Hive Introduction

DS 730

Overview

In this activity, you will be running some Hive code on the local Hortonworks system and on AWS.

Activity Tasks

Task 1: Move Files

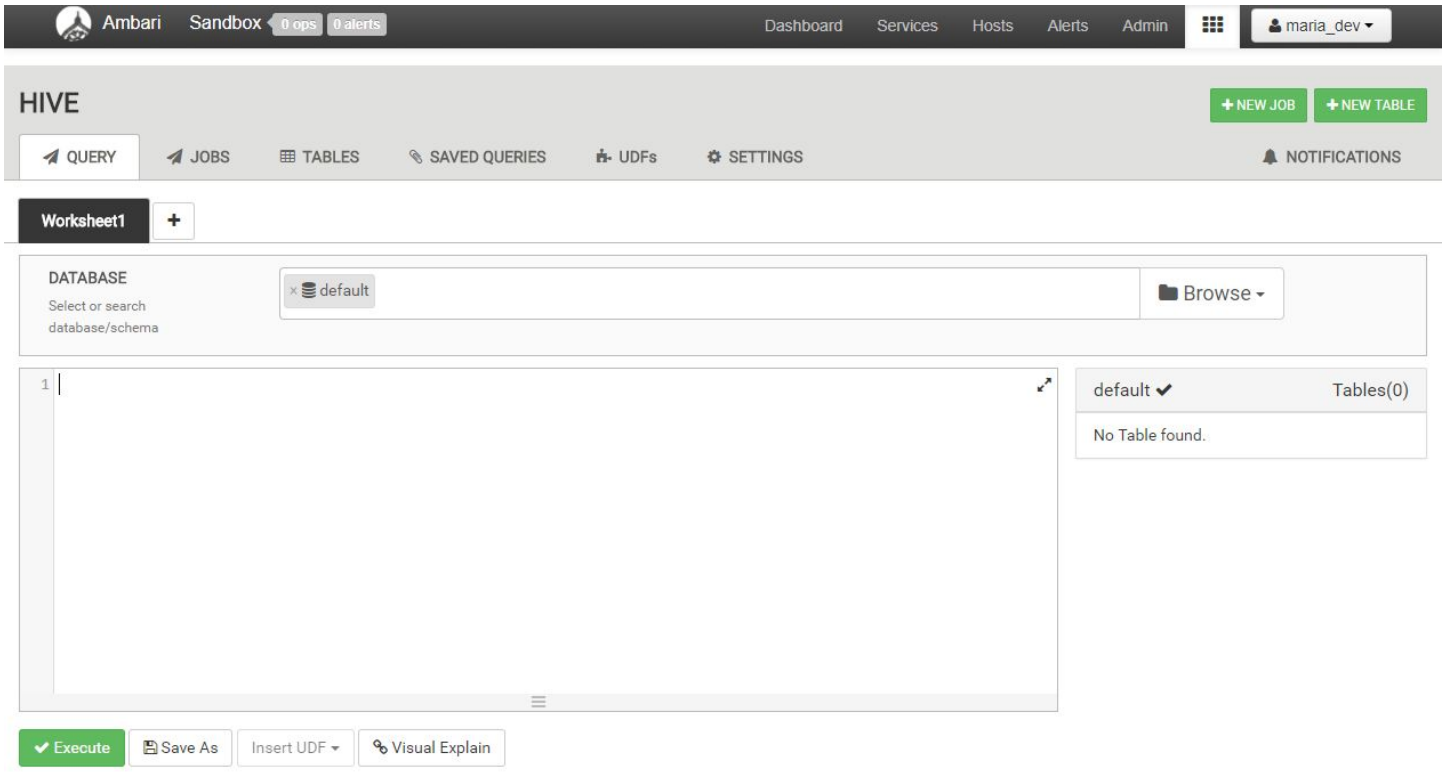
1. Retrieve the following files and decompress it:
   http://www.uwosh.edu/faculty_staff/krohne/ds730/hiveFiles.tar.gz
2. Move the decompressed files to the HDFS so that the hierarchy looks like this:
   /user/maria_dev/hivetest/numbers.txt
   /user/maria_dev/hivetest/Batting.csv
   /user/maria_dev/hivetest/master/Master.csv

Task 2: Work With Internal Tables

1. First, open up the csv files using Microsoft Excel so you can get a handle on what is saved in each file. As we are solving each of these problems, you should think about how difficult it would be to do this using MapReduce. Similarly, if your filesize is on the order of terabytes or higher, Microsoft Access or some other software would struggle to even load it.

We will spend quite a bit of time going through the HiveQL syntax now. Each step will contain a statement and an explanation of the statement.

