Daniel Foley Follow
Oct 2, 2018 · 8 min read ★ · Listen

Streaming Twitter Data into a MySQL Database

Given the frequency that I have seen database languages listed as a requirement for data science jobs, I thought it would be a good idea to do a post on MySQL today. In particular, I wanted to look at how we can use python and an API to stream data directly into a MySQL database. I did this recently for a personal project and thought I would share my code and provide an introductory tutorial for those who may not be familiar with these tools.

We are going to be using the Twitter API to search for tweets containing specific keywords and stream this directly into our database. Once we have done this, the data will be available for further analysis at any time. This task requires a few things:

1. A Twitter account and API credentials
2. A MySQL database
3. The Tweepy and mysql-connector Python Libraries

Twitter API

Before we access the API you need to set up a twitter app. I won't do an in-depth tutorial on this but briefly, you need to do the following:

- go to the following website https://developer.twitter.com/ and create an account. (This step is a bit more involved then it used to be and involves

- Fill in all the details about your app and then create your access token.
- Make a note of your consumer key, consumer secret, OAuth access token and OAuth access token secret. These are needed to connect to the API.

For a more complete tutorial on this, I suggest this blog post. After these steps, our app is now able to connect to the Twitter streaming API provided we write the correct code. Next up, I will go through setting up the MySQL database so we have somewhere to store all of our data.

MySQL Workbench

There are many different types of databases we could use for this particular task including NoSQL databases such as MongoDB or Redis. I have, however, chosen to use MySQL as I am more familiar with it and it is still one of the most popular databases out there. Before we begin, we will need to install MySQL Workbench and MySQL server. Here is a video tutorial explaining how to install both and set everything up to start collecting the data.

Once you have finished the tutorial above, you should have a connection and a schema/database set up (My database is imaginatively called **twitterdb**). After we have set up MySQL workbench and are somewhat familiar with the interface, we can finally create a table to store our twitter data. Creating a table is very straightforward and we can use the UI or even use queries. Using the UI we just right click on our database and click create a table. We can then input our column names and data types directly. At this point, it is worth thinking about the data we want to store and what kind of data types they will be. To get a better understanding of the data types we need we should take a peek at the TwitterAPI documentation. Essentially I want the **username** of the person who wrote the **tweet**, the time it was **created**, the **tweet**, the **retweet count**, the **place** the tweet originated and the **location** (more on these below). This corresponds to 6 columns plus the **primary key** and we can define the datatypes as follows:

- primary key: INT(11)
- username: VARCHAR(255)
- created_at: VARCHAR(45)
- tweet: TEXT
- retweet_count: INT(11)
- location: VARCHAR(100)
- place: VARCHAR(100)

Python

OK now that we have our database set up, its' time to jump into the Python code. There are a few things we want our code to do:

1. We want to create a class that allows us to connect to the Twitter API.
2. We also need to create some code that connects to our database and reads the data into the correct columns.

We are going to be using the Tweepy library which will make it very easy for us to connect to the API and start streaming the data. Before we start, we are again going to look at some delicious documentation. In the Tweepy documentation, we can find some really useful examples of the classes and methods we need to use to interact with the API. The code below is a simple example that allows us to connect to the API and print tweets from our timeline:

```
import tweepy

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

api = tweepy.API(auth)

public_tweets = api.home_timeline()
for tweet in public_tweets:
    print(tweet.text)
```

Alright, so hopefully this is pretty straightforward. It looks like we just need to set our credentials, access the timeline and loop through it to print it out. What we want to do is a bit different. We want to stream live tweets into our database and according to the documentation, need to do the following three things:

1. Create a class inheriting from **StreamListener.**
2. Instantiate an object from this class.

and keys as environment variables.

```
subprocess.call(“./settings.sh”, shell=True)
```

Note that in our imports, we need **mysql-connector**. Again, here is some useful examples of how the library works. We can install any libraries that we don’t have using the pip command in the terminal (I am on Mac) like below. We should then be able to import these libraries from our script.

```
pip install mysql-connector
pip install tweepy
```

Next up, we need to set up our class inheriting from StreamListener. We are going to give the class three methods. These are methods that the class already implements which we are going to override. The code below implements this.

