olympic;

USE olympic;

CREATE TABLE olympic_data

(Athlete STRING, Age INT, Country STRING, Year INT, Closing_Date STRING, Sport STRING, Gold_Medals INT, Silver_Medals INT, Bronze_Medals INT,Total_Medals INT)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';

LOAD DATA LOCAL INPATH '/home/acadgild/hadoop/olympic_data.csv'

INTO TABLE olympic_data;

PROBLEM STATEMENT –

Task 1

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

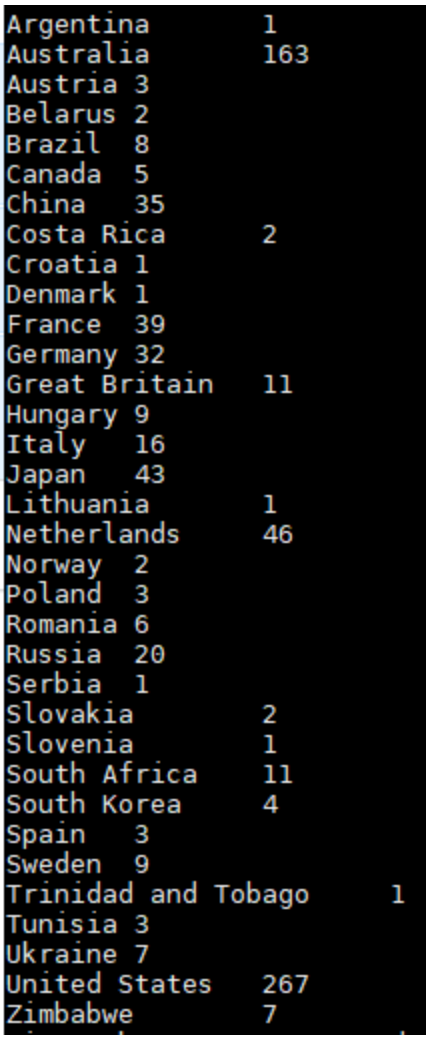
2. Write a Hive program to find the number of medals that India won year wise.

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

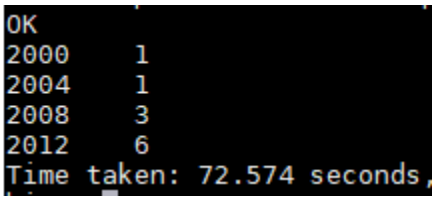
4. Write a Hive program to find the number of gold medals each country won.

SOLUTION –

1. **SELECT country, SUM(total_medals) as Total_Medals FROM olympic_data WHERE sport='Swimming' GROUP BY country;**



2. **SELECT year, SUM(total_medals) FROM olympic_data Where country='India' GROUP BY year;**



3. **SELECT country, SUM(total_medals) as Total_Medals FROM olympic_data GROUP BY country;**

```
OK
country total_medals
Afghanistan     2
Algeria 8
Argentina       141
Armenia 10
Australia       609
Austria 91
Azerbaijan      25
Bahamas 24
Bahrain 1
Barbados        1
Belarus 97
Belgium 18
Botswana        1
Brazil  221
Bulgaria        41
Cameroon        20
Canada  370
Chile   22
China   530
Chinese Taipei  20
Colombia        13
Costa Rica      2
Croatia 81
Cuba    188
Cyprus  1
Czech Republic  81
Denmark 89
Dominican Republic      5
Ecuador 1
Egypt   8
Eritrea 1
Estonia 18
Ethiopia        29
Finland 118
France  318
Gabon   1
Georgia 23
Germany 629
Great Britain   322
Greece  59
Grenada 1
Guatemala       1
Hong Kong       3
Hungary 145
Iceland 15
India   11
Iran    24
Ireland 9
Israel  4
Italy   331
Jamaica 80
Japan   282
Kazakhstan      42
Kenya   39
Kuwait  2
Kyrgyzstan      3
Latvia  17
Lithuania       30
Macedonia       1
Malaysia        3
Mauritius       1
Mexico  38
Moldova 5
Mongolia        10
Montenegro      14
Morocco 11
Mozambique      1
Netherlands     318
New Zealand     52
Nigeria 39
North Korea     21
Norway  192
Panama  1
Paraguay        17
Poland  80
Portugal        9
Puerto Rico     2
Qatar   3
Romania 123
Russia  768
Saudi Arabia    6
Serbia  31
Serbia and Montenegro   38
Singapore       7
Slovakia        35
Slovenia        25
South Africa    25
South Korea     308
Spain   205
Sri Lanka       1
Sudan   1
Sweden  181
Switzerland     93
Syria   1
Tajikistan      3
Thailand        18
```

