

MANUALES PROFESIONALES

Guía Práctica

Microsoft®

Curso Teórico práctico de Programación

Domine Visual FoxPro 9.0

Desarrolle aplicaciones
de alta performance y
robustez en pocos
minutos

SP2

Ernesto Fabián Coronel

Analista de Sistemas en Computación



**Guía Práctica de consulta para el Ingeniero,
Analista de Sistemas, Profesor y Desarrollador Profesional**

Sobre el Autor

Ernesto Fabián Coronel ~ Analista de Sistemas en Computación.

Es egresado de la U.E.P 69 de la ciudad de Villa Ángela, Chaco, con el Título de Analista de Sistemas en computación. Se ha iniciado en el mundo de la informática desde los principios de los años 1980 cuando los microordenadores eran grotescos teclados que se conectaban a los viejos televisores y el único lenguaje en estos dispositivos era el famoso Basic, y los manuales estaban en ingles y portugués ya que la mayoría de ellos venían del Brasil.

Actualmente trabaja como desarrollador de software y consultor informático en el área de empresas y usuarios hogareños.

Es estudiante de Abogacía en la Facultad de Derecho de la UNNE de la Extensión Universitaria de General Pinedo, es autodidacta en Idioma de Ingles y Ruso.



INTRODUCCION

El avance que ha experimentado el desarrollo del software hace que un programador o desarrollador de sistemas informáticos deba usar como herramienta de programación dos y hasta tres lenguajes para obtener una aplicación final. Por suerte con Visual FoxPro no necesitamos llegar a este punto, disponemos de todas las herramientas para generar tantas las “clásicas aplicaciones” de sistemas tales como administración, gestión, pos y como si eso fuera poco podemos desarrollar aplicaciones para Pda y Palm, permitiendo portabilidad de datos y aplicaciones en forma remota y online en caso de ser necesario.!!

Visual FoxPro no conoce límites, su gran capacidad de almacenamiento en tablas y su enriquecido entorno I.D.E, cumpliendo con los estándares del tipo RAD, esto nos permite generar todo tipo aplicación en pocas horas y minutos en algunos casos.

A partir de la versión 7.0 Visual FoxPro se hace multi lenguaje, se puede desarrollar aplicaciones en diferentes idiomas con tan solo tener configurada la configuración regional de Windows y Listo!!. Esta prestación entre otras, hizo posicionar a esta herramienta de desarrollo en Número 1 para los programadores y desarrollares en distintos países del mundo. Después de varios años Microsoft incorpora esta utilidad en otros lenguajes.

Esta manual pretende ser una guía para toda persona que le gusta o son entusiastas en el “arte de programar” o desarrollador, concepto adoptado para adaptarlo a los nuevos paradigmas de los lenguajes de programación actuales.

En programación el mejor profesor es la experiencia, y al igual que el piloto comercial o militar, uno elige al mas avezado y con mejor experiencia, en el desarrollo de sistemas las horas de desarrollo y herramientas de avanzada convierten al programador o desarrollador en el favorito para ganarse un lugar entre los “genios” de eso que es tan temido por algunos llamado “programación”. Al igual que en medicina, la teoría y la practica es un todo, en desarrollo de programación es lo mismo.

La programación de Pcs tiene algo divino: Absolutamente de la nada aparecen sistemas o programas maravillosos y muy complejos, sea Usted un creador y desarrollar de sistemas, a esto apunta este libro.

Antes de comenzar con el desarrollo de Visual FoxPro, damos una breve reseña útil en ciertos conceptos generales a cualquier lenguaje de programación y “refrescamos” la memoria para esos que cometen el error de “saltar” la teoría, para ir a escribir código en la Pcs.

AGRADECIMIENTOS

La inmortalidad es un sueño casi inalcanzable para algunos, no para los más futuristas, pero una forma de tener una pizca de inmortalidad, es tener hijos, una familia, unos amigos y escribir un libro.

Agradezco a mi familia por siempre, por apoyarme en el tiempo que no les dedique, por estar abocado en mi “proyecto de escribir este libro”.

A mi esposa Julia e hijos: Igor, Ernesto y Cristina.

DEDICATORIAS

A mis amigos:

A Michi, Marcos, Luís, Julio, Guille, Cacho, Pablo y Sergio alias “peguí” por el apoyo incondicional en la Universidad y las incontables cenas de “estudio”.

A Vitalii Schev SPACIBO DRUG (gracias amigo en ruso)

Especialmente a nuestra Querida Presidenta de la Nación: Cristina Isabel Fernández de Kirchner. DIOS SALVE VUESTRA HONORABILIDAD.

A todos GRACIAS, os quiero.

Todas las marcas mencionadas en este libro son propiedad de sus respectivos dueños.

ef_coronel@hotmail.com

VISUAL FOXPRO 9.0 SP2 – Capitulo Preliminar

¿QUÉ ES UN LENGUAJE DE PROGRAMACION?

Es un lenguaje que se utiliza para controlar el comportamiento de una máquina, particularmente de las computadoras. Un conjunto de reglas sintácticas y semánticas que definen la estructura y el significado de sus elementos, respectivamente. Un lenguaje de programación le permite al programador especificar con precisión sobre qué datos una computadora debe operar, cómo deben ser almacenados y transmitidos, y qué acciones debe tomar bajo determinadas circunstancias. Todo esto, mediante un lenguaje que intenta estar relativamente cerca del lenguaje humano o natural.

¿QUÉ ES LA PROGRAMACIÓN?

En términos informáticos, se denomina programación a la creación de un programa de computadora, un conjunto de instrucciones que la computadora puede interpretar, ya sea código máquina compilado e interpretado directamente desde el núcleo del sistema operativo o desde un Script, o código fuente interpretado en tiempo de ejecución o uso. El programa se puede escribir en un lenguaje de programación o directamente en lenguaje de máquina. Éste último trae aparejadas ciertas dificultades. Un programa se puede dividir en partes o módulos, que pueden estar escritos en lenguajes distintos.

¿QUÉ ES UN PROGRAMADOR?

Se denomina programador a toda aquella persona que escribe programas o software para computadoras, que permiten optimizar tareas de diversos tipos. Un programador que también realiza análisis de sistemas y diseño puede denominarse analista / programador.

¿QUÉ ES UN ALGORITMO?

Un algoritmo es secuencia ordenada y sistemática de pasos que lleva a la solución de un problema o realizar cierto cálculo complejo. Se debe tener en cuenta la importancia del orden de las operaciones. El programador debe tener la habilidad de para deducir algoritmos a partir de un problema, ya que esta tarea conforma el primer paso para la implementación de cualquier aplicación de software de software. Los diagramas de flujo ayudan en forma visual y precisa en los puntos de decisión.

PROGRAMACIÓN ORIENTADA A OBJETOS: DEFINICIÓN

La programación orientada a objetos (POO u OOP, según sus siglas en inglés) define los programas en términos de clases de objetos, objetos que son entidades que combinan estado (es decir, datos), comportamiento (esto es, procedimientos o métodos) e identidad (propiedad del objeto que lo diferencia del resto). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar. De esta forma, un objeto contiene toda la información, (los denominados atributos) que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases (e incluso entre objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos). A su vez, dispone de mecanismos de interacción (los llamados métodos), que favorecen la comunicación entre los objetos (de una misma clase o de distintas), y en consecuencia, el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separan (ni deben separarse) información (datos) y procesamiento (métodos). - Dada esta propiedad de conjunto de una clase de objetos, que al contar con una serie de atributos definitorios requieren de unos métodos para poder tratarlos (lo que hace que ambos conceptos están muy entrelazados), el programador debe pensar indistintamente en ambos términos, ya que no nunca debe separar o dar mayor importancia a los atributos en favor de los métodos, ni viceversa. Hacerlo puede llevar al programador a seguir el hábito erróneo de crear clases contenedoras de información por un lado y clases con métodos que manejen esa información por otro lado (llegando a una programación estructurada camuflada en un lenguaje de programación orientado a objetos). - Esto difiere de la programación estructurada tradicional, en la que los datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el procesamiento de unos datos de entrada para obtener otros de salida. La programación estructurada anima al programador a pensar sobre todo en términos de procedimientos o funciones, y en segundo lugar en las estructuras de datos que esos procedimientos manejan. En la programación estructurada se escriben funciones y después se les pasan datos. Los programadores que emplean lenguajes orientados a objetos definen objetos con datos y métodos, y después les envían mensajes a los objetos para que realicen esos métodos. - Algunas personas también distinguen la POO sin clases, que a veces se denomina programación basada en objetos.

BIENVENIDOS!

El Administrador de Proyectos.

El Administrador de Proyectos.

El Administrador de proyectos es la herramienta principal para construir nuestras aplicaciones. A través de esta herramienta comenzaremos a diseñar, modificar y ejecutar todas las tareas comunes a nuestro proyecto. Todo nuestro desarrollo rondará entorno a un proyecto.

A través de esta herramienta contenedora de tipo visual, estará todo al alcance de nuestras manos pudiendo de esta manera organizar todo en un solo proyecto, siendo esto las bases de datos, Tablas libres, Formularios, Consultas, Menús, Clases, Librerías, Programas, imágenes que harán a nuestras aplicaciones más vistosas y también podemos incorporar los famosos formatos Gifs animados para que nuestras aplicaciones tengan una apariencia de estar “vivos” al tener movimientos. El buen diseño gráfico de nuestros sistemas hará que sean más vendibles nuestros productos.

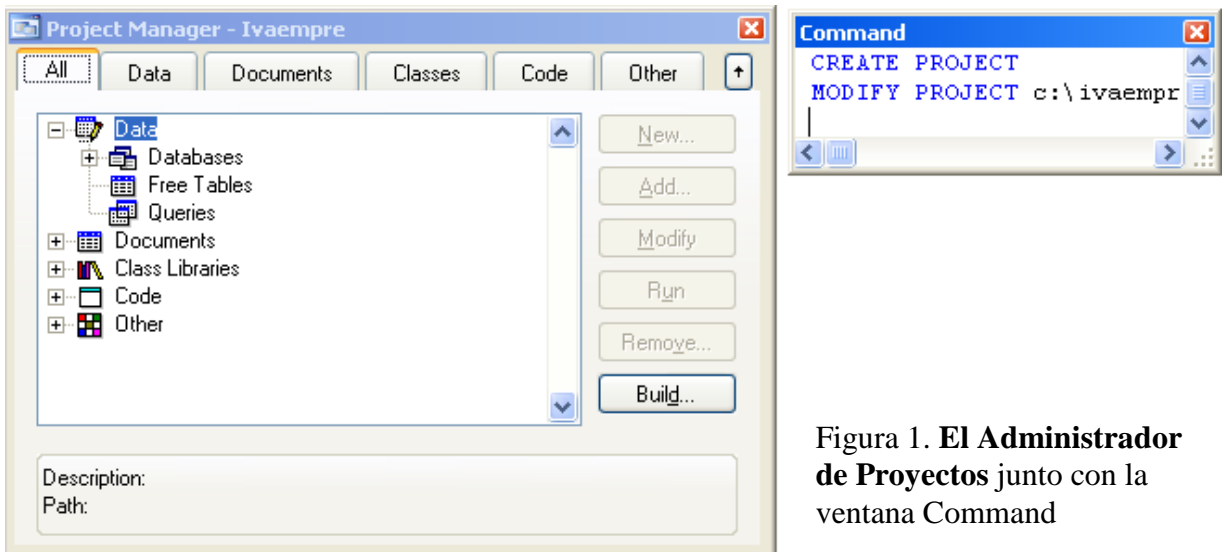


Figura 1. El Administrador de Proyectos junto con la ventana Command

En la Figura 1 podemos observar el proyecto llamado “Ivaempre”, podemos ver que contiene seis solapas principales ordenadas por el contenido según el tipo de archivos. Si hacemos clic en **Data**, podremos ver a las bases de datos, tablas libres y consultas contenidas en ella.

Iniciando El Administrador de Proyectos.

Cuando iniciamos un nuevo proyecto, el mismo estará vacío, en el podemos ir agregando nuestros elementos o bien agregar los existentes en el.

Iniciemos un proyecto nuevo a modo de ejemplos. Iniciemos Visual Foxpro y cuando aparezca el menú hacemos clic en File, Luego en New y se nos aparecera un cuadro de opciones que estará marcada por defecto en proyecto y hacemos clic en New File.

Se debe ingresar un nombre de Proyecto, ingresamos el nombre de “**example 1**” y hacemos clic en Save. En la figura 2. Tenemos un ejemplo de cómo empezar.

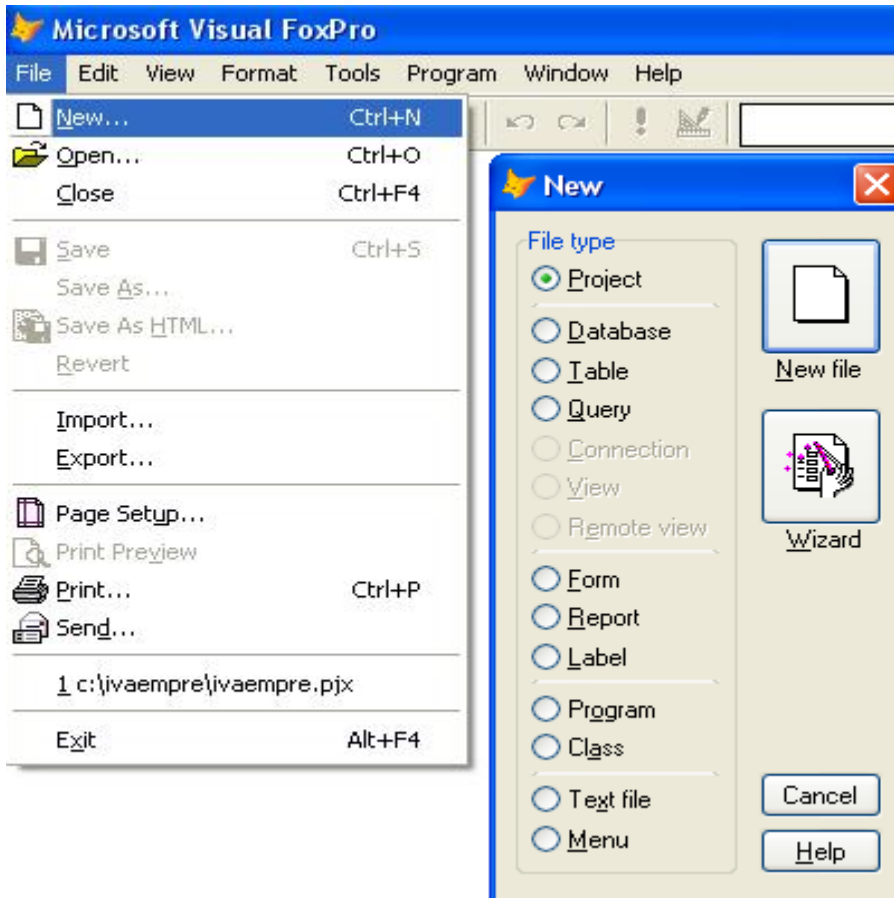


Figura 2. Iniciando el Administrador de proyectos

Estaremos frente a una pantalla similar a la figura 3. Inicio de un proyecto Nuevo.

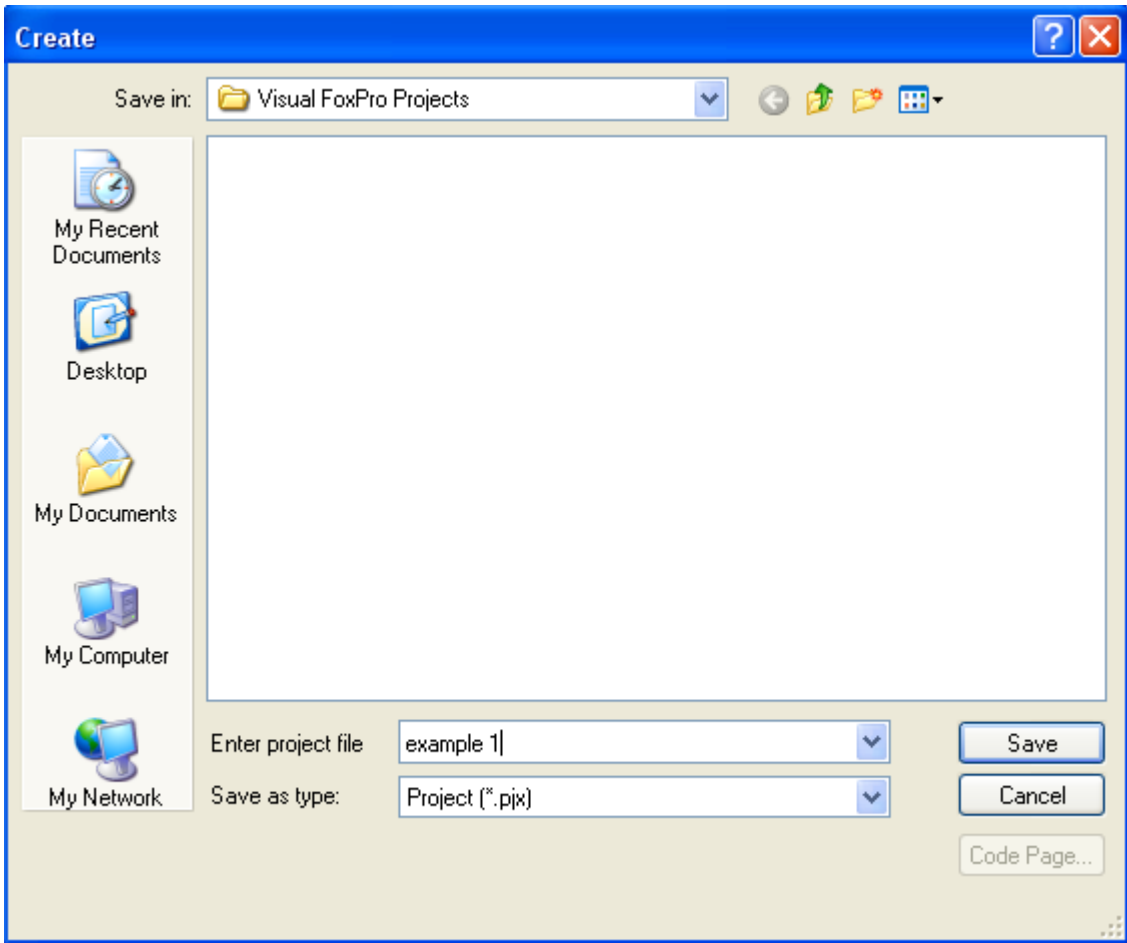


Figura 3. Proyecto **example 1**

Proyecto Example 1 con todas las opciones principales desplegadas. Figura 4

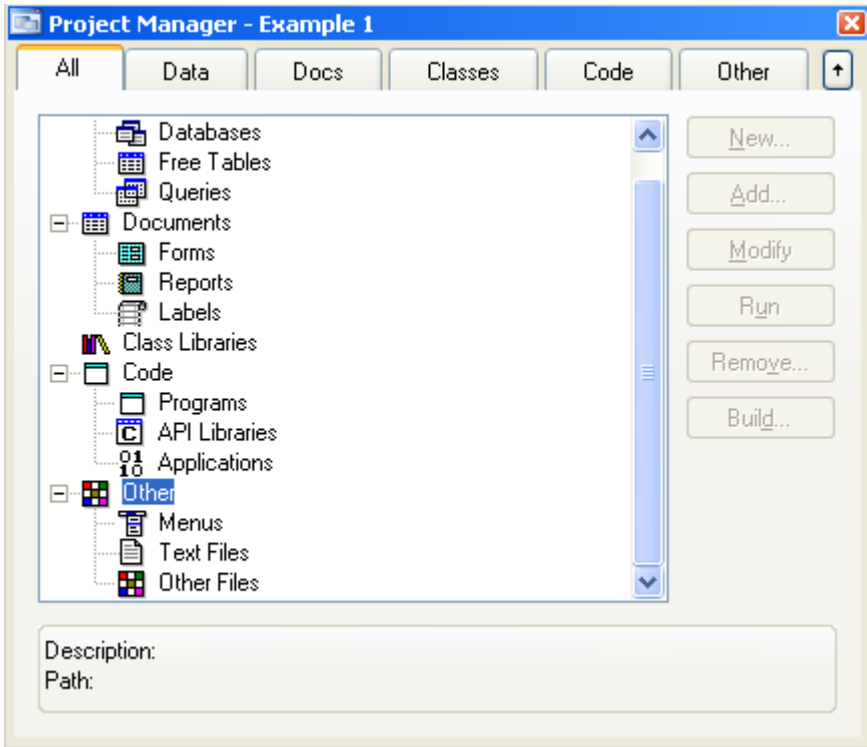


Figura 4. Proyecto example 1

Ahora estamos listos para empezar a desarrollar una aplicación con todas sus opciones de: Bases de Datos, Tablas Libres, Consultas, formularios, Reportes, Etiquetas, Librerías de Clases, Código de programa (muy útil para configurar las opciones de inicio de nuestras aplicaciones), Menús al mejor estilo Windows y además poder agregar las imágenes e iconos que harán mas intuitivos nuestros sistemas.

Ahora es el momento de empezar a conocer las opciones de trabajo del Administrador de Proyectos. Empecemos!!

Si disponemos de un monitor de 17 Pulgadas como mínimo, no será necesario hacer clic en cada solapa, ya que teniendo desplegada la solapa **todos**, podremos ver todas las opciones necesarias para trabajar.

Comencemos mencionando brevemente las cinco solapas del Administrador de Proyectos.

Datos:

Se desplegará una figura tal cual como lo muestra la **figura 5**.

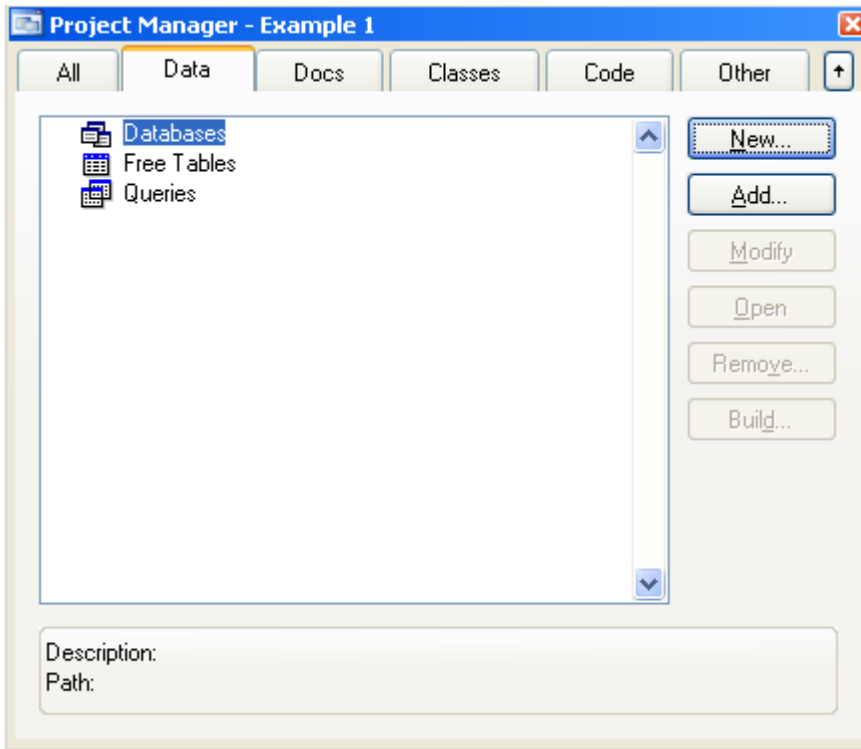


Figura 5. Datos

Desde aquí estamos en condiciones de crear o agregar una base de datos, que actuara como contenedor de las tablas dependientes, lo cual nos va a permitir desplegar todo la potencia relacional que dispone Visual FoxPro con las tablas dependientes y para los programadores tipo Xbase (Dbase, Clipper, etc) existe la posibilidad de trabajar con tablas libres y contamos

Además con la opción de crear o agregar consultas, esto no es más que comandos SQL automatizados en cuestión de minutos con un asistente intuitivo.

Documentos:

En esta solapa sera pasaremos la mayor parte de nuestro tiempo, ya que desde aquí pondremos toda nuestra potencia y creatividad a la hora de diseñar aplicaciones. **La Figura 6** nos muestra las opciones que disponemos: Formularios, Informes, y etiquetas. Es la solapa conocida como la de “trabajo” ya que la interfase que verán los usuarios de nuestras aplicaciones, serán creadas, modificadas y rediseñadas desde aquí.

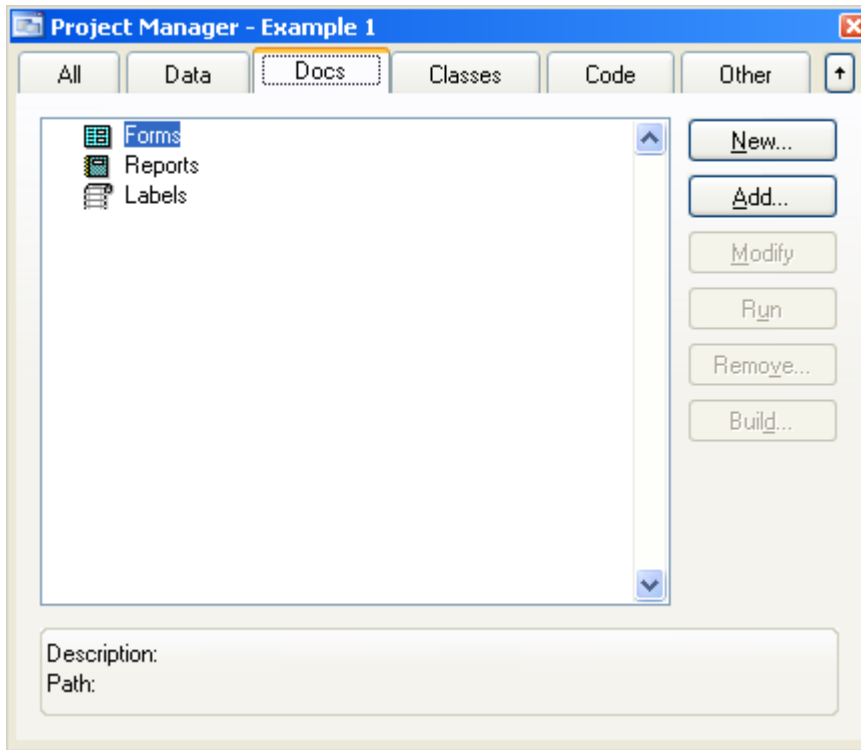


Figura 6. Documentos

Clases:

en la figura 7 tenemos la figura de clases, esta herramienta nos permite ahorrar horas y horas de programación una vez que hayamos desarrollado varias aplicaciones ya que demanda un tiempo considerable el diseño de las misma, ya que tenemos clases visuales y no visuales, tambien tenemos la posibilidad de utilizar clases de otros programadores que no esten protegidas, Visual FoxPro es pionero y uno de los primeros lenguajes en ofrecer esta herramienta de avanzada desde hace mas de 10 años atrás, ya que evita que el programador o desarrollador malgaste horas de escritura de código y diseño repetitivo para la misma aplicación como así tambien como para otras. Aquí entra en juego conceptos nuevos en programación de Clases, Herencia, y polimorfismos. Se ha dedicado un capitulo entero sobre este tema ya que es muy complejo explicarlo en unas pocas líneas.

La clase debe tener un nombre y estar basada en un objeto o en una clase no visual y estar almacenada en un lugar determinado.

