**Assignment 1**

- create a database named assign1

CREATE DATABASE assign1\_zeina;

- What is the database path on HDFS?

DESCRIBE DATABASE EXTENDED assign1\_zeina;

- create a database name assign1\_loc and set its location to /hp\_db/[db\_name]

CREATE DATABASE assign1\_loc\_zeina location ‘/hp\_db/assign1\_loc\_zeina’;

- create a hive managed table assign1\_intern\_tab inside the assign1 database with the right data types to host the data file employees

use assign1\_zeina;

create table assign1\_intern\_tab\_zeina(emolyee\_id integer, name string, age integer, city string)

row format delimited fields terminated by ',';

- What is the table path in HDFS?

DESCRIBE FORMATTED assign1\_intern\_tab\_zeina;

- load the data from the local file system into the table using two different commands

LOAD DATA LOCAL INPATH '/employee/employee\_details.txt' OVERWRITE INTO TABLE assign1\_intern\_tab\_zeina;

!hdfs dfs -copyFromLocal /employee/employee\_details.txt /user/hive/warehouse/assign1\_zeina.db/assign1\_intern\_tab\_zeina;

- select 10 records from the table as a sample to ensure the data was correctly loaded

Select \* from assign1\_intern\_tab\_zeina

Limit 10;

- Create external table assign1\_intern\_tab inside the assign1\_loc database

use assign1\_loc\_zeina;

create external table assign1\_intern\_tab\_zeina(emolyee\_id integer, name string, age integer, city string);

- What is the table path in HDFS?

DESCRIBE FORMATTED assign1\_intern\_tab\_zeina;

- move the data from local filesystem to the directory created in step 7

hdfs dfs -copyFromLocal /employee/employee\_details.txt /zeina01

-- Checking whether the file is copied successfully or not.

hdfs dfs -ls /zeina01

- drop both tables, is the data present after deletion or not?

use assign1\_zeina;

drop table assign1\_intern\_tab\_zeina;

use assign1\_loc\_zeina;

drop table assign1\_intern\_tab\_zeina;

**yes, data is present after deletion.**

- recreate both tables

use assign1\_zeina;

create table assign1\_intern\_tab\_zeina(emolyee\_id integer, name string, age integer, city string)

row format delimited fields terminated by ',';

use assign1\_loc\_zeina;

create external table assign1\_intern\_tab\_zeina(emolyee\_id integer, name string, age integer, city string);

- list both table directories

DESCRIBE FORMATTED assign1\_intern\_tab\_zeina;

DESCRIBE FORMATTED assign1\_loc\_zeina.assign1\_intern\_tab\_zeina;

- create internal table 'staging' inside the assign1 database

use assign1\_zeina;

create table staging(emolyee\_id integer, name string, age integer, city string)

row format delimited fields terminated by ',';

- load the staging table with the data from file employees

LOAD DATA LOCAL INPATH '/employee/employee\_details.txt' OVERWRITE INTO TABLE staging;

- load tables assign1\_intern\_tab and assign1\_extern\_tab from the staging table using INSERT SELECT statement

insert into assign1\_zeina.assign1\_intern\_tab\_zeina (emolyee\_id, name, age , city )

select \* from staging;

insert into assign1\_loc\_zeina.assign1\_intern\_tab\_zeina (emolyee\_id, name, age , city )

select \* from staging;

- List both directory tables and check if there is data or not

DESCRIBE FORMATTED assign1\_intern\_tab\_zeina;

DESCRIBE FORMATTED assign1\_loc\_zeina.assign1\_intern\_tab\_zeina;

- count the lines inside the file songs

 !wc /employee/songs.csv;

- create a table with the right types to host the data in file.

create table songs(artist\_id string, artist\_latitude double, artist\_location string, artist\_longitude double, artist\_name string, duration double, num\_songs integer, song\_id string, title string, year integer)

ROW FORMAT serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde';

LOAD DATA LOCAL INPATH '/employee/songs.csv' OVERWRITE INTO TABLE songs ;

- select 10 records from the table to ensure it's loaded correctly

Select \* from songs limit 10;

- count the number of records

Select count(artist\_id) from songs;

- is the hive count similar to the file count? is the data quality ok? If there is an issue, show how to resolve it

Yes they are similar to each other.

