CSC 346 Fall 2014

November 4, 2014

CSC 346 : Assignment 5

Twitter

Milestone Due: Wednesday November 12 at 11:59pm
Final Version Due: Tuesday November 18 at 11:59pm

Overview

For this assignment you will write the core functionality of a Twitter-like social media service -- the functionality that stores and retrieves messages in a database. Users use command-line tools to post and view messages (hence this is the core of the service, we are not going to implement a GUI). The messages will be stored in a DynamoDB database, so that posting a message stores it in the database and viewing messages retrieves them from the database. You may implement your solution in either Java or Python.

Post

The `post` command has the following syntax:

```
post user body
```

where *user* is a user name, and *body* is string of text. Both the user name and the body may contain any valid ASCII character; the user name is limited to 10 characters and the body 140 characters. The body may contain hashtags -- single words that start with the '#' character. Hashtags can be used to filter messages as described in the `view` command below.

Messages cannot be deleted, nor can they be modified once they have been posted.

View

The `view` command has the following syntax:

```
view [key] [limit]
```

where *key*, if specified, is either a user name (prefaced with '@') or a hashtag (prefaced with '#'). If a key is specified, then the `view` command only displays messages posted by that user, or containing that hashtag, as appropriate. Note that either a user name or a hashtag can be specified, not both, and not more than one. If a key is not specified then all messages are displayed.

The *limit*, if specified, specifies the maximum number of messages displayed. Messages are displayed in reverse chronological order (newest message first), so that specifying a limit of N will only display the N most recent messages. If a limit is not specified then all messages are displayed.

Messages are displayed in the following format:

```
user date body
```

where `user` is the user name that posted the message, `date` is the time at which the message was posted (in the format returned by Python's `time.ctime()` function), and `body` is the body of the message that was posted. These fields are separated by a tab character ('\t'). Here are some examples:

```
> view

bob   Tue Nov  4 09:30:24 2014   this is my last test
alice Tue Nov  4 09:26:59 2014   this is also a #test
bob   Tue Nov  4 09:23:57 2014   this is a #test

> view @alice

alice Tue Nov  4 09:26:59 2014   this is also a #test

> view @bob

bob   Tue Nov  4 09:30:24 2014   this is my last test
bob   Tue Nov  4 09:23:57 2014   this is a #test

> view @bob 1

bob   Tue Nov  4 09:30:24 2014   this is my last test

> view #test

alice Tue Nov  4 09:26:59 2014   this is also a #test
bob   Tue Nov  4 09:23:57 2014   this is a #test
```

DynamoDB

Your solution must make use of DynamoDB to store the messages. Your solution must have the following properties:

- The `view` command must use the `query_2()` function (or the Java equivalent) to retrieve from DynamoDB only messages posted by the specified user or containing the specified hashtag. In particular, the `view` command cannot scan the entire table of messages and perform its own filtering.
- Your solution must be robust in the face of concurrent posts, i.e. it must remain consistent even if the same user simultaneously posts multiple messages, or if multiple messages containing the same hashtag are simultaneously posted.

- Note that each message must have a unique primary key, and that the combination of the user name and body is not unique -- the same user may post the same message twice. You might consider using a combination of the user name and the current time as the primary key, but note that the current time must be finer resolution than seconds (as you do not want to limit a user to one message a second), and that even with the finest resolution possible there might still be multiple messages posted at exactly the same time. You'll need a retry loop to handle collisions on the primary key.

Configuration

The `post` and `view` commands are configured via a `settings.cfg` file with the following contents:

```
[Twitter]
aws_access_key_id: <your AWS access key ID>
aws_secret_access_key: <your AWS secret access key>
region: <AWS region to use>
local: <True or False>
```

The first three should be self-explanatory; the `local` option indicates whether or not DynamoDB Local should be used instead of the DynamoDB Service. If set to True, the command should connect to DynamoDB Local using the default parameters (i.e. as shown in lecture).

Milestone

The milestone consists of a simplified version of the `post` command. This version stores the message in DynamoDB and then uses the `scan()` function to display all messages (in the order in which they are returned by `scan()`).

You must turn in the milestone even if you complete the entire assignment by the milestone deadline.