**Q) Design a database application using python GUI to modify specified record of an product for product id using database and display the modified record . (product id, name, quantity, price and star rating)**

1) Import Tkinter and sqlite

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

2) Create database

```
# Database setup
def create_table():
    conn = sqlite3.connect('products.db')
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS products
                      (product_id INTEGER PRIMARY KEY,
                       name TEXT,
                       quantity INTEGER,
                       price_per_unit REAL,
                       rating REAL)''')
    conn.commit()
    conn.close()
```

3) Define function add_product, get_all_products, get_product, update_product, delete_product, modify_record, save_changes, clear_fields, update_product_list, update_total_price, delete_record and select_product.

A) add_product

```
# Function to add a product to the database with error handling
def add_product(product_id, name, quantity, price_per_unit, rating):
    conn = sqlite3.connect('products.db')
    cursor = conn.cursor()
    try:
        cursor.execute('''INSERT INTO products (product_id, name, quantity, price_per_unit, rating)
                          VALUES (?, ?, ?, ?, ?)''', (product_id, name, quantity, price_per_unit, rating))
        conn.commit()
        messagebox.showinfo("Success", "Product added successfully!")
        update_product_list()  # Update product list after adding
        update_total_price()    # Update total price after adding
    except sqlite3.IntegrityError:
        messagebox.showerror("Error", "Product ID already exists. Use a unique Product ID.")
    except Exception as e:
        messagebox.showerror("Database Error", str(e))
    finally:
        conn.close()
```

B) Get_all_products

```
# Function to retrieve all products from the database
def get_all_products():
    conn = sqlite3.connect('products.db')
    cursor = conn.cursor()
    cursor.execute('''SELECT * FROM products''')
    products = cursor.fetchall()
    conn.close()
    return products
```

C) Get_product

```
# Function to retrieve a product's details from the database
def get_product(product_id):
    conn = sqlite3.connect('products.db')
    cursor = conn.cursor()
    cursor.execute('''SELECT * FROM products WHERE product_id = ?''', (product_id,))
    product = cursor.fetchone()
    conn.close()
    return product
```

D) Update_product

```
# Function to update a product's details in the database
def update_product(product_id, name, quantity, price_per_unit, rating):
    conn = sqlite3.connect('products.db')
    cursor = conn.cursor()
    cursor.execute('''UPDATE products SET name = ?, quantity = ?, price_per_unit = ?, rating = ?
                      WHERE product_id = ?''', (name, quantity, price_per_unit, rating, product_id))
    conn.commit()
    conn.close()
    update_product_list()  # Update product list after modifying
    update_total_price()   # Update total price after modifying
```

E) delete_product

```
# Function to delete a product from the database
def delete_product(product_id):
    conn = sqlite3.connect('products.db')
    cursor = conn.cursor()
    cursor.execute('''DELETE FROM products WHERE product_id = ?''', (product_id,))
    conn.commit()
    conn.close()
    update_product_list()  # Update product list after deleting
    update_total_price()   # Update total price after deleting
```

F) modify_record

```
# Function to modify product (opens a new pop-up window)
def modify_record():
    product_id = product_id_entry.get()
    if product_id:
        try:
            product = get_product(int(product_id))
            if product:
                # Open a new window for modifying the record
                modify_window = tk.Toplevel(root)
                modify_window.title("Modify Product")

                tk.Label(modify_window, text="Product ID:").grid(row=0, column=0)
                tk.Label(modify_window, text=product[0]).grid(row=0, column=1)  # Product ID is not editable

                tk.Label(modify_window, text="Name:").grid(row=1, column=0)
                name_entry = tk.Entry(modify_window)
                name_entry.grid(row=1, column=1)
                name_entry.insert(0, product[1])

                tk.Label(modify_window, text="Quantity:").grid(row=2, column=0)
                quantity_entry = tk.Entry(modify_window)
                quantity_entry.grid(row=2, column=1)
                quantity_entry.insert(0, product[2])

                tk.Label(modify_window, text="Price per Unit:").grid(row=3, column=0)
                price_entry = tk.Entry(modify_window)
                price_entry.grid(row=3, column=1)
                price_entry.insert(0, product[3])

                tk.Label(modify_window, text="Rating:").grid(row=4, column=0)
                rating_entry = tk.Entry(modify_window)
                rating_entry.grid(row=4, column=1)
                rating_entry.insert(0, product[4])
```

