```
from tkinter import *
from tkinter import ttk
# NEW
import mysql.connector
from mysql.connector import Error
from datetime import datetime

class StudentDB:
    headers = ['ID', 'First Name', 'Last Name', 'Email', 'Street', 'City', 'State', 'Zip', 'Phone',
'Birth', 'Sex', 'Lunch']
    student_info = []

    # DB connection
    conn = 0
    # Cursor used to traverse results
    cursor = 0
    # Stores results of last query
    query = 0

    def __init__(self):
        self.tree = None
        self.setup_db() # NEW
        self.create_widgets()

    # NEW
    def setup_db(self):
        try:
            self.conn = mysql.connector.connect(host='localhost', database='students',
user='studentadmin', password='TurtleDove')
        except mysql.connector.Error as error:
            print("Error :", error)

    def create_widgets(self):
        # ----- ROW 1 -----
        sid_label = Label(root, text='ID')
        sid_label.grid(row=0, column=0, padx=5, pady=10, sticky=W)
        self.sid_entry_value = StringVar(root, value="")
        self.sid_entry = ttk.Entry(root,
                                   textvariable=self.sid_entry_value)
        self.sid_entry.grid(row=0, column=1, padx=5, pady=10, sticky=W)

        f_name_label = Label(root, text='First Name')
        f_name_label.grid(row=0, column=2, padx=5, pady=10, sticky=W)
        self.f_name_entry_value = StringVar(root, value="")
        self.f_name_entry = ttk.Entry(root,
                                      textvariable=self.f_name_entry_value)
        self.f_name_entry.grid(row=0, column=3, padx=5, pady=10, sticky=W)

        l_name_label = Label(root, text='Last Name')
        l_name_label.grid(row=0, column=4, padx=5, pady=10, sticky=W)
        self.l_name_entry_value = StringVar(root, value="")
        self.l_name_entry = ttk.Entry(root,
                                      textvariable=self.l_name_entry_value)
        self.l_name_entry.grid(row=0, column=5, padx=5, pady=10, sticky=W)
```

```
email_label = Label(root, text='Email')
email_label.grid(row=0, column=6, padx=5, pady=10, sticky=W)
self.email_entry_value = StringVar(root, value="")
self.email_entry = ttk.Entry(root,
                    textvariable=self.email_entry_value)
self.email_entry.grid(row=0, column=7, padx=5, pady=10, sticky=W)

street_label = Label(root, text='Street')
street_label.grid(row=0, column=8, padx=5, pady=10, sticky=W)
self.street_entry_value = StringVar(root, value="")
self.street_entry = ttk.Entry(root,
                    textvariable=self.street_entry_value)
self.street_entry.grid(row=0, column=9, padx=5, pady=10, sticky=W)

# ----- 2nd ROW -----
city_label = Label(root, text='City')
city_label.grid(row=1, column=0, padx=5, pady=10, sticky=W)
self.city_entry_value = StringVar(root, value="")
self.city_entry = ttk.Entry(root,
                    textvariable=self.city_entry_value)
self.city_entry.grid(row=1, column=1, padx=5, pady=10, sticky=W)

state_label = Label(root, text='State')
state_label.grid(row=1, column=2, padx=5, pady=10, sticky=W)
self.state_entry_value = StringVar(root, value="")
self.state_entry = ttk.Entry(root,
                    textvariable=self.state_entry_value)
self.state_entry.grid(row=1, column=3, padx=5, pady=10, sticky=W)

zip_label = Label(root, text='Zip Code')
zip_label.grid(row=1, column=4, padx=5, pady=10, sticky=W)
self.zip_entry_value = StringVar(root, value="")
self.zip_entry = ttk.Entry(root,
                    textvariable=self.zip_entry_value)
self.zip_entry.grid(row=1, column=5, padx=5, pady=10, sticky=W)

phone_label = Label(root, text='Phone')
phone_label.grid(row=1, column=6, padx=5, pady=10, sticky=W)
self.phone_entry_value = StringVar(root, value="")
self.phone_entry = ttk.Entry(root,
                    textvariable=self.phone_entry_value)
self.phone_entry.grid(row=1, column=7, padx=5, pady=10, sticky=W)

birth_label = Label(root, text='Birth')
birth_label.grid(row=1, column=8, padx=5, pady=10, sticky=W)
self.birth_entry_value = StringVar(root, value="")
self.birth_entry = ttk.Entry(root,
                    textvariable=self.birth_entry_value)
self.birth_entry.grid(row=1, column=9, padx=5, pady=10, sticky=W)

# ----- 3RD ROW -----
sex_label = Label(root, text='Sex')
sex_label.grid(row=2, column=0, padx=5, pady=10, sticky=W)
```

```
        self.sex_entry_value = StringVar(root, value="")
        self.sex_entry = ttk.Entry(root,
                                   textvariable=self.sex_entry_value)
        self.sex_entry.grid(row=2, column=1, padx=5, pady=10, sticky=W)

        lunch_label = Label(root, text='Lunch')
        lunch_label.grid(row=2, column=2, padx=5, pady=10, sticky=W)
        self.lunch_entry_value = StringVar(root, value="")
        self.lunch_entry = ttk.Entry(root,
                                     textvariable=self.lunch_entry_value)
        self.lunch_entry.grid(row=2, column=3, padx=5, pady=10, sticky=W)

        add_button = ttk.Button(root, text='Add Student', command=self.add_student)
        add_button.grid(column=4, row=2, sticky=(W, E))

        update_button = ttk.Button(root, text='Update Student', command=self.update_student)
        update_button.grid(column=5, row=2, sticky=(W, E))

        delete_button = ttk.Button(root, text='Delete Student', command=self.delete_student)
        delete_button.grid(column=6, row=2, sticky=(W, E))

        # ----- TREEVIEW -----
        self.tree = ttk.Treeview(root, height=15, columns=('ID', 'First Name', 'Last Name', 'Email',
'Street', 'City', 'State', 'Zip', 'Phone', 'Birth', 'Sex', 'Lunch'), selectmode='browse')

        self.tree.grid(row=3, column=0, columnspan=17)
        self.tree['show'] = 'headings'

        i = 1
        for col in self.headers:
