Impala Introduction

By: Matthew Bollinger

Note:

This tutorial borrows heavily from Cloudera's provided Impala tutorial, located here. As such, it uses the Cloudera Quick Start VM, located here. The quick start VM contains a fully functioning Hadoop and Impala installation.

It is recommended that you become familiar with HiveQL, Hadoop's SQL variant, before attempting this tutorial. A basic knowledge of SQL will suffice however, a thorough understanding of HiveQL will allow for more complex queries.

Set Up Some Basic .csv Tables

This scenario illustrates how to create some very small tables, suitable for first-time users to experiment with Impala SQL features. `TAB1` and `TAB2` are loaded with data from files in HDFS. A subset of data is copied from `TAB1` into `TAB3`.

Populate HDFS with the data you want to query. To begin this process, create one or more new subdirectories underneath your user directory in HDFS. The data for each table resides in a separate subdirectory. Substitute your own user name for `cloudera` where appropriate. This example uses the `-p` option with the `mkdir` operation to create any necessary parent directories if they do not already exist.

```
$ whoami
cloudera
$ hdfs dfs -ls /user
Found 3 items
drwxr-xr-x   - cloudera cloudera            0 2013-04-22 18:54
/user/cloudera
drwxrwx---   - mapred   mapred              0 2013-03-15 20:11
/user/history
drwxr-xr-x   - hue      supergroup          0 2013-03-15 20:10
/user/hive

$ hdfs dfs -mkdir -p /user/cloudera/sample_data/tab1
/user/cloudera/sample_data/tab2
```

Here is some sample data, for two tables named `TAB1` and `TAB2`.

Copy the following content to `.csv` files in your local filesystem:

`tab1.csv`:

```
1,true,123.123,2012-10-24 08:55:00
2,false,1243.5,2012-10-25 13:40:00
3,false,24453.325,2008-08-22 09:33:21.123
4,false,243423.325,2007-05-12 22:32:21.33454
```

```
5,true,243.325,1953-04-22 09:11:33
```

`tab2.csv`:

```
1,true,12789.123
2,false,1243.5
3,false,24453.325
4,false,2423.3254
5,true,243.325
60,false,243565423.325
70,true,243.325
80,false,243423.325
90,true,243.325
```

Put each `.csv` file into a separate HDFS directory using commands like the following, which use paths available in the Impala Demo VM:

```
$ hdfs dfs -put tab1.csv /user/cloudera/sample_data/tab1
$ hdfs dfs -ls /user/cloudera/sample_data/tab1
Found 1 items
-rw-r--r--   1 cloudera cloudera        192 2013-04-02 20:08
/user/cloudera/sample_data/tab1/tab1.csv


$ hdfs dfs -put tab2.csv /user/cloudera/sample_data/tab2
$ hdfs dfs -ls /user/cloudera/sample_data/tab2
Found 1 items
-rw-r--r--   1 cloudera cloudera        158 2013-04-02 20:09
/user/cloudera/sample_data/tab2/tab2.csv
```

The name of each data file is not significant. In fact, when Impala examines the contents of the data directory for the first time, it considers all files in the directory to make up the data of the table, regardless of how many files there are or what the files are named.

To understand what paths are available within your own HDFS filesystem and what the permissions are for the various directories and files, issue `hdfs dfs -ls /` and work your way down the tree doing `-ls` operations for the various directories.

Use the `impala-shell` command to create tables, either interactively or through a SQL script.

