

Program: Coursera Python for Everybody Specialization

Course #4: Using Databases with Python

Created: December 2020

@author: Steven Ponce

Week 2 Assignment – Our First Database

Instructions If you don't already have it, install the SQLite Browser from <http://sqlitebrowser.org/>.

Then, create a SQLITE database or use an existing database and create a table in the database called "Ages":

```
CREATE TABLE Ages ( name VARCHAR(128), age INTEGER )
```

Then make sure the table is empty by deleting any rows that you previously inserted, and insert these rows and only these rows with the following commands:

```
DELETE FROM Ages;
```

```
INSERT INTO Ages (name, age) VALUES ('Alessandra', 13);
```

```
INSERT INTO Ages (name, age) VALUES ('Reggie', 18);
```

```
INSERT INTO Ages (name, age) VALUES ('Clove', 32);
```

```
INSERT INTO Ages (name, age) VALUES ('Joanne', 14);
```

```
INSERT INTO Ages (name, age) VALUES ('Kacey', 37);
```

```
INSERT INTO Ages (name, age) VALUES ('Titi', 24);
```

Once the inserts are done, run the following SQL command: `SELECT hex(name || age) AS X FROM Ages ORDER BY X`

Find the first row in the resulting record set and enter the long string that looks like 53656C696E613333.

In []:

Create Table Ages

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Create Table | Create Index | Print

Name	Type	Schema
▼ Tables (1)		
▼ Ages		CREATE TABLE Ages (name VARCHAR(128), age INT)
name	VARCHAR(128)	"name" VARCHAR(128)
age	INTEGER	"age" INTEGER
Indices (0)		
Views (0)		
Triggers (0)		

In []:

Empty Table

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: Ages

name	age
Filter	Filter

In []:

Adding rows to the table

Database Structure Browse Data Edit Pragmas Execute SQL		
Table:	Ages	
	name	age
	Filter	Filter
1	Alessandra	13
2	Reggie	18
3	Clove	32
4	Joanne	14
5	Kacey	37
6	Titi	24

In []:

Running the following SQL command:

```
SELECT hex(name || age) AS X FROM Ages ORDER BY X
```

Database Structure Browse Data Edit Pragmas Execute SQL		
SQL 1		
1	SELECT hex(name age) AS X FROM Ages ORDER BY X	
	X	
1	416C657373616E6472613133	
2	436C6F76653332	
3	4A6F616E6E653134	
4	4B616365793337	
5	5265676769653138	
6	546974693234	

In []:

Week 2 Assignment – Counting Email in a Database

To get credit for this assignment, perform the instructions below and upload your SQLite3 database here: (Must have a .sqlite suffix)

Hint: The top organizational count is 536.

You do not need to export or convert the database - simply upload the .sqlite file that your program creates. See the example code for the use of the connect() statement.

Counting Organizations

This application will read the mailbox data (mbox.txt) and count the number of email messages per organization (i.e. domain name of the email address) using a database with the following schema to maintain the counts.

CREATE TABLE Counts (org TEXT, count INTEGER) When you have run the program on mbox.txt upload the resulting database file above for grading. If you run the program multiple times in testing or with different files, make sure to empty out the data before each run.

You can use this code as a starting point for your application:

<http://www.py4e.com/code3/emaildb.py>.

The data file for this application is the same as in previous assignments:

<http://www.py4e.com/code3/mbox.txt>.

Because the sample code is using an UPDATE statement and committing the results to the database as each record is read in the loop, it might take as long as a few minutes to process all the data. The commit insists on completely writing all the data to disk every time it is called.

The program can be speeded up greatly by moving the commit operation outside of the loop. In any database program, there is a balance between the number of operations you execute between commits and the importance of not losing the results of operations that have not yet been committed.

In []:

In []:

```
import sqlite3

# connecting to the file
conn = sqlite3.connect('counting_emaildb.sqlite')
cur = conn.cursor()

# delete any previous table
cur.execute('DROP TABLE IF EXISTS Counts')

# create new table
cur.execute('''
CREATE TABLE Counts (org TEXT, count INTEGER)''')

# user input file name
fname = input('Enter file name: ')
if (len(fname) < 1): fname = 'mbox.txt'
fh = open(fname)
```

```

# reading ea line and splitting into name and org
for line in fh:
    if not line.startswith('From: '): continue
    pieces = line.split()
    email = pieces[1]
    domain = email.find('@')
    org = email[domain+1:len(email)]

# Updating the table
cur.execute('SELECT count FROM Counts WHERE org = ? ', (org,))
row = cur.fetchone()

if row is None:
    cur.execute('''INSERT INTO Counts (org, count)
                VALUES (?, 1)''', (org,))
else:
    cur.execute('UPDATE Counts SET count = count + 1 WHERE org = ?',
                (org,))

# top 10 results
sqlstr = 'SELECT org, count FROM Counts ORDER BY Count DESC LIMIT 10'

for row in cur.execute(sqlstr):
    print(str(row[0]), row[1])

# commit the changes and close connection
conn.commit()
cur.close()

```

In []:

Code Output:

Enter file name: mbox.txt

iupui.edu 536

umich.edu 491

indiana.edu 178

caret.cam.ac.uk 157

vt.edu 110

uct.ac.za 96

media.berkeley.edu 56

ufp.pt 28

gmail.com 25

et.gatech.edu 17

you run the program multiple times in testing or with different files, make sure to empty out the data before each run.

