Zoltan Marchant

Prof. Hassebo

7/30/22

ELEC 3225-03

Group Assignment 5

```python
import sqlite3
from sqlite3 import Error
from datetime import datetime


def create_connection(db_file):
    conn = None
    try:
        conn = sqlite3.connect(db_file)
    except Error as e:
        print(e)

    return conn

def create_table(conn, table, attribute):
    cur = conn.cursor()
    try:
        print("CREATE TABLE {} ({})".format(table, attribute))
        cur.execute("CREATE TABLE {} ({})".format(table, attribute))
    except Error as e:
        print(e)

def remove_table(conn, table):
    cur = conn.cursor()
    try:
        print("DROP TABLE {}".format(table))
        cur.execute("DROP TABLE {}".format(table))
    except Error as e:
        print(e)

def get_table(conn, table):
    cur = conn.cursor()
    try:
        print("SELECT * FROM {}".format(table))
        cur.execute("SELECT * FROM {}".format(table))
        return cur.fetchall()
    except Error as e:
```

```
    except Error as e:
        print(e)

def search_table(conn, table, attribute, value):
    cur = conn.cursor()
    try:
        print("SELECT * FROM {} WHERE {} = '{}'".format(table, attribute, value))
        cur.execute("SELECT * FROM {} WHERE {} = '{}'".format(table, attribute, value))
        data = cur.fetchall()
        return data
    except Error as e:
        print(e)

def insert_row(conn, table, attributes, values):
    cur = conn.cursor()
    try:
        print("INSERT INTO {} {} VALUES {}".format(table, attributes, values))
        cur.execute("INSERT INTO {} {} VALUES {}".format(table, attributes, values))
    except Error as e:
        print(e)

def remove_row(conn, table, attribute, value):
    cur = conn.cursor()
    try:
        print("DELETE FROM '{}' WHERE '{}' = '{}'".format(table, attribute, value))
        cur.execute("DELETE FROM '{}' WHERE '{}' = '{}'".format(table, attribute, value))
    except Error as e:
        print(e)

def update_value(conn, table, id, id_value, attribute, new_value):
    cur = conn.cursor()
    try:
        print("UPDATE {} SET {} = '{}' WHERE {} = '{}'".format(table, attribute, new_value, id, id_value))
        cur.execute("UPDATE {} SET {} = '{}' WHERE {} = '{}'".format(table, attribute, new_value, id, id_value))
        data = cur.fetchall()
        return data
    except Error as e:
```

```
    except Error as e:
        print(e)

def get_table_names(conn):
    list = []
    cur = conn.cursor()
    try:
        print("SELECT name FROM sqlite_master WHERE type='table';")
        cur.execute("SELECT name FROM sqlite_master WHERE type='table';")
        tables = cur.fetchall()
        for i in tables:
            list.append(i[0])
        return list
    except Error as e:
        print(e)

def get_table_info(conn, table):
    list = []
    cur = conn.cursor()
    try:
        cur.execute("PRAGMA table_info({})".format(table))
        info = cur.fetchall()
        for i in info:
            list.append(i)
        return list
    except Error as e:
        print(e)

def find_matching_instructors(conn):
    cur = conn.cursor()
    print("SELECT INSTRUCTOR.NAME, INSTRUCTOR.SURNAME, COURSES.TITLE FROM INSTRUCTOR INNER JOIN COURSES ON INSTRUCTOR.DEPT = COURSES.DEPT")
    cur.execute("SELECT INSTRUCTOR.NAME, INSTRUCTOR.SURNAME, COURSES.TITLE FROM INSTRUCTOR INNER JOIN COURSES ON INSTRUCTOR.DEPT = COURSES.DEPT")
    fetch = cur.fetchall()
    print(fetch)

def add_course_to_schedule(conn, student_id, course_crn):
```

```
def add_course_to_schedule(conn, student_id, course_crn):
    cur = conn.cursor()
    try:
        print("INSERT INTO SCHEDULE('STUDENT_ID', 'COURSE_ID') VALUES ('{}', '{}')".format(student_id, course_crn))
        cur.execute("INSERT INTO SCHEDULE('STUDENT_ID', 'COURSE_ID') VALUES ('{}', '{}')".format(student_id, course_crn))
    except Error as e:
        print(e)

def remove_course_from_schedule(conn, student_id, course_crn ):
    cur = conn.cursor()
    try:
        print("DELETE FROM SCHEDULE WHERE STUDENT_ID = '{}' AND COURSE_ID = '{}'".format(student_id, course_crn))
        cur.execute("DELETE FROM SCHEDULE WHERE STUDENT_ID = '{}' AND COURSE_ID = '{}'".format(student_id, course_crn))
    except Error as e:
        print(e)

def print_student_schedule(conn, student_id):
    cur = conn.cursor()
    try:
        print("SELECT COURSES FROM STUDENT, COURSES, SCHEDULE WHERE STUDENT.ID = SCHEDULE.STUDENT_ID AND COURSES.CRN = SCHEDULE.COURSE_ID AND ID =
        cur.execute("SELECT COURSES FROM STUDENT, COURSES, SCHEDULE WHERE STUDENT.ID = SCHEDULE.STUDENT_ID AND COURSES.CRN = SCHEDULE.COURSE_ID AND
    except Error as e:
        print(e)

#print all students in class
#def assemble_roster():
    #assign class to instructor based off department

def add_course_to_system(conn):
    #getting list of attributes in COURSES table
    courses_attributes = []
    for i in get_table_info(conn, "COURSES"):
        courses_attributes.append(i[1])

    #getting list of CRN #'s from COURSES table (make sure no duplicates when adding)
    crn_list = []
    cur = conn.cursor()
```

```
    cur = conn.cursor()
    cur.execute("SELECT COURSES.CRN FROM COURSES")
    fetch = cur.fetchall()
    crn_list = [int(item) for t in fetch for item in t]

    #prompting user for course information and error checking inputs
    response = []
    i = 0
    while i < len(courses_attributes[0:9]):
        answer = input(courses_attributes[i] + "?: ")
        #crn
        if (i == 0):
            try:
                int(answer)
            except ValueError:
                print("Input is not an integer!")
                continue
            if ((len(answer) == 5) and (int(answer) not in crn_list)):
                response.append(int(answer))
            else:
                print("Error Adding " + courses_attributes[i] + ". (Make sure not already a CRN and is 5 digits long)")
                continue
        #title
        if (i == 1):
            if (len(answer) > 0):
                response.append(answer.upper())
            else:
                print("Error Adding " + courses_attributes[i] + ". (Make sure to enter a string)")
                continue
        #department
        if (i == 2):
            if((len(answer) == 4) and not(any([char.isdigit() for char in answer]))):
                response.append(answer.upper())
            else:
                print("Error Adding " + courses_attributes[i] + ". (Make sure answer is 4 characters long).")
                continue
        #start/end time
```

```
        #start/end time
        if (i == 3 or i == 4):
            format = '%I:%M %p'
            try:
                time = datetime.strptime(answer, format)
                response.append(time.strftime(format))
            except ValueError:
                print("Error Adding " + courses_attributes[i] + ". (Make sure in format hh:mm AM/PM).")
                continue
        #days of week
        if(i == 5):
            flag = 0
            days = []
            possible_days = ('M', 'T', 'W', 'TR', 'F')
            if (len(answer) > 0):
                for j in answer.split():
                    if (j.upper() in possible_days):
                        days.append(j.upper())
                    else:
                        flag = 1
                        break
            else:
                flag = 1
            if (flag):
                print("Error Adding " + courses_attributes[i] + ". (Make sure answer is M T W TR or F. For multiple days, seperate with a space)."
                continue
            delim = ','
            response.append(delim.join(days))
        #semester
        if (i == 6):
            semester_list = ['SPRING', 'SUMMER', 'FALL']
            if((len(answer) > 0) & (answer.upper() in semester_list)):
                response.append(answer.upper())
            else:
                print("Error Adding " + courses_attributes[i] + ". (Make sure answer is SPRING, SUMMER, or FALL).")
                continue
        #year
```

```
        #year
        if (i == 7):
            try:
                int(answer)
            except ValueError:
                print("Input is not an integer!")
                continue
            if ((len(answer) == 4)):
                response.append(int(answer))
            else:
                print("Error Adding " + courses_attributes[i] + ". (Make sure answer is a 4 digit year).")
                continue
        #credits
        if (i == 8):
            try:
                int(answer)
                response.append(int(answer))
            except ValueError:
                print("Error Adding " + courses_attributes[i] + ". (Make sure answer is an integer).")
                continue
        print(response)
        i += 1
    insert_row(conn, "COURSES", tuple(courses_attributes[0:9]), tuple(response))

def remove_course_from_system(conn, course_crn):
    cur = conn.cursor()
    try:
        print("DELETE FROM COURSES WHERE CRN = '{}'".format(course_crn))
        cur.execute("DELETE FROM COURSES WHERE CRN = '{}'".format(course_crn))
    except Error as e:
        print(e)

def remove_course_from_schedule(conn, student_id, course_crn):
    cur = conn.cursor()
    try:
        print("DELETE FROM SCHEDULE WHERE STUDENT_ID = '{}' AND COURSE_ID = '{}'".format(student_id, course_crn))
```

Dom predominantly worked on this, interfacing the functions to the database, while I created the shell for the functions. This was with the understanding that we would trade roles for assignment 6. The most difficult of this portion was learning concatenation in order to update the tables with the entered information. This especially included adding a course to the schedule.