2. Go to the dashboard (http://localhost:8080) and login using maria_dev. Click on the 9 squares in the upper right hand corner and click on **Hive View 2.0**. You should see something like this image. Everything that is entered will be in the textarea where the cursor is (above the Execute button and to the right of the 1). For each command, do the following: enter the command, hit the execute button, delete the command from the textarea. You do not want to execute a command more than once.



3. **`CREATE TABLE favnumbers(num INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ' LINES TERMINATED BY '\n';`**

   Explanation: This will create a new table that has one column: an integer column. The fields of our table are terminated by a space and our lines are terminated by a new line character. The favnumbers table may not show up in the UI on the right hand side immediately. To ensure the table was created, click on the **Tables** tab. The favnumbers table should be listed. Then click back on the **Query** tab. On the right hand side under **default**, it should now say favnumbers.

4. To see what this query actually did, go to the Ambari Files View and navigate to this folder: /apps/hive/warehouse. You will see a favnumbers folder there. This is where our data will actually be stored once we load our data into the table.
5. **`DESCRIBE favnumbers;`**

Explanation: This statement will essentially give us the schema for this table. The Results tab on the bottom will display our column name (**num**) and its type (**int**).

6. **`LOAD DATA INPATH 'hdfs://sandbox-hdp.hortonworks.com/user/maria_dev/hivetest/numbers.txt' INTO TABLE favnumbers;`**

   Explanation: This command loads the data from our /user/maria_dev/hivetest folder and and stores it in our ***internal*** table. Our numbers are separated by a new line and this was accounted for when we created our table before when we used the **`LINES TERMINATED BY '\n'`**. Be careful with **`LOAD`** as we are ***moving*** the **numbers.txt** file to the /**app/hive/warehouse** folder. We are ***not*** making a copy. This is especially important when we drop our table below. Dropping a table will delete the folder and all data in that folder.

7. **`SELECT * FROM favnumbers;`**

   Explanation: A simple HiveQL statement to return all of the rows in the favnumbers table.

8. **`SELECT count(*) FROM favnumbers;`**

   Explanation: There are many built in functions that we can use to help with aggregation. **`count`** obviously tells us how many rows are in our table. There are 50 of them.

9. **`SELECT * FROM favnumbers WHERE num>100;`**

   Explanation: Another simple Hive QL statement to return all of the numbers that are greater than 100. We can also count how many are greater than 100. There are 24 numbers greater than 100.

10. **`DROP TABLE favnumbers;`**

    Explanation: Since we no longer need this (mostly) useless table, we will drop it. If you look at the Ambari Files View for /apps/hive/warehouse, you'll notice the favnumbers folder has been deleted. Not only that, since we loaded our data the way we did, the numbers.txt file no longer exists in the /user/maria_dev/hivetest folder.

> **Important:** Be careful when dropping an internal table, as ***the data in the folder will be deleted***.

Task 3: Work With External Tables and Partitions

The next part deals with using an external table and also creating a partition. A partition is what it sounds like: it's taking our original table and splitting it up into smaller pieces. This is very important for speeding up our queries. It is never a necessary thing to do but is highly recommended.

1. **`CREATE EXTERNAL TABLE batting(id STRING, year INT, team STRING, league STRING, games INT, ab INT, runs INT, hits INT, doubles INT, triples INT, homeruns INT, rbi INT, sb INT, cs INT, walks INT, strikeouts INT, ibb INT, hbp INT, sh INT, sf INT, gidp INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION '/user/maria_dev/hivetest/batting';`**

   Explanation: This creates an ***external*** table. The difference is that when we drop our table, the data inside of that folder will ***not*** be deleted. However, when we load our data, it is still moved to this folder.

2. **`LOAD DATA INPATH 'hdfs://sandbox-hdp.hortonworks.com/user/maria_dev/hivetest/Batting.csv' INTO TABLE batting;`**

   Explanation: This command will move the file from **hdfs:/user/maria_dev/hivetest/Batting.csv** to our location at **hdfs://user/maria_dev/hivetest/batting/Batting.csv**. Since our table is external, if we drop our table, the data will **not** be deleted. A better solution would be to specify the location of our

data when we create the table instead of having to load the data with this command (we will do this later with the Master table). By doing it that way, Hive builds its tables on top of our data without having to move the data. You should never use the LOAD DATA INPATH command. It is shown in this step to show you what is possible, not what is recommended.

