This code demo the use of the artist_term.db
This is almost the same as demo_artist_term.py in
the github repository.

This is part of the Million Song Dataset project from
LabRÓSA (Columbia University) and The Echo Nest.

Copyright 2010, Thierry Bertin-Mahieux

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.

```
import os
import sys
import glob
import time
import datetime
import numpy as np
try:
    import sqlite3
except ImportError:
    print 'you need sqlite3 installed to use this program'
    sys.exit(0)

def encode_string(s):
    """
    Simple utility function to make sure a string is proper
    to be used in a SQLite query
    (different than postgresql, no N to specify unicode)
    EXAMPLE:
    That's my boy! -> 'That''s my boy!'
    """
    return '""'+s.replace("""", '""')+"""

# PATH TO artist_term.db
# CHANGE THIS TO YOUR LOCAL CONFIGURATION
# IT SHOULD BE IN THE ADDITIONAL FILES
# (you can use 'subset_artist_term.db')
dbfile = "/home/thierry/Columbia/MSongsDB/Tasks_Demos/SQLite/artist_term.db"

# connect to the SQLite database
conn = sqlite3.connect(dbfile)
# from that connection, get a cursor to do queries

c = conn.cursor()

# SCHEMA OVERVIEW
# we got 3 tables
# table1: name=artists #cols=1 (artist_id text)
# One row per artists, no duplicates, usually alphabetical order
# table2: name=terms #cols=1 (term text)
# One row per term, no duplicates, usually alphabetical order
# table3: name=artist_term #cols=2 (artist_id text, term text)
# One row per pair artist_id/term, no duplicate pairs
# Entries in table3 are constrained by table1 and table2,
# e.g. an artist_id must exist in table1 before it is used in table3.
# NOT ALL ARTISTS HAVE TERMS. They will still all be in table1, but
# some artists are not in table3 at all.

print '******************* GENERAL SQLITE DEMO *******************

# list all tables in that dataset
# note that sqlite does the actual job when we call fetchall() or fetchone()
q = "SELECT name FROM sqlite_master WHERE type='table' ORDER BY name"
res = c.execute(q)
print "* tables contained in that SQLite file/database (there should be 3):

print res.fetchall()

[(u'artist_mbtag'), (u'artist_term'), (u'artists'), (u'mbtags'), (u'terms'),]

# list all indices
q = "SELECT name FROM sqlite_master WHERE type='index' ORDER BY name"
print '  * indices in the database to make reads faster:

print res.fetchall()

[(u'idx_artist_id_mbtag'), (u'idx_artist_id_term'), (u'idx_mbtag_artist_id'), (u'idx_term_artist_id'), (u'sqlite_autoindex_artists_1'), (u'sqlite_autoindex_mbtags_1'), (u'sqlite_autoindex_terms_1'),]

print '*************** ARTISTS TABLE DEMO ***********************

# list all artists
q = "SELECT * FROM artists"
res = c.execute(q)
print '  * list all known artists in the database (display first 3):

print res.fetchall()[:3]

[(u'AR002U1187B9A637D'), (u'AR003FB1187B994355'), (u'AR006821187FB5192B')]

# list all artists that id starts with ARB
q = "SELECT artist_id FROM artists WHERE SUBSTR(artist_id,1,3)='ARB' LIMIT 2"
res = c.execute(q)

print 'list artists whose ID starts with ARB (we ask for 2 of them):

print res.fetchall()

[(u'ARBO00VV1187B9AF612',), (u'ARBO01F41187B9894C9',)]

# count all artists
q = "SELECT COUNT(artist_id) FROM artists"
res = c.execute(q)

print 'count the number of artists (with or without tags):

print res.fetchone()

(44745,)

print '*************** TERMS TABLE DEMO ***************************************

*************** TERMS TABLE DEMO ***************************************

# list all terms (=tags)
q = "SELECT * FROM terms"
res = c.execute(q)

print 'list all known terms in the database (display first 3):

print res.fetchall()[0:3]

[(u'00s'), (u'00s alternative'), (u'00s country')]

# list all terms that start with 'indie'
q = "SELECT term FROM terms WHERE SUBSTR(term,1,5)='indie' LIMIT 3"
res = c.execute(q)

print 'list terms that start with 'indie' (we ask for 3 of them):

print res.fetchall()

[(u'indie'), (u'indie acoustic'), (u'indie alternative')]

# check if a tag is in the dataset
q1 = "SELECT term FROM terms WHERE term='rock' LIMIT 1"
q2 = "SELECT term FROM terms WHERE term='abc123xyz'"
res = c.execute(q1)
res1_str = str(res.fetchone())
res = c.execute(q2)
res2_str = str(res.fetchone())

print 'we check if two tags are in the database, (the first one is):

print 'rock:', res1_str, ', abc123xyz:', res2_str

rock: (u'rock'), abc123xyz: None
```python
# similar for mtags, list all mbtags
q = "SELECT * FROM mbtags"
res = c.execute(q)
print ' * mbtags work the same as terms, e.g. list all known mbtags (display first 3):

