

ENLACE A DATOS - CRUD

La palabra CRUD es un acrónimo que hace referencia a:

- **Create** -> Crear. INSERT
- **Read** -> Leer. SELECT
- **Update** -> Actualizar. UPDATE
- **Delete** -> Borrar. DELETE

Primero vamos a realizar un par de cambios necesarios en los códigos anteriores.

- Función ***selPago()***.

Los cambios son necesarios para controlar que checks están marcados y que no. Además, para evitar duplicidades si el usuario se dedicase a marcar y desmarcar repetidamente. Devuelve una lista que contiene las formas de pago, que además, se han agrupado en un ButtonGroup, al igual que se hizo con los radiobutton.

```
def selPago():
    try:
        var.pay = []
        for i, data in enumerate(var.ui.grpbtnPay.buttons()):
            #agrupamos en QtDesigner los checkbox en un ButtonGroup
            if data.isChecked() and i == 0:
                var.pay.append('Efectivo')
            if data.isChecked() and i == 1:
                var.pay.append('Tarjeta')
            if data.isChecked() and i == 2:
                var.pay.append('Transferencia')
        #var.pay = set(var.pay)
        print(var.pay)
        return var.pay
    except Exception as error:
        print('Error: %s' % str(error))
```

- En **showClientes** recogemos esta función, **selPago()** que realiza el testeo de los checkbox marcados, así como comprobar que los campos principales, nombre, apellidos y dni no están vacíos.

```

        k += 1
        newcli.append(vpro)
        var.pay2 = Clientes.selPago()
        newcli.append(var.sex)
        newcli.append(var.pay2)
        if client:
            #comprobamos que no esté vacío lo principal
            #aquí empieza como trabajar con la TableWidget
            row = 0
            column = 0
            var.ui.tableCli.insertRow(row)
            for registro in clitable:
                cell = QTableWidgetItem(registro)
                var.ui.tableCli.setItem(row, column, cell)
                column += 1

            conexion.Conexion.cargarCli(newcli)
        else:
            print('Faltan Datos')
            Clientes.limpiarCli(client, var.rbtsex, var.chk pago)
    except Exception as error:
        print('Error cargar fecha: %s ' % str(error))

```

Operaciones con la Base de Datos.

Insertar

```

def cargarCli(cliente):
    query = QSqlQuery()
    query.prepare('insert into clientes (dni, apellidos, nombre, fechalta, direccion, provincia, sexo, formaspago)')
    query.bindValue(':dni', str(cliente[0]))
    query.bindValue(':apellidos', str(cliente[1]))
    query.bindValue(':nombre', str(cliente[2]))
    query.bindValue(':fechalta', str(cliente[3]))
    query.bindValue(':direccion', str(cliente[4]))
    query.bindValue(':provincia', str(cliente[5]))
    query.bindValue(':sexo', str(cliente[6]))
    # pagos = ' '.join(cliente[7]) si quisiésemos un texto, pero nos viene mejor meterlo como una lista
    query.bindValue(':formaspago', str(cliente[7]))
    # print(pagos)
    if query.exec():
        print("Inserción Correcta")
    else:
        print("Error: ", query.lastError().text())

