Lab 21: MySQL

Apr 19, 2018

Main Event

1. Introduction

Ensure that the MySQL Connector/Python module is installed on your system:

```
$> conda install mysql-connector-python
```

Your code should make a connection to the “twitter” database located at 10.224.45.113, with username cs101. Within twitter, you should operate on the table “tweet”.

2. Database schema

Before getting into the actual application, it is a good idea to first get comfortable with the database that the application will use.

The twitter database consists of a single relation called “tweet”. Each tuple in the relation consists of values corresponding to a single tweet. Specifically, a tweets message, author, likes, unique identifier, and creation date. Formally, the database schema is as follows:

```
CREATE TABLE tweet (
   id INTEGER NOT NULL AUTO_INCREMENT,
   user VARCHAR(32) NOT NULL,
   message VARCHAR(140) NOT NULL,
   likes INTEGER DEFAULT 0,
   tstamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   PRIMARY KEY (id)
);
```

For purposes of this lab, that schema can be interpreted as:

Attribute	Type	Description
id	integer	A tweets unique ID.
user	string	Username of the tweeter.
message	string	The tweet!
likes	integer	Number of likes for this tweet.
tstamp	TIMESTAMP	Time at which the tweet was tweeted.

3. Twitter

For now, we will interact with the twitter database via the command line. To this end, you should develop a program that firsts asks the user for their user name, then provides a set of options that in turn do various things to the database. Selecting option “1”, for example, displays existing messages. A skeleton to get you started can be found here.

Specifically, after the user enters their name, provide the following options:

- **Show tweets** Show the *n* most recent tweets, ordered by time-stamp. You can use whatever value of *n* you like.

  There are two ways to limit the number of tweets displayed: by getting all tweets from MySQL and using a loop in your program to focus on the ones you want; or by specifying a limit in your SQL statement. It is strongly recommended that you do the latter. (Why?)

- **Show users** Display the distinct set of tweeters, ordered alphabetically.

- **Show tweets from user** When selected, prompts the user for a name, then retrieves all of the tweets from that user, ordered by time-stamp. You can format individual tweets anyway you like, but all information for a single tweet —user, id, date, message, and likes—should be included.

- **Tweet!** When selected, prompts the user for a message. The option then adds that message to the database. You can assume the author is the current user. Make sure you only include the first 140 characters!

  Note that in the schema the declarations of `likes` and `tstamp` contain the keyword `DEFAULT`, while `id` contains the keyword `AUTO_INCREMENT`. These keywords tell MySQL to generate their values automatically when new data is inserted. Thus, when inserting a new tweet, your SQL statement need only be inclusive of definitions for `user` and `message`; MySQL will generate reasonable values for all of the rest.

- **Like** Prompts the user for a tweet id. It then increments the “like” value of the specified tweet. Your increment should take place within the update statement. (Why?) To do this, set the like value to like-plus-one:

  ```
  UPDATE ... SET likes = likes + 1
  ```

  and don’t forget to add a where-clause! (Why?)

- **Quit** Close the cursor and connection, and quit the program.

Introduction to Computer Science

Introduction to Computer Science
jerome.white@nyu.edu

jerome-white

Learning computer science concepts through practice.