3. **`CREATE EXTERNAL TABLE split_bat(id STRING, runs INT) PARTITIONED BY(year INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION '/user/maria_dev/hivetest/split_bat';`**

   Explanation: This will create a new table called **split_bat** that is similar to the original batting table. The difference is that we only want a subset of the original data and we want to partition the data by year. What this will do is allow us to split up the data into separate folders to decrease the query times of future select statements. If a lot of our queries need the id and runs and depend on looking at a specific year, then it makes sense to split up the data so that we only need to look at relevant rows in our table. We could certainly add more columns to our table but those two are sufficient for demonstration. Once we have our table created, we need to load up data into that table by…

4. The following two commands must be executed serially:

   `SET hive.exec.dynamic.partition.mode=nonstrict;`

   `INSERT OVERWRITE TABLE split_bat PARTITION(year) SELECT id, runs, year FROM batting WHERE year<1950;`

   Explanation:

   - A variable that needs to be set to nonstrict in order for our partitioning to work dynamically.
   - The **`INSERT`** is self-explanatory.
   - The **`OVERWRITE`** is there in case there is data already in our table.
   - We are inserting into the **split_bat** table and we are partitioning our data by year. It is a bit strange to say the **split_bat** is a table because it is better to think of it as a collection of tables. Inside of our **split_bat** folder, we have many folders that look like this:

     year=1948

     year=1930

     year=1901

     and so on.

Our data is partitioned in these folders based on the year.

- The **`WHERE`** clause is there because of an unfortunate issue with MapReduce, Hive and our virtual machine. If we create too many partitions dynamically, we might end up using too much memory and the MapReduce program will fail. If you omit the **`WHERE`** clause, your insert will likely fail. This is fine though, because we can just create two insert calls to insert all of the data (i.e. change the year<1950 to be year>=1950 in the second insert, see step 6).
- When we query our **split_bat** table, we can query it as we normally would with a **`SELECT * FROM split_bat`** and we would get back all of the data. However, if we run the following query…

5. **`SELECT * FROM split_bat WHERE year=1940;`**

   Explanation: …now we are only querying inside of that smaller folder that only contains rows that have a year of 1940. Since our dataset is on the order of megabytes, we will not see a huge difference in query time. However, if our dataset were gigabytes or petabytes in size, this partitioning would speed up our execution by a fair amount. This is a classic space vs. time issue. Our speed increases but we are paying for it with duplicated space.

6. **`SET hive.exec.dynamic.partition.mode=nonstrict;`**

   `INSERT OVERWRITE TABLE split_bat PARTITION(year) SELECT id, runs, year FROM batting WHERE year>=1950;`

   Explanation: Both commands must be executed in the same execution. Identical to step 4 except now all of the years appear in the split_bat folder.

Task 4: Create a View and Index

The previous task created a whole new table and copied over a bunch of data into a new folder. If we are only looking at selecting a certain subset of the data and we do it often enough, creating a view might be a better solution because we won't be duplicating data. In this task, we will create a view and then create an index to help speed up our queries.

1. **`CREATE VIEW hits AS SELECT id, year, doubles, triples FROM batting WHERE doubles >= 20 AND triples >= 20;`**

   Explanation: This creates a view of our table without having to create a new folder to store all of the data. Our new view contains only the relevant columns that we want and also limits the rows that are in that table by rows that have doubles and triples greater than or equal to 20. We can now treat our view just like it is a table…

2. **`SELECT * FROM hits;`**

   Explanation: Assuming all went well, we should have 96 rows in our view.

All of the execution done with Hive so far have been done using the Tez engine. The specific details of what that means are beyond this course. However, for those of you who are familiar with indexes in SQL, know that the Tez engine does not support indexes. Therefore, all of the commands that deal with indexes must contain the following code at the top of the script to change the engine to MapReduce:
SET hive.execution.engine=mr;

Unless you have a pressing reason to use mr, you should stick with the Tez engine. Tez is, in general, faster.

3. **`CREATE INDEX year_index ON TABLE batting(year) AS 'COMPACT' WITH DEFERRED REBUILD;`**

   Explanation: If we have a lot of queries that use the same columns in our tables, it might be better to create an index for each of the columns we are using often. This step is never necessary but it will often speed up the query time similar to what a partition does for us. The question becomes: when do you use a partition, when do you use a view and when do you use an index? The answer is, of course, it depends. Even though the content is geared towards an Oracle database, this somehwat describes the differences: https://docs.oracle.com/cd/E18283_01/server.112/e16541/partition.htm

   The short and simple answer is that partitioning actually creates new tables for you so you pay for the partition with space. However, a partition is almost certainly faster than an index. An index will not create a new table. The deferred rebuild tells Hive to wait to do the indexing until another time. Indexing is generally not a fast process. You may want to wait until an off-time to index the table, which can be done with this statement…

4. **`ALTER INDEX year_index ON batting REBUILD;`**

   Explanation: Our table isn't that large so the indexing will not take that long.

