**Assignment 1**

- create a database named assign1

create database assign1;

- What is the database path on HDFS?

describe database extended assign1;

the database path is hdfs://namenode:8020/user/hive/warehouse/assign1.db

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

create database assign1\_loc location 'hdfs://namenode:8020/user/hive/warehouse/nayra/assign2.db';

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

create table assign1.assign1\_intern\_tab (eid int, ename string, age int, jobtype string, storeid int, storelocation string, salary bigint, yrsofexp int) row format delimited fields terminated by ',';

- What is the table path in HDFS?

describe formatted assign1.assign1\_intern\_tab;

the table path is hdfs://namenode:8020/user/hive/warehouse/assign1.db/assign1\_intern\_tab

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

First command: load data local inpath 'employee.csv' into table assign1.assign1\_intern\_tab;

Second command: !hadoop fs -put employee.csv /nayieraAssignment1;

load data inpath '/nayieraAssignment1' into table assign1.assign1\_intern\_tab;

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

select \* from assign1.assign1\_intern\_tab limit 10;

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

create external table assign1\_loc.assign1\_external\_tab (eid int, ename string, age int, jobtype string, storeid int, storelocation string, salary bigint, yrsofexp int) row format delimited fields terminated by ',' location '/nayieraAssignment1';

- What is the table path in HDFS?

describe formatted assign1\_loc.assign1\_external\_tab;

the table path is hdfs://namenode:8020/nayraAssignment1

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

!hadoop fs -put employee.csv /nayieraAssignment1;

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

drop table assign1.assign1\_intern\_tab;

drop table assign1\_loc.assign1\_external\_tab;

The data is still present on the hdfs after dropping the external table.

- recreate both tables

create table assign1.assign1\_intern\_tab (eid int, ename string, age int, jobtype string, storeid int, storelocation string, salary bigint, yrsofexp int) row format delimited fields terminated by ',';

create external table assign1\_loc.assign1\_external\_tab (eid int, ename string, age int, jobtype string, storeid int, storelocation string, salary bigint, yrsofexp int) row format delimited fields terminated by ',' location '/nayieraAssignment';

- list both table directories

describe formatted assign1.assign1\_intern\_tab;

hdfs://namenode:8020/user/hive/warehouse/assign1.db/assign1\_intern\_tab

describe formatted assign1\_loc.assign1\_external\_tab;

hdfs://namenode:8020/nayraAssignment1

- create internal table 'staging' inside the assign1 database

create table assign1.staging (eid int, ename string, age int, jobtype string, storeid int, storelocation string, salary bigint, yrsofexp int) row format delimited fields terminated by ',';

- load the staging table with the data from file employees

load data local inpath 'employee.csv' into table assign1.staging;

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

insert into assign1.assign1\_intern\_tab select \* from assign1.staging;

insert into table assign1\_loc.assign1\_external\_tab select \* from assign1.staging;

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

!hadoop fs -ls / hdfs://namenode:8020/user/hive/warehouse/assign1.db/assign1\_intern\_tab;

!hadoop fs -ls hdfs://namenode:8020/nayieraAssignment;

- count the lines inside the file songs

!wc -l songs.csv;

80 lines

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

create table songs\_tab (artist\_id string, artist\_latitude string, artist\_location string, artist\_longitude string, artist\_name string, duration string, num\_songs string, song\_id string, title string, year string) row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde';

load data local inpath 'songs.csv' into table songs\_tab;

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

select \* from songs\_tab limit 10;

- count the number of records

select count(\*) from songs\_tab;

80 lines

- 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. The data quality is okay.

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

create external table songs\_tab\_external (artist\_id string, artist\_latitude string, artist\_location string, artist\_longitude string, artist\_name string, duration string, num\_songs string, song\_id string, title string, year string) row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde' location '/nayiera2Assignment' ;

- load the table using put command

!hadoop fs -put songs.csv /nayiera2Assignment;

- is the data readable through the table? Why?

select \* from songs\_tab\_external limit 10;

Yes. It’s readable.

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

hive -S -e 'select \* from songs\_tab\_external limit 10'

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

drop table if exists songs\_tab\_external;

create external table songs\_tab\_external (artist\_id string, artist\_latitude string, artist\_location string, artist\_longitude string, artist\_name string, duration string, num\_songs string, song\_id string, title string, year string) row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde' location '/nayiera3Assignment' ;

load data local inpath 'songs.csv' into table songs\_tab\_external;

- execute it from shell without accessing hive CLI /beeline

hive -f script1.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.

