**Assignment 1**

- create a database named assign1

create database assign1\_mariem;

- What is the database path on HDFS?

describe database assign1\_mariem;

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

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

create database assign1\_loc location '/hp\_db/ assign1\_loc';

- 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\_mariem;

-create table if not exists 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 ','

lines terminated by '\n' ;

- What is the table path in HDFS?

describe formatted assign1\_intern\_tab ;

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

Text

Description automatically generated

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

1-

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

2-

!hadoop fs  -mkdir /employee\_data;

!hadoop fs -put employee.csv /employee\_data;

load data inpath '/employee\_data/employee.csv' into table assign1\_intern\_tab ;

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

select \* from assign1\_intern\_tab

limit 10;

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

use assign1\_loc;

create external table if not exists external\_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 ','

lines terminated by '\n'

location 'hdfs://namenode:8020/employee\_data';

- What is the table path in HDFS?

describe formatted external\_assign1\_intern\_tab;

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

!hadoop fs -put employee.csv /employee\_data;

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

drop table assign1\_intern\_tab ; (not present)

drop table external \_assign1\_intern\_tab; (present)

- recreate both tables

- use assign1\_mariem;

-create table if not exists 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 ','

lines terminated by '\n' ;

use assign1\_loc;

create external table if not exists external\_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 ','

lines terminated by '\n'

location 'hdfs://namenode:8020/employee\_data';

- list both table directories

describe formatted assign1\_intern\_tab;

(hdfs://namenode:8020/user/hive/warehouse/assign1\_mariem /assign1\_intern\_tab)

describe formatted external\_assign1\_intern\_tab;

(hdfs://namenode:8020/user/hive/warehouse/assign1\_mariem / external\_assign1\_intern\_tab)

- create internal table 'staging' inside the assign1 database

- use assign1\_mariem;

-create table if not exists staging ( eid int,

ename string,

age int,

jobtype string,

storeid int,

storelocation string,

salary bigint,

yrsofexp int

)

row format delimited

fields terminated by ','

lines terminated by '\n' ;

-load the staging table with the data from file employees

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

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

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

-insert into assign1\_loc.external\_assign1\_intern\_tab select \* from assign1\_mariem.staging;

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

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

(hdfs://namenode:8020/user/hive/warehouse/assign1\_mariem /assign1\_intern\_tab)

describe formatted assign1\_loc.external\_assign1\_intern\_tab;

(hdfs://namenode:8020/user/hive/warehouse/assign1\_mariem / external\_assign1\_intern\_tab)

- count the lines inside the file songs

select count(\*) from songs;

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

use assign1\_mariem;

create table if not exists songs (

artist\_id string,

artist\_latitude float,

artist\_location string,

artist\_longitude float,

artist\_name string,

duration float,

num\_songs int,

song\_id string,

title string,

year int

)

row format delimited

fields terminated by ','

lines terminated by '\n' ;

load data local inpath 'songs.csv' into table assign1\_mariem.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(\*) from songs;

count (80)

- is the hive count similar to the file count? Yes

is the data quality ok? No

If there is an issue, show how to resolve it

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

!hadoop fs  -mkdir /songs\_dir;

use assign1\_mariem;

create external table if not exists external\_songs (

artist\_id string,

artist\_latitude float,

artist\_location string,

artist\_longitude float,

artist\_name string,

duration float,

num\_songs int,

song\_id string,

title string,

year int

)

row format delimited

fields terminated by ','

lines terminated by '\n'

location 'hdfs://namenode:8020/songs\_dir';

load the table using put command

!hadoop fs -put songs.csv /songs\_dir;

- is the data readable through the table? Why?

NO.

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

hive -S -e 'select \* from assign1\_mariem.external\_songs limit 10';

hadoop fs -cat /songs\_dir/songs.csv;

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

DROP TABLE IF EXISTS assign1\_intern\_tab;

create table if not exists assign1\_intern\_tab (

emp\_id int ,

emp\_name string,

age int,

job\_title string,

dept\_id int,

city string,

salary int,

kilos\_from\_home int

) ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',';

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

- execute it from shell without accessing hive CLI /beeline

hive -f script.hdl

- What is a hive 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

-how can you create it?

CREATE TEMPORARY TABLE Table\_name()

why would someone use a temp table?

to store the data temporarily within an active session and the temporary tables get automatically removed when the active session end.

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

create database test;

use assign1\_mariem;

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

- check the table directory and list its components

describe formatted assign1\_intern\_tab ;

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

**Assignment 2**

1- Create a database named assign2

create database assign2;

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

!hadoop fs -mkdir -p /songs\_assign2/songs\_assign3;

create external table assign1\_mariem.songs\_external(

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 delimited

fields terminated by ','

lines terminated by '\n'

location 'hdfs://namenode:8020/songs\_assign2';

3- Load data into table HDFS directory using put command

!hadoop fs -put songs.csv /songs\_assign2/songs\_assign3;

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

select \* from songs\_external;

No data found..

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

alter table songs\_external add partition(artist\_name='mariem',year= '2022')

location 'hdfs://namenode:8020/songs\_assign2/songs\_assign3';

6- Load data to the created partitions

!hadoop fs -put songs.csv /songs\_assign2/songs\_assign3;

7- List the partition directories to check for presence of files

!hadoop fs -ls /songs\_assign2/songs\_assign3;

8- Create a staging table to host songs data

create table staging\_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 delimited

fields terminated by ','

lines terminated by '\n';

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

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

Insert overwrite table assign1\_mariem.songs\_external partition (artist\_name , year)

select artist\_id,

artist\_latitude,

artist\_location,

artist\_longitude,

artist\_name,

duration,

num\_songs,

song\_id,

title,

year

From staging\_tab;

10- Truncate songs table and ensures no data in the table

drop table assign1\_mariem.songs\_external;

Select \* from assign1\_mariem.songs\_external;

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

Insert overwrite table assign1\_mariem.songs\_external partition (artist\_name , year)

select artist\_id,

artist\_latitude,

artist\_location,

artist\_longitude,

artist\_name,

duration,

num\_songs,

song\_id,

title,

year

From staging\_tab;

12- Truncate

truncate table assign1\_mariem.songs\_external;

Select \* from assign1\_mariem.songs\_external;

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

drop table assign1\_mariem.songs\_external;

create table assign1\_mariem.songs\_external(

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 delimited

fields terminated by ','

lines terminated by '\n';

Insert overwrite table assign1\_mariem.songs\_external

partition (year='2007', artist\_name)

select artist\_id,

artist\_latitude,

artist\_location,

artist\_longitude,

artist\_name,

duration,

num\_songs,

song\_id,

title

From staging\_tab WHERE YEAR='2007';

14- 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

15-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

**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, song, sessionId, last\_value(song)over(partition by sessionId order by itemInSession ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED following), first\_value(song)over(partition by sessionId order by itemInSession ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED following)

from event\_tab limit 20;

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 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 sessionId, userId,lead(song) over

(partition by userId order by sessionId desc) from event\_tab

where userId= '98'

order by sessionId desc;

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

select userId,song ,ts from event\_tab

cluster by userId, song, ts;