INSTITUTO METROPOLITANO DE EDUCACIÓN
PROGRAMACIÓN DE COMPUTADORES
GUIA #1 DE VISUAL FOXPRO
DOCENTE: MAURICIO CANO

REPASO MANEJO BÁSICO DE TABLAS

a) La empresa Instituto Metropolitano de Educación, requiere el diseño de una base de datos que le permita almacenar la siguiente información:

1. Maestro de Programas: programas de estudio
2. Maestro de asignaturas: asignaturas de estudio.
3. Hojas de vida de los estudiantes: información básica de los estudiantes.
4. Información de matrícula: información de matrícula semestral.

Se requiere diseñar la estructura de las 4 tablas tal y como se indica a continuación:

1. Maestro de Programas		
Nombre Campo	*Tipo de datos*	*Ancho*
Codigo	Carácter	2
Nombre	Carácter	50
Campo clave ascendente: Codigo		

2. Maestro de Asignaturas		
Nombre Campo	*Tipo de datos*	*Ancho*
Codigo	Carácter	10
Nombre	Carácter	50
Programa	Carácter	2
Campo clave ascendente: Codigo		

3. Hojas de vida		
Nombre Campo	*Tipo de datos*	*Ancho*
Codigo	Carácter	10
Nombre	Carácter	50
Apellidos	Carácter	50
Programa	Carácter	2
TelResidencia	Carácter	20
DirResidencia	Carácter	40
FechaNacimto	Fecha	8
Campo clave ascendente: Codigo		

4. Matrícula		
Nombre Campo	*Tipo de datos*	*Ancho*
Codigo	Carácter	10
Fecha	Fecha	8
Nivel	Carácter	2
Jornada	Carácter	1
Estado	Carácter	1
Observación	Memo	10
Campo clave ascendente: Codigo		

b) Genere los campos claves indicados en cada estructura.
c) Introduzca información dentro de cada tabal. Mínimo cinco registros por tabla. Trate de que la información sea ceñida a la realidad.
d) Genere los formularios para automatizar procesos con las 4 tablas.
e) Genere cuatro consultas. Una para cada tabla.
f) Ejecute los comandos de la hoja Resumen de Comandos dados por el docente. Abra manualmente las tablas y ejecute los comandos. Esto con el objetivo de memorizar órdenes que durante las tareas de programación, se hace muy común su uso.

Nota: *La programación es un arte que se logra mediante la práctica. La memoria juega un papel importante a la hora de escribir código de programas. Una de las ventajas de los nuevos lenguajes Visuales, es el de esconder un poco el código, pero eso no significa que no sea necesario aprender un poco de las órdenes que caracterizan a cada lenguaje.*

REPASO COMANDOS DEL FOXPRO

COMANDOS PARA MANIPULAR TABLAS

CREATE <Tabla>	Crear una tabla.
USE <Tabla>	Abrir una tabla.
USE <Tabla> ALIAS <Alias>	Abre una tabla y selecciona el área de trabajo en la cual se desea trabajar.
USE	Cierra cualquier tabla abierta en el área de trabajo activa
MODIFY STRUCTURE	Modificar la estructura de la tabla abierta.
CLOSE ALL	Cerrar todos los ficheros abiertos en el sistema FoxPro.
SELECT <*N*>	Selecciona el área *N* de trabajo

COMANDOS PARA MANIPULAR REGISTROS

APPEND	Agregar registros a la tabla abierta
APPEND BLANK	Agregar un registro en blanco a la tabla abierta
SCATTER MEMVAR	Llevar el registro activo a un arreglo de memoria
GATHER MEMVAR	Llevar un arreglo de memoria al registro activo
DELETE	Borrar el registro activo
DELETE ALL	Borrar todos los registros de la tabla abierta
DELETE ALL FOR <Condición>	Borrar todos los registros que cumplan con la condición.
RECALL	Recuperar o retirar la marca de borrado lógica de un registro
RECALL ALL	Recuperar todos los registros con la marca de borrado lógica.
RECALL ALL FOR <Condición>	Recuperar todos los registros que cumplan con la condición.

COMANDOS PARA MOVERSE ENTRE REGISTROS

GO TOP	Ir al primer registro de la tabla abierta
GO BOTTOM	Ir al ultimo registro de la tabla abierta.
SKIP +<N>	Mueve el puntero de los registros +N posiciones (Avanzar)
SKIP -<N>	Mueve el puntero de los registros –N posiciones (Retroceder)

COMANDOS PARA TRABAJAR ÍNDICES COMPUESTOS

USE <Tabla> ORDER TAG <Índice>	Abre la tabla indicada y selecciona el *<Índice>* como clasificación de los registros, lo indicado por *Índice*.
SET ORDER TO TAG <Índice>	Selecciona el índice como orden para los registros.
SET ORDER TO	Cierra cualquier fichero índice activado para una tabla.
SEEK <Expresión>	Busca un dato en un campo clave.
FOUND()	Función utilizada para verificar el éxito de una búsqueda.
INDEX ON <Campo Clave> TAG <Nombre Tag> FOR <Condición>	Genera un fichero índice compuesto basado en el campo clave y utiliza como filtrado de datos la *<Condición>* especificada.

CONSULTAS COMPLEMENTO DE LAS CLASES

Un programador debe consumir y devorar gran cantidad de documentación. En una clase es imposible darle todas las técnicas disponibles a un estudiante. El estudiante debe poner mucho de su parte y documentarse acerca de los temas vistos en clase. Se plantean las siguientes consultas para que apoyemos como estudiante, una buena labor por parte del docente.

- Investigar los comandos SET
- Investigar las funciones SYS()
- Que son las áreas de trabajo? Cuales son las ordenes básicas para manipular áreas de trabajo? Como se aplican dentro de la programación de Visual FoxPro?
- La Programación orientada a objetos en el Visual Foxpro. Como funciona? Que son las clases? Como se programan las Clases? Que es la clase wizstyle?
- Que es el lenguaje SQL? Cuales son las ordenes básicas del lenguaje SQL? Como se aplican dentro del FoxPro?

COMPLEMENTO GUÍA DE CLASES #1

LAS ÁREAS DE TRABAJO

Cuando una aplicación requiere más de una tabla para poder funcionar, se hace necesario el trabajar las áreas de trabajo. Aunque es bien sabido, con las recientes versiones del Visual Foxpro y su maravilloso entorno de programación, se hace innecesaria utilizar las órdenes que permiten controlar las áreas de trabajo. El mismo entorno y estilo de programación del Visual Foxpro ocultan el funcionamiento de las áreas de trabajo y dejan todo el control a la opción "*entorno de datos*".

Sin embargo, es muy útil el conocer los comandos que permiten el manejo de las áreas de trabajo.

a. USE / ALIAS

Permite abrir una tabla y asignar un alisa o nombre interno para referencias subsecuentes. Muchos diseños de aplicaciones requieren de largos y complejos nombres en las tablas. El asignar un alias resulta útil a la hora de hacer referencia en forma fácil dentro del código de la aplicación.

Sintaxis: USE <*Tabla*> ORDER TAG <*Orden*> ALIAS <*Id Alias*>

El Id Alias es un nombre corto para hacer referencia a la tabla en órdenes subsecuentes a su ejecución. Si no se utiliza la opción ALIAS, el sistema asignará automáticamente como alias el mismo nombre de la tabla.

Ver el ejemplo siguiente para comprender el uso de esta orden.

b. SELECT *n*

Activa el área de trabajo indicada por la *n*. Un área de trabajo es un espacio asignado en memoria para la tabla que se abra inmediatamente después de ejecutada la orden. FoxPro tiene gran capacidad para asignar áreas, así que si la aplicación requiere de muchas tablas simultáneamente abiertas en memoria, no vamos a tener ningún problema.

Para designar el área de trabajo podemos utilizar número del 1 hasta donde lleguemos en cantidad de tablas. O podemos utilizar letras para hacer referencias a las áreas, iniciando por la letra A. Existe un identificador especial para nombrar áreas y es el cero (0). Este identificador le indica al FoxPro que utilice el área máxima disponible para áreas de trabajo. Esta referencia es bastante útil para utilizar en rutinas subsecuentes y distantes de la rutina inicial cuando se abrieron todas las tablas requeridas.

Una vez se selecciona el área de trabajo y se abre una tabla, para hacer referencia a la tabla, basta con aplicar la orden SELECT acompañada del área programada.

