**Session 8: ADVANCED HIVE**

Assignment 8.3

Student Name: Abarajithan SA

Course: Big Data Hadoop & Spark Training

Start Date:  2017-09-09

End Date:  2017-11-26

**Assignment 8.3**– Refer the 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.

Contents

[Introduction 2](#_Toc498946019)

[Associated Data Files 2](#_Toc498946020)

[Problem Statement 2](#_Toc498946021)

[Transactions in Hive 2](#_Toc498946022)

[Row-level Transactions Available in Hive 0.14 3](#_Toc498946023)

[Creating a Table That Supports Hive Transactions 3](#_Toc498946024)

[HIVE Code: 3](#_Toc498946025)

[Inserting Data into a Hive Table 4](#_Toc498946026)

[HIVE Code: 4](#_Toc498946027)

[Updating the Data in Hive Table 6](#_Toc498946028)

[HIVE Code: 6](#_Toc498946029)

[Deleting a Row from Hive Table 8](#_Toc498946030)

[HIVE Code: 8](#_Toc498946031)

# Introduction

In this assignment you need to implement transaction concepts in Hive. I’m using a dataset created on my own.

# Associated Data Files

<https://acadgild.com/blog/transactions-in-hive/>

# Problem Statement

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:

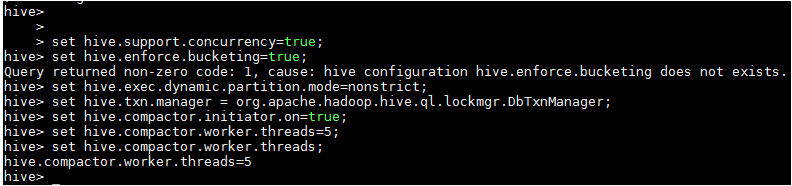
* Insert
* Delete
* Update

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:



# Creating a Table That Supports Hive Transactions

The above syntax will create a table with name **‘movie’** and the columns present in the table are **‘*movie\_id*, *movie\_code*, *movie\_name,screen\_number and screen\_loc’*** . We are bucketing the table by ‘***movie\_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 movie***

***(movie\_id int,***

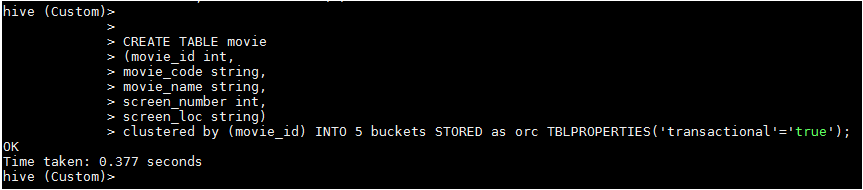
***movie\_code string,***

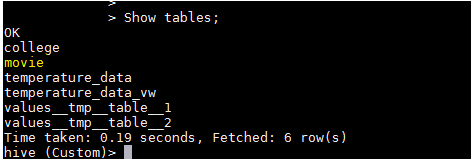
***movie\_name string,***

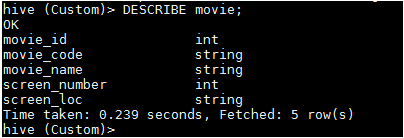
***screen\_number int,***

***screen\_loc string)***

***clustered by (movie\_id) INTO 5 buckets STORED as orc TBLPROPERTIES('transactional'='true');***







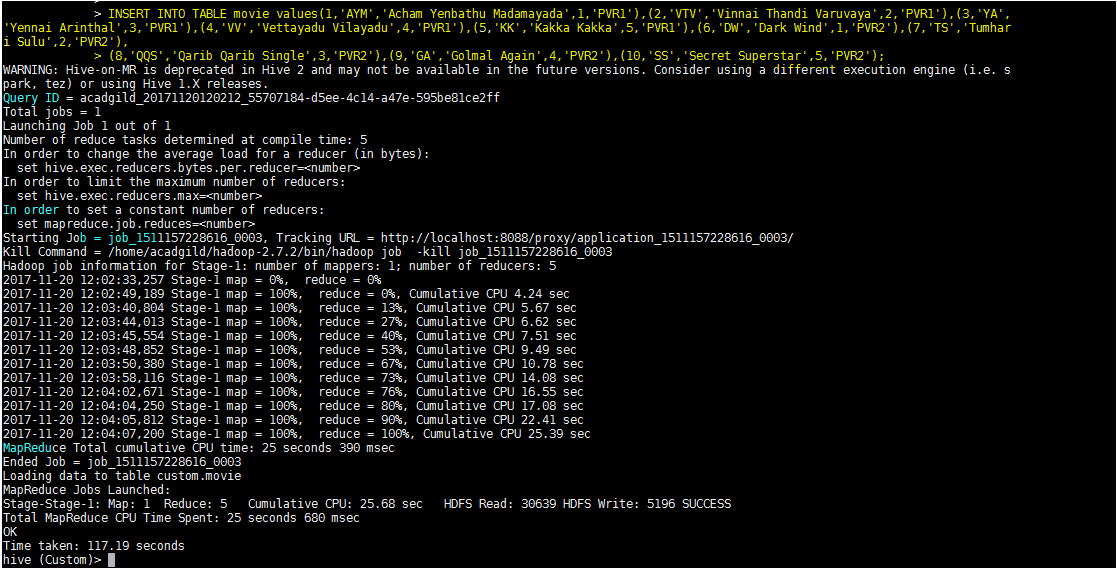
The table ‘**movie’** is successfully created which supports row level transactions of HIVE.

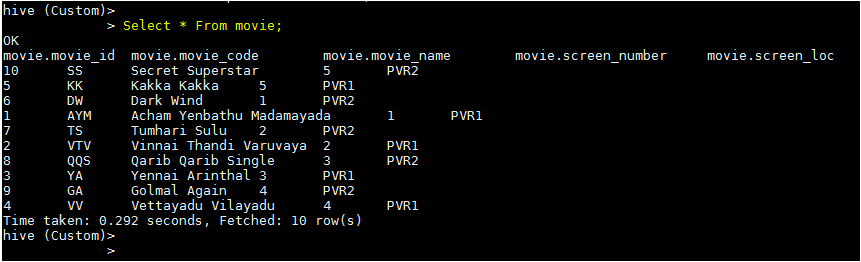
# Inserting Data into a Hive Table

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

## HIVE Code:

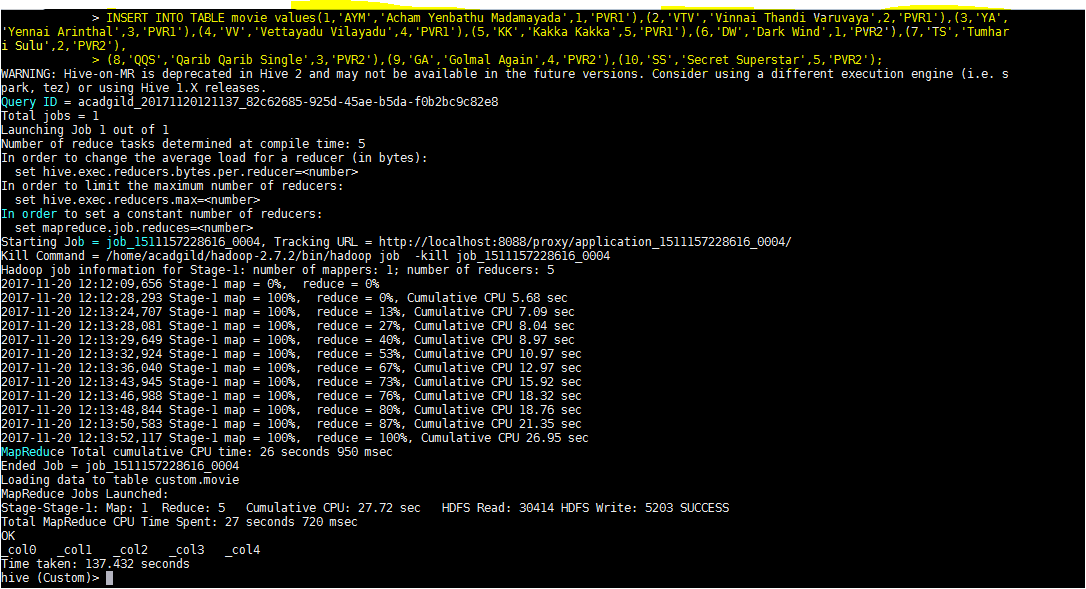
***INSERT INTO TABLE movie values(1,'AYM','Acham Yenbathu Madamayada',1,'PVR1'),(2,'VTV','Vinnai Thandi Varuvaya',2,'PVR1'),(3,'YA','Yennai Arinthal',3,'PVR1'),(4,'VV','Vettayadu Vilayadu',4,'PVR1'),(5,'KK','Kakka Kakka',5,'PVR1'),(6,'DW','Dark Wind',1,'PVR2'),(7,'TS','Tumhari Sulu',2,'PVR2'),(8,'QQS','Qarib Qarib Single',3,'PVR2'),(9,'GA','Golmal Again',4,'PVR2'),(10,'SS','Secret Superstar',5,'PVR2');***

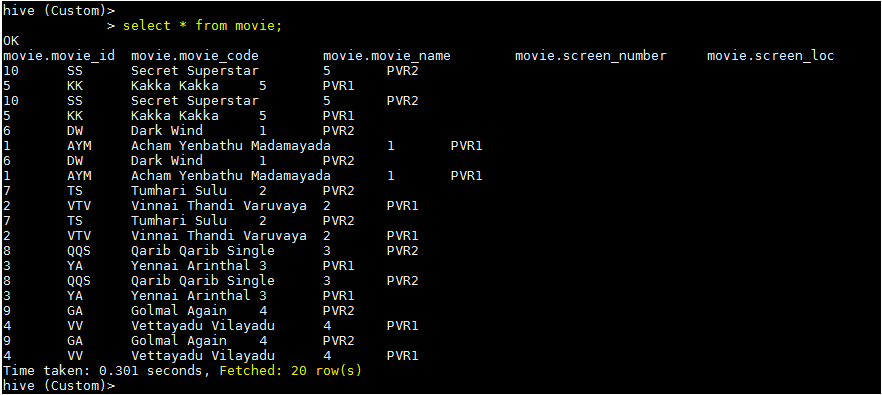




Now, we have successfully inserted the data into the Hive table and we saw the table data above using ***Select \* From movie;***

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





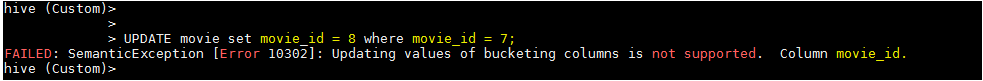
Earlier, we inserted **10** rows, now the same command has been executed and the same data is appended to the previous data and we have fetched **20** rows.

# Updating the Data in Hive Table

## HIVE Code:

***UPDATE movie set movie\_id = 8 where movie\_id = 7;***

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



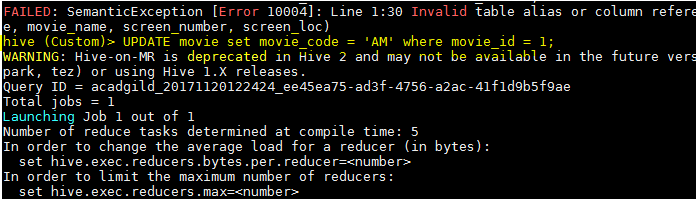
From the above image, we can see that we have received an error message. This means that the Update command is not supported on the columns that are bucketed.

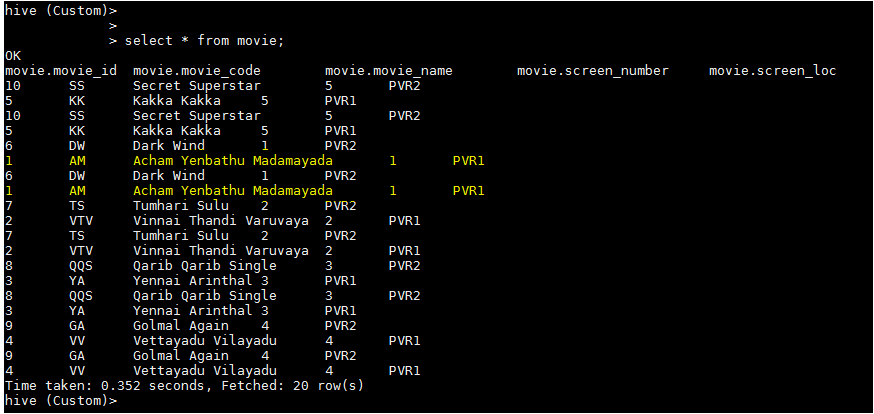
In this table, we have bucketed the **‘movie\_id’** column and performing the Update operation on the same column, so we have go the error

***“FAILED: SemanticException[Error 10302]: Updating values of bucketing columns is not supported. Column clg\_id”***

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

***UPDATE movie set movie\_code = 'AM' where movie\_id = 1;***



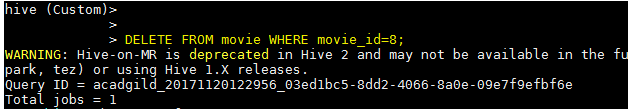


We have successfully updated the data **movie\_code** where the **movie\_id =1**. It can be seen above that the ***movie\_code*** for the ***movie\_id=1*** was **’AYM’** and now it is updated to **‘AM’**

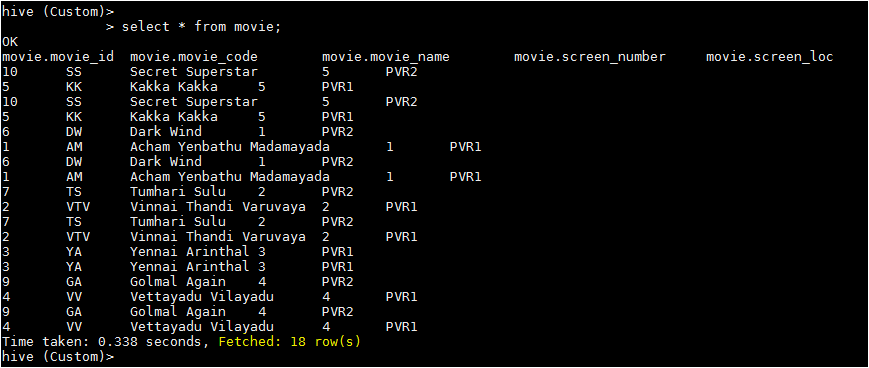
# Deleting a Row from Hive Table

## HIVE Code:

***DELETE FROM movie WHERE movie\_id=8;***



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



***---------------------------------------xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-----------------------------------------***