4. **SELECT country, SUM(gold_medals) as GOLD_Medals FROM olympic_data GROUP BY country;**

country	gold_medals
Afghanistan	0
Algeria	2
Argentina	49
Armenia	0
Australia	163
Austria	36
Azerbaijan	6
Bahamas	11
Bahrain	0
Barbados	0
Belarus	17
Belgium	2
Botswana	0
Brazil	46
Bulgaria	8
Cameroon	20
Canada	168
Chile	3
China	234
Chinese Taipei	2
Colombia	2
Costa Rica	0
Croatia	35
Cuba	57
Cyprus	0
Czech Republic	14
Denmark	46
Dominican Republic	3
Ecuador	0
Egypt	1
Eritrea	0
Estonia	6
Ethiopia	13
Finland	11
France	108
Gabon	0
Georgia	6
Germany	223
Great Britain	124
Greece	12
Grenada	1
Guatemala	0
Hong Kong	0
Hungary	77
Iceland	0
India	1
Indonesia	5
Iran	10
Ireland	1
Italy	86
Jamaica	24
Japan	57
Kazakhstan	13
Kenya	11
Kuwait	0
Kyrgyzstan	0
Latvia	3
Lithuania	5
Macedonia	0
Malaysia	0
Mauritius	0
Mexico	19
Moldova	0
Mongolia	2
Montenegro	0
Morocco	2
Mozambique	1
Netherlands	101
New Zealand	18
Nigeria	6
North Korea	6
Norway	97
Panama	1
Paraguay	0
Poland	20
Portugal	1
Puerto Rico	0
Qatar	0
Romania	57
Russia	234
Saudi Arabia	0
Serbia	1
Serbia and Montenegro	11
Singapore	0
Slovakia	10
Slovenia	5
South Africa	10
South Korea	110
Spain	19
Sri Lanka	0
Sudan	0
Sweden	57
Switzerland	21
Syria	0
Tajikistan	0
Thailand	6
Togo	0
Trinidad and Tobago	1
Tunisia	2

Task 2

Write a hive UDF that implements functionality of string **concat_ws**(string SEP, array<string>).

This UDF will accept two arguments, one string and one array of string.

It will return a single string where all the elements of the array are separated by the SEP.

SOLUTION –

To implement the above case study we use an employee database where,
Employee_ID = Firstname_Lastname

Dataset –

The below data contains, the column name as,

EMPNO	FNAME	LNAME
1	John	Bravo
2	Steve	Austin
3	Karl	Marx
4	Stephen	Hawkings
5	Robert	Gerald
6	Luis	Fane
7	Jake	Smith
8	Donald	Ben
9	Zen	Zeplin
10	Jesus	Navas

Create DATABASE and TABLE –

```
CREATE TABLE EMPLOYEE(empno int, fname string,lname string)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

LOAD DATA LOCAL INPATH '/home/acadgild/hadoop/emp.txt'

INTO TABLE emp.EMPLOYEE;
```

```
hive> select * from EMPLOYEE;
OK
1       John    Bravo
2       Steve   Austin
3       Karl    Marx
4       Stephen Hawkings
5       Robert  Gerald
6       Luis    Figo
7       Jake    Smith
8       Donald  Ben
9       Zen     Zeplin
10      Jesus   Navas
Time taken: 0.318 seconds, Fetched: 10 row(s)
```

HIVE UDF Java code –

```
package concatws;

import org.apache.hadoop.hive.ql.exec.UDF;

public class concatws extends UDF

{
	public String evaluate(String param1, String[] param2)

{
	String Output = "";
	if(param1==null && param2==null)
	{
		return null;
	}
	for(int i = 0; i < param2.length; i++)
	{
		Output+= param2[i];
	}
	return(param1.concat(Output));
}
}
```

Adding the JAR created from the JAVA class which is defining the UDF using below syntax-

HIVE UDF CONCATWS function –

add jar /home/acadgild/hadoop/hiveudf.jar;

```
hive> add jar /home/acadgild/hadoop/hiveudf.jar;
Added [/home/acadgild/hadoop/hiveudf.jar] to class path
Added resources: [/home/acadgild/hadoop/hiveudf.jar]
hive>
```

After that we are creating a temporary function “CONCAT_WS” using below syntax-

CREATE TEMPORARY FUNCTION CONCAT_WS AS 'concatws.concatws';

```
hive> CREATE TEMPORARY FUNCTION CONCAT_WS AS 'concatws.concatws';
OK
Time taken: 0.052 seconds
hive>
```

After that we run the below query to take one column (empno) input as Int and another array(fname,’_’,lname) as Array of Strings and concatenate them,

HIVE QL –

SELECT empno, fname, lname, CONCAT_WS(fname,'_',lname) from EMPLOYEE;

OUTPUT –

```
hive> SELECT empno, fname, lname, CONCAT_WS(fname,'_',lname) from EMPLOYEE;
OK
1       John    Bravo   John_Bravo
2       Steve   Austin  Steve_Austin
3       Karl    Marx    Karl_Marx
4       Stephen Hawkings        Stephen_Hawkings
5       Robert  Gerald  Robert_Gerald
6       Luis    Figo    Luis_Figo
7       Jake    Smith   Jake_Smith
8       Donald  Ben     Donald_Ben
9       Zen     Zeplin                          Zen_Zeplin
10      Jesus   Navas   Jesus_Navas
Time taken: 0.445 seconds, Fetched: 10 row(s)
```

Task 3 -

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.

The below properties needs to be set appropriately in hive shell, order-wise to work with transactions 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=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=5;
hive> set hive.compactor.worker.threads;
hive.compactor.worker.threads=5
hive>
```

Creating a Table That Supports Hive Transactions –

The below syntax will create a table with name **'movie'** and the columns present in the table are **'emp_id, emp_name and emp_loc'** . We are bucketing the table by '**emp_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'**

HIVE Code:

```
CREATE TABLE EMP
(emp_id int,
emp_name string,
emp_loc string)
clustered by (emp_id) INTO 5 buckets STORED as orc TBLPROPERTIES('transactional'='true');
```

```
hive> CREATE TABLE EMP
    > (emp_id int,
    > emp_name string,
    > emp_loc string)
    > clustered by (emp_id) INTO 5 buckets STORED as orc TBLPROPERTIES('transactional'='true');
OK
Time taken: 0.398 seconds
hive>
```

Inserting Data into a Hive Table -

The below command is used to insert row wise data into the Hive table. Here, each row is separated by '( )' brackets.

HIVE Code:

INERT INTO TABLE EMP values
(1,'JACK','BANGALORE'),(2,'JILL','CHENNAI'),(3,'STONE','MUMBAI'),(4,'HENRY','PUNE'),(5,'STEVE','RANCHI'),(6,'COOK','PUNJAB'),(7,'KANE','SURAT'),(8,'HARRY','DELHI'),(9,'JOHN','NEPAL'),(10,'ERIC','NOIDA');

```
hive> INSERT INTO TABLE EMP values (1,'JACK','BANGALORE'),(2,'JILL','CHENNAI'),(3,'STONE','MUMBAI'),(4,'HENRY','PUNE'),(5,'STEVE','RANCHI'),(6,'COOK','PUNJAB'),(7,'
KANE','SURAT'),(8,'HARRY','DELHI'),(9,'JOHN','NEPAL'),(10,'ERIC','NOIDA');
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_20181002080452_159dfff8-6ef0-47b8-aafb-d9bc120784bd
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_1538442298147_0004, Tracking URL = http://localhost:8088/proxy/application_1538442298147_0004/
Kill Command = /home/acadgild/install/hadoop/hadoop-2.6.5/bin/hadoop job  -kill job_1538442298147_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 5
2018-10-02 08:06:07,387 Stage-1 map = 0%,  reduce = 0%
2018-10-02 08:06:49,939 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 8.92 sec
2018-10-02 08:07:51,052 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 8.92 sec
2018-10-02 08:08:26,474 Stage-1 map = 100%,  reduce = 53%, Cumulative CPU 19.98 sec
2018-10-02 08:08:29,076 Stage-1 map = 100%,  reduce = 67%, Cumulative CPU 21.01 sec
2018-10-02 08:09:17,226 Stage-1 map = 100%,  reduce = 76%, Cumulative CPU 59.91 sec
2018-10-02 08:09:19,035 Stage-1 map = 100%,  reduce = 82%, Cumulative CPU 84.02 sec
2018-10-02 08:09:23,974 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 91.85 sec
MapReduce Total cumulative CPU time: 1 minutes 33 seconds 780 msec
Ended Job = job_1538442298147_0004
Loading data to table emp.emp
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 5   Cumulative CPU: 93.78 sec   HDFS Read: 26794 HDFS Write: 4217 SUCCESS
Total MapReduce CPU Time Spent: 1 minutes 33 seconds 780 msec
OK
Time taken: 344.578 seconds
hive>
```

