This code demo the use of the track_metadata.db
It is almost the same as demo_track_metadata.py
in the github repository

This is part of the Million Song Dataset project from
LabRÖSA (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 posgtresql, no N to specify unicode)
   EXAMPLE:
   That's my boy! -> 'That''s my boy'
   ""
   return """+s.replace("", "'")+""

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

# connect to the SQLite database
conn = sqlite3.connect(dbfile)

# from that connection, get a cursor to do queries

# so there is no confusion, the table name is 'songs'

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

# from that connection, get a cursor to do queries

c = conn.cursor()

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

# list all tables in that dataset

q = "SELECT name FROM sqlite_master WHERE type='table' ORDER BY name"

res = c.execute(q)

print "* tables contained in that SQLite file/database (should be only 'songs'):"

result = res.fetchall()

print result

# list all columns names from table 'songs'

q = "SELECT sql FROM sqlite_master WHERE tbl_name = 'songs' AND type = 'table'"

res = c.execute(q)

print '* get info on columns names (original table creation command):'

result = res.fetchall()[0][0]

print result

CREATE TABLE songs (track_id text PRIMARY KEY, title text, song_id text, release text, artist_id text, artist_mbid text, artist_name text, duration real, artist_similarity real, artist_hottness real, year int)

# list all indices

q = "SELECT name FROM sqlite_master WHERE type='index' AND tbl_name='songs' ORDER BY name"

res = c.execute(q)

print '* one of the index we added to the table to make things faster:'

result = res.fetchone()

print result

# find the PRIMARY KEY of a query

# by default it's called ROWID, it would have been redefined if our primary

# key

# was of type INTEGER

q = "SELECT ROWID FROM songs WHERE artist_name='The Beatles'"

res = c.execute(q)

print '* get the primary key (row id) of one entry where the artist is The Beatles:'

result = res.fetchone()

print result
Beatles:

```python
print res.fetchone()
(963783,)
```

# find an entry with The Beatles as artist_name
# returns all info (the full table row)
q = "SELECT * FROM songs WHERE artist_name='The Beatles' LIMIT 1"
res = c.execute(q)
print 'get all we have about one track from The Beatles:

* get all we have about one track from The Beatles:

```python
print res.fetchone()
(u 'TRYWZKO12903CC36DC', u 'Adelaide Town Hall Balcony With Bob Francis and Bob Rogers', u 'SOYLVG12AB018922D', u 'The Beatles Interviews 1 'June 1984 Australia''', u 'AR6XZS61187FB4CECD', u 'b10b6bfc-cf9e-42e0-be17-e2c3e1d2600d', u 'The Beatles', 197.17178999999999, 0.840406621540004, 0.8404626880269997, 0)
```

```python
print '*************** DEMOS AROUND ARTIST_ID ***************

*************** DEMOS AROUND ARTIST_ID ***************
```

```python
# query for all the artists Echo Nest ID
# the column name is 'artist_id'
# DISTINCT makes sure you get each ID returned only once
q = "SELECT DISTINCT artist_id FROM " + TABLENAME
res = c.execute(q)
artists = res.fetchall() # does the actual job of searching the db

```python
print '* found', len(artists), 'unique artist IDs, response looks like:

* found 44745 unique artist IDs, response looks like:

```python
print artists[:3]
[(u 'AR002UA1187B9A637D'), (u 'AR003FB1187B994355'), (u 'AR006821187FB5192B')]
```

# more cumbersome, get unique artist ID but with one track ID for each.
# very useful, it gives you a HDF5 file to query if you want more
# information about this artist
q = "SELECT artist_id,track_id FROM songs GROUP BY artist_id"
res = c.execute(q)
artist_track_pair = res.fetchone()

```python
print '* one unique artist with some track (chosen at random) associated with it:

* one unique artist with some track (chosen at random) associated with it:

```python
print artist_track_pair
(u 'AR002UA1187B9A637D', u 'TRYEKZM12903CDF71D')
```

# get artists having only one track in the database
q = "SELECT artist_id,track_id FROM songs GROUP BY artist_id HAVING ( COUNT( artist_id ) = 1 )"
q += " ORDER BY RANDOM()"
res = c.execute(q)
artist_track_pair = res.fetchone()

print ' * one artist that has only one track in the dataset:

* one artist that has only one track in the dataset:

(u 'ARHNQ3J1187B9AA375', u 'TREACSB128F4243587')

# get artists with no musicbrainz ID
# of course, we want only once each artist
# for demo purpose, we ask for only two at RANDOM
q = "SELECT artist_id, artist_mbid FROM songs WHERE artist_mbid=''"
q += " GROUP BY artist_id ORDER BY RANDOM() LIMIT 2"
res = c.execute(q)

print ' * two random unique artists with no musicbrainz ID:

print res.fetchall()

[(u 'ARKQGYN11F4C83D508', u ''), (u 'ARRMZEE1269FB3625C', u '')]

print '************** DEMOS AROUND NAMES ****************************

*************** DEMOS AROUND NAMES ****************************

# get all tracks by artist The Beatles
# artist name must be exact!
# the encode_string function simply deals with ' (by doubling them)
# and add ' after and before the string.
q = "SELECT track_id FROM songs WHERE artist_name="
q += encode_string('The Beatles')
res = c.execute(q)

print ' * two track id from 'The Beatles', found by looking up the artist by name:

print res.fetchall()[:2]

[(u 'TRYWZKO12903CC56DC',), (u 'TRUAGKW128F9311E03',)]

print '************** DEMOS AROUND FLOATS ****************************'
--- DEMOS AROUND FLOATS ---

```python
# get all artists whose artist familiarity is > .8
q = "SELECT DISTINCT artist_name, artist_familiarity FROM songs WHERE artist_familiarity > .8"
res = c.execute(q)
print '* one artist having familiarity >0.8:'
    * one artist having familiarity >0.8:
        print res.fetchone()
```

```python
(u '1 Giant Leap feat. Robbie Williams & Maxi Jazz', 0.84976688384900001)
```

```python
# get one artist with the highest artist_familiarity but no artist_hotttness
# notice the alias af and ah, makes things more readable
q = "SELECT DISTINCT artist_name, artist_familiarity as af, artist_hotttness as ah" +
    " FROM songs WHERE ah<0 ORDER BY af"
res = c.execute(q)
print '* get the artist with the highest familiarity that has no computed hotttness:'
    * get the artist with the highest familiarity that has no computed hotttness:
        print res.fetchone()
```

```python
(u 'Kendall "K – Mac" McCarter featuring ROCSI', -1.0, -1.0)
```

```python
# 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()
```