

# **Hito 2. 1<sup>er</sup> Trimestre.**

## **Sistemas de gestión empresarial**

**David Millán Domínguez**

**07/12/2023**

# Índice

Base de datos.....	3
Diseño.....	3
Main.....	4
Añadir .....	4
ClienteAdd.....	5
PedidoAdd.....	6
ProductoAdd .....	8
Actualizar .....	9
ClienteUpdate .....	9
PedidoUpdate .....	10
ProductoUpdate.....	11
Filtrar .....	12
ClienteFilter .....	13
ProductoFilter .....	14
PedidoFilter .....	15
Ordenar tabla.....	15
Eliminar.....	16
Exportar .....	17
Gráfico .....	18
GraficoClientes .....	19
GraficoPedidos .....	20
GraficoProductos.....	21

# Base de datos

La base de datos que conecta con la aplicación costa de tres tablas: Clientes, Pedidos y Productos

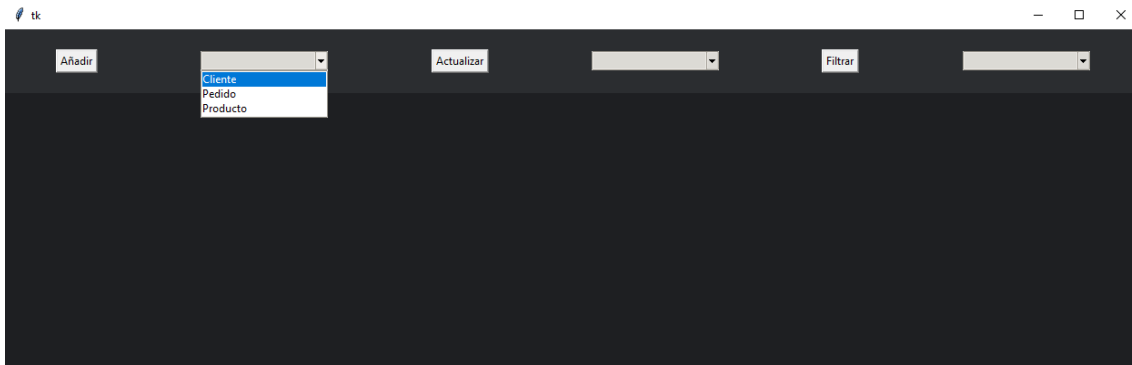
## Diseño

Mi aplicación tiene diez ventanas en total. El código de cada ventana está en un archivo Python diferente. De esta forma el código queda mucho más limpio y ordenado.

Ahora empezaré a explicar el código de mi aplicación.

# Main

La ventana Main cuenta con un frame en la parte superior que incluye tres botones junto con tres ComboBox.



```
raiz = tk.Tk()
raiz.config(bg="#1e1f22")
raiz.geometry("1250x600")

frame = tkinter.Frame(raiz, bg='#2b2d30', width = 1250, height=70).grid(row=0, columnspan=6)

btnAdd = tk.Button(frame, text="Añadir", command=add)
btnAdd.grid(row=0, column=0)
comboAdd = ttk.Combobox(frame, state="readonly", values=["Cliente", "Pedido", "Producto"], textvariable="Añadir")
comboAdd.grid(row=0, column=1)

btnAdd = tk.Button(frame, text="Actualizar", command=update)
btnAdd.grid(row=0, column=2)
comboUpdate = ttk.Combobox(frame, state="readonly", values=["Cliente", "Pedido", "Producto"], textvariable="Actualizar")
comboUpdate.grid(row=0, column=3)

btnAdd = tk.Button(frame, text="Filtrar", command=filter)
btnAdd.grid(row=0, column=4)
comboFilter = ttk.Combobox(frame, state="readonly", values=["Clientes", "Pedidos", "Productos"], textvariable="filtrar")
comboFilter.grid(row=0, column=5)

style = ttk.Style()
style.theme_use("clam")
style.configure(style: "Treeview",
                background="#333344", # Color de fondo
                foreground="#fff", # Color del texto
                rowheight=40,
                fieldbackground="#333333") # Altura de la fila
style.map(style: "Treeview", background=[('selected', '#214283')])

raiz.mainloop()
```

Cada botón tiene una función que recoge el valor seleccionado del ComboBox adyacente para poder ejecutar el programa correctamente.

# Añadir

Al pulsar el botón “añadir”, se abre la ventana de la opción seleccionada.

```
btnAdd = tk.Button(frame, text="Añadir", command=add)
btnAdd.grid(row=0, column=0)
comboAdd = ttk.Combobox(frame, state="readonly", values=["Cliente", "Pedido", "Producto"], textvariable="Añadir")
comboAdd.grid(row=0, column=1)
```

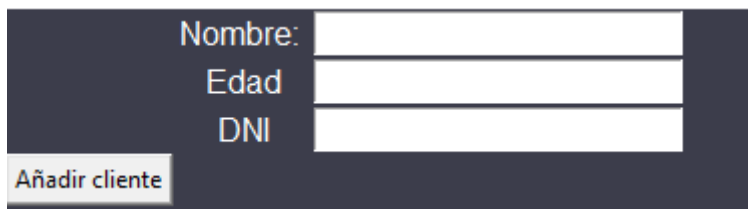
```
def add():
    seleccion = comboAdd.get()
    match seleccion:
        case 'Cliente':
            ClienteAdd.clienteAdd()
        case 'Producto':
            ProductoAdd.productoAdd()
        case 'Pedido':
            PedidoAdd.pedidoAdd()
```

Todas las ventanas de añadir tienen la misma lógica y casi el mismo diseño. Por lo que no explicaré todas. Aun así incluiré las imágenes que sean necesarias.