Analicemos el siguiente código:

1	*SELECT 1* *USE PROGRAMAS ORDER TAG CODIGO ALIAS PROG*	Activar el área 1 o A Abrir la tabla programas y utilizar como alias PROG.
2	*SELECT 2* **USE ASIGNA ORDER TAG CODIGO**	Activar el área 2 o B. Abrir la tabla ASIGNA. El alias automáticamente de esta tabla es ASIGNA, el mismo nombre de la tabla.
3	*SELECT C* *USE MATRIC ORDER TAG CODIGO*	Activar el área 3 o C. Abrir la tabla MATRIC. El alias automáticamente de esta tabla es MATRIC, el mismo nombre de la tabla.
4	*SELECT PROG* *DELE ALL*	Selecciona y activa el área 1. Se utiliza el mismo alias asignado a la tabla. Borra todos los registros de la tabla PROGRAMAS
5	*SELECT B* *LIST*	Selecciona y activa el área 2. Lista el contenido de la tabla ASIGNA.
6	*SELECT 3* *BROWSE*	Selecciona y activa el área 3. Ejecuta la ventana Examinar en la tabla MATRIC.

Si ejecutamos la orden:

```
SELECT 0
USE PROFESOR ORDER TAG CODIGO ALIAS PROFE
```

Se utilizaría como área de trabajo la 4 o D. Como alias asignado a la tabla, quedaría PROFE. El comando SELECT 0 es muy útil cuando se desea abrir una tabla de forma temporal, para luego volver a la tabla que estaba abierta cuando se invoco. Por ejemplo, la siguiente sección de código, guarda en una variable el área seleccionada, abre una tabla en el área máxima disponible, examinar la información, la cierra y regresa al área inicial de trabajo.

```
mAlias = alias()
mFilter = filter()

SELECT 0
USE NOTAS ORDER TAG CODIGO
BROWSE
USE

SELECT mAlias
SET FILTER TO FILTER
```

Nota: Para cerrar todas las tablas abiertas, puede utilizar CLOSE ALL. Si solo desea cerrar la tabla de un área en especial, utilice la orden USE. Por ejemplo, de acuerdo al código mostrado hace unos momentos, si desea cerrar la tabla ASIGNA ejecute las siguientes ordenes:

```
SELE 2
USE
```

INSTITUTO METROPOLITANO DE EDUCACIÓN
PROGRAMACIÓN DE COMPUTADORES
GUIA #2 DE VISUAL FOXPRO
DOCENTE: MAURICIO CANO

EL ACCESO A UNA APLICACIÓN

Lo normal en cualquier aplicación, es que una vez se carguen en el computador, nos reciban con una pantalla de acceso, solicitándonos datos de identificación tales como *Código de la Compañía, Nombre de Usuario* y *Contraseña del Usuario*. Es por esta razón, que se plantea como primer tema de clases, el diseñar una pantalla que permita controlar el acceso de los usuarios e internamente, enseñarle al sistema en que lugar del computador reside la información.

En esta pantalla se maneja internamente variables públicas que se utilizarán en cuanto formulario se diseñe dentro de la aplicación. La variable pública *_CodCia* incluye el código de la compañía ingresada, *_NomCia* el nombre de la empresa (usada en los reportes), *_DirCia* es el directorio de datos de la compañía, *_CodUsua* es el código del usuario conectado al sistema. En la guía de clases #3 aparece un tip de programación que permite agilizar el diseño de aplicaciones que dependan de variables de configuración tal como las anteriores.

1: mCodigo
2: mUsuario
3: mClave

4: btnAceptar
5: btnCancelar

BtnAceptar.click

```
*verificar información entrada por el usuario
select cias
_CodCia = alltrim( thisform.mCodigo.value)
seek _CodCia
if found()
        *verificar existencia tabla de usuarios
         mTabla = alltrim( cias.directorio) +"usuarios.dbf"
         if !file( mtabla)
              wait window"No existen Usuarios" nowait
               release thisform
               cancel
         endif

         *abrir tabla de usuarios de la cia entrada
         sele 2
         use &mTabla order tag codigo
         _CodUsua = alltrim (thisform.mUsuario.value)
         seek _CodUsua
         if found()
                 *verificar clave
                 mclave = alltrim( thisform.mclave.value)
                 if alltrim (usuarios.clave)=mclave
                    *dejar entrar
                    wait window"Puede entrar el Sistema" nowait
                    release thisform
                 else
                     wait window "Clave Incorrecta" nowait
                 endif
          else
                 *no existe usuario
                 wait window "Usuario no Existe" nowait
           endif
else
         wait window"Compañia no Existe" nowait
endif
```

Formulario.Init	**BtnCancelar.Click**
set echo off set talk off set date to mdy *enrutar al directorio base de la aplicación set default to c:\clasefox public _CodCia, _NomCia, _DirCia, _CodUsua _CodCia = space(2) _NomCia = space(50) _DirCia = space(100) _CodUsua = space(10) *tabla de las compañías select 1 use cias order tag codigo	close all release thisform

NOTA: La técnica empleada para esta pantalla, es la de diseñar aplicaciones independientes de los datos. Muchos programdores "amarran" sus aplicaciones a las bases de datos. Los sistemas tienden a crecer día a día, puede que inicialmente diseñemos una aplicación con directorio de almacenamiento local (en el mismo disco duro), y cuando menos lo pensamos, se nos hace la petición de que se requiere almacenar las bases de datos en un servidor (algo normal en una empresa).

Debemos diseñar aplicaciones que reconozcan el origen de los datos y que internamente en una sencilla operación, se dirijan al sitio de los datos y los utilicen en forma transparente para el usuario.

La rutina planteada, permite que el sistema tome la información de instalación de los datos de la tabla de configuración empresa (cias.dbf), capture el campo directorio y se "*enrute*" hacia ese sitio para leer los datos de los usuarios.

Observe que en la parte en donde se permite el acceso al sistema, no existe aún una instrucción que continúe la ejecución normal de la aplicación. Esta será explicado en otra guía de clases. Por el momento lo importante es captar la forma en que se debe diseñar una aplicación inteligente, con capacidad de hallar el origen de sus datos, bien sea que estén en el mismo disco duro, o en otro lugar (servidor o estación) en una red local.

RETO DE PROGRAMACIÓN

Trate de mejorar esta rutina colocando controles que le permitan al usuario ejecutar solo tres intentos de acceder al sistema.

COMPLEMENTO A LA GUÍA #2

LA BÚSQUEDA DE INFORMACIÓN EN TABLAS INDEXADAS

La búsqueda de información permite tener mayor control sobre los datos almacenados en las tablas. Para poder buscar información, es importante saber manipular las órdenes de indexados, activación de índices, ordenes de búsqueda y funciones para el chequeo de la efectividad de las búsquedas. FoxPro trae muchas órdenes suficientes para que nuestras aplicaciones tengan control de la información clave de nuestras tablas.

- **USE *Tabla* ORDER TAG *NombreTag***

  Abre la tabla indicada y activa como índice el especificado por *NombreTag*. El éxito de ejecución de esta orden requiere que la tabla haya sido previamente indexada.

  Por ejemplo, esta instrucción abre la tabla PROGRAMA y activa CODIGO como orden de la tabla. Ahora, podremos listar la información de la tabla clasificada por este campo o podremos ejecutar búsquedas de datos basado en este campo clave.

  USE PROGRAMA ORDER TAG CODIGO

- **SET ORDER TO TAG *NombreTag***

  Permite activar el campo índice indicado por *NombreTag*. Por ejemplo, la siguiente orden abre la tabla PROGRAMA en una instrucción aparte, luego ejecuta SET ORDER TO TAG para establecer el campo clave a utilizar para el ordenado de los datos o ejecución de búsquedas:

  USE PROGRAMA
  SET ORDER TO TAG CODIGO

  En este caso, es mejor utilizar una línea para abrir la tabla y simultáneamente asignar el campo de ordenamiento. Esta orden resulta muy útil, cuando se necesita establecer un orden para la tabla, sin necesidad de volverla a abrir.

  Por ejemplo, abrimos la tabla PROGRAMA e inmediatamente asignamos la clasificación de los datos.

  USE PROGRAMA ORDER TAG CODIGO

  Si ejecutamos la siguiente orden, se listaría la información clasificada por el la entrada clave CODIGO.

  LIST

  Ahora, activaremos otro campo de ordenamiento y ejecutaremos nuevamente LIST. Esta vez se lista la información clasificada por el campo NOMBRE.