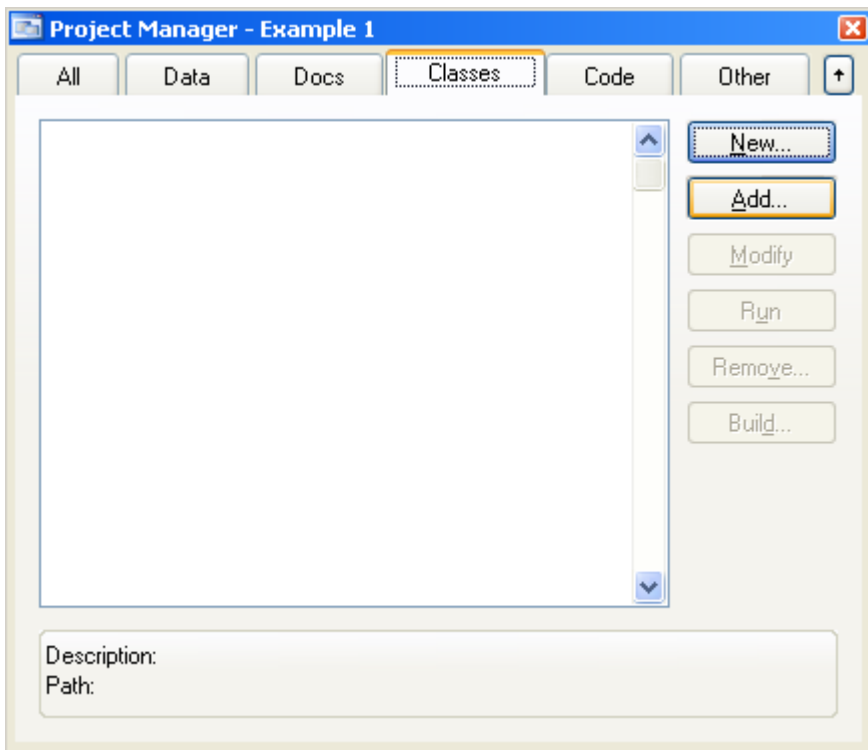
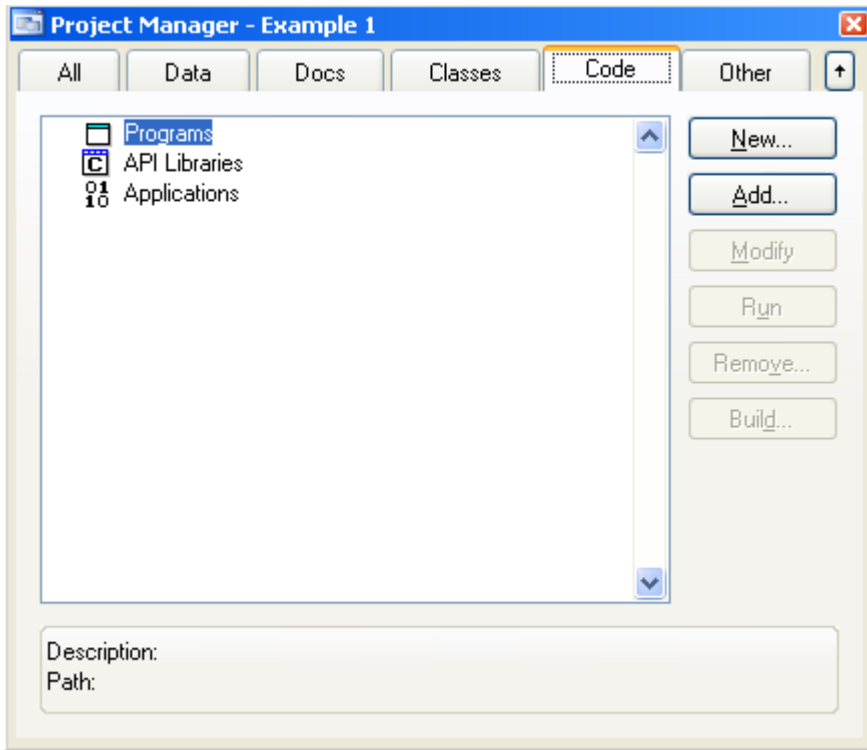


Figura 7. Clases

Code:

En esta solapa **code** (**Figura 8**) podremos incluir nuestros programas (prg), que serán muy útiles al momento de iniciar nuestras aplicaciones y las bibliotecas de Windows (API) y otras librerías propias como lo que maneja Visual FoxPro (FLL). Al momento de desarrollar nuestra aplicación de ejemplo daremos ejemplos prácticos sobre estos temas.

**Figura 8. Code****Other :**

En esta solapa contamos con la posibilidad de agregar menús a nuestras aplicaciones y también imágenes tipo bmp, jpg, y archivos gif animados.

Asistentes rápidos.

Comenzaremos con mostrar con un ejemplo para ir viendo las virtudes del tipo R.A.D que tenemos para desarrollar aplicaciones en tiempo mínimo. Empecemos con dos ejemplos:

Iniciemos Visual FoxPro y en caso de no estar abierto el proyecto ejemplo 1, (Visual FoxPro crea por defecto dentro de la carpeta mis documentos una carpeta especial llamada “Visual FoxPro Projects”) debemos abrirlo haciendo Clic en File -> Open y buscamos el proyecto ejemplo 1 y hacemos clic en OK. Tal cual como lo muestra la siguiente figura 9.

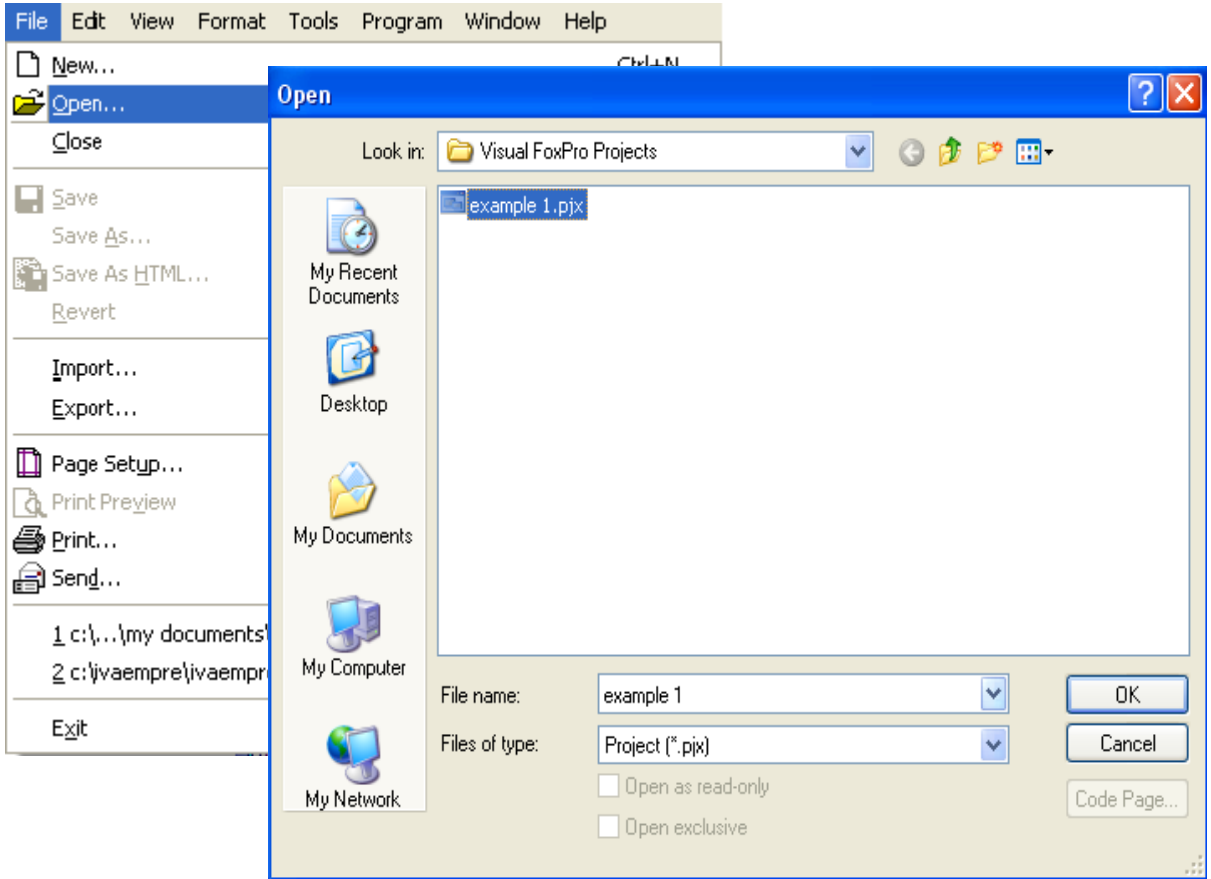


Figura 9. abrir el proyecto **example 1**

De esta manera ya estamos listos para trabajar con nuestro primer proyecto **example 1**.

Teniendo abierto el proyecto **example 1**, ya estamos comenzando a diseñar nuestra primera aplicación. En primer lugar debemos selección la solapa **data** ó en caso de estar activada la opción de todos, hacemos clic en **Databases**, y luego en **New** para crear la **Base de datos**, esto es para crear una base de datos que actuara como “contenedor” de todas nuestras tablas dependiente de la base de datos a crear, luego se mostrara en un pequeño formulario titulado **New Database**, hacemos clic en la opción “**New Database**” que muestra como figura una hoja en Blanco con un pequeño doblado en el margen superior derecho .

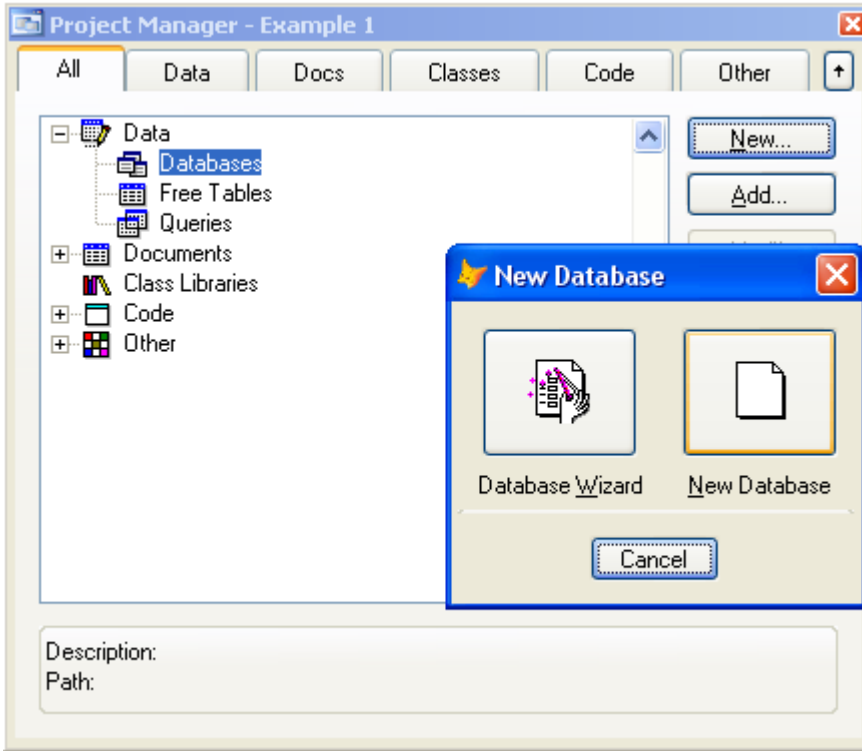


Figura 10. Creando una

base de datos.

Una vez hecho clic en el botón **New Database**, Tendremos una imagen similar en pantalla a la figura 11 en la que se nos pedirá en nombre de la “Base de Datos” , la llamaremos con el nombre de “**sistema**” y hacemos clic en save o guardar.

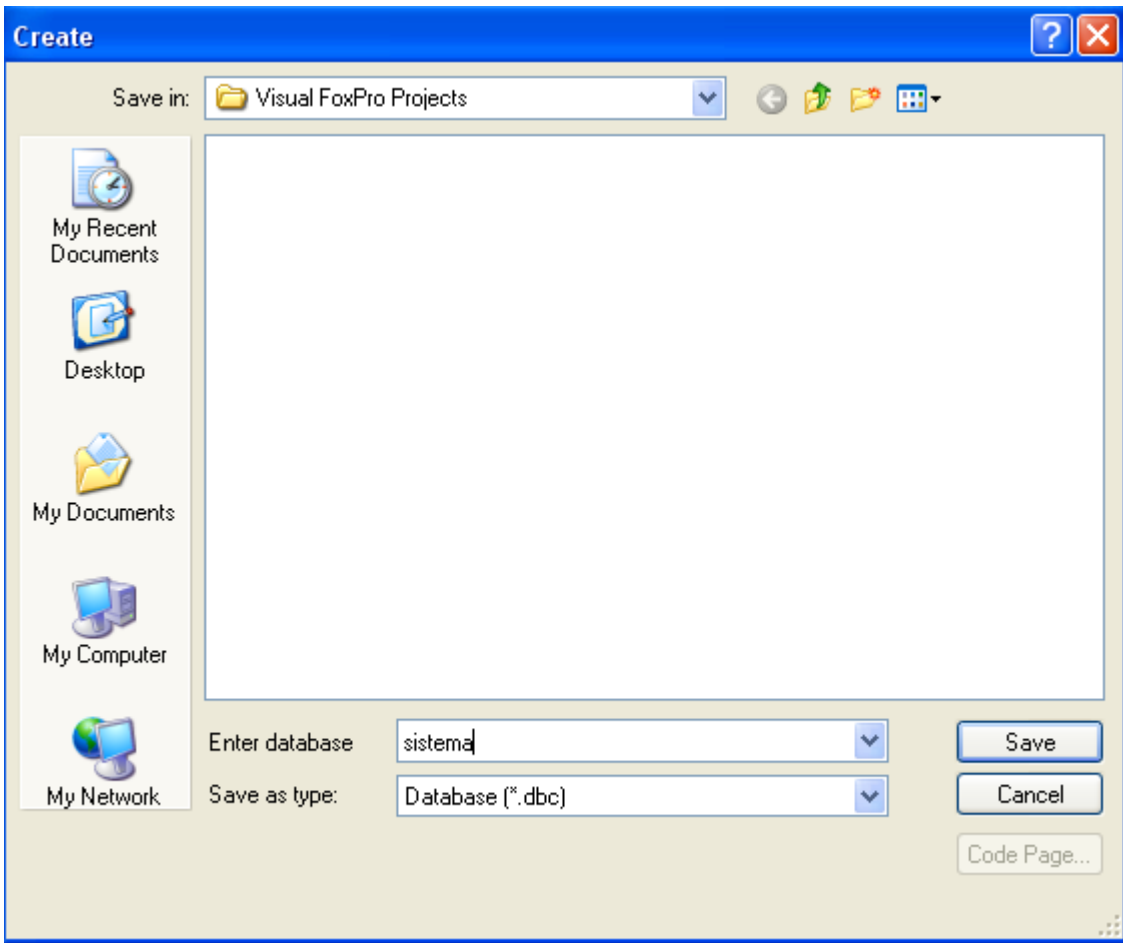


Figura 11. Creación de la base de datos “sistema.

Ahora estamos listos para empezar a trabajar y definir nuestras tablas dependientes en nuestro contenedor de tablas llamado “**sistema**”, **RECORDAMOS QUE EN CAPITULOS MAS ADELANTE**, tablas es el nombre con que se designa al archivo que en forma física se guardan los datos de nuestro sistema, por ejemplo el de clientes, proveedores, etc. detallaremos todo sobre bases de datos y tablas dependiente y libres. Manos a la obra ahora estamos con el diseñador de base de datos y podemos comenzar a crear nuestras tablas para nuestro proyecto. A partir de este momento tendremos en pantalla la imagen de la **figura 12** que nos permitirá crear, agregar, modificar nuestras tablas.

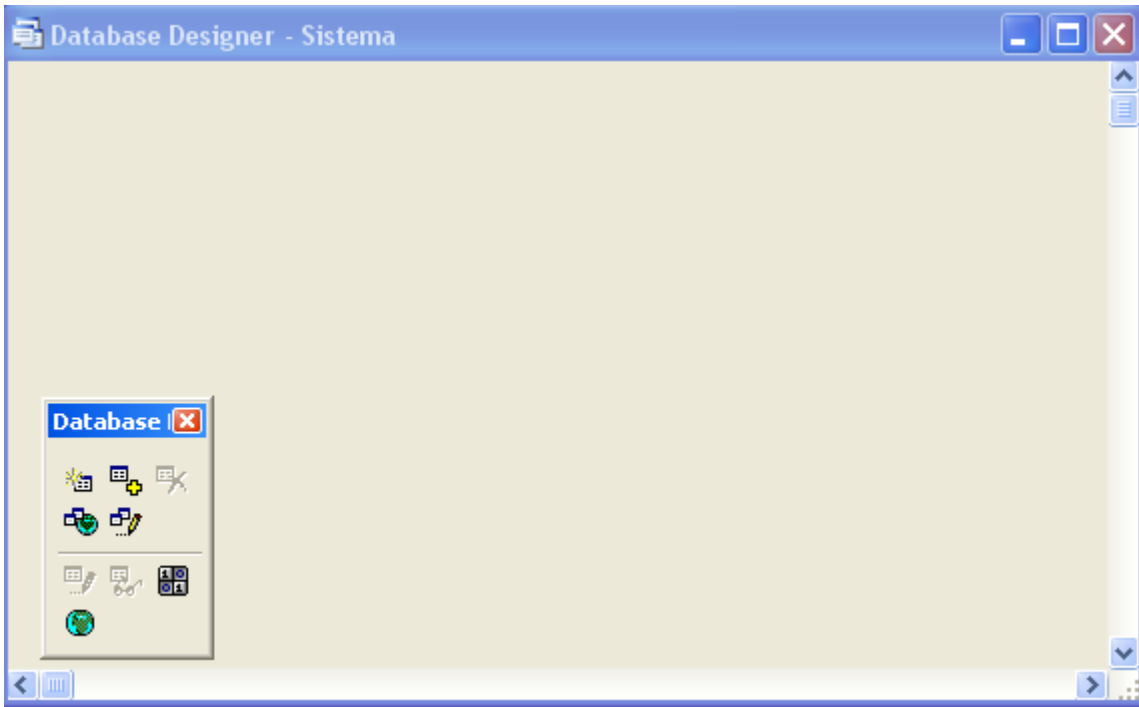



Figura 12. Creación de tablas

Ahora tenemos dos opciones podemos hacer clic con el lado derecho del Mouse y elegimos la opción de New Table o bien hacer clic en  para crear nuestra primera tabla, hagamos clic en cualquiera de estas dos opciones y estaremos en condiciones de empezar a definir los campos que serán de nuestra utilidad para el desarrollo de nuestra aplicación. Ahora debemos escoger la opción de New tabla como lo muestra la **figura 13**.

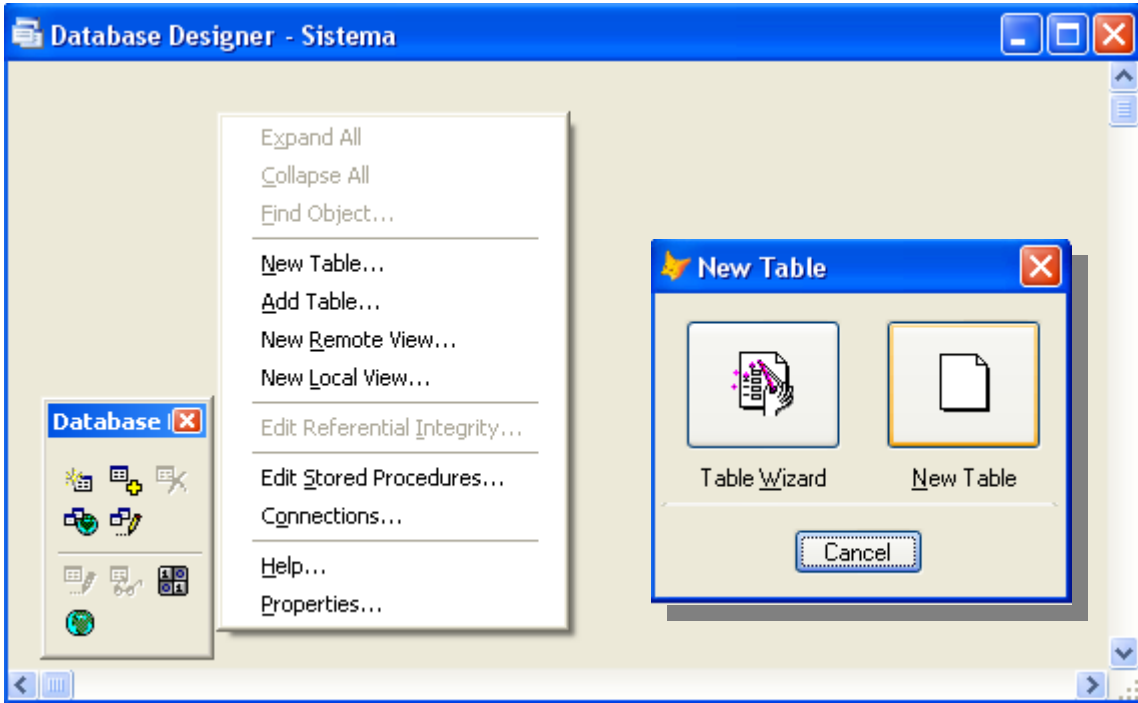


Figura 13. Creación de tablas

Ahora es el momento de introducir el nombre de nuestra tabla, la llamaremos “**clientes**” y hacemos clic en guardar. Ahora debemos designar cuales serán nuestros campos a utilizar para nuestra tabla, solamente utilizares a modo de ejemplo unos nombres de campos básicos tales como: idcliente, nombres, dirección, teléfono, ciudad, provincia, email.

Los campos a completar serán como los muestra la **figura 14**. En la definición del tipo de campo nos limitaremos a definirlos a todos como carácter a modo de ejercicio practico y en el ancho podemos aumentar o disminuir según nuestras necesidades, tengamos en cuenta que al aumentar el ancho de los campos y ser estos muy extensos se superpondrán unos sobre otros o la información se “cortara” al no alcanzar el ancho del tamaño de la hoja en la cual debemos imprimir el informe. Luego hacemos clic en OK y estaremos listos para introducir nuestros datos.

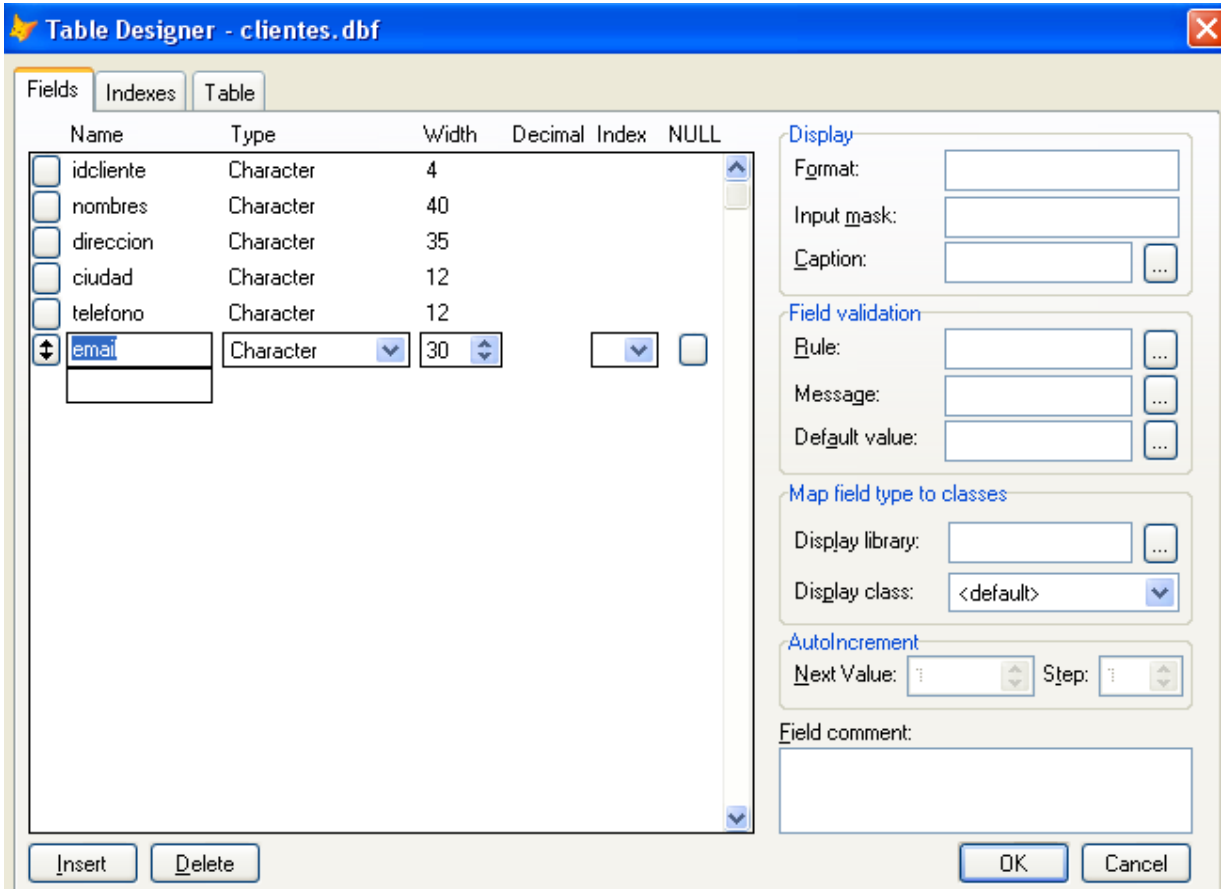


Figura 14. Diseñador de Tablas

Ahora el paso a seguir es muy fácil, una vez hecho clic en botón OK, estaremos de nuevo en el diseñador de bases de datos, lo cerramos y ahora estamos listos para diseñar nuestro primer formulario para la introducción de datos.

Hacemos clic en caso de ser necesario en la solapa **Docs**, y luego en clic en Form y por ultimo clic en New y luego en **Form Wizard** (Asistente de Formularios). Esto esta graficado en la **figura 15**.

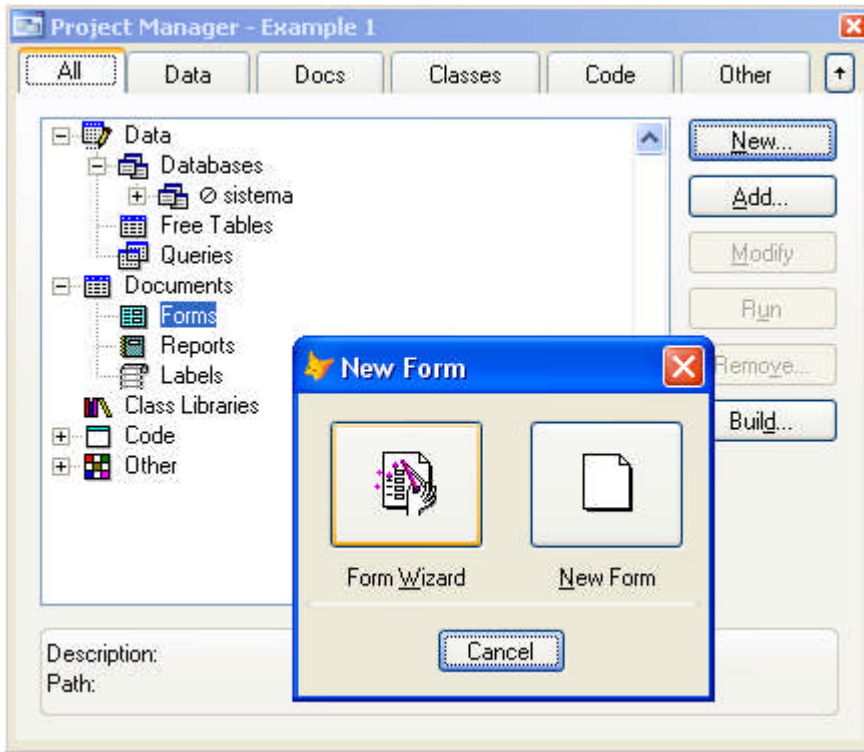



Figura 15. Asistente de Formularios.

A partir de este momento comenzamos a diseñar nuestro formulario. Escogemos la primera opción de Form Wizard y presionamos OK. Tal como lo muestra la figura 16.



a partir de ahora debemos hacer clic en todos los next (siguientes) a fin de obtener nuestro tan querido formulario. Pero antes vamos a mostrar una parte importante antes de hacer clic en todos los next.

Figura 16. Asistente de Formularios

Ahora debemos hacer clic en la flecha doble  que señala hacia la derecha para que todos los campos sean seleccionados. Aquí vemos un detalle muy importante que es que en forma automática ha seleccionado la base de datos **SISTEMA** y la tabla **CLIENTES**, esto se ha hecho posible ya que al tener activado el proyecto “**example 1**” las base de datos con todas sus tablas están en memoria y en la carpeta “Visual FoxPro Projects” que es donde estamos trabajando como lo muestra la **figura 17**.

