Student management system

```
import tkinter as tk
from tkinter import ttk, messagebox
import sqlite3

# --- Database Setup ---
conn = sqlite3.connect('student_management.db')
cursor = conn.cursor()
cursor.execute('''
    CREATE TABLE IF NOT EXISTS students (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        student_id TEXT UNIQUE,
        name TEXT,
        grade TEXT,
        gender TEXT,
        dob TEXT,
        degree TEXT,
        stream TEXT,
        phone TEXT,
        email TEXT,
        address TEXT
    )
''')
conn.commit()

# Insert sample data if table empty
cursor.execute("SELECT COUNT(*) FROM students")
if cursor.fetchone()[0] == 0:
    sample_students = [
        ('S001', 'Alice Johnson', '10', 'Female', '2007-05-12', 'High School', 'Science', '1234567890',
'alice@example.com', '123 Green St'),
        ('S002', 'Bob Smith', '11', 'Male', '2006-08-23', 'High School', 'Commerce', '2345678901',
'bob@example.com', '456 Blue St'),
        ('S003', 'Charlie Lee', '12', 'Male', '2005-12-01', 'High School', 'Arts', '3456789012',
'charlie@example.com', '789 Red St'),
    ]
    cursor.executemany('''
        INSERT INTO students
        (student_id, name, grade, gender, dob, degree, stream, phone, email, address)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    ''', sample_students)
    conn.commit()

# --- Main Application ---
class StudentManagementApp:
    def _init_(self, root):
        self.root = root
        self.root.title("STUDENT MANAGEMENT SYSTEM")
        self.root.geometry("1150x700")
        self.root.configure(bg="skyblue")

        # Title
    title_lbl = tk.Label(root, text="STUDENT MANAGEMENT SYSTEM", font=("Arial Black", 24, "bold"),
                    bg="skyblue", fg="black")
    title_lbl.pack(pady=10)

    # Frame for form inputs
    form_frame = tk.Frame(root, bg="skyblue")
    form_frame.pack(pady=10, padx=20, fill=tk.X)

    # Labels and entries
    labels = ["Student ID", "Student Name", "Grade", "Gender", "DOB (YYYY-MM-DD)",
          "Degree", "Stream", "Phone No", "Email", "Address"]
    self.entries = {}

    # Gender dropdown colors
    gender_options = ['Male', 'Female', 'Other']

    for i, label in enumerate(labels):
      row = i // 2
      col = (i % 2) * 2
      tk.Label(form_frame, text=label + ":", font=("Arial", 12, "bold"), bg="skyblue").grid(row=row,
column=col, padx=10, pady=6, sticky=tk.W)

      if label == "Gender":
        combo = ttk.Combobox(form_frame, values=gender_options, state="readonly", font=("Arial",
11))
        combo.grid(row=row, column=col+1, padx=10, pady=6, sticky=tk.W)
        combo.config(background="violet")  # May not always reflect in ttk, depends on OS
        self.entries[label] = combo
      elif label == "Address":
        txt = tk.Text(form_frame, width=30, height=3, font=("Arial", 11))
        txt.grid(row=row, column=col+1, padx=10, pady=6, sticky=tk.W)
        self.entries[label] = txt
      else:
        ent = tk.Entry(form_frame, font=("Arial", 12))
        ent.grid(row=row, column=col+1, padx=10, pady=6, sticky=tk.W)
        self.entries[label] = ent

    # Buttons Frame
    btn_frame = tk.Frame(root, bg="skyblue")
    btn_frame.pack(pady=10)

    btn_specs = [
      ("Add", self.add_student),
      ("Delete", self.delete_student),
      ("Update", self.update_student),
      ("Refresh", self.refresh_data),
      ("Clear", self.clear_form),
      ("Reset", self.reset_db),
      ("Modify", self.modify_student)
    ]

    for i, (text, cmd) in enumerate(btn_specs):
      btn = tk.Button(btn_frame, text=text, command=cmd,
                font=("Arial", 12, "bold"), fg="white", bg="darkgreen",
                width=10, relief=tk.RAISED, cursor="hand2")
      btn.grid(row=0, column=i, padx=8)
```

```
        # Search Frame
        search_frame = tk.Frame(root, bg="skyblue")
        search_frame.pack(pady=10)

        tk.Label(search_frame, text="Search by Name:", font=("Arial", 12, "bold"),
bg="skyblue").grid(row=0, column=0, padx=5)
        self.search_entry = tk.Entry(search_frame, font=("Arial", 12), width=30)
        self.search_entry.grid(row=0, column=1, padx=5)
        search_btn = tk.Button(search_frame, text="Search", command=self.search_student,
                               font=("Arial", 12, "bold"), fg="white", bg="red", width=10)
        search_btn.grid(row=0, column=2, padx=5)

        # Treeview Frame
        tree_frame = tk.Frame(root)
        tree_frame.pack(padx=20, pady=10, fill=tk.BOTH, expand=True)

        columns = ("ID", "Student ID", "Name", "Grade", "Gender", "DOB", "Degree",
                   "Stream", "Phone", "Email", "Address")

        style = ttk.Style()
        style.theme_use("clam")
        style.configure("Treeview.Heading", font=("Arial", 12, "bold"), background="violet",
foreground="white")
        style.configure("Treeview", font=("Arial", 11), rowheight=25)
        style.map('Treeview', background=[('selected', 'yellow')], foreground=[('selected', 'black')])

        self.tree = ttk.Treeview(tree_frame, columns=columns, show='headings')

        for col in columns:
            width = 100 if col != "Address" else 200
            self.tree.heading(col, text=col)
            self.tree.column(col, width=width, anchor=tk.CENTER)

        self.tree.pack(fill=tk.BOTH, expand=True)
        self.tree.bind("<<TreeviewSelect>>", self.on_tree_select)

        self.refresh_data()

    # Clear form fields
    def clear_form(self):
        for key, widget in self.entries.items():
            if isinstance(widget, tk.Text):
                widget.delete('1.0', tk.END)
            else:
                widget.delete(0, tk.END)

    # Refresh data in treeview
    def refresh_data(self):
        for row in self.tree.get_children():
            self.tree.delete(row)
        cursor.execute("SELECT * FROM students")
        for row in cursor.fetchall():
            self.tree.insert("", tk.END, values=row)
        self.clear_form()

    # Add student
    def add_student(self):
        data = self.get_form_data()
        if not data:
            return
        try:
            cursor.execute('''
                INSERT INTO students
                (student_id, name, grade, gender, dob, degree, stream, phone, email, address)
                VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
            ''', data)
            conn.commit()
            messagebox.showinfo("Success", "Student added successfully!")
            self.refresh_data()
        except sqlite3.IntegrityError:
            messagebox.showerror("Error", "Student ID must be unique.")
        except Exception as e:
            messagebox.showerror("Error", str(e))

    # Delete student
    def delete_student(self):
        selected = self.tree.selection()
        if not selected:
            messagebox.showwarning("Warning", "Please select a student to delete.")
            return
        student_db_id = self.tree.item(selected[0])['values'][0]
        cursor.execute("DELETE FROM students WHERE id=?", (student_db_id,))
        conn.commit()
        messagebox.showinfo("Deleted", "Student deleted successfully.")
        self.refresh_data()

    # Update student
    def update_student(self):
        selected = self.tree.selection()
        if not selected:
            messagebox.showwarning("Warning", "Please select a student to update.")
            return
        student_db_id = self.tree.item(selected[0])['values'][0]
        data = self.get_form_data()
        if not data:
            return
        try:
            cursor.execute('''
                UPDATE students SET
                student_id=?, name=?, grade=?, gender=?, dob=?, degree=?, stream=?, phone=?, email=?,
address=?
                WHERE id=?
            ''', data + (student_db_id,))
            conn.commit()
            messagebox.showinfo("Updated", "Student updated successfully.")
            self.refresh_data()
        except sqlite3.IntegrityError:
            messagebox.showerror("Error", "Student ID must be unique.")
        except Exception as e:
            messagebox.showerror("Error", str(e))

    # Modify (alias for update)
    def modify_student(self):
        self.update_student()
    # Reset database (clear all)
    def reset_db(self):
        confirm = messagebox.askyesno("Confirm Reset", "Are you sure you want to delete ALL student
records?")
        if confirm:
            cursor.execute("DELETE FROM students")
            conn.commit()
            self.refresh_data()

    # Search student by name
    def search_student(self):
        query = self.search_entry.get().strip()
        if not query:
            self.refresh_data()
            return
        self.tree.delete(*self.tree.get_children())
        cursor.execute("SELECT * FROM students WHERE name LIKE ?", ('%' + query + '%',))
        rows = cursor.fetchall()
        for row in rows:
            self.tree.insert("", tk.END, values=row)
        if not rows:
            messagebox.showinfo("No Results", "No students found matching your search.")

    # Get data from form, return tuple or None if validation fails
    def get_form_data(self):
        sid = self.entries["Student ID"].get().strip()
        name = self.entries["Student Name"].get().strip()
        grade = self.entries["Grade"].get().strip()
        gender = self.entries["Gender"].get()
        dob = self.entries["DOB (YYYY-MM-DD)"].get().strip()
        degree = self.entries["Degree"].get().strip()
        stream = self.entries["Stream"].get().strip()
        phone = self.entries["Phone No"].get().strip()
        email = self.entries["Email"].get().strip()
        address = self.entries["Address"].get('1.0', tk.END).strip()

        # Simple validations
        if not all([sid, name, grade, gender, dob, degree, stream, phone, email, address]):
            messagebox.showwarning("Validation Error", "Please fill all fields.")
            return None
        # Optional: add further validation (email format, phone digits, dob format)

        return (sid, name, grade, gender, dob, degree, stream, phone, email, address)

    # Load selected student from treeview to form
    def on_tree_select(self, event):
        selected = self.tree.selection()
        if not selected:
            return
        values = self.tree.item(selected[0])['values']
        # Map values to form fields (skip DB ID index 0)
        keys = ["Student ID", "Student Name", "Grade", "Gender", "DOB (YYYY-MM-DD)",
                "Degree", "Stream", "Phone No", "Email", "Address"]
        for i, key in enumerate(keys):
            if key == "Address":
                self.entries[key].delete('1.0', tk.END)
                self.entries[key].insert(tk.END, values[i+1])
            elif key == "Gender":
                self.entries[key].set(values[i+1])
            else:
                self.entries[key].delete(0, tk.END)
                self.entries[key].insert(0, values[i+1])

# --- Run the app ---
if _name_ == "_main_":
    root = tk.Tk()
    app = StudentManagementApp(root)
    root.mainloop()
```