**Assignment 3.6**

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.

**Transactions in Hive**

Transactions in Hive are introduced in Hive 0.13, but they only partially fulfill the ACID properties like atomicity, consistency, durability, at the partition level. Here, Isolation can be provided by turning on one of the locking mechanisms available with zookeeper or in memory.

But in Hive 0.14, new API’s have been added to completely fulfill the ACID properties while performing any transaction.

Transactions are provided at the row-level in Hive 0.14. The different row-level transactions available in Hive 0.14 are as follows:

1. Insert
2. Delete
3. Update

There are numerous limitations with the present transactions available in Hive 0.14. ORC is the file format supported by Hive transaction. It is now essential to have ORC file format for performing transactions in Hive. The table needs to be bucketed in order to support transactions.

### Row-level Transactions Available in Hive 0.14

Let’s perform some row-level transactions available in Hive 0.14. 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>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 = a positive number on at least one instance of the Thrift metastore service;

Time taken: 0.126 seconds, Fetched: 6 row(s)

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 = a positive number on at least one instance of the Thrift metastore service;

Query returned non-zero code: 1, cause: 'SET hive.compactor.worker.threads=a positive number on at least one instance of the Thrift metastore service' FAILED because hive.compactor.worker.threads expects INT type value.

hive> set hive.compactor.worker.threads = 2;

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');

OK

Time taken: 11.915 seconds

The above syntax will create a table with name ‘*college’*and the columns present in the table are ‘*clg\_id, clg\_name, clg\_loc’. W*e are *bucketing* the table by ‘*clg\_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’*

OK

Time taken: 11.915 seconds

hive> show tables ;

OK

college

complex\_data\_type

emp

emp\_dtl

emp\_view

emp\_view1

employee\_details

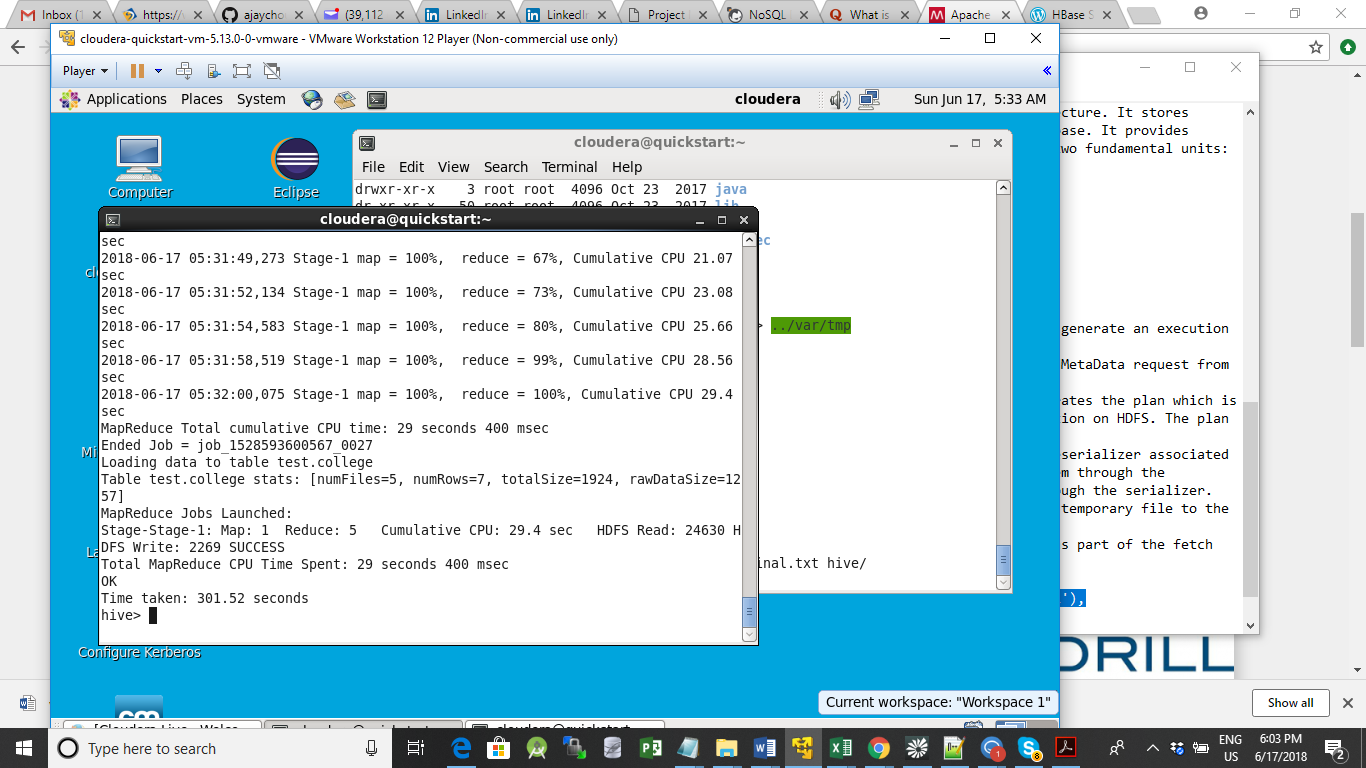
olympics\_data

Time taken: 0.428 seconds, Fetched: 8 row(s)

**We have successfully created a table with name ‘*college’*which supports row-level transactions of Hive.**

## Inserting Data into a Hive Table

**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');**



*Now, we have successfully inserted the data into the Hive table.*

*The contents of the table can be viewed using the command****select \* from college***

**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: 0.506 seconds, Fetched: 7 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 college values(1,'nec','nlr'),(2,'vit','vlr'),(3,'srm','chen'),(4,'lpu','del'),(5,'stanford','uk'),(6,'JNTUA','atp'),(7,'cambridge','us');

Query ID = cloudera\_20180617053636\_a0427b4f-c2df-478f-8ad5-e7ccaec22d6d

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\_1528593600567\_0028, Tracking URL = http://quickstart.cloudera:8088/proxy/application\_1528593600567\_0028/

Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job\_1528593600567\_0028

Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 5

2018-06-17 05:36:51,706 Stage-1 map = 0%, reduce = 0%

2018-06-17 05:37:04,077 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.42 sec

2018-06-17 05:37:59,749 Stage-1 map = 100%, reduce = 13%, Cumulative CPU 4.51 sec

2018-06-17 05:38:03,840 Stage-1 map = 100%, reduce = 40%, Cumulative CPU 8.7 sec

2018-06-17 05:38:06,577 Stage-1 map = 100%, reduce = 67%, Cumulative CPU 12.72 sec

2018-06-17 05:38:09,248 Stage-1 map = 100%, reduce = 73%, Cumulative CPU 14.26 sec

2018-06-17 05:38:10,515 Stage-1 map = 100%, reduce = 80%, Cumulative CPU 15.65 sec

2018-06-17 05:38:11,736 Stage-1 map = 100%, reduce = 87%, Cumulative CPU 17.19 sec

2018-06-17 05:38:13,074 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 20.29 sec

MapReduce Total cumulative CPU time: 20 seconds 290 msec

Ended Job = job\_1528593600567\_0028

Loading data to table test.college

Table test.college stats: [numFiles=10, numRows=14, totalSize=3848, rawDataSize=2514]

MapReduce Jobs Launched:

Stage-Stage-1: Map: 1 Reduce: 5 Cumulative CPU: 20.29 sec HDFS Read: 25095 HDFS Write: 2269 SUCCESS

Total MapReduce CPU Time Spent: 20 seconds 290 msec

OK

Time taken: 100.695 seconds

hive> select \* from college ;

OK

5 stanford uk

5 stanford uk

6 JNTUA atp

1 nec nlr

6 JNTUA atp

1 nec nlr

7 cambridge us

2 vit vlr

7 cambridge us

2 vit vlr

3 srm chen

3 srm chen

4 lpu del

4 lpu del

Time taken: 0.343 seconds, Fetched: 14 row(s)

hive>

## Updating the Data in Hive Table

UPDATE college set clg\_id = 8 where clg\_id = 7;

hive> UPDATE college set clg\_id = 8 where clg\_id = 7;

FAILED: SemanticException [Error 10294]: Attempt to do update or delete using transaction manager that does not support these operations.

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 ***‘clg\_id’*** column and performing the Update operation on the same column, so we have go the error

**Now let’s perform the update operation on Non bucketed column**

UPDATE college set clg\_name = 'IIT' where clg\_id = 6;

hive> UPDATE college set clg\_loc = 'us' where clg\_id = 5;

FAILED: SemanticException [Error 10294]: Attempt to do update or delete using transaction manager that does not support these operations.

## Deleting a Row from Hive Table

delete from college where clg\_id=5;

hive> delete from college where clg\_id=5;

FAILED: SemanticException [Error 10294]: Attempt to do update or delete using transaction manager that does not support these operations.

hive>

**This is how the transactions or row-wise operations are performed in Hive**.

***The CDH distribution of Hive does not support transactions****(*[*HIVE-5317*](https://issues.apache.org/jira/browse/HIVE-5317)*). Currently, transaction support in Hive is an experimental feature that only works with the ORC file format. Cloudera recommends using the Parquet file format,*[*which works across many tools*](http://blog.cloudera.com/blog/2014/02/native-parquet-support-comes-to-apache-hive/)*. Merge updates in Hive tables using existing functionality, including statements such as INSERT, INSERT OVERWRITE, and CREATE TABLE AS SELECT.*

[*https://www.cloudera.com/documentation/enterprise/latest/topics/hive\_ingesting\_and\_querying\_data.htm...*](https://www.cloudera.com/documentation/enterprise/latest/topics/hive_ingesting_and_querying_data.html#hive_transaction_support)

*If you require these features, please inquire about Apache Kudu.*

*Kudu is storage for fast analytics on fast data—providing a combination of fast inserts and updates alongside efficient columnar scans to enable multiple real-time analytic workloads across a single storage layer.*