            num = f'#{i}' # Format string to produce incrementing numbers
            self.tree.heading(num, text=col)
            self.tree.column(num, width=115)
            i += 1

        self.update_table()

    # Check that there is an entry in all entries required
    # Verify if student id is required
    def all_entries_filled(self, sid_required):
        if len(self.f_name_entry_value.get()) == 0 or len(self.l_name_entry_value.get()) == 0 or
len(self.email_entry_value.get()) == 0 or len(self.street_entry_value.get()) == 0 or
len(self.city_entry_value.get()) == 0 or len(self.state_entry_value.get()) == 0 or
len(self.zip_entry_value.get()) == 0 or len(self.phone_entry_value.get()) == 0 or
len(self.birth_entry_value.get()) == 0 or len(self.sex_entry_value.get()) == 0 or
len(self.lunch_entry_value.get()) == 0:
            return False
        elif sid_required:
            if len(self.sid_entry_value.get()) == 0:
                return False
            else:
                return True
        else:
            return True
    # NEW Executes the query and fetches result from
    # the query if it is expected
    def execute_query(self, result_expected):
        try:
            # Get connection for cursor
            self.cursor = self.conn.cursor()
            self.cursor.execute(self.query)
            # Check if a result is expected from the query
            if result_expected:
                self.student_info = self.cursor.fetchall()

            # Move changes to DB
            self.conn.commit()
            # Reset results and close the cursor
            self.cursor.close()
        except mysql.connector.Error as error:
            print("Error :", error)

    # NEW clear the tree and then get updated data
    def update_table(self):
        for i in self.tree.get_children():
            self.tree.delete(i)

        # Get all student data for the treeview
        self.query = "SELECT student_id, first_name, last_name, email, street, city, state, zip,
phone, birth_date, sex, lunch_cost FROM students"

        self.execute_query(True)

        i = 1
        for stud_info in self.student_info:
            num = f'#{i}'
            self.tree.insert('', 'end', values=stud_info)
            i += 1

    def add_student(self):
        # Check if connected to DB and all required entries are filled
        if(self.conn.is_connected() and not self.all_entries_filled(False)):
            self.popup_msg("Enter All the Student Data")
        else:
            # Get the current time and format it to fit what
            # MySQL expects
            now_time = datetime.now()
            format_date = now_time.strftime('%Y-%m-%d %H:%M:%S')
            f_name = self.f_name_entry_value.get()
            l_name = self.l_name_entry_value.get()
            email = self.email_entry_value.get()
            street = self.street_entry_value.get()
            city = self.city_entry_value.get()
            state = self.state_entry_value.get()
            zip = self.zip_entry_value.get()
            phone = self.phone_entry_value.get()
            birth = self.birth_entry_value.get()
            sex = self.sex_entry_value.get()
            lunch = self.lunch_entry_value.get()

            self.query = f"INSERT INTO students VALUES( NULL, '{f_name}', '{l_name}', '{email}',
'{street}', '{city}', '{state}', {zip}, '{phone}', '{birth}', '{sex}', '{format_date}', {lunch})"

            self.execute_query(False)

            # Update the table
            self.update_table()

    def update_student(self):
        if(self.conn.is_connected() and not self.all_entries_filled(True)):
            self.popup_msg("Enter All the Student Data")
        else:
            sid = self.sid_entry_value.get()
            f_name = self.f_name_entry_value.get()
            l_name = self.l_name_entry_value.get()
            email = self.email_entry_value.get()
            street = self.street_entry_value.get()
            city = self.city_entry_value.get()
            state = self.state_entry_value.get()
            zip = self.zip_entry_value.get()
            phone = self.phone_entry_value.get()
            birth = self.birth_entry_value.get()
            sex = self.sex_entry_value.get()
            lunch = self.lunch_entry_value.get()

            self.query = f"UPDATE students SET first_name = '{f_name}', last_name = '{l_name}',
email = '{email}', street = '{street}', city = '{city}', state = '{state}', zip = {zip}, phone = '{phone}',
birth_date = '{birth}', sex = '{sex}', lunch_cost = {lunch} WHERE student_id = {sid}"
            self.execute_query(False)

            # Update the table and don't ask for a result
            self.update_table()

    def delete_student(self):
        if(self.conn.is_connected()):
            if len(self.sid_entry_value.get()) == 0:
                self.popup_msg("Enter A Student ID Number")
            else:
                sid = self.sid_entry_value.get()
                self.query = f"DELETE FROM students WHERE student_id = {sid}"
                self.execute_query(False)

                # Update the table
                self.update_table()

    # Used to create a popup dialog
    def popup_msg(self, msg):
        popup = Tk()
        popup.geometry("235x85")
        popup.resizable(width=False, height=False)
        popup.wm_title("Enter All Values")
        err_msg = Text(popup, font=("Verdana", 16))
        err_msg.insert(INSERT, msg)
        err_msg.pack()
        ok_but = ttk.Button(popup, text="OK", command=popup.destroy)
        ok_but.place(relx=.5, rely=.8, anchor="center")
        popup.mainloop()

root = Tk()
root.geometry("1400x600")
root.title("Student Database")
student_db = StudentDB()
root.mainloop()
```