Las ventanas de añadir cuentan con un formulario que hay que rellenar para poder pulsar el botón. Ese botón recoge todos los datos introducidos en el formulario y llama a la función “add”.

La función add hace un Insert a la base de datos, con la que nos conectamos al principio del código, con los datos recogidos del formulario.

## ClienteAdd



```
def add(nombre, edad, dni):
    if (nombre and edad and dni):
        conexion.execute( sql: "INSERT INTO Clientes (nombre, edad, dni) VALUES (?, ?, ?)",
                           parameters: (nombre, edad, dni))
        conexion.commit()
        print("Hola")
    else:
        print("no")
```

```

raiz = tk.Tk()
raiz.config(bg="#3C3D4B")
raiz.geometry("750x750")

label_nombre = tk.Label(raiz, text="Nombre: ", font=('AdobeDevanagari-Regular'), bg="#3C3D4B", fg="ffff")
label_nombre.grid(row=0,column=2)
entry_nombre = tk.Entry(raiz, text="", font=('AdobeDevanagari-Regular'))
entry_nombre.grid(row=0,column=3)

label_edad = tk.Label(raiz, text="Edad", font=('AdobeDevanagari-Regular'), bg="#3C3D4B", fg="ffff")
label_edad.grid(row=2,column=2)
entry_edad = tk.Entry(raiz, text="", font=('AdobeDevanagari-Regular'))
entry_edad.grid(row=2,column=3)

label_dni = tk.Label(raiz, text="DNI", font=('AdobeDevanagari-Regular'), bg="#3C3D4B", fg="ffff")
label_dni.grid(row=4,column=2)
entry_dni = tk.Entry(raiz, text="", font=('AdobeDevanagari-Regular'))
entry_dni.grid(row=4,column=3)

btnAdd = tk.Button(raiz, text="Añadir cliente", command=lambda:
    add(
        entry_nombre.get(),
        entry_edad.get(),
        entry_dni.get()
    )
)

```

## PedidoAdd

```

def add(id_cliente, producto, cantidad, precio):
    if (id_cliente and producto and cantidad and precio):
        try:
            cantidad = int(cantidad)
            precio = precio*cantidad
            conexion.execute( sql: "INSERT INTO Pedidos (id_pedido, id_cliente, producto, cantidad, precio) VALUES (null, ?, ?, ?, ?)",
                             parameters: (id_cliente, producto, cantidad, precio))
            conexion.commit()
            print("Hola")
        except ValueError:
            messagebox.showinfo(
                title="ERROR",
                message="Has introducido un dato no numerico"
            )
    else:
        print("no")

```

Las siguientes funciones sirven para rellenar los ComboBox y calcular el precio total del pedido.

```
def getProductos():
    cursor = conexion.cursor()
    productos = []
    cursor.execute("SELECT * FROM Productos")
    rs = cursor.fetchall()
    for registro in rs:
        productos.append(registro[1])
    return productos

1 usage
def getClientes():
    cursor = conexion.cursor()
    clientes = []
    cursor.execute("SELECT * FROM Clientes")
    rs = cursor.fetchall()
    for registro in rs:
        clientes.append(registro[0])
    return clientes

1 usage
def getPrecio(producto):
    cursor = conexion.cursor()
    cursor.execute(f"SELECT * FROM Productos WHERE producto = '{producto}'")
    rs = cursor.fetchone()
    return rs[2]
```

```
raiz = tk.Tk()
raiz.config(bg="#3C3D4B")
raiz.geometry("750x750")

label_idCliente = tk.Label(raiz, text="DNI Cliente: ", font=('AdobeDevanagari-Regular'), bg="#3C3D4B", fg="ffff")
label_idCliente.grid(row=0, column=2)
comboCliente = ttk.Combobox(raiz, state="readonly", values=getClientes())
comboCliente.grid(row=0, column=3)

label_idCliente = tk.Label(raiz, text="Producto: ", font=('AdobeDevanagari-Regular'), bg="#3C3D4B", fg="ffff")
label_idCliente.grid(row=1, column=2)
comboProducto = ttk.Combobox(raiz, state="readonly", values=getProductos())
comboProducto.grid(row=1, column=3)

label_cantidad = tk.Label(raiz, text="Cantidad: ", font=('AdobeDevanagari-Regular'), bg="#3C3D4B", fg="ffff")
label_cantidad.grid(row=2, column=2)
entry_cantidad = tk.Entry(raiz, text="", font=('AdobeDevanagari-Regular'))
entry_cantidad.grid(row=2, column=3)

#btnAdd = tk.Button(raiz, text="Añadir producto", command=getProductos())
btnAdd = tk.Button(raiz, text="Añadir producto", command=lambda:
    add(
        comboCliente.get(),
        comboProducto.get(),
        entry_cantidad.get(),
        getPrecio(comboProducto.get())
    )
)
```