G) save_changes

```
                # Function to save changes in the pop-up window
                def save_changes():
                    try:
                        update_product(
                            int(product[0]),  # product_id is fixed
                            name_entry.get(),
                            int(quantity_entry.get()),
                            float(price_entry.get()),
                            float(rating_entry.get())
                        )
                        messagebox.showinfo("Success", "Record updated successfully!")
                        modify_window.destroy()
                    except ValueError:
                        messagebox.showerror("Input Error", "Please enter valid data.")

                # Save button inside the pop-up window
                save_button = tk.Button(modify_window, text="Save", command=save_changes)
                save_button.grid(row=5, column=0, columnspan=2)

            else:
                messagebox.showerror("Error", "Product ID not found.")
        except ValueError:
            messagebox.showerror("Input Error", "Please enter a valid Product ID.")
    else:
        messagebox.showerror("Input Error", "Please enter a Product ID.")
```

H) clear_fields

```
# Function to clear input fields
def clear_fields():
    product_id_entry.delete(0, tk.END)
    name_entry.delete(0, tk.END)
    quantity_entry.delete(0, tk.END)
    price_entry.delete(0, tk.END)
    rating_entry.delete(0, tk.END)
```

I) update_product_list

```
# Function to update the product list
def update_product_list():
    product_listbox.delete(0, tk.END)  # Clear the listbox
    products = get_all_products()
    for product in products:
        total_price = product[2] * product[3]  # Quantity * Price per unit
        product_listbox.insert(tk.END, f"ID: {product[0]}, Name: {product[1]}, Qty: {product[2]}, Price per Unit: {product[3]:.2f}, Total Price: {total_price:.2f}, Rating: {product[4]}")
```

J) update_total_price

```
# Function to calculate and display the total price of all products
def update_total_price():
    total_price = sum(product[2] * product[3] for product in get_all_products())  # Sum up total prices
    total_price_label.config(text=f"Total Price of All Products: {total_price:.2f}")
```

K) delete_record

```
# Function to delete a record from the database
def delete_record():
    product_id = product_id_entry.get()
    if product_id:
        try:
            product = get_product(int(product_id))
            if product:
                delete_product(int(product_id))
                messagebox.showinfo("Success", f"Product ID {product_id} deleted successfully!")
                clear_fields()  # Clear fields after deletion
            else:
                messagebox.showerror("Error", "Product ID not found.")
        except ValueError:
            messagebox.showerror("Input Error", "Please enter a valid Product ID.")
    else:
        messagebox.showerror("Input Error", "Please enter a Product ID.")
```

L) select_product

```
# Function to populate fields when selecting an item from the list
def select_product(event):
    selected_product = product_listbox.curselection()
    if selected_product:
        index = selected_product[0]
        product_info = product_listbox.get(index)
        product_id = int(product_info.split(", ")[0].split(": ")[1])
        product = get_product(product_id)

        if product:
            product_id_entry.delete(0, tk.END)
            product_id_entry.insert(0, product[0])
            name_entry.delete(0, tk.END)
            name_entry.insert(0, product[1])
            quantity_entry.delete(0, tk.END)
            quantity_entry.insert(0, product[2])
            price_entry.delete(0, tk.END)
            price_entry.insert(0, product[3])
            rating_entry.delete(0, tk.END)
            rating_entry.insert(0, product[4])
```

4) GUI setup

```
# GUI Setup
root = tk.Tk()
root.title("Product Record Modifier")
root.configure(bg='#f0f8ff')  # Light blue background
```

a) Add Labels and input fields with styling.

```
# Labels and input fields with styling
tk.Label(root, text="Product ID:", bg='#f0f8ff', font=("Arial", 12)).grid(row=0, column=0)
product_id_entry = tk.Entry(root, font=("Arial", 12))
product_id_entry.grid(row=0, column=1)

tk.Label(root, text="Name:", bg='#f0f8ff', font=("Arial", 12)).grid(row=1, column=0)
name_entry = tk.Entry(root, font=("Arial", 12))
name_entry.grid(row=1, column=1)

tk.Label(root, text="Quantity:", bg='#f0f8ff', font=("Arial", 12)).grid(row=2, column=0)
quantity_entry = tk.Entry(root, font=("Arial", 12))
quantity_entry.grid(row=2, column=1)

tk.Label(root, text="Price per Unit:", bg='#f0f8ff', font=("Arial", 12)).grid(row=3, column=0)
price_entry = tk.Entry(root, font=("Arial", 12))
price_entry.grid(row=3, column=1)

tk.Label(root, text="Star Rating:", bg='#f0f8ff', font=("Arial", 12)).grid(row=4, column=0)
rating_entry = tk.Entry(root, font=("Arial", 12))
rating_entry.grid(row=4, column=1)
```

b) Product listbox

```
# Product listbox
product_listbox = tk.Listbox(root, width=70, font=("Arial", 12))
product_listbox.grid(row=5, column=0, columnspan=5)
```

c) Total price label

```
# Total price label
total_price_label = tk.Label(root, text="Total Price of All Products: 0.00", bg='#f0f8ff', font=("Arial", 12))
total_price_label.grid(row=6, column=1, columnspan=2)
```

d) Buttons with styling

```
# Buttons with styling
tk.Button(root, text="Add Product", command=lambda: add_product(
    int(product_id_entry.get()),
    name_entry.get(),
    int(quantity_entry.get()),
    float(price_entry.get()),
    float(rating_entry.get())
), bg='#add8e6', font=("Arial", 12)).grid(row=7, column=0)

tk.Button(root, text="Modify Product", command=modify_record, bg='#add8e6', font=("Arial", 12)).grid(row=7, column=1)
tk.Button(root, text="Delete Product", command=delete_record, bg='#add8e6', font=("Arial", 12)).grid(row=7, column=2)
```

e) select event

```
# Binding select event
product_listbox.bind('<<ListboxSelect>>', select_product)
```

f) Create the database table

```
# Create the database table
create_table()
```

g) Initialize the product list and total price

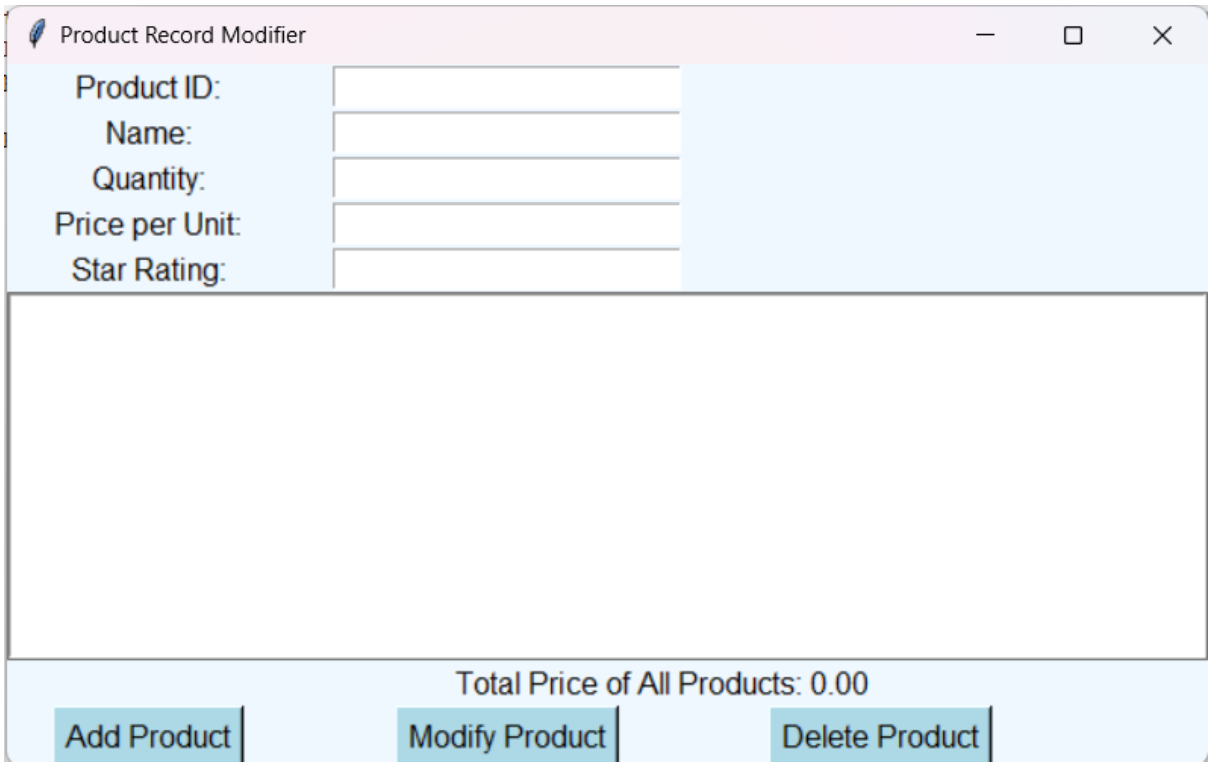
```
# Initialize the product list and total price
update_product_list()
update_total_price()
```