* mbtags work the same as terms, e.g. list all known mbtags (display first 3):

print res.fetchall()[:3]

[(u'00s'), (u'00s 10s'), (u'1 13 165900 150 7672 22647 34612 48720 59280 74602 87545 95495 107182 131087 141522 153710'),]

# get one badly encoded, fix it...
# is it a problem only when we write to file??
# we want to show the usage of t.encode('utf-8') with t a term

print '*************** ARTIST / TERM TABLE DEMO ******************

*************** ARTIST / TERM TABLE DEMO ***************

# note that the Beatles artist ID is: AR6XZS61187FB4CECD

# get all tags from the Beatles
q = "SELECT term FROM artist_term WHERE artist_id='AR6XZS61187FB4CECD"
res = c.execute(q)
print ' * we get all tags applied to the Beatles (we know their artist ID), we show 4:

* we get all tags applied to the Beatles (we know their artist ID), we show 4:

print res.fetchone()[:4]

[(u'60s'), (u'acoustic'), (u'am pop'), (u'ambient'),]

# count number of tags applied to The Beatles
q = "SELECT COUNT(term) FROM artist_term WHERE artist_id='AR6XZS61187FB4CECD"
res = c.execute(q)
print ' * we count the number of unique tags applied to The Beatles:

* we count the number of unique tags applied to The Beatles:

print res.fetchone()

(30,)

# get artist IDs that have been tagged with 'jazz'
# note the encode_string function, that mostly doubles the ' sign
q = "SELECT artist_id FROM artist_term WHERE term="+encode_string('jazz')
q := " ORDER BY RANDOM() LIMIT 2"
res = c.execute(q)
print ' * we get artists tagged with 'jazz', we display 2 at random:

* we get artists tagged with 'jazz', we display 2 at random:

print res.fetchall()

[(u'ARLJWVM11F50C4DD6E'), (u'ARHBVWA1187B99172D'),]```
# count number of artists tagged with 'rock'
q = "SELECT COUNT(artist_id) FROM artist_term WHERE term="+ encode_string('rock')
res = c.execute(q)
print "* we count the number of unique artists that got term 'rock':"
print res.fetchone() 

# count number of artists mb tagged with 'rock'
q = "SELECT COUNT(artist_id) FROM artist_mbtag WHERE mbtag="+ encode_string('rock')
res = c.execute(q)
print "* something with musicbrainz tag 'rock':" 
print res.fetchone() 

# get artists that have term 'rock' but not mbtag 'rock'
q = "SELECT artist_id FROM artist_term WHERE term="+ encode_string('rock')
q += " EXCEPT SELECT artist_id FROM artist_mbtag WHERE mbtag="+ encode_string('rock')
q += " LIMIT 1"
res = c.execute(q)
print "* one artist that has term 'rock' but not mbtag 'rock':" 
print res.fetchone() 

# get artists that have no terms
# simple with the EXCEPT keyword
# other cool keywords: UNION, UNION ALL, INTERSECT
q = "SELECT artist_id FROM artists EXCEPT SELECT artist_id FROM artist_term LIMIT 1"
res = c.execute(q)
artist_notag = res.fetchone()
print "* we show an artist with no terms:" 

if artist_notag is None:
    # debug, make sure all artists have at least one tag, can be slow
    q = "SELECT * FROM artists"
    res = c.execute(q)
    allartists = map(lambda x: x[0], res.fetchall())
    for art in allartists:
        q = "SELECT COUNT(term) FROM artist_term WHERE artist_id="+art+
        res = c.execute(q)
        assert res.fetchone()[0] > 0
    print '(found no artist with no terms, we double-checked)'
else:
    print artist_notag
(u'A1A5C01187B99D70',)

# DONE
# close the cursor and the connection
# (if for some reason you added stuff to the db or alter
# a table, you need to also do a conn.commit())
c.close()
conn.close()