**Task 1**

**Data:** Dataset1

**Instructions:**

* Create database targetdb at HDFS location /data/target/targetdb

hadoop fs -mkdir /data/target

hadoop fs -mkdir /data/target/targetdb

CREATE DATABASE targetdb LOCATION '/data/target/targetdb';

use targetdb;

* Create tables: movies, ratings, tags in targetdb database

create table movies (movieId int, title string, genres string) row format delimited fields terminated by ',';

create table ratings (userId int,movieId int,rating double,t\_stamp decimal(12,0)) row format delimited fields terminated by ',';

create table tags (userId int,movieId int,tag string,t\_stamp decimal(12,0)) row format delimited fields terminated by ',';

* Load the relevant data files to those tables

impala-shell

invalidate metadata;

hadoop fs -put /home/cloudera/Desktop/Exercise\_7/Dataset1/movies.csv /data/target/targetdb/movies/

hadoop fs -put /home/cloudera/Desktop/Exercise\_7/Dataset1/ratings.csv /data/target/targetdb/ratings/

hadoop fs -put /home/cloudera/Desktop/Exercise\_7/Dataset1/tags.csv /data/target/targetdb/tags/

Refresh movies; --- this command needds to be executed after the put commands above, so that meta data can be refreshed.

Select \* from movies limit 5;

///--- no need to start – a different way to load data

hadoop fs -copyFromLocal /home/cloudera/Desktop/Exercise\_7/Dataset1/\* /data/source/dataset1

hadoop fs -copyFromLocal /home/cloudera/Desktop/Exercise\_7/Dataset2/\* /data/source/dataset2

hadoop fs -copyFromLocal /home/cloudera/Desktop/Exercise\_7/Dataset3/\* /data/source/dataset3

LOAD DATA INPATH '/data/source/dataset1/movies.csv' INTO TABLE movies;

LOAD DATA INPATH '/data/source/dataset1/ratings.csv' INTO TABLE ratings;

LOAD DATA INPATH '/data/source/dataset1/tags.csv' INTO TABLE tags;

///--- no need to end

* If multiple source files, then files can be copied first using hdoop fs -put

1. Use hadoop move comment

Hadoop fs –mv sourcexx /data/target/targetdb/movies

1. Or under impala

Load data in path from the source into table

Hadoop fs –chmod –R 777 /data

* Prepare a SQL to load movie\_stats table.

Following is the column detail for movie\_stats table.

movieid, movie\_name,avg\_rating,hash\_tag,hash\_tag\_cnt

**Note**: Choose appropriate column data type as per your knowledge.

desc movie\_stats

truncate table movies\_stats;

* Check SQL:
  + summary
  + profile

INSERT INTO movie\_stats

Select t1.\*,  t2.tag, t2.tag\_cnt from

(select m.movieid, m.title, CAST(avg(r.rating) AS DECIMAL(9,2)) from movies m join ratings r on m.movieid = r.movieid  group by m.movieid, m.title) t1

JOIN

(select  movieid, tag, CAST(count(tag) AS INT) as tag\_cnt from tags group by movieid, tag) t2

ON

T1.movieid = t2.movieid

INSERT INTO movie\_stats2

SELECT m.movieid, m.title, CAST(avg(r.rating) over(partition by m.movieid) AS DECIMAL(9,2)) , tag, CAST(count(tag) over(partition by t.movieid, t.tag) AS INT)

FROM

    Movies m

JOIN

    Ratings r

ON

    M.movieid = r.movieid

JOIN

    Tags t

ON

    M.movieid = t.movieid

create table movies2 like movies;

insert into movies2 values(1, 'Avenger', null);i

create table ratings2 like ratings;

insert into ratings2 values(1,1,3,null);

insert into ratings2 values(2,1,4,null);

insert into ratings2 values(3,1,5,null);

create table tags like tags;

insert into tags2 values(1,1, 'Action',null);

insert into tags2 values(2,1, 'Action',null);

insert into tags2 values(3,1, 'Action',null);

* 1 record returned

Select t1.\*,  t2.tag, t2.tag\_cnt from

(select m.movieid, m.title, CAST(avg(r.rating) AS DECIMAL(9,2)) from movies2 m join ratings2 r on m.movieid = r.movieid  group by m.movieid, m.title) t1

JOIN

(select  movieid, tag, CAST(count(tag) AS INT) as tag\_cnt from tags2 group by movieid, tag) t2

ON

T1.movieid = t2.movieid

-- 9 records returned

SELECT m.movieid, m.title, CAST(avg(r.rating) over(partition by m.movieid) AS DECIMAL(9,2)) , tag, CAST(count(tag) over(partition by t.movieid, t.tag) AS INT)

FROM

    Movies2 m

JOIN

    Ratings2 r

ON

    M.movieid = r.movieid

JOIN

    Tags2 t

ON

    M.movieid = t.movieid

--9 record returned

SELECT m.movieid, m.title, CAST(avg(r.rating) over(partition by m.movieid) AS DECIMAL(9,2)) , tag, CAST(count(tag) over(partition by t.movieid, t.tag) AS INT)

FROM

    Movies2 m

Left JOIN

    Ratings2 r

ON

    M.movieid = r.movieid

Left JOIN

    Tags2 t

ON

    M.movieid = t.movieid

=======================================================================

**Task 2**

Let’s apply some integrity constraints to our table

* Modify movies table to make column movieId UPI and NOT NULL

--- not supported in Hive or Impala

=======================================================================

**Task 3**

**Data:** Dataset2

**Instructions:**

* Create new database stagedb at location /data/staging/stagedb
* Create tables movies, ratings, tags under database stagedb with same details as targetdb tables

create table movies like targetdb.movies;

create table ratings as select \* from targetdb.ratings limit 0

create table ratings like targetdb.ratings;

create table tags like targetdb.tags;

select \* from movies where movieid = 123 --- the process will look in the all the files under the folder and scan all of them, which will be slow. If the table is constructed by many files and data volume is huge. This will be slow. --- > in this case, the table partition (maybe be date) technic can be used.

* Prepare three different scripts, one for each to load stage tables with given data files considering all possible integrity checks as
  + Movie ID is unique and not null field. Hence have to check for any unique key as well as not null violation
  + Rating should be for a valid movie. Hence movieId in rating data file should be present in Movie table
  + If there is any exact duplicate record then we must discard one.

=======================================================================

**Task 4**

**Instructions:**

* Prepare full refresh load script to refresh movie\_stats table

=======================================================================

**Task 5**

**Data:** Dataset 3

**Instructions:**

* Movies table is Slowly Changing Dimension (SCD) type 2 table.
* Generally, we receive Insert/Update/Delete flag with source file to load Slowly Changing Dimensions (SCD)
* Modify the movies table load script to implement IUD logic

=======================================================================

**Task 6**

**Instructions:**

We have some clients using MySQL database to generate reports. Hence we need to push our movie\_stats table to that MySQL database.