**PROJECT Working with Tkinter (part 2) and SQLite3 database 100 points**

**Objective** To modify your lab 7 program to work with a SQLite database you will create and work with. Basically you will include the same gui style and operational

logic & functionality of lab 7 with SQLite added in. Note that SQLite3 is built into Python (for Windows/Mac environments). All you need to do is import its module. You will see you can use a nice IDE for SQLite for any ancillary needs you like and also run SQLite in a CLI (for that interaction in windows- you may need to download SQLite for that.

***PROJECT DESCRIPTION***

This project will have you create a **myDatabase.py** file to interact with your existing **contacts.py** and **tkContacts.py** main file you modified for lab 7.

Your myDatabasefile.py will include the following database functions:

A function to perform each the following **CRUD** operations…

1. **C**reate a table
2. **R**ead from the table
3. **U**pdate the table
4. **D**elete from the table
5. Insert into the table

To assist you with SQL code for Python for performing CRUD operations as well as using cool tools for database assistance go to the following urls for great references for options,etc:

<http://www.tutorialspoint.com/sqlite/sqlite_python.htm>

<http://sqlitestudio.pl/>

<https://www.sqlite.org/cli.html> , ([for Windows CLI tools](https://www.sqlite.org/download.html) –> choose **sqlite-tools-**

**win32-x86-3320300.zip**)

**STEP 1**  **Instructions to complete this lab:**

1. Create a py file called **myDatabasefile.py**.

Fill it with with the following functions:

1. A function to create a table
2. A function to update a table
3. A function to delete from a table
4. A function to insert into a table
5. A function to read (load in) record(s) from a table

Perform the necessary SQL for each function above so when a function is called from your **tkContacts.py** file, the proper action will be executed. How you name your function and set your function arguments, if any, will be your choice.

When referring to or creating a table, use **contacts** as your **database** name. Your database **table** name will be your **first name**.

Example of a connect statement in your myDatabasefile.py file:

**import sqlite3**

**# Open database connection**

**conn = sqlite3.connect('contacts.db')**

**print ("Opened database successfully");**

Build your table with appropriate column names and datatypes you deem necessary for your logic, namely to store the contact information from lab7.

2. When completed with your **myDatabasefile.py** file, if you like, run your app to create your database table and insert the list items (records) *from* your **contacts.py** file from the **last save** you performed in lab 7 **into** your database table. Otherwise if you prefer to have your table created on a function call from your tkContacts.py file that’s fine on an initial run of your file, but again make sure you are inserting data into your table somehow that is from your last save from lab 7!

A helpful note! If you need to run thru **multiple** rows in your table (for selects, etc.), definitely use a cursor!

Example of a cursor statement/execution on an **update**:

**# Create a cursor**

**cursor = conn.cursor()**

**def updateContact (name,phone,id) :**

**# Execute update**

**cursor.execute(updateQuery, (name, phone, id))**

**STEP 2**  **Adjusting your tkContacts.py file**

1. Adjust any existing functions you see fit from your **tkContacts.py** script, so that any

updates, deletes, loads, adds, etc. are performed by the said operations defined in your **myDatabasefile.py** script.

*Note* - your GUI no longer really needs a ‘Save’ button, unless you deem it worthy to have somehow, so just delete it from the GUI and any respective callback function defined that’s glued to it. You see when the user presses your  button for example, your callback function should automatically get passed the right contact information selected by the user which in turn will make the necessary changes to the particular contact record in your database table immediately! Same logic goes for your add or even delete buttons, really.

2. Keep any remaining functions in your file you deem necessary to have the correct running app, such as selection(), buildFrame(), setList(), etc. Your program should ultimately load in all the records you inserted in STEP 1 into the listbox, similarly to how you had the records load into the listbox from the contacts.py file in lab 7.

Example database CRUD function calls in **tkContacts.py**

**# a method to add a contact to the list**

**def addContact() :**

**entry = (nameVar.get(), phoneVar.get())**

**try :**

**insertContact(entry) # call meth. in myDatabasefile**

**except :**

**messagebox.showwarning("Field cannot be blank",**

**"Please enter a value in both the fields.")**

**else :**

**setList()**

**# a method to set the contents of the contacts variable to # the selection element**

**def setList() :**

**global contacts**

**contacts = readContacts() # meth. in myDatabasefile**

**# sort the list**

**contacts.sort(key = lambda x : x[1])**

**# delete all elements from the select element**

**select.delete(0, END)**

**# insert each name from the list to the end of the # select element**

**for id, name, phone in contacts :**

**select.insert(END, name)**

Example of main in **tkContacts.py**

**# the main program**

**# initialize the application by building the GUI elements**

**root = buildFrame()**

**# initialize the database table / load records**

**createTable()**

**# set the contents of the list initially**

**setList()**

**# to keep the program from exiting**

**root.mainloop()**

**# end of program**

**STEP 3**  **Running your program and submitting appropriate files & Output**

Run your **tkContacts.py** script and perform the following functions:

1. Add in one record, namely your personal name if it doesn’t exist or a family relative and a desired contact phone number.

2. Make a change to the contact number from your newly added record.

3. Delete any record from your contact list.

*Take appropriate snapshots labeled amply in the following order, and place them into a*

*Word document as follows for full credit:*

-Your interface at runtime with an initial snapshot of your ENTIRE listbox of contacts

-Show your contact (name) being added via a print statement to the IDLE shell

-Show your contact (name/phone number) you changed via a print statement to the

IDLE shell

-Show the contact name being deleted via a print statement to the IDLE shell

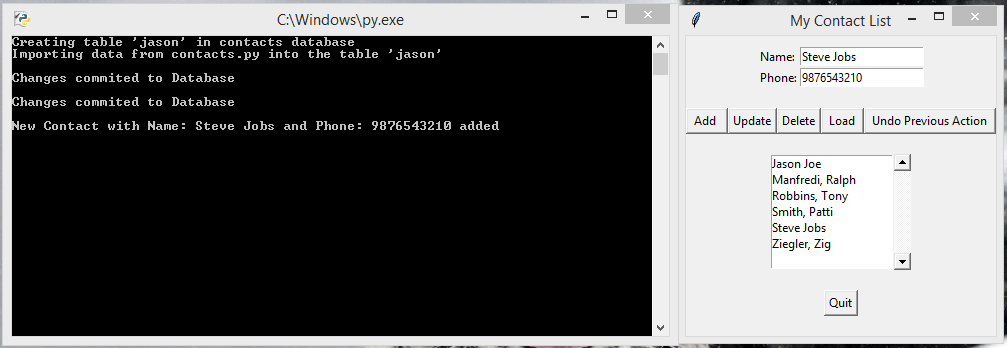
Example IDLE shell output:

James Papademas has been just added to the database.

James Papademas has had a contact number changed to xxx-xxxx

Sample GUI / console snapshots

**Screen Shot – 2: Adding a contact**

****

\*\*\*(Grads only) include as snapshots:

-Your IDLE shell print out of the old contact number and the *change* to the new contact number

-Your IDLE shell print out of all records each time from DB during any CRUD operations

-A message confirmation box asking the user ‘Are you sure you want to delete from the database the contact name’ - (make sure to show the contact name in the messagebox) whatever the contact name that’s selected.

-Include try blocks to check for possible errors that should be trapped given various CRUD operations.

\*\*\*Extra credit for all (point weights vary)

Include commit and rollback logic to rescue input errancies. Example maybe an update or

delete that occurred but needs to be undone. Create a “Rollback” button to perform a rollback. Allow a message box to ask if this will be what the user wants, that is, to rollback on any deletes/updates that may have occured before any commits. Demonstrate with some snapshots.

All:

Please include a program description in your main file along with comments for all your CRUD methods, etc.

Submit **ALL** your source **py**/**db** files plus a separate file of your snapshots into a Word Doc file into Blackboard when complete.