

CSE3BDC/CSE5BDC

Lab 04: Hive, Part 2

Department of Computer Science and IT, La Trobe University

Objectives

- Reinforce skills related to basic Hive syntax and execution
- Learn more advanced Hive techniques for optimising queries

This lab will again require the use of the Cloudera VM Image that is used throughout this course. You will also be required to apply knowledge from the previous lab on Hive—**so if you have not completed last week's lab, please do so before continuing.**

Task 1: Web Traffic Analysis

You have gained some familiarity with basic Hive syntax and execution; using grouping, ordering, and joins in Lab 03. Now it's time to practice these skills further, with a more useful dataset.

Included in the directory "Input_data/1/" is a set of data files that define four sets of data:

- A set of domains in the file `domains.csv`, comprised of URL and category. Below is an example of the data in the file:

URL	category
www.facebook.com	Social
www.netflix.com	Entertainment
australia.gov.au	Government

- A set of unique IPv4 addresses and their integer representations, used for comparing to other IP address ranges, in the file `ips.csv`. Below is an example of the data in the file:

ipAddress	intAddress
75.73.40.6	1263085574
223.23.156.177	3742866609
91.230.237.94	1541860702

- A set of IPv4 address ranges (min and max), their integer representations (min and max), as well as their region of origin, in the file `regions.csv`. Below is an example of the data in the file.

ipMin	ipMax	intMin	intMax	regionCode	regionName
1.0.0.0	1.0.0.255	16777216	16777471	AU	Australia
1.0.1.0	1.0.3.255	16777472	16778239	CH	China
1.0.4.0	1.0.7.255	16778240	16779263	AU	Australia

- A set of traffic history over the month of February, comprised of URL, IPv4 Address, and a timestamp of occurrence, in the file `traffic.csv`.

URL	ipAddress	time
www.facebook.com	15.116.12.84	2014-02-12 04:25:43
www.netflix.com	35.229.57.93	2014-02-10 12:01:31
australia.gov.au	47.18.175.209	2014-02-06 01:10:38

You will use Hive to join and group these data sets by common keys in order to analyse the data and produce sets of meaningful output.

The datasets are presented in .csv format (comma separated value), which means that there is one row of data per line and each column of data is separated by a comma. You will need to set Hive's default delimiter when creating each table by specifying at the end of the create table statement:

```
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

Task 1A: Load the CSV Data

Because Hive stores its data in tables on HDFS when it loads from a file, it is useful to spend some initial time reading the raw dataset into Hive tables. As such, you will first be required to create a number of tables, one for each dataset, and then load data from the corresponding .csv files provided. Open the file `t1-loaddata.hql` in a text editor to complete the below tasks.

1. **This one has been done for you as an example:** Load the contents of the domains data file, `domains.csv`, to a table called `mydomains`, which stores a web URL as a string, and a category as a string.
2. Load the contents of the IPv4 data file, `ips.csv`, to a table called `myips`, which stores an `ipAddress` as a string, and the IP integer `intAddress` as a BIGINT—this is very important to avoid integer overflow.
3. Load the contents of the IP regions data file, `regions.csv`, to a table called `myregions`, which stores two IPv4 addresses (`ipMin` and `ipMax`) as strings, two IPv4 integer (`intMin` and `intMax`) represented as BIGINTs, a `regionCode` as a string, and a `regionName` as a string.
4. Load in the contents of the traffic data file, `traffic.csv`, to a table called `mytraffic`, which stores a web URL as a string, an `ipAddress` as a string, and a time as a TIMESTAMP.

Run the script, then use the Hive interpreter to take a look at the produced tables to make sure the data is loaded correctly.

Task 1B: Ordered Traffic Count by Domain URL

One of the most essential skills in using Hive is simply to be able to produce a query that will analyse a dataset and provide some key insight to the nature of the data. You are hence required to produce a set of the most popular web domains, as determined by the URLs with the highest amount of traffic. The top 3 should be the following:

<u>URL</u>	<u>Count</u>
www.MyFace.com	23006
www.TotalaMad.com.fr	23000
www.Blooger.com	22997

1. Create a new script called `t1B.hql` and in it do the following operations.

- a. Using the `mytraffic` table, count the number of rows for each unique URL using the `group by` function.
- b. Order the data by count in descending order.
- c. Write the resultant table data to the local directory `./task1b-out/`, such that each line contains the URL and traffic count separated with a tab character (`\t`).

Task 1C: Traffic History by Date Range for a Domain

On top of analysing data, it is also important to be able to selectively query for data that conforms to particular constraints or requirements. You are required to produce a set of all traffic history that occurred on a particular day, for a particular site. An example of the output is shown below:

<u>URL</u>	<u>IPv4 address</u>	<u>Time</u>
www.Facebook.com	124.241.168.42	2014-02-14 01:41:24
www.Facebook.com	134.15.231.51	2014-02-14 10:51:06
www.Facebook.com	75.73.40.6	2014-02-14 01:47:56
...

1. Create a new script called `t1C.hql` and in it do the following operations.
 - a. Filter the data from the `mytraffic` table using a `WHERE` clause in order to find **only** the traffic for the domain URL `"www.Facebook.com"`.
 - b. Filter the data further by extending the `WHERE` clause in order to find **only** the traffic that occurred between `2014-02-14 00:00:00` and `2014-02-15 00:00:00`.

Note: You will need to use the `unix_timestamp()` function to convert date strings (like `'2014-02-14 00:00:00'`) to `TIMESTAMPS` before using them in a comparison. So, for example, the expression which matches records where the value of the `time` column comes after 2014-02-14 is:

```
time > unix_timestamp('2014-02-14 00:00:00')
```

- c. Write the resultant table data to `./task1c-out/`. Check the results to make sure that they are for the correct URL and are in the expected date range.

Task 1D: Traffic Count by Category

Moving toward the more advanced side of working with big data, it is extremely useful, often even necessary, to be able to join two or more datasets in order to produce some more meaningful result that each individual dataset alone could not provide. You are hence required to produce a list of the top 5 most popular *categories* of domains on the web. This can be determined by the sum of all traffic for every domain that belongs to each category. The list should be in descending order of count. The output should be exactly the following:

<u>Category</u>	<u>Count</u>
Education	136232
News	91272
Commerce	68072
Social	67962
Entertainment	67957

Hint: You will need to combine data from the tables `mytraffic` and `mydomains` using an inner join. Refer to Lab 03 for guidance on how to use joins.

1. Create a new script called `t1D.hql` and in it write the script required to complete this task.
2. Make sure that your script writes the results to `./task1d-out/`.
3. It is up to you to figure out how to do this. You may collaborate with a neighbour or ask your demonstrator for help, however only minimal assistance will be provided!

Task 2: Partitioning

In Hive we can horizontally partition the data by using the partition command. Horizontally partitioning the data means rows with the same partitioning key value are grouped together in the same partition. Each partition of that data will be stored in a separate directory. This allows queries on particular key values to be narrowed to a smaller data set.

For example, let's say that a multinational company has an employee table with a CREATE TABLE statement that looks like this:

```
CREATE TABLE employee(emp_id INT, name STRING, country STRING);
```

If the company frequently queries the table with a WHERE clause that restricts the results to a particular country, the table would benefit from being partitioned by country. The partitioned version of the CREATE TABLE statement would look like the following:

```
CREATE TABLE employee(emp_id INT, name STRING)
PARTITIONED BY(country STRING);
```

Note that the PARTITIONED BY clause **does not** reference the name of a column already defined in the table, like you might expect. Columns defined within PARTITIONED BY(...) are instead logically *added* to the table, as shown in the example above.

Task 2A: Creating Partitioned Tables

For this task we will create partitioned versions of the mydomains and mytraffic tables from Task 1.

In order to create a partitioned version of a table, we need to do two things. Firstly, we need to write a CREATE TABLE statement (see above) to create the partitioned table. Secondly, we need to use an INSERT OVERWRITE TABLE statement to populate the partitioned table with data from the original table.

So, let's say we have a table called albums that has three columns: an artist, a title, and a genre. We now want to create a partitioned version of that table, albumsbygenre, which is partitioned using the genre column. We can use the following code to achieve this:

```
CREATE TABLE albumsbygenre(artist STRING, title STRING)
PARTITIONED BY(genre STRING);
```

```
INSERT OVERWRITE TABLE albumsbygenre PARTITION(genre)
SELECT artist, title, genre FROM originaltable;
```

Note: because the table is partitioned on genre, that means that genre *must* appear last in the SELECT statement.

1. Open t2-partitioned.hql in a text editor. The first two lines configure Hive so that we can use dynamic partitioning. Dynamic partitioning means that Hive will partition inserted data automatically by a specified column. In contrast, static partitioning (the default) requires you to specify the partition key manually on each insert statement.
2. Create a partitioned table called domainPartitioned, which contains the same columns and data as mydomains but is partitioned by category. You can copy the CREATE TABLE statement for mydomains from t1-loadData.hql as a starting point.

3. Create a partitioned table called `trafficPartitioned`, which contains the same columns and data as `mytraffic` but is partitioned by `url`. Hint: don't forget that the partition column must appear last in the `SELECT` clause.
4. In the Hive interpreter perform a simple `SELECT * ... LIMIT 100` statement on each of the partitioned tables to verify their correctness. You should now find that, even without specifying any ordering clause in the select statement, the results are ordered by their partitioned column.

Task 2B: Properties of Partitioned Tables

1. Let's take a look at how Hive stores the partitioned table. Go to the Hue file browser to look at the Hive table stored in HDFS. Look under the following directory.

```
/user/hive/warehouse/trafficpartitioned
```

You should see that there are multiple directories—one directory per URL. If you go into a particular directory and look at the data file in it, you will see that the data only has two columns. This is because the value for the URL column does not need to be stored in the file (since it is already part of the directory name!).

2. Now the fun begins! Open up the Job Browser in Hue, and run the Hive interpreter in a new terminal window. Do not use an old Hive interpreter session from before. Let's compare the performance of the following two queries (type them in the new Hive interpreter, one after the other):

```
SELECT COUNT(1) FROM mytraffic WHERE url = 'www.Amazon.com';  
SELECT COUNT(1) FROM trafficPartitioned WHERE url = 'www.Amazon.com';
```

3. It may be difficult to tell since we are using a fairly small dataset, but the second query should be a bit faster. This speed increase is much more noticeable when working with bigger data.
4. Check the "Map Input Records" counter for both jobs, and make a note of these values so that you can show your demonstrator. You should see that by using partitions we can dramatically reduce the amount of data read by mappers.