You can use this code as a starting point for your application:

<http://www.py4e.com/code3/tracks.zip>. The ZIP file contains the Library.xml file to be used for this assignment. You can export your own tracks from iTunes and create a database, but for the database that you turn in for this assignment, only use the Library.xml data that is provided.

To grade this assignment, the program will run a query like this on your uploaded database and look for the data it expects to see:

```
SELECT Track.title, Artist.name, Album.title, Genre.name

FROM Track JOIN Genre JOIN Album JOIN Artist

ON Track.genre_id = Genre.ID and Track.album_id = Album.id

AND Album.artist_id = Artist.id

ORDER BY Artist.name LIMIT 3
```

The expected result of the modified query on your database is: (shown here as a simple HTML table with titles)

Select Language▼

Track Artist Album Genre

Chase the Ace AC/DC Who Made Who Rock

D.T. AC/DC Who Made Who Rock

For Those About To Rock (We Salute You) AC/DC Who Made Who Rock

```
In [ ]: # week 3 assignment - Multi-Table Database - Tracks

import xml.etree.ElementTree as ET
import sqlite3

# connecting to the file
conn = sqlite3.connect('week3-musical track db.sqlite')
cur = conn.cursor()

# Delete any previous table and create a new ones
cur.executescript('''
DROP TABLE IF EXISTS Artist;
DROP TABLE IF EXISTS Genre;
DROP TABLE IF EXISTS Album;
DROP TABLE IF EXISTS Track;

CREATE TABLE Artist (
    id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    name    TEXT UNIQUE
```

```

);
CREATE TABLE Genre (
    id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    name TEXT UNIQUE
);
CREATE TABLE Album (
    id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    artist_id INTEGER,
    title TEXT UNIQUE
);
CREATE TABLE Track (
    id INTEGER NOT NULL PRIMARY KEY
        AUTOINCREMENT UNIQUE,
    title TEXT UNIQUE,
    album_id INTEGER,
    genre_id INTEGER,
    len INTEGER, rating INTEGER, count INTEGER
);
'''

# User input file name
fname = input('Enter file name: ')
if ( len(fname) < 1 ) : fname = 'Library.xml'

# <key>Track ID</key><integer>369</integer>
# <key>Name</key><string>Another One Bites The Dust</string>
# <key>Artist</key><string>Queen</string>

def lookup(d, key):
    found = False
    for child in d:
        if found : return child.text
        if child.tag == 'key' and child.text == key :
            found = True
    return None

# Create ET and find branch
stuff = ET.parse(fname)
all = stuff.findall('dict/dict/dict')
print('Dict count:', len(all))
for entry in all:

    if ( lookup(entry, 'Track ID') is None ) : continue

    name = lookup(entry, 'Name')
    artist = lookup(entry, 'Artist')
    album = lookup(entry, 'Album')
    genre = lookup(entry, 'Genre')
    count = lookup(entry, 'Play Count')
    rating = lookup(entry, 'Rating')
    length = lookup(entry, 'Total Time')

    if name is None or artist is None or genre is None or album is None :
        continue

    print(name, artist, album, count, rating, length)

    cur.execute('INSERT OR IGNORE INTO Artist (name)
        VALUES ( ? )', ( artist, ) )
    cur.execute('SELECT id FROM Artist WHERE name = ? ', (artist, ))
    artist_id = cur.fetchone()[0]

```



```
cur.execute('''INSERT OR IGNORE INTO Genre (name)
VALUES ( ? )''', ( genre, ) )
cur.execute('SELECT id FROM Genre WHERE name = ? ', (genre, ))
genre_id = cur.fetchone()[0]

cur.execute('''INSERT OR IGNORE INTO Album (title, artist_id)
VALUES ( ?, ? )''', ( album, artist_id ) )
cur.execute('SELECT id FROM Album WHERE title = ? ', (album, ))
album_id = cur.fetchone()[0]

cur.execute('''INSERT OR REPLACE INTO Track
(title, album_id, genre_id, len, rating, count)
VALUES ( ?, ?, ?, ?, ?, ? )''',
( name, album_id, genre_id, length, rating, count ) )

# commit the changes
conn.commit()
```

In []:

In []:

In []: