15.8: PROGRAMMING WITH MULTIPLE TABLES

Contributed by Chuck Severance
Clinical Associate Professor (School of Information) at University of Michigan

We will now redo the Twitter spider program using two tables, the primary keys, and the key references as described above. Here is the code for the new version of the program:

```
import urllib.request, urllib.parse, urllib.error
import twurl
import json
import sqlite3
import ssl

TWITTER_URL = 'https://api.twitter.com/1.1/friends/list.json'

conn = sqlite3.connect('friends.sqlite')
cur = conn.cursor()

cur.execute('''CREATE TABLE IF NOT EXISTS People
            (id INTEGER PRIMARY KEY, name TEXT UNIQUE, retrieved INTEGER)''')
cur.execute('''CREATE TABLE IF NOT EXISTS Follows
            (from_id INTEGER, to_id INTEGER, UNIQUE(from_id, to_id))''')

# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

while True:
    acct = input('Enter a Twitter account, or quit: ')
    if (acct == 'quit'): break
    if (len(acct) < 1):
        cur.execute('SELECT id, name FROM People WHERE retrieved = 0 LIMIT 1')
        try:
            (id, acct) = cur.fetchone()
        except:
            print('No unretrieved Twitter accounts found')
            continue
    else:
        cur.execute('SELECT id FROM People WHERE name = ? LIMIT 1',
                    (acct, ))
        try:
            id = cur.fetchone()[0]
        except:
            cur.execute('''INSERT OR IGNORE INTO People
                        (name, retrieved) VALUES (?, 0)''', (acct, ))
            conn.commit()
            if cur.rowcount != 1:
                print('Error inserting account:', acct)
                continue
            id = cur.lastrowid

    url = twurl.augment(TWITTER_URL, {'screen_name': acct, 'count': '100'})
    print('Retrieving account', acct)
```

```
        connection = urllib.request.urlopen(url, context=ctx)
    except Exception as err:
        print('Failed to Retrieve', err)
        break

    data = connection.read().decode()
    headers = dict(connection.getheaders())

    print('Remaining', headers['x-rate-limit-remaining'])

    try:
        js = json.loads(data)
    except:
        print('Unable to parse json')
        print(data)
        break

    # Debugging
    # print(json.dumps(js, indent=4))

    if 'users' not in js:
        print('Incorrect JSON received')
        print(json.dumps(js, indent=4))
        continue

    cur.execute('UPDATE People SET retrieved=1 WHERE name = ?', (acct, ))

    countnew = 0
    countold = 0
    for u in js['users']:
        friend = u['screen_name']
        print(friend)
        cur.execute('SELECT id FROM People WHERE name = ? LIMIT 1',
                    (friend, ))
        try:
            friend_id = cur.fetchone()[0]
            countold = countold + 1
        except:
            cur.execute('''INSERT OR IGNORE INTO People (name, retrieved)
                        VALUES (?, 0)''', (friend, ))
            conn.commit()
            if cur.rowcount != 1:
                print('Error inserting account:', friend)
                continue
            friend_id = cur.lastrowid
            countnew = countnew + 1
        cur.execute('''INSERT OR IGNORE INTO Follows (from_id, to_id)
                    VALUES (?, ?)''', (id, friend_id))
    print('New accounts=', countnew, ' revisited=', countold)
    print('Remaining', headers['x-rate-limit-remaining'])
    conn.commit()
cur.close()

# Code: http://www.py4e.com/code3/twfriends.py
```

This program is starting to get a bit complicated, but it illustrates the patterns that we need to use when we are using integer keys to link tables. The basic patterns are:

1. Create tables with primary keys and constraints.
2. When we have a logical key for a person (i.e., account name) and we need the `id` value for the person, depending on whether or not the person is already in the `People` table we either need to: (1) look up the person in the `People` table and retrieve the `id` value for the person or (2) add the person to the `People` table and get the `id` value for the newly added row.
3. Insert the row that captures the "follows" relationship.

We will cover each of these in turn.