- create external table ...... to host

create external table songs\_ext(artist\_id string, artist\_latitude double, artist\_location string, artist\_longitude double, artist\_name string, duration double, num\_songs integer, song\_id string, title string, year integer)

ROW FORMAT serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde';

- load the table using put command

!hdfs dfs -put /employee/songs.csv /user/hive/warehouse/assign1\_zeina.db/songs\_ext;

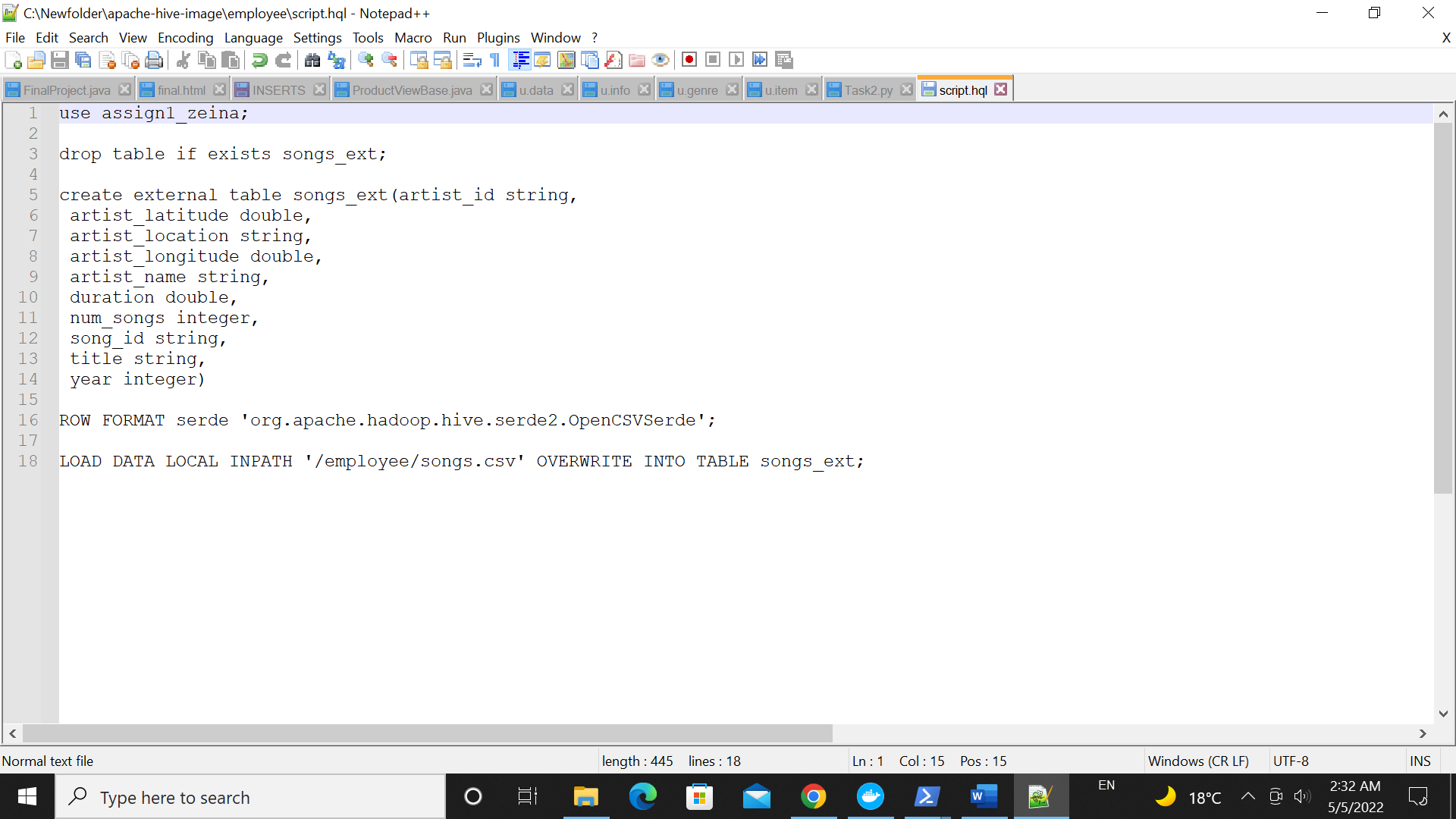
- is the data readable through the table? Why?

Yes, data is readable.

- select [logic] from table [] through shell without accessing hive or beeline

hive -e "select \* from assign1\_zeina.songs\_ext limit 10";

- create a hive script that drop table if exists, creates it and load data with data.



- execute it from shell without accessing hive CLI /beeline

hive -f /employee/script.hql

- What is a hive Temp table? how can you create it? why would someone use a temp table?

- Hive temporary tables are similar to temporary tables that exist in SQL Server or any RDBMS databases, As the name suggests these tables are created temporarily within an active session. Usually, temporary tables are created at the run time to store the intermediate data that are used to perform further data processing. once the processing is done either you can explicitly drop the temporary table or session termination will drop these tables.

- CREATE TEMPORARY TABLE temp1(col1 string);

- Because temporary tables execute more quickly.