```

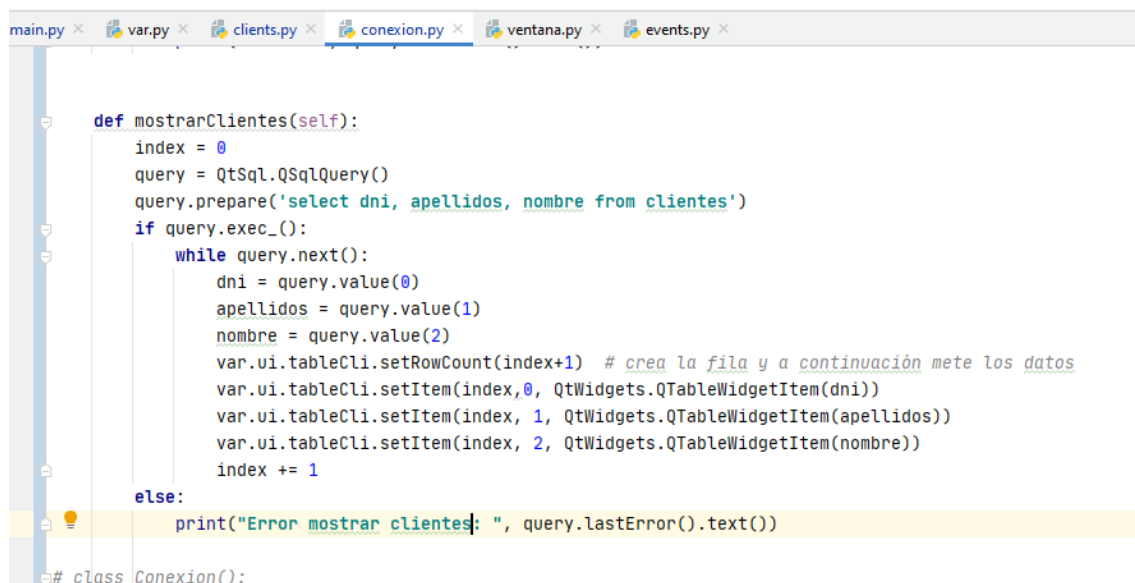
No olvidarse de llamar a esta función desde **showClientes()**

```
Clientes LimpianCli(client, var, phtcox, var, chknago)
conexion.Conexion.cargarCli(newcli)
```

Convendría ahora hacer dos cambios. El primer cambiar el nombre del botón *Aceptar* por el de **Alta Cliente** o simplemente **Alta**. El segundo cambio sería el nombre de la función *showClientes* por **altaClientes**, ya que es más intuitivo. Otra opción sería mantener ambas funciones de forma independiente.

SELECT

En primer lugar realizamos una búsqueda general cuyo objetivo será para que cuando se inicie el programa se carguen los datos de todos los clientes en un listado. Este módulo lo llamaremos desde el principal. Su nombre será **mostrarClientes()**. En el fichero **conexion.py** generamos el módulo **mostrarClientes()**



```
main.py x var.py x clients.py x conexion.py x ventana.py x events.py x

def mostrarClientes(self):
    index = 0
    query = QSqlQuery()
    query.prepare('select dni, apellidos, nombre from clientes')
    if query.exec_():
        while query.next():
            dni = query.value(0)
            apellidos = query.value(1)
            nombre = query.value(2)
            var.ui.tableCli.setRowCount(index+1) # crea la fila y a continuación mete los datos
            var.ui.tableCli.setItem(index,0, QTableWidgetItem(dni))
            var.ui.tableCli.setItem(index, 1, QTableWidgetItem(apellidos))
            var.ui.tableCli.setItem(index, 2, QTableWidgetItem(nombre))
            index += 1
    else:
        print("Error mostrar clientes: ", query.lastError().text())

# class Conexion():
```

Que llamamos desde main.py para que se ejecute al lanzar el programa.

```
main.py x var.py x clients.py x conexion.py x ventana.py x events.py x
52         i.toggleled.connect(clients.Cientes.selSexo)
53     for i in var.chkpago:
54         i.stateChanged.connect(clients.Cientes.selPago)
55     var.ui.cmbProv.activated[str].connect(clients.Cientes.selProv)
56     var.ui.tableCli.clicked.connect(clients.Cientes.cargarCli)
57     var.ui.tableCli.setSelectionBehavior(QtWidgets.QTableWidget.SelectRows)
58     ...
59     Llamada a módulos iniciales
60     ...
61     events.Eventos.cargarProv()
62
63     ...
64     módulos conexion base datos
65     ...
66     conexion.Conexion.db_connect(var.filebd)
67     #conexion.Conexion()
68     conexion.Conexion.mostrarClientes(self)
69
70     def closeEvent(self, event):
71         if event:
72             events.Eventos.Salir(event)
```

Pero también tendremos que **lanzarlo cada vez que insertemos un cliente para que se actualice la tabla** con todos los valores de la tabla clientes de la base de datos.