  SET ORDER TO TAG NOMBRE
  LIST

- **INDEX ON *ExpIndice* TAG *NombreTag* FOR !DELETED()**

  Esta orden permite generar una entrada especial de índice compuesto, clasificando la información por la expresión de índice *ExpIndice*. EL nombre de la entrada sería el indicado por *NombreTag*. La cláusula FOR permite establecer un filtro para los datos clasificados. En este caso, se utiliza como filtrado de datos, a todos los registros que no se hallan borrado de la tabla. Acostúmbrese a utilizar esta cláusula en cuanto indexado construya. Esta instrucción le permitirá tener mayor control de los datos existentes en una tabla, y su respectivo manejo a través de los formularios (recuerde que cuando se retira un registro con la orden DELETE este no se borra físicamente -borrado lógico- de la tabla hasta no ejecutar PACK –borrado físico-). Si no utiliza esta cláusula, los informes, y muchas órdenes de recorrido de registros, incluirían a los borrados de las tablas.

  Por ejemplo, la siguiente instrucción crea una entrada índice denominada CODIGO para la tabla PROGRAMA. Adicionalmente, con la cláusula FOR se filtran todos los borrados de la tabla. En la tercer orden, se crea una entrada clave denominada NOMBRE.

  USE PROGRAMA
  INDEX ON CODIGO TAG CODIGO FOR ¡DELETED()
  INDEX ON NOMBRE TAG NOMBRE FOR !DELETED()

  Nota: Tenga en cuenta que esta órden crea inicialmente un fichero compuesto con el mismo nombre de la tabla y con la extensión .CDX. Para el ejemplo anterior, si miramos el sitio donde se encuentra almacenada la tabla *PROGRAMA.DBF* aparece un nuevo archivo denominado *PROGRAMA.CDX*. Las siguientes entradas índices se agregarán al índice compuesto.

  Expresiones de índices compuestos

  Muchas veces una aplicación requiere la construcción de índices compuestos para poder funcionar. Por ejemplo, el detalle de las notas de unos estudiantes, podría ser un claro ejemplo de ello. La tabla de notas (por diseño del software) contiene solo el campo CODIGO del estudiante y el campo CODASIGNA que incluye el código de las asignaturas.

  USE NOTAS
  INDEX ON CODIGO +CODASIGNA TAG CODASIGNA FOR ¡DELETED()

  La clase de Ingeniería de Software le enseñará el momento preciso en que se requieren índices compuestos para una aplicación.

  Esta clase de índices también se requieren a veces para listar información referente a una misma clave. Por ejemplo, si se tiene la tabla ESTGRUPO para almacenar los estudiantes en los grupos de clase, con un campo de CODGRP para el código de los grupos y otro campo denominado CODEST para guardar el código de los estudiantes, la instrucción para crear el campo clave compuesto es la siguiente:

  USE ESTGRP
  INDEX ON CODGRP +CODEST TAG CODEST FOR ¡DELETED()

  Esta forma especial de definir campos claves involucrando a más de un campo, debe considerarse al momento de ejecutar las búsquedas de información en tales tablas. El sistema FoxPro crea un solo campo clave, donde almacena la expresión resultante para clasificar la información de la tabla.

Los campos involucrados en una expresión compuesta deben ser del mismo tipo de datos. Por ejemplo, si CODGRP es carácter, entonces CODEST también debe serlo. Si el diseño de la tabla indica que se deben involucrar campos con diverso tipo de datos, entonces se requiere la aplicación de funciones para la conversión de los datos.

Funciones para conversión de datos

Las funciones más utilizadas para formar expresiones de campos claves compuestos son las siguientes:

STR(ExpresiónNumérica) : convierte la expresión numérica a cadena de caracteres.

VAL(ExpresiónC aracter) : convierte la expresión de caracteres a un valor numérico. La expresión aunque sea de tipo carácter, debe tener forma o estructura numérica.

DTOC(ExpresiónFecha): convierte la expresión fecha a una cadena o expresión de tipo carácter.

CTOD(ExpresiónFecha): convierte una expresión tipo fecha a una cadena de caracteres.

Tenga presente que la expresión de clave debe tener el mismo tipo de datos. Para saber que funciones utilizar para armar la expresión, analice los tipos de datos y la estuctura o forma de los campos onvolucrados y aplique las funciones para armar una expesión unificada de datos. Por ejemplo, si analiza los datos de un campo, y detecta uno de tipo carácter y otro de cualquier tipo, la clave es llevarlos a una expresión de tipo carácter.

Por ejemplo, si se tiene una tabla de KARDEX y en ella tenemos unos campos CODREF para los códigos de los artículos y otro campo FECHA para almacenar la fecha en que se registro una referencia en el kardex. La forma de definir el índice es el siguiente:

```
USE KARDEX
INDEX ON CODREF +DTOC( FECHA) TAG CODFECHA FOR !DELETED()
```

Debido a que el campo CODREF es carácter y FECHA es de tipo fecha, mejor se llevó la conversión a cadena de caracteres. La experiencia le enseñará a aplicar en forma adecuada la conversión de datos y la generación de expresiones para índices compuestos.

BUSCAR INFORMACIÓN EN UNA TABLA INDEXADA

Una de las aplicaciones de los índices es la de permitir la búsqueda de información en las tablas. Además, el uso de funciones que permiten analizar si una búsqueda fue exitosa o no.

Estas son las órdenes y funciones que permiten manipular búsquedas en campos claves activados (ver la orden *SET ORDER TO*).

- **SEEK Expresión_a_buscar**

  Ejecuta una búsqueda en el campo clave, utilizando como criterio de búsqueda la *Expresión_a_buscar*.

La *Expresión_a_buscar* debe ser del mismo tipo de datos de la expresión clave generada al momento de indexar. Si ocurre lo contrario, FoxPro generará un error de tipos de datos incompatibles. Esta expresión puede ser una variable. Tenga cuidado al definir variables para buscar información, por ejemplo, si define la variables CODIGO y resulta que en su tabla existe un campo llamado CODIGO, el sistema, simplemente no busca nada. Para evitar esto, agregue identificadores a los nombres de las variables para identificarlos de los nombres de los campo, por ejemplo, agregue una *m* a cuanta variable defina: mCODIGO o simplemente utilice el indicador de variable de memoria (*.m*) del FoxPro *m.codigo*.

En este ejemplo, se busca el dato almacenado en la variable mCodigo, en el campo clave CODIGO de la tabla PROGRAMA.

```
SELECT PROGRAMA
SEEK mCodigo
```

La orden Seek mueve automáticamente el puntero al registro hallado. Lo que significa que después de ejecutada tal orden, basta activar el área de trabajo de la tabla y ejecutar la orden deseada sobre el registro activo.

- **FOUND() :** Esta función retorna el estado de éxito de la orden buscar recién ejecutada. Retorna .t. si la búsqueda fue exitosa. Retorna .f. en caso contrario. Utilice una estructura de control de flujo para determinar las acciones a tomar acorde al estado de la búsqueda.

  Si a la secuencia de ordenes anteriores agregamos la siguiente, se obtiene un control de lo que deseamos hacer si se halló algo o no.

```
IF FOUND()
    WAIT WINDOW "SE HALLÓ ALGO..." NOWAIT
ELSE
    WAIT WINDOW "NO SE HALLÓ NADA..." NOWAIT
ENDIF
```

  Nota: como caso curioso del FoxPro, muchas veces cuando se coloca algo de por medio entre la orden SEEK y la función FOUND() el sistema muestra resultados inesperados, como si se le olvidase el resultado de la búsqueda inmediatamente ejecutado el chequeo. Acostúmbrese a colocar el chequeo del resultado -IF FOUND()-, inmediatamente después del SEEK.

- **SEEK(*mExpresion_a_buscar*, *Alias*) :** Función que permite buscar un registro y retornar si se halló algo o no (más o menos, SEEK más FOUND() combinadas). Esta función resulta útil para utilizar en sitios donde no se puede colocar la orden SEEK más el chequeo con FOUND(), tales como reportes, formularios, etc. Esta función no mueve el puntero al registro hallado, simplemente se limita a afirmar si el dato está o no en la tabla y campo clave seleccionado.

- **LOCATE FOR *Expresión_de_Condición***

  Ejecuta una búsqueda secuencial, revisando registro a registro. Esta orden se puede utilizar para buscar información en tablas que no están indexadas. Esta orden permite un rango mayor de búsqueda, al permitir establecer la condición de búsqueda.

  Ejemplo 1:

  LOCATE FOR APELLIDOS = "CANO" .AND. NOMBRE = "MAURICIO"
  CONTINUE

  Este ejemplo, busca la primer ocurrencia de datos de la condición apellidos igual a "CANO" y nombre correspondiente a "MAURICIO". La instrucción CONTINUE le indica al FoxPro que continúe con la búsqueda recién iniciada.