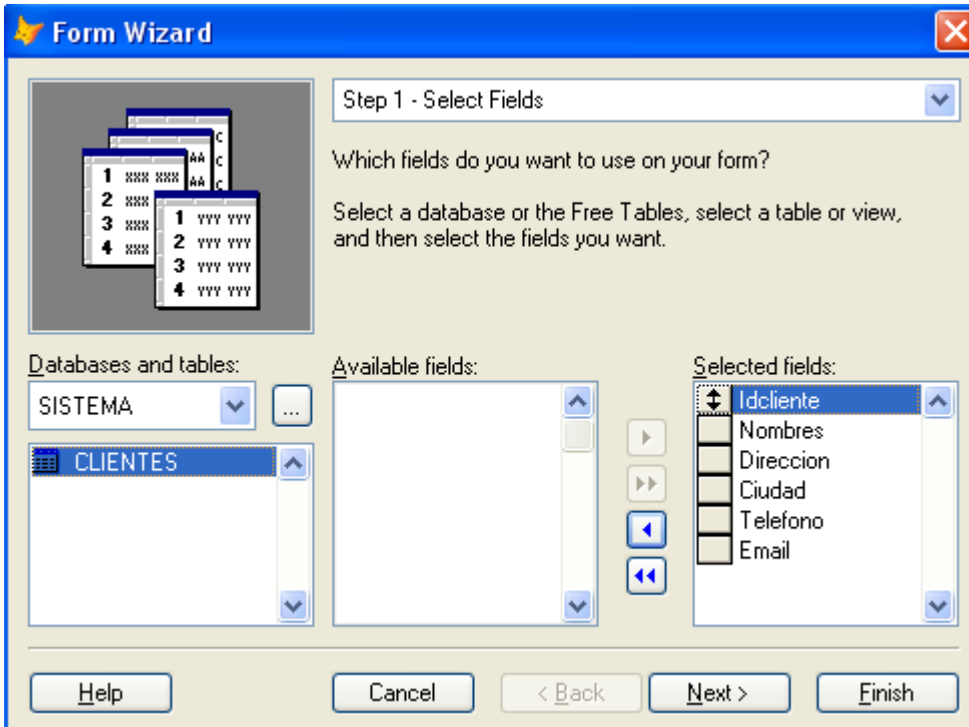
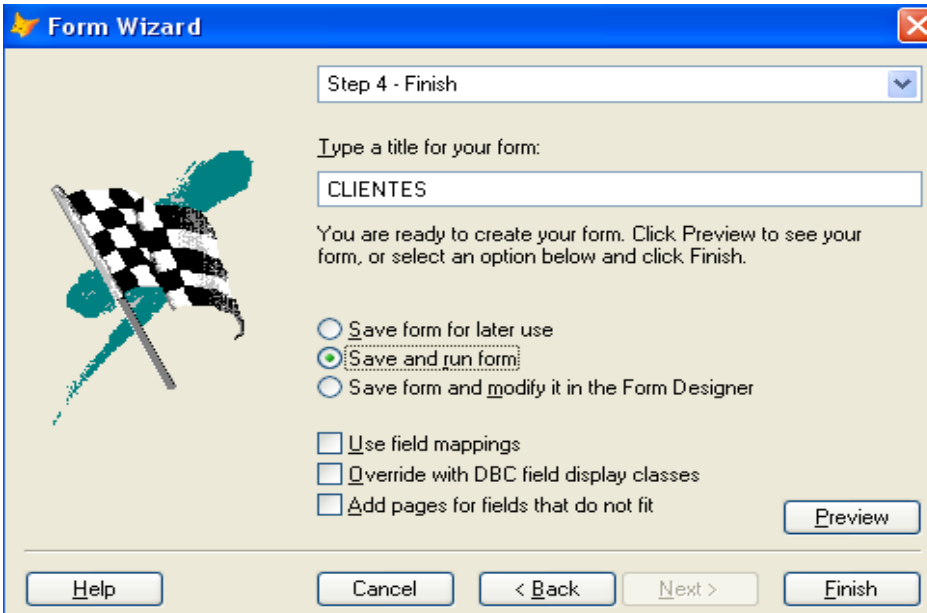


Figura 17.
Asistente de
Formularios

De aquí en mas solo debemos hacer clic en Next, hasta llegar al Paso 4 en que debemos seleccionar la opción **save and run form** y luego hacemos clic en Finish como lo muestra la **figura 18**. Al final se nos pedirá un nombre para guardar el formulario, lo llamaremos clientes, y haremos clic en guardar, si hemos hecho todos los pasos en forma correcta el asistente nos sugerirá el nombre clientes. En lo posible debemos introducir por lo menos 10 datos de personas para verlos reflejados en el informe.



Form Wizard

Step 4 - Finish

Type a title for your form:

CLIENTES

You are ready to create your form. Click Preview to see your form, or select an option below and click Finish.

☐ Save form for later use
☒ Save and run form
☐ Save form and modify it in the Form Designer

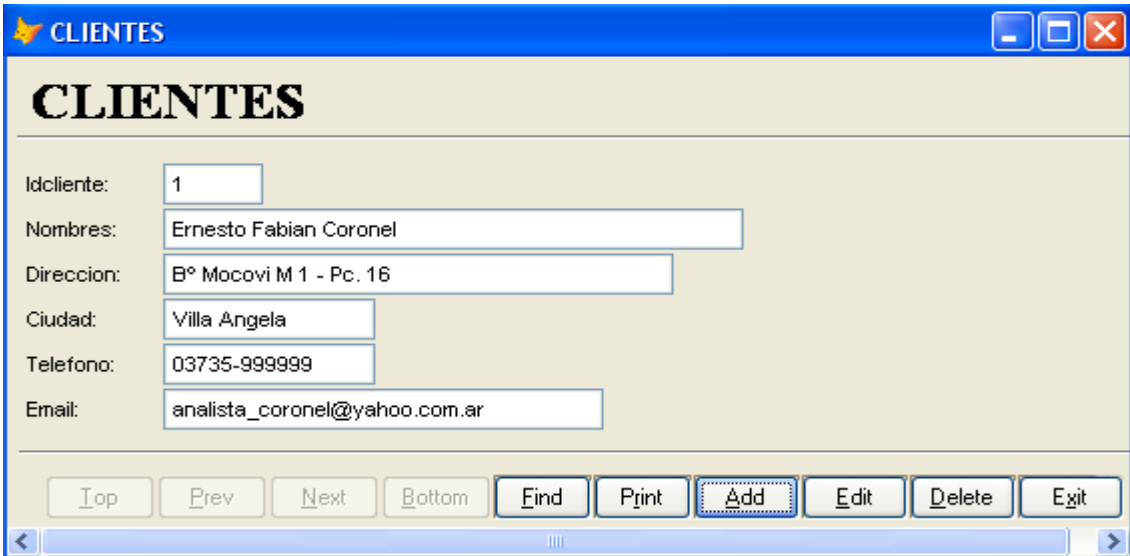
☐ Use field mappings
☐ Override with DBC field display classes
☐ Add pages for fields that do not fit

Preview

Help Cancel < Back Next > Finish

Figura 18.
Asistente de
Formularios

Aquí tenemos un ejemplo de nuestro primer formulario creado para la introducción de datos.



CLIENTES

Idcliente: 1

Nombres: Ernesto Fabian Coronel

Direccion: B° Mocovi M 1 - Pc. 16

Ciudad: Villa Angela

Telefono: 03735-999999

Email: analista_coronel@yahoo.com.ar

Top Prev Next Bottom Find Print Add Edit Delete Exit

Figura 19. Vista de un formulario creado con el asistente

Asistente para Informes Rápidos.

Con este ultimo ejemplo mostraremos como crear Informe con el asistente rápido, todos los pasos a ejecutar son similares al igual que en la creación de formularios.

Comencemos. Estando abierto el proyecto abierto en la solapa **All** nos situamos en la opción **Documents** y hacemos clic en **Report** y hacemos clic en la opción de **Report Wizard** y después hacemos clic en **Report Wizard** y se nos mostrara una pantalla similar a la **figura 20**.

A partir de aquí debemos prestar especial atención porque vamos a seleccionar los campos necesarios para emitir nuestro “informe” de acuerdo a nuestras necesidades.

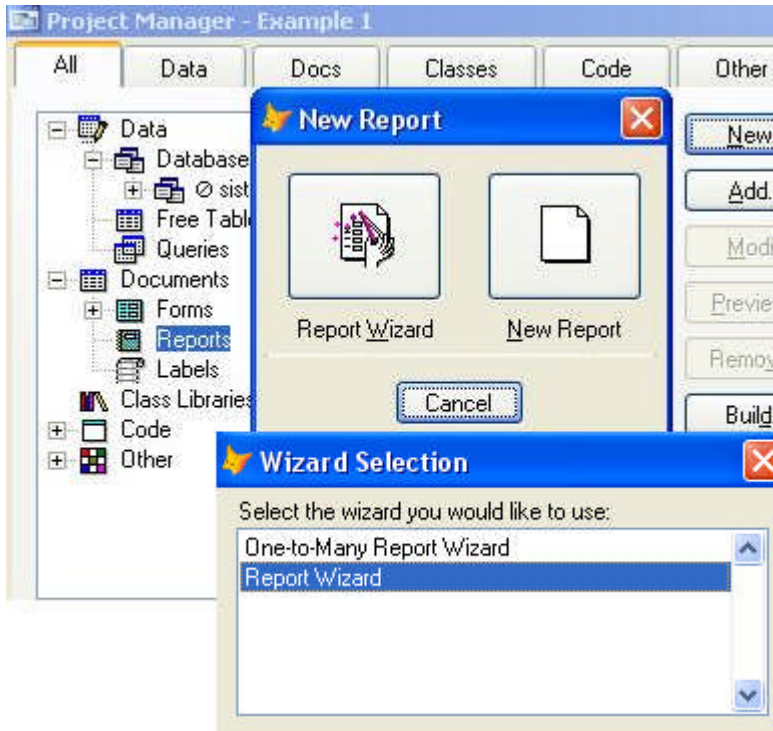



Figura 20. Asistente para la creación de informes.

Desde este momento nuestro desafío aumenta ya que debemos seleccionar los campos: nombres, dirección, ciudad, teléfono y nada más. Para esto utilizaremos la flecha de selección de campos.

Es muy posible que necesitemos abrir la tabla “cliente” para diseñar nuestro **informe**, entonces haremos lo siguiente.

En caso de ver una pantalla similar a la figura 21, en la se no este “activa” ninguna tabla, debemos hacer clic en el boton  que esta a la derecha de **Free Tables** y buscamos la tabla “clientes”. La seleccionamos y hacemos clic y aceptamos o directamente pulsamos doble clic sobre ella y de esta manera tendremos en memoria la tabla clientes activa que depende de la base de datos example 1.

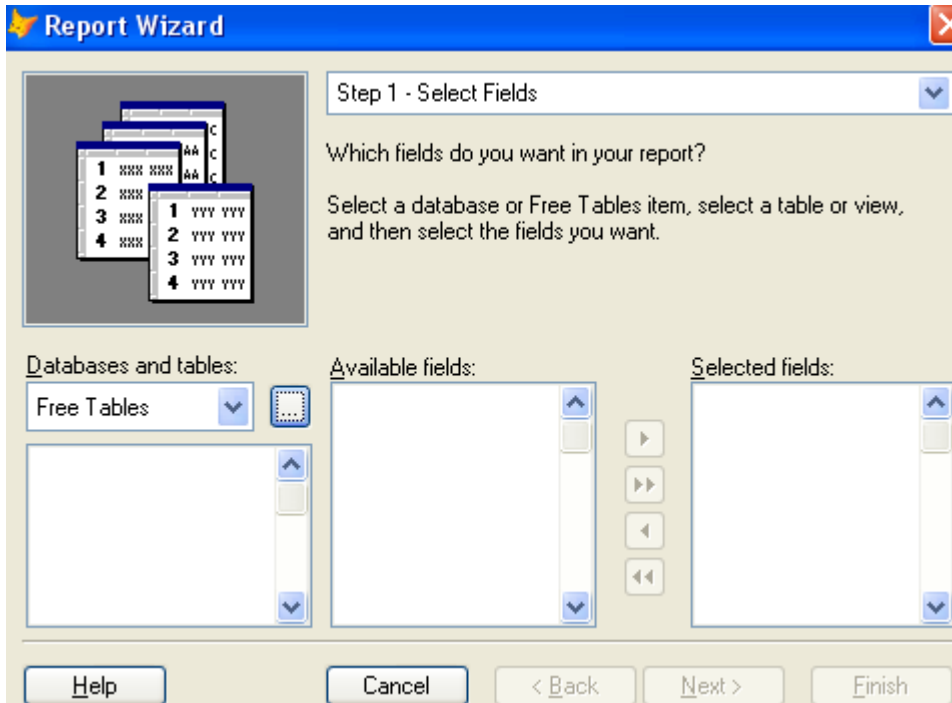



Figura 21. Diseño de informes.

Una vez hecho esto tendremos una figura similar a la figura 22 y demos seleccionar los campos necesarios para nuestro informe final y estos campos serán: nombres, dirección, ciudad, teléfono.

Esto se puede hacer seleccionando los Campos Disponibles con la flecha  y los campos que has sido seleccionado pasaran al lado derecho en el cuadro de “*campos seleccionados*” y de esta manera haremos clic en Next.

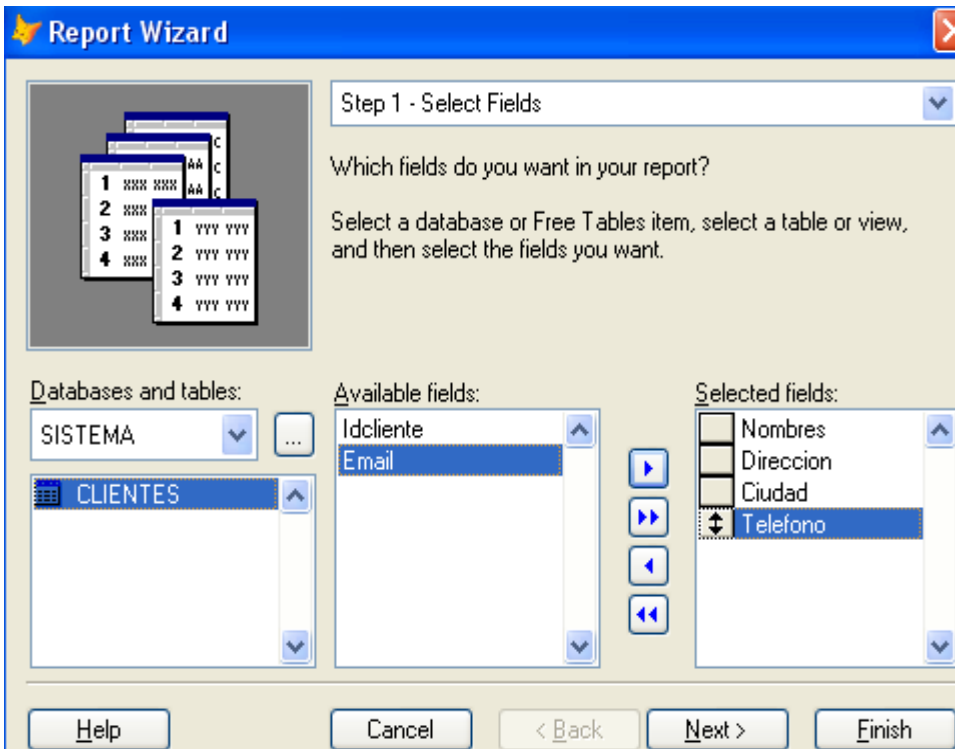


Figura 22.
Selección de campos.

Ahora se nos pedirá que seleccionemos por que grupo de registros lo queremos agrupar, seleccionamos **None** , y hacemos clic en Next > y tendremos una pantalla similar a la figura 23.

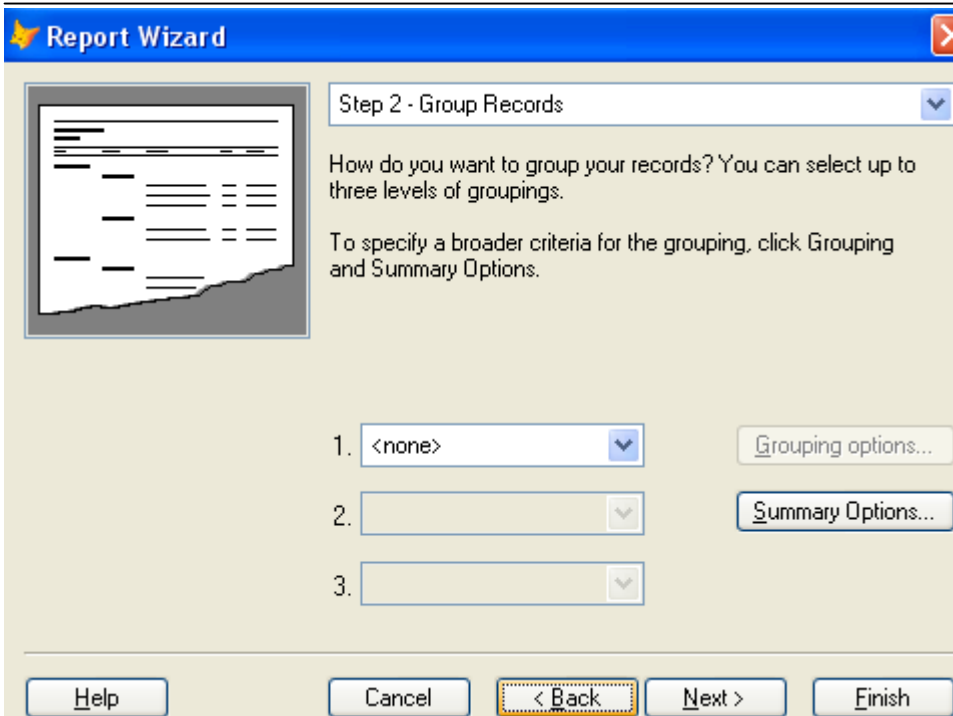


Figura 23.
Agrupamiento de registros.

Hacemos clic en Next > y en el paso 3 elegimos el estilo de reporte **estándar** y en el paso 4 debemos seleccionar en **definir el diseño de informe** el estilo **porta retrato**, pantalla representada en la figura 24. Hacemos clic en Next >

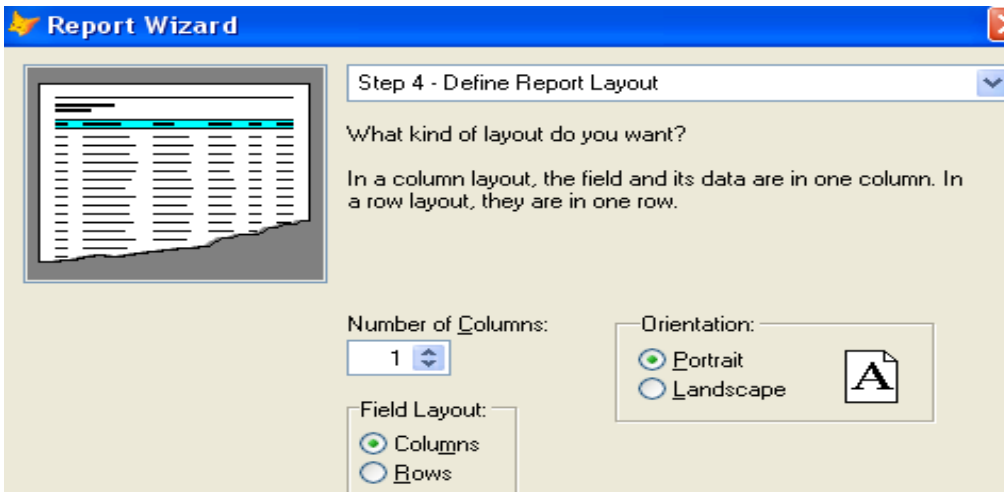


Figura 24.
Diseño de informe

Ahora estaremos en el paso 5 debemos optar por que campo ordenaremos los registros de nuestro informe, seleccionamos el campo idcliente y hacemos clic en **Add >** para que quede ordenado por el campo idclientes, hacemos clic en **Next >** y estaremos frente a la figura 25, por el cual el campo ya estará ordenado por el campo idclientes.

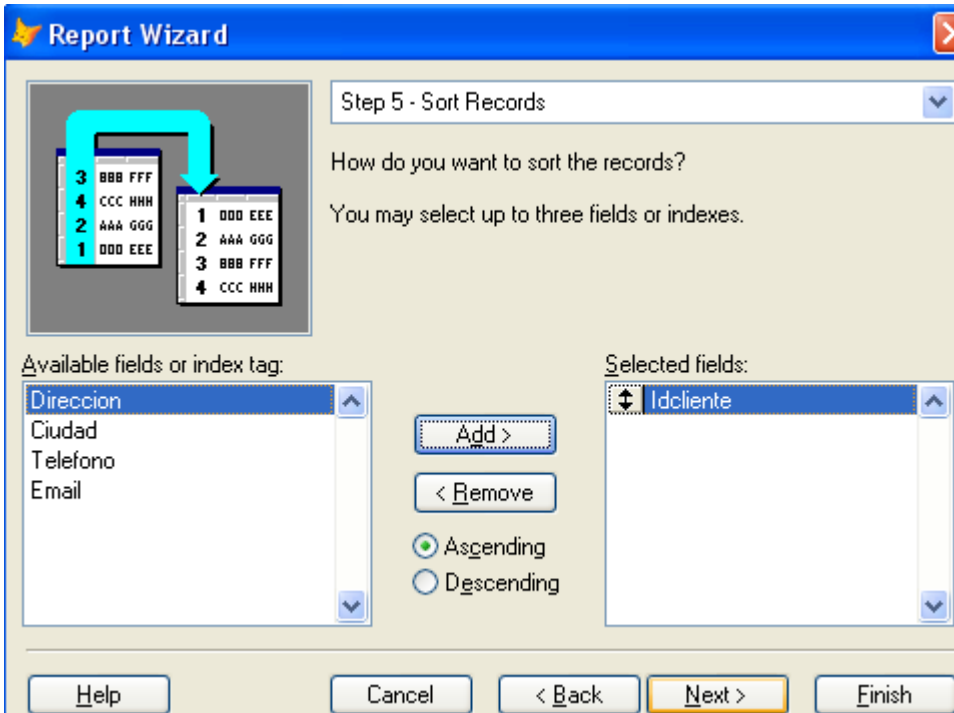


Figura 25.
Ordenación de
registros.

En este momento estamos en el paso 6 en el cual debemos hacer clic en la opción **guardar y ejecutar el informe**, clic en **Finish** para finalizar. De esta manera estaremos en la presencia de la figura 26. Se nos pedirá un nombre para nuestro informe, lo llamaremos **clientes** tal cual lo sugiere el sistema.

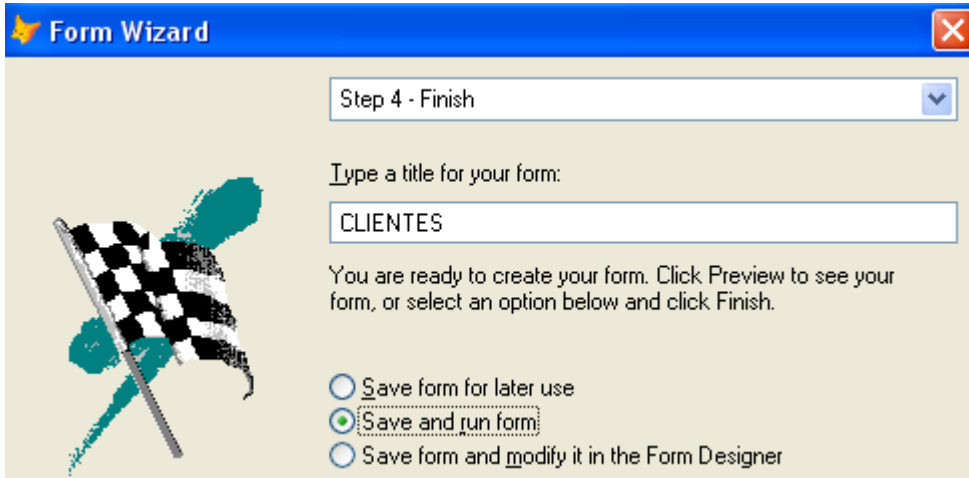


Figura 26.
guardar y
ejecutar el
formulario

Et Voila Monsieur !. Aquí tenemos un ejemplo de lo fácil que es diseñar sistemas en Visual FoxPro 9 sp2.



Bases de Datos y Tablas

Tabla: Conjunto de datos almacenados en un archivo DBF, si una tabla no forma parte de una Base de Datos se le denomina Tabla Libre. **Base de Datos:** Conjunto de Tablas relacionadas

En el mundo de la programación es muy común de hablar de Bases de Datos, ya existen diferentes lenguajes que utilizan este concepto lo cierto es que adaptado a VFP podríamos decir que constituye una BD una colección de tablas y demás elementos que iremos detallando detenidamente, esto quiere decir que la BD en VFP actuaría como contenedor de las tablas, sean estas libres o dependientes, entonces el termino tabla utilizaremos para designar a una archivo tipo DBF, que esto si seria una colección de registros. A modo de ejemplo, lo graficamos en la Figura 1, aquí tenemos a la BD con el nombre “Contenedor” que esta compuesta las tablas: empresa, clientes, proveedor, tpfiscal, compfac, compras, ventas.

También existen 2 vistas locales: ivdh y icdh

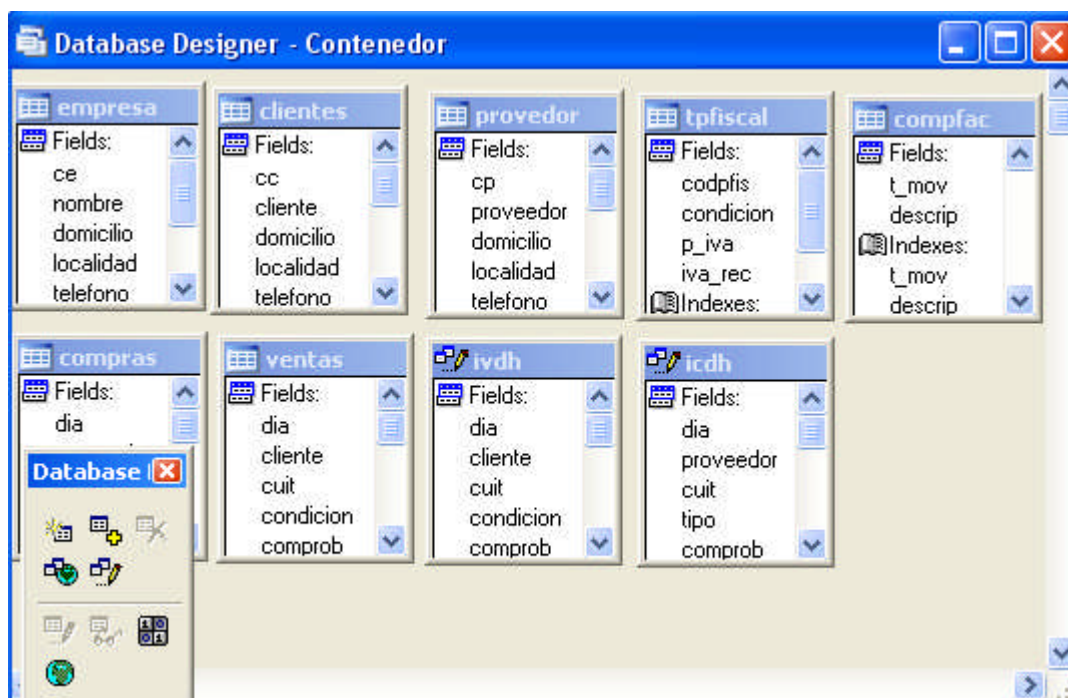


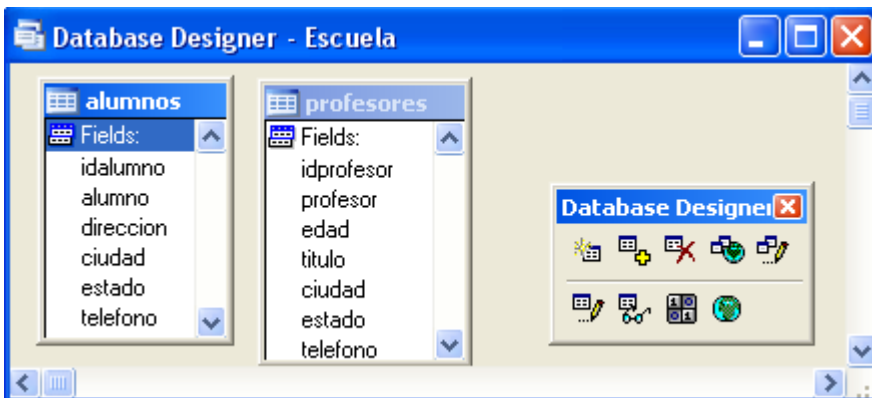
Figura 1. Base de Datos con tablas y dos vistas locales.