5. **`SHOW INDEX ON batting;`**

   Explanation: Shows all of the indices we created for our table.

Task 5: Join Tables

Our last task in Hive will be to load up the **Master.csv** file and join two tables together. This will be very similar to SQL and you should have no trouble understanding most of the statements in this task.

1. **CREATE EXTERNAL TABLE master(id STRING, byear INT, bmonth INT, bday INT, bcountry STRING, bstate STRING, bcity STRING, dyear INT, dmonth INT, dday INT, dcountry STRING, dstate STRING, dcity STRING, fname STRING, lname STRING, name STRING, weight INT, height INT, bats STRING, throws STRING, debut STRING, finalgame STRING, retro STRING, bbref STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION '/user/maria_dev/hivetest/master';**
2. **SELECT * FROM master;**

   Explanation: You'll notice that we did not have to load any data into the master table before using it. This is because of what we did initially when we moved Master.csv into the master folder: **/user/maria_dev/hivetest/master/Master.csv.** Because of this, when we created the external table in the previous step, we were simply laying a schema on top of existing data.

3. **SELECT b.id, b.year, b.runs FROM batting b JOIN (SELECT year, max(runs) AS best FROM batting GROUP BY year) o WHERE b.runs=o.best AND b.year=o.year;**

   Explanation: Similar to regular SQL, the **batting b** allows us to shorten the length of our query. Instead of having to use batting all the time in our query, we can just use b. This was why we put an "o" after the (SELECT year, max(runs)...) subquery so that this inner query has a name we can reference elsewhere. This query will display the ids of the player who scored the most runs in a particular year. If there are duplicates (see 1966 for example), then both players will show up. We can make that query even more complicated by printing out the names instead of the id by doing this…

4. **SELECT n.fname, n.lname, x.year, x.runs FROM master n JOIN (SELECT b.id as id, b.year as year, b.runs as runs FROM batting b JOIN (SELECT year, max(runs) AS best FROM batting GROUP BY year) o WHERE b.runs=o.best AND b.year=o.year) x ON x.id=n.id ORDER BY x.runs DESC;**

   Explanation: This query was created partially to show you how difficult it can be if you don't name your tables well. Using table names like b, o, x and n are not really the best idea. If the

query is much longer or if you have to share this query with others, determining what each of these tables are can be difficult. Unless it is a one-time query that you will throw away immediately after getting your answer, the best practice is to give meaningful names to your variables like you would any other programming language.

Task 6: Create a Script

1. It is a good idea to drop the table you are planning on using in your script so you are using the correct table in your query. But be careful doing this in a production setting. You don't want to drop a table someone else is using, especially if that table refers to an internal table. Your output will be printed to the terminal:

```
DROP TABLE IF EXISTS batting;
CREATE EXTERNAL TABLE IF NOT EXISTS batting(id STRING, year INT,
team STRING, league STRING, games INT, ab INT, runs INT, hits
INT, doubles INT, triples INT, homeruns INT, rbi INT, sb INT, cs
INT, walks INT, strikeouts INT, ibb INT, hbp INT, sh INT, sf
INT, gidp INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/user/maria_dev/hivetest/batting';
DROP TABLE IF EXISTS master;
CREATE EXTERNAL TABLE IF NOT EXISTS master(id STRING, byear INT,
bmonth INT, bday INT, bcountry STRING, bstate STRING, bcity
STRING, dyear INT, dmonth INT, dday INT, dcountry STRING, dstate
STRING, dcity STRING, fname STRING, lname STRING, name STRING,
weight INT, height INT, bats STRING, throws STRING, debut
STRING, finalgame STRING, retro STRING, bbref STRING) ROW FORMAT
DELIMITED FIELDS TERMINATED BY ',' LOCATION
'/user/maria_dev/hivetest/master';
SELECT n.fname, n.lname, x.year, x.runs FROM master n JOIN
(SELECT b.id as id, b.year as year, b.runs as runs FROM batting
b JOIN (SELECT year, max(runs) AS best FROM batting GROUP BY
year) o WHERE b.runs=o.best AND b.year=o.year) x ON x.id=n.id
ORDER BY x.runs DESC;
```