```
hive> select * from emp;
OK
10      ERIC    NOIDA
5       STEVE   RANCHI
6       COOK    PUNJAB
1       JACK    BANGALORE
7       KANE    SURAT
2       JILL    CHENNAI
8       HARRY   DELHI
3       STONE   MUMBAI
9       JOHN    NEPAL
4       HENRY   PUNE
Time taken: 1.32 seconds, Fetched: 10 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 EMP values (1,'JACK','BANGALORE'),(2,'JILL','CHENNAI'),(3,'STONE','MUMBAI'),(4,'HENRY','PUNE'),(5,'STEVE','RANCHI'),(6,'COOK','PUNJAB'),(7,'
KANE','SURAT'),(8,'HARRY','DELHI'),(9,'JOHN','NEPAL'),(10,'ERIC','NOIDA');
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_20181002081244_ff0aff26-1565-4b9b-86d3-26dbebb985b2
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_1538442298147_0005, Tracking URL = http://localhost:8088/proxy/application_1538442298147_0005/
Kill Command = /home/acadgild/install/hadoop/hadoop-2.6.5/bin/hadoop job  -kill job_1538442298147_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 5
2018-10-02 08:13:28,874 Stage-1 map = 0%,  reduce = 0%
2018-10-02 08:13:56,300 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 10.1 sec
2018-10-02 08:14:57,649 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 10.1 sec
2018-10-02 08:15:11,817 Stage-1 map = 100%,  reduce = 13%, Cumulative CPU 15.53 sec
2018-10-02 08:15:14,234 Stage-1 map = 100%,  reduce = 27%, Cumulative CPU 16.68 sec
2018-10-02 08:15:32,513 Stage-1 map = 100%,  reduce = 47%, Cumulative CPU 24.94 sec
2018-10-02 08:15:37,815 Stage-1 map = 100%,  reduce = 53%, Cumulative CPU 30.31 sec
2018-10-02 08:15:40,738 Stage-1 map = 100%,  reduce = 67%, Cumulative CPU 31.6 sec
2018-10-02 08:16:20,041 Stage-1 map = 100%,  reduce = 70%, Cumulative CPU 46.03 sec
2018-10-02 08:16:25,657 Stage-1 map = 100%,  reduce = 73%, Cumulative CPU 58.89 sec
2018-10-02 08:17:15,338 Stage-1 map = 100%,  reduce = 80%, Cumulative CPU 66.82 sec
2018-10-02 08:17:26,068 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 107.71 sec
MapReduce Total cumulative CPU time: 1 minutes 49 seconds 800 msec
Ended Job = job_1538442298147_0005
Loading data to table emp.emp
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 5   Cumulative CPU: 109.8 sec   HDFS Read: 26669 HDFS Write: 4216 SUCCESS
Total MapReduce CPU Time Spent: 1 minutes 49 seconds 800 msec
OK
Time taken: 290.24 seconds
```