CONTENIDO DE UNA BASE DE DATOS.

De acuerdo a lo que estamos describiendo, mencionaremos detallaremos los elementos mas importantes que conforman una base de datos en VFP

- Tablas
- Relaciones
- Integridad Referencial
- Procedimientos almacenados
- Consultas
- Vistas locales y remotas

Al crear una BD, como es fácil de advertir, la BD estará completamente vacía y por lo tanto debemos ir creando nuestras tablas para almacenar nuestra información, el nombre de las tablas deberán ser nombradas de acuerdo a la finalidad de la misma, como por ejemplo si creamos una tabla que contendrá los datos de los alumnos profesores de una escuela, lo lógico es que una tabla se le nombre como “*alumnos*” y a la otra “*profesores*” y a la BD “*escuela*”



COMANDOS E INSTRUCCIONES PARA EL MANEJO EN BASES DE DATOS.

Existe dos formas de trabajar con las instrucciones en BD, una es a través de los menús contextuales que dispone VFP, que es la mas amigable e intuitiva, o podemos escribirlos directamente en la ventana Command, muchos programadores que trabajan desde versiones antiguas de VFP, hacen casi todo desde la ventana Command, pero para los nuevos programadores pueden optar por un método u otro.

Create Database

Creamos una BD nueva y se activa automáticamente. Desde la ventana Command , podriamos escribir por ejemplo CREATE DATABASE **escuela**. Desarrollemos un ejemplo mas elaborada desde la *Ventana Comandos*, escribiremos lo siguiente para crear la BD “escuela” y una tabla “materias”

```
CREATE DATABASE escuela
```

```
CREATE TABLE materias (CodeName C (3), DescName C (20))
```

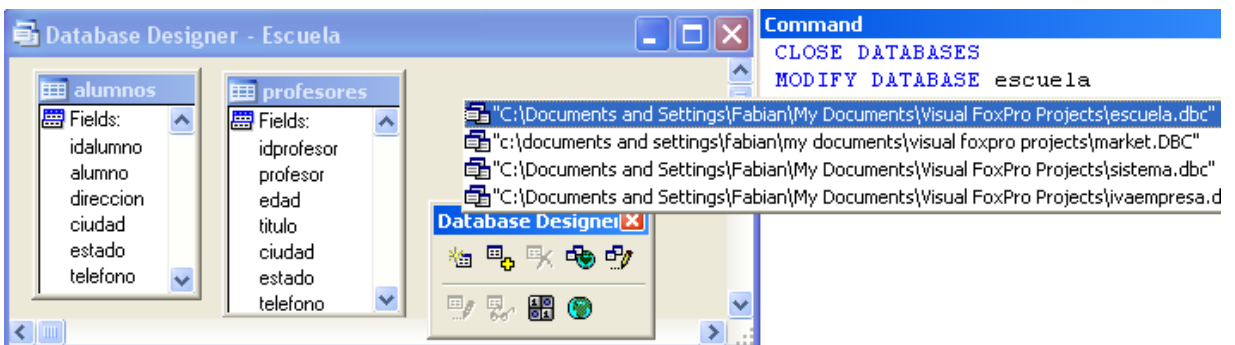
```
CLEAR
```

```
DISPLAY TABLES && despliega las tablas en la base de datos
```

```
DISPLAY DATABASES && despliega la información de las tablas incluidas en la base de datos
```

Modify Database

Abre la el generador de BD para modificar o introducir tablas nuevas por ejemplo, y ahí podemos ver todas las bases creadas desde VFP, en la cual se abre en un cuadro desplegable.



OPEN DATABASE

Nos permite abrir una BD creada previamente, recordemos que VFP abre las bases de datos en forma **SHARED** (compartida) esto significa que si necesitamos modificar la misma base de datos o tablas cuando estemos programando será necesario en algunos casos abrirla en forma **EXCLUSIVE** .

OPEN DATABASE [FileName | ?] [**EXCLUSIVE** | **SHARED**] [**NOUPDATE**] [**VALIDATE**]

FileName

Especifica el nombre de la base de dato para abrir. No es necesario especificar una extensión para el nombre de archivo, Visual FoxPro asigna Automáticamente la extensión .dbc

?

Despliega la caja del diálogo Abierto que usted puede escoger en una base de datos existente o puede entrar el nombre de un nuevo formulario para crear. Esto lo hemos visto al usar el comando **MODIFY DATABASE**.

EXCLUSIVE

Abre la base de dato en modo exclusivo. Si usted abre la base de datos exclusivamente, otros usuarios no podrán acceder y ellos recibirán un error si ellos intentan tener acceso. De todas formas podemos fijar con el comando **SET EXCLUSIVE** el estado de compartido o exclusivo.

SHARED

VFP abre las bases de datos en modo Compartido (shared). De esta forma todos los usuarios que estén conectados en red con nuestro sistema podrán usar nuestra base de datos. Una vez que distribuyamos nuestras aplicaciones para evitar errores de acceso a los datos podemos fijar el comando **SET EXCLUSIVE ON** para que todos los usuarios puedan acceder desde la red a la base de datos.

NOUPDATE

Especifica que los cambios no pueden ser hechos en la base de datos, en otras palabras, la base de dato es de solo lectura solamente. El contenido de la base de datos no será modificado ni tampoco sus tablas, esta es una utilidad a considerar para los usuarios novatos.

VALIDATE

Específica que VFP verifique si las referencias in la base de datos son validas. VFP chequea las referencias de los índices están disponibles en el disco. VFP también chequea los campos que contienen índices de etiquetas dentro de la tabla si están validos o corruptos.

SET DATABASE TO

Fija una base de datos determinada, cuando tenemos varias en memoria, esto también es posible utilizado

DELETE DATABASE

DELETE DATABASE DatabaseName | ? [DELETETABLES] [RECYCLE]

DatabaseName

Especifica el nombre de la base de datos para anular del disco. La base de datos que usted especifica no puede estar abierta. DatabaseName puede incluir el camino a la base de datos con el nombre de la base de datos. Es muy importante utilizar esta comando ya que elimina tanto la base de datos como las referencias que tengan las tablas contenidas en la base de datos, no es recomendable borrar desde el sistema operativo ya las tablas contendrán la información de las cabeceras que apuntan a la base de datos, esto quiere decir que al intentar abrir las tablas que están contenidas dentro de base de datos dará un error.

PACK DATABASE

Luego de eliminar una tabla de la BD, queda la referencia de esa tabla, que la hemos borrado en forma lógica, de la misma manera que con los registros de una tabla, tenemos la posibilidad de “empaquetar” los datos y borrar físicamente las tablas.

?

Despliega el cuadro de dialogo desde donde usted puede elegir la base de datos a eliminar.

DELETETABLES

Elimina las tablas contenidas en la base de datos desde el disco y la base de datos.

RECYCLE

Especifica que la base de datos no se borra inmediatamente del disco y se la envía a la Papelera de reciclaje.

VALIDATE DATABASE [RECOVER] [NOCONSOLE] [TO PRINTER [PROMPT] TO FILE FileName]

RECOVER

Despliega el cuadro de dialogo para localizar las tablas e índices que no encuentran ubicados dentro de la base de datos. Comenzando desde la versión de VFP 7, la cláusula es soportada en los programas.

NOCONSOLE

Suprime el mensaje de salida en la ventana principal de VFP o la ventana activa definida por el usuario.

TO PRINTER [PROMPT]

Envía el mensaje de salida de error desde VALIDATE DATABASE hacia la impresora

PROMPT despliega el cuadro de dialogo antes de iniciar la impresión. Ubique el parámetro inmediatamente después de **TO PRINTER**.

TO FILE *FileName*

Envía la salida del mensaje de error para un archive especificado con *FileName*. Si el archivo existe y **SET FAFTY** esta en **ON**, se le preguntara si usted quiere sobrescribir el archivo.

CLOSE DATABASE

CLOSE [ALL | ALTERNATE | DATABASES [ALL] | DEBUGGER | FORMAT | INDEXES
| PROCEDURE | TABLES [ALL]]

Cierra la Base de Datos que esta en memoria y todas sus tablas, si establecemos el parámetro **ALL** cierra todas las bases datos abiertas.

Dbc()

Despliega el nombre de la base de datos en memoria y su ruta de acceso. No debemos olvidarnos de anteponer el ? antes del comando para que la podamos visualizar.

DbUsed()

Con esta función podemos saber si una Base de Datos esta en memoria.

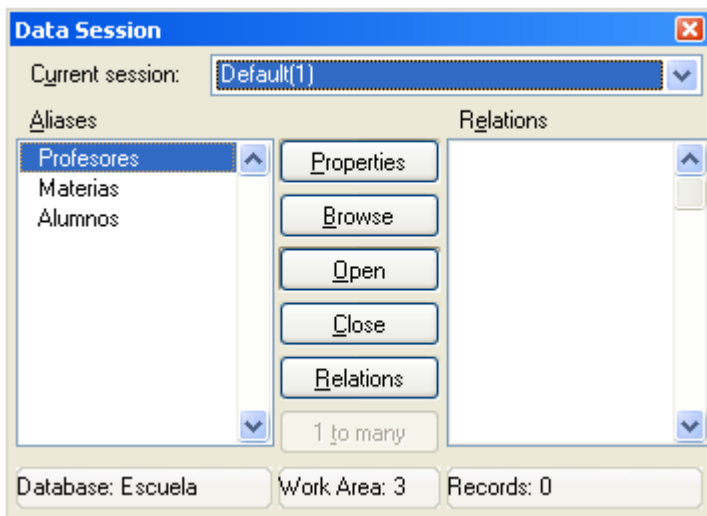
**DISPLAY DATABASE [TO PRINTER [PROMPT] | TO FILE *FileName* [ADDITIVE]]
[NOCONSOLE]**

Despliega la información sobre la base de datos.

CONCEPTO DE AREA.

Este es el momento estamos en condiciones, de comenzar a trabajar con tablas. Con el simple hecho de abrir una tabla, ya estamos accediendo a su información, esta acción de abrir una tabla cualquiera supone que se carga en un lugar de la memoria, en otras palabras en un área determinada. Podemos abrir tantas tablas como sean necesarias, el único limite es de 255 tablas, claro esta que difícilmente necesitemos mas 64 tablas por base de datos, esto ya es mas que suficiente.

Es muy fácil acceder a la información de todas las tablas, ya las mismas están accesibles en forma paralela. Podemos hacer esto con la ventana Sesión de Datos, como lo muestra la figura siguiente. A modo de comentario es necesario recordar que con VFP 9 SP2 podemos desarrollar nuestras aplicaciones para plataformas de Windows 98 en adelante, y nuestras aplicaciones podrían corren en Pcs con 64 Mb de R.A.M Y UN PROCESADOR DE 400 MHZ, hacer esto es casi imposibles con otros lenguajes de programación . Por lo tanto vale la pena programar en VFP 9 SP2 por lo escasos recursos que necesitamos en hardware.



Ventana Sesión de Datos.

TABLAS DEPENDIENTES Y TABLAS LIBRES.

Recordaremos que una tabla que se encuentra en una base de datos esta enlazada a ella. A esto es lo que llamamos ***tabla dependiente***. Por otro lado podemos crear la tabla fuera de ella, lo que la convertiría en ***tabla libre***. Las diferencias entre ellas son abismales, ya que las ***tablas dependientes*** disponen de mayores prestaciones tanto a nivel de campos y la misma base de datos. En lo posible debemos trabajar con tablas dependientes, pero esto queda a criterio del la experiencia y gusto del programador ya que en la actualidad todavía existen muchos programadores de VFP que prefieren trabajar con tablas libres.

Detallaremos las principales prestaciones que de las ***tablas dependientes*** en una base datos.

- Nombres largos para la tabla y los campos
- Propiedades de campos
- Propiedades de registros
- Triggers
- Índices principal y candidatos
- Relaciones permanentes
- Integridad referencial
- Procedimientos almacenados

Se amplia las prestaciones eliminando las antiguas limitaciones que teníamos con las tablas libres, ocho mas tres de extensión para el nombre de las tablas y diez posiciones para nombres de campo. Si por algún motivo especial necesitáramos hacerlas a estas tablas libres, lo ideal seria que mantuviéramos los ocho nombres para las tablas y diez para los campos, de esta manera evitaremos que los nombres se trunquen y al pasar el tiempo es difícil recordar nombres largos.

Al crear una tabla dependiente, tendremos una pantalla similar a esta ventana:

Table Designer - alumnos.dbf

Fields | Indexes | Table

Name	Type	Width	Decimal	Index	NULL
↑ idalumno	Character	4			
alumno	Character	35			
direccion	Character	30			
ciudad	Character	10			
estado	Character	2			
telefono	Character	10			

Display

Format:

Input mask:

Caption: ...

Field validation

Rule: ...

Message: ...

Default value: ...

Map field type to classes

Display library: ...

Display class: <default> ▾

AutoIncrement

Next Value: Step:

Field comment:

Insert Delete OK Cancel

Automáticamente se activan las propiedades a nivel de campo y tabla tal como aparecen en la siguiente figura.

Table Designer - alumnos.dbf

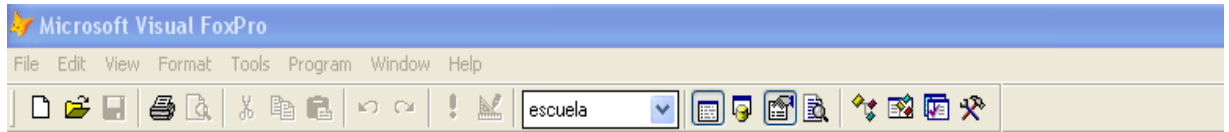
Fields | Indexes | Table

Order	Name	Type	Expression	Filter	Collate
↑	idalumno	Regula	idalumno		Machine

Primary
Candidate
Binary
Regular

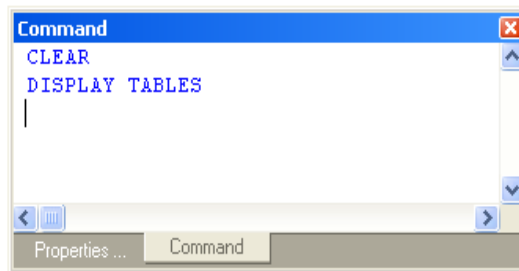
COMANDOS Y PARAMETROS EN EL MANEJO DE TABLAS DEPENDIENTES.

Estos comandos deberán ser tipiados en forma manual desde la ventana command, como lo muestra la figura siguiente.



Tables in Database ESCUELA:

Name	Source
alumnos	c:\documents and settings\fabian\my documents\visual foxpro projects\alumnos.dbf
profesores	c:\documents and settings\fabian\my documents\visual foxpro projects\profesores.dbf
materias	c:\documents and settings\fabian\my documents\visual foxpro projects\materias.dbf
tablalibre	c:\documents and settings\fabian\my documents\visual foxpro projects\tablalibre.dbf
factura	c:\documents and settings\fabian\my documents\visual foxpro projects\factura.dbf



Display Tables y List Tables

DISPLAY TABLES [TO PRINTER [PROMPT] | TO FILE FileName [ADDITIVE]] [NOCONSOLE]

LIST TABLES [TO PRINTER [PROMPT] | TO FILE FileName [ADDITIVE]] [NOCONSOLE]

Permite visualizar las tablas que se encuentran dentro de la base de datos, así como su ruta de acceso.

Add Table

ADD TABLE TableName | ? [NAME LongTableName]

Agrega una tabla libre a una base de datos, podemos modificar el nombre de la tabla para que sea mas largo agregando la cláusula, **NAME** seguido del nuevo nombre.

Remove Table

REMOVE TABLE TableName | ? [DELETE] [RECYCLE]

Elimina una tabla de la base de datos que la contiene y la convierte en una tabla libre, esto significa que perderemos todas sus propiedades, índices principales y las relaciones que pudieran existir con otras tablas.

Si por alguna necesidad deseamos borrar la tabla no solamente de la base de datos sino que tambien del disco rígido, debemos anteponer el parámetro **DELETE**.

Free Table

FREE TABLE TableName

Elimina la referencia de la cabecera de la tabla contenida en la base de datos. Aclaremos que la tabla permanecerá en la base de datos, pero esta vez como libre y no dependiente.

Drop Table

Elimina una tabla de la base de datos y la borra del disco. Incluyendo el argumento **NORECYCLE** evitamos que quede en la papelera de Windows.

InDbc()

INDBC(cDatabaseObjectName, cType)

Devuelve verdadero si el objeto por el se pregunta esta contenido dentro de la base datos active. Se debe indicar como segundo parámetro el tipo de objeto que es, **TABLE**, **INDEX**, **FIELD**, **VIEW** o **CONNECCTION**.

COMANDOS GENERALES EN LA OPERACION DE TABLAS.

Al tener abierta una BD o cuando necesitemos usar tablas libres, utilizaremos las siguientes relaciones de instrucciones. Para los programadores de versiones tipo Xbase les resultara muy familiar todo esto, por lo cual VFP aun las usa por cuestiones de compatibilidad con versiones anteriores.

Use

```
USE [[DatabaseName!] TableName | SQLViewName | ?]  
[IN nWorkArea | cTableAlias] [ONLINE] [ADMIN] [AGAIN]  
[NOREQUERY [nDataSessionNumber]] [NODATA]  
[INDEX IndexFileList | ? [ORDER [nIndexNumber | IDXFileName  
| [TAG] TagName [OF CDXFileName] [ASCENDING | DESCENDING]]]]  
[ALIAS cTableAlias] [EXCLUSIVE] [SHARED] [NOUPDATE]  
[CONNSTRING cConnectionString | nStatementHandle ]
```

No necesitamos nada mas que el nombre de la tabla anteponiendo el comando **USE** , se carga la tabla, junto con sus índices en caso de existir, en el primer área disponible para poder trabajar. Si necesitamos que la tabla se abra en un área determinada debemos incluir la cláusula **IN** seguido del número de área. Si queremos que VFP asigne la primera área no asignada, pondremos el valor 0 en el numero de área. Ejemplo:

```
CLOSE DATABASES  
OPEN DATABASE (HOME(2) + 'Data\Data')  
ACTIVATE WINDOW View && activa la ventana Sesión de datos  
USE alumnos IN 0 && abre la tabla alumnos  
USE profesores IN 0 && abre la tabla profesores  
USE materias IN 0 && abre la tabla
```

Select

SELECT nWorkArea | cTableAlias

Especifica un área de trabajo para activar. Por defecto el área que activa VFP

Para cambiar de area de debemos acompañar al **SELECT** con el numero de area, o bien escribir el alias de la tabla.

Para los programadores que gustan de las ventajas de SQL, preferirán cambiar el comando **SELECT** por sentencias de SQL lo cual en algunas circunstancias es lo mejor.

```

CLOSE DATABASES
OPEN DATABASE (HOME(2) + 'Data\testdata')
SELECT 1      && area de trabajo 1
USE clientes  && abre la tabla clientes
SELECT 2      && area de trabajo 2
USE ordenes  && abre tabla ordenes
SELECT clientes      && Work area 1
BROWSE

```

Append Blank

Agrega uno o más registro al final de la tabla. Puede editar un nuevo registro con browse, change, o edit.

APPEND [BLANK] [IN nWorkArea | cTableAlias] [NOMENU]

El siguiente ejemplo muestra como agregar 10 registro con la cláusula for en un programa .prg

Este ejemplo es muy útil cuando necesitemos programar para una tienda o comercio cualquiera y los pagos se harán en cuotas o varios pagos por medio de una tarjeta de crédito.

En algunos casos es mejor usar comandos tipo SQL, **INSERT INTO**, ya que reemplaza al **APPEND BLANK** y **REPLACE**.

```

CLOSE DATABASES
CREATE TABLE curRandom (nValue N(3))
FOR nItem = 1 TO 10
APPEND BLANK
REPLACE nValue WITH 1 nItem
ENDFOR
? nValue

```

Replace

```
REPLACE FieldName1 WITH eExpression1 [ADDITIVE][, FieldName2 WITH
eExpression2 [ADDITIVE]] ... [Scope] [FOR lExpression1] [WHILE
lExpression2] [IN nWorkArea | cTableAlias][NOOPTIMIZE]
```

Reemplaza o actualiza la información de cada campo en una tabla. Su sintaxis es :

REPLACE campo with datos

CLOSE DATABASES

CREATE TABLE Random (cValue N(3)), (cLastnames C (30))

FOR nItem = 1 **TO** 10 **&& Append 10 records,**

APPEND BLANK

REPLACE cValue **WITH** 22 **&& Reemplaza todos los registros por el valor 22**

ENDFOR

CLEAR

LIST **&& Display the values**

Delete

Borra lógicamente el o los registros donde esa situado el puntero en la tabla. Podemos borrar selectivamente con los argumentos **FOR** o **WHILE**, mediante **DELETE ALL**, podemos borrar todos los registros. Podemos borrar registros que estén en otra área de trabajo mediante **IN**.

Se puede trabajar y visualizar los registros marcados para borrar mediante set delete off

CLOSE DATABASES

SET DELETE OFF & Activa los registros marcados para su eliminación

OPEN DATABASE (HOME(2) + 'Data\testdata')

USE random && Abre la tabla random

DELETE ALL ' && Marca para la eliminación

CLEAR

LIST FIELDS cValue FOR DELETED() && Lista registros marcados para borrar en caso de no este activo el SET DELETED OFF

Recall

Recupera un registro borrado lógicamente en la tabla seleccionada. Al igual que **DELETE** podemos recuperar todos los registros mediante **ALL** o seleccionarlos con **FOR** o **WHILE**.

```
CLOSE DATABASES
SET DELETE OFF & Activa los registros marcados para su eliminación
OPEN DATABASE (HOME(2) + 'Data\testdata')
USE random && Abre la tabla random
CLEAR
RECALL ALL
LIST FIELDS cValue,cLastnames FOR DELETED( ) && Lista registros marcados
para borrar en caso de no este activo el SET DELETED OFF
```

Pack

Borra físicamente todos los registros marcados con la marca de borrado y actualiza el archivo de índices y los archivos .FTP que contienen los campos memo. En ningún caso debemos utilizar **ZAP**, ya que no borraría los campos memos asociados a la tabla.

PACK [MEMO | DBF] [Tablename] [IN nWorkarea | cTableAlias]

```
CLOSE DATABASES
OPEN DATABASE (HOME(2) + 'Data\testdata')
USE random && Abre la tabla random
CLEAR
DELETE ALL
PACK
LIST FIELDS cValue,cLastnames
```

TABLAS EN MODO BROWSE

Permite visualizar los datos de una tabla y en caso de ser necesario, agregar, modificar y borrar. En la ayuda de VFP tenemos sobrados ejemplos de cómo utilizar este comando, lo cual recomendamos una lectura muy detenida del mismo ya que tiene muchas variantes en su implementación, a modo de ejemplo daremos un breve ejemplo del mismo.

```
BROWSE [FIELDS FieldList] [FONT cFontName [, nFontSize [, nFontCharSet]]]
[STYLE cFontStyle] [FOR lExpression1 [REST]] [FORMAT] [FREEZE FieldName]
[KEY eExpression1 [, eExpression2]] [LAST | NOINIT] [LOCK nNumberOfFields]
[LPARTITION] [NAME ObjectName] [NOAPPEND] [NOCAPTIONS] [NODELETE] [NOEDIT |
NOMODIFY] [NOLGRID] [NORGRID][NOLINK] [NOMENU] [NOOPTIMIZE] [NOREFRESH]
[NORMAL] [NOWAIT][PARTITION nColumnNumber [LEDIT] [REDIT]][PREFERENCE
PreferenceName] [SAVE] [TIMEOUT nSeconds] [TITLE cTitleText] [VALID [:F]
lExpression2 [ERROR cMessageText]]
[WHEN lExpression3] [WIDTH nFieldWidth] [WINDOW WindowName1]
[IN [WINDOW] WindowName2 | IN SCREEN] [COLOR SCHEME nSchemeNumber]
```

```
CLOSE DATABASES
USE c:\ivaempre\bases\clientes.dbf SHARED
SELECT 1      && area de trabajo 1
USE clientes  && abre  la tabla clientes
CLOSE DATABASES
BROWSE FIELDS cust_id:R, company NOEDIT
```

CAMPOS.

TIPOS DE CAMPOS.

VFP 9 tiene predefinidos los campos a utilizar, ya que disponemos de todos los tipos de campos necesarios para desarrollar nuestras aplicaciones.

Aquí tenemos un detalle de los principales campos usados en VFP 9. a efectos de tener una idea mas pura de los tipos de campos describimos aquí todos los tipos de datos transcritos en la ayuda de Visual FoxPro. Una recomendación para todos los usuarios de este libro es que como es muy fácil de advertir, que tengan un dominio en Ingles técnico, ya que siempre se mantiene la ayuda en su idioma puro y las páginas web, las mejores, en su mayoría están en idioma Ingles.

Visual FoxPro Tipos de Datos

Tipo de dato	Descripción	Tamaño	Rango
	Binary data of indeterminate length.		
<u>Blob</u>	Blob values are in a memo (.fpt) file. No code page translation is performed on Blob data.	4 bytes in a table	Limited by available memory and/or 2GB file size limit.
<u>Character</u>	Alphanumeric text For example, a customer address	1 byte per character to 254	Any characters
<u>Currency</u>	Monetary amounts For example, the price of an item	8 bytes	- \$922337203685477.5807 to \$922337203685477.5807
<u>Date</u>	Chronological data consisting of month, day, and year For example, an order date	8 bytes	When using strict date formats, {^0001-01-01}, January 1 st , 1 A.D to {^9999-12-31}, December 31 st , 9999 A.D.
<u>DateTime</u>	Chronological data consisting of month, day, year, hours, minutes, and seconds	8 bytes	When using strict date formats, {^0001-01-01}, January 1 st , 1 A.D to {^9999-12-31}, December 31 st , 9999 A.D., plus 00:00:00 a.m. to 11:59:59 p.m.

For example, date and time of arrival

Boolean value of True or False

Logical

For example, whether or not an order has been filled

1 byte

True (.T.) or False (.F.)

Numeric

Integers or decimal numbers

For example, the quantity of items ordered

8 bytes in memory; 1 to 20 bytes in table

- .9999999999E+19 to .9999999999E+20

Binary values.

Varbinary

Varbinary data is similar to **Varchar** data in that values do not include padding with zero (0) bytes. The length of the contained value is stored internally.

1 byte per hexadecimal value up to 255 total bytes

Any hexadecimal value

No code page translation is performed on **Varbinary** data.

Variant data can be any of the Visual FoxPro data types and the null value.

Variant

Once a value is stored to a variant, the variant assumes the data type of the data it contains.

See other data types.

See other data types.

Variants are designated with an *e* prefix in language syntax.

Además, FoxPro Visual proporciona tipos de datos que sólo se aplican a los campos en las tablas.

Visual FoxPro Tipos de Datos

Field type	Description	Size	Range
Character (Binary)	Any Character data that you do not want translated across code pages For example, user passwords stored in a table and used in different countries or regions.	1 byte per character to 254	Any characters
Double	A double-precision floating-point number For example, scientific data requiring a high degree of precision.	8 bytes	+/-4.94065645841247E-324 to +/-8.9884656743115E307
Float	Same as Numeric	8 bytes in memory; 1 to 20 bytes in table	-.9999999999E+19 to .9999999999E+20
General	Reference to an OLE object For example, a Microsoft Excel worksheet.	4 bytes in table	Limited by available memory.
Integer	Numeric value with no decimals For example, a line number in an order.	4 bytes	-2147483647 to 2147483647

<u>Integer (Autoinc)</u>	Same as Integer but also an automatically incrementing value. Read-only.	4 bytes	Value controlled by autoincrement Next and Step values.
<u>Memo</u>	Alphanumeric text of indeterminate length or reference to a block of data For example, notes about a phone call in a phone log.	4 bytes in table	Limited by available memory.
<u>Memo (Binary)</u>	Same as Memo except that memo field data does not change across code pages For example, a login script used in different countries or regions. Alphanumeric text.	4 bytes in table	Limited by available memory.
<u>Varchar</u>	Varchar is similar to Character except values in Varchar fields do not include padding with additional spaces. The length of the contained value is stored internally.	1 byte per character up to 254 total bytes	Any characters
<u>Varchar (Binary)</u>	Varchar type data that you do not want translated across code pages.	1 byte per character up to 254 total bytes	Any characters

Recordamos que los campos mas usados en cualquier aplicación son: carácter, numérico, lógico, fecha, general y memo, hacemos notar que el campo [Character \(Binary\)](#) es muy interesante para los campos que no serán traducidos a través de los códigos de pagina.

