```
"""
Thierry Bertin-Mahieux (2010) Columbia University
tb2332@columbia.edu

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
LabROSA (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_metadat.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
```

```
84 conn = sqlite3.connect(dbfile)
85
86 # from that connection, get a cursor to do queries
87 c = conn.cursor()
88
89 # so there is no confusion, the table name is 'songs'
90 TABLENAME = 'songs'
91
92 print '*************** GENERAL SQLITE DEMO ***************************'
```

```
*************** GENERAL SQLITE DEMO ***************************
```

```
69
70 # list all tables in that dataset
71 # note that sqlite does the actual job when we call fetchall() or fetchone()
72 q = "SELECT name FROM sqlite_master WHERE type='table' ORDER BY name"
73 res = c.execute(q)
74 print "* tables contained in that SQLite file/database (should be only '
   songs'):"
```

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

```
75 print res.fetchall()
```

```
[(u'songs',)]
```

```
76
77 # list all columns names from table 'songs'
78 q = "SELECT sql FROM sqlite_master WHERE tbl_name = 'songs' AND type = '
   table'"
79 res = c.execute(q)
80 print '* get info on columns names (original table creation command):'
```

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

```
81 print res.fetchall()[0][0]
```

```
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_familiarity real, artist_hotttnesss real, year
    int)
```

```
82
83 # list all indices
84 q = "SELECT name FROM sqlite_master WHERE type='index' AND tbl_name='songs'
   ORDER BY name"
85 res = c.execute(q)
86 print '* one of the index we added to the table to make things faster:'
```

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

```
87 print res.fetchone()
```

```
(u'idx_artist_id',)
```

```
88
89 # find the PRIMARY KEY of a query
90 # by default it's called ROWID, it would have been redefined if our primary
   key
91 # was of type INTEGER
92 q = "SELECT ROWID FROM songs WHERE artist_name='The Beatles'"
93 res = c.execute(q)
94 print '* get the primary key (row id) of one entry where the artist is The
   Beatles:'
```

```
* get the primary key (row id) of one entry where the artist is The
```

```
    Beatles:
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:
```

```
print res.fetchone()
```

```
(u'TRYWZKO12903CC56DC', u'Adelaide Town Hall Balcony With Bob Francis and
    Bob Rogers', u'SOYLKVG12AB018922D', u"The Beatles Interviews 1 'June
    1984 Australia'", u'AR6XZ861187FB4CECD', u'b10bbbfc-cf9e-42e0-be17-
    e2c3e1d2600d', u'The Beatles', 197.17178999999999, 0.84040966215400004,
     0.84046268802699997, 0)
```

```

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

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

```

# 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

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

```
* found 44745 unique artist IDs, response looks like:
```

```
print artists[:3]
```

```
[(u'AR002UA1187B9A637D',), (u'AR003FB1187B994355',), (u'AR006821187FB5192B
    ',)]
```

```

# more cumbersome, get unique artist ID but with one track ID for each.
# very usefull, 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()
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:
```

```
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)
```

```
128 artist_track_pair = res.fetchone()
129 print '* one artist that has only one track in the dataset:'
```

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

```
130 print artist_track_pair
```

```
(u'ARHNQ3J1187B9AA375', u'TREACSB128F4243587')
```

```
131
132 # get artists with no musicbrainz ID
133 # of course, we want only once each artist
134 # for demo purpose, we ask for only two at RANDOM
135 q = "SELECT artist_id,artist_mbid FROM songs WHERE artist_mbid=''"
136 q += " GROUP BY artist_id ORDER BY RANDOM() LIMIT 2"
137 res = c.execute(q)
138 print '* two random unique artists with no musicbrainz ID:'
```

```
* two random unique artists with no musicbrainz ID:
```

```
139 print res.fetchall()
```

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

```
142
143
144 print '*************** DEMOS AROUND NAMES ***************************'
```

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

```
143
144 # get all tracks by artist The Beatles
145 # artist name must be exact!
146 # the encode_string function simply deals with ' (by doubling them)
147 # and add ' after and before the string.
148 q = "SELECT track_id FROM songs WHERE artist_name="
149 q += encode_string('The Beatles')
150 res = c.execute(q)
151 print "* two track id from 'The Beatles', found by looking up the artist by
      name:"
```

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

```
152 print res.fetchall()[:2]
```

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

```
153
154 # we find all release starting by letter 'T'
155 # T != t, we're just looking at albums starting with capital T
156 # here we use DISTINCT instead of GROUP BY artist_id
157 # since its fine that we find twice the same artist, as long as it is not
158 # the same (artist,release) pair
159 q = "SELECT DISTINCT artist_name,release FROM songs WHERE SUBSTR(release
      ,1,1)='T'"
160 res = c.execute(q)
161 print '* one unique artist/release pair where album starts with capital T:'
```

```
* one unique artist/release pair where album starts with capital T:
```

```
162 print res.fetchone()
```

```
(u'!Bazz feat. Leverne', u'The Nexxt Generation Mixtape')
```

```
165
166
167 print '*************** DEMOS AROUND FLOATS ***************************'
```

```
********** DEMOS AROUND FLOATS **********
```

```
166
167 # get all artists whose artist familiarity is > .8
168 q = "SELECT DISTINCT artist_name, artist_familiarity FROM songs WHERE
      artist_familiarity >.8"
169 res = c.execute(q)
170 print '* one artist having familiaryt >0.8:'
```

```
* one artist having familiaryt >0.8:
```

```
171 print res.fetchone()
```

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

```
172
173 # get one artist with the highest artist_familiarity but no
      artist_hotttnesss
174 # notice the alias af and ah, makes things more readable
175 q = "SELECT DISTINCT artist_name, artist_familiarity as af,
      artist_hotttnesss as ah"
176 q += " FROM songs WHERE ah<0 ORDER BY af"
177 res = c.execute(q)
178 print '* get the artist with the highest familiarity that has no computed
      hotttnesss:'
```

```
* get the artist with the highest familiarity that has no computed
  hotttnesss:
```

```
179 print res.fetchone()
```

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

```
180
181 # close the cursor and the connection
182 # (if for some reason you added stuff to the db or alter
183 #  a table, you need to also do a conn.commit())
184 c.close()
185 conn.close()
```