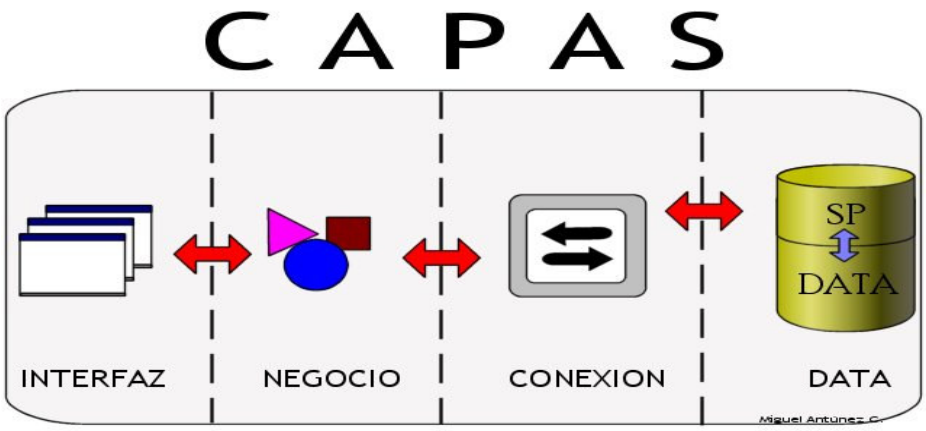


En este diseño de capas se puede observar claramente que **no existe una relación directa entre la capa interfaz y la capa de datos**, para que dicha relación exista tiene que transitar a través de una capa de negocios y seguidamente de una capa conexión, esta última se comunica mediante los SP (Stored Procedures) a la data.

Ejemplos de código que van en diferentes capas

La capa interfaz está basada en clases visuales de VFP, la de negocios en clase no visuales de VFP y para esta capa de tipo "custom", la de conexión también utilizará una clase no visual VFP de tipo "session" y la de data será un SqlServer que puede ser 2000 o 2005. Cabe resaltar que cualquiera de las capas puede ser reemplazada por el tipo que uds. crean conveniente.

Para el ejemplo tenemos un objeto de negocio llamado "miobjeto" el cual tiene los siguientes métodos: selecciona, inserta, edita, borra, revisa y las propiedades _alias, campo1, campo2, campo3. Con lo cual tendríamos:

- Interface

  En nuestro formulario tenemos un método **refrescagrid()**, con un código similar:

  ```
  Thisform.grdata1.recordsource=""

  Thisform.miobjeto.seleccciona()

  Thisform.grdata1.recordsource=thisform.miobjeto._alias
  ```

- Negocio

  Este objeto tiene el método **selecciona**, cuyo código sería:

  Cone.execsp("Mi_Stored_Procedure","? param1,?param2,

  ?param3", "micursor")

  This._alias="micursor"

- Conexión

  Este objeto tiene un método **execsp**, su código tendría que ser:

  =sqlexec(ncon,"execute stored procedure+parámetros","_cursor")

- Data

  Esta capa constará de dos partes (SP, Tablas):

  - **SP**

    Create stored procedure Mi_Stored_Procedure

    @param1 as int

    @param2 as int

    @param3 as int

    As

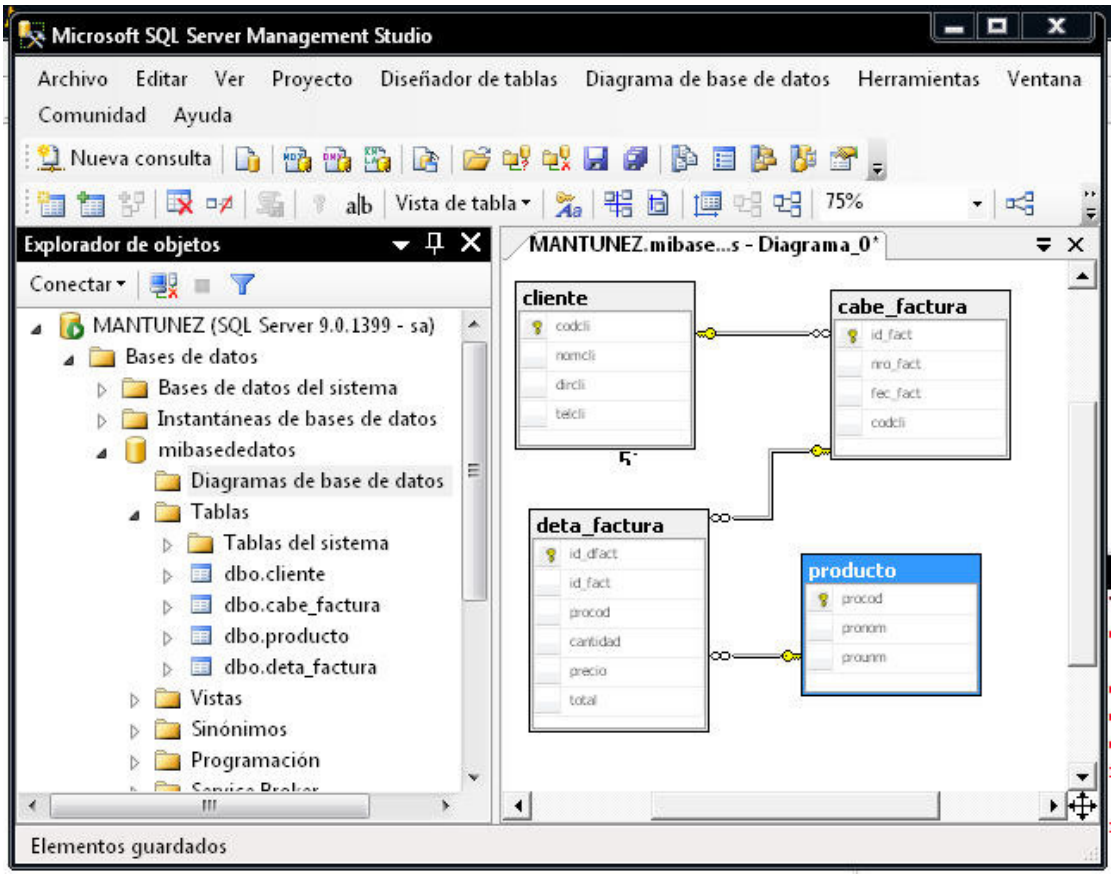
    Select campo1, campo2, campo3, campo4 from mitabla where campox=@param1 and campoy=@param2 and campoz=@param3

  - **Tabla**

    La tabla se llama **mitabla,** con sus campos: campo1, campo2, campo3, campo4, campox, campoy, campoz

CREANDO LA CAPA DE DATOS

DISEÑO DE TABLAS



Crearemos la siguiente base de datos “mibasededatos”. Y las tablas serán las siguientes:

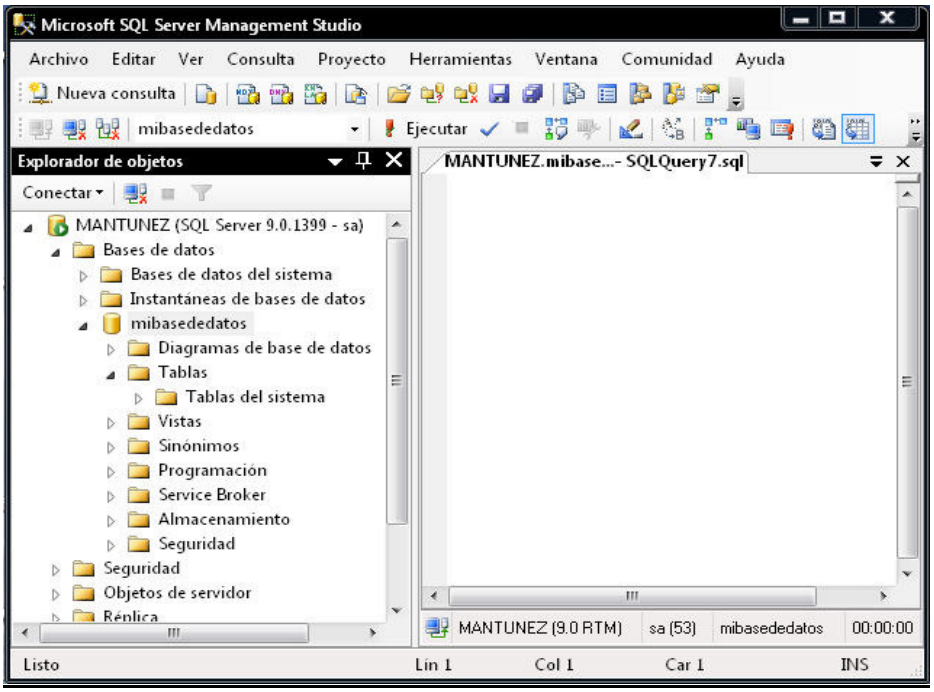
cliente		
codcli	int	pk
nomcli	varchar(50)	
dircli	varchar(60)	
telcli	char(10)	

cabe_factura		
id_fact	int	pk
nro_fact	char(10)	
fec_fact	datetime	
codcli	int	fk

producto		
procod	int	pk
pronom	varchar(50)	
prounm	char(10)	

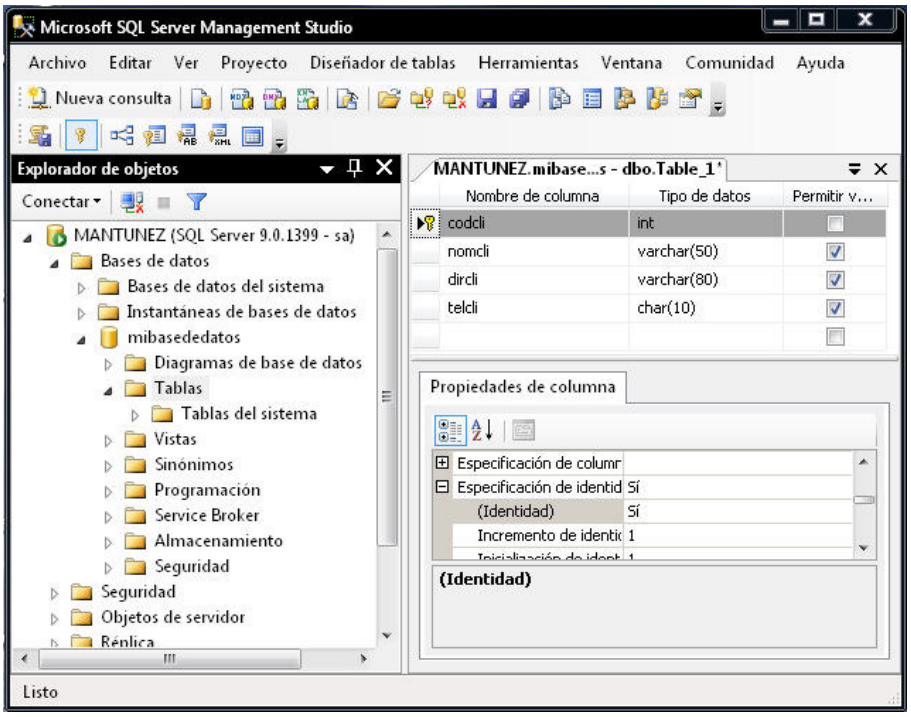
deta_factura		
id_dfact	int	pk
id_fact	int	fk
procod	int	fk
cantidad	numeric(12,3)	
precio	numeric(12,2)	
total	calculado	

ENTORNO SQL SERVER 2005



Este será el entorno de trabajo de nuestra capa de datos:

DEFINIENDO INDICE IDENTIDAD



Las ventajas de utilizar este tipo de campo como “Primary Key”, tiene muchas ventajas tanto de librarnos de pensar, ¿como hago mi programa de correlativos?, y además ayuda al performance de la base de datos tanto al momento de grabar como al momento de hacer las relaciones y búsquedas ya que el índice “Primary Key” también es un índice tipo cluster.

En el grafico se ve como se ve la creación de la tabla y la definición del campo identidad y el incremento en este caso de 1 en 1 y empieza en 1 el correlativo.

En lo personal poner un número pequeño como en este caso a los clientes, facilita el uso para los usuarios, es más práctico recordar números pequeños que codificaciones con pseudocodigos o procedimientos complejos.

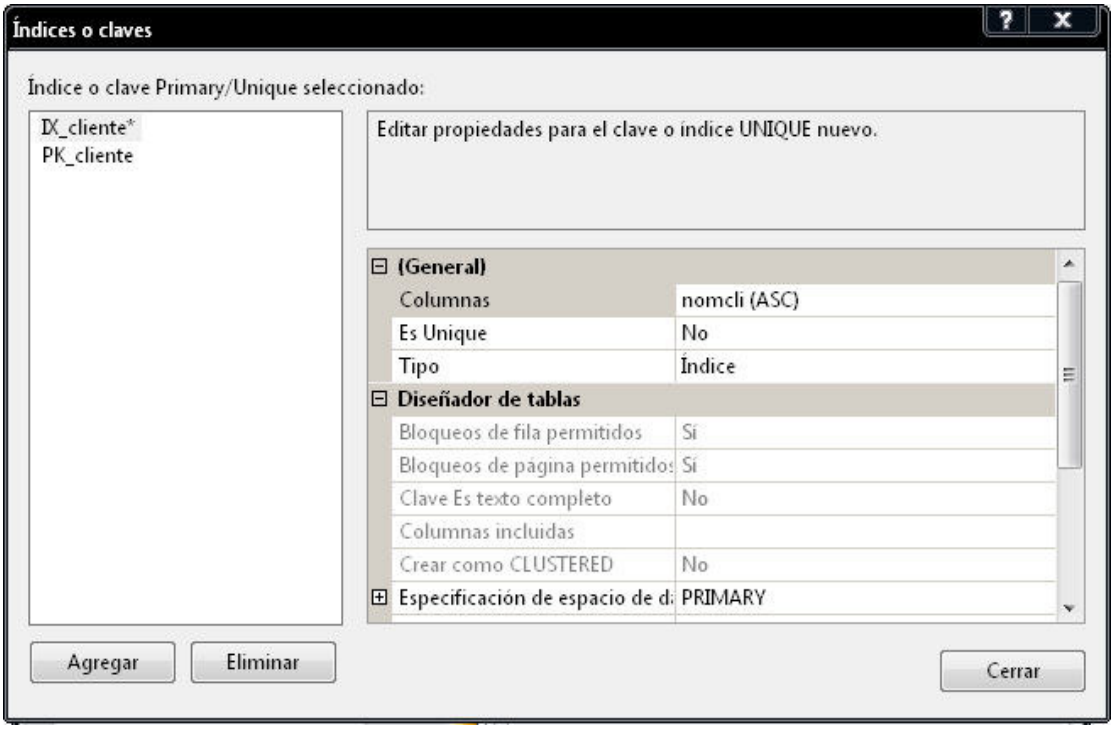
Algunos comandos de utilidad al manejar este tipo de campos:

```
DBCC CHECKIDENT
(
'table_name'
    [ , {
    NORESEED | { RESEED [ , new_reseed_value ] }
        }
    ]
)
[ WITH NO_INFOMSGS ]

Ejemplo:
 DBCC CHECKIDENT ('cliente', RESEED, 0).
```

```
Reseteará el campo identidad a 0.

INSERT INTO clientes
values
('Juan Perez','mi direccion', '88377666')

SELECT @@IDENTITY
```

Dicho "Select" devolverá el número de identidad asignado.

DEFINIENDO ÍNDICES DE BÚSQUEDA



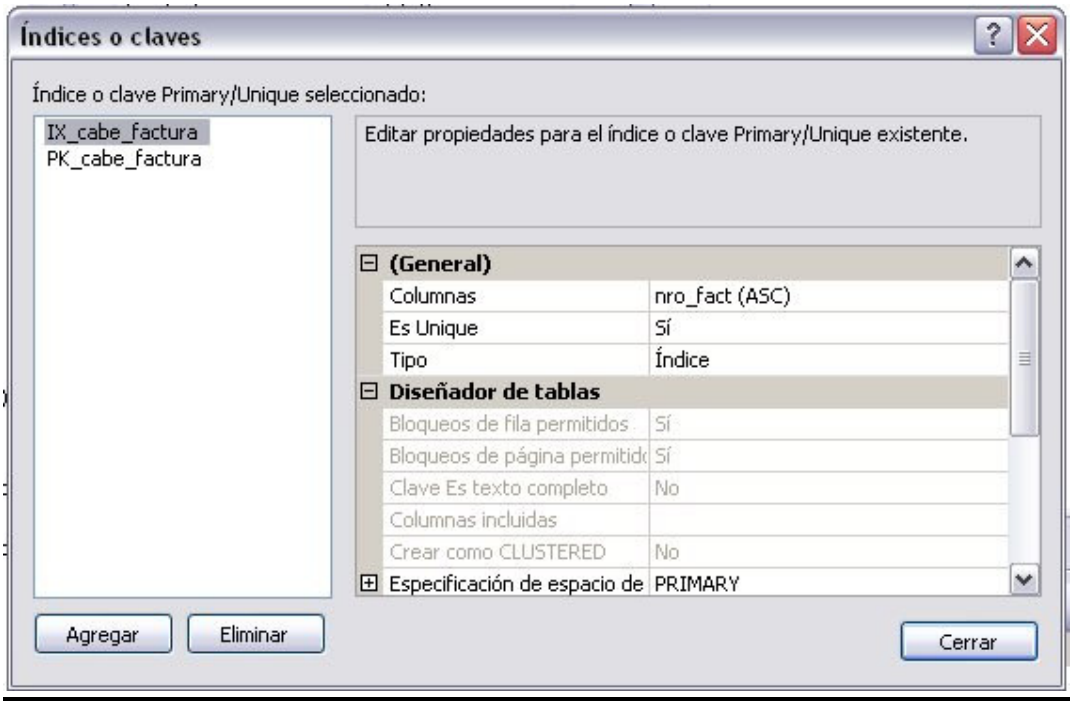
Solo crear estos índices a campos de búsqueda mas frecuentes, no abusar de la definición de campos índices ya que al final hace lento la grabación y la búsqueda. Para nuestro ejemplo solo estamos creando el índice al nombre del cliente porque suponemos que será el campo por el cual realizaremos la búsqueda siempre y cuando no sabemos el código.

Aquí un ejemplo para hacerlo vía sentencia:

```
CREATE NONCLUSTERED INDEX [IX_cliente] ON [dbo].[cliente]
([nomcli] ASC)
WITH
(PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON)
ON
```

```
[PRIMARY]
```

DEFINIENDO INDICES UNICOS

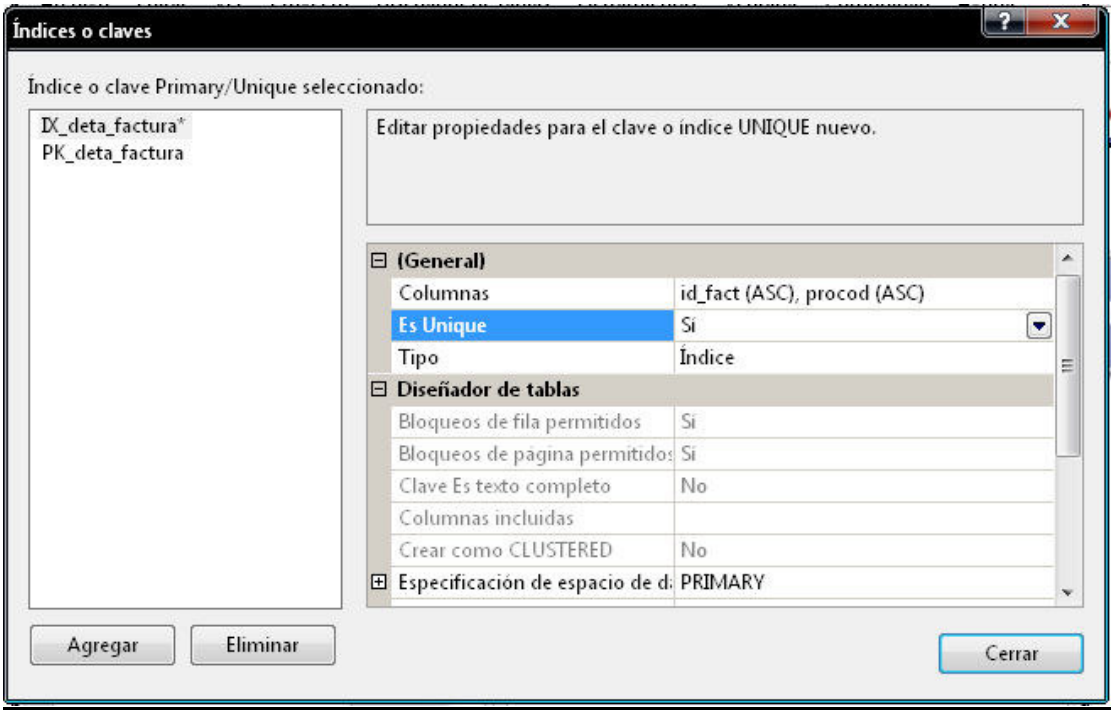


La utilidad de este índice es la de controlar duplicidad de datos para nuestro ejemplo será el Número de factura, ya que es un dato que no tendría que repetirse; ya se sabe que un índice de tipo PK ya por defecto es de tipo “unique”.

Aquí el ejemplo vía sentencia:

```
CREATE UNIQUE NONCLUSTERED INDEX [IX_cabe_factura] ON
[dbo].[cabe_factura]
([nro_fact] ASC)
WITH
(PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB =
OFF,
 IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS  = ON
, ALLOW_PAGE_LOCKS  = ON)
 ON [PRIMARY]
```

DEFINIENDO INDICE UNICO COMPUESTO

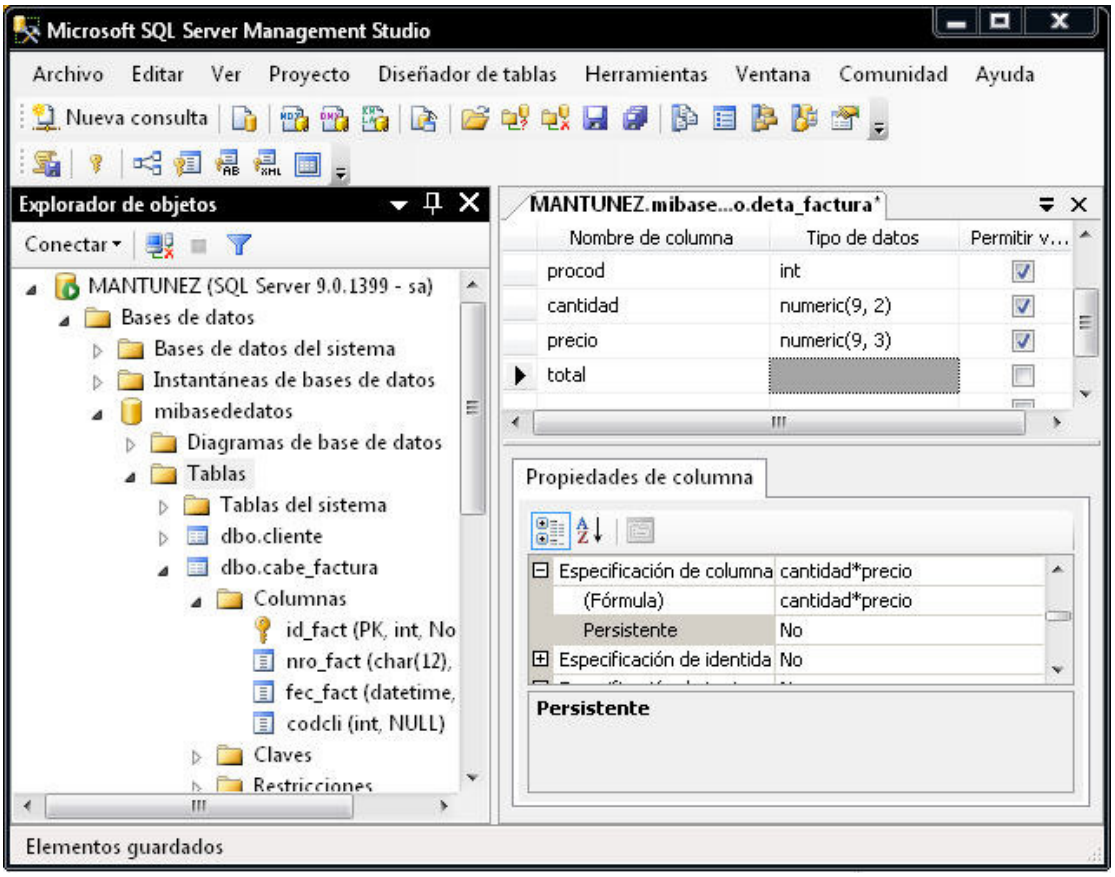


Así como se puede definir índice único a un solo campo también se puede hacer de la misma forma con mas campos, en el ejemplo se restringe la duplicidad de productos por factura.

Y aquí el ejemplo de hacerlo vías sentencia.

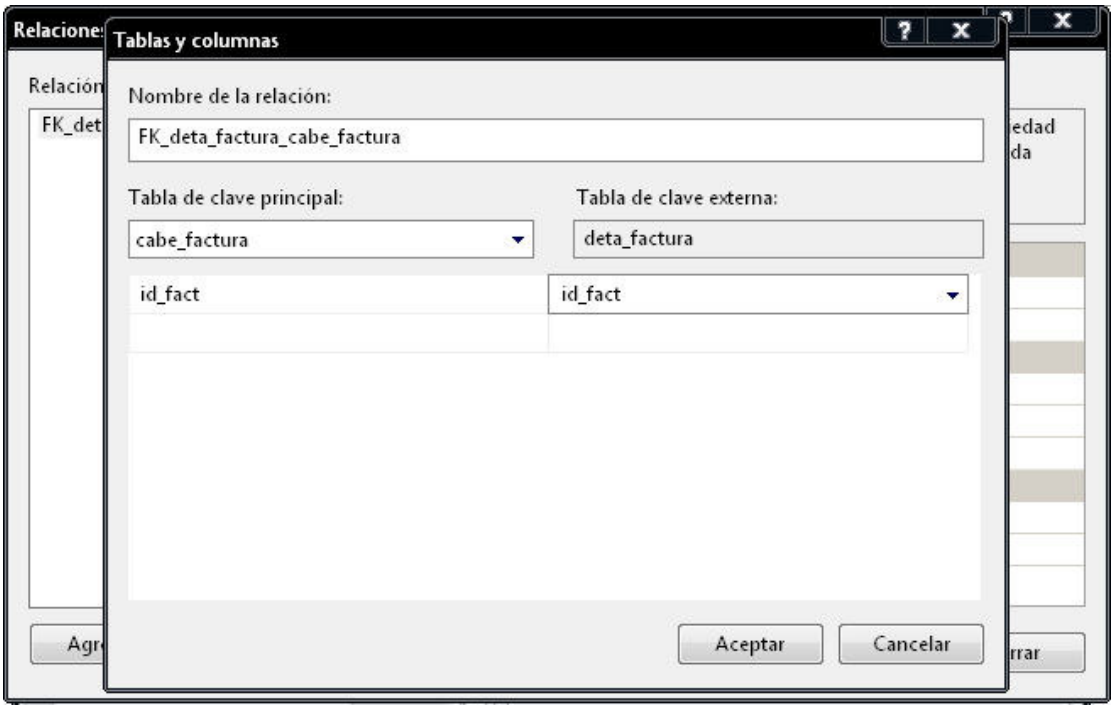
```
CREATE UNIQUE NONCLUSTERED INDEX [IX_deta_factura] ON
[dbo].[deta_factura]
([id_fact] ASC,[procod] ASC)
WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB
= OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
```

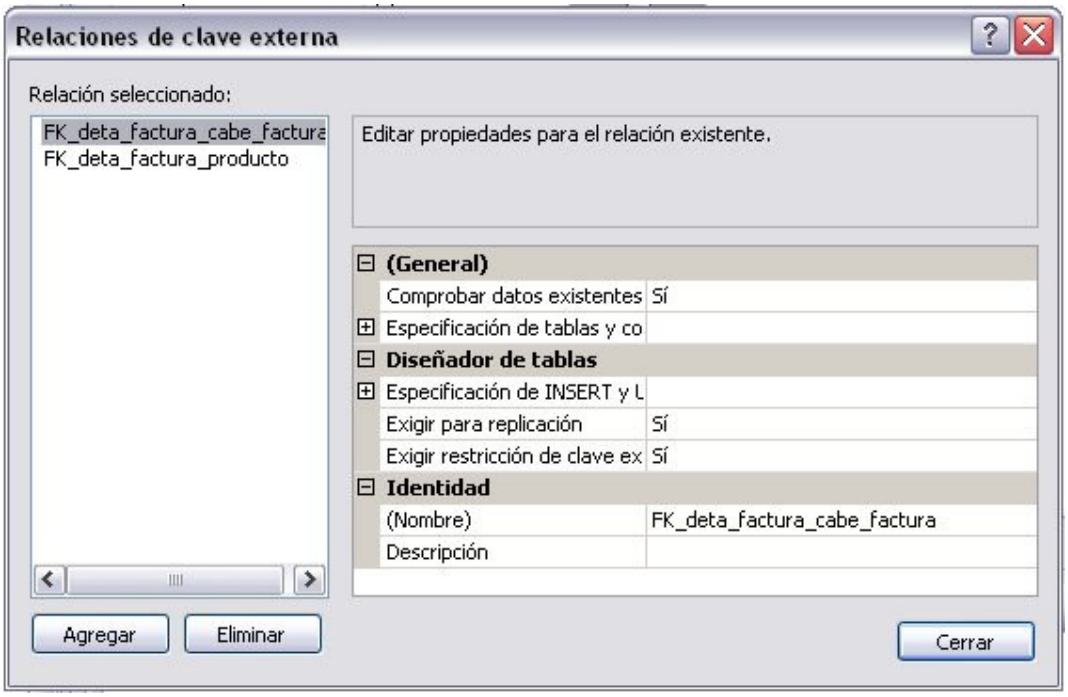
CAMPO CALCULADO



Hay campos que nos ayudan mucho, como son los campos calculados, la idea es no grabar directamente datos que son calculados, en el ejemplo el campo “total” es un cálculo de “Cantidad*precio” y de la forma que está definido no ocupa espacio en disco, ya que la propiedad persistente está como “no”.

RELACIONES





Teniendo ya nuestras tablas creadas tenemos que relacionarlas, ¿Por qué relacionarlas?, hay muchas ventajas de relacionar nuestras tablas una de ellas es mantener una integridad referencial.

Las relaciones en nuestra base de datos es como sigue:

"Cliente.codcli" con "cabe_factura.codcli"

"cabe_factura.id_fact" con "deta_factura.id_fact"

"producto.procod" con "deta_factura.procod"

DEFINIENDO PROCEDIMIENTOS ALMACENADOS.

Lo ideal es tener todas nuestras interacciones con las tablas a través de "stored procedures"

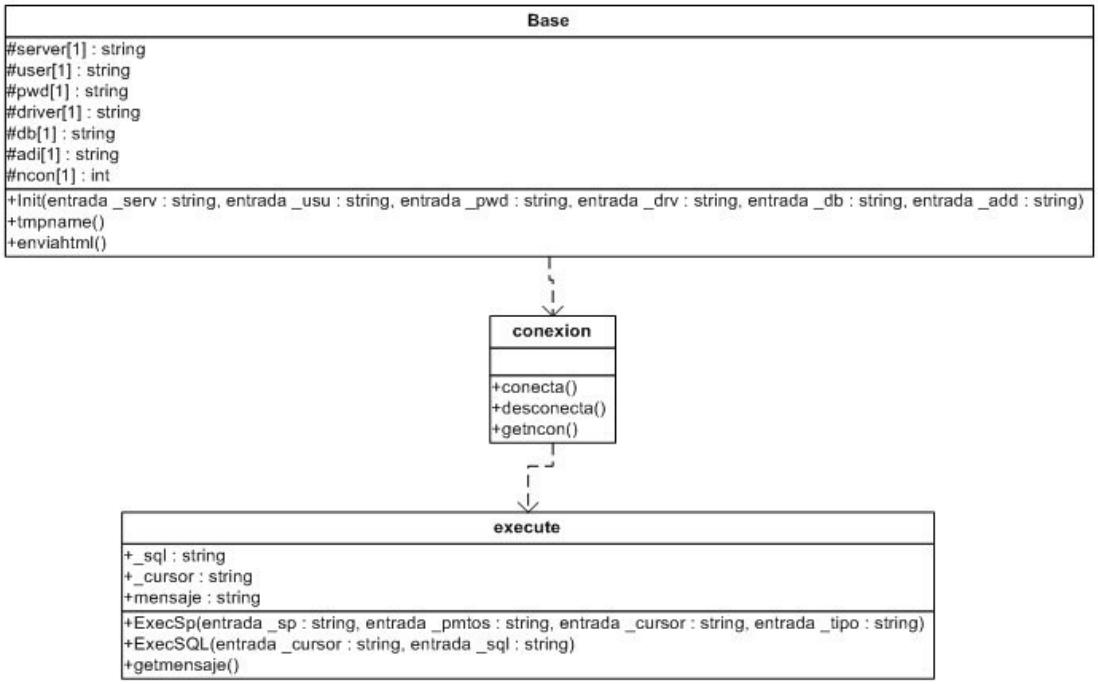
Aquí un ejemplo para la interacción con la tabla cliente.

```
CREATE PROCEDURE MIAPLI_Cliente_Select
as
select codcli,nomcli,dircli,telcli from cliente
GO
GRANT  EXECUTE  ON MIAPLI_Cliente_SELECT TO [usuarios_execute]
GO

CREATE PROCEDURE MIAPLI_Cliente_Inserta
@codcli int output,
@nomcli varchar(50),
@dircli varchar(80),
@telcli char(10)
AS
insert into cliente (nomcli,dircli,telcli) VALUES
(@nomcli,@dircli,@telcli)
set @codcli=(SELECT @@IDENTITY)
GO
GRANT  EXECUTE  ON MIAPLI_Cliente_Inserta TO [usuarios_execute]
GO

CREATE PROCEDURE MIAPLI_Cliente_Edita
@codcli int,
@nomcli varchar(50),
@dircli varchar(80),
@telcli char(10)
AS
update cliente set nomcli=@nomcli,dircli=@dircli,telcli=@telcli where
codcli=@codcli
GO
GRANT  EXECUTE  ON MIAPLI_Cliente_Edita TO [usuarios_execute]
GO

CREATE PROCEDURE MIAPLI_Cliente_Borra
@codcli int
AS
delete from cliente where codcli=@codcli
GO
GRANT  EXECUTE  ON MIAPLI_Cliente_Borra TO [usuarios_execute]
GO
```

Si deseamos hacer un diferente select o otra interacción con nuestra tabla tendria que ser a traves de otro “stored procedures “ los procedures de ejemplo son bastante basicos pero suficientes para las interacciones basicas que se tiene sobre una tabla. A igual modo se tendria que hacer con las demas tablas (producto,cabe_factura,deta_factura).

CREANDO LA CAPA CONEXIÓN

La clase conexión tiene la siguiente composición:



CLASE BASE

base.prg

```
define class Base as session
    PROTECTED servidor,usuario,pwd,driver,db,adi,ncon
    function init(_serv as String@,_usu as String, _pwd as String,_drv as String,_db as String,_add as string)
        this.servidor=alltrim(_serv)
        this.usuario=alltrim(_usu)
        this.pwd=alltrim(_pwd)
        this.driver=alltrim(_drv)
        this.db=alltrim(_db)
        this.adi=alltrim(_add)
        this.ncon=0
        set talk        off
        set safety      off
        set echo        off
        set notify      off
        set status      off
        set brstatus    off
        set bell        off
        *
        set carry       off
        set delete      on
        set exact       on
        set near        off
        set exclusive   off
        set multilock   on
        set cursor      on
        *
        set century     on
        set century     to 19 ROLLOVER 50
        set date        to dmy
        set hours       to 24
        set point       to "."
        set separator   to ","
    ENDFUNC
    !!
```

Ahí notamos que la clase base está basado en una clase “session” y se ve la creación del método “init()”, que tiene a demás las propiedades protegidas como servidor, usuario, pwd, etc. Y el método “init()” recibe parámetros y le asigna a las propiedades mencionadas.

CLASE CONEXION

conex.prg

```
DEFINE CLASS conexion as base
    FUNCTION conecta
        LOCAL _ncon,_cadena
        WITH this
        IF EMPTY(.PWD)
            _cadena="Driver="+.driver+";Server="+.servidor+";DATABASE="+.db+iif(!empty(.adi)
        ELSE
            _cadena="Driver="+.driver+";UID="+.usuario+";PWD="+.pwd+";Server="+.servidor+";I
        ENDIF
        IF .ncon<=0
            .NCON= sqlstringconnect(_cadena)
            IF .ncon>0
                IF !.usuario$"sa/administrador"
                    VDEV=SQLEXEC(.ncon,"EXEC sp_setapprole 'Usuarios_execute', 'mipwd'")
                endif
            endif
        endif
        RETURN .NCON
        ENDWITH
    ENDFUNC
    PROCEDURE getncon
        RETURN this.ncon
    ENDPROC
    FUNCTION desconecta
        IF this.ncon>0
            =SQLDISCONNECT(this.ncon)
        endif
        this.ncon=0
    ENDFUNC
ENDDEFINE
```

Es una clase basada en la clase “Base” en cual cuenta solo con tres métodos, el primero el método “conecta” hace uso de las propiedades de la clase base como driver, usuario, pwd, etc. La principal función del método es crear la conexión con la base de datos y reservar dicho número de conexión. Adema notamos que cuando el usuario no es “sa” o “administrador” ejecuta un rol , este rol es aquel que tiene accesos a los “stored procedures “ porque los usuarios no tienen acceso directamente a los “stored procedures”, menos a las tablas, si no a través de este rol “usuarios_execute”, que es un rol de aplicación.

El método “getncon” lo único que hace es devolver el número de conexión a la base de datos.

Y el método desconecta cierra la conexión a la base de datos.

CLASE EXECU

execu.prg

```
DEFINE CLASS EXECU as conexion
    PROTECTED _sql,_cursor,MENSAJE
    PROCEDURE ExecSp(_SP as String@,_pmtos as string,_cursor as String,_tipo)
        IF EMPTY(_SP)
            THIS.MENSAJE="SQL VACIO"
            RETURN -1
        ENDIF
        IF EMPTY(_pmtos)
            _pmtos=''
        ENDIF
        THIS.Conecta()
        IF EMPTY(_cursor)
            _DEV=SQLEXEC(this.ncon,"EXECUTE "+_SP+" "+_pmtos)
            IF _DEV<0
                AERROR(_ERR)
                THIS.MENSAJE=STR(_ERR(1))+' '+_ERR(2)
            ELSE
                THIS.MENSAJE=""
            ENDIF
            RETURN _DEV
        ELSE
            IF _tipo=.f.
                _DEV=SQLEXEC(this.ncon,"EXECUTE "+_SP+" "+_pmtos,_cursor)
                IF _DEV<0
                    AERROR(_ERR)
                    THIS.MENSAJE=_ERR(2)
                ELSE
                    THIS.MENSAJE=""
                ENDIF
            ELSE
                IF !EMPTY(_pmtos)
                    DO &_sp WITH _pmtos,_cursor
                    _DEV=1
                ELSE
                    DO &_sp WITH _cursor
                    _DEV=1
                endif
```

Esta clase basada en la clase conexión es la que al fin permite hacer la interacción con la base de datos a través de este método “ExecSP” pasan todos los “stored procedures” invocados de nuestra capa de negocios. Controla el tipo de “stored procedure” si devuelve o no un cursor y además con el parámetro”_tipo”, se podría determinar el tipo de origen de base de datos, en lo personal solo he tenido dos orígenes uno el SQLSERVER y Base de datos nativa de VFP. Lo único que diferencia es la manera de ejecutar el “stored procedures” en el SQLSERVER es el “EXECUTE” y en el VFP es el “DO”

Cualquier error ocurrido grabara en la propiedad “mensaje” .

CREANDO EL OBJETO CONE

Ya teniendo nuestra clase creada la pregunta de todas maneras es: ¿ya como lo uso? , aquí viene el ejemplo:

```
m.SERVIDOR=”MISERVIDOR”
```

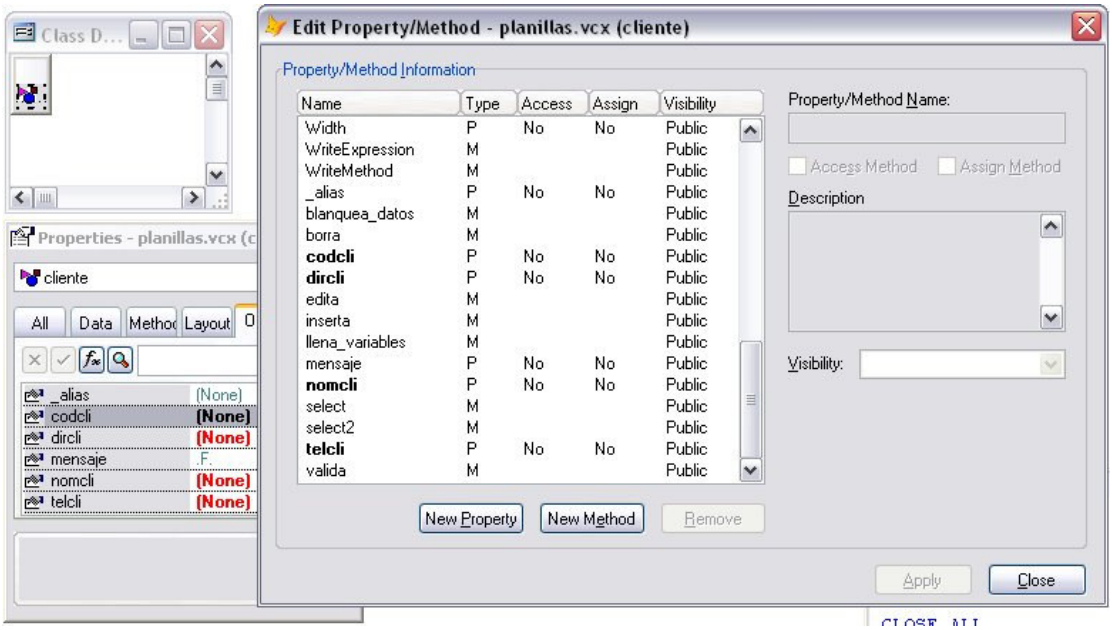
```
m.USUARIO=”JPEREZ”
m.PWD=”MIPWD”
m.Driver=”SQL SERVER”
m.Db=”MIBASEDEDATOS”
m.adi=”APP=MI SOFTWARE PERSONAL;LANGUAGE=Español”

SET PROCEDURE TO base.prg,conex.prg,EXECU.PRG
cone=CREATEOBJECT("EXECU",m.SERVIDOR,m.USUARIO,m.PWD,m.Driver,m.Db,m.a
di)
SET DATASESSION TO CONE.DataSessionId
SET DELETED on
```

CREANDO LA CAPA NEGOCIOS

Para esta capa estoy utilizando las clases no visuales de VFP que son los de tipo “Custom”, que pueden ser reemplazados por el que más se adecuen por ejemplo tipo “cursoradapter”.

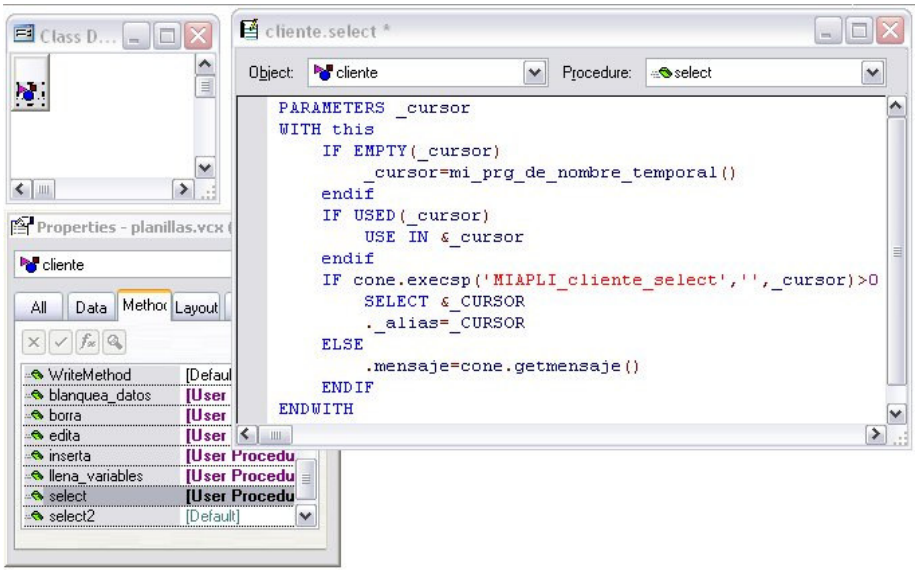
La idea es tener algunos métodos básicos de interacción con nuestros “stored procedures” pero a través de nuestra capa de conexión.

Para el ejemplo crearemos el objeto cliente que es laque interactúa con la taba de clientes.



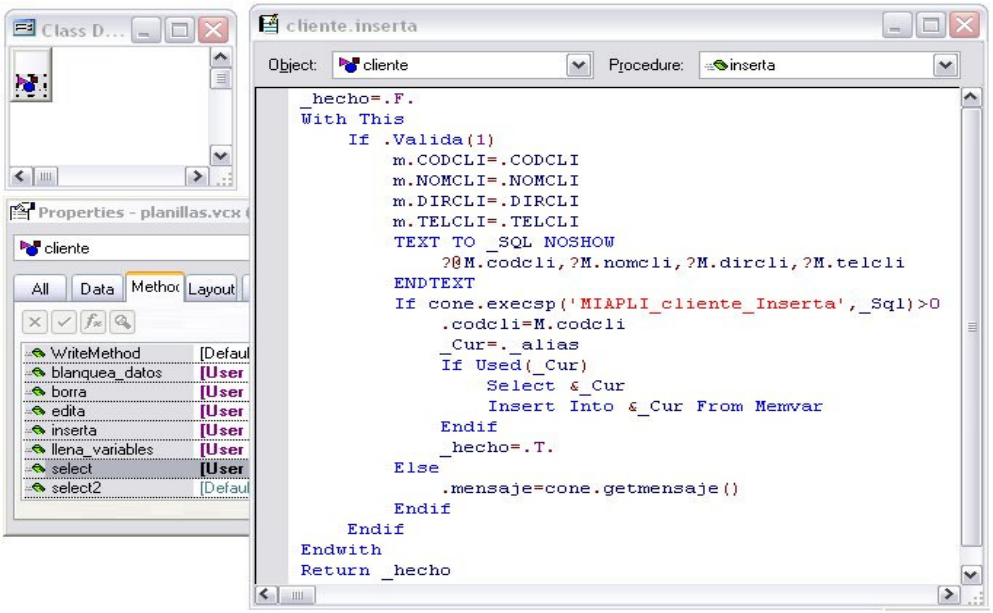
Tenemos como propiedades los campos de la tabla, y los métodos (select, inserta, edita, borra, valida, etc.)

Aquí la programación que tendría los métodos



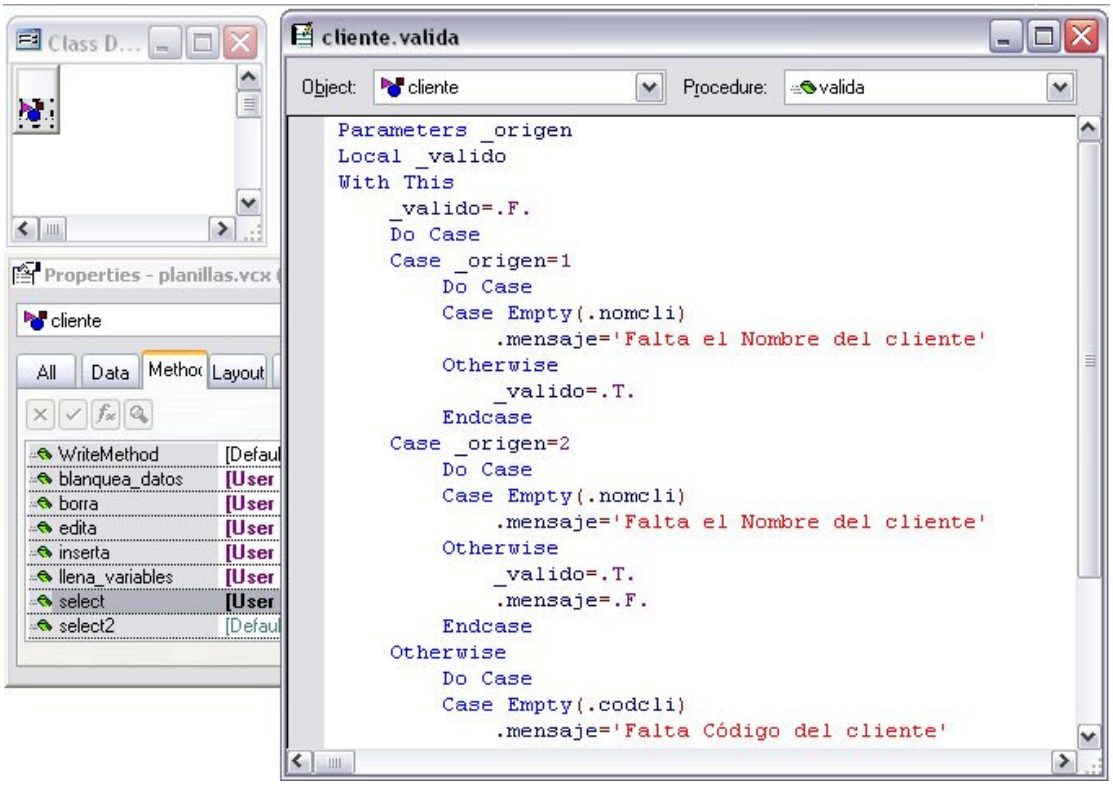
En el grafico se nota claramente la interacción del objeto conexión.

La idea es cuando se invoque este método es la de devolver el cursor de la base de datos. Si no se le envía como parámetro el nombre del cursor optara por poner un nombre temporal, en lo personal la mejor forma. Cualquier interacción con el cursor devuelto lo hago a través de la propiedad "_alias" ya que es la que tendría el nombre del cursor asignado.



Con este método se hace la inserción de datos, además notaremos que la variable “m.codcli” tiene un @ adelante, es porque la variable en el “stored procedures” “@codcli” es de tipo “output”, después de ejecutar este procedimiento dicha variable tendrá el correlativo asignado por nuestro campo “identity”.

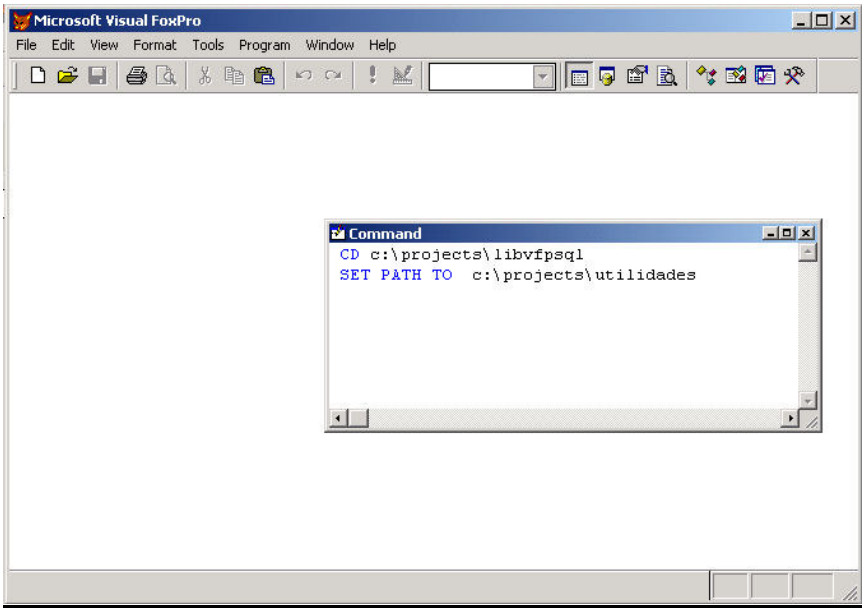
Además antes de ejecutar el “stored prodecures” pasa por un filtro que es el método valida, este método como su nombre lo dice validara la información como en el ejemplo valida que el “nomcli” no este vacia.



De igual manera completaremos el método “edita” y “borra” con sus “Stored Procedures” respectivos.

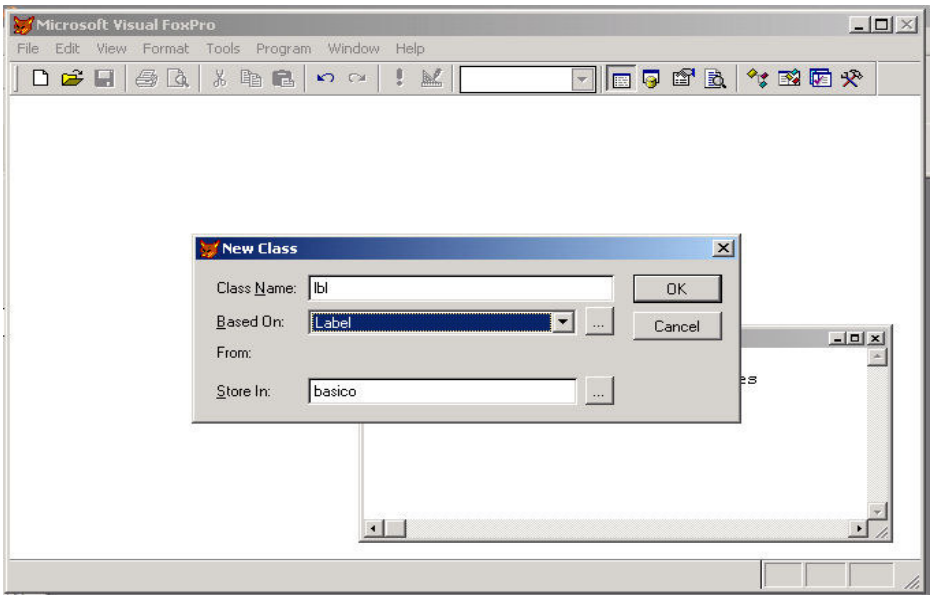
CREANDO LA CAPA INTERFAZ

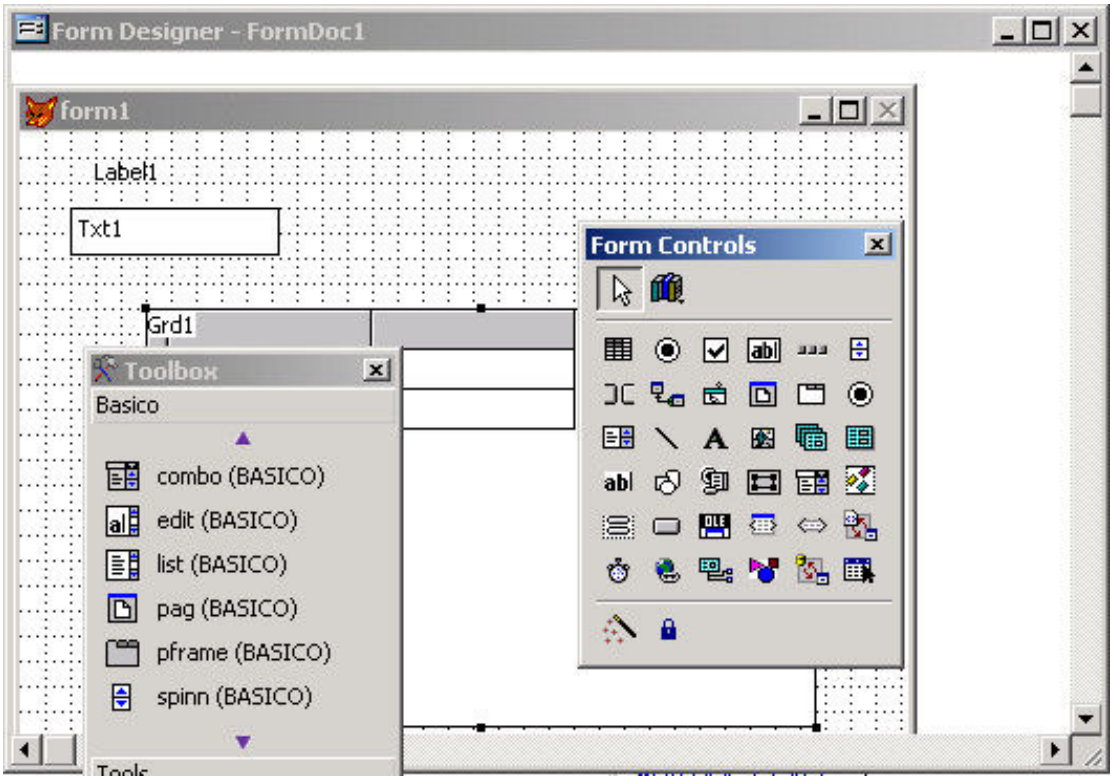
ENTORNO VFP



PREPARANDO CONTROLES PERSONALIZADAS

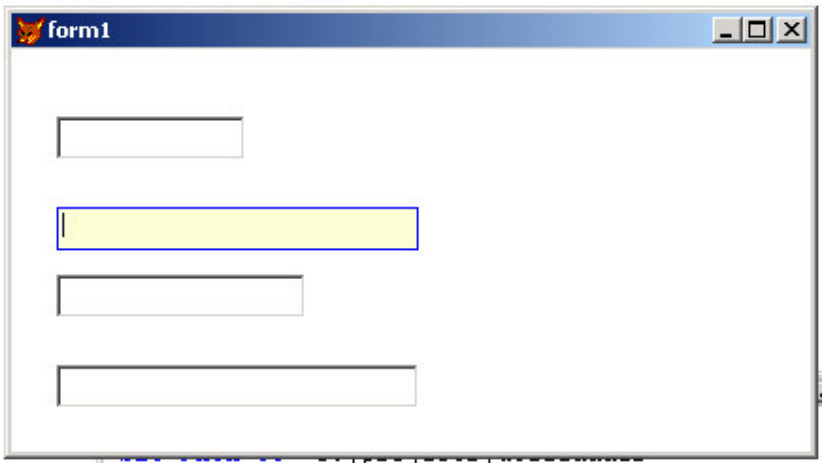
La idea es copiar todos los controles de VFP y ponerlos en una clase propia, en este caso a esa biblioteca lo nombre como básico, a estos controles le ponemos nombres mas sencillos, como por ejemplo a los “textbox” solo lo llamo “txt”, a los “label”,”lbl”, etc. Además de personalizalo a nuestro gusto en cuanto a aspecto. Para copiar dichos controles es sumamente sencillo, lo que hacemos es crea una nueva biblioteca de clases y nombrar los controles ya existentes por uno propio. En el grafico se ve la creación de nuestro control personalizado “lbl” basado en “label” y almacenado en la biblioteca de clases “basico”


Microsoft Visual FoxPro
File Edit View Format Tools Program Window Help
New Class
Class Name: lbl
OK
Based On: Label
Cancel
From:
Store In: basico

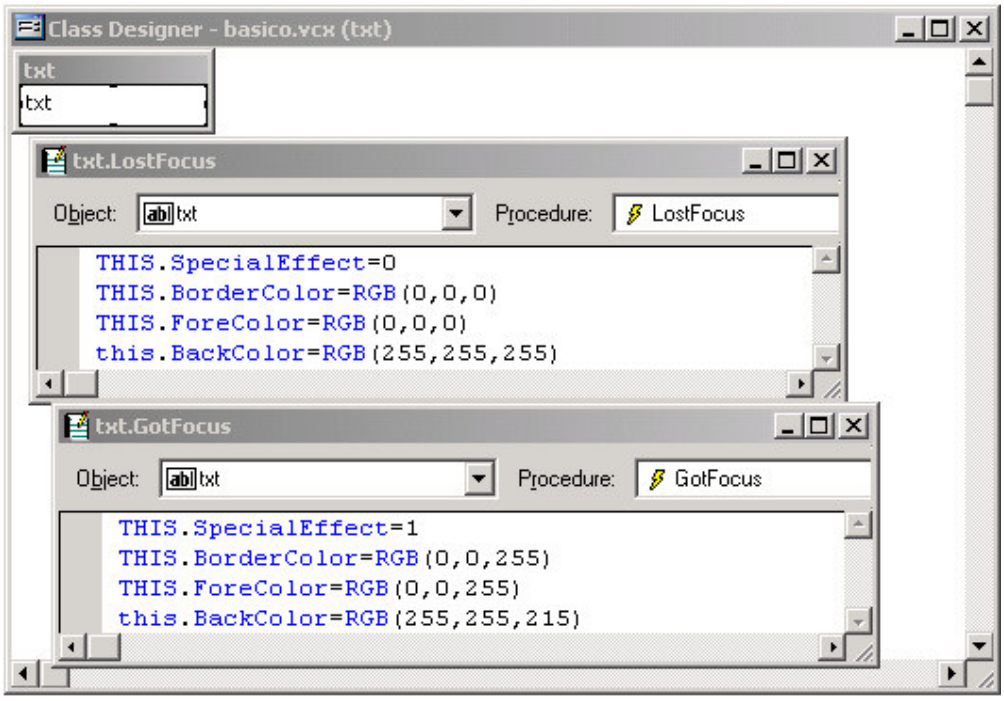

Form Designer - FormDoc1
form1
Label1
Txt1
Grd1
Toolbox
Basico
combo (BASICO)
edit (BASICO)
list (BASICO)
pag (BASICO)
pframe (BASICO)
spinn (BASICO)
Form Controls

y si es posible que pueda tener cierta interacción como por ejemplo:

Al control “textbox” vamos a cambiarlo de color cuando esta seleccionado



El código para tener este efecto es el siguiente: La programación está en el “lostfocus” y el “gotfocus”

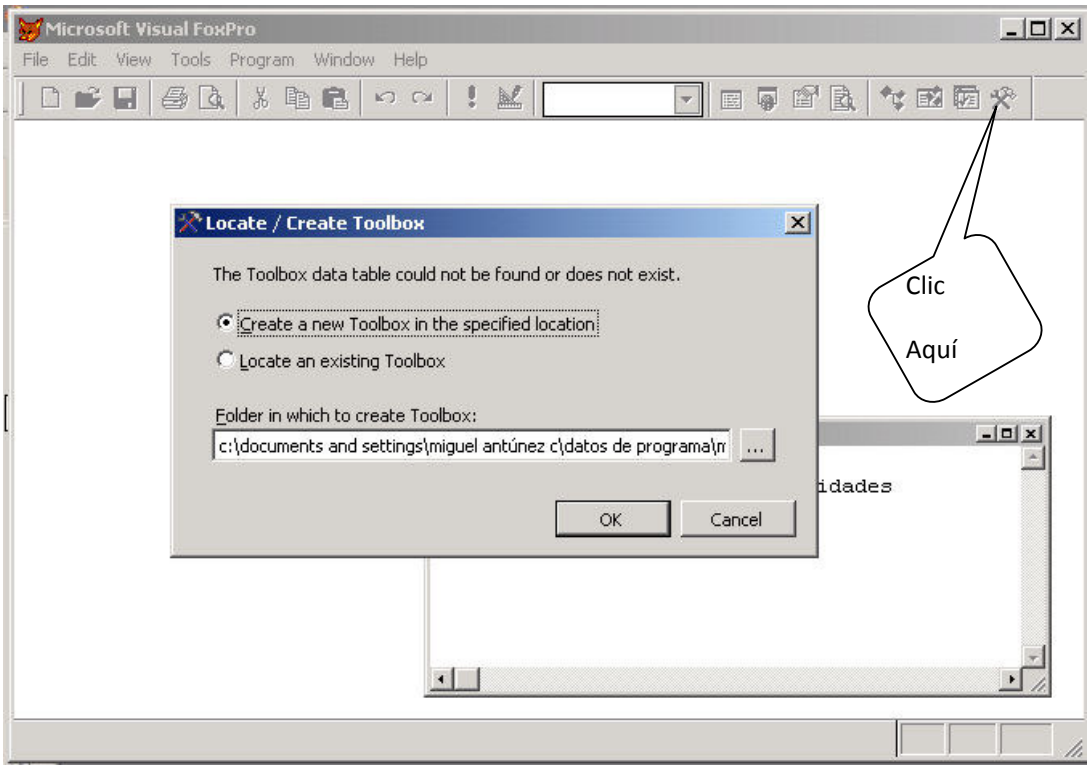


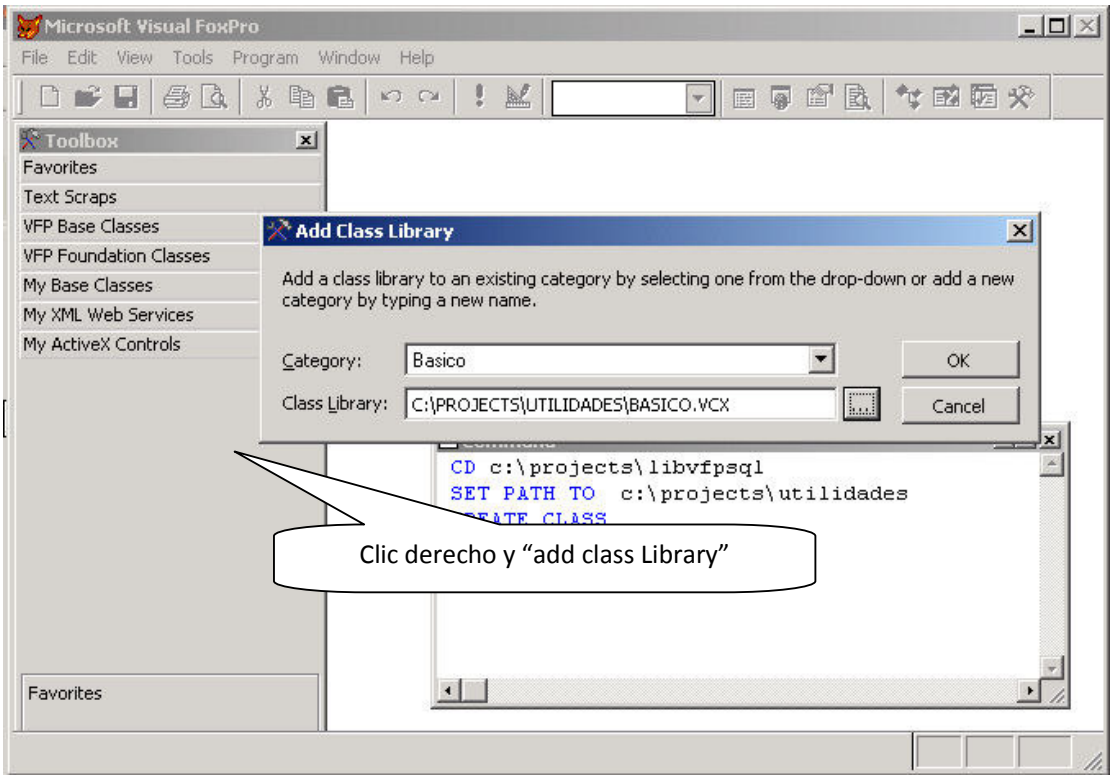
Esta demás decir que de ahora en adelante solo usaremos nuestros controles para todo tipo de interfaz, desde un form, textbox, grid, labels, etc.

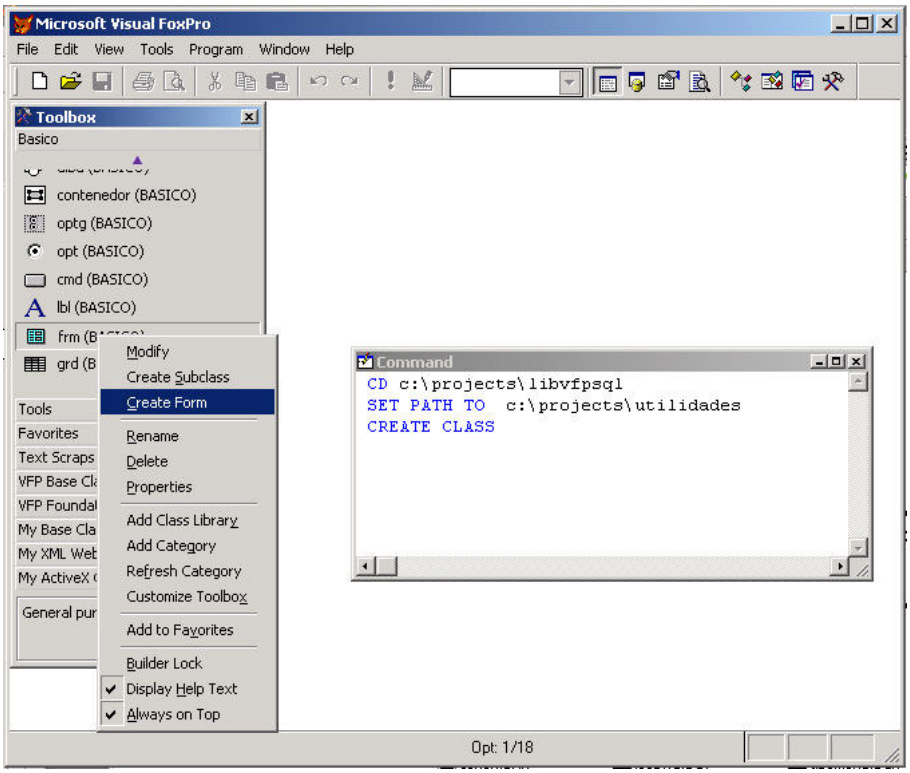
FACILIDADES PARA EL USO DE CONTROLES PERSONALIZADOS

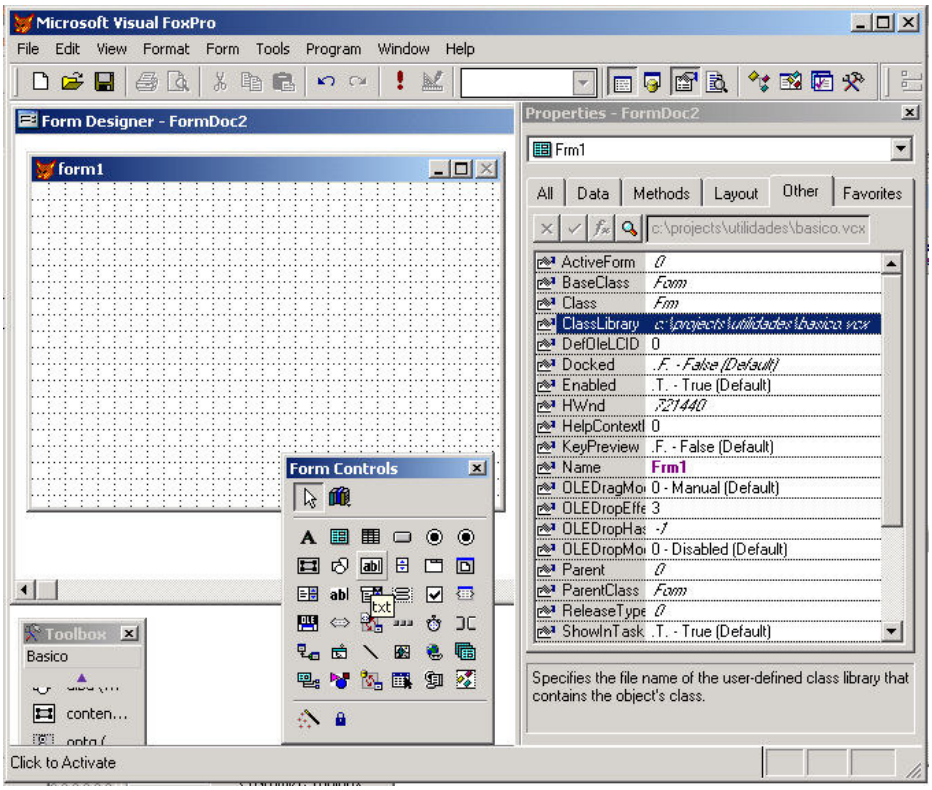
Hay una herramienta que trae VFP que se llama “toolbox” el cual facilita el manejo de nuestros controles creados.

Aquí un ejemplo de su creación, anexar nuestros controles personalizados a dicho “toolbox” y a final la creación de un formulario basado en nuestra clase “form” personalizada.



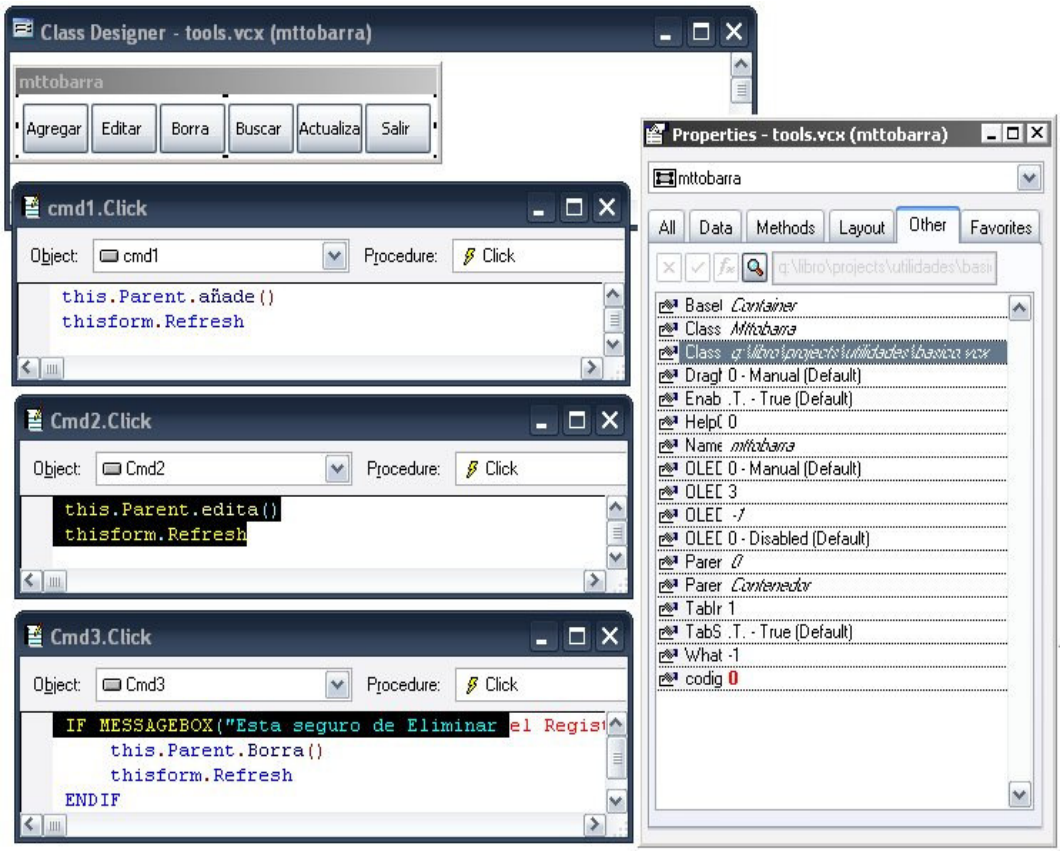

Microsoft Visual FoxPro
File Edit View Tools Program Window Help
Toolbox
Favorites
Text Scraps
VFP Base Classes
VFP Foundation Classes
My Base Classes
My XML Web Services
My ActiveX Controls
Add Class Library
Add a class library to an existing category by selecting one from the drop-down or add a new category by typing a new name.
Category:
Basico
Class Library:
C:\PROJECTS\UTILIDADES\BASICO.VCX
OK
Cancel
CD c:\projects\libvfpsql
SET PATH TO c:\projects\utilidades
Clic derecho y "add class Library"
Favorites


Microsoft Visual FoxPro
File Edit View Tools Program Window Help
Toolbox
Basico
contenedor (BASICO)
optg (BASICO)
opt (BASICO)
cmd (BASICO)
lbl (BASICO)
Modify
Create Subclass
Create Form
Rename
Delete
Properties
Add Class Library
Add Category
Refresh Category
Customize Toolbox
Add to Favorites
Builder Lock
Display Help Text
Always on Top
Command
CD c:\projects\libvfpsql
SET PATH TO c:\projects\utilidades
CREATE CLASS
Opt: 1/18



CREANDO CLASE MANTENIMIENTO DE TABLAS

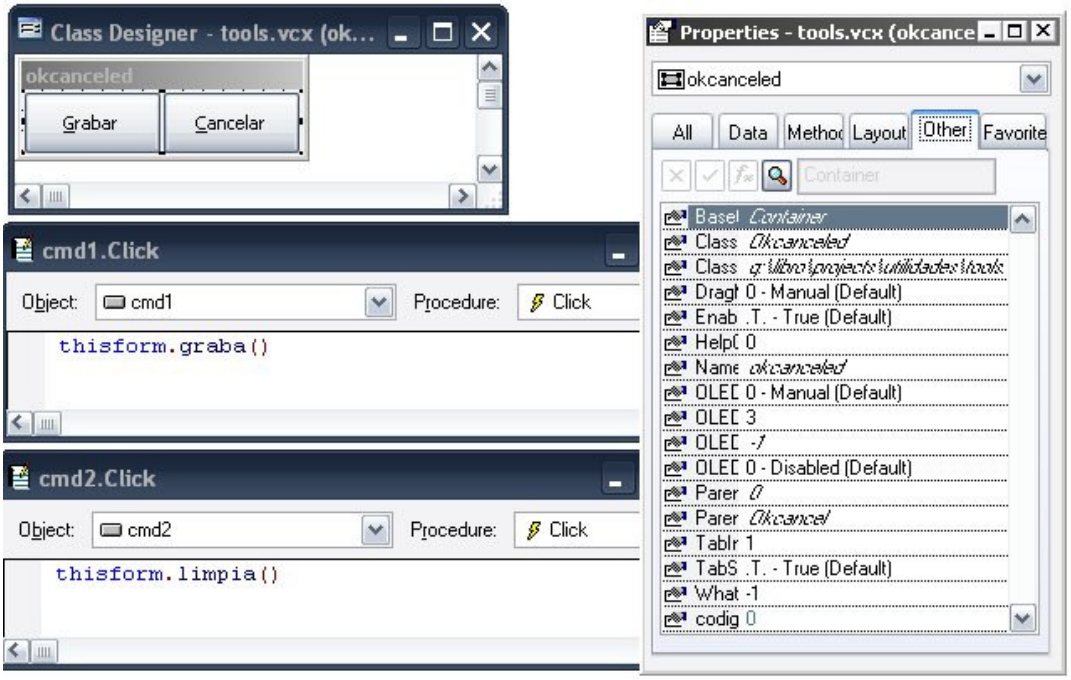
Crearemos una clase base basada en “Cmd” de nuestra clase “Base” con funciones comunes, como agregar, edita, eliminar, salir. Y otra de Aceptar y Cancelar



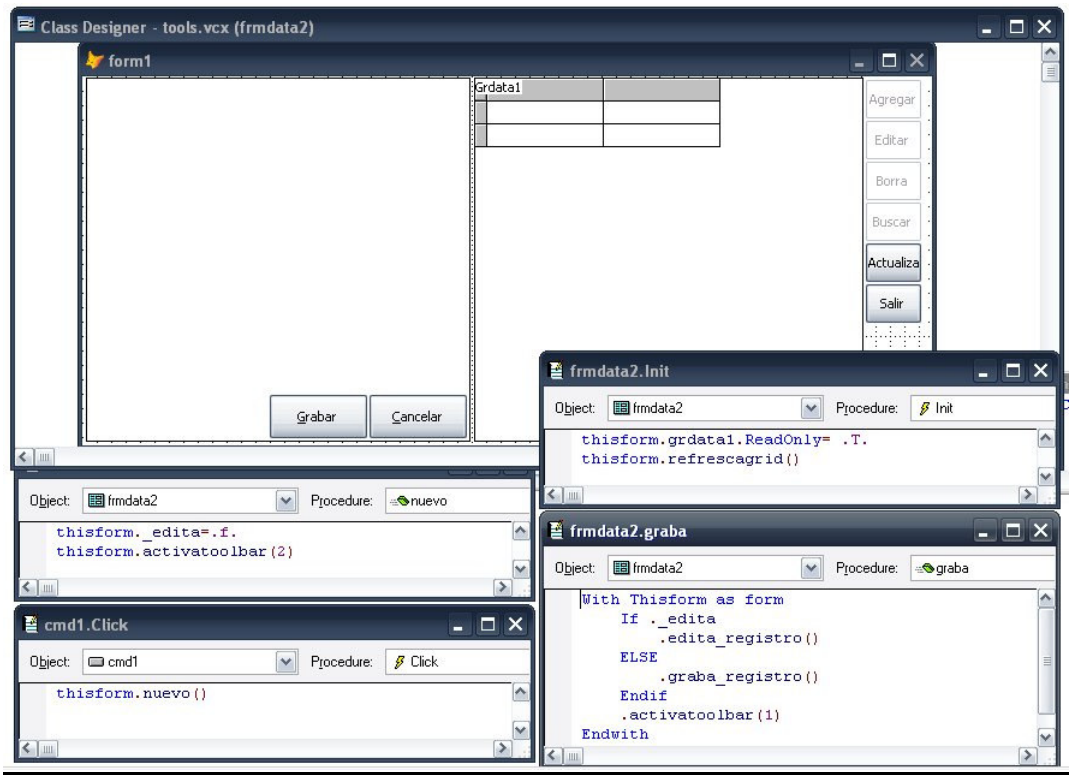
Creamos los métodos (añade, edita, borra, etc.) en la clase padre en este caso el “container”,

Y a los botones asignamos esos el código para invocarlos.

Y la clase “Ok_Cancel” de la siguiente forma:



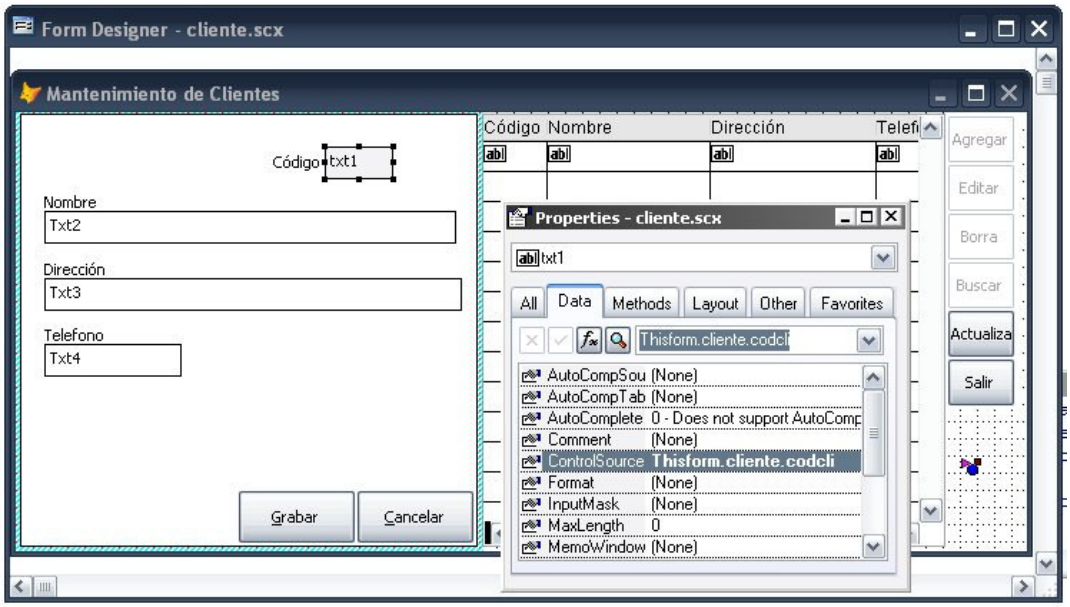
CREACIÓN CLASE DE FORMULARIO DE MTTO.



CREACIÓN DE LA INTERFAZ PARA EL OBJETO CLIENTE

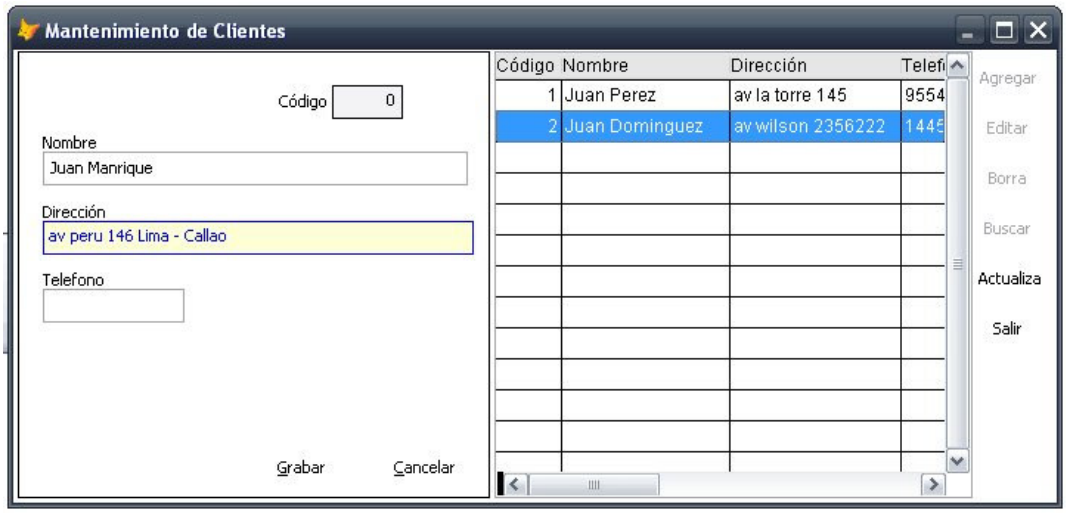
Empezamos con el código:

```
CREATE FORM cliente as frmdata2 FROM tools.
```



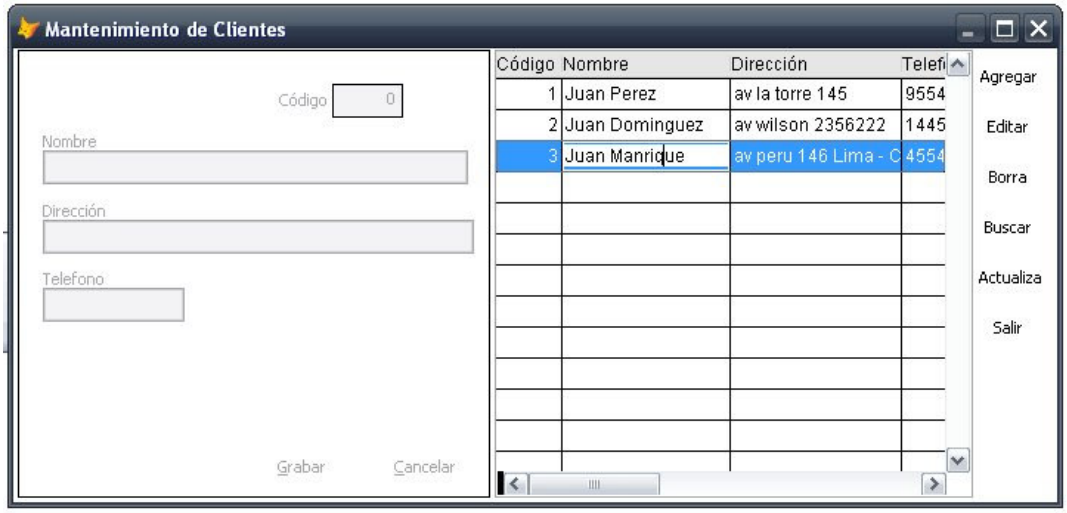
Lo único que tenemos que hacer en nuestro formulario es enlazar nuestros objetos interfaz con nuestro objeto negocio, en este caso enlazarlos por medio del "controlsource" del "Textbox", en cada uno de estros controles enlazar con la propiedades del objeto negocio, en el ejemplo es el ("txt1" con cliente.codcli), etc.

Uan vez hecho eso ya podremos probar nuestro formulario, sin olvidarnos que deben ser ejecutados desde nuestro programa principal, ya que ese programa hace la conexión a la base de datos ademas que carga nuestra configuración de usuario.

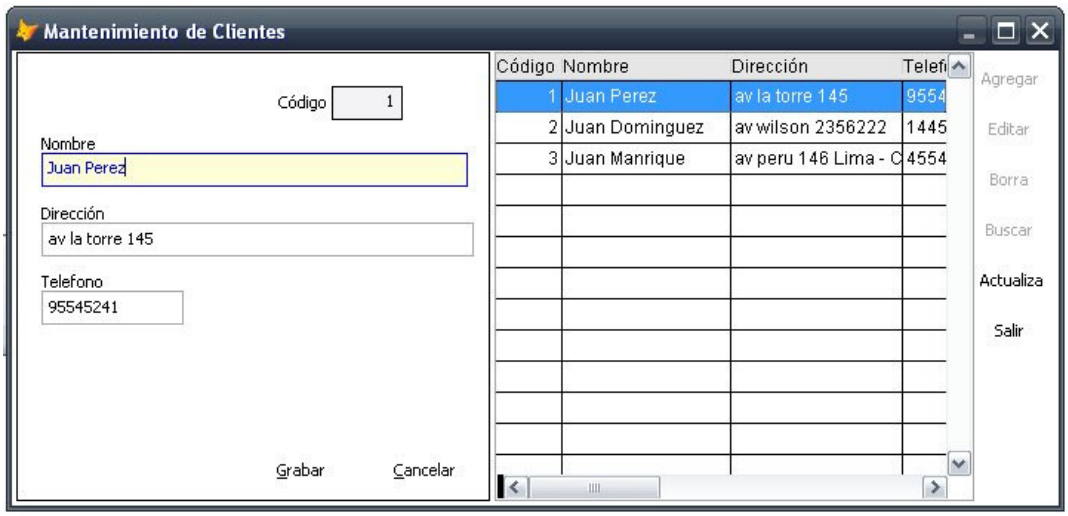


Para el ejemplo de formulario se puso por defecto que se va a agregar un cliente por eso que aparecen desactivados los botones de nuestra barra de mantenimiento y aparece activada el grabar o cancelar,

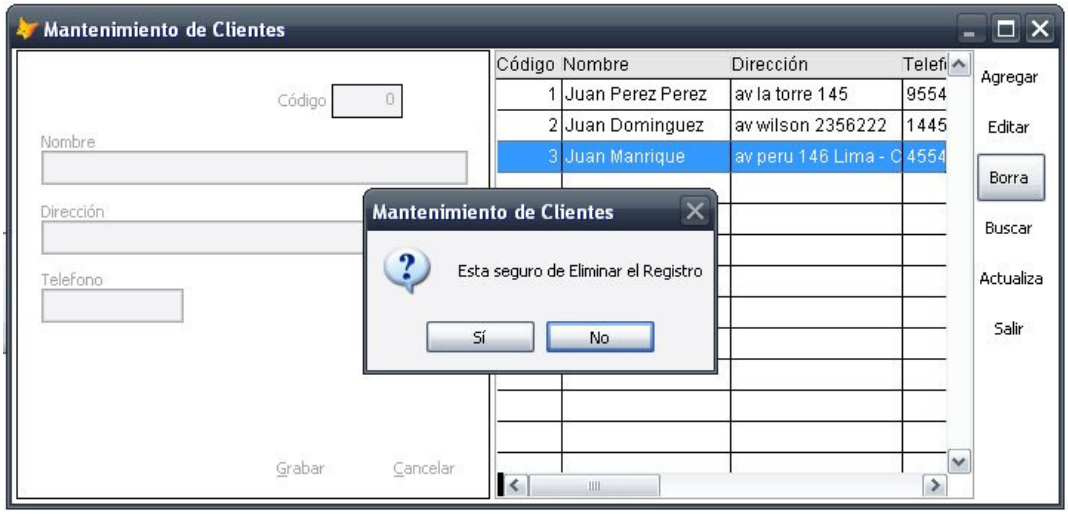
Al hacer cualquiera de estas dos acciones activaran los botones de la barra de mantenimiento y desactivaran el grabar y cancelar.



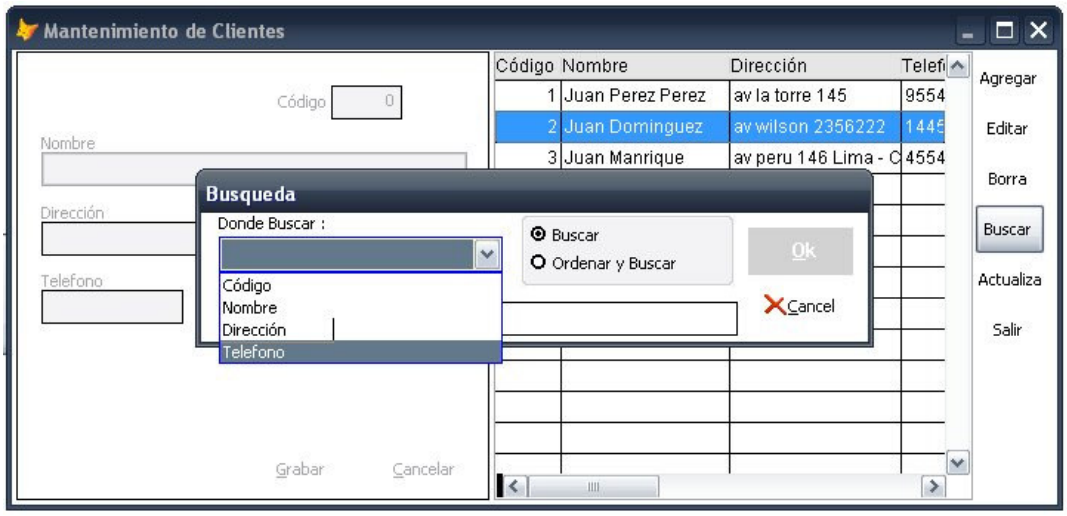
La edición tiene un efecto similar al agregar pero el "codcli" ya esta con dato el cual no puede ser editado por ser el "Primary Key".



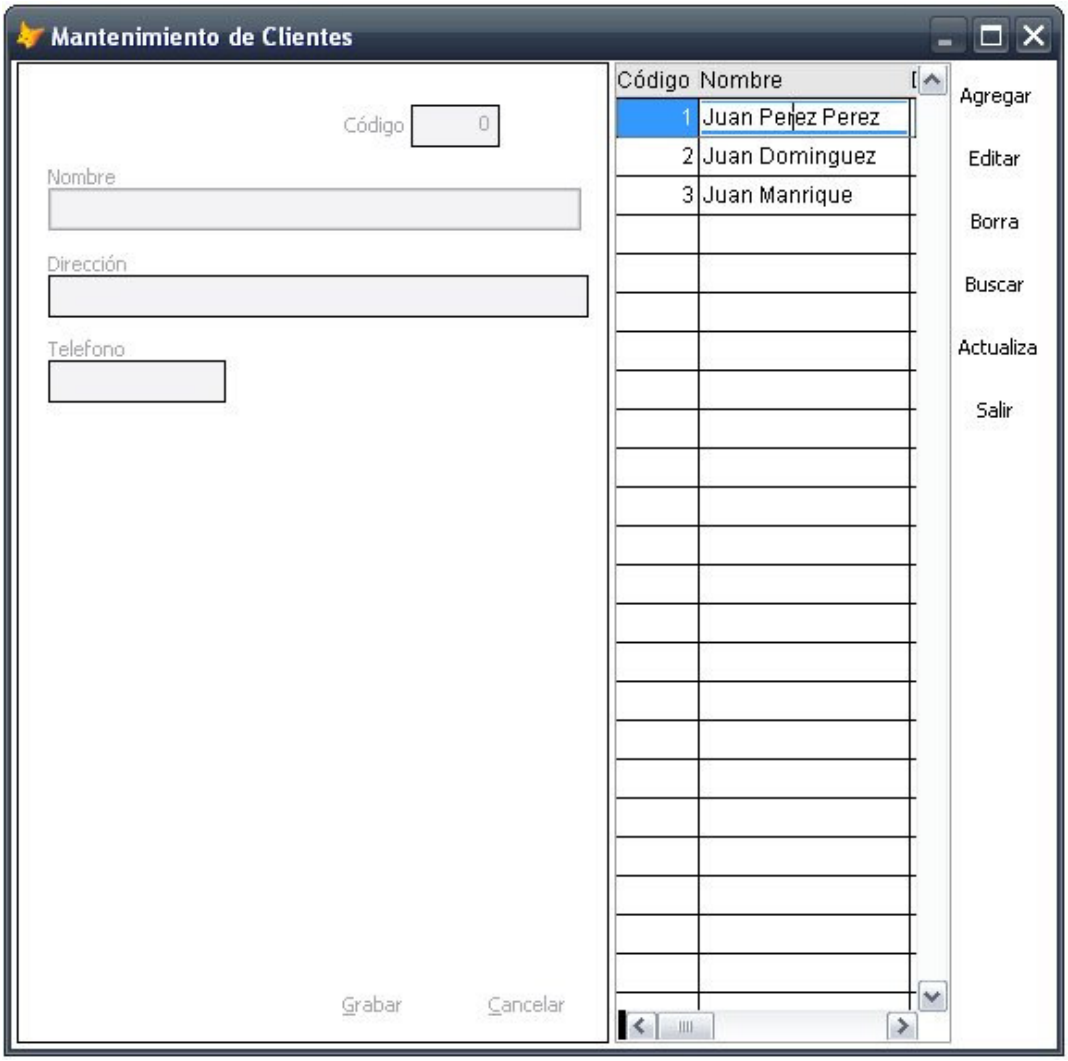
Al momento de eliminar la clase ya tiene esta preguntada asignada. Así que va a salir en todas nuestra pantallas de la misma manera.



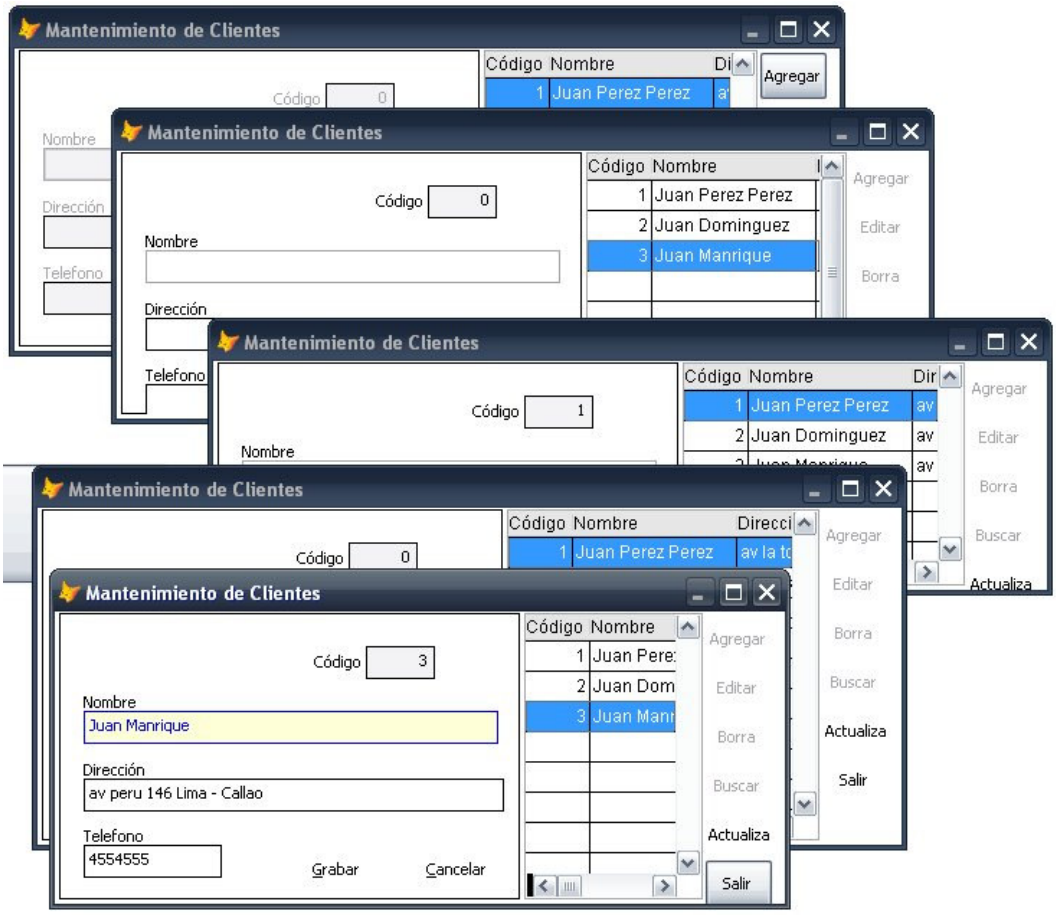
Nuestra búsqueda también esta integrada en la clase y toma el nombre de todas las columnas del "grid".



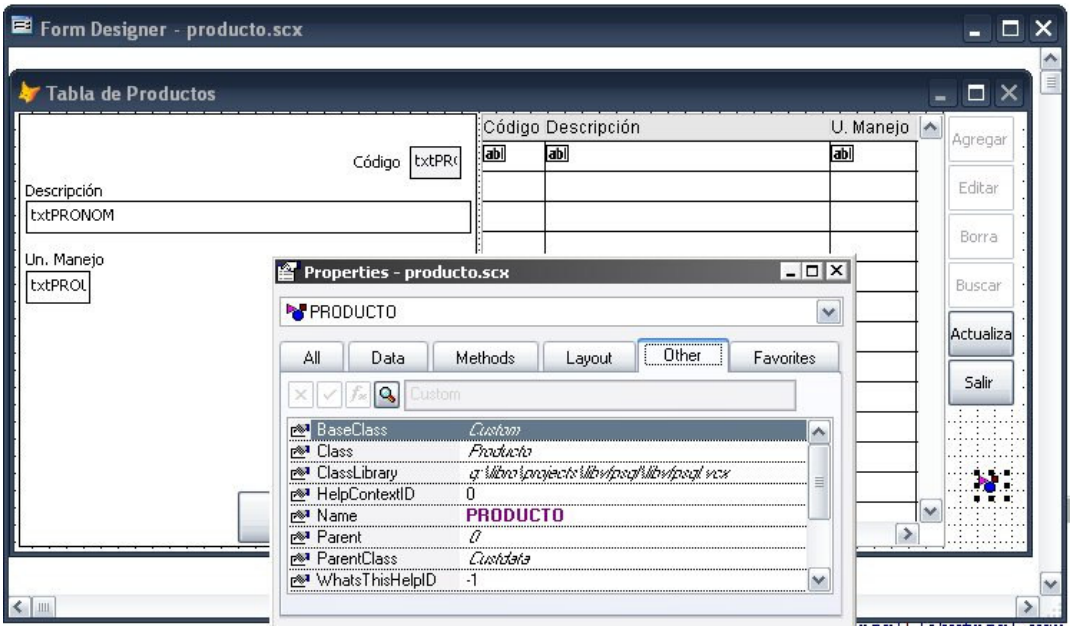
Estas pantallas también son "Resizables" para la comodidad del usuario.



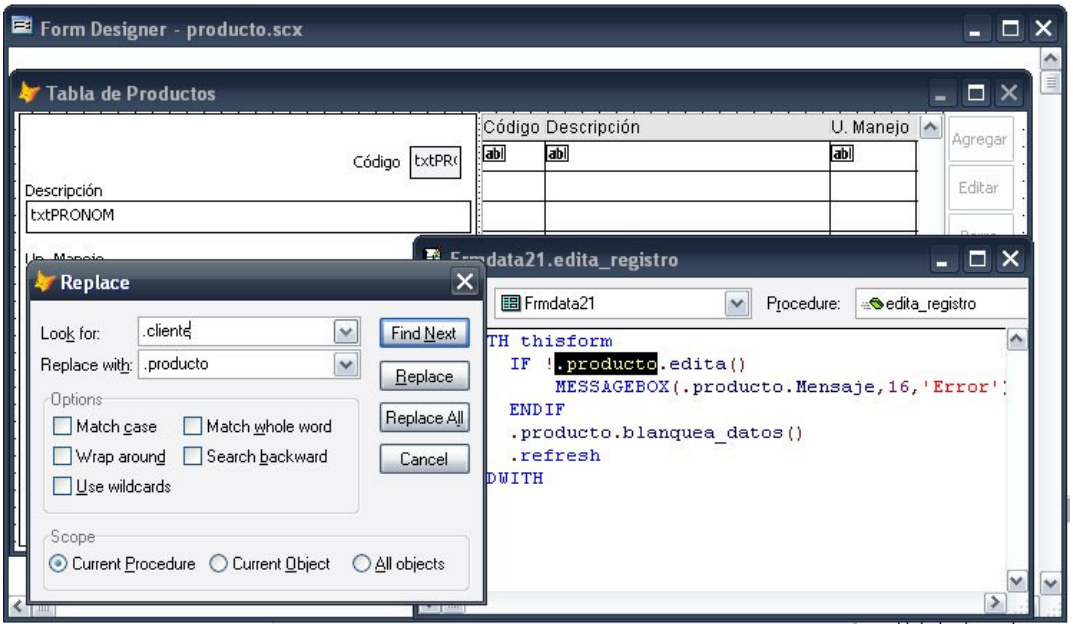
Además que soportan ser cargadas mas de una vez y sin problema alguno ya que utilizan cursores con nombres aleatorios.

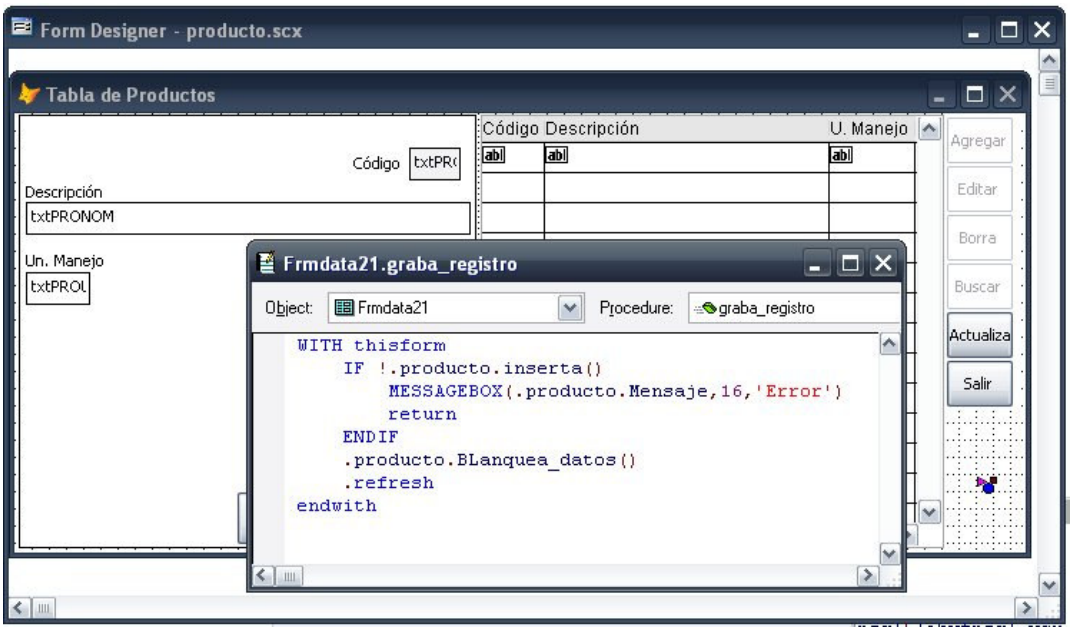


Para hacer la siguiente pantalla se nos hace muy sencillo, la tabla cliente guardamos con el nombre de producto, cambiamos nuestro objeto negocio de cliente por producto.



Y en nuestros métodos reemplazamos para nuestro caso la palabra cliente por producto y asignamos nuestro controlsource de los objetos interfaz a nuestro nuevo objeto negocio.





Y ya podemos probar nuestra nueva pantalla con la misma funcionalidad de la primera