# ProductoAdd

Producto:

Precio:

Añadir producto

```
def add(producto, precio):
    if (producto and precio):
        try:
            float(precio)
            conexion.execute(sql: "INSERT INTO Productos (id_producto, producto, precio) VALUES (null, ?, ?)",
                             parameters: (producto, precio))
            conexion.commit()
            print("Hola")
        except ValueError:
            messagebox.showinfo(
                title="ERROR",
                message="Has introducido un dato no numerico"
            )
    else:
        print("no")
```

```
raiz = tk.Tk()
raiz.config(bg="#3C3D4B")
raiz.geometry("750x750")

label_producto = tk.Label(raiz, text="Producto: ", font=('AdobeDevanagari-Regular'), bg="#3C3D4B", fg="#fff")
label_producto.grid(row=0,column=2)
entry_producto = tk.Entry(raiz, text="", font=('AdobeDevanagari-Regular'))
entry_producto.grid(row=0,column=3)

label_precio = tk.Label(raiz, text="Precio: ", font=('AdobeDevanagari-Regular'), bg="#3C3D4B", fg="#fff")
label_precio.grid(row=2,column=2)
entry_precio = tk.Entry(raiz, text="", font=('AdobeDevanagari-Regular'))
entry_precio.grid(row=2,column=3)

btnAdd = tk.Button(raiz, text="Añadir producto", command=lambda:
    add(
        entry_producto.get(),
        entry_precio.get()
    )
)
```



# Actualizar

Al pulsar el botón “actualizar”, se abre la ventana de la opción seleccionada.

```
btnAdd = tk.Button(frame, text="Actualizar", command=update)
btnAdd.grid(row=0, column=2)
comboUpdate = ttk.Combobox(frame, state="readonly", values=["Cliente", "Pedido", "Producto"], textvariable="Actualizar")
comboUpdate.grid(row=0, column=3)
```

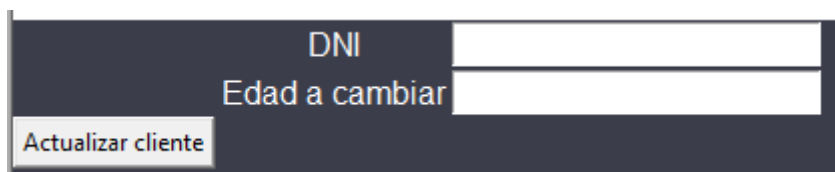
```
def update():
    seleccion = comboUpdate.get()
    match seleccion:
        case 'Cliente':
            ClienteUpdate.clienteUpdate()
        case 'Producto':
            ProductoUpdate.productoUpdate()
        case 'Pedido':
            PedidoUpdate.pedidoUpdate()
```

Todas las ventanas de actualizar tienen la misma lógica y casi el mismo diseño. Por lo que no explicaré todas. Aun así incluiré las imágenes que sean necesarias.

Las ventanas de actualizar cuentan con un formulario que hay que rellenar para poder pulsar el botón. Ese botón recoge todos los datos introducidos en el formulario y llama a la función “update”.

La función update hace un Update en la base de datos, con la que nos conectamos al principio del código, con los datos recogidos del formulario.

## ClienteUpdate



```
def update(dni, edad):
    if (dni and edad):
        conexion.execute(sql="UPDATE Clientes set edad=? where dni=?",
                        parameters=(edad, dni))
        conexion.commit()
        print("Hola")
    else:
        print("no")
```

```

raiz = tkr.Tk()
raiz.config(bg="#3C3D4B")
raiz.geometry("750x750")

label_dni = tkr.Label(raiz, text="DNI", font=('AdobeDevanagari-Regular'), bg="#3C3D4B", fg="#fff")
label_dni.grid(row=2, column=2)
entry_dni = tkr.Entry(raiz, text="", font=('AdobeDevanagari-Regular'))
entry_dni.grid(row=2, column=3)

label_edad = tkr.Label(raiz, text="Edad a cambiar", font=('AdobeDevanagari-Regular'), bg="#3C3D4B", fg="#fff")
label_edad.grid(row=4, column=2)
entry_edad = tkr.Entry(raiz, text="", font=('AdobeDevanagari-Regular'))
entry_edad.grid(row=4, column=3)

btnAdd = tkr.Button(raiz, text="Actualizar cliente", command=lambda:
    update(
        entry_dni.get(),
        entry_edad.get()
    )
)