  Ejemplo 2:

  La posibilidad de aceptar condiciones como parámetro de búsqueda, convierte a esta sencilla instrucción en potente comando de programación, que bien utilizado, puede salvarnos el día.

  LOCATE FOR "MAURICIO" $ NOMBRE
  CONTINUE

  En este ejemplo, se buscarán todos los nombres iguales a "MAURICIO".

  Comentarios

  La tabla no necesita estar indexada.

  Si *LOCATE* encuentra un registro coincidente, podrá utilizar *RECNO( )* para devolver el número del registro coincidente. Si se encuentra un registro coincidente, *FOUND( )* devolverá verdadero (.T.) y EOF( ) devolverá falso (.F.).

  Después de que *LOCATE* encuentre un registro coincidente, puede emitir *CONTINUE* para buscar registros coincidentes adicionales en el resto de la tabla. Cuando se ejecuta *CONTINUE*, se reanuda el proceso de búsqueda, empezando por el registro que sigue inmediatamente al registro coincidente. Puede emitir *CONTINUE* repetidamente hasta llegar al final del alcance o de la tabla.

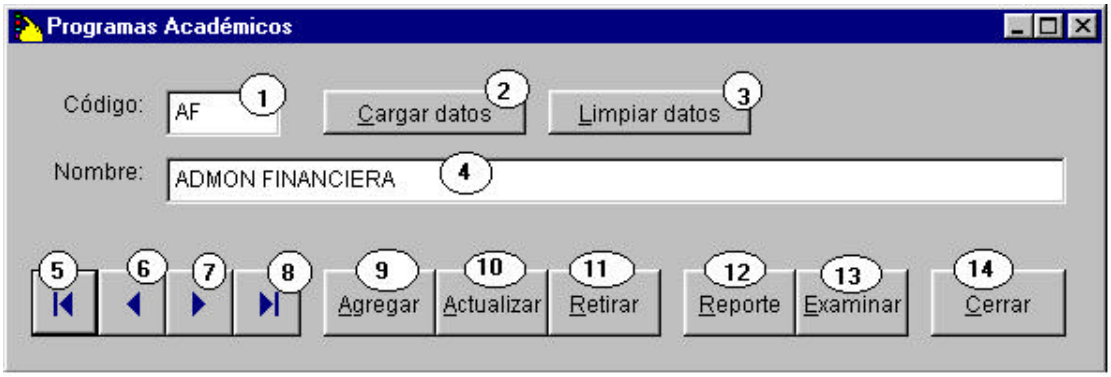
  Si no se encuentra ninguna coincidencia, *RECNO( )* devolverá el número de registros de la tabla más uno, *FOUND( )* devolverá falso (.F.) y *EOF( )* devolverá verdadero (.T.).

  LOCATE y *CONTINUE* son específicos del área de trabajo actual. Si se selecciona otro área de trabajo, el proceso de búsqueda original podrá continuarse cuando se vuelva a seleccionar el área de trabajo original.

INSTITUTO METROPOLITANO DE EDUCACIÓN
PROGRAMACIÓN DE COMPUTADORES
GUIA #3 DE VISUAL FOXPRO
DOCENTE: MAURICIO CANO

Las técnicas de programación varían de un programador a otro. Cada programador asume la técnica que desee para desarrollar sus aplicaciones. Sea cual sea la técnica empleada, lo más importante es que sea funcional.

Se propone la siguiente técnica de programación diseñada por nuestra dependencia, muy útil para el trabajo en línea. Permitiendo mayor velocidad al entrar, buscar , retirar y consultar datos. Cada grupo de estudio deberá elaborar los diseños de las otras tablas diseñadas y propuestas por la guía de clases #1. Cada grupo deberá experimentar con el método de trabajo propuesto y hallar las posibles fallas técnicas de diseño (algo normal en grandes aplicaciones).



1: mCodigo	2: btnCargar	3: btnLimpiar	4: mNombre
5: btnPrimero	6: btnAnterior	7: btnSiguiente	8: btnUltimo
9: btnAgregar	10: btnActual	11: btnRetirar	12: btnReporte
13: btnExaminar	14: btnCerrar		

BtnPrimero.click

```
*ir al primero
wait window "Primer registro..." nowait
sele 1
go top
scatter memvar memo
thisform.refresh
return
```

BtnAnterior.Click

```
*ir al anterior
sele 1
if !bof()
        skip -1
endif
if bof()
        go top
endif
scatter memvar memo
thisform.refresh
return
```

BtnSiguiente.Click

```
*ir al siguiente
sele 1
if !eof()
        skip 1
endif
if eof()
        go bottom
endif
scatter memvar memo
thisform.refresh
return
```

BtnUltimo.Click

```
*ir al ultimo
wait window "Último registro..." nowait
sele 1
go bottom
scatter memvar memo
thisform.refresh
return
```

BtnAgregar.Click

```
*agregar registro con los datos en pantalla
sele 1

*verificar si hay datos en blanco
if empty( thisform.mCodigo.value)
	wait window "Código incorrecto..." nowait
	thisform.mCodigo.setfocus
	return
endif

if empty( thisform.mNombre.value)
	wait window "Nombre incorrecto..." nowait
	thisform.mNombre.setfocus
	return
endif

*verificar campo clave
mCodigo = alltrim( thisform.mCodigo.value)
sele 1
seek mCodigo
if found()
	mMens1 = "Imposible agregar, el código " +mCodigo
+" ya existe!" +chr(13)
	mMens2 = "Utilice otro código e intente de nuevo..."
		mOpc = messagebox( mMens1 +mMens2,
0+64, "Atención!")
	thisform.mCodigo.setfocus
	return
endif

*agregar nuevo registro con los datos en pantalla
sele 1
insert into programa from memvar

*cargar nuevos datos en blanco
scatter memvar blank

*ir al primer campo
thisform.mCodigo.setfocus

thisform.refresh
```

BtnActual.Click

```
*actualizar el contenido de un campo

*verificar si hay datos en blanco
if empty( thisform.mCodigo.value)
	wait window "Código incorrecto..." nowait
	thisform.mCodigo.setfocus
	return
endif

if empty( thisform.mNombre.value)
	wait window "Nombre incorrecto..." nowait
	thisform.mNombre.setfocus
	return
endif

*actualizar datos del campo clave
sele 1
mCodigo = alltrim( thisform.mCodigo.value)
seek mCodigo
if found()  && actualizar registro hallado
	gather memvar memo
	wait window "Datos actualizados..." nowait
else
	mMens1 = "Imposible actualizar, el código "
+mCodigo +" no existe!" +chr(13)
	mMens2 = "Utilice otro código e intente actualizar de
nuevo..."
	mOpc = messagebox( mMens1 +mMens2, 0+64,
"Atención!")
	thisform.mCodigo.setfocus
	return
endif

return
```

BtnExaminar.Click

```
*examinar registros
sele 1
browse fields codigo, nombre noedit noappend nodelete

*activar el seleccionado
scatter memvar memo

thisform.refresh
return
```

BtnLimpiar.Click

```
*limpiar el contenido de los campos
wait window "Datos en blanco..." nowait
scatter memvar memo blank
thisform.refresh
```

BtnRetirar.Click

```
*retirar el registro clave hallado
sele 1
mCodigo = alltrim( thisform.mCodigo.value)
seek mCodigo
if !found()  && no existe codigo
	mMens1 = "Imposible retirar, el código "
+mCodigo +" no existe!" +chr(13)
	mMens2 = "Utilice otro código e intente retirar
de nuevo..."
	mOpc = messagebox( mMens1 +mMens2, 0+64,
"Atención!")
	thisform.mCodigo.setfocus
	return
endif

mMens1 = "Desea retirar el código " + mCodigo +" del
sistema!" +chr(13)
mOpc = messagebox( mMens1 , 1+32, "Retirar
registro!")

if mOpc = 1 && Clic en botón Aceptar
	sele 1
	delete
	wait window "Registro retirado del sistema..."
nowait

	*mover puntero al próximo disponible
	if !eof()
		skip 1
	endif
	if eof()
		go bottom
	endif

endif

scatter memvar memo
thisform.refresh
return
```

BtnCargar.Click

```
*cargar el contenido de los campos con la clave entrada

*verificar si hay datos en blanco
if empty( thisform.mCodigo.value)
	wait window "Código incorrecto..." nowait
		thisform.mCodigo.setfocus
	return
endif

*cargar datos del campo clave
sele 1
mCodigo = alltrim( thisform.mCodigo.value)
seek mCodigo
if found()  && actualizar registro hallado
	scatter memvar memo
	wait window "Datos cargados..." nowait
else
	mMens1 = "Imposible cargar datos, el código "
+mCodigo +" no existe!" +chr(13)
	mMens2 = "Utilice otro código e intente cargar
datos de nuevo..."
	mOpc = messagebox( mMens1 +mMens2, 0+64,
"Atención!")
endif

thisform.refresh
thisform.mCodigo.setfocus

return
```

BtnCerrar.Click

```
thisform.release
```

Form.Init

```
*abrir entorno utilizando las variables públicas

mFile1 = _dircia +"programa.dbf"
if !file( mFile1)
	do cPrograma in prg\creartbl
endif

sele 1
use &mFile1 order tag codigo alias programa

scatter memvar memo

*ir al objeto inicial
thisform.mCodigo.setfocus
return
```

Rutinas adicionales para que este diseño funcione:

Se requiere una rutina que defina las tablas en forma automática. Esta rutina dará un grado de inteligencia al sistema, al permitirle definir las tablas que no existan, en forma automática. Un sistema que incluya tal funcionalidad, simplemente, nunca fallará por falta de datos.

CREATBL.PRG

Esta rutina debe crearse en el generador del proyecto en *Codigo/Programas*. Defina la carpeta *prg* para almacenar las rutinas que definamos dentro de la carpeta del proyecto.

```
*creatbl.prg
*   Definir las tablas del sistema
*   invocar: do cPrograma in prg\creatbl

procedure cPrograma
        *chequear y definir tabla de asignaturas

        *definir una cadena de tal forma "c:\clasefox\01\programa.dbf"
        mTabla = _dircia + "programa.dbf"

        if !file( mTabla)
                create table &mTabla  ;
                        ( codigo   c ( 10), ;
                        nombre   c ( 50))
                index on codigo tag codigo for !deleted()
                index on nombre tag nombre for !deleted()
                close data
                wait window "Tabla "+mTabla +" definida..." nowait
        endif
return   && cPrograma

*<eof> creatbl.prg
```

TIP DE PROGRAMACIÓN

Es muy útil el definir una rutina independiente denominada public con todas las variables públicas a utilizar durante el diseño del sistema. Con esto nos ahorramos tiempo al no tener que hacer todo el recorrido de la aplicación para verificar el funcionamiento de una aplicación.

```
*public.prg
*  utilizar solo durante diseño de programas
*  los datos en este programa deben ser inicializados por
*  la pantalla de entrada al sistema

public _codcia, _nomcia, _dircia, _codusua
_codcia  = "01"
_nomcia  = "COLEGIO 01"
_dircia  = "c:\clasefox\01\"
_codusua = "01"
*<eof> public.prg
```

INSTITUTO METROPOLITANO DE EDUCACIÓN
PROGRAMACIÓN DE COMPUTADORES
GUIA #4 DE VISUAL FOXPRO
DOCENTE: MAURICIO CANO

Continuando con la técnica #1, ha llegado el momento de demostrar las habilidades que como programadores, hemos desarrollado hasta el momento. Gran parte del arte de la programación, se adquiere imitando modelos hechos por otros programadores. Si como programadores, desarrollamos rutinas que pueden ser perfectamente reutilizables en otras partes de la aplicación, lo normal es tomar el modelo base y copiarlo tantas veces lo necesitemos. Más adelante aprenderemos que cuando se requiere hacer esto muy a menudo, lo más práctico es definir una clase y basar todas nuestras pantallas en ella.

EL FORMULARIO PARA EL MAESTRO DE ASIGNATURAS

Para la elaboración del siguiente formulario abra el formulario de la guía #3 y grábelo (*Menú Archivo/Salvar Como*) con otro nombre (*frmasigna*) en la carpeta donde residen los formularios (*c:\clasefox\documentos*). Recuerde agregarlo al proyecto en la ventana donde aparecen listados los formularios (*Ventana proyecto, clic en botón Agregar*).

Lo primero que haremos ahora, es abrir el formulario recién agregado, abrirlo en la ventana de diseño y modificar el *Caption* del formulario, para que diga "*Maestro de Asignaturas*". Esto es importante, ya que el tener correctamente identificadas las pantallas, nos puede llegar a ayudar a ubicarnos rápidamente en la pantalla de trabajo, y evitar posibles dolores de cabeza al sobrescribir código de pantallas ya terminadas.

Reorganice la pantalla original de programas (la recién copiada) para que incluya el campo *programa* de la tabla *ASIGNA.DBF*.

1: mPrograma	2: btnAgregar	3: btnActual	4: btnExaminar

Todos los controles continúan con los mismos nombres. El nuevo objeto de campo es el referenciado con el numeral 1 (mPrograma). Los objetos que aparecen referenciados con números (del 1 al 4) indican que el código del evento clic presenta variaciones. Los demás objetos, continúan con el mismo código programado en la lección anterior.

Ahora lo que debemos hacer, es personalizar el formulario para que apunte a la tabla en cuestión, en nuestro caso, *Asigna.dbf*. Debe alterar el código del evento *INIT* del formulario, y modificar el código para que diseñe automáticamente la tabla *Asigna* y abra simultáneamente la tabla de programa (*utilizando áreas de trabajo*).

BtnAgregar.Clic

```
*agregar registro con los datos en pantalla
sele 1

*verificar si hay datos en blanco
if empty( thisform.mCodigo.value)
        wait window "Código incorrecto..." nowait
        thisform.mCodigo.setfocus
        return
endif

if empty( thisform.mNombre.value)
        wait window "Nombre incorrecto..." nowait
        thisform.mNombre.setfocus
        return
endif

if empty( thisform.mPrograma.value)
        wait window "Programaincorrecto..." nowait
        thisform.mPrograma.setfocus
        return
endif

*verificar campo clave
mCodigo = alltrim( thisform.mCodigo.value)
sele 1
seek mCodigo
if found()
        mMens1 = "Imposible agregar, el código " +mCodigo
+" ya existe!" +chr(13)
        mMens2 = "Utilice otro código e intente de nuevo..."
        mOpc = messagebox( mMens1 +mMens2, 0+64,
"Atención!")
        thisform.mCodigo.setfocus
        return
endif

*validar el programa entrado por el usuario
mPrograma = alltrim( thisform.mPrograma.value)
sele programa
seek mPrograma
if !found()
        mMens1 = "El código de Programa "
+mPrograma +" no existe!" +chr(13)
        mMens2 = "Utilice un código válido e intente de
nuevo..."
        mOpc = messagebox( mMens1 +mMens2, 0+64,
"Atención!")
        thisform.mPrograma.setfocus
        return
endif

*agregar nuevo registro con los datos en pantalla
sele 1
insert into asigna from memvar

*cargar nuevos datos en blanco
scatter memvar blank

*ir al primer campo
thisform.mCodigo.setfocus

thisform.refresh
```

BtnActual.Click

```
*actualizar el contenido de un campo

*verificar si hay datos en blanco
if empty( thisform.mCodigo.value)
   wait window "Código incorrecto..." nowait
    thisform.mCodigo.setfocus
    return
endif

if empty( thisform.mNombre.value)
    wait window "Nombre incorrecto..." nowait
     thisform.mNombre.setfocus
     return
endif

if empty( thisform.mPrograma.value)
   wait window "Programaincorrecto..." nowait
   thisform.mPrograma.setfocus
   return
endif

*validar el programa entrado por el usuario
mPrograma = alltrim( thisform.mPrograma.value)
sele programa
seek mPrograma
if !found()
     mMens1 = "El código de Programa " +mPrograma +"
no existe!" +chr(13)
     mMens2 = "Utilice un código válido e intente de
nuevo..."
     mOpc = messagebox( mMens1 +mMens2, 0+64,
"Atención!")
     thisform.mPrograma.setfocus
     return
endif

*actualizar datos del campo clave
sele 1
mCodigo = alltrim( thisform.mCodigo.value)
seek mCodigo
if found()  && actualizar registro hallado
    gather memvar memo
    wait window "Datos actualizados..." nowait
else
            mMens1 = "Imposible actualizar, el código " +mCodigo +"
no existe!" +chr(13)
    mMens2 = "Utilice otro código e intente actualizar de
nuevo..."
     mOpc = messagebox( mMens1 +mMens2, 0+64,
"Atención!")
     thisform.mCodigo.setfocus
     return
endif

sele 1

return
```

BtnExaminar.Click

```
*examinar registros
sele 1
browse fields codigo, nombre, programa noedit
noappend nodelete

*activar el seleccionado
scatter memvar memo

thisform.refresh
return
```

Form.Init

```
*abrir entorno utilizando las variables públicas

*tabla mae stro asignaturas
mFile1 = _dircia +"asigna.dbf"
if !file( mFile1)
        do cAsigna in prg\creatbl
endif

sele 1
use &mFile1 order tag codigo alias asigna

*tabla maestro programas
mFile2 = _dircia +"programa.dbf"
if !file( mFile2)
        do cPrograma in prg\creatbl
endif

sele 2
use &mFile2 order tag codigo alias programa

sele 1
scatter memvar memo

*ir al objeto inicial
thisform.mCodigo.setfocus

return
```

Observe que las modificaciones hechas al código, aparecen resaltadas en **Negrita**. Analice que cambios se hicieron al código elaborado en la guía #3 y analice el efecto que esas nuevas instrucciones tienen sobre el formulario.

Nota: Para que cada formulario se ejecute independiente de los demás, ajuste la propiedad *DataSession* del formulario a *2-Sesion Privada de Datos*. Esto le indicará al FoxPro que maneje el entorno de datos de cada formulario en porciones diferentes de la memoria. Si no se hace esto *(1-Sesión predeterminada de datos)*, se generarían innumerables errores de *"El archivo ya está en uso"* cada vez que se ejecute un formulario, inmediatamente después de haber ejecutado otro. Esta configuración especial le permitirá al usuario abrir varias instancias de un mismo formulario, tantas veces como la memoria se lo permita (existe en la documentación del FoxPro sobre la programación utilizando instancias de formularios).

Rutinas adicionales para que este diseño funcione:

Se requiere modificar la rutina que define las tablas en forma automática. Se debe agregar una nueva subrutina que permita definir la tabla *Asigna.dbf*.

CREATBL.PRG

Agregue este segmento de código a la rutina CREATBL.PRG

```
procedure cAsigna
        *definir tabla de asignaturas
        mFile = "c:\clasefox\01\asigna.dbf"
        create table &mFile ;
                ( codigo   c ( 10), ;
                  nombre   c ( 50), ;
                  programa c (  2))
        index on codigo tag codigo for !deleted()
        index on nombre tag nombre for !deleted()
        close data
return  && cAsigna
```

RETO DE PROGRAMACIÓN

Agregue al formulario un control de cuadro combiando que le permita al usuario el seleccionar de una lista de programas un código. Esto agilizará y ayudará al usuario a no tener que memorizar los códigos de los programas.

Para obtener una pantalla como la anterior, haga el siguiente proceso:

1. Elimine el objeto *mPrograma* del formulario *Asigna.*

2. Agregue un control de *Cuadro Combinado* de la barra de herramientas del formulario. Seleccione la herramienta Cuadro Combinado y demarque el área en el lugar donde estaba el objeto mPrograma. No exagere con el tamaño del nuevo objeto. Haga un diseño proporcional acorde a la cantidad de información a mostrar.

3. Modifique las siguientes propiedades del objeto recién agregado al formulario:
   - Name: ***mPrograma***
   - ControlSource: ***m.programa***
   - RowSource: ***programa.codigo, nombre***
   - RowSourceType: ***2-Alias***
   - ColumnWidths: ***40,200***

4. Si hizo todo el trabajo bien hecho, al ejecutar el formulario, verá una forma muy efectiva de trabajo, al permitir que el usuario seleccione los datos de las tablas relacionadas con las tablas maestras. Esto descarga al usuario, la obligación de tener que aprenderse una serie de códigos para introducir en los formularios.

INSTITUTO METROPOLITANO DE EDUCACIÓN
PROGRAMACIÓN DE COMPUTADORES
GUIA #5 DE VISUAL FOXPRO
DOCENTE: MAURICIO CANO

TÉCNICA DE PROGRAMACIÓN #2 – DISEÑO TIPO FOXPRO

La técnica de programación tipo Visual FoxPro es un diseño propuesto desde hace mucho tiempo por los fabricantes del producto en los ejemplos y ayudas del producto. Los programadores desde entonces vienen imitando ese modelo de programación, que para el viejo sistema DOS y Windows 3.1. era bastante funcional.

Muchos gerentes de sistemas prefieren que sus programadores diseñen de tal forma por ser un tipo de programación que controla bastante a las acciones del usuario. Esta técnica consiste en tener botones para indicarle a la aplicación cuando se desea *Agregar*, *Modificar* y *Retirar* registros. También posee unos botones para indicar si deseamos *Guardar* o *Deshacer* los cambios.

Lo interesante de esta técnica radica en la forma en que activan y se activan elementos en la pantalla. Esto brinda un mayor control de las acciones de los usuarios, al bloquear los campos de datos y aquellos objetos que se desean. El activar y desactivar elementos va reduciendo el modo de operación del usuario, limitándolo a aquello que solo deseamos que el usuario manipule en un momento determinado.

Recuerde que ahora que estamos en la etapa de *principiantes*, nos dedicaremos a construir cada pantalla sin los asistentes. Cuando tengamos los elementos básicos de la programación orientada a objetos aprenderemos a jugar con la clase *Wystile* para personalizar las pantallas de un proyecto.

EL FORMULARIO PARA EL MAESTRO DE ASIGNATURAS

Para esta pantalla, inicie un formulario en blanco y agregue los siguientes elementos:

1: mAgregar	2: btnModificar	3: btnRetirar	4: btnGuardar	5: btnDeshacer

Respecto a la técnica número 1, se han hecho cambios importantes en el funcionamiento de los botones *Agregar* y *Retirar*. Se han agregado los tres botones de *Modificar*, *Guardar* y *Deshacer*.

Para esta técnica, el usuario se ve obligado a utilizar los botones de agregar o modificar para actualizar los datos. Cuando haga clic en el botón *Agregar*, el sistema debe habilitar los campos de datos y botones *Guardar* o *Deshacer*. El resto de los botones, debe deshabilitarse durante los procesos de agregar o modificar los datos. Para empezar, coloque la propiedad *Enabled* igual a *.F. Falso* para los botones *btnGuardar, btnDeshacer* y los campos de datos (*mCodigo* y *mNombre*).

Evento INIT del Formulario	***BtnAgregar.click***
*variables públicas public pAgregando, pEditando *abrir entorno utilizando las variables públicas close data mFile1 = _dircia +"programa.dbf" if !file(mFile1) do cPrograma in prg\creartbl endif sele 1 use &mFile1 order tag codigo alias programa scatter memvar memo *ir al objeto inicial thisform.btnCerrar.setfocus return	*indicador (flag) de proceso para agregar datos pAgregando = .t. pEditando = .f. *cargar nuevos datos en blanco sele 1 scatter memvar blank *activar campos de datos thisform.mCodigo.enabled = .t. thisform.mNombre.enabled = .t. *desactivar botones innecesarios en el proceso *de agregar thisform.btnPrimero.enabled = .f. thisform.btnAnterior.enabled = .f. thisform.btnSiguiente.enabled = .f. thisform.btnUltimo.enabled = .f. thisform.btnAgregar.enabled = .f. thisform.btnModificar.enabled = .f. thisform.btnRetirar.enabled = .f. thisform.btnReporte.enabled = .f. thisform.btnExaminar.enabled = .f. thisform.btnCerrar.enabled = .f. *activar botones salvar y deshacer thisform.btnGuardar.enabled = .t. thisform.btnDeshacer.enabled = .t. *ir al primer campo thisform.mCodigo.setfocus thisform.refresh

BtnModificar.Clic

```
*indicador (flag) de proceso para agregar datos
pEditando  = .t.
pAgregando = .f.

*cargar datos en memoria
sele 1
scatter memvar memo

*el campo clave debe permanecer inactivo
thisform.mCodigo.enabled = .f.

*activar el resto de los campos
thisform.mNombre.enabled = .t.

*desactivar botones innecesarios en el proceso
*de agregar
thisform.btnPrimero.enabled = .f.
thisform.btnAnterior.enabled = .f.
thisform.btnSiguiente.enabled = .f.
thisform.btnUltimo.enabled = .f.

thisform.btnAgregar.enabled = .f.
thisform.btnModificar.enabled = .f.
thisform.btnRetirar.enabled = .f.

thisform.btnReporte.enabled = .f.
thisform.btnExaminar.enabled = .f.

thisform.btnCerrar.enabled = .f.

*activar botones salvar y deshacer
thisform.btnGuardar.enabled = .t.
thisform.btnDeshacer.enabled = .t.

*ir al siguiente campo disponible después
*del campo clave
thisform.mNombre.setfocus

thisform.refresh
```

BtnRetirar.Clic

```
*retirar el registro activo
mCodigo = alltrim( thisform.mCodigo.value)
mMens1 = "Desea retirar el código " + mCodigo +" del sistema!" +chr(13)
mOpc = messagebox( mMens1 , 1+32, "Retirar registro!")

if mOpc = 1 && Clic en botón Aceptar
	sele 1
	delete
	wait window "Registro retirado del sistema..." nowait

	*mover puntero al próximo disponible
	if !eof()
		skip 1
	endif
	if eof()
		go bottom
	endif
endif

scatter memvar memo
thisform.refresh
return
```

Al igual que en las rutinas anteriores, se requiere ejecutar la rutina *Public.prg* para programar el acceso a los datos del sistema y la rutina creadora de la tabla de los programas académicos *CREATBL.PRG* y la rutina (procedure) *cPrograma*. Ver las guías anteriores para mayor información.

BtnSalvar.clic

```
*guardar los datos

*verificar si hay datos en blanco
if empty( thisform.mCodigo.value)
	wait window "Código incorrecto..." nowait
	thisform.mCodigo.setfocus
	return
endif

if empty( thisform.mNombre.value)
	wait window "Nombre incorrecto..." nowait
	thisform.mNombre.setfocus
	return
endif

*verificar el campo clave si se esta agregando
if pAgregando
	sele 1
	mCodigo = alltrim( thisform.mCodigo.value)
	seek mCodigo
	if found()
		mMens1 = "Imposible guardar, el código " +mCodigo +" existe!" +chr(13)
		mMens2 = "Utilice otro código e intente Agregar de nuevo..."
		 mOpc = messagebox( mMens1 +mMens2, 0+64, "Atención!")
		thisform.mCodigo.setfocus
		return
	endif
endif

*chequear indicador de proceso
if pAgregando
	*agregar registro
	insert into programa from memvar
	wait window "Registro agregado..." nowait
else
	*actualizar datos del registro
	gather memvar memo
	wait window "Registro modificado..." nowait
endif
```

```
*colocar indicadores a falso
pAgregando = .f.
pEditando  = .f.

*desactivar campos de datos
thisform.mCodigo.enabled = .f.
thisform.mNombre.enabled = .f.

*activar botones necesarios
thisform.btnPrimero.enabled = .t.
thisform.btnAnterior.enabled = .t.
thisform.btnSiguiente.enabled = .t.
thisform.btnUltimo.enabled = .t.

thisform.btnAgregar.enabled = .t.
thisform.btnModificar.enabled = .t.
thisform.btnRetirar.enabled = .t.

thisform.btnReporte.enabled = .t.
thisform.btnExaminar.enabled = .t.

thisform.btnCerrar.enabled = .t.

*desactivar botones salvar y deshacer
thisform.btnGuardar.enabled = .f.
thisform.btnDeshacer.enabled = .f.

return
```

BtnDeshacer.clic

```
*deshacer los cambios hechos a los datos
sele 1
scatter memvar memo

wait window "Proceso de Edición cancelado..." nowait

*colocar indicadores a falso
pAgregando = .f.
pEditando  = .f.

*desactivar campos de datos
thisform.mCodigo.enabled = .f.
thisform.mNombre.enabled = .f.

*activar botones necesarios
thisform.btnPrimero.enabled = .t.
thisform.btnAnterior.enabled = .t.
thisform.btnSiguiente.enabled = .t.
thisform.btnUltimo.enabled = .t.

thisform.btnAgregar.enabled = .t.
thisform.btnModificar.enabled = .t.
thisform.btnRetirar.enabled = .t.

thisform.btnReporte.enabled = .t.
thisform.btnExaminar.enabled = .t.

thisform.btnCerrar.enabled = .t.

*desactivar botones salvar y deshacer
thisform.btnGuardar.enabled = .f.
thisform.btnDeshacer.enabled = .f.

thisform.refresh

return
```

Y ahora manos a la obra!
Aplique esta técnica de programación,
con la tabla maestro de Asignaturas.

COMPLEMENTO A LA GUÍA #5

LA PROGRAMACIÓN MULTIUSUARIA (PARTE 1)

En esta primer entrega estudiaremos un poco el concepto de la programación multiusuaria, un tema que pone a temblar a más de un programador. En realidad, el programar acceso múltiple de usuarios a las mismas tablas es algo sencillo en el FoxPro, basta con tener en cuenta las consideraciones planteadas en este documento, y usted quedará listo para desarrollar aplicaciones en el sistema operativo de red que sea (lógico está, siempre y cuando soporte Visual FoxPro).

ACCESO EXCLUSIVO (EXCLUSIVE) Y COMPARTIDO (SHARED) A LAS TABLAS

Cuando se desarrolla una aplicación que se requiere funcionando en redes, el programador debe considerar la posibilidad de muchos usuarios accesando a las mismas tablas. Para esto es importante saber que existe el acceso exclusivo por parte de un usuario a una tabla y el acceso compartido a las mismas.

Cuando se ejecuta acceso exclusivo a una tabla, el sistema solo le permite el acceso a la tabla al usuario que hizo la solicitud, cuando otros usuarios traten de acceder a la tabla abierta como de uso exclusivo para otro usuario, se generará un error de acceso a la tabla, lo cual puede paralizar una aplicación si no se tienen controles adecuados de error.

El acceso compartido, FoxPro permite que varios usuarios utilicen la información de las mismas tablas simultáneamente.

BLOQUEO (RECORD LOCK) DE REGISTROS Y ARCHIVOS (FILE LOCK)

El bloqueo de un registro consiste en que el sistema asegura el registro solicitado con el objetivo de que otras personas no puedan utilizarlo durante el tiempo en que se halle bloqueado por otro usuario. Muchos procesos requieren que los registros estén bloqueados con el objetivo de mantener la integridad de la información de una tabla determinada. Procesos tales como la modificación de registros, el borrado, etc, requieren ejecutar una orden de bloqueo; una vez el usuario suelte el registro, la aplicación debe desbloquear el registro (UNLOCK) para dejarlo disponible a los otros usuarios, de lo contrario, el registro seguiría bloqueado hasta que se cierre la sesión del usuario en la aplicación.

El bloqueo de archivos (FILE LOCK) consiste en asegurar un archivo completo con el objetivo de utilizar un archivo, ejecutar algún proceso delicado, y estar seguros de que nadie más manipulará la información o registros del mismo. Procesos como la generación de entradas de índices (INDEX ON), el empaquetamiento (PACK) de los registros marcados para borrar (con la orden DELETE), la modificación de la estructura de una tabla (MODIFY STRUCTURE), etc, requieren del bloqueo total de una tabla para su ejecución.

Es importante además, el tener en cuenta que muchas órdenes y funciones del FoxPro requieren el bloqueo de registros o archivos para su ejecución. Cada vez que el programador requiere utilizar una de estas órdenes, deberá ejecutar el respectivo bloqueo, ejecutar la orden, y liberar el registro o la tabla bloqueada para hacerla disponible al resto de los usuarios.

Debe consultar el manual del sistema FoxPro con el objetivo de conocer las órdenes que requieren tales bloqueos en los registros y en las tablas.

Nota: *En la siguiente entrega, mostraremos las órdenes requeridas para la programación de aplicaciones multiusarias.*

LA RUTINA PARA EL CONTROL DE ERRORES

El FoxPro permite la manipulación de los errores ocurridos durante la ejecución de una aplicación. A estas alturas, el alumno ya habrá descubierto que existen varias clases de errores cuando se trabaja en una aplicación:

Errores en tiempo de compilación (errores de sintaxis): estos errores ocurren cuando se ha escrito mal un comando o función. Cuando se ha escrito mal un comando o se han utilizado parámetros incorrectos, FoxPro paralizará la ejecución del programa y sacará un mensaje de error en una caja de diálogo. Este tipo de errores son causa de una mala programación, y el programador deberá ejecutar y probar sus rutinas para depurar el código de su aplicación y estar "*tranquilo*" de que todo funciona. El mismo programador, es el único responsable de que su aplicación trabaje sin errores y el solo, debe analizar la causa del error encontrado durante la ejecución de sus aplicaciones.

Errores en tiempo de ejecución: Estos errores aparecen cuando, una vez depurado el código de una aplicación, el FoxPro paraliza la ejecución por alguna razón que se sale de nuestras manos; sin embargo, con una buena rutina de control de errores, podremos indicarle a la aplicación que hacer cuando sucedan tales errores. Errores tales como *disco lleno, Registro o archivo bloqueado por otro usuario, archivo no existe, etc,* son errores que pueden ser controlados y manipulados por una buena rutina de control de errores. Más adelante en las clases, se tratará el tema de el manejo de los errores dentro de los programas.