Let’s go through this step by step to make sure everything is clear. The first method, **on_connect()** simply notifies us when we are connected to the stream. The **on_error()** method prints an error whenever our HTTP status code is not 200 (200 means everything worked). List of codes for those interested: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

OK, the next method, **on_data()** is a little more complex. To understand this we will need to know a little bit more about the structure of the tweets. When we access the API we are getting a JSON response (very similar structure to a python dictionary). More info here.

Essentially our tweet object that is returned looks something like this:

```
{ “created_at”:”Thu Apr 06 15:24:15 +0000 2017",
  “id”: 850006245121695744,
  “id_str”: “850006245121695744”,
  “text”: “1/ Today we’re sharing our vision for the future of the    Twitter API platform!nhttps://t.co/XweGngmxlP",
  “user”: {},
  “entities”: {} }
```

So we have a JSON object which contains key, value pairs (note there are a few attributes not listed here that we will use). So what data do we actually want from this? There is actually quite a lot of information available from the tweet object and I recommend going through the documentation to see what attributes might interest you. For this analysis I chose to collect data on the username, the time the tweet was created (this is more useful if we collect tweets over time), the actual tweet, the country of the tweet, the location (which is more local) and finally the retweet count and this is reflected in the code above.

The final few lines call the **connect()** method which takes our variables as parameters. This code is all wrapped in a try, except statement to catch any errors, we may run into.

Alright, you may have noticed that we haven’t created the connect() method yet so let’s create it. This method is not surprisingly, going to connect to our database and feed in all the data. As I said before, the method takes in our variables created in the on_data() method of the StreamListener class as arguments and inserts them into the columns of the same name in our database. To connect to our database we simply use the **connector.connect** method and pass in our database info which we can find from MySQL workbench. If our connection was successful, a cursor object is created allowing us to execute SQL statements. Now we can write our query and insert the data into the correct table in our **twitterdb** database using the execute command. While is_connected() is true our database connection stays open and continually feeds the data into the database until we kill it in the terminal (using Ctrl+C).

We can create a list of words that we want to filter the stream for. I am a bit of a golf fan so I decided to search for words relating to golf. In practice, you can put whatever you want in this list.

Now we just need to set up our script to call these functions when we execute the file from the terminal. To access the API we need to pass our credentials as arguments to the **OAuthHandler** method and the **set_access_token** method. Next, we create the stream by passing in our verified api object and our listener. We can also create our list of words to filter for here. To start the stream we simply call the **filter** method on our stream object and pass in our list of words as an argument. I saved this script as **StreamSQL.py.**

If we want to run this code and start collecting tweets we can use the terminal. One important thing to note here is that we need to make sure our SQL server is up and running for the script to work so it is worth double checking this before we run the script.

We can open a terminal directly from the folder where we stored the script and simply type:

Conclusion

Hopefully, this has shown that it is not that complicated to set up a data pipeline with MySQL especially when taking advantage of the powerful libraries Python has to offer. SQL is a really important aspect of doing data science and this is pretty obvious if you have ever looked at the requirements of a data scientist in job advertisements. Although this didn’t really involve any deep understanding of SQL or writing complex queries I think it is still a really useful to be able to understand do this kind of task.

For those interested in learning more about SQL, I took the following free course which I found excellent for practicing some of the key concepts and understanding some of the more advanced functionality. I recommend this to anyone interested in improving their SQL skills. In a future post, I will focus on extracting the raw data we collected, clean it and perform some simple analysis which will illustrate some of the interesting things we can do with this type of data.

Until next time!!

Link to Part 2: https://towardsdatascience.com/building-an-etl-pipeline-in-python-f96845089635