The following example shows creating three tables. For each table, the example shows creating columns with various attributes such as Boolean or integer types. The example also includes commands that provide information about how the data is formatted, such as rows terminating with commas, which makes sense in the case of importing data from a `.csv` file. Where we already have `.csv` files containing data in the HDFS directory tree, we specify the location of the directory containing the appropriate `.csv` file. Impala considers all the data from all the files in that directory to represent the data for the table.

```
DROP TABLE IF EXISTS tab1;
-- The EXTERNAL clause means the data is located outside the central
location for Impala data files
```

```
-- and is preserved when the associated Impala table is dropped. We
expect the data to already
-- exist in the directory specified by the LOCATION clause.
CREATE EXTERNAL TABLE tab1
(
   id INT,
   col_1 BOOLEAN,
   col_2 DOUBLE,
   col_3 TIMESTAMP
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/user/cloudera/sample_data/tab1';

DROP TABLE IF EXISTS tab2;
-- TAB2 is an external table, similar to TAB1.
CREATE EXTERNAL TABLE tab2
(
   id INT,
   col_1 BOOLEAN,
   col_2 DOUBLE
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/user/cloudera/sample_data/tab2';

DROP TABLE IF EXISTS tab3;
-- Leaving out the EXTERNAL clause means the data will be managed
-- in the central Impala data directory tree. Rather than reading
-- existing data files when the table is created, we load the
-- data after creating the table.
CREATE TABLE tab3
(
   id INT,
   col_1 BOOLEAN,
   col_2 DOUBLE,
   month INT,
   day INT
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

You are now able to execute queries on the tables `tab1`, `tab2`, and `tab3`. Write your queries in HiveQL. Below are some example queries however, you are encouraged to play around on your own!

```
[localhost.localdomain:21000] > SELECT * FROM tab1;
Query: SELECT * FROM tab1
Query finished, fetching results ...
+----+-------+------------+-------------------------------+
| id | col_1 | col_2      | col_3                         |
+----+-------+------------+-------------------------------+
```

```
| 1  | true  | 123.123    | 2012-10-24 08:55:00           |
| 2  | false | 1243.5     | 2012-10-25 13:40:00           |
| 3  | false | 24453.325  | 2008-08-22 09:33:21.123000000 |
| 4  | false | 243423.325 | 2007-05-12 22:32:21.334540000 |
| 5  | true  | 243.325    | 1953-04-22 09:11:33           |
+----+-------+------------+-------------------------------+
Returned 5 row(s) in 1.81s

[localhost.localdomain:21000] > SELECT * FROM tab2;
Query: SELECT * FROM tab2
Query finished, fetching results ...
+----+-------+---------------+
| id | col_1 | col_2         |
+----+-------+---------------+
| 1  | true  | 12789.123     |
| 2  | false | 1243.5        |
| 3  | false | 24453.325     |
| 4  | false | 2423.3254     |
| 5  | true  | 243.325       |
| 60 | false | 243565423.325 |
| 70 | true  | 243.325       |
| 80 | false | 243423.325    |
| 90 | true  | 243.325       |
+----+-------+---------------+
Returned 9 row(s) in 0.56s

[localhost.localdomain:21000] > SELECT * FROM tab3;
Query: SELECT * FROM tab3
Query finished, fetching results ...

Returned 0 row(s) in 0.57s

[localhost.localdomain:21000] > INSERT INTO tab3 VALUES(1, true,
1234.321, 3, 20);
Query: INSERT INTO tab3 VALUES(1, true, 1234.321, 3, 20)
Inserted 1 rows in 0.95s

[localhost.localdomain:21000] > SELECT * FROM tab3;
Query: SELECT * FROM tab3
Query finished, fetching results ...
+----+-------+----------+-------+-----+
| id | col_1 | col_2    | month | day |
+----+-------+----------+-------+-----+
| 1  | true  | 1234.321 | 3     | 20  |
+----+-------+----------+-------+-----+
Returned 1 row(s) in 0.51s
```

Describe the Impala Table

To see what databases Impala has access to, issue the `show databases` command. To see which tables are in the currently connected database, issue the `show tables` command. To see the schema of a table, issue the `describe` command. These commands are demonstrated below using the current example.