Debemos tener especial cuidado al concatenar variables, sobre todo al hacer comparaciones. A modo de ayuda ilustramos una breve tabla de conversiones.

En el próximo capítulo instrucciones básicas, detallaremos, funciones de conversión y comandos necesarios para una correcta programación en VFP.

PROPIEDADES DE CAMPO.

Es muy útil, al momento de fijar nuestros campos dentro de la tabla, así de esta manera podemos controlar el tipo de dato ingresado por el operador a fin de evitar errores por parte del usuario no experimentado, de todas maneras también podemos hacer todo esto de forma manual desde las propiedades de cada objeto, ya que esta manera podemos tener un control mas detallado sobre nuestras aplicaciones.

En los ejemplos prácticos que detallaremos en los próximos capítulos, mostraremos como usar las técnicas de validación desde los objetos. El programador podrá por decidirse por uno u otro método, ya que en algunos casos los resultados son los mismos.

Table Designer - clientes.dbf

Fields | Indexes | Table

Name	Type	Width	Decimal	Index	NULL
idcliente	Character	4		↑	
nombres	Character	40		↑	
direccion	Character	35			
ciudad	Character	12			
telefono	Numeric	12	0		
email	Character	30			

Display

Format:

Input mask:

Caption: ...

Field validation

Rule: ...

Message: ...

Default value: ...

Map field type to classes

Display library: ...

Display class: ...

AutoIncrement

Next Value: Step:

Field comment:

Insert Delete OK Cancel

Display (Mostrar)

En este apartado esta pensado para ver como aparareceran las propiedad del control al ejecutarse en un formulario.

Format: Es lo mismo que la propiedad **Format** del control **TextBox** del mismo nombre que permite especificar el tipo de información que el usuario puede incluir en el campo.

InputMask: equivale a la propiedad **InputMask** del control **TextBox**, la cual coloca en el control correspondiente un formato preestablecido de entrada de datos, el cual no será grabado en el campo, es a modo de ayuda para el usuario.

Caption: Valor que aparecerá por defecto en las propiedades del control asociado al campo en el formulario correspondiente. En este caso, se mostrara en la propiedad Caption.

Field Validation (Validacion de Campos)

Es muy útil para evitar que el usuario del sistema introduzca datos erróneos en el campo correspondiente, así también para establecer un valor por defecto.

Regla de validación: condición que examina cuando introducimos un nuevo valor en el campo correspondiente y se efectúa un movimiento a otro campo de la tabla.

Texto de validación: al no cumplirse la regla de validación desde aquí podemos emitir un mensaje.

Valor predeterminado: valor que tomara por defecto al crearse un nuevo registro.

Asignar tipo de campos a clases (Map field type to Classes)

Cuando creamos un formulario, cada campo esta asociado a un control o clase de base que VFP determinara por defecto. Las clases bases se trataran en un capitulo dedicado al mismo.

Librería de Clases: es la librería donde esta alojada la clase que queremos asociar al campo.

Clase: tipo de control que aparecerá por defecto al incluir el campo en un formulario.

PROPIEDADES DE REGISTRO. DESENCADENANTES.

Table Designer - clientes.dbf

Fields Indexes **Table**

Name:

Database: c:\documents and settings\fabian\my documents\visual foxpro

Statistics

Table file: c:\... \fabian\my documents\visual foxpro projects\clientes.dbf

Records: 3 Fields: 6 Length: 134

Record validation

Rule: ...

Message: ...

Triggers

Insert trigger: ...

Update trigger: ...

Delete trigger: ...

Table Comment:

OK Cancel

Aquí están definidas como propiedades de tabla, pero en realidad afectan directamente a los registros. Los conocidos en inglés como **Triggers** es un código que es ejecutado cuando se produce una inserción, una modificación o borrado de un registro. Este código no cabe en la ventana, por lo que debemos incluir aquí en nombre de la función que tendremos que definir en *procedimientos almacenados*, lugar donde VFP la buscare.

INDICES

Cuando creamos una tabla, podemos ordenar los datos para acelerar la obtención de datos mediante índices. Con los índices, puede procesar rápidamente los registros para mostrarlos, consultarlos o imprimir. También puede seleccionar registros, controlar si se introducen valores duplicados en un campo y admitir relaciones entre tablas. Esto hace en bases de datos de millones de registros podamos obtener informes en cuestión de segundos para emitirlos por pantalla o impresora.

VFP ofrece cuatro tipos de índice diferentes, cada uno con características propias:

- Principal
- Candidato
- Normal
- Único

Los *índices principales* aseguran que sólo se introducen valores únicos en un campo y determinan el orden en el que se procesan los registros. Puede crear un índice principal para cada tabla si la tabla está incluida en la base de datos. Si la tabla ya tiene un índice principal, agregue un índice candidato. Este índice es el que define el campo clave

Los *índices candidatos* también imponen valores únicos y determinan en qué orden se procesan los registros, como un índice primario. Puede tener varios índices candidatos por tabla en bases de datos y en tablas libres.

Los *índices normales* determinan el orden en que se procesan los registros pero permite que se introduzcan valores duplicados en un campo. Puede agregar más de un índice normal a una tabla. Es muy útil ya la mayoría del tipo **índices normales**.

Por compatibilidad con versiones anteriores, también puede crear un *índice único* que selecciona y ordena un subconjunto de registros según la primera aparición de un valor en el campo que especifique. Si quiere seleccionar registros de esta manera, es posible que quiera crear una consulta o una vista en lugar de ello.

Cuando deseamos

Ordenar los registros para aumentar la velocidad a la que se presentan, se consultan o se imprimen.

Índices a utilizar

Un índice normal, candidato o principal

Controlar la entrada de valores Duplicados en un campo y ordenar los registros

Un índice principal o candidato para una tabla de base de datos o un índice candidato para una tabla libre.

INTRODUCCIONES PARA USAR CON LOS INDICES

Index

Permite crear índices tipo .IDX y .CDX en las tablas. Los índices tipo .IDX no se utilizan y la utilidad de crear índices la podemos incluir dentro de un programa .prg, en el caso que necesitemos reconstruir los índices que estén corruptos en caso de cerrarse incorrectamente el sistema o haberse producido una falla eléctrica, o también cuando se produce una interrupción por corte de energía eléctrica.

Para nuestra comodidad es mejor crear los índices dentro del diseñador de tablas. La función que tienen los índices es de crear un archivo que apunta en forma directa a la tabla para ordenar a los mismos en forma virtual, permitiendo de esta manera un acceso mas rápido, siendo esto de gran utilidad para tablas “gigantes”. Indexar también puede ser usado como sinónimo de ordenado.

```
INDEX ON eExpression TO IDXFileName | TAG TagName [BINARY]
[COLLATE cCollateSequence] [OF CDXFileName] [FOR lExpression]
[COMPACT] [ASCENDING | DESCENDING] [UNIQUE | CANDIDATE] [ADDITIVE]
```

Ejemplo de como crear indices sobre la tabla clientes.

```
CLOSE DATABASES
USE Clientes
INDEX ON idcliente TAG cliente
INDEX ON direccion TAG direccion
CLEAR
DISPLAY STATUS
```

Set Order to Tag

Fija el orden de una tabla una vez que el archivo de índices esta abierto, por defecto las tablas se orden de acuerdo al primer índice increado, esto quiere decir que al crear el TAG será considerado el primer tag en caso de tener mas de uno, y para establecer el segundo tag por defecto solo debemos activarlo:

Set Order TO Tag *direccion*

Si necesitamos dejarlo en el orden físico inicial de la tabla, solamente debemos fijarlo de nuevo con:

Set Order TO

REINDEX

Reconstruye los índices de una tabla, lo cual la tabla debe estar abierta en forma exclusiva, se aconseja utilizarlo en el entorno de programación. Es mas útil utilizar *reindex* que borrar completamente el archivo de índices y crear todos sus tag, claro esta que esto depende del criterio de cada programador.

Relaciones Permanentes y Temporales

Recordemos brevemente que en VFP existen las **relaciones permanentes**, y que tienen las siguientes características:

Se establecen dentro de la BD

Relaciona índices, no campos.

La relación existe siempre que tengamos abierta la BD abierta

Permite al igual que access, utilizar integridad referencial con los índices, esto debe ser en el caso de la tabla *madre*, principal o candidato, “nunca normal”, y en la tabla *hija* pueden ser de los tres tipos, estableciendo la relación en *Uno a Uno* con los dos primeros índices y en *Uno a Varios* si el índice es normal.

La relación permanente se consigue arrastrando el índice de la tabla madre y soltándolo sobre el índice de la tabla hija con el que queramos relacionarlo. Por comodidad a la hora de programar o actualizar nuestros sistemas veremos que es más útil establecer relaciones temporales, ya que son más versátiles en todos los aspectos, sobre todo cuando necesitemos reconstruir los índices de nuestras relaciones por eventuales “caídas del sistema”.

Características de las relaciones temporales:

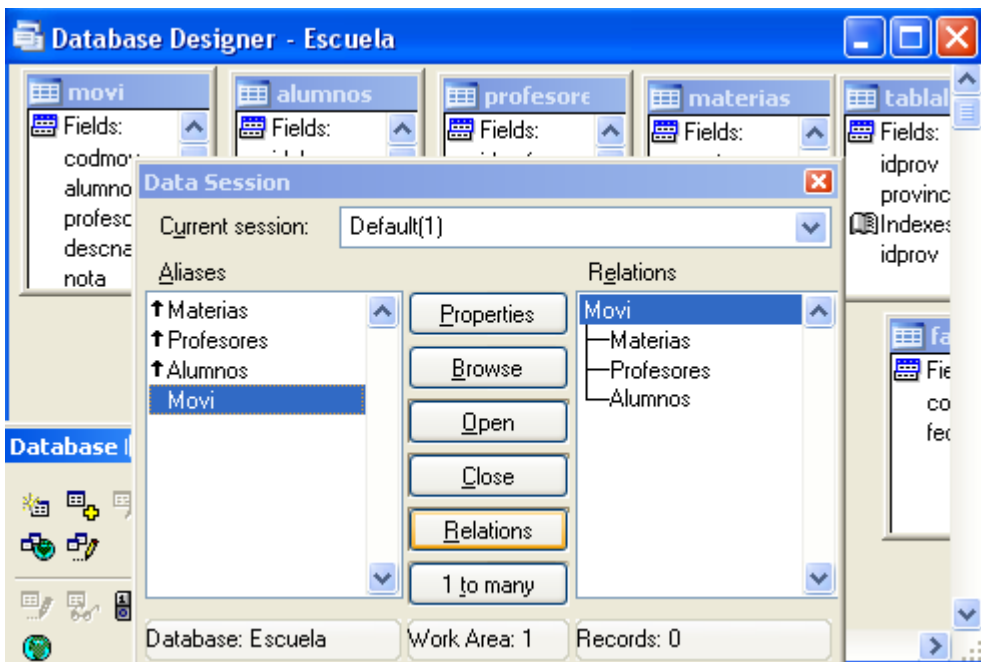
Se pueden crear sobre tablas dependientes o libres

Se relaciona un campo de la tabla madre con un índice de la tabla hijas.

No es necesario que la tabla madre este ordenada, ni que el índice sea de un tipo especial.

Es necesario que la tabla hija este ordenada por el índice coincidente con la expresión de la relación.

La relación se pierde cuando se cierra una de las tablas de la relación.



Establecimiento de la relación la podemos hacer mediante la ventana *Sesión de Datos* o por medio de una instrucción en la ventana de comandos, claro esta que es más simple desde la ventana *Sesión de Datos*. No permite utilizar integridad referencial.

Set Relation

Establece una relación temporal entre dos tablas, debemos colocarnos en el área de la tabla madre y luego tipiar.

SET RELATION TO alumno INTO alumnos

Claro esta como hemos mencionado, todo esto se resuelve, con la ventana *Sesión de Datos*.

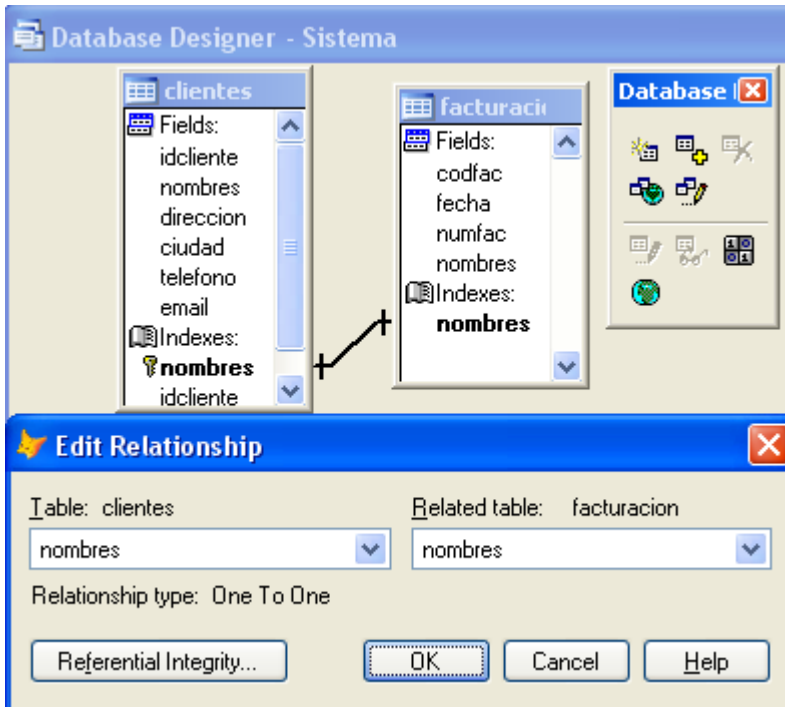
INTEGRIDAD REFERENCIAL.

Para todos los programadores de VFP de versiones anteriores, desde la versión 5 más exactamente, disponemos la posibilidad de trabajar al igual que Access, con integridad referencial. Debemos tener en cuenta que en la mayoría de casos, los programadores con experiencia en VFP optan por no utilizar integridad referencial, ya que en algunos casos puede complicarnos las cosas.

La integridad referencial consiste en que los datos contenidos en la tabla madre e hija en un relación no tengan incoherencias, esto es que sean concordantes entre si. Para esto debemos tener en cuenta que:

- 1) *Modificación del campo clave de la relación en la tabla madre*, si existen registros en la tabla hija con esa clave, habrá que cambiarlos igualmente, caso contrario tendríamos en la tabla hija registros sin padre.
- 2) Eliminación de un registro en la tabla madre, cuando eliminamos un registro de la madre, debe borrarse este y también todos los tengan la misma clave en la tabla hija, ya que de lo contrario, estaríamos en la misma situación descripta anteriormente.
- 3) Inserción de un nuevo registro en la tabla, al un registro nuevo en la tabla hija debemos comprar que exista un registro con la misma clave en la tabla madre, si no, estaríamos en la misma situación que en los casos anteriores, este registro estaría huérfano.

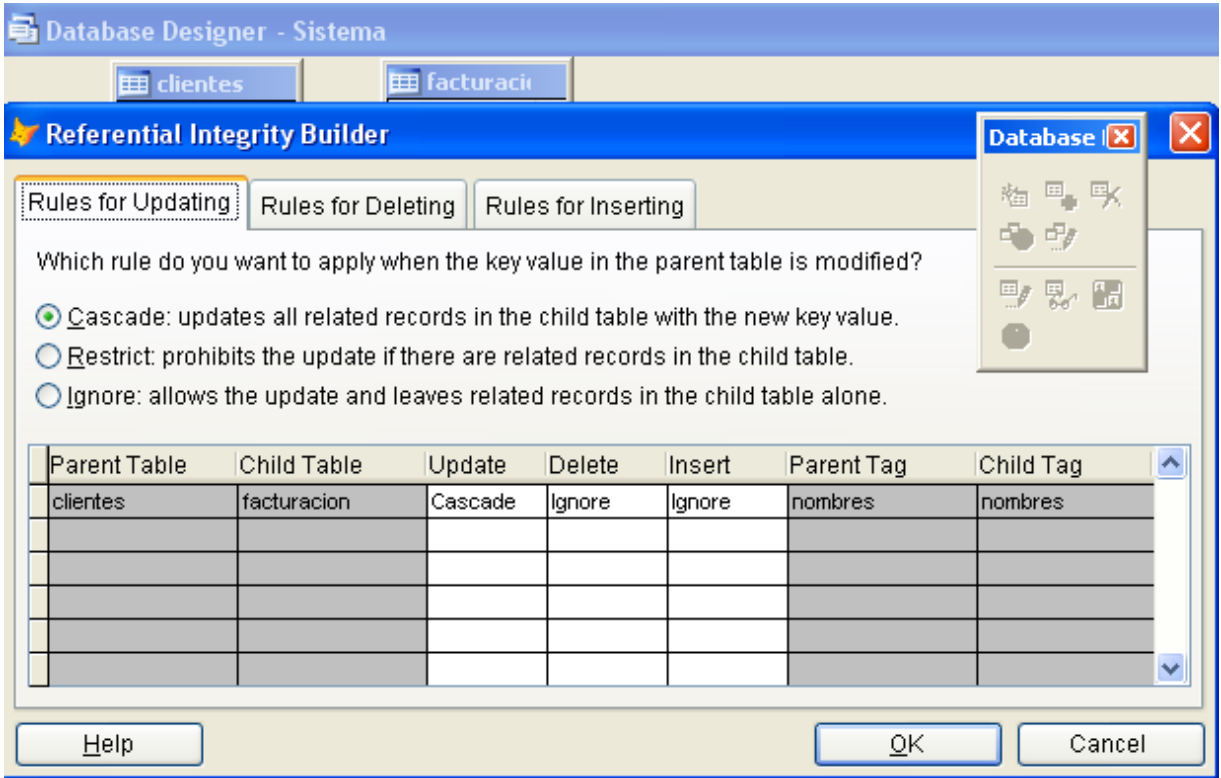
En la imagen de la próxima pagina, mostramos brevemente lo descrito en los tres ítems anteriores.



En estos tres ítems podemos ver como trabaja la integridad referencial en el mantenimiento de una relación. No nos olvidemos que la integridad referencial es posible dentro de la Base de Datos.

Haciendo doble clic sobre la línea que une a las dos tablas relacionadas podremos ver la ventana informativa de la relación que es muy similar a la Access.

Recordemos que para editar o modificar todas las opciones que nos ofrece la integridad referencial, la base de datos debe estar abierta forma exclusiva, ya que estamos trabajando sobre ella en modo de diseño. La ventana será similar a la que mostramos en la siguiente página.



A modo de recordar, debemos hacer notar que la tabla madre deberá tener un índice tipo Principal, y la tabla hija uno de tipo candidato, en caso de tener un índice del tipo normal, la integridad referencial simplemente no funcionara.

Nuestro consejo, que a modo de inicio para los “programadores juniors” no es muy aconsejable trabar con IR, ya que en algunos casos por falta de experiencia, termina complicando la programación, sobre todo cuando debemos actualizar los sistemas de nuestros clientes, ya que al tener mas de 20 tablas es difícil que recordemos todos los índices establecidos en nuestras tablas.

COMANDOS DE BUSQUEDAS EN TABLAS.

Después de que tenemos a las tablas indexadas, podemos acceder con rapidez a sus registros.

Skip

Avanza o retrocede dentro de los registros. Debemos controlar siempre que el avance o el retroceso no desborden el principio o el final de la tabla. Para ello contamos con dos comandos.

Bof()

Retorna (verdadero) .T. si el puntero del esta mas alla del primer registro, es decir en la cabecera.

Eof()

Retorna (verdadero) .T. al llegar al final del registro el puntero, cuando esta la cabecera de la tabla.

Go

Permite ir directamente a un registro, también podemos indicarte por medio de un numero de registro determinado. Con **GO TOP** nos ubicamos en el primero y con **GO BOTTOM** nos ubicamos en el último. Recordemos que el desplazamiento esta supeditado al campo en que este indexada la tabla.

Por medio de la cláusula **IN** podemos mover el puntero en otras áreas de trabajo.

Seek

Busca directamente un valor dentro del archivo de índices, ubicándose el puntero en ese registro en el caso de encontrarlo, o bien accediendo al final de la tabla en caso de no encontrarlo. En caso de tener activo en VFP en Herramientas, Opciones, Data, *el check SET NEAR on* dentro del cuadro de *String Comparison*, podemos acceder al valor mas próximo de nuestra búsqueda. Activar esta opción de de gran utilidad al momento de realizar búsquedas.

Found()

Con este comando podremos saber si un valor cualquiera sea ha encontrado o no. Se lo debe ubicar después de la instrucción de búsqueda.

Seek()

Sustituye a las dos instrucciones anteriores y al **SET ORDER**, lo que la hace mucho mas practica. Un ejemplo a implementar seria el siguiente:

```
CLOSE DATABASES
OPEN DATABASE (HOME(2) + 'Data\testdata')
USE clientes ORDER cliente
? SEEK('CORONEL') && Retorna .T., si encontró el registro
```

Do While

Este clasico bucle es muy utilizado en cualquier lenguaje de programación, en lo posible debemos olvidarnos de este bucle y sustituirlo por el Comando **Scan**, que es mucho mas rápido y practico. Un ejemplo seria:

```
DO WHILE !EOF()
    Skip 1
ENDDO
```

Esto hara que avance por la tabla de un registro por ves hasta que encuentre el final del registro.

Scan

Es mucho mas practico que el anterior, porque además de recorrer toda la tabla en forma automática, permite filtrar las búsquedas mediante la cláusula **FOR** y **WHILE**. Para ello debemos indicarle por que valor empezara la búsqueda. Es muy importante volver al área de la tabla sobre la cual se esta trabajando con el **SCAN** si nos hemos situado en otra tabla dentro del bucle para realizar alguna operación. Un simple ejemplo seria.


```
CLOSE DATABASES
OPEN DATABASE (HOME(2) + 'Data\testdata')
USE profesores
CLEAR
SCAN FOR UPPER(profesor) = 'CORONEL'
? idprofesor, profesor, edad
ENDSCAN
```

COMANDOS DE CALCULOS EN TABLAS.

El uso de estos comandos en forma correcta, puede simplificarlos a la hora de escribir código.

Average

Realiza la media de un campo o campos determinados por la clausula **FOR** o **WHILE**. Podemos guardar el resultado de una variable o varias variables dependiendo de los campos promediados.

Sum

Similar al comando anterior, pero este hace una suma

Count

Cuenta los registros dependiendo del índice y del estado de eliminación.

Reccount ()

Cuenta todos los registros estén o no eliminados. No se pueden incluir las cláusulas **FOR** y **WHILE**.

Calculate

Podemos realizar varios cálculos, utilizando una sola instrucción, los cálculos principales entre otros, que podemos hacer son:

Argumento	Se usa para
AVG()	Realiza la media. Muy similar a AVERAGE
CNT()	Cuenta el numero de registros. Igual a COUNT
MAX()	Encuentra el valor maximo de todos los campos
MIN()	Halla el valor minimo
NPV ()	Calcula el valor neto de una inversion
STD ()	Podemos realizar la desviación típica.
SUM ()	Suma los valores de los campos.
VAR ()	Halla la varianza de un promedio

En algunos ejemplos que daremos, vamos a mostrar algunos de estos comandos ya que detallarnos en este omento no es necesario.

SISTEMA DE ALMACENAMIENTO EN BUFFER

Si hemos llegado detenidamente hasta aquí, es porque la programación o el desarrollo de sistemas es lo nuestro. Y como no serlo si programando podemos construir y darle vida a nuestra pc!. Entonces felicitaciones, a partir de ahora nos introduciremos en la programación en forma mas detallada, en el caso que el ejemplo no sea lo suficientemente claro de entender, con el desarrollo de las aplicaciones ejemplos daremos por finalizado este tema.

¿Que es el Sistema de almacenamiento en buffer?

Consiste en la creación automática de un espacio de memoria destinado a guardar el o los registros que se estén editando sin necesidad de tipiar largos códigos. VFP se encarga por nosotros de esto en forma inteligente, a nosotros nos queda chequear si se ha producido alguna modificación por parte del usuario, y por ultimo en caso de ser necesario guardar o descartar los cambios. Esto evita que se graben registros en blanco en las tablas en forma innecesaria.

Esto es una gran utilidad que trae aparejado VFP entre tantas herramientas de avanzada que iremos descubriendo a lo largo de este libro, La programación orientada a objetos y entornos visuales se han impuesto! y ni hablar de cuando tratemos la Herencia y Polimorfismo que VFP es un pionero desde hace ya aproximadamente 15 años!!! Visual FoxPro siempre ha estado un paso adelante en todo. Esto lo convierte en el favorito a la hora de diseñar aplicaciones comerciales o financieras por ejemplo.

El almacenamiento para el trabajo en red

Además de facilitar el mantenimiento de las tablas, también podemos controlar el tipo de acceso de los usuarios a las tablas desde los distintos puestos de red. Tenemos la técnica de bloqueo pesimista, cuando se produce una modificación, o por el otro lado, dejar absoluta libertad a los usuarios de la red bloqueando solo a la hora de grabar, que es lo que se llama bloque optimista.

Tanto para un caso, como para el otro, tenemos defensores y detractores. Por un lado están los que afirman que no se debe permitir tocar a otro usuario de la red si uno está editando el mismo registro, ya que si esto ocurriera se grabaría la actualización o modificación que realizara el último, con lo cual el programa actualizaría una modificación hecha por uno de los usuarios y no la del otro; por otra parte están los que aseguran si se bloquea el registro y el usuario en cuestión no cierra la ventana o aplicación donde está modificando, puede dejar bloqueada la red, las dos teorías son aceptadas y VFP las acepta, ya que permite ambos tipos de bloqueos.

Detallaremos los 3 tipos de Buffers existentes VFP, para el almacenamiento de datos:

- 1 buffer: Valor predeterminado, contiene los valores que el usuario visualiza en pantalla.
- 2 buffer : Almacena la información original que tenía la tabla.
- 3 buffer : Guarda el valor último recibido de la red.

Como activar el almacenamiento en buffer

Esto se logra a través de la función **CURSORSETPROP ()**. Por cada tabla que abramos tendremos que escribir un **CURSORSETPROP**. La sintaxis será diferente de acuerdo al tipo de almacenamiento que hayamos elegido, de acuerdo al formato:

=CURSORSETPROP (“Buffering”, < valor de almacenamiento >)

En este tipo de almacenamiento podemos usar 5 tipos (valores) de almacenamiento:

Valor	Significado del Valor Numérico
1	Es ignorado el almacenamiento en buffer, por lo tanto no lo utilizamos
2	Se bloquea a nivel registro y pesimista. El registro estará bloqueado hasta Que se grabe la modificación.
3	El bloqueo será a nivel registro, aunque esta vez será optimista. Entonces se Produce el bloqueo cuando se actualiza la modificación o se produce un desplazamiento a otro registro.
4	Bloqueo pesimista a nivel de tabla. Funciona en forma muy similar que a Nivel registro, lógicamente que a nivel tabla.
5	Bloqueo optimista a nivel tabla. Sucede cuando se actualizan las modificaciones

Para todos los casos debemos tener, el acceso no exclusivo (**SET EXCLUSIVE OFF**), el **SET MULTILOCKS ON**.

Después de incluir este comando ya podremos disfrutar de las cualidades del sistema.

Comprobar si ha habido modificaciones

A partir de ahora debemos realizar la comprobación si el usuario ha realizado alguna modificacion.

La verificación la podemos realizar de dos maneras con las siguientes funciones: **GETFLDSTATE** (), como es fácil de advertir, esta función nos informa el estado de los registros y con **GETNEXTMODIFIED**() nos informara el estado de la tabla. Esto quiere decir que podemos utilizar **GETFLDSTATE** () tanto para el bloqueo a nivel de registro

Como para el de tabla, mientras que con **GETNEXTMODIFIED**() esta recomendado a nivel tabla, aunque a continuación veremos cómo se puede utilizar también a nivel de registro.

La sintaxis de **GETFLDSTATE** () es:

=GETFLDSTATE (<nombre ó nº del campo>, <alias>)

Los valores posibles a devolver son:

Valor	Significado
1	No se ha editado el campo o no ha cambiado el estado de eliminación.
2	Se ha editado el campo o ha cambiado el estado de eliminación.
3	No se ha editado el campo de un registro añadido o ha cambiado el estado de Eliminación del registro añadido.
4	Se ha editado el campo de un registro añadido o ha cambiado el estado de eliminación para el registro añadido.
0	Nos permite saber si ha cambiado el estado de eliminación del registro.
-1	Nos devuelve todos los estados del registro, eliminación, edición, etc

Mostramos esta forma de programar con el sistema de buffers, por una cuestión de compatibilidad con las versiones 6,7,8 ya que en la versión 9 tenemos una forma mas simplificada de hacer todo esto. De todas maneras no esta de más saberlo.