```

## PedidoUpdate

Id del pedido:	<input type="text"/>
Cantidad a cambiar:	<input type="text"/>
<input type="button" value="Actualizar cliente"/>	

```

def update(id_pedido, cantidad):
    if (id_pedido and cantidad):
        try:
            int(cantidad)
            conexion.execute( sql: "UPDATE Productos set cantidad=? where id_pedido=?",
                               parameters: (cantidad, id_pedido))
            conexion.commit()
            print("Hola")
        except ValueError:
            messagebox.showinfo(
                title="ERROR",
                message="Has introducido un dato no numerico"
            )

    else:
        print("no")

```

```

raiz = tk.Tk()
raiz.config(bg="#3C3D4B")
raiz.geometry("750x750")

label_idPedido = tk.Label(raiz, text="Id del pedido: ", font=('AdobeDevanagari-Regular'), bg="#3C3D4B", fg="fff")
label_idPedido.grid(row=2, column=2)
label_idPedido = tk.Entry(raiz, text="", font=('AdobeDevanagari-Regular'))
label_idPedido.grid(row=2, column=3)

label_cantidad = tk.Label(raiz, text="Cantidad a cambiar: ", font=('AdobeDevanagari-Regular'), bg="#3C3D4B", fg="fff")
label_cantidad.grid(row=4, column=2)
label_cantidad = tk.Entry(raiz, text="", font=('AdobeDevanagari-Regular'))
label_cantidad.grid(row=4, column=3)

btnAdd = tk.Button(raiz, text="Actualizar cliente", command=lambda:
    update(
        label_idPedido.get(),
        label_cantidad.get()
    )
)

```

## ProductoUpdate

Id del producto:	<input type="text"/>
Precio a cambiar:	<input type="text"/>
<input type="button" value="Actualizar cliente"/>	

```

def update(id_producto, precio):
    if (id_producto and precio):
        try:
            float(precio)
            conexion.execute(sql="UPDATE Productos set precio=? where id_producto=?",
                             parameters=(precio, id_producto))
            conexion.commit()
            print("Hola")
        except ValueError:
            messagebox.showinfo(
                title="ERROR",
                message="Has introducido un dato no numerico"
            )

    else:
        print("no")

```

```

raiz = tk.Tk()
raiz.config(bg="#3C3D4B")
raiz.geometry("750x750")

label_idProducto = tk.Label(raiz, text="Id del producto: ", font=('AdobeDevanagari-Regular'), bg="#3C3D4B", fg="fff")
label_idProducto.grid(row=2, column=2)
label_idProducto = tk.Entry(raiz, text="", font=('AdobeDevanagari-Regular'))
label_idProducto.grid(row=2, column=3)

label_precio = tk.Label(raiz, text="Precio a cambiar: ", font=('AdobeDevanagari-Regular'), bg="#3C3D4B", fg="fff")
label_precio.grid(row=4, column=2)
label_precio = tk.Entry(raiz, text="", font=('AdobeDevanagari-Regular'))
label_precio.grid(row=4, column=3)

btnAdd = tk.Button(raiz, text="Actualizar cliente", command=lambda:
    update(
        label_idProducto.get(),
        label_precio.get()
    )
)

```

# Filtrar

A diferencia de los demás botones, al pulsar el botón “Filtrar” no se abre una ventana nueva, sino que se crea un TreeView en la misma ventana Main

```
btnAdd = tk.Button(frame, text="Filtrar", command=filter)
btnAdd.grid(row=0, column=4)
comboFilter = ttk.Combobox(frame, state="readonly", values=["Clientes", "Pedidos", "Productos"], textvariable="filtrar")
comboFilter.grid(row=0, column=5)
```

```
def filter():
    seleccion = comboFilter.get()
    match seleccion:
        case 'Clientes':
            clienteFilter()
        case 'Productos':
            productoFilter()
        case 'Pedidos':
            pedidoFilter()
```

Todas las tablas TreeView tienen la misma lógica y casi el mismo diseño. Por lo que no explicaré todas. Aun así incluiré las imágenes que sean necesarias.

Se empieza creando el TreeView con ttk.

Se sigue creando las cabeceras correspondientes a cada tabla.

Se eliminan las filas “hijas” para actualizarlas.

Se guardan los registros de la base de datos en una variable y se insertan en la tabla.

Finalmente, se crean tres botones: exportar, que exporta la tabla actual a un “.xlsx”; gráfico, que crea una ventana con un gráfico de barras de la tabla actual y eliminar, que elimina el registro seleccionado de la tabla en la base de datos.