```
[localhost.localdomain:21000] >  show databases;
Query: show databases
Query finished, fetching results ...
+---------+
| name    |
+---------+
| default |
+---------+
Returned 1 row(s) in 0.12s

[localhost.localdomain:21000] >  show tables;
Query: show tables
Query finished, fetching results ...
+------+
| name |
+------+
| tab1 |
| tab2 |
| tab3 |
+------+
Returned 3 row(s) in 0.11s

[localhost.localdomain:21000] >  describe tab1;
Query: describe tab1
Query finished, fetching results ...
+-------+-----------+---------+
| name  | type      | comment |
+-------+-----------+---------+
| id    | int       |         |
| col_1 | boolean   |         |
| col_2 | double    |         |
| col_3 | timestamp |         |
+-------+-----------+---------+
Returned 4 row(s) in 0.12s
```

Query the Impala Table

You can query data contained in the tables. Impala coordinates the query execution across a single node or multiple nodes depending on your configuration, without the overhead of running MapReduce jobs to perform the intermediate processing.

There are a variety of ways to execute queries on Impala:

- Using the `impala-shell` directly:

```
$ impala-shell
Connected to localhost.localdomain:21000
[localhost.localdomain:21000] > select count(*) from tab1;
Query: select count(*) from tab1
Query finished, fetching results ...
+----------+
| count(*) |
+----------+
| 5        |
+----------+
Returned 1 row(s) in 0.55s
```

- Passing a set of commands contained in a file:

```
$ impala-shell -f myquery.sql
Connected to localhost.localdomain:21000
Query: select count(*) from tab1
Query finished, fetching results ...
+----------+
| count(*) |
+----------+
| 5        |
+----------+
Returned 1 row(s) in 0.27s
```

- Passing a single command to the `impala-shell` command. The query is executed, the results are returned, and the shell exits. Make sure to quote the command, preferably with single quotation marks to avoid shell expansion of characters such as `*`.

```
$ impala-shell -q 'select count(*) from tab1'
Connected to localhost.localdomain:21000
Query: select count(*)
from tab1
Query finished, fetching results ...
+----------+
| count(*) |
+----------+
| 5        |
+----------+
Returned 1 row(s) in 0.26s
```

Data Loading and Querying Examples

This section describes how to create some sample tables and load data into them. These tables can then be queried using the Impala shell.

Loading Data

Loading data involves:

- Establishing a data set. The example below uses `.csv` files. (Already covered)
- Creating tables to which to load data. (Already covered)
- Loading the data into the tables you created. (Below)

Sample Queries

To run these sample queries, create a SQL query file `query.sql`, copy and paste each query into the query file, and then run the query file using the shell. For example, to run `query.sql` on `impala-host`, you might use the command:

```
impala-shell -i impala-host -f query.sql
```

The examples and results below assume you have loaded the sample data into the tables as described above.

Example: Examining Contents of Tables

Let's start by verifying that the tables do contain the data we expect. Because Impala often deals with tables containing millions or billions of rows, when examining tables of unknown size, include the `LIMIT` clause to avoid huge amounts of unnecessary output, as in the final query. (If your interactive query starts displaying an unexpected volume of data, press `Ctrl-C` in `impala-shell` to cancel the query.)

```
SELECT * FROM tab1;
SELECT * FROM tab2;
SELECT * FROM tab2 LIMIT 5;
```

Results:

```
+----+-------+------------+-------------------------------+
| id | col_1 | col_2      | col_3                         |
+----+-------+------------+-------------------------------+
| 1  | true  | 123.123    | 2012-10-24 08:55:00           |
| 2  | false | 1243.5     | 2012-10-25 13:40:00           |
| 3  | false | 24453.325  | 2008-08-22 09:33:21.123000000 |
| 4  | false | 243423.325 | 2007-05-12 22:32:21.334540000 |
| 5  | true  | 243.325    | 1953-04-22 09:11:33           |
+----+-------+------------+-------------------------------+

+----+-------+---------------+
| id | col_1 | col_2         |
+----+-------+---------------+
| 1  | true  | 12789.123     |
```

```
| 2  | false | 1243.5        |
| 3  | false | 24453.325     |
| 4  | false | 2423.3254     |
| 5  | true  | 243.325       |
| 60 | false | 243565423.325 |
| 70 | true  | 243.325       |
| 80 | false | 243423.325    |
| 90 | true  | 243.325       |
+----+-------+---------------+