Temporary tables don’t store data in the Hive warehouse directory instead the data get stored in the user’s scratch directory /tmp/hive/<user>/\* on HDFS.

If you create a temporary table in Hive with the same name as a permanent table that already exists in the database, then within that session any references to that permanent table will resolve to the temporary table, rather than to the permanent table.

Below is a simple example of creating a temporary table.

CREATE TEMPORARY TABLE emp.employee\_tmp (

id int,

name string,

age int,

gender string)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',';

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

use assign1;

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

- check the table directory and list its components

describe formatted assign1\_loc.assign1\_intern\_tab;

select \* from assign1\_loc.assign1\_intern\_tab;

**Assignment 2**

- Create a database named assign2

create database assign2;

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

create external table assign2.songs\_tab\_test (artist\_id string, artist\_latitude string, artist\_location string, artist\_longitude string, duration string, num\_songs string, song\_id string, title string) partitioned by (artist\_name string, year string) row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde' location 'hdfs://namenode:8020/nayieraassign2';

- Load data into table HDFS directory using put command

!hadoop fs -put songs.csv /nayieraassign2;

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

select \* from assign2.songs\_tab\_test;

No. The partitions can’t see the data.

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

alter table assign2.songs\_tab\_test add partition (artist\_name = 'afnan', year = '2020') location 'hdfs://namenode:8020/nayieraassign2/nayra';

- Load data to the created partitions

!hadoop fs -put songs.csv /nayieraassign2/nayra;

- List the partition directories to check for presence of files

describe formatted assign2.songs\_tab\_test;

- Create a staging table to host songs data

create table assign2.staging2 (artist\_id string, artist\_latitude string, artist\_location string, artist\_longitude string, artist\_name string, duration string, num\_songs string, song\_id string, title string, year string) row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde';

load data local inpath 'songs.csv' into table assign2.staging2;

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

set hive.exec.dynamic.partition.mode=nonstrict;

insert overwrite table assign2.songs\_tab\_test partition (artist\_name, year) select artist\_id, artist\_latitude, artist\_location, artist\_longitude, artist\_name, duration, num\_songs, song\_id, title, year from assign2.staging2;

- Truncate songs table and ensures no data in the table

drop table assign2.songs\_tab\_test;

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

create external table assign2.songs\_tab\_test (artist\_id string, artist\_latitude string, artist\_location string, artist\_longitude string, duration string, num\_songs string, song\_id string, title string) partitioned by (artist\_name string, year string) row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde' location 'hdfs://namenode:8020/nayieraassign2';

insert into table assign2.songs\_tab\_test partition (artist\_name, year) select artist\_id, artist\_latitude, artist\_location, artist\_longitude, artist\_name, duration, num\_songs, song\_id, title, year from assign2.staging2;

- Truncate

drop table assign2.songs\_tab\_test;

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

create table assign2.songs\_tab\_test (artist\_id string, artist\_latitude string, artist\_location string, artist\_longitude string, duration string, num\_songs string, song\_id string, title string) partitioned by (year string, artist\_name string) row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde';

insert into table assign2.songs\_tab\_test partition (year = '2020', artist\_name) select artist\_id, artist\_latitude, artist\_location, artist\_longitude, artist\_name, duration, num\_songs, song\_id, title from assign2.staging2 where year = '2020';

- 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\_xx like assign2.staging2 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\_xx like assign2.staging2 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 event\_tab(artist string, auth string, firstName string, gender string, itemInSession string, lastName string, length string, level string, location string, method string, page string, registration string, sessionId string, song string, status string, ts string, userAgent string,

userId string) 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 'events.csv' into table event\_tab;

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

select userId, sessionId, first\_value(song)over(partition by sessionId ORDER BY song ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING), last\_value(song)over(partition by sessionId ORDER BY song ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) from event\_tab;

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 userId, count(distinct song), RANK() OVER (Order BY COUNT(distinct song) DESC) FROM event\_tab group by userId;

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 userId, count(distinct song), Row\_number() OVER (Order BY COUNT(distinct song) DESC) FROM event\_tab where page = 'NextSong' group by userId;

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 event\_tab GROUP BY location, artist GROUPING SETS ((location,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 COUNT(song) FROM event\_tab GROUP BY location, artist GROUPING SETS ((location,artist),location, artist, ());

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 userId, song, LEAD(song, 1, 0) OVER (PARTITION BY userId ORDER BY song) next\_song FROM event\_tab;

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 event\_tab order by userId, song, ts;

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

select userId, song, ts from event\_tab cluster by userId, song, ts;