# ClienteFilter

```
def clienteFilter():
    lista_registros = ttk.Treeview(raiz, columns=("dni", "nombre", "edad"), show="headings")

    lista_registros.heading("dni", text="DNI cliente", command=lambda: ordenar_columna(lista_registros, col: "dni", reverse: False))
    lista_registros.heading("nombre", text="Nombre", command=lambda: ordenar_columna(lista_registros, col: "nombre", reverse: False))
    lista_registros.heading("edad", text="Edad", command=lambda: ordenar_columna(lista_registros, col: "edad", reverse: False))

    lista_registros.grid(row=2, column=0, columnspan=5, padx=10, pady=10, sticky="nsew")
    lista_registros.delete(*lista_registros.get_children())

    registros = leer_registros("Clientes")
    for registro in registros:
        lista_registros.insert(parent="", index="end", values=registro)

    btnExport = tkr.Button(frame, text="Exportar", command=lambda: exportar(tabla: "Clientes", registros))
    btnExport.grid(row=3, column=1)

    btnGrafico = tkr.Button(frame, text="Grafico", command=lambda: grafico(tabla: "Clientes"))
    btnGrafico.grid(row=3, column=2)

    btnEliminar = tkr.Button(frame, text="Eliminar", command=lambda: eliminar_registro(tabla: "Clientes", lista_registros))
    btnEliminar.grid(row=3, column=3)
```

```
def leer_registros(tabla):
    with sqlite3.connect("databaseSupermercado.db") as conn:
        cursor = conn.cursor()
        cursor.execute(f"SELECT * FROM {tabla}")
        registros = cursor.fetchall()
    return registros
```

Añadir

Cliente

Actualizar

Producto

Filtrar

Clientes

DNI cliente	Nombre	Edad
123456789	David	Paco
dfg	d	34.0
dfgghijn	David	2323.0
dfgghijnsth	David	2323.0

Exportar

Grafico

Eliminar

# ProductoFilter

```
def productoFilter():
    lista_registros = ttk.Treeview(raiz, columns=("id_producto", "producto", "precio"), show="headings")

    lista_registros.heading("id_producto", text="ID Producto", command=lambda: ordenar_columna(lista_registros, col: "id_producto", reverse: False))
    lista_registros.heading("producto", text="Producto", command=lambda: ordenar_columna(lista_registros, col: "producto", reverse: False))
    lista_registros.heading("precio", text="Precio", command=lambda: ordenar_columna(lista_registros, col: "precio", reverse: False))

    lista_registros.grid(row=2, column=0, columnspan=5, padx=10, pady=10, sticky="nsew")
    lista_registros.delete(*lista_registros.get_children())

    registros = leer_registros("Productos")
    for registro in registros:
        lista_registros.insert(parent="", index="end", values=registro)

    btnExport = tk.Button(frame, text="Exportar", command=lambda: exportar(tabla: "Productos", registros))
    btnExport.grid(row=3, column=1)

    btnGrafico = tk.Button(frame, text="Grafico", command=lambda: grafico("Productos"))
    btnGrafico.grid(row=3, column=2)

    btnEliminar = tk.Button(frame, text="Eliminar", command=lambda: eliminar_registro(tabla: "Productos", lista_registros))
    btnEliminar.grid(row=3, column=3)
```

```
def leer_registros(tabla):
    with sqlite3.connect("databaseSupermercado.db") as conn:
        cursor = conn.cursor()
        cursor.execute(f"SELECT * FROM {tabla}")
        registros = cursor.fetchall()
    return registros
```

Añadir

Ciente

Actualizar

Producto

Filtrar

Productos

ID Producto	Producto	Precio
1	Manzanas	89.6

Exportar

Grafico

Eliminar

# PedidoFilter

```
def pedidoFilter():
    lista_registros = ttk.Treeview(raiz, columns=("id_pedido", "dni", "producto", "cantidad", "precio"), show="headings")

    lista_registros.heading("id_pedido", text="ID Pedido", command=lambda: ordenar_columna(lista_registros, col="id_pedido", reverse=False))
    lista_registros.heading("dni", text="DNI", command=lambda: ordenar_columna(lista_registros, col="dni", reverse=False))
    lista_registros.heading("producto", text="Producto", command=lambda: ordenar_columna(lista_registros, col="producto", reverse=False))
    lista_registros.heading("cantidad", text="Cantidad", command=lambda: ordenar_columna(lista_registros, col="cantidad", reverse=False))
    lista_registros.heading("precio", text="Precio", command=lambda: ordenar_columna(lista_registros, col="precio", reverse=False))

    lista_registros.grid(row=2, column=0, colspan=5, padx=10, pady=10, sticky="nsew")
    lista_registros.delete(*lista_registros.get_children())

    registros = leer_registros("Pedidos")
    for registro in registros:
        lista_registros.insert(parent="", index="end", values=registro)

    btnExport = tk.Button(frame, text="Exportar", command=lambda: exportar(tabla="Pedidos", registros))
    btnExport.grid(row=3, column=1)

    btnGrafico = tk.Button(frame, text="Grafico", command=lambda: grafico("Pedidos"))
    btnGrafico.grid(row=3, column=2)

    btnEliminar = tk.Button(frame, text="Eliminar", command=lambda: eliminar_registro(tabla="Pedidos", lista_registros))
    btnEliminar.grid(row=3, column=3)
```

```
def leer_registros(tabla):
    with sqlite3.connect("databaseSupermercado.db") as conn:
        cursor = conn.cursor()
        cursor.execute(f"SELECT * FROM {tabla}")
        registros = cursor.fetchall()
    return registros
```

ID Pedido	DNI	Producto	Cantidad	Precio
1	1	Manzanas	3	56.849999999999994

## Ordenar tabla

Puede pasar desapercibido, pero al pulsar sobre cada cabecera, la tabla se ordena según la cabecera seleccionada.

```
def ordenar_columna(tree, col, reverse):
    data = [(tree.set(child, col), child) for child in tree.get_children('')]
    data.sort(reverse=reverse)
    for i, item in enumerate(data):
        tree.move(item[1], '', i)
    tree.heading(col, command=lambda: ordenar_columna(tree, col, not reverse))
```

# Eliminar

Al seleccionar un registro de la tabla se podrá eliminar ese registro pulsando el botón.