Con una rutina de control de errores, el programador controlará la situación de si permite la continuación o no de la aplicación, o determinar soluciones para continuar con la rutina iniciada. Un error por ejemplo, se presentaría cuando el sistema trate de abrir una tabla que por algún motivo fue retirada del sistema por otro usuario; la rutina de control de errores se encontraría con la ausencia de tal tabla, generando automáticamente un error de archivo no existe, esa misma rutina podría determinar el nombre origen de la tabla faltante y por supuesto, crearla automáticamente dentro del directorio de datos. Luego de definido el nuevo archivo, puede indicarle al FoxPro que repita la orden que generó el error y que siga la ejecución de la aplicación como si nada hubiese pasado.

Para adelantar trabajo, busque en el manual del sistema FoxPro, el comando ON ERROR y analice el ejemplo que acompaña a la teoría para ir comprendiendo este tema tan importante a la hora de programar aplicaciones.

INSTITUTO METROPOLITANO DE EDUCACIÓN
PROGRAMACIÓN DE COMPUTADORES
GUIA #5 DE VISUAL FOXPRO
DOCENTE: MAURICIO CANO

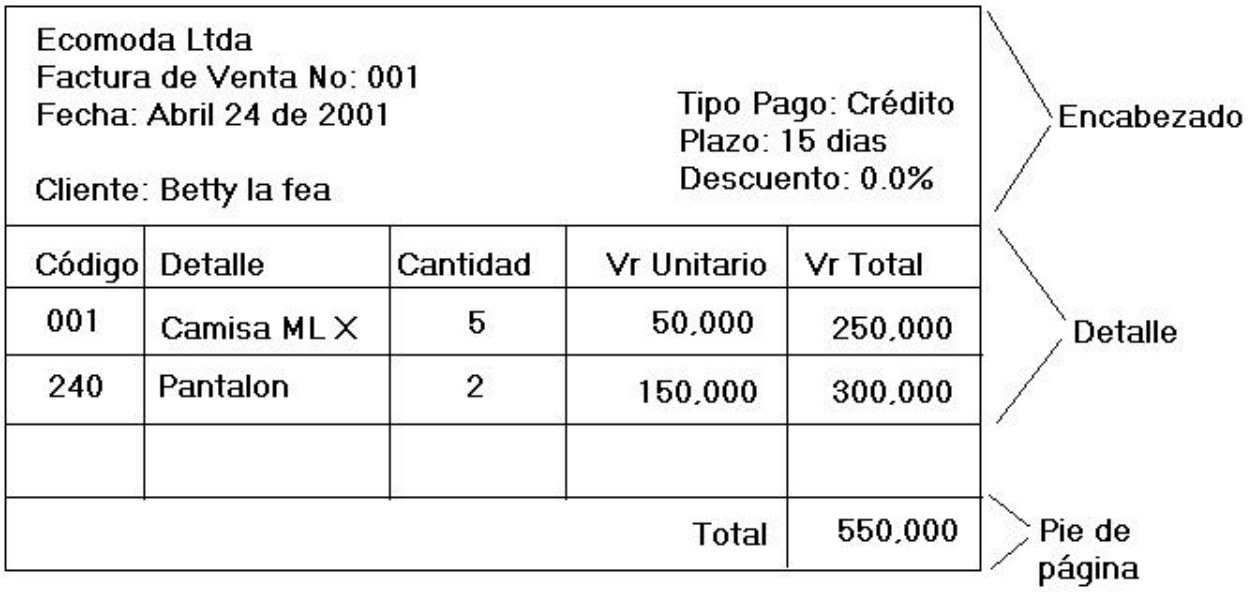
TÉCNICA DE PROGRAMACIÓN #3 – FORMULARIOS TIPO FACTURA

Existe una rutina muy popular entre los programadores y la cual se constituye en el "*quebradero de cabeza*" de más de un iniciado en la programación: el diseño de formularios que permitan la captura de información en documentos tales como la factura.

Un documento como la factura (bien sea de compra o de venta) tiene un encabezado y un detalle. También tienen esta característica los pedidos, los documentos soporte de inventario, una lista de grupos, un proceso de calificaciones, etc.

Cualquier documento empleado por una empresa y que tenga una presentación que incluya encabezado del documento y una descripción de detalle, debe ser automatizado empleando la técnica número 3. Es de aclarar que tal situación de programación se puede solucionar de muchas formas, planteamos en este curso, un método sencillo y práctico de automatización.

La siguiente gráfica muestra un posible documento de factura de compra. Aunque es bastante sencillo ilustra la característica de los documentos que requieren aplicar la técnica de programación propuesta en esta guía.



En donde reside lo complicado de programar un formulario para este documento? Cuando se efectúa el diseño de la base de datos, se descubre que es necesario dos tablas para solucionar el requerimiento del documento: la principal y el detalle, relacionando ambas tablas por un campo común, usualmente el campo clave de la tabla principal, en este caso *Número de factura.* Lo cual indica, que en la parte del detalle, el programa deberá visualizar solamente a los registros que posean el mismo número de factura del documento visualizado en la parte del encabezado. Otro asunto importante, es el de permitir manipular en forma ágil el detalle, permitiendo agregar, modificar y retirar información de las líneas que constituyen el detalle, y actualizando el total del documento en el pie de página del documento. Es decir, cada vez que se actualice la cantidad o el valor unitario de una de las líneas del detalle, se debe reflejar tales cambios en todo el documento.

FORMULARIOS TIPO FACTURA PROPUESTO

Como modelo de programación utilizaremos el control de los grupos de clase de una institución. Como parte principal del documento aparecerá la información relevante a un grupo de clases y como detalle, se colocarán todos los estudiantes pertenecientes a cada grupo. Utilizaremos además, uno de los mejores objetos incluidos en el Visual FoxPro, el *Grid (cuadrícula)*. Como parámetro de los elementos a mostrar en el *grid*, utilizaremos una instrucción *Select de SQL*.

El diseño utilizará las siguientes tablas: Programas, Asignaturas, Profesores, Hoja de vida, Matrícula, Grupos y Estudiantes por Grupo.

1. Grupos (*Grupos*)		
Nombre Campo	*Tipo de datos*	*Ancho*
Codigo	Carácter	10
Nombre	Carácter	50
Asignatura	Carácter	10
Horario	Carácter	50
Aula	Carácter	50
Profesor	Carácter	15
Campo clave ascendente: Codigo		

2. Estudiantes por Grupo (*EstGrupo*)		
Nombre Campo	*Tipo de datos*	*Ancho*
Codigo	Carácter	10
CodEstud	Carácter	10
ExParcial	Numerico	4,2
Seguim	Numerico	4,2
ExFinal	Numerico	4,2
Wfinal	Numerico	4,2
NotaFinal	Numerico	4,2
Estado	Carácter	1
Campo clave ascendente: Codigo + CodEstud		

3. Hojas de vida (Hvida)		
Nombre Campo	*Tipo de datos*	*Ancho*
Codigo	Carácter	10
Nombre	Carácter	50
Apellidos	Carácter	50
Programa	Carácter	2
TelResiden	Carácter	20
DirResiden	Carácter	40
FechaNac	Fecha	8
Campo clave ascendente: Codigo		

4. Matrícula (Matric)		
Nombre Campo	*Tipo de datos*	*Ancho*
Codigo	Carácter	10
Fecha	Fecha	8
Nivel	Carácter	2
Jornada	Carácter	1
Estado	Carácter	1
Observa	Memo	10
Campo clave ascendente: Codigo		

5. Maestro de Profesor (*Profe*)		
Nombre Campo	*Tipo de datos*	*Ancho*
Codigo	Carácter	15
Nombre	Carácter	50
TelResiden	Carácter	15
TelTrabajo	Carácter	15
Asignatura	Carácter	100
Campo clave ascendente: Codigo		

6. Maestro de Programas (Programa)		
Nombre Campo	*Tipo de datos*	*Ancho*
Codigo	Carácter	2
Nombre	Carácter	50
Campo clave ascendente: Codigo		

7. Maestro de Asignaturas (Asigna)		
Nombre Campo	*Tipo de datos*	*Ancho*
Codigo	Carácter	10
Nombre	Carácter	50
Programa	Carácter	2
Campo clave ascendente: Codigo		

Esta clase de rutina es un poco compleja debido al gran número de tablas que se requieren para su funcionamiento. Observe que para la rutina de grupos se requiere tener almacenados datos de programas, asignaturas, profesores, hojas de vida y datos de matrícula de los estudiantes. Si no se tienen datos en tales tablas, el diseño no funcionará debido a los controles de datos que efectúa sobre los datos entrados por el usuario.