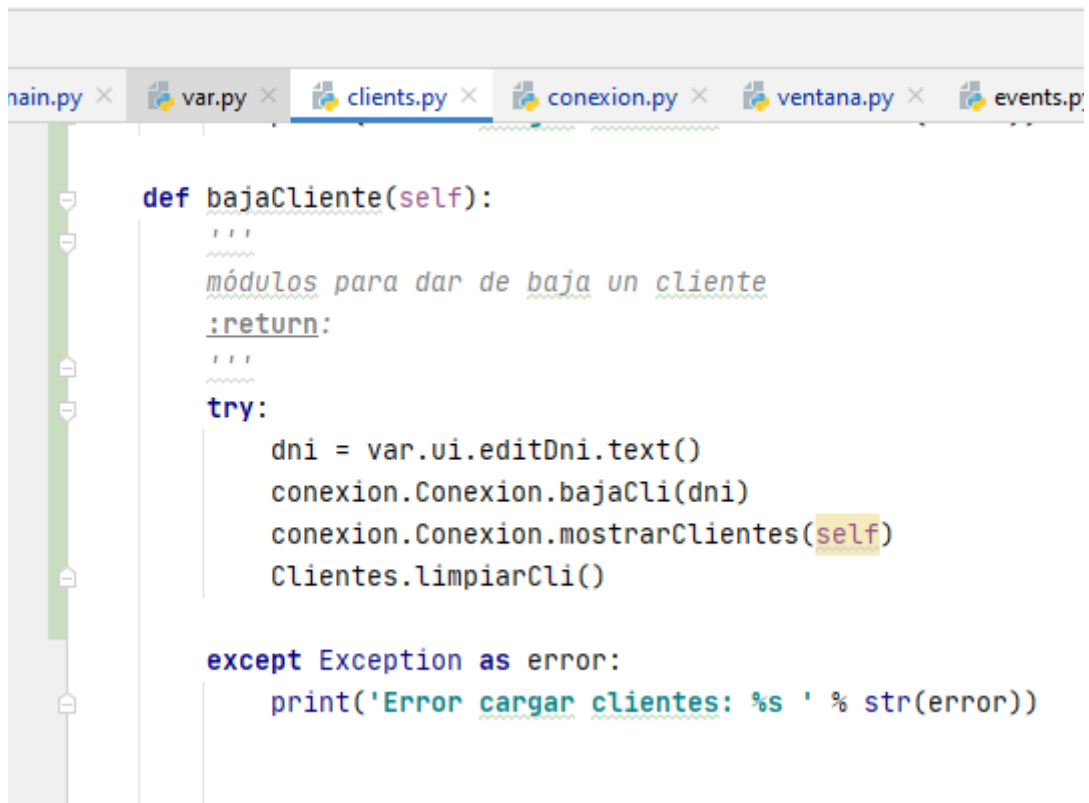
```
main.py x var.py x clients.py x conexion.py x ventana.py x events.py x
        print('Conexión Establecida')
        return True

def cargarCli(cliente):
    query = QSql.QSqlQuery()
    query.prepare('insert into clientes (dni, apellidos, nombre, fe
                    'VALUES (:dni, :apellidos, :nombre, :fechalta, :di
    query.bindValue(':dni', str(cliente[0]))
    query.bindValue(':apellidos', str(cliente[1]))
    query.bindValue(':nombre', str(cliente[2]))
    query.bindValue(':fechalta', str(cliente[3]))
    query.bindValue(':direccion', str(cliente[4]))
    query.bindValue(':provincia', str(cliente[5]))
    query.bindValue(':sexo', str(cliente[6]))
    # pagos = ' '.join(cliente[7]) si quiesesemos un texto, pero no
    query.bindValue(':formasPago', str(cliente[7]))
    # print(pagos)
    if query.exec_():
        print("Inserción Correcta")
        Conexion.mostrarClientes(self)
    else:
        print("Error: ", query.lastError().text())
```

ELIMINAR

El código para un registro de la base de datos es muy sencillo. Tendremos que tener un elemento que o bien es la clave primaria, en nuestro caso el código, o bien podemos utilizar en este caso de personas el DNI ya que no deja de ser único para cada registro. El código sería:

En el módulo **clientes.py** estaría la función **bajaCliente** que llama a **bajaCli** del módulo **conexion.py**



```
main.py x  var.py x  clients.py x  conexion.py x  ventana.py x  events.p
def bajaCliente(self):
    """
    módulos para dar de baja un cliente
    :return:
    """
    try:
        dni = var.ui.editDni.text()
        conexion.Conexion.bajaCli(dni)
        conexion.Conexion.mostrarClientes(self)
        Clientes.limpiarCli()

    except Exception as error:
        print('Error cargar clientes: %s ' % str(error))
```

```
var.py × clients.py × conexion.py × ventana.py × events.py ×

    print("Error mostrar clientes: ", query.lastError().text())

def bajaCli(dni):
    """
    modulo para eliminar cliente. se llama desde fichero clientes.py
    :return None
    """
    query = QSqlQuery()
    query.prepare('delete from clientes where dni = :dni')
    query.bindValue(':dni', dni)
    if query.exec_():
        print('Baja cliente')
        var.ui.lblstatus.setText('Cliente con dni ' + dni + ' dado de baja')
    else:
        print("Error mostrar clientes: ", query.lastError().text())
```

No olvidarse de llamar a **limpiarCli** para restaurar el formulario y recargar con **mostrarClientes** la lista con la actualización hecha. Todo ello desde **bajaCliente** del fichero **clientes.py**

Tampoco debemos olvidarnos de conectar el botón correspondiente del **main.py** con **bajaCliente**.

MODIFICAR

Podemos modificar todo menos la clave primaria o código del cliente. La forma más cómoda es mostrar los datos del cliente, modificar todo o todos aquellos datos que deseamos y cargarlos todos juntos, aunque algunos no se hayan cambiado.

Los códigos serían, en **clientes.py**:

```
.py x clientes.py x var.py x conexion.py x ventana.py x events.py x
afor x Aa W * 0 results ↑ ↓ ↺ ↻ ↵ ↶ ↷ ↸ ↹
def modifCliente(self):
    """
    módulos para dar de modificar datos de un cliente
    :return:
    """
    try:
        newdata = []
        client = [var.ui.editDni, var.ui.editApel, var.ui.editNome, var.ui.editClialta, var.ui.editDir]
        for i in client:
            newdata.append(i.text()) # cargamos los valores que hay en los editline
        newdata.append(var.ui.cmbProv.currentText())
        newdata.append(var.sex)
        var.pay = Clientes.selPago()
        print(var.pay)
        newdata.append(var.pay)
        cod = var.ui.lblCodcli.text()
        conexion.Conexion.modifCli(cod, newdata)
        conexion.Conexion.mostrarClientes(self)

    except Exception as error:
        print('Error cargar clientes: %s ' % str(error))
```

Y el correspondiente a **modifCli** de **conexion.py**.

```
py x clientes.py x var.py x conexion.py x ventana.py x events.py x
def modifCli(codigo, newdata):
    """
    modulo para modificar cliente. se llama desde fichero clientes.py
    :return None
    """
    query = QSql.QSqlQuery()
    codigo = int(codigo)
    query.prepare('update clientes set dni=:dni, apellidos=:apellidos, nombre=:nombre, fechalta=:fechalta, '
        'direccion=:direccion, provincia=:provincia, sexo=:sexo, formaspago=:formaspago where codigo=:codigo')
    query.bindValue(':codigo', int(codigo))
    query.bindValue(':dni', str(newdata[0]))
    query.bindValue(':apellidos', str(newdata[1]))
    query.bindValue(':nombre', str(newdata[2]))
    query.bindValue(':fechalta', str(newdata[3]))
    query.bindValue(':direccion', str(newdata[4]))
    query.bindValue(':provincia', str(newdata[5]))
    query.bindValue(':sexo', str(newdata[6]))
    query.bindValue(':formaspago', str(newdata[7]))
    if query.exec_():
        print('Cliente modificado')
        var.ui.lblStatus.setText('Cliente con dni ' + str(newdata[0]) + ' modificado')
    else:
        print("Error modificar cliente: ", query.lastError().text())
```