- move the table assign1\_intern\_tab from one database to another

use assign1\_zeina;

alter table assign1\_intern\_tab\_zeina rename to assign1\_loc\_zeina.assign1\_intern\_tab;

- check the table directory and list its components

DESCRIBE FORMATTED assign1\_intern\_tab;

**Assignment 2**

Create a database named assign2

CREATE DATABASE assign2\_zeina;

Create table for songs table partitioned by artist and year. ensure the right data types are selected and the right SERDEPROPERTIES are used

set hive.exec.dynamic.partition=true;

CREATE TABLE songs(artist\_id string, artist\_latitude double, artist\_location string, artist\_longitude double, duration double, num\_songs integer, song\_id string, title string)

PARTITIONED BY (year integer, artist\_name string)

ROW FORMAT serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde';

- create external table ...... to host

CREATE EXTERNAL TABLE songs\_ext(artist\_id string, artist\_latitude double, artist\_location string, artist\_longitude double, duration double, num\_songs integer, song\_id string, title string)

PARTITIONED BY (artist\_name string, year integer)

ROW FORMAT serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde'

LOCATION '/songs';

Load data into table HDFS directory using put command

!hdfs dfs -put /employee/songs.csv /user/hive/warehouse/songs/assign2\_zeina.db/songs\_ext;

Run a SELECT check on the table, is there any data found? why?

select \* from songs\_ext limit 10;

No, data is not found, because it’s partitioned dynamically.

Add static partition using Alter and set partitions location in a separate directory from that of the table

ALTER TABLE songs\_ext ADD IF NOT EXISTS PARTITION (artist\_name='Marc Shaiman', year=2008)

LOCATION '/part\_table';

!hdfs dfs -put /employee/songs.csv /part\_table;

List the partition directories to check for presence of files

Show partitions songs;

Create a staging table to host songs data

CREATE TABLE staging\_tab(artist\_id string, artist\_latitude double, artist\_location string, artist\_longitude double, artist\_name string, duration double, num\_songs integer, song\_id string, title string, year integer)

ROW FORMAT serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde';

LOAD DATA LOCAL INPATH '/employee/songs.csv' OVERWRITE INTO TABLE staging\_tab;

Load the data from the staging table into songs table partitions dynamically

from staging\_tab

insert overwrite table songs partition (artist\_name, year)

select \*;

Truncate songs table and ensures no data in the table

Truncate table songs;

Use multi inserts to reload the data into the table fully dynamically

from staging\_tab

insert overwrite table songs partition (year,artist\_name )

select \*;

Truncate

Truncate table songs;

Use multi inserts to reload the data statically over year and dynamically by artist

from staging\_tab

insert overwrite table songs partition (year = 1994, artist\_name)

select artist\_id, artist\_latitude, artist\_location, artist\_longitude,

duration, num\_songs , song\_id , title string, artist\_name

where year = 1994;

Use CREATE TABLE LIKE statement to create a table with a schema similar to the staging table. The new table should be able to read Avro files

CREATE TABLE avro\_table LIKE staging\_tab

STORED AS AVRO;

Use CREATE TABLE LIKE statement to create a table with a schema similar to the staging table. The new table should be able to read Parquet files

CREATE TABLE parquet\_table LIKE staging\_tab

STORED AS PARQUET;

use the avro-tools getschema [avro\_file\_name] command to get the avro schema of the file

**Assignment 3**

File to be used events.csv

1. Create a table with the right data types and SERDEPROPERTIES to host the data from the events.csv files

CREATE TABLE events(artist string,auth string, firstName string, gender string, itemInSession integer, lastName string,length float, level string, location string, method string,page string, registration string, sessionId integer,song string, status integer, ts string, userAgent string, userId integer)

ROW FORMAT serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde';

1. Load the file from local filesystem to the hive table using LOAD statement

LOAD DATA LOCAL INPATH '/employee/events.csv' OVERWRITE INTO TABLE events;

1. Select the user, session, first song and last song played per session

Select firstName, sessionId,

First\_value(song) over(partition by sessionId), last\_value(song) over(partition by sessionId rows between unbounded preceding and unbounded following)

From events;

1. Rank users according to the number of distinct songs they played. If two users shared the same counts, they should have the same rank

Select distinct song, firstName, dense\_rank() over(partition by song order by userId)

from events;

1. Rank users according to the number of distinct songs they played. If two users shared the same counts, each user should have his/her own number. Note that records indicating s a played song are those with column ‘page’ equals to NextPage

Select distinct song, firstName, row\_number() over(partition by song order by userId)

from events

where page = "NextSong";

1. In the same table, show the count of songs played per location and artists, per location only and the total count

select count(song)

from events

group by artist,location

GROUPING SETS (artist,(artist,location),());

1. In the same table, show the count of songs played per location and artists, per location only , per artist only and the total count

select artist,location,count(song)

from events

group by artist,location

GROUPING SETS (artist,(artist,location),());

1. For each song played by a user, get the previous song and next song played. Get the count of each path, and fetch the top 10 paths found

Select firstName, lastName, lag(song) over(order by userId) as previous\_song, lead(song) over(order by userId) as next\_song

From events;

1. Select userid, song ordered by userid, song, ts. The query should be written to run on a single reducer

Select userId, song, ts

From events

Order by userId, song, ts;

1. Select userid, song ordered by userid, song, ts. The query should be written to run on a multiple reducer

Select userId, song, ts

From events

cluster by userId, song, ts;