Manipulating Data With Hive

Test data (HDFS):

movie

movierating

Exercise directory: ~/workspace/hive

**In this exercise, you will practice data processing in Hadoop using Hive.**

Go to movielens.sql and verify everything looks good and then execute the following command:

$ sudo mysql < movieslens.sql

The data sets for this exercise are the movie and movierating data imported from MySQL into Hadoop in the “Importing Data with Sqoop” exercise

**Review the Data**

1. Make sure you’ve completed the “Importing Data with Sqoop” exercise. Review the data you already loaded into HDFS in that exercise:

$ hdfs dfs -cat movie/part-m-00000 | head

...

$ hdfs dfs -cat movierating/part-m-00000 | head

**Prepare The Data For Hive**

For Hive data sets, you create *tables*, which attach field names and data types to your Hadoop data for subsequent queries. You can create *external* tables on the movie and movierating data sets, without having to move the data at all.

Prepare the Hive tables for this exercise by performing the following steps:

1. Invoke the Hive shell, or use Hue:

$ hive

1. Create the movie table:

hive>

CREATE EXTERNAL TABLE movie

(id INT, name STRING, year INT)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

LOCATION '/user/[username]/movie';

4. Create the movierating table:

hive>

CREATE EXTERNAL TABLE movierating (userid INT, movieid INT, rating INT)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

LOCATION '/user/[username]/movierating';

1. Quit the Hive shell:

hive> QUIT;

**Practicing HiveQL**

If you are familiar with SQL, most of what you already know is applicably to HiveQL. Skip ahead to section called “The Questions” later in this exercise, and see if you can solve the problems based on your knowledge of SQL.

If you are unfamiliar with SQL, follow the steps below to learn how to use HiveSQL to solve problems.

1. Start the Hive shell.
2. Show the list of tables in Hive:

hive> SHOW TABLES;

The list should include the tables you created in the previous steps.

NOTE:By convention, SQL (and similarly HiveQL) keywords are shown in uppercase. However, HiveQL is not case sensitive, and you may type the commands in any case you wish.

1. View the metadata for the two tables you created previously:

hive> DESCRIBE movie;

hive> DESCRIBE movieratings;

Hint: You can use the up and down arrow keys to see and edit your command history in the hive shell, just as you can in the Linux command shell.

1. The SELECT \* FROM *TABLENAME* command allows you to query data from a table. Although it is very easy to select *all* the rows in a table, Hadoop generally deals with very large tables, so it is best to limit how many you select. Use LIMIT to view only the first *N* rows:

hive> SELECT \* FROM movie **LIMIT 10**;

1. Use the WHERE clause to select only rows that match certain criteria. For example, select movies released before 1930:

hive> SELECT \* FROM movie **WHERE year < 1930**;

1. The results include movies whose year field is 0, meaning that the year is unknown or unavailable. Exclude those movies from the results:

hive> SELECT \* FROM movie WHERE year < 1930 **AND year != 0**;

1. The results now correctly include movies before 1930, but the list is unordered. Order them alphabetically by title:

hive> SELECT \* FROM movie WHERE year < 1930 AND year != 0 **ORDER BY name**;

1. Now let’s move on to the movierating table. List all the ratings by a particular user, e.g.

hive> SELECT \* FROM movierating **WHERE userid=149**;

1. SELECT \* shows all the columns, but as we’ve already selected by userid, display the other columns but not that one:

hive> **SELECT movieid,rating** FROM movierating WHERE userid=149;

1. Use the JOIN function to display data from both tables. For example, include the name of the movie (from the movie table) in the list of a user’s ratings:

hive> select movieid,rating,**name** from movierating **join**

**movie on movierating.movieid=movie.id** where userid=149;

1. How tough a rater is user 149? Find out by calculating the average rating she gave to all movies using the AVG function:

hive> SELECT **AVG(rating)** FROM movierating WHERE userid=149;

1. List each user who rated movies, the number of movies they’ve rated, and their average rating.

hive> SELECT userid, COUNT(userid),AVG(rating) FROM movierating GROUP BY userid;

13. Take that same data, and copy it into a new table called userrating.

hive> CREATE TABLE USERRATING (userid INT,

numratings INT, avgrating FLOAT);

hive> **insert overwrite table userrating**

SELECT userid,COUNT(userid),AVG(rating)

FROM movierating GROUP BY userid;

Now that you’ve explored HiveQL, you should be able to answer the questions below.

**The Questions**

Now that the data is imported and suitably prepared, write a HiveQL command to implement each of the following queries.

**Working Interactively or In Batch**

**Hive:**

You can enter Hive commands interactively in the Hive shell:

$ hive

. . .

hive> *Enter interactive commands here*

Or you can execute text files containing Hive commands, or use Hue:

$ hive -f *file\_to\_execute*

1. What is the oldest known movie in the database? Note that movies with

unknown years have a value of 0 in the year field; these do not belong in your answer.

1. List the name and year of all unrated movies (movies where the movie data has no related movierating data).
2. Produce an updated copy of the movie data with two new fields:

numratings -‐ the number of ratings for the movie

avgrating -‐ the average rating for the movie

Unrated movies are not needed in this copy.

1. What are the 10 highest-‐rated movies? (Notice that your work in step 3 makes this question easy to answer.)

Note: The solutions for this exercise are in ~/workspace/hive.

**END**