+----+-------+-----------+
| id | col_1 | col_2     |
+----+-------+-----------+
| 1  | true  | 12789.123 |
| 2  | false | 1243.5    |
| 3  | false | 24453.325 |
| 4  | false | 2423.3254 |
| 5  | true  | 243.325   |
+----+-------+-----------+
```

Example: Aggregate and Join

```
SELECT tab1.col_1, MAX(tab2.col_2), MIN(tab2.col_2)
FROM tab2 JOIN tab1 USING (id)
GROUP BY col_1 ORDER BY 1 LIMIT 5;
```

Results:

```
+-------+-----------------+-----------------+
| col_1 | max(tab2.col_2) | min(tab2.col_2) |
+-------+-----------------+-----------------+
| false | 24453.325       | 1243.5          |
| true  | 12789.123       | 243.325         |
+-------+-----------------+-----------------+
```

Example: Subquery, Aggregate and Joins

```
SELECT tab2.*
FROM tab2,
(SELECT tab1.col_1, MAX(tab2.col_2) AS max_col2
 FROM tab2, tab1
 WHERE tab1.id = tab2.id
 GROUP BY col_1) subquery1
WHERE subquery1.max_col2 = tab2.col_2;
```

Results:

```
+----+-------+-----------+
| id | col_1 | col_2     |
+----+-------+-----------+
```

```
| 1   | true   | 12789.123 |
| 3   | false  | 24453.325 |
+----+-------+-----------+
```

Example: INSERT Query

```
INSERT OVERWRITE TABLE tab3
SELECT id, col_1, col_2, MONTH(col_3), DAYOFMONTH(col_3)
FROM tab1 WHERE YEAR(col_3) = 2012;
```

Query `TAB3` to check the result:

```
SELECT * FROM tab3;
```

Results:

```
+----+-------+---------+-------+-----+
| id | col_1 | col_2   | month | day |
+----+-------+---------+-------+-----+
| 1  | true  | 123.123 | 10    | 24  |
| 2  | false | 1243.5  | 10    | 25  |
+----+-------+---------+-------+-----+
```

DLRL Example 1: Pothole data

This section shows the process of using Impala with the pothole dataset from the IDEAL project. This dataset contains tweets pertaining to potholes. Contact Sunshin Lee, sslee777@vt.edu, to get a copy of this dataset.

Preparing the data

The first step is to prep the dataset to be imported into Impala. The raw tweet has the following format.

```
archivesource, text, to_user_id, from_user, id, from_user_id,
iso_language_code, source, profile_image_url, geo_type,
geo_location_0, geo_location_1, created_at, time
```

Below is an example of a raw tweet.

```
twitter-search, Opened Pothole report via Web at 330-398 Mount Vernon
Avenue Northwest Grand Rapids Lots of potholes - right nea, ,
GrandRapids311, 452137831478337536, 199370650, en, <a
href="http://grcity.us/index.pl?page_id=11831" rel="nofollow">Grand
Rapids 311</a>,
http://pbs.twimg.com/profile_background_images/184356099/Calder_Detai
```

```
l_-_Copy.jpg, Point, 42.96971381, -85.67896364, Fri Apr 04 17:37:13
+0000 2014, 1396633033
```

Notice how the source field has extraneous characters, such as the '`<a href`' tag. This will slow down any queries related to this field so we will need to strip out the base URL. Also, note that the '`created_at`' field is also poorly formatted for large queries. We will need to parse the day, month, and year from this field and make new columns for these fields to improve our search abilities. To do this, run the following commands.

Note: You will need Python 2.6.6 and the `dateutil` module installed to use `Twitter_Data_Editor.py`.

```
$ sed -i 's/\x0//g' pothole.csv
$ python Twitter_Data_Editor.py pothole.csv
```

The first command is used to remove any `NULL` characters from the file, as they will prevent the python script from executing correctly. The second command runs a python script on the dataset. The script will prepare the CSV file as discussed earlier. The source code and a technical explanation of this script can be found later in this tutorial.

Import data to HDFS

The next step is to import the CSV file into HDFS. This is done exactly as before, but the steps are repeated below for convenience.