Al pulsar el botón, se llama a la función “eliminar\_registro” junto con el nombre de la tabla y el TreeView actuales.

```
btnEliminar = tk.Button(frame, text="Eliminar", command=lambda:eliminar_registro(tabla:"Clientes", lista_registros))
btnEliminar.grid(row=3, column=3)
```

La función “eliminar\_registro” busca en un Match/Case el nombre de la tabla desde la que se ha pulsado el botón y se elimina el registro que se había seleccionado. Finalmente, se vuelve a llamar al filtro que estaba activo para actualizar la tabla.

```
def eliminar_registro(tabla, lista_registros):
    seleccion = lista_registros.selection()
    if seleccion:
        id_seleccionado = lista_registros.item(seleccion, "values")[0]

        match tabla:
            case "Clientes":
                with sqlite3.connect("databaseSupermercado.db") as conn:
                    cursor = conn.cursor()

                    cursor.execute(_sql: "DELETE FROM Clientes WHERE dni=?", _parameters: (id_seleccionado,))
                    conn.commit()
                    clienteFilter()

            case "Productos":
                with sqlite3.connect("databaseSupermercado.db") as conn:
                    cursor = conn.cursor()

                    cursor.execute(_sql: "DELETE FROM Productos WHERE id_producto=?", _parameters: (id_seleccionado,))
                    conn.commit()
                    productoFilter()

            case "Pedidos":
                with sqlite3.connect("databaseSupermercado.db") as conn:
                    cursor = conn.cursor()

                    cursor.execute(_sql: "DELETE FROM Pedidos WHERE id_pedido=?", _parameters: (id_seleccionado,))
                    conn.commit()
                    pedidoFilter()
```



# Exportar

Al pulsar sobre el botón exportar, se llamará a la función exportar junto con el nombre de la tabla actual y la variable que incluye todos los registros de la tabla.

```
btnExport = tk.Button(frame, text="Exportar", command=lambda:exportar(tabla: "Clientes", registros))
btnExport.grid(row=3, column=1)
```

La función “exportar” crea un Workbook y una hoja.

Según la tabla que estaba activa cuando hemos pulsado el botón, se guarda un array con los nombres de las cabeceras del Excel en una variable.

```
match tabla:
    case "Clientes":
        header = ["Dni", "Nombre", "Edad"]
    case "Productos":
        header = ["ID Producto", "Producto", "Precio"]
    case "Pedidos":
        header = ["ID Pedido", "ID Cliente", "Producto", "Cantidad", "Precio"]
```

Se establecen las celdas del Excel en base a las cabeceras y a los registros.

```
for col_num, header in enumerate(header, 1):
    hoja.cell(row=1, column=col_num, value=header)

for row_num, fila in enumerate(registros, 2):
    for col_num, valor in enumerate(fila, 1):
        hoja.cell(row=row_num, column=col_num, value=valor).alignment = Alignment(horizontal="center")
```

Finalmente se le pide al usuario que guarde el Excel en la ruta deseada.

```
archivo = filedialog.asksaveasfilename(defaultextension=".xlsx", filetypes=[("Archivos de Excel", "*.xlsx")])
libro.save(archivo)
```

# Gráfico

Al pulsar el botón “Gráfico” se llama a la función “grafico” junto con el nombre de la tabla actual y se abre la ventana que lleva el nombre de esa tabla.

```
btnGrafico = tk.Button(frame, text="Gráfico", command=lambda:grafico("Clientes"))
btnGrafico.grid(row=3, column=2)
```

```
def grafico(tabla):
    match tabla:
        case "Clientes":
            GraficoClientes.hacerGrafico()
        case "Productos":
            GraficoProductos.hacerGrafico()
        case "Pedidos":
            GraficoPedidos.hacerGrafico()
```

Todas las ventanas de grafico tienen la misma lógica y casi el mismo diseño. Por lo que no explicaré todas. Aun así incluiré las imágenes que sean necesarias.

Lo primero que se hace en el código, después de crear la ventana, es una consulta que recoge algunos parámetros de la tabla de la ventana.

Después se guardan los registros en los ejes “x” e “y”.

Se le pone nombre a los ejes para que el gráfico sea más comprensible.

Por último se muestra el gráfico en la ventana.