h) Run the application

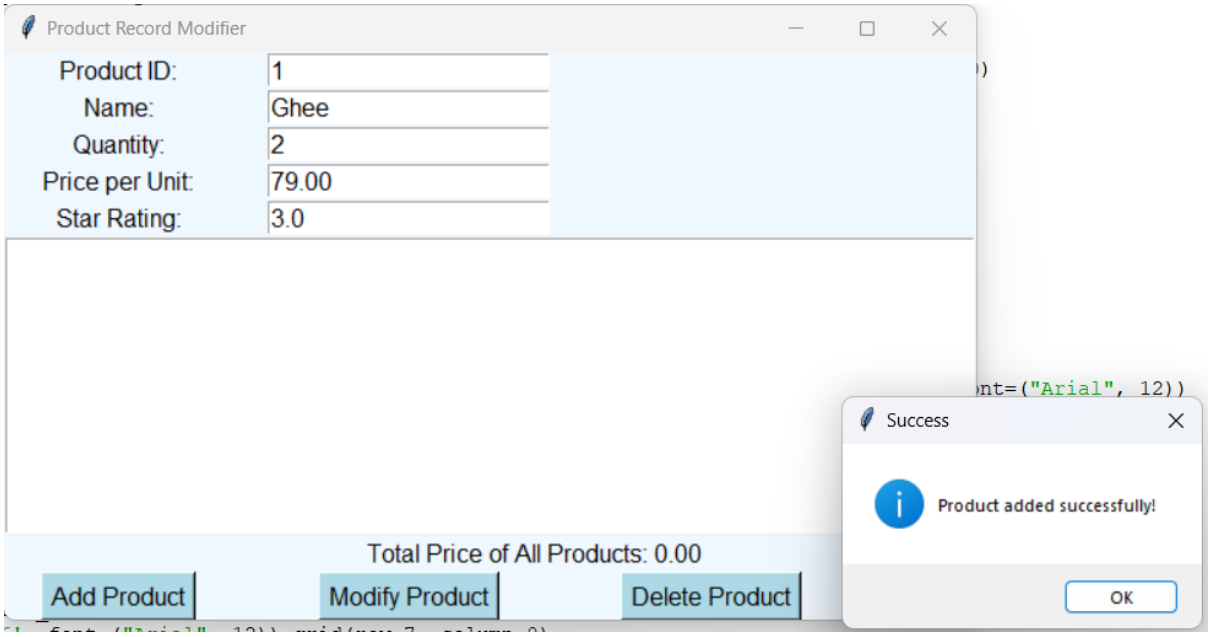
```
# Run the application
root.mainloop()
```

OUTPUT:

1) GUI INTERFACE

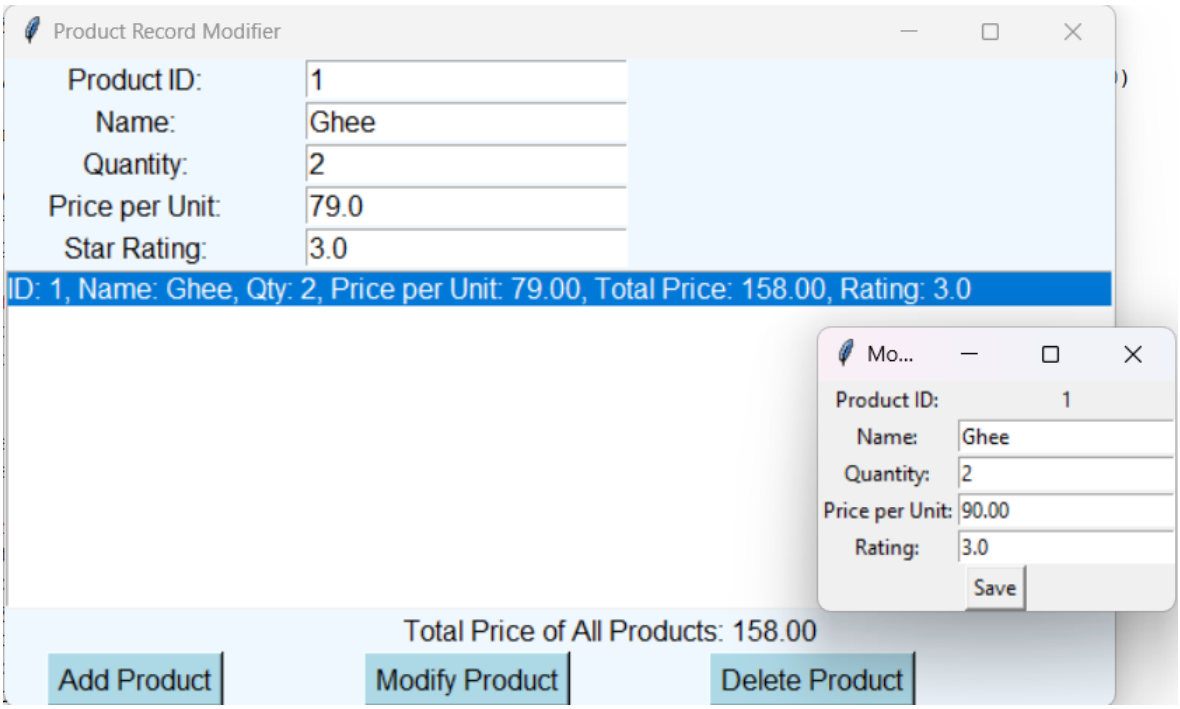



2) ADD PRODUCT

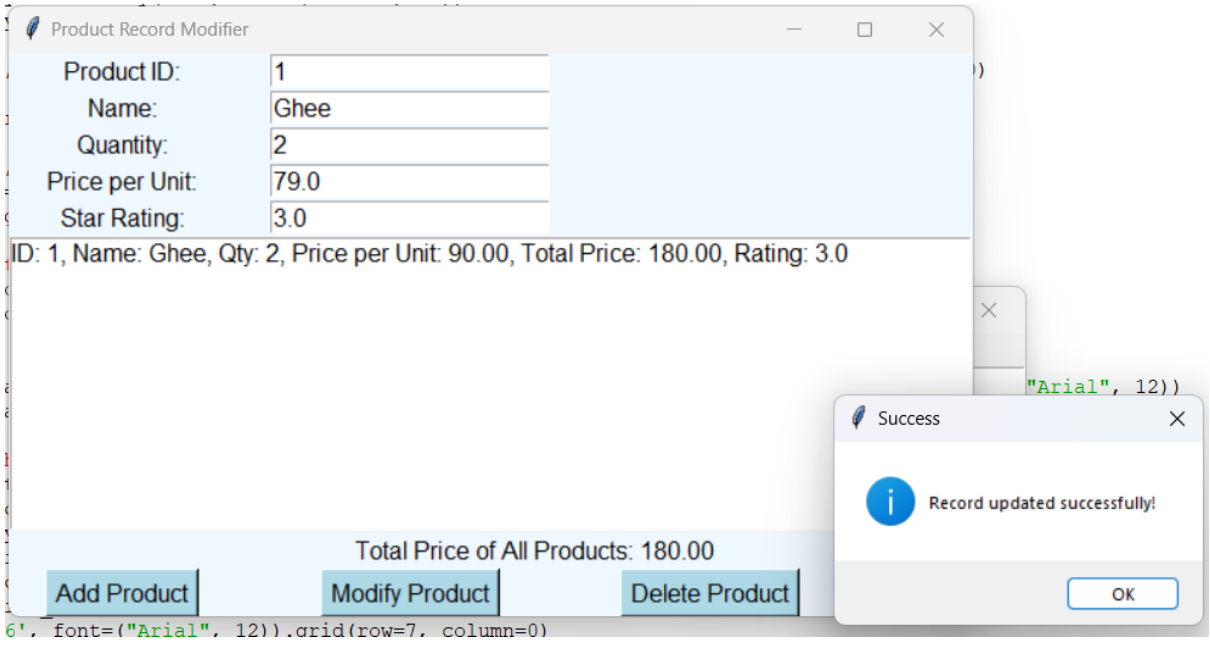


3) MODIFY PRODUCT

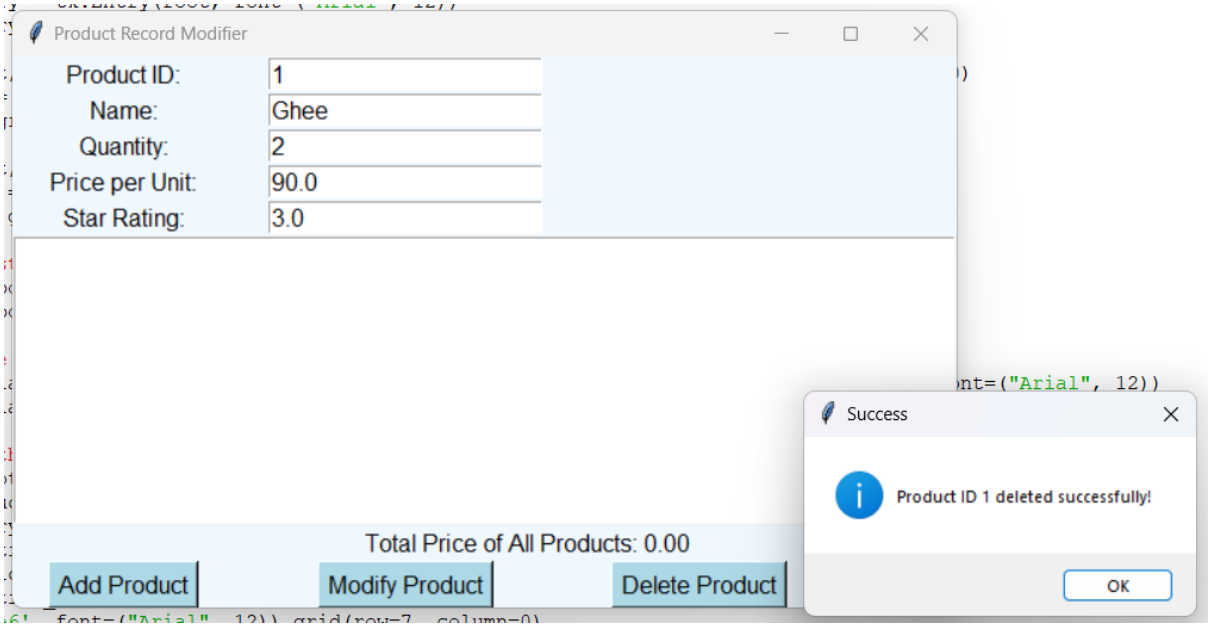
A) CHANGING PRODUCT PRICE




B) MODIFIED SUCCESSFULLY



4) DELETE PRODUCT




## CONCLUSION: -

Our application allows users to:

Add products with details such as quantity, price, and rating. Modify or delete existing products. Display all products in a list. Calculate and display the total price of all products. All of these operations are performed through an intuitive graphical interface built using “Tkinter”, and data is stored securely using SQLite.