```
$ hdfs dfs -mkdir /user/cloudera/pothole
$ hdfs dfs -put pothole.csv /user/cloudera/pothole
$ hdfs dfs -ls /user/cloudera/pothole
Found 1 items
-rw-r--r--   3 cloudera cloudera   90397154 2014-05-07 20:43
/user/cloudera/pothole/pothole.csv
```

Setup Impala Table

Now that the data is successfully in HDFS, we can setup our table. The SQL script used for this is below, along with the `impala-shell` command to execute it.

```
DROP TABLE IF EXISTS pothole;

CREATE EXTERNAL TABLE pothole
(
       archivesource STRING,
       text STRING,
       to_user_id STRING,
```

```
        from_user STRING,
        id STRING,
        from_user_id STRING,
        iso_language_code STRING,
        source STRING,
        profile_image_url STRING,
        geo_type STRING,
        geo_coordinates_0 DOUBLE,
        geo_coordinates_1 DOUBLE,
        created_at STRING,
        time INT,
        month INT,
        day INT,
        year INT
)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION
'/user/cloudera/pothole/';
```

```
$ impala-shell –f pothole_setup.sql
```

Setup Impala Table

We can execute queries on our data. Below are some example queries that can be run on the pothole dataset. You are encouraged to try out your own queries for practice.

```
SELECT count(*) as num_tweets, from_user FROM pothole GROUP BY
from_user ORDER BY num_tweets DESC LIMIT 15;
```

Results:

```
+------------+------------------+
| num_tweets | FROM_user        |
+------------+------------------+
| 2912       |  GrandRapids311  |
| 2714       |  mrpotholeuk     |
| 1720       |  citizensconnect |
| 1435       |  NJI95thm        |
| 1202       |  baltimore311    |
| 1189       |  NYI95thm        |
| 1135       |  NYI78thm        |
| 843        |  NJI78thm        |
| 656        |  MarquelatTPV    |
| 576        |  FixedInDC       |
| 498        |  NYI87thm        |
| 497        |  csreports       |
| 374        |  BridgeviewDemo  |
```

```
| 355        |  MPLS311        |
| 340        |  edm_pothole    |
+------------+-----------------+
```

```
SELECT count(*) as num_tweets, source FROM pothole GROUP BY source
ORDER BY num_tweets DESC LIMIT 15;
```

Results:

```
+------------+---------------------------+
| num_tweets | source                    |
+------------+---------------------------+
| 131495     |  pothole.com              |
| 33074      |  web                      |
| 8042       |  potholefeed.com          |
| 6869       |  blackberry.com           |
| 6501       |  www.TheHighwayMonitor.com |
| 6326       |  www.hootsuite.com        |
| 5035       |  about.pothole.com        |
| 4587       |  www.facebook.com         |
| 3466       |  mobile.pothole.com       |
| 3217       |  dlvr.it                  |
| 2912       |  grcity.us                |
| 2684       |  www.tweetdeck.com        |
| 2555       |  roundteam.co             |
| 2537       |  tapbots.com              |
| 2455       |  www.echofon.com          |
+------------+---------------------------+
```

```
SELECT count(*) as num_tweets, month FROM pothole GROUP BY month
ORDER BY num_tweets DESC LIMIT 15;
```

Results:

```
+------------+-------+
| num_tweets | month |
+------------+-------+
| 61243      | 2     |
| 60555      | 3     |
| 25212      | 1     |
| 23009      | 4     |
| 12706      | 5     |
| 11897      | 12    |
| 10947      | 8     |
| 9906       | 10    |
| 9779       | 9     |
```

```
| 9538       | 6      |
| 8602       | 11     |
| 7809       | 7      |
+------------+-------+
```

DLRL Example 2: Shooting data

This section shows the process of using Impala with the shooting dataset from the IDEAL project. This dataset contains tweets pertaining to shootings.

This section is very similar to the previous except that it will be run on the DLRL cluster and that the dataset is very large. Altogether, there are more than 22 million tweets in the set.

Contact Sunshin Lee, sslee777@vt.edu, for a copy of the dataset and for access to the DLRL cluster. All work in this section uses the `dlrlhive` user account on the DLRL cluster.

Preparing the data

Again, the first step is to prep the data to be imported into Impala. This is more difficult with this dataset as the size and variance of the tweets causes problems. Many of the tweets in this set come from other countries and sources which causes issues with encoding and format. Because we are unable to identify every variant format, the Python script will drop any tweet it unable to parse. This results in roughly 1 million tweets being lost but this is a small portion of the overall set. Executing the script is done the same way as before.