Task 7: Adjust Script

1. In order to run our Hive example on AWS, we need to make a few tweaks to the above script and also make sure our data is in the right location. This script adds a location to store our final relation. Create a file called **baseball.q** and store this into it:

```
DROP TABLE IF EXISTS batting;
CREATE EXTERNAL TABLE IF NOT EXISTS batting(id STRING, year INT,
team STRING, league STRING, games INT, ab INT, runs INT, hits
INT, doubles INT, triples INT, homeruns INT, rbi INT, sb INT, cs
INT, walks INT, strikeouts INT, ibb INT, hbp INT, sh INT, sf
INT, gidp INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://yourS3Bucket/batters';
DROP TABLE IF EXISTS master;
CREATE EXTERNAL TABLE IF NOT EXISTS master(id STRING, byear INT,
bmonth INT, bday INT, bcountry STRING, bstate STRING, bcity
STRING, dyear INT, dmonth INT, dday INT, dcountry STRING, dstate
STRING, dcity STRING, fname STRING, lname STRING, name STRING,
weight INT, height INT, bats STRING, throws STRING, debut
STRING, finalgame STRING, retro STRING, bbref STRING) ROW FORMAT
DELIMITED FIELDS TERMINATED BY ',' LOCATION
's3://yourS3Bucket/master';
INSERT OVERWRITE DIRECTORY 's3://yourS3Bucket/output' SELECT
n.fname, n.lname, x.year, x.runs FROM master n JOIN (SELECT b.id
as id, b.year as year, b.runs as runs FROM batting b JOIN
(SELECT year, max(runs) AS best FROM batting GROUP BY year) o
WHERE b.runs=o.best AND b.year=o.year) x ON x.id=n.id ORDER BY
x.runs DESC;
```

Task 8: Run Script

Store the script on your S3 drive in some folder.

1. Setup your S3 drive correctly (i.e. upload your **Batting.csv** and **Master.csv** file to the right folder on your S3 bucket).
2. Once your bucket is set up, go back to the AWS console, go to EMR, and create a cluster as we've done before.

3. Once the cluster has been created, add a step to the cluster as we've done before:
   - The **Step Type** will be a **Hive program.**
   - Let's assume you uploaded your **baseball.q** script to **s3://yourS3Bucket/baseball.q**. If so, then set the **Script S3 location** to be that location.
   - You do not need to edit or add anything else.
4. Click the **Add** button and run your script.
   - In a minute, you should see your output file stored in **s3://yourS3Bucket/output**.

Note: The preceding examples were just that--examples. There is nothing to submit from the above tasks. For more information on anything else you may want to know about Hive, post to the discussion board or go to https://hive.apache.org/.

Task 9: Solve Problems

Solve the following problems using the batting and master table and create a script to solve each problem. You should assume that:

- **Batting.csv** will be stored in this **hdfs** folder: **/user/maria_dev/hivetest/batting**
- **Master.csv** will be stored in this **hdfs** folder: **/user/maria_dev/hivetest/master**

Make sure to load your tables correctly in each script given that information. You should create 1 script per problem and call your script **hiveX.q** where **X** is the question number you are solving. Do not assume that tables have already been created in each script. Also do not assume that no tables have been created (i.e. drop table XYZ before creating table XYZ).

Since we already solved these problems for the Pig activity, you should already know the answers to these problems. It is now just a matter of writing the Hive code to do it. For each question, print out the playerID(s) of the player(s) who answer the question. If there are any ties, report all players who tied.

Problem #	Description
1	Who was the heaviest player to hit more than 5 triples (3B) in 2005?
2	In the batting file, if a player played for more than 1 team in a season, that player will have his name show up in multiple tuples with the same year. For example, in 2011, Francisco Rodriguez (**rodrifr03**) played for the New York Mets and then played for the Milwaukee Brewers (see tuples **95279** and **95280**). The question you have to answer is this: what player played for the most teams in any single season? A player may have played for the

	same team twice in the same season at different times in the season. If this is the case, you should count this as two different teams.
3	What player had the most extra base hits during the entire 1980's (1980 to 1989)? Note that this question is not asking about any 1 specific year. It is asking about the entire 10 year span in the 80's. An extra base hit is a double, triple or home run (columns **2B**, **3B**, **HR**).
4	Of the right-handed batters who were born in October and died in 2011, which one had the most hits in his career? The column with the heading of **H** is the hits column. Do not consider switch hitters to be right-handed batters.

Submitting Your Work

1. Submit the the answers to the four problems in Task 9 in a single text file called `answers.txt`
2. Submit the Hive scripts containing Hive code you used to get your answers stored in files called:
    - `hive1.q`
    - `hive2.q`
    - `hive3.q`
    - `hive4.q`

When you are finished, create a zip file containing all of the above folders/files and upload the resulting `a5.zip` file to the **Activity 5 dropbox**.