```
hive> select * from emp;
OK
10      ERIC    NOIDA
5       STEVE   RANCHI
10      ERIC    NOIDA
5       STEVE   RANCHI
6       COOK    PUNJAB
1       JACK    BANGALORE
6       COOK    PUNJAB
1       JACK    BANGALORE
7       KANE    SURAT
2       JILL    CHENNAI
7       KANE    SURAT
2       JILL    CHENNAI
8       HARRY   DELHI
3       STONE   MUMBAI
8       HARRY   DELHI
3       STONE   MUMBAI
9       JOHN    NEPAL
4       HENRY   PUNE
9       JOHN    NEPAL
4       HENRY   PUNE
Time taken: 1.651 seconds, Fetched: 20 row(s)
hive>
```

Earlier, we inserted **10** rows, now the same command has been executed and the same data is appended to the previous data and we have fetched **20** rows.

Updating the Data in Hive Table –

HIVE Code:

UPDATE emp SET emp_id = 7 where emp_id = 2;

The above command is used to update a row in Hive table.

```
hive> UPDATE emp SET emp_id = 7 where emp_id = 2;
FAILED: SemanticException [Error 10302]: Updating values of bucketing columns is not supported.  Column emp_id.
hive>
```

From the above image, we can see that we have received an error message. This means that the Update command is not supported on the columns that are bucketed.

In this table, we have bucketed the **'emp_id'** column and performing the update operation on the same column, so we have got the error.

Now let's perform the update operation on Non bucketed column,

UPDATE emp SET emp_loc = 'HYDERABAD' where emp_id = 7;

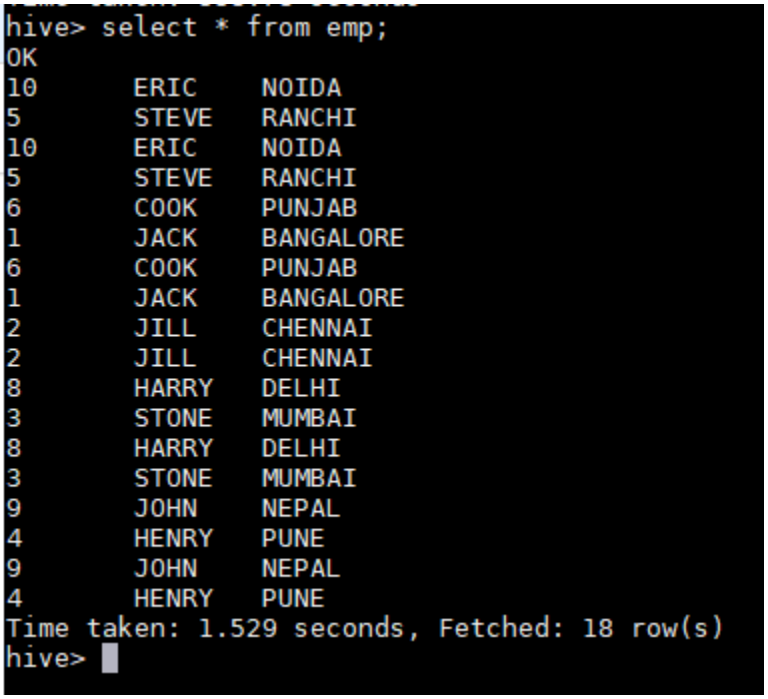
```
hive> select * from emp;
OK
10      ERIC    NOIDA
5       STEVE   RANCHI
10      ERIC    NOIDA
5       STEVE   RANCHI
6       COOK    PUNJAB
1       JACK    BANGALORE
6       COOK    PUNJAB
1       JACK    BANGALORE
7       KANE    HYDERABAD
2       JILL    CHENNAI
7       KANE    HYDERABAD
2       JILL    CHENNAI
8       HARRY   DELHI
3       STONE   MUMBAI
8       HARRY   DELHI
3       STONE   MUMBAI
9       JOHN    NEPAL
4       HENRY   PUNE
9       JOHN    NEPAL
4       HENRY   PUNE
Time taken: 2.287 seconds, Fetched: 20 row(s)
hive>
```

We have successfully updated the data **emp_loc** where the **emp_id =7**. It can be seen above that the **emp_loc** for the **emp_id=*7*** was **'SURAT'** and now it is updated to **'HYDERABAD'.**

Deleting a Row from Hive Table -

HIVE Code:

DELETE FROM emp WHERE emp_id=7;



We have now successfully deleted a row from the Hive table. This can be checked using the command **select * from emp.** We can see only **18** rows where our actual data is **20** rows. We can see there is not **emp_id=8.**