```
$ sed -i 's/\x0//g' pothole.csv
$ python Twitter_Data_Editor.py pothole.csv
```

Please be aware that executing the script on this dataset will take roughly 2.5 hours to complete since it is so large.

Import data to HDFS

The next step is to import the CSV file into HDFS. This is done exactly as before, but the steps are repeated below for convenience. Again, this will take a long time due to the size of the file.

```
$ hdfs dfs -mkdir /user/dlrlhive/shooting
$ hdfs dfs -put pothole.csv /user/dlrlhive/shooting
$ hdfs dfs -ls /user/dlrlhive/shooting
Found 1 items
-rw-r--r--   3 dlrlhive supergroup 6814267433 2014-05-08 01:00
/user/dlrlhive/shooting/shooting.csv
```

Note: On the DLRL cluster, only node2 has permission to execute `impala-shell` commands so `ssh` to that node before continuing.

Setup Impala Table

Now that the data is successfully in HDFS, we can setup our table. The SQL script used for this is below, along with the `impala-shell` command to execute it.

```
DROP TABLE IF EXISTS shooting;

CREATE EXTERNAL TABLE shooting
(
        archivesource STRING,
        text STRING,
        to_user_id STRING,
        from_user STRING,
        id STRING,
        from_user_id STRING,
        iso_language_code STRING,
        source STRING,
        profile_image_url STRING,
        geo_type STRING,
        geo_coordinates_0 DOUBLE,
        geo_coordinates_1 DOUBLE,
        created_at STRING,
        time INT,
        month INT,
        day INT,
        year INT
)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION
'/user/dlrlhive/shooting/';
```

```
$ impala-shell –f shooting_setup.sql
```

Setup Impala Table

We can execute queries on our data. Below are some example queries that can be run on the pothole dataset. You are encouraged to try out your own queries for practice.

```
SELECT count(*) as num_tweets, from_user FROM shooting GROUP BY
from_user ORDER BY num_tweets DESC LIMIT 15;
```

Results:

```
+------------+------------------+
| num_tweets | from_user        |
+------------+------------------+
| 22592      |  Shooting_ray    |
```

```
| 20001      |  PulpNews        |
| 11943      |  Shooting_Beauty |
| 11781      |  Shooting_Sarr   |
| 11016      |  shooting_star03 |
| 9426       |  shooting_kyuri  |
| 8147       |  TLW3            |
| 7291       |  shooting_star50 |
| 4880       |  g8keds          |
| 4627       |  shooting_star56 |
| 4523       |  Police_Dispatch |
| 4092       |  USRadioNews     |
| 3863       |  shooting_rocka  |
| 3766       |  OldHydePark     |
| 3714       |  BrianBrownNet   |
+------------+-----------------+
```

```
SELECT count(*) as num_tweets, source FROM shooting GROUP BY source
ORDER BY num_tweets DESC LIMIT 15;
```

Results:

```
+------------+---------------------+
| num_tweets | source              |
+------------+---------------------+
| 10805547   |  twitter.com        |
| 3439406    |  web                |
| 886741     |  twitterfeed.com    |
| 885813     |  blackberry.com     |
| 449664     |  instagram.com      |
| 418707     |  www.facebook.com   |
| 413397     |  mobile.twitter.com |
| 338992     |  dlvr.it            |
| 312117     |  www.tweetdeck.com  |
| 247272     |  www.echofon.com    |
| 222873     |  www.tweetcaster.com |
| 207485     |  www.hootsuite.com  |
| 184123     |  ubersocial.com     |
| 159245     |  tapbots.com        |
| 147060     |  about.twitter.com  |
+------------+---------------------+
```