IF “2” \$GETFLDSTATE (-1) OR “3” \$GETFLDSTATE(-1) ;

OR “4” \$GETFLDSTATE(-1)

WAIT WINDOW “Tenemos una modificación”

ENDFOR

A nivel registro podemos hacer la comprobación con **SETFLDSTATE()**. Para saber que registro se ha modificado o eliminado dentro de la tabla, esto significa que tenemos una técnica mas que podemos incluir en nuestros programas. A modo de ejemplo:

```
=SETFLDSTATE(cFieldName | nFieldNumber, nFieldState [, cTableAlias  
| nWorkArea])
```

Si no se ha modificado o borrado registro alguno, el valor devuelto será 1. Debemos tener en cuenta que **SETFLDSTATE** devuelve el valor del primer registro que encuentra con *flag* de modificación, por lo cual es importante el número de registro que se le pase como parámetro.

Tanto una técnica como la otra son validas, Cualquiera de las dos técnicas son validas.

Como actualizar la tabla.

Hemos llegado al último paso que es *Guardar o Descartar las modificaciones*.

Esto se lleva a cabo, con las funciones: **=TABLEUPDATE()** y **=TABLEREVERT ()**. Debemos recordar que estamos trabajando sobre una tabla intermedia, que esta en memoria, y que todos estos datos debe ser volcada a la tabla física. Este paso difiere del tipo de almacenamiento que hayamos elegido.

TABLEUPDATE () dispone de cuatro parámetros:

```
=TABLEUPDATE( [nRows [, lForce]] [, cTableAlias | nWorkArea] [, cErrorArray] )
```

Al actualizar la tabla física, lo podemos hacer para el registro en que se esta trabajando o para todos los registros de la tabla. El primer parámetro está destinado a este fin, los valores posibles a tomar son:

Valor	Significado del Valor
0	Por defecto. Se actualizara el registro en el que está el puntero sea cual sea el tipo de almacenamiento elegido.
1	Se actualizarán todos los registros si el almacenamiento es a nivel de tabla y sólo El registro actual si el almacenamiento se estableció a nivel de registro.
2	Similar funcionamiento que el valor 1 pero en este caso VFP no emitirá un si la actualización no terminó con éxito.

Recordemos lo dicho que para el almacenamiento en buffer optimista, habrá que establecer **.T.** el segundo de los parámetros, con lo que se producirá la actualización desde el buffer de red.

Si por el contrario, queremos descartar los cambios hechos, deberemos utilizar la función **TABLEREVERT** de la siguiente manera:

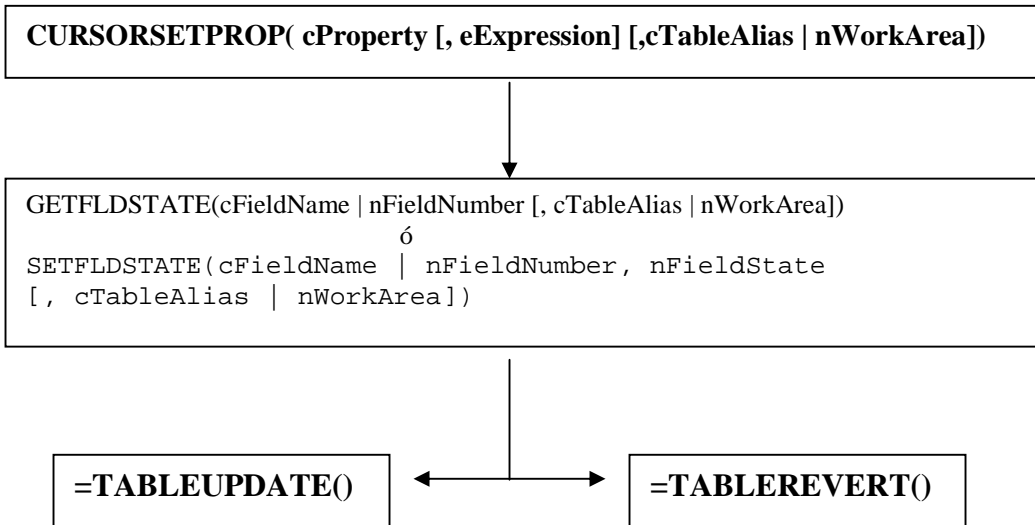
=TABLEREVERT([AllRows [, cTableAlias | nWorkArea]])

Deberemos establecer **.T.** para que se realice el descarte de los cambios realizados en todos los registros. Con esta función recuperamos los datos del 2º buffer, es decir los datos originales de la tabla.

En nuestros programas o aplicaciones, usaremos los 2 clásicos botones de **Guardar** y **Cancelar** y simplemente incluiremos dentro de los botones **=TABLAUPDATE** o **=TABLEREVERT** según sea necesario. Debemos tener cuidado de completar todos los comandos para que todo quede bien programado, si no ejecutamos alguna de estas dos instrucciones, VFP mostrara en pantalla un mensaje de error cuando queramos cerrar la tabla en cuestión. Esto sucederá cuando hemos realizado el bloqueo a nivel tabla, esto quiere decir que hasta que no grabemos o descartemos los cambios no se podrá cerrar.

Todo esto será bien ampliado al momento de crear nuestras aplicaciones de ejemplo

PROCESO DE ACTUALIZACION



Implementar todo esto es más fácil de lo que parece.

CONSULTAS Y VISTAS.

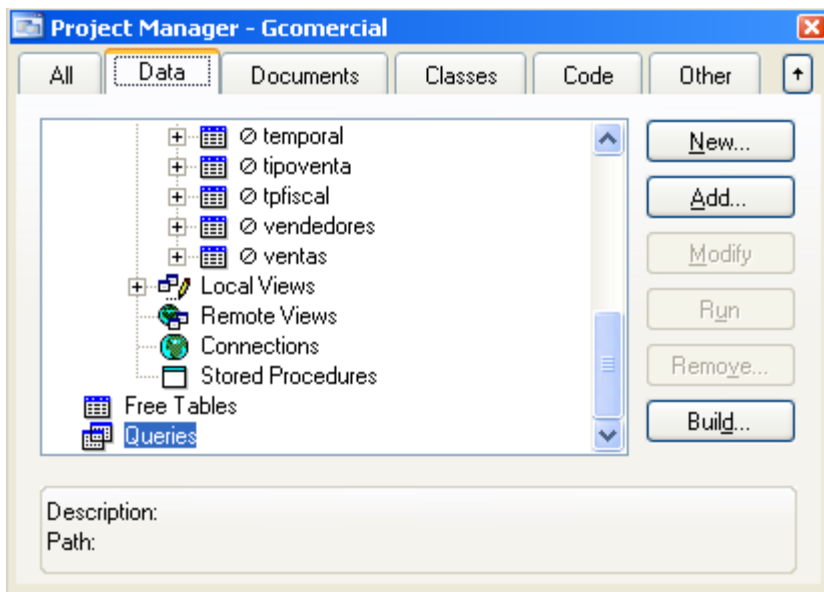
Con VFP 9, hacer consultas y vistas es muy fácil, tan fácil que no debemos en principio incluir comandos tipo **SQL**, ya que al construir consultas y vistas en forma interactiva, lo que estamos haciendo es establecer comandos 100 % de **SQL**. Debemos dejar bien en claro, que la principal diferencia entre vistas y consultas, es que las primeras permiten la actualización sobre las tablas de origen, mientras que las consultas no.

No vamos a detallar excesivamente en consultas y vistas ya que son muy intuitivas, pero si debemos tener en claro, que el programador debe tener amplio dominio en el lenguaje de consultas **SQL**.

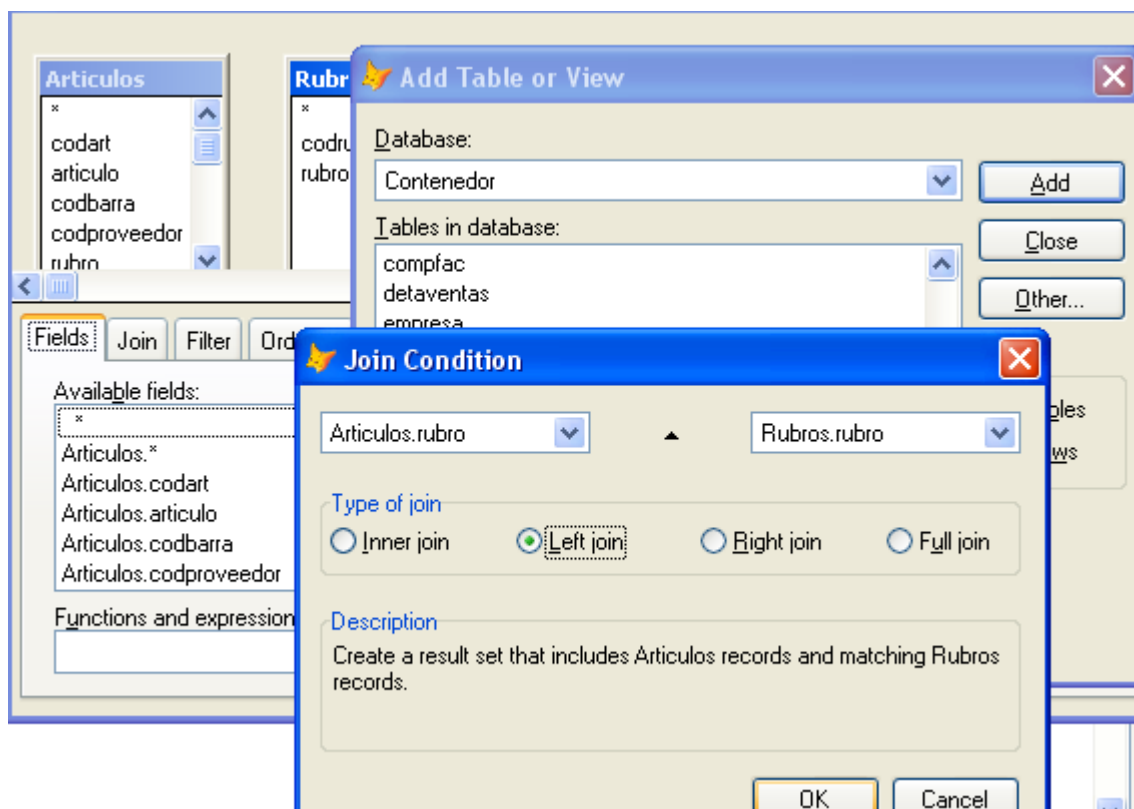
Hacemos notas que en el caso de las consultas, se crea un archivo por separado, con extensión .QPR, que puede ser ejecutado en cualquier momento. En cambio las vistas que pueden ser Locales o Remotas se almacenan dentro de la Base de Datos y son separables de ellas, lo que las convierte en menos flexibles.

CONSULTAS

Teniendo abierto nuestro proyecto, en el cual debemos tener por lo menos 2 tablas, hacemos clic en la solapa **Data** y luego clic en **New**, similar a la figura de abajo y elegimos 2 tablas en las cuales una sera por ejemplo la tabla madre y la otra actuara como tabla hija, esto hacemos a los fines de tener un ejemplo mas elaborado. Para nuestro caso usamos una tabla llamada Articulos y la otra Rubros



En la siguiente figura se nos solicitara que tipo de combinación queremos para nuestra consulta, elegimos **left join**, esto quiere decir que creara un conjunto de resultados que incluye los registros de Artículos y los que coincidan con Rubros. La figura de abajo ilustra este ejemplo. Claro esta que podemos usar tantas tablas como sean necesarias para construir nuestra consulta. Luego de escoger nuestras tablas debemos cerrar la ventana de Agregar Tabla o Vista.



En detalle la forma de realizar esta “union” entre estas dos tablas seria las siguientes:

1) Interna: se tomaran solo aquellos registros de las dos tablas que cumplan la condición específica.

2) Izquierda: Se tomaran todos los registros de la tabla de la izquierda y aquellos de la tabla de la derecha que cumplan la condición.

3) Derecha: de igual manera que la anterior, pero con la diferencia que esta vez se tomaran todos los registros de la tabla de la derecha y solo aquellos de la tabla de la izquierda que cumplan la condición.

4) Completa: se usaran todos los registros de ambas tablas, cumplan o no la condición establecida.

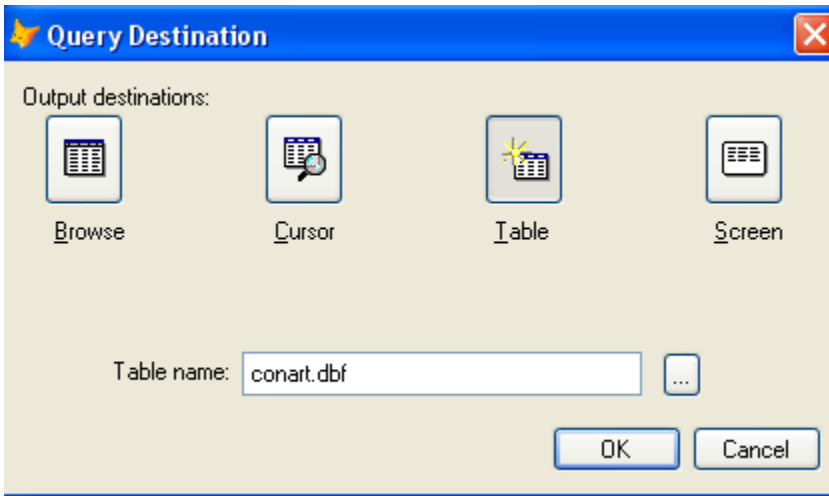
Luego de esto debemos ir seleccionando en las distintas solapas las opciones que nos ofrece el diseñador de consultas que son las siguientes, hacemos notar que siempre que se trabajemos tanto con vistas como con consultas se deben agregar la menor cantidad de campos, uniones, filtros, y agrupamientos, esto a fin de entender mejor su funcionamiento.

Las solapas que nos ofrece el diseñador de consulta a fin de “extraer información”, todo esto son comandos **SQL** ocultos en el asistente para que resulte más intuitivo para el programador, estas solapas son:

- 1) **Fields:** es donde debemos indicar cuales son los campos de la tabla o tablas que queremos que aparezcan en la consulta.
- 2) **Join:** aquí aparecen las “uniones” que hemos hecho al principio al escoger las tablas y los campos a relaciones.
- 3) **Filter:** desde aquí establecemos relaciones que deben cumplir los registros resultantes de la consulta. No debemos confundirlo con **Join**.
- 4) **Oder By:** fija un orden determinado para los datos que se mostraran en la consulta. Aquí se refiere al orden físico de los registros de la tabla serán usados en la consulta.
- 5) **Group By:** Realiza un agrupamientos de la consulta. Muy útil al momento de tener un informe detallado por grupo.
- 6) **Micellaneous:** entre varias opciones que ofrece, tenemos la de forzar a la unión y no permitir valores duplicados.

por último debemos establecer haciendo clic en la consulta, debemos establecer el destino de nuestra consulta “Output Settings” en el cual debemos establecer por ejemplo si queremos

Como destino de la consulta el de Tabla, de esta manera por ejemplo, tendremos una unión de dos tablas, ya que a través del asistente la hemos unidos a dos tablas en forma oculta con comandos **SQL**. En la figura de abajo tenemos las opciones para escoger el destino de nuestra consulta.



No entraremos en muchos detalles ya que en VFP las consultas son incluidas por cuestión de compatibilidad, en cambio en las vistas, daremos mas detalle ya que son mas flexibles a la hora de trabajar con ellas y en las aplicaciones practicas daremos ejemplos prácticos con vistas del tipo locales.

VISTAS

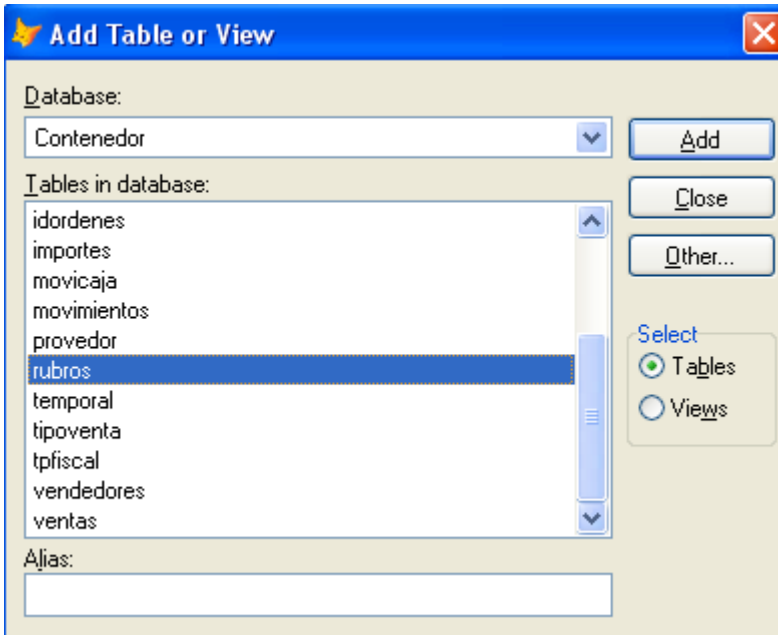
Lo primero que tenemos que saber que contamos con la posibilidad de trabajar con **Vistas Locales** y **Vistas Remotas**, en las **Vistas Locales**, nosotros trabajaremos con tablas propias de VFP; en cambio con las **Vistas Remotas** podemos trabajar con tablas ajenas a VFP, lo cual es muy util si deseamos construir **Cliente-Servidor**. Las vistas remotas son útiles en el caso de que un sistema administrativo en una empresa o un banco, tenga un sistema próximo a ser

dado de baja y el Gerente no quiere perder ciertos datos que son de vital importancia para la empresa. De todas maneras contamos con muchas herramientas de conversión de datos, siendo Access una de ellas por ejemplo, en las cuales podemos **exportar las tablas o bases de datos a un Formato tipo DBF**. Esto hará en algunas circunstancias los programadores de VFP, queden como héroes al permitir que todos los datos sean traspasados a un nuevo sistema sin necesidad de cargarlos todo de nuevo.

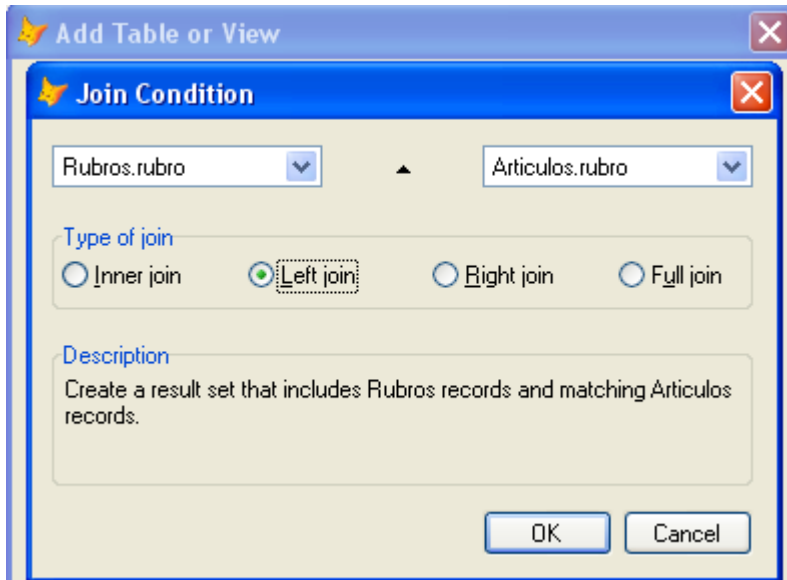
El generador de **Vistas** es prácticamente igual al visto al de **Generador de Consultas**, salvo en la última de las páginas que permite actualizar el origen de datos.

Iniciemos una Vista Local, a modo de ejemplo:

Iniciemos un proyecto y no posicionamos en **Databases** luego hacemos clic en **Local Views** y hacemos otro clic en **NEW** y por ultimo en **new view** y tendremos delante nuestro la ventana de **Agregar Tabla o Vista**. Aquí debemos elegir la tabla Rubros y Artículos, tal cual como lo muestra la figura de abajo.

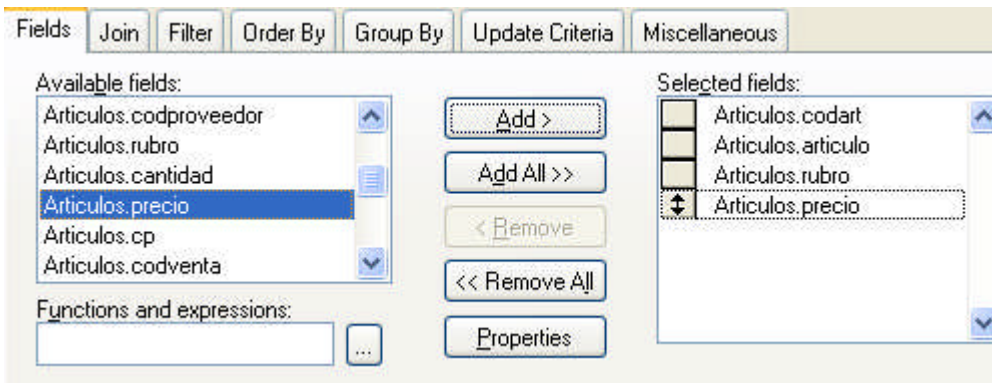


Luego hacemos clic en **Close**, y veremos el cuadro que nos preguntara que tipo de relación queremos establecer, elegimos **Left join**, y hacemos clic en Ok, luego en caso de que aparezca de nuevo la ventana para **agregar Tablas o Vistas** hacemos clic en Cerrar.

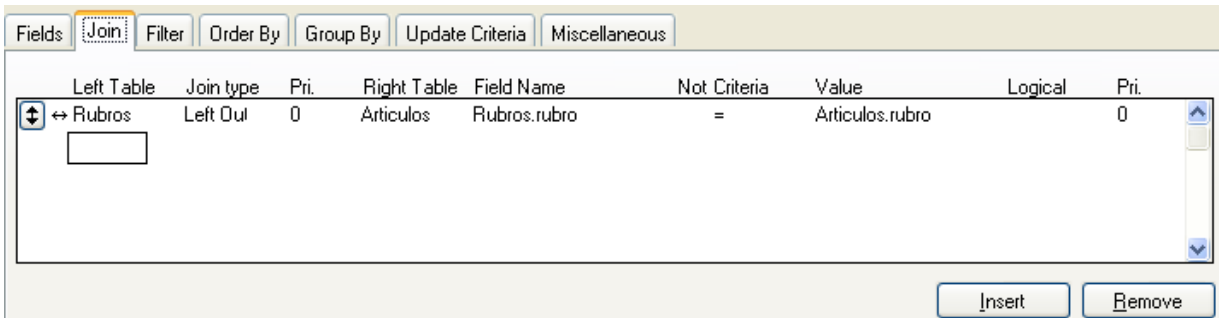


Ahora tendremos que trabajar en todas y cada una de las solapas que disponemos en el diseñador de vistas. Las solapas que disponemos son las siguientes:

Fields: aquí debemos elegir algunos campos de la **tabla artículos** que necesitemos visualizar, entre ellos debemos elegir los campos mas elementales a modo de ejercicio, ellos son: Articulos.codart, Articulos.articulo, Articulos.rubro, Articulo.precio. esto selección la debemos hacer con el boton **Add >** y la solapa **Fields** quedara de la siguiente manera como lo muestra la figura de la siguiente pagina.



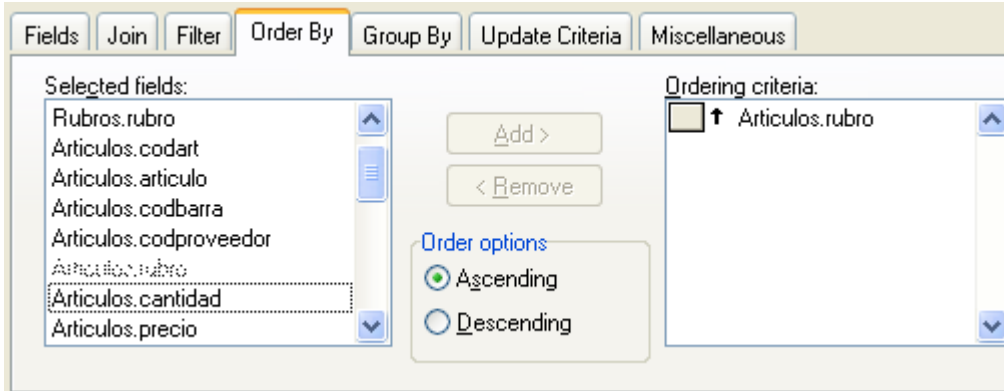
Ahora hacemos clic en la solapa **Join** y veremos como quedaron unidas las dos tablas por medio de la relación. En caso de haber hecho todo bien la solapa **Join** quedara de la siguiente manera tal cual como lo grafica la imagen de abajo.



Aquí vemos que la tabla Rubros esta unida de izquierda a derecha con la tabla Rubros que esta a la derecha, es muy facil advertir que siempre que queramos relacionar tablas tanto dentro de una Base de Datos o dentro de una Vista, ambas tablas tienen un campo en común, este campo en común o igual hace que dos o tablas por ejemplo estén relacionadas, esto es común a cualquier lenguaje de programación, sobre todo en el ámbito relacional de bases de datos.

En la solapa **Filter**: no hacemos nada.

En Cambio en la Solapa **Order By:** seleccionamos el campo **Articulos.rubro** y hacemos clic en boton **Add >** y la solapa quedara de la siguiente manera como lo muestra la figura de abajo.



A partir de ahora términos con la construcción de nuestra consulta !. Ahora lo que debemos es hacer con el lado derecho del Mouse y hacemos clic en **Run Query** y podremos ver como queda terminada nuestra vista local, lo cual la podremos utilizar en el diseño de nuestros informes. Y por su puesto, después de tanto trabajo, no debemos olvidarnos de salvar o grabar nuestra primera vista local, la grabaremos con el nombre de **viewart**.

En caso de haber cometido errores la vista será visualizada en pantalla de la siguiente manera como lo muestra la figura de la pagina siguiente.

Éxitos en el estudio de este libro estimado lector y lectora, ya que nos falta un largo camino por recorrer.

Update Criteria: Aquí es donde trabajaremos directamente con opciones propias de **SQL**, no es necesario por el momento detallarlas ya que para el ejemplo que hemos mostrado no es necesario activar algunas de estas opciones, pero de acuerdo al tipo proyecto que estemos trabajando mas de una vez será necesario activar algunas de estas opciones.

Micellaneous: aquí disponemos de las opciones, entre otras, la de forzar las uniones y de la posibilidad de evitar que se emitan valores duplicados.

Viewart			
Codart	Articulo	Rubro	Precio
60	GATO MIX X 1 KG	ALIMENTOS PARA GATOS	0.00
69	GATY X 1 KG	ALIMENTOS PARA GATOS	0.00
70	GATY X 15 KG	ALIMENTOS PARA GATOS	0.00
52	SABROFOOD X 22 KG	ALIMENTOS PARA PERROS	0.00
53	SABROFOOD X 10 KG	ALIMENTOS PARA PERROS	0.00
54	SABROFOOD X 1 KG	ALIMENTOS PARA PERROS	0.00
55	SABROSITOS MIX X 15 KG	ALIMENTOS PARA PERROS	0.00
56	SABROSITOS MIX X 1 KG	ALIMENTOS PARA PERROS	0.00
57	SABROSITOS CACHORROS X 18 KG	ALIMENTOS PARA PERROS	0.00
58	SABROSITOS CACHORROS X 1 KG	ALIMENTOS PARA PERROS	0.00
62	DOGUI VARIEDAD X 21 KG	ALIMENTOS PARA PERROS	0.00
63	DOGUI CACHORRO X 8 KG	ALIMENTOS PARA PERROS	0.00
64	DOGUI CACHORRO X 21 KG	ALIMENTOS PARA PERROS	0.00
65	DOG CHOW ADULTO X 1 KG	ALIMENTOS PARA PERROS	0.00
66	DOG CHOW CACHORRO X 1 KG	ALIMENTOS PARA PERROS	0.00
67	DOG CHOW CACHORRO X 21 KG	ALIMENTOS PARA PERROS	0.00
68	DOG CHOW ADULTO X 21 KG	ALIMENTOS PARA PERROS	0.00
81	CASCARILLA P.PERROS X 25 KG	ALIMENTOS PARA PERROS	0.00
86	VORAZ X 10 KG	ALIMENTOS PARA PERROS	0.00
87	VORAZ X 1 KG	ALIMENTOS PARA PERROS	0.00
89	ULISES ADULTO X 25 KG	ALIMENTOS PARA PERROS	0.00
90	ULISES ADULTO X KG	ALIMENTOS PARA PERROS	0.00
91	ULISES ADULTO X 15 KG	ALIMENTOS PARA PERROS	0.00
92	ULISES CACHORROS X 15 KG	ALIMENTOS PARA PERROS	0.00
94	SABROFOOD X 1/2 KG	ALIMENTOS PARA PERROS	0.00
104	DOGUI SUETO X 25 KGS	ALIMENTOS PARA PERROS	0.00

Vista final de nuestra primera “vista creada a partir de dos tablas”. A esta vista que la hemos llamado **visart** la podremos usar con dentro de un reporte por ejemplo para emitir un listado que luego podrá ser impreso.

Ya estamos en condiciones de decir que estamos **programando o diseñando aplicaciones!**

Funciones de Microsoft Visual FoxPro.

A partir de aquí ingresaremos a ver o en el caso de los programadores de versiones mas antiguas de VFP, o de otros lenguajes, un repaso de los comandos mas usados en el desarrollo de nuestros programas y aplicaciones, al final del capitulo veremos un pequeño repaso de los comandos mas importantes de **SQL**. Estos ejemplos pueden ser escritos dentro un programa tipo .PRG o bien dentro de nuestros aplicaciones, a ser mas exactos dentro de nuestros objetos de la clase base por ejemplo. a los antiguos programadores de D.O.S que es probable que todavía existan algunos, esto será muy facial de aprender. A los fines didácticos, y para ahorrar espacios y hojas usaremos un tamaño mas pequeño al escribir este capitulo. Es increíble las horas que se pierden en cualquier lenguaje de programación al no conocer como trabajar con funciones, por eso consideramos muy importe el correcto uso de ellas, ya que la tarea de que desarrollan muchos programadores o desarrolladores lo hacen en forma solitaria.

FUNCIONES DE CADENAS.

ASC() (Función)

Retorna el código ANSI del carácter situado más a la izquierda de una expresión de caracteres. Tipo de valor devuelto: Carácter.

ASC(cExpression)

```
STORE 'AaBbCcDdEeFf' TO gcANSI  && 10 characters
CLEAR
FOR nCOUNTER = 1 TO 10
    ? SUBSTR(gcANSI, nCOUNTER,1) && Display a character
    ?? ASC(SUBSTR(gcANSI, nCOUNTER)) && Display ANSI value
ENDFOR
```

Ltrim (), Rtrim () y Alltrim (). Funciones

LTRIM(Expression [, nFlags] [, cParseChar [, cParseChar2 [, ...]]])

RTRIM(cExpression [, nFlags] [, cParseChar [, cParseChar2 [, ...]]])

ALLTRIM(Expression [, nFlags] [, cParseChar [, cParseChar2 [, ...]]])

Para las tres funciones, la utilidad que tenemos es que podemos quitar los que pudieran tener una cadena caracteres, ya sea esta desde la izquierda, por la derecha o por ambos lados.

Valor devuelto: Carácter.

Left () y Right ().Funciones

Devuelve un numérico especificado de caracteres de una expresión de caracteres, empezando por el carácter situado a la izquierda o bien a la derecha.

Un ejemplo de todo lo anteriormente detallado podría ser:

```
a = "      bad boy"
b = "Fabian is ... a man ...."
c = "  Hi Master  "
? a
? LTRIM(Right(a,5))
? LTRIM(LEFT(a,8))
? b
? RTRIM(RIGHT(b,2))
? c
? ALLTRIM(c)
```

AT(), Rat () y Atc (). Funciones

Devuelve la posición numérica inicial de la primera aparición de una expresión de caracteres o de un campo memo dentro de otra expresión de caracteres o de campo memo, contando desde el carácter situado más a la izquierda. Con **Rat ()** por la derecha, pero el valor devuelto siempre sera la posición que empieza por la izquierda.

Si queremos ignorar las letras tanto en mayúsculas como en minúsculas podemos utilizar **ATC()**. no tiene mucha utilidad a la hora de desarrollar aplicaciones comerciales.

CHR() (Función)

Devuelve el carácter asociado al código ANSI numérico especificado. Valor devuelto tip o Carácter.

nCódigo ANSI Especifica un número entre 0 y 255 cuyo carácter ANSI equivalente devuelve CHR(). Utilice ASC() para devolver el valor ANSI de un carácter especificado.

```
CLEAR
FOR nCode = 0 TO 255
    ? nCode && Display numeric value
    ?? ' ' + CHR(nCode) && Display character
ENDFOR
```

CHRTRAN() (Función)

Reemplaza cada carácter de una expresión de caracteres que coincida con un carácter de una segunda expresión de caracteres con el carácter correspondiente de una tercera expresión de caracteres.

CHRTRAN(cSearchedExpression, cSearchExpression, cReplacementExpression)

CTOBIN() (Función)

Convierte una representación de caracteres binarios en un valor entero.

CTOBIN(cExpression [, cFlags])

Argumentos: cExpresión Especifica la representación de caracteres binarios que se va a convertir.
Valor devuelto: Numérico.

CURVAL() (Función)

Devuelve valores de campo directamente desde disco para una tabla o un origen de datos remoto.

CURVAL(cExpression [, cTableAlias | nWorkArea])

MAX() (Función)

Evalúa un conjunto de expresiones y devuelve la expresión con el valor máximo.

MAX(eExpression1, eExpression2 [, eExpression3 ...])

Tipos devueltos: Carácter, Numérico, Currency, Double, Float, Date o DateTime

MIN() (Función)

Evalúa un conjunto de expresiones y devuelve la expresión que tenga el valor mínimo.

MIN(eExpression1, eExpression2 [, eExpression3 ...])

Tipos devueltos: Carácter, Numérico, Currency, Double, Float, Date o DateTim e.

Tanto para Min() y Max() un ejemplo a considerar seria el siguiente:

```
CLOSE DATABASES
CREATE TABLE azar (cValor N(3))
FOR nItem = 1 TO 10  && Append 10 records,
    APPEND BLANK
    REPLACE cValor WITH 1 + 1000 * RAND( )  && Insert random values
ENDFOR
CLEAR
LIST  && Display the values
gnMaximum = 1  && Initialize minimum value
gnMinimum = 1000  && Initialize maximum value

SCAN
    gnMinimum = MIN(gnMinimum, cValor)
    gnMaximum = MAX(gnMaximum, cValor)
ENDSCAN

? 'The minimum value is: ', gnMinimum  && Display minimum value
? 'The maximum value is: ', gnMaximum  && Display maximum value
```

OCCURS() (Función)

Devuelve el número de veces que ocurre una expresión dentro de otra expresión de caracteres. Tipo de valor devuelto: Numérico.

OCCURS(cSearchExpression, cExpressionSearched)

Función muy útil al momento de hacer un programa del tipo multimedia, ya que con esta función podemos interactuar con el usuario, sobre todo niños, para saber por ejemplo si a razonado correctamente.

```
STORE 'tallarines' TO charstring
CLEAR
? OCCURS('a', charstring)  && Displays 2
? OCCURS('e', charstring)  && Displays 2
? OCCURS('i', charstring)  && Displays 1
? OCCURS('l', charstring)  && Displays 2
```

GETTCP() (Función)

Solicita una página de códigos mostrando el cuadro de diálogo Página de códigos y a continuación devuelve el número de la página de códigos elegida. Valor devuelto: Numérico.

GETTCP([nCodePage] [, cText] [, cDialogTitle])

```
x = GETTCP( )
? x
```

REPLICATE() (Función)

Devuelve una cadena de caracteres que contiene una expresión de caracteres especificada que se repite un determinado número de veces. Valor devuelto: Carácter.

REPLICATE(cExpression, nTimes)

? REPLICATE(' PRIBIET ',4) && Displays PRIBIET PRIBIET PRIBIET PRIBIET

Str(Función)

Devuelve el carácter equivalente a una expresión numérica. También convierte un valor numero a cadena de carácter

STR(nExpression [, nLength [, nDecimalPlaces]])

```
? STR( 22779819.627788,12,3 )

22779819.628
```

El valor devuelto será, un total de doce posiciones incluyendo al punto, y tres decimales.

SUBSTR() (Función)

Devuelve un número de caracteres específico de una expresión de caracteres o un campo memo. Valor devuelto: Carácter.

SUBSTR(cExpression, nStartPosition [, nCharactersReturned])

```
STORE 'HelloMyDearWorld' TO charString
CLEAR
? SUBSTR(charString, 1, 5)
? SUBSTR(charString, 6)
```

\$ (Operador)

Retorna verdadero (T) si una expresión de caracteres está contenida dentro de otra expresión de caracteres; si no lo está, devuelve falso (F). Tipo devuelto: Lógico.

cSearchFor \$ cSearchIn

Si TALK está ON y nPosiciónInicial es mayor que el número de caracteres de cExpresión, Visual FoxPro genera un mensaje de error. Si TALK está OFF, se devolverá una cadena vacía.

Los campos memo pueden manipularse de la misma forma que las expresiones de caracteres, los campos de tablas, las variables de memoria o los elementos de matriz. Por ejemplo, si MEMO_FLD es un campo memo, lo siguiente será aceptable:

```
LIST FOR 'FOXPRO' $ UPPER(memo_fld)
```

FUNCIONES MATEMATICAS.

ABS() (Función)

Retorna el valor absoluto de la expresión numérica especificada. Valor devuelto: Numérico.

ABS(*nExpression*)

nExpression especifica la expresión numérica cuyo valor absoluto devuelve ABS().

CLEAR

nFortyFive=45

nTeen=10

nThirty=30

nForty=40

? **ABS**(-nFortyFive) && Displays 45

? **ABS**(nTeen-nThirty) && Displays 20

? **ABS**(nFortyFive-nTeen)

? **ABS**(nThirty-nTeen) && Displays 20

STORE 100 **TO** gnNumber1

STORE 12 **TO** gnNumber2

? **REPLICATE**("=", 20)

? **ABS**(gnNumber2-gnNumber1) && Displays 88

ACOS() (Función)

Devuelve el arco coseno de una expresión numérica especificada. Valor devuelto: Numérico.

ACOS(*nExpression*)

nExpression Especifica la expresión numérica cuyo arco coseno devuelve ACOS(). El valor de *nExpression* puede estar comprendido entre -1 y +1. El valor que devuelve ACOS() está comprendido entre 0 y pi (3,141592). El número de decimales que devuelve ACOS() está determinado por SET DECIMALS.

Utilice RTOD() para convertir radianes a grados.

CEILING() (Función)

Devuelve el entero más próximo que sea mayor o igual que la expresión numérica especificada. Valor devuelto: Numérico.

CEILING(nExpression)

nExpresión Especifica el número cuyo próximo entero mayor devuelve CEILING().

```
STORE 20.2 TO num1
STORE -1.2 TO num2

? CEILING(num1)    && Displays 11
? CEILING(num2)    && Displays -10
? CEILING(10.0)    && Displays 10
? CEILING(-10.0)   && Displays -10
```

INT() (Función)

Evalúa una expresión numérica y devuelve la parte entera de dicha expresión. Valor devuelto: Numérico.

INT(nExpression)

```
CLEAR
? INT(12.5)        && Displays 12
? INT(6.25 * 2)    && Displays 12
? INT(-12.5)       && Displays -12
STORE -12.79 TO nNumber
? REPLICATE ("*", 20)
? INT(nNumber)     && Displays -12
```

ROUND() (Función)

Devuelve una expresión numérica redondeada a un número especificado de cifras decimales. Valor devuelto: Numérico.

ROUND(nExpresión, nLugaresDecimales)

```
? ROUND (12313.42659,2) && display 12313.43
```

RECCOUNT() (Función)

Devuelve el número total de registros de la tabla actual o especificada. Valor devuelto: Numérico.

RECCOUNT([nWorkArea | cTableAlias])

```
USE c:\ivaempre\bases\clientes.dbf SHARED
X = RECCOUNT ( )
? REPLICATE ( '-',54)
? 'TOTAL DE REGISTROS', X

? REPLICATE ( '-',54)
```

RECNO() (Función)

Devuelve el número del registro actual de la tabla actual o la especificada. Valor devuelto: Numérico.

RECNO([nWorkArea | cTableAlias])

```
USE c:\ivaempre\bases\clientes.dbf SHARED
GO 3
X = RECNO ( )
DISPLAY X
? REPLICATE ( '-',74)
DISPLAY FIELDS CLIENTE,DOMICILIO,LOCALIDAD
? 'REGISTRO ACTUAL ', X

? REPLICATE ( '-',74)
```

SUM (Comando)

Totaliza los valores de todos los campos numéricos especificados de la tabla actual seleccionada.

```
SUM [eExpressionList] [Scope] [FOR lExpression1] [WHILE lExpression2]
[TO MemVarNameList | TO ARRAY ArrayName] [NOOPTIMIZE]
```

Detallaremos todos los argumentos que tenemos disponibles con la **Función Sum**, ya que es muy útil cuando trabajamos con Bases de datos y necesitamos elaborar resultados finales e informes.

Argumentos:

eExpressionList especifica uno o varios campos o expresiones de campo que se van a totalizar. Si se omite la lista de expresiones, se totalizarán todos los campos numéricos.

Scope especifica un intervalo de registros que se van a utilizar en el total. Las cláusulas de alcance son: ALL, NEXT nRegistros, RECORD nNúmeroRegistro y REST.

El alcance predeterminado de SUM es ALL (todos los registros).

FOR lExpression1 Especifica que solamente se incluyan en el total los registros para los cuales la condición lógica ***lExpression1*** se evalúa como verdadera (.T.). La inclusión de FOR le permite totalizar registros condicionalmente, desechando los registros no deseados.

Si ***lExpression1*** es una expresión optimizable, Rushmore optimizará un comando SUM ... FOR. Para obtener un mejor rendimiento, utilice una expresión optimizable en la cláusula FOR. Para conseguir el rendimiento óptimo, utilice una expresión optimizable en la cláusula FOR.

WHILE lExpression2 especifica una condición por la cual los registros de la tabla activa se incluyen en el total siempre que la expresión lógica ***lExpression2*** se evalúe como verdadera (.T.).

TO MemVarNameList

Almacena cada total en una variable de memoria. Si especifica en ***TO MemVarNameList*** el nombre de una variable de memoria que no existe, Visual FoxPro la creará automáticamente. Separe los nombres de variables de memoria de la lista con comas.

TO ARRAY NombreMatriz Almacena los totales en una matriz de variables de memoria. Si la matriz que especifica en SUM no existe, Visual FoxPro la creará automáticamente. Si la matriz existe y es demasiado pequeña para albergar todos los datos, se aumentará automáticamente el tamaño de la matriz para acoger los totales.

NOOPTIMIZE Desactiva la optimización Rushmore de SUM.

Para obtener más información, consulte SET OPTIMIZE y “Uso de Rushmore para acelerar el acceso a datos” en, “Rushmore Optimization”, en la ayuda de VFP 9.

```
CLEAR
SET DECIMALS TO 2
USE c:\ivaempre\bases\compras.dbf SHARED
SUM no_grav,mon1,no_grav+mon1 ;
TO neto,iva,totalticket
CLEAR
? 'Total Neto      :', neto
? 'Total Impuestos:', iva
? 'Total Neto + Impuestos:',totalticket
```

Str()

Devuelve el carácter equivalente a una expresión numérica. *También convierte un valor número a cadena de carácter* , para el caso de tratamiento de fechas, por ejemplo.

STR(nExpression [, nLength [, nDecimalPlaces]])

```
? STR(22779819.627788,12,3)
```

```
22779819.628
```

El valor devuelto será, un total de doce posiciones incluyendo al punto, y tres decimales.

Upper(), Lower(), Proper()

UPPER(cExpression)

LOWER(cExpression)

PROPER(cExpression)

Podemos convertir cadenas de caracteres tanto en MAYUSCULAS, minúsculas o bien la primer letra en Mayúsculas y las demás en minúsculas. Ejemplo:

```
upp = " hello world"
low = " YOU WIN"
pro = " fabian"
? UPPER(upp)
? LOWER(low)
? PROPER(pro)
```

```
HELLO WORLD
you win
Fabian
```

VAL() (Función)

Devuelve un valor numero o de moneda de una expresión de caracteres compuesta por números. Podemos usar VAL () para convertir una cadena de caracteres devuelto por VFP con SYS (), funciones para valores numéricos. Esta conversión se hace necesaria en casos de que en nuestra base de datos necesitemos convertir valores numéricos que han sido ingresados como alfanuméricos siendo ellos numéricos. Para ello creamos un programa tipo .prg

VAL(cExpression)

Argumentos: ***cExpression*** Especifica una expresión de caracteres compuesta de hasta 16 números. Si en ***cExpression*** se incluyen más de 16 números, se redondeará.

```
STORE SPACE(8) TO a,b,names
a = "45"
b = "20"
names="Jack Ed"
* here converted values alfanumerics to numerics
c = VAL(a) + VAL(b)
? names
? c
```

El resultado sera:

```
Jack Ed
65.00
45      20
Jack Ed
65.00
```

Upper(), Lower (), Proper ()

UPPER(cExpression)

LOWER(cExpression)

PROPER(cExpression)

Podemos convertir cadenas de caracteres tanto en MAYUSCULAS, minúsculas o bien la primer letra en Mayúsculas y las demás en minúsculas. Ejemplo:

```
upp = " hello world"  
low = " YOU WIN"  
pro = " fabian"  
? UPPER(upp)  
? LOWER(low)  
? PROPER(pro)
```

```
HELLO WORLD  
you win  
Fabian
```

Ltrim (), Rtrim () y Alltrim ()

LTRIM(Expression [, nFlags] [, cParseChar [, cParseChar2 [, ...]]])

RTRIM(cExpression [, nFlags] [, cParseChar [, cParseChar2 [, ...]]])

ALLTRIM(Expression [, nFlags] [, cParseChar [, cParseChar2 [, ...]]])

Para las tres funciones, la utilidad que tenemos es que podemos quitar los que pudieran tener una cadena caracteres, ya sea esta desde la izquierda, por la derecha o por ambos lados.
Valor devuelto: Carácter.

FUNCIONES LOGICAS.

BETWEEN() (Función)

Determina si el valor de una expresión queda dentro de los valores de otras dos expresiones del mismo tipo de datos. Valor devuelto: Logico o Valor Nulo.

BETWEEN(eTestValue, eLowValue, eHighValue)

Parametros:

eTestValue. Especifica la expresión cuyo valor verifica BETWEEN(). Si el valor de ***eTestValue*** es mayor o igual que el valor de ***eLowValue*** y menor o igual que el valor de ***eHighValue***, BETWEEN() devolverá verdadero (.T.). De lo contrario, BETWEEN() devolverá falso (.F.). BETWEEN() devuelve el valor nulo si ***eLowValue*** o ***eHighValue*** son el valor nulo.

eLowValue especifica el valor inferior del intervalo que evalúa BETWEEN(). ***eHighValue*** especifica el valor superior del intervalo que evalúa BETWEEN().

```
CLOSE DATABASES
OPEN DATABASE c:\ivaempre\bases\contenedor.dbc SHARED
USE compras
CLEAR
SCAN FOR BETWEEN (no_grav,73.10,78.50)
? proveedor,no_grav
ENDSCAN
```

DELETED() (Función)

Devuelve un valor lógico que indica si el registro actual está marcado para su eliminación. Valor devuelto: Logico

DELETED([cTableAlias / nWorkArea])

Parámetros:

cTableAlias | *nWorkArea*. Puede comprobar el estado del registro actual de una tabla abierta en otra área de trabajo si especifica el número de área de trabajo con *nWorkArea* o el alias de la tabla con *cAliasTabla*. Si una tabla no está abierta en el área de trabajo que especifique, DELETED() devolverá falso.

Si omite *cTableAlias* y *nWorkArea*, el estado de eliminado que se devuelve es el del registro actual del área de trabajo actual.

```
OPEN DATABASE c:\ivaempre\bases\contenedor.dbc SHARED
USE COMPRAS
DELETE FROM COMPRAS WHERE proveedor = 'CIMET CABLES'
CLEAR
LIST FIELDS dia,proveedor FOR DELETED () && Lista Registros Marcados
WAIT WINDOW " Ahora recuperamos los archivos marcados!!"
RECALL ALL
```

EMPTY() (Función)

Determina si una expresión está vacía o no. Tipos devueltos: **Caracter**, **Numeric**, **Date**, **Varbinary**, **Blob**, o **Logical** tipo, o un campo Memo o General dentro de una tabla abierta.

EMPTY(eExpression)

```
OPEN DATABASE c:\ivaempre\bases\contenedor.dbc SHARED
USE COMPRAS
IF EMPTY(PROVEEDOR) = .T. THEN
    =MESSAGEBOX('EL CAMPO ESTA VACIO')
    ? PROVEEDOR
ELSE
    =MESSAGEBOX('EL CAMPO CONTIENE VALORES')
    ? PROVEEDOR
ENDIF
```


IIF() (Función)

Devuelve uno de los dos valores dependiendo del valor de una expresión lógica. Valores devueltos: **Character, Numeric, Currency, Date, or DateTime**

IIF(lExpression, eExpression1, eExpression2)

Esta función es muy útil, en la construcción de Vistas y de informes.

```
CLOSE DATABASES
OPEN DATABASE c:\ivaempre\bases\contenedor.dbc SHARED
USE compras
CLEAR
SCAN
    ? IIF(EMPTY(proveedor), 'saldo', proveedor)
ENDSCAN
```

SEEK() (Función)

Busca en una tabla indizada la primera aparición de un registro cuya clave de índice coincida con una expresión especificada. SEEK() devuelve un valor lógico que indica si la búsqueda tuvo éxito. Valor devuelto: Logical

*SEEK(eExpression [, nWorkArea | cTableAlias
[, nIndexNumber | cIDXIndexFileName | cTagName]])*

```
CLEAR
CLOSE DATABASES
OPEN DATABASE c:\ivaempre\bases\contenedor.dbc SHARED
USE compras && Opens Customer table
SET ORDER TO PROVEEDOR && DIA
? SEEK ( 'WARBEL' )
```

A modo de aclaración, en el desarrollo de las aplicaciones desarrollaremos la *Funcion Seek*.

FUNCIONES DE FECHAS.

CDOW() (Función)

Retorna el día de la semana a partir de una expresión de Date o de DateTime dada. Tipo devuelto: Carácter.

CDOW(dExpression / tExpression)

```
* Year - Month - Day
STORE {^2010-05-15} TO gdDate
MB='MY BIRTHDAY'
CLEAR
? CDOW(gdDate)+ ' ' + MB    && Displays Saturday
```

CMONTH() (Función)

Devuelve el nombre del mes a partir de una expresión de fecha o de DateTime dada. Tipo devuelto: Carácter.

CMONTH(dExpression / tExpression)

```
CLEAR
? CMONTH( DATE(  ) )
STORE {^2008-05-15} TO CurrentDate
? 'Your Birthday is in ', CMONTH(CurrentDate)
STORE CurrentDate+360 TO NextBirthDay
? 'You Next BirthDay Will Be ', CMONTH(NextBirthDay)
```

CTOD() (Función)

Convierte una expresión de caracteres en una expresión de fecha. Tipo devuelto : Date

DTOC(dExpression / tExpression [, 1])

El formato predeterminado de **dExpression** es mm/dd/aa. Puede utilizar SET DATE y SET CENTURY para cambiar el formato predeterminado.

```
SET CENTURY ON && FOR LONG DATE
SET DATE TO GERMAN && ALSO IS VALID FOR RUSSIAN FORMAT
STORE CTOD('15/05/2008') TO gdThisDate
CLEAR
? DTOC(gdThisDate)
STORE DTOC({^2009-05-15}+365) TO NextBirthDay
? 'Your Next BirthDay is ', NextBirthDay
? DTOC({^2008-05-15})&& FORMAT USA
```

DATE() (Función)

Devuelve la fecha actual del sistema, que está controlada por el sistema operativo. Tipo devuelto: Date.

DATE([nYear, nMonth, nDay])

```
CLEAR
SET CENTURY OFF
SET DATE TO BRITISH
? DATE( ) && Displays today's date without the century
SET CENTURY ON
? DATE( ) && Displays today's date with the century
? DATE(2009, 04, 19) && Displays a year 2000-compliant Date value
```

DATETIME() (Función)

Devuelve la fecha y la hora actuales como un valor DateTime. Tipo devuelto: DateTime

DATETIME([nYear, nMonth, nDay [, nHours [, nMinutes [, nSeconds]]]])

```
tNewyear = DATETIME(YEAR( DATE( ) ) + 1, 1, 1) && Next New Year
tToday = DATETIME( )
nSecondstoneyear = tNewyear - tToday
```

```
CLEAR
? "There are " + ALLTRIM (STR(nSecondstoneyear)) + " seconds to the next
New Year."
CLEAR
```

```
SET CENTURY ON
SET DATE TO AMERICAN
? tNewyear
? tToday
? DATETIME(1998, 02, 16, 12, 34, 56) && Displays 02/16/1998 12:34:56 PM
```

DAY() (Función)

Devuelve el número del día del mes correspondiente a una expresión de Date o de DateTime dada. Tipo devuelto: Numérico.

DAY(dExpression / tExpression)

```
STORE {^2009-04-21} TO gdBDate
CLEAR
? CDOW(gdBDate) && Displays Thursday
? DAY(gdBDate) && Displays 21
? 'That date is ', CMONTH(gdBDate), STR(DAY(gdBDate),2)
```

DMY() (Función)

Devuelve una expresión de caracteres con el formato día-mes-año (por ejemplo, 31 de mayo de 1996) a partir de una expresión de Date o de DateTime. El nombre del mes no se abrevia. Tipo devuelto: Carácter.

DMY(dExpression / tExpression)

```
CLEAR
SET CENTURY OFF
? DMY( DATE( ) )
SET CENTURY ON
? DMY( DATE( ) )
```

DOW() (Función)

Devuelve el número del día de la semana a partir de una expresión Date o DateTime dada. Valor devuelto: Numérico.

DOW(dExpression / tExpression [, nFirstDayOfWeek])

```
STORE DATE( ) TO gdDayNum
CLEAR
? DOW(gdDayNum)
? CDOW(gdDayNum)
```

DTOC() (Función)

Devuelve una fecha de tipo Carácter a partir de una expresión de tipo Date o DateTime. Valor devuelto: Carácter.

DTOC(dExpression / tExpression [, 1])

```
SET STRICTDATE TO 1
STORE CTOD('12/31/2009') TO gdThisDate
CLEAR
? DTOC(gdThisDate)
STORE DTOC({^2009-12-31}+90) TO gcExpireDate
? 'Your 90-day warranty expires ', gcExpireDate
? DTOC({^2009-12-31},1)
```

DTOS() (Función)

Devuelve una cadena de caracteres de fecha con el formato AAAAMMDD a partir de una expresión de Date o de DateTime especificada. Valor devuelto: Carácter.

DTOS(dExpression / tExpression)

```
CLEAR

? DTOS( DATE( ) )
```

DTOT() (Función)

Devuelve un valor de DateTime a partir de una expresión de Date. Valor devuelto: DateTime

DTOT(dDateExpression)

```
SET HOURS TO 12
? DTOT({^2009-12-31})  && Displays 02/16/2004 12:00:00 AM
```

GOMONTH() (Función)

Devuelve la fecha que es especificada en numero o meses antes o después en expresión de Date o DateTime

GOMONTH(dExpression / tExpression, nNumberOfMonths)

```
CLEAR
? GOMONTH({^1998-12-31}, 2)  && Displays 02/28/1999
? GOMONTH({^1998-12-31}, -2) && Displays 10/31/1998
```

 HOUR() (Función)

Devuelve la hora de una expresión DateTime. Valor devuelto: Numérico.

HOUR(tExpression)

```
CLEAR
? HOUR(DATETIME( ))
? HOUR({^2009-04-30 10:42a})  && Displays 10
```

MDY() (Función)

Devuelve la expresión de tipo Date o DateTime especificada en formato mes -día-año con el nombre completo del mes. Tipo devuelto: Carácter.

MDY(dExpression / tExpression)

```

SET CENTURY OFF
CLEAR
? Longdate({^1998-02-16})  && Displays Monday, February 16, 98
SET CENTURY ON
  Longdate({^1998-02-16})  && Displays Monday, February 16, 1998
*** LongDate ***
FUNCTION longdate
PARAMETERS gdDate
RETURN CDOW(gdDate) + ', ' + MDY(gdDate)

```

MINUTE() (Función)

Devuelve los minutos de una expresión DateTime. Valor devuelto: Numérico.