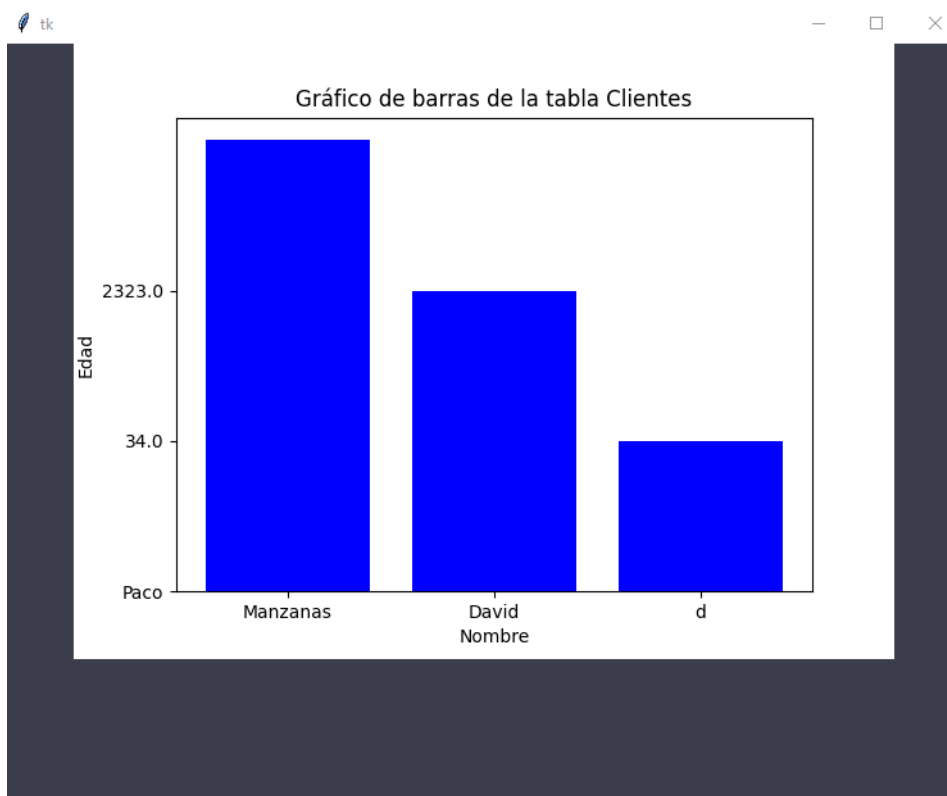
# GraficoClientes

```
def hacerGrafico():
    raiz = tk.Tk()
    raiz.config(bg="#3C3D4B")
    raiz.geometry("750x750")

    with sqlite3.connect("databaseSupermercado.db") as conn:
        cursor = conn.cursor()
        cursor.execute(f"SELECT nombre, edad FROM Clientes")
        registros = cursor.fetchall()

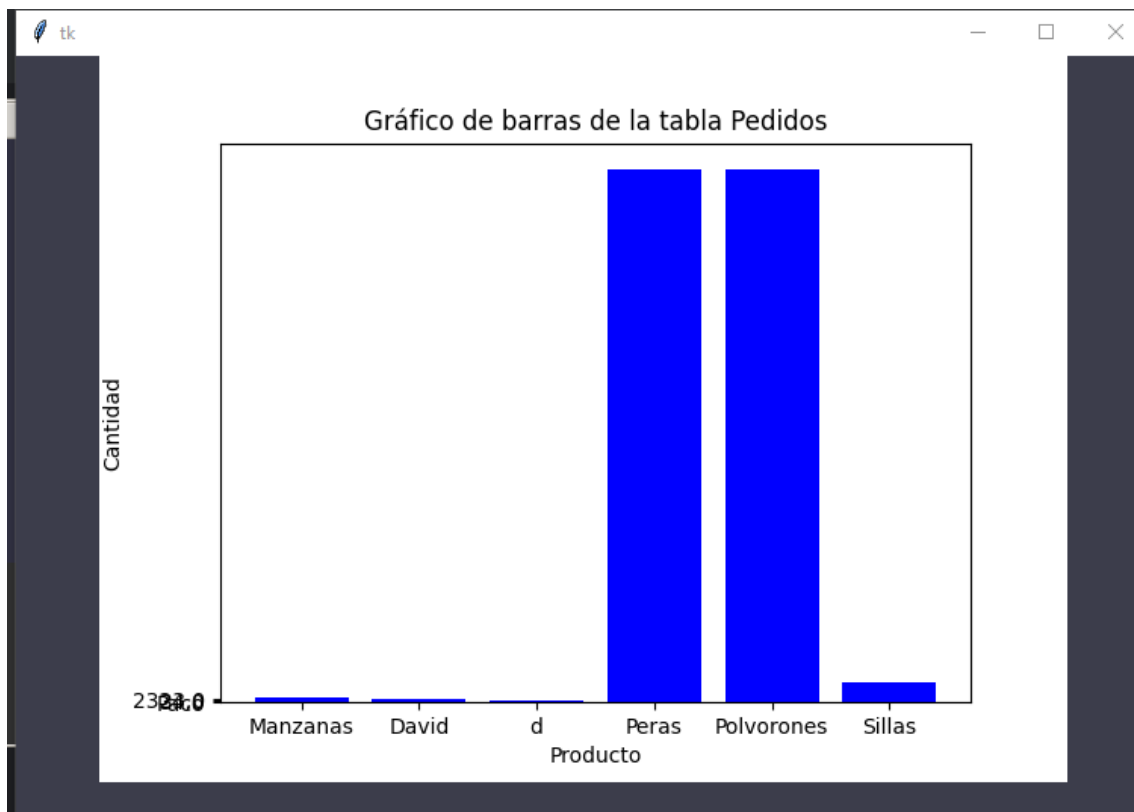
    x_data, y_data = zip(*registros)
    y_data = [str(y) for y in y_data]
    plt.bar(x_data, y_data, color='blue')
    plt.title(f'Gráfico de barras de la tabla Clientes')
    plt.xlabel('Nombre')
    plt.ylabel('Edad')

    canvas = FigureCanvasTkAgg(plt.gcf(), master=raiz)
    canvas.draw()
    canvas.get_tk_widget().pack()
```