```
SELECT count(*) as num_tweets, month FROM shooting GROUP BY month
ORDER BY num_tweets DESC LIMIT 15;
```

Results:

```
+------------+-------+
| num_tweets | month |
+------------+-------+
| 2744268    | 4     |
| 2735448    | 3     |
| 2429363    | 2     |
| 1976646    | 9     |
| 1828213    | 8     |
| 1717971    | 12    |
| 1620966    | 11    |
| 1394857    | 10    |
| 1359162    | 5     |
| 1287062    | 6     |
| 1108271    | 1     |
| 1065465    | 7     |
+------------+-------+
```

Twitter_Data_Editor.py Explanation

This section gives a technical explanation of the Twiiter_Data_Editor python script. The source code has been copied below for convenience.

```
#!/usr/bin/env python
#
# Twitter_Data_Editor.py
# python version 2.6.6
# invocation: python Twitter_Data_Editor.py [filename]

import re
import sys
import csv
import shutil
import datetime
from dateutil import parser
from tempfile import NamedTemporaryFile

filename = str(sys.argv[1])

with open('temp.csv', 'wb') as tempfile:
	writer = csv.writer(tempfile, delimiter=',')
	with open(filename, 'rb') as orig:
		reader = csv.reader(orig, delimiter=',', quotechar='"')

		for row in reader:
			# If there are extra commas in the row (ie in
the source/text)
			# there is no way to identify them and we should
drop the row
```

```
                            # to prevent it from messing with the database
import.
                            if (len(row) != 14):
                                        continue;

                            # Strip source URL of extra junk
                            text = row[7]
                            text = re.sub(r'<a href="https?://', '', text)
                            text = re.sub(r'/.*</a>', '', text)
                            text = re.sub(r'" rel=.*</a>', '', text)
                            text = re.sub(r'"? rel=.*>.*"?', '', text)
                            text = re.sub(r'&lt;a href=&quot;https?://', '',
text)
                            text = re.sub(r'/.*&lt;/a&gt;', '', text)
                            text = re.sub(r'&quot;&gt;.*&lt;/a&gt;', '',
text)
                            row[7] = text

                            # Parse created_at and append month, day, year
columns
                            text = row[12]
                            try:
                                        dt = parser.parse(text, fuzzy=True)
                            except ValueError: # Catches Feb 29 error
                                        dt = datetime.datetime(dt.year,
dt.month, 1, dt.hour, dt.minute)
                            except TypeError:
                                        continue
                            row.append(str(dt.month))
                            row.append(str(dt.day))
                            row.append(str(dt.year))

                            # Remove spaces from geo coordinate and time
columns to make it easier to import to other DBs
                            row[10] = row[10].strip();
                            row[11] = row[11].strip();
                            row[13] = row[13].strip();

                            writer.writerow(row)

shutil.move(tempfile.name, filename)
```

The script uses the Python CSV module to parse the traverse the file, row by row, and pick out the rows that require editing. It writes the new rows to a temporary file during execution and at the end replaces the source file with the temporary file.

The first thing the script will do to each row of the CSV is check how many columns it has. The standard tweet should have 14 rows. Many of the tweets in the shooting dataset had more than 14 rows and, originally, this threw the parser off. These extra rows were caused by commas in the text or source fields. To avoid this issue, we ignore any row with more than 14 columns. This results in some data loss but the amount lost is small; roughly 6% for the shooting dataset.

If a tweet has the appropriate number of columns, the script then strips extraneous information from the `source` column. This is done via regular expressions. There are different character encodings depending on where the tweet originated from so both cases are handled. The `source` column is then overwritten with the changes.

Next, the script will parse the day, month, and year from the `created_at` column. This is done via the Python dateutil and datetime modules. After parsing the values, `day`, `month`, and `year` columns are appended to the row.

Finally, the script will strip the whitespace from the geo coordinate and time fields so that they can be properly imported into impala as doubles and integers respectfully.