MINUTE(tExpression)

```

CLEAR
? MINUTE(DATETIME( ))
? MINUTE({^1998-02-16 10:42a})  && Displays 42

```

MONTH() Función

Devuelve el número de mes de una expresión determinada de tipo Date o DateTime. Valor devuelto: Numérico.

MONTH(dExpression / tExpression)

```

CLEAR
? DATE( )  && Displays today's date
? MONTH(DATE( ))  && Displays the month number
STORE {^2009-05-10} TO gdBuy
STORE MONTH(gdBuy + 31) TO gdMonth

? gdMonth  && Displays 6

```

SEC() (Función)

Devuelve los segundos de una expresión DateTime. Valor devuelto: Numérico.

SEC(tExpression)

CLEAR

? SEC(DATETIME())

? SEC({^2009-04-21 10:42:25AM}) && Displays 25

SECONDS() (Función)

Devuelve el número de segundos que transcurrieron desde la medianoche. Valor devuelto: Numérico.

SECONDS()

CLEAR

? SECONDS()

? SECONDS() / (60 * 60)

TIME() (Función)

Devuelve la hora actual del sistema en formato de 24 horas, en una cadena de 8 caracteres (HH: MM:SS). Valor devuelto: Carácter.

TIME([nExpression])

TTOC() (Función)

Devuelve un valor de tipo Carácter a partir de una expresión Da teTime.

TTOC(tExpression [, 1 | 2 | 3])


```
CLEAR
STORE DATETIME( ) TO gtDateTime
? "gtDateTime has type: "+TYPE('gtDateTime')
gtDateTime = TTOC(gtDateTime)

? "gtDateTime is now type: "+TYPE('gtDateTime')
```

T TOD() (Función)

Devuelve un valor de tipo Date a partir de una expresión de DateTime. Valor devuelto: Date.

T TOD(tExpression)

```
STORE DATETIME( ) TO gtDtime  && Creates a Datetime type memory variable
CLEAR
? "gtDtime is type: "
?? TYPE('gtDtime')  && Displays T, Datetime type value
gtDtime = TTOD(gtDtime)  && Converts gtDtime to a date value
? "gtDtime is now type: "
?? TYPE('gtDtime')  && Displays D, character type value
```

WEEK() (Función)

Devuelve un número que representa la semana del año a partir de una expresión de Fecha o FechaHora. Valor devuelto: Numérico.

WEEK(dExpression | tExpression [, nFirstWeek] [, nFirstDayOfWeek])

```
CLEAR
? WEEK( DATE( ) )
? WEEK( {^2010-02-28} )  && Displays 10
```

Parametros:

dExpresión | tExpresión Especifica la expresión de Fecha o de FechaHora para la cual WEEK() devuelve la semana del año.

Si omite los Argumentos: opcionales nPrimeraSemana y nPrimerDíaSemana, WEEK() utilizará el domingo como primer día de la semana.

nPrimeraSemana Especifica los requisitos para la primera semana del año.
nPrimeraSemana puede tener uno de los valores siguientes.

- 0 WEEK() devuelve la semana seleccionada actualmente en el cuadro de lista “Primera semana del año” de la ficha Internacional, en el cuadro de diálogo Opciones.
- 1 La primera semana contiene 1 de enero. Este es el valor predeterminado cuando se omite nPrimeraSemana.
- 2 La mitad mayor (cuatro días) de la primera semana está en el año actual.
- 3 La primera semana tiene siete días. nPrimerDíaSemana Especifica el primer día de la semana. nPrimerDíaSemana puede tener uno de los valores siguientes.

- 0 WEEK() devuelve el día seleccionado actualmente en el cuadro de lista “La semana empieza en” de la ficha Internacional, en el cuadro de diálogo Opciones .
- 1 Domingo. Éste es el valor predeterminado cuando se omite nPrimerDíaSemana, y es el primer día de la semana empleado en versiones anteriores de FoxPro.
- 2 Lunes
- 3 Martes
- 4 Miércoles
- 5 Jueves
- 6 Viernes
- 7 Sábado

YEAR() (Función)

Devuelve el año a partir de la expresión de fecha o fecha-hora especificada. Valor devuelto: Numérico.

YEAR(dExpression / tExpression

CLEAR

? YEAR (DATE ())

SQL y RushMore.

SQL (Structured Query Language) es un lenguaje universal que permite consultar y actualizar cualquier tipo de bases de datos relacionales. Recordemos antes de adentrarnos brevemente en el lenguaje SQL, que este, está íntimamente relacionado con la tecnología *RushMore*, esto no es más que la utilización de índices en sus tablas, esto a fin de agilizar el acceso, el borrado, y la actualización a la información cuando la base de datos contiene tablas con una cantidad gigantesca de datos. Visual FoxPro es pionero en usar bases de datos relacionales y en la utilización de índices. Fue tal el éxito del uso de esta tecnología, que Microsoft decidió extenderlo a otros productos tales como; Access, SQL Server y Visual Basic, así también otros fabricantes de herramientas de desarrollo decidieron utilizar esta técnica, ya que asegura consistencia y rapidez en el proceso de datos. En síntesis la tecnología *RushMore* consiste en una técnica de búsqueda interna que aprovecha los índices de la tabla para acceder a grupos de registros, técnicas optimizadas para la indexación, compresión de datos, optimización de consultas, almacenamiento de bitmaps, gestión de la entrada/salida de datos, gestión de memoria entre otras técnicas.

La premisa que debemos seguir al momento de usar esta tecnología es que la utilización de índices tienen vital importancia, como así también el uso de la cláusula **FOR**. No todas las instrucciones en VFP son optimizables; solo las que dispongan de la posibilidad de incluir la cláusula **FOR**, estas son:

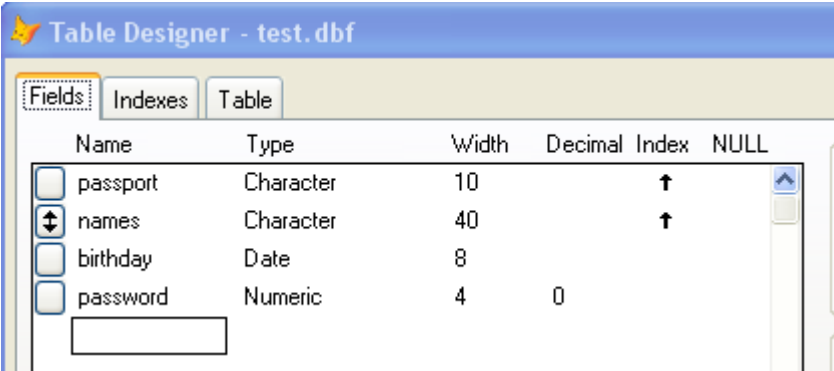
COMANDOS

Average	Index
Blank	Label
Browse	List
Calculate	Locate
Change	Recall
Copy To	Replace
Copy To Array	Replace from Array
Count	Report
Delete	Scan
Display	Sort
Edit	Sum
Export	Total

Comencemos con la creación de una tabla con de mas de 800000 registros que tendrán los siguientes campos e índices:

Accedemos desde el menú de VFP 9 y accedemos a File , New y elegimos Table y definimos los mismos campos que se muestran en la figura de abajo. Los índices serán tipo “regular” sobre los campos passport,names:

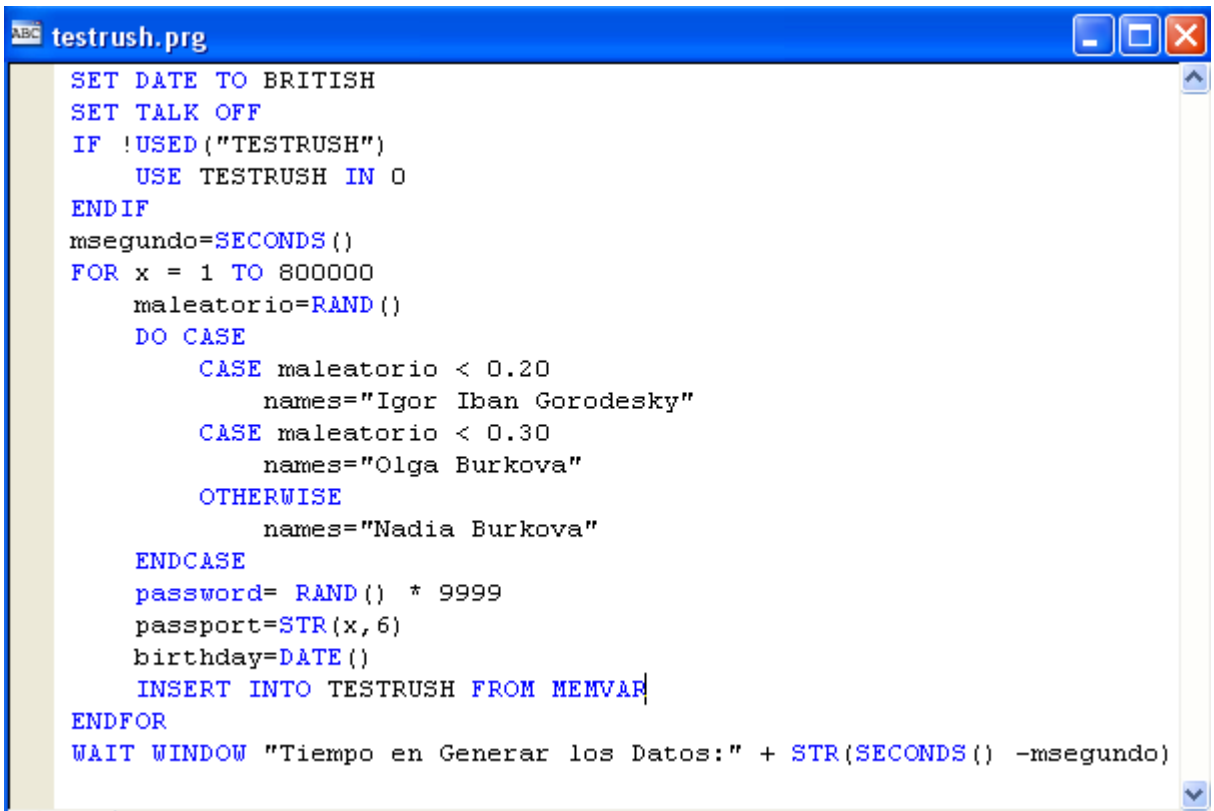
TABLE NAME : **TEST.DBF**



Name	Type	Width	Decimal	Index	NULL
passport	Character	10		↑	
names	Character	40		↑	
birthday	Date	8			
password	Numeric	4	0		

Esta tabla la sera llenada con información pseudos-aleatoria sobre la que realizaremos busquedas. Luego de crear la tabla y de establecer sus indices, desde el menú principal de VFP 9 hacemos clic en **File** y elegimos **New** y hacemos clic sobre **program**, crearemos el programa que lo llamaremos **testrush.prg**

El codigo debe ser escrito de la siguiente manera como lo muestra la si guiente figura: basta con tener una pc superior a 256 mb de R.A.M con un procesador superior a 1 GHz para que en cuestion de segundos podamos crear 500000 registros. Con este sencillo ejemplo podemos ver como responde VFP 9. para no entrar en cientos de ejemplos innecesarios, debemos decir que sie mpre es mejor trabajar con indices ya que esto agiliza los procesos, en caso de no querer trabajar con indice la diferencia de datos es notoria cuando trabajemos con tablas superior a 1 GB, lógicamente que para lograr este tamaño en una tabla tendremos que estar muchos años cargando datos en forma continua y con un centenar de operadores cargando datos, como es en el caso de realizar un padrón de datos de votantes de una provincia o estado determinado en que existen mas de un millón de votantes.



```
testrush.prg
SET DATE TO BRITISH
SET TALK OFF
IF !USED("TESTRUSH")
    USE TESTRUSH IN 0
ENDIF
msegundo=SECONDS()
FOR x = 1 TO 800000
    maleatorio=RAND()
    DO CASE
        CASE maleatorio < 0.20
            names="Igor Iban Gorodesky"
        CASE maleatorio < 0.30
            names="Olga Burkova"
        OTHERWISE
            names="Nadia Burkova"
    ENDCASE
    password= RAND() * 9999
    passport=STR(x,6)
    birthday=DATE()
    INSERT INTO TESTRUSH FROM MEMVAR
ENDFOR
WAIT WINDOW "Tiempo en Generar los Datos:" + STR(SECONDS() -msegundo)
```

LENGUAJE SQL.

La mayor ventaja que tenemos al usar SQL, es que estamos usando toda la potencia de la tecnología RushMore, basada en los índices, ya que estos comandos operan sobre conjuntos de registros. Comencemos a trabajar con ellos.

LENGUAJE SQL.

SELECT

3. Select – SQL en visual FOX PRO 9.0 sp2 *

Visual FoxPro admite comandos de Lenguaje de consultas estructurado SQL. Los comandos SQL de Visual FoxPro utilizan la tecnología Rushmore para optimizar el rendimiento y puede utilizarse un sólo comando SQL para sustituir a varios comandos Visual FoxPro.

Visual FoxPro admite los siguientes comandos SQL:

SELECT - SQL: Especifica los criterios en los que se basa una consulta y ejecuta la consulta. Visual FoxPro interpreta la consulta y recupera los datos especificados de la tabla o tablas. El comando **SELECT** se construye dentro de Visual FoxPro como cualquier otro comando de Visual FoxPro.

Puede crear una consulta con el comando **SELECT**

- En la ventana Comandos.
- En un programa Visual FoxPro (como cualquier otro comando de Visual FoxPro).

En el Diseñador de consultas.

ALTER TABLE – SQ: Modifica una tabla existente. Puede modificar el nombre, el tipo, la precisión, la escala, la admisión de un valor nulo y las reglas de integridad referencial para cada campo de la tabla.

CREATE CURSOR – SQL: Crea una tabla temporal. Cada campo de la tabla temporal se define con un nombre, tipo, precisión, escala, soporte de valor nulo y reglas de integridad referencial. Las definiciones pueden obtenerse del propio comando o de una matriz.

CREATE TABLE – SQL: Crea una tabla. Cada campo de la tabla nueva se define con un nombre, tipo, precisión, escala, aceptación de valores nulos y reglas de integridad referencial. Estas definiciones pueden obtenerse del propio comando o de una matriz.

DELETE – SQL: Marca para su eliminación los registros de una tabla mediante la sintaxis de SQL.

INSERT – SQL: Anexa un registro al final de una tabla existente. El nuevo registro incluye los datos mostrados en el comando **INSERT** o incluidos en la matriz especificada.

UPDATE – SQL: Actualiza los registros de una tabla. Los registros se pueden actualizar según los resultados de una instrucción **SELECT – SQL**

El propósito de este material es brindar al estudiante ejemplos que le permitan utilizar **SELECT – SQL** en el diseño de consultas usando para ello una o más tablas diseñadas en Visual Fox Pro.

* Trabajo que corresponde a: *Carlos Roberto Izquierdo González.*

Componentes del SQL

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

Comandos

Existen dos tipos de comandos SQL:

- los **DLL** que permiten crear y definir nuevas bases de datos, campos e índices.
- los **DML** que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos. Estos son los que serán más utilizados.

Comandos DLL

Comando Descripción

CREATE Utilizado para crear nuevas tablas, campos e índices

DROP Empleado para eliminar tablas e índices

ALTER Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

Comandos DML

Comando Descripción

SELECT Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado

INSERT Utilizado para cargar lotes de datos en la base de datos en una única operación.

UPDATE Utilizado para modificar los valores de los campos y registros especificados

DELETE Utilizado para eliminar registros de una tabla de una base de datos

Cláusulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

Cláusula	Descripción
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros
WHERE	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar
GROUP BY	Utilizada para separar los registros seleccionados en grupos específicos
HAVING	Utilizada para expresar la condición que debe satisfacer cada grupo
ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico

Operadores Lógicos

Operador	Uso
AND	Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdadero si alguna de las dos es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

Operadores de Comparación

Operador	Uso
<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor ó Igual que
>=	Mayor ó Igual que
=	Igual que
BETWEEN	Utilizado para especificar un intervalo de valores.
LIKE	Utilizado en la comparación de un modelo
In	Utilizado para especificar registros de una base de datos

CONSULTAS BÁSICAS

La sintaxis básica de una consulta de selección es la siguiente:

```
SELECT Campos FROM Tabla
```

En donde campos es la lista de campos que se deseen recuperar y tabla es el origen de los mismos, por ejemplo:

```
SELECT Nombre, Teléfono FROM Clientes
```

Esta consulta devuelve una visualización de la tabla Clientes con el campo nombre y teléfono. Esta visualización en VFP es un browse pero no es necesario agregar esta orden, SQL lo invoca automáticamente. Vea el orden de los campos en el browse.

Ordenar Los Registros

Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula ORDER BY Lista de Campos. En donde Lista de campos representa los campos a ordenar.

Ejemplo:

```
SELECT Cod_Postal, Nombre, Telefono FROM Clientes ORDER BY Nombre
```

Esta consulta devuelve los campos CodigoPostal, Nombre, Teléfono de la tabla Clientes ordenados por el campo Nombre.

Se pueden ordenar los registros por más de un campo, como por ejemplo:

```
SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY Cod_Postal, Nombre
```

Incluso se puede especificar el orden de los registros: ascendente mediante la cláusula (ASC -se toma este valor por defecto) ó descendente (DESC)

```
SELECT CodigoPostal, Nombre, Teléfono FROM Clientes ORDER BY CodigoPostal, nombre ASC
```

Consultas Con Predicado

El predicado se incluye entre la cláusula y el primer nombre del campo a recuperar, los posibles predicados son:

Predicado	Descripción
*	Devuelve todos los campos de la tabla
TOP	Devuelve un determinado número de registros de la tabla
DISTINCT	Omite los registros cuyos campos seleccionados coincidan totalmente
DISTINCTROW	Omite los registros duplicados basándose en la totalidad del registro y no sólo en los campos seleccionados.

* (ALL)

Si no se incluye ninguno de los predicados se asume ALL. El Motor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL. No es conveniente abusar de este predicado ya que obligamos al motor de la base de datos a analizar la estructura de la tabla para averiguar los campos que contiene, es mucho más rápido indicar el listado de campos deseados.

SELECT * FROM Clientes

TOP

Devuelve un cierto número de registros que entran entre al principio o al final de un rango especificado por una cláusula ORDER BY. Supongamos que queremos recuperar los nombres de los tres primeros clientes tomando como referencia un orden ASC para el campo cedula

SELECT TOP 3 cedula, Nombre FROM clientes ORDER BY cedula DESC

El resultado de la consulta mostrará en pantalla los tres primeros registros que correspondan al orden. Observe en la figura los números de cedula.

Como la sentencia ORDER BY cedula DESC indica que se deben ordenar en orden descendente los registros, se muestran los tres primeros contando del último en adelante.

Observe el cambio en el resultado de la consulta si ORDER BY cedula ASC

SELECT TOP 3 cedula, Nombre FROM clientes ORDER BY cedula ASC

El orden establecido para el campo cedula es ASC, por lo tanto toma los tres primeros registros en forma ascendente.

Si no se incluye la cláusula ORDER BY, la consulta devolverá un conjunto arbitrario de 3 registros de la tabla Clientes. El predicado TOP no elige entre valores iguales. Si se incluye la palabra clave PERCENT, se redondeará al número entero más alto el número de columnas devuelto en el resultado. Los valores permitidos para *nExpr* cuando se incluye la palabra clave PERCENT son 0.01 a 99.99.

Ejemplo: La tabla Clientes tiene un total de 10 registros, si se quiere visualizar en pantalla un 20% ordenado por cedula en orden ascendente la línea de código será:

SELECT TOP 20 PERCENT cedula, nombre FROM Clientes ORDER BY cedula ASC

Y el resultado de la búsqueda como se puede apreciar son dos registros correspondientes al 20 % de 10 que hay en total

El valor que va a continuación de TOP debe ser un Integer sin signo.

DISTINCT

Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta deben ser únicos.

Por ejemplo, suponga que se desea saber los códigos de área de los números de teléfono de los clientes almacenados en la tabla. Es de suponer que con que aparezca una sola vez 0274 o 0416 es suficiente así existan otros clientes con el mismo código de área.

```
SELECT DISTINCT SUBSTR(telefono, 1, 4) from clientes
```

En vista que el código de área y el número del teléfono están integrados en un atributo atómico, se debe extraer el dato que se necesita para la consulta usando la Función SUBSTR(), y así en la consulta se podrán observar los diferentes códigos de área que conforman los teléfonos de los clientes.

Vista de la consulta usando DISTINCT

Con otras palabras el predicado DISTINCT devuelve aquellos registros cuyos campos indicados en la cláusula SELECT posean un contenido diferente. El resultado de una consulta que utiliza DISTINCT no es actualizable y no refleja los cambios subsiguientes realizados por otros usuarios.

En el ejemplo anterior también se puede apreciar como se mezcla de forma eficiente código de SQL con el de VFP y se consiguen mejores resultados.

Criterios de selección

Se vio la forma de recuperar los registros de las tablas, las formas empleadas devolvían todos los registros de la mencionada tabla. A lo largo de este apartado se estudiarán las posibilidades de filtrar los registros con el fin de recuperar solamente aquellos que cumplan una o unas condiciones preestablecidas

WHERE

Indica a Visual FoxPro que incluya únicamente ciertos registros en el resultado de la consulta. WHERE es necesario para recuperar datos de varias tablas. Para las condiciones de filtro se puede utilizar cualquiera de los siguientes operadores:

Operador	Comparación
=	Igual
==	Exactamente igual
LIKE	SQL LIKE
<>	Distinto de
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que

Cuando utiliza el operador = con cadenas, actúa de forma distinta dependiendo del ajuste de SET ANSI. Cuando SET ANSI está OFF, Visual FoxPro trata las comparaciones de cadenas en la forma

Habitual en Xbase. Cuando SET ANSI está a ON, Visual FoxPro sigue las normas ANSI para comparaciones de cadenas. Vea SET ANSI y SET EXACT para obtener información adicional sobre la forma en que Visual FoxPro realiza las comparaciones de cadenas.

Ejemplo 1

Suponga que deseamos conocer todos los clientes Movilnet (código 0416) de la tabla y además queremos enmascarar el título del campo. Para esto la línea de código SQL será:

```
SELECT nombre AS Clientes_MovilNet FROM clientes WHERE substr (telefono,1,4) = '0416'
```

SELECT nombre = selecciona el campo nombre de la tabla

AS Clientes_Movilnet = crea la mascara para la consulta. Se debe tomar en cuenta que no pueden existir espacios en blanco en la cadena de caracteres.

FROM clientes = establece el origen de los datos para la consulta.

WHERE substr (telefono, 1,4) = '0416' => criterio de filtro para la consulta.

Ejemplo 2

Se necesita conocer el nombre y el teléfono de los clientes con crédito hasta Bs. 100.000,00

```
SELECT nombre AS Credito_Hasta_100000, telefono FROM clientes WHERE credito <= 100000
```

Ejemplo 3

Diseñar una consulta SQL de todos los clientes foráneos (código postal diferente de 5101).

Para resolver este problema usaremos la cláusula IN la cual indica que el campo debe contener uno de los valores antes de que el registro se incluya en los resultados de la consulta. Pero para este caso en particular la negaremos con NOT

```
SELECT NOMBRE AS CLIENTES_FORANEOS, TELEFONO, COD_POSTAL FROM CLIENTES;
```

```
WHERE COD_POSTAL NOT IN ('5101')
```

Observe el resultado de la consulta, están todos los clientes a excepción de los que tienen como código postal 5101.

Intervalos de valores

El operador BETWEEN

Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo emplearemos el operador BETWEEN cuya sintaxis es:

Campo [NOT] BETWEEN valor1 AND valor2 (la condición Not es opcional)

En este caso la consulta devolvería los registros que contengan en "campo" un valor incluido en el intervalo valor1, valor2 (ambos inclusive). Si antepone la condición Not devolverá aquellos valores no incluidos en el intervalo.

Ejemplo 1

Se necesita conocer los datos de los clientes que tienen una carta de crédito entre dos y tres millones de Bolívares.

La línea de código SQL será:

```
SELECT NOMBRE AS CREDITO_ENTRE_200000_Y_3000000, TELEFONO, COD_POSTAL,;  
CREDITO FROM CLIENTES WHERE CREDITO BETWEEN 2000000 AND 3000000
```

El operador like

Se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es:

Expresión LIKE modelo

En donde expresión es una cadena modelo o campo contra el que se compara expresión. Se puede utilizar el operador LIKE para encontrar valores en los campos que coincidan con el modelo especificado. Por modelo puede especificar un valor completo (Ana María), o se pueden utilizar caracteres comodín para encontrar un rango de valores (LIKE 'An%').

El operador LIKE se puede utilizar en una expresión para comparar un valor de un campo con una expresión de cadena. Por ejemplo, si introduce LIKE 'C%' en una consulta SQL, la consulta devuelve todos los valores de campo que comiencen por la letra C. En una consulta con parámetros, puede hacer que el usuario escriba el modelo que se va a utilizar.

Puede utilizar el signo de porcentaje (%) y subrayado (_) como parte de la expresión. El signo de porcentaje representa a cualquier secuencia de caracteres desconocidos en la cadena. El subrayado representa un solo carácter desconocido en la cadena.

Ejemplo

Listar por pantalla todos los clientes cuyos nombres comiencen con la letra 'I'

La línea SQL será:

```
SELECT NOMBRE AS CLIENTES_CUYOS_NOMBRES_EMPIEZAN_POR_I FROM CLIENTES;  
WHERE UPPER (NOMBRE) LIKE 'I%'
```

Observe el resultado de la consulta, no importa que siga a la letra 'I' en el campo nombre, SQL lo muestra en pantalla.

8. Agrupamiento de registros

GROUP BY *ColumnaGrupo* [, *ColumnaGrupo*...]

Agrupar las filas de la consulta basándose en los valores de una o más columnas. *ColumnaGrupo* puede ser el nombre de un campo normal de una tabla, o un campo que incluya una función de campo SQL, o una expresión numérica indicando la posición de la columna en la tabla resultado (la columna más a la izquierda tiene el número 1).

Su sintaxis es:

```
SELECT campos FROM tabla WHERE criterio GROUP BY campos del grupo HAVING condición
```

Los valores Null en los campos GROUP BY se agrupan y no se omiten. No obstante, los valores Null no se evalúan en ninguna de las funciones SQL agregadas.

Se utiliza la cláusula WHERE para excluir aquellas filas que no desea agrupar, y la cláusula HAVING para filtrar los registros una vez agrupados.

A menos que contenga un dato Memo u Objeto OLE , un campo de la lista de campos GROUP BY puede referirse a cualquier campo de las tablas que aparecen en la cláusula FROM, incluso si el campo no está incluido en la instrucción SELECT.

Todos los campos de la lista de campos de SELECT deben o bien incluirse en la cláusula GROUP BY.

HAVING es similar a WHERE, determina qué registros se seleccionan. Una vez que los registros se han agrupado utilizando GROUP BY, HAVING determina cuáles de ellos se van a mostrar.

Si tenemos una tabla llamada clientes con los siguientes registros:

Se nos plantea la siguiente interrogante ¿Cuáles registros cumplen con la condición de poseer Código Postal 5101 y además son clientes Movilnet?

Agrupamos los clientes con Cod_Postal 5101 y luego con HAVING seleccionamos los clientes Movilnet (0416)

```
SELECT cedula, nombre, cod_postal FROM clientes WHERE cod_postal = '5101';  
GROUP BY cedula, nombre, cod_postal HAVING SUBSTR (TELEFONO,1,4) = '0416'
```

AVG

Calcula la media aritmética de un conjunto de valores contenidos en un campo especificado de una consulta. Su sintaxis es la siguiente

AVG (expr)

En donde *expr* representa el campo que contiene los datos numéricos para los que se desea calcular la media o una expresión que realiza un cálculo utilizando los datos de dicho campo. La media calculada por AVG es la media aritmética (la suma de los valores dividido por el número de valores). La función AVG no incluye ningún campo Null en el cálculo.

Ejemplo

Usando la tabla CLIENTES se desea conocer el promedio de crédito de los clientes de la ciudad de Mérida (cod_postal = 5101)

```
SELECT AVG (CREDITO) AS CLIENTES_MERIDA FROM CLIENTES;  
  
WHERE COD_POSTAL IN ('5101')
```

El resultado de la consulta será:

COUNT

Calcula el número de registros devueltos por una consulta. Su sintaxis es la siguiente

COUNT (expr)

En donde *expr* contiene el nombre del campo que desea contar. Los operandos de *expr* pueden incluir el nombre de un campo de una