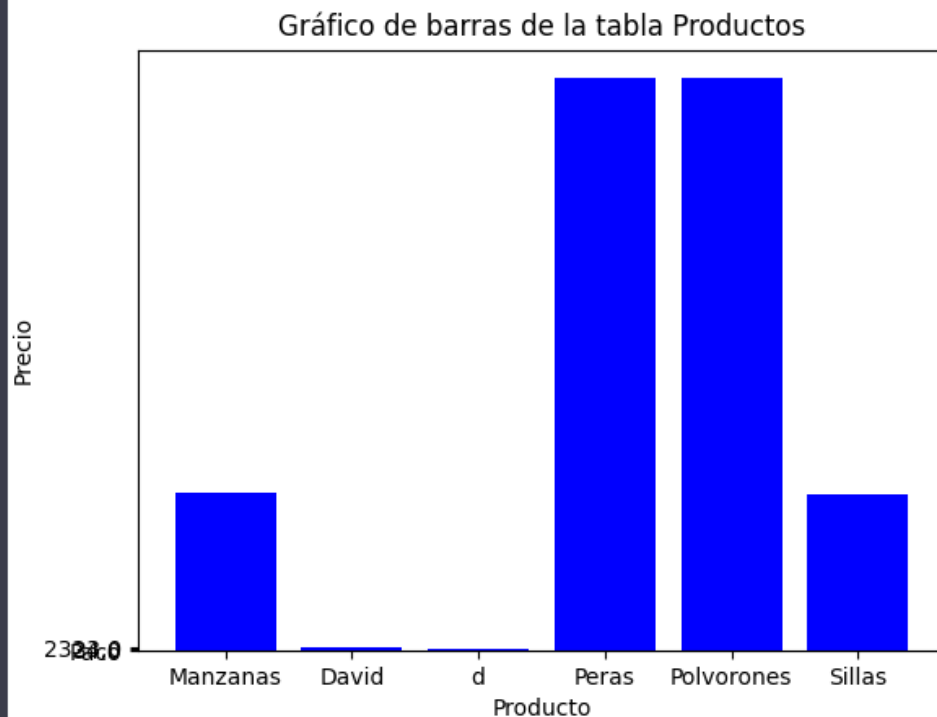


# GraficoPedidos

```
def hacerGrafico():  
    raiz = tk.Tk()  
    raiz.config(bg="#3C3D4B")  
    raiz.geometry("750x750")  
  
    with sqlite3.connect("databaseSupermercado.db") as conn:  
        cursor = conn.cursor()  
        cursor.execute(f"SELECT producto, cantidad FROM Pedidos")  
        registros = cursor.fetchall()  
  
    x_data, y_data = zip(*registros)  
    plt.bar(x_data, y_data, color='blue')  
    plt.title(f'Gráfico de barras de la tabla Pedidos')  
    plt.xlabel('Producto')  
    plt.ylabel('Cantidad')  
  
    canvas = FigureCanvasTkAgg(plt.gcf(), master=raiz)  
    canvas.draw()  
    canvas.get_tk_widget().pack()
```



# GraficoProductos



```
def hacerGrafico():  
    raiz = tk.Tk()  
    raiz.config(bg="#3C3D4B")  
    raiz.geometry("750x750")  
  
    with sqlite3.connect("databaseSupermercado.db") as conn:  
        cursor = conn.cursor()  
        cursor.execute(f"SELECT producto, precio FROM Productos")  
        registros = cursor.fetchall()  
  
    x_data, y_data = zip(*registros)  
    plt.bar(x_data, y_data, color='blue')  
    plt.title(f'Gráfico de barras de la tabla Productos')  
    plt.xlabel('Producto')  
    plt.ylabel('Precio')  
  
    canvas = FigureCanvasTkAgg(plt.gcf(), master=raiz)  
    canvas.draw()  
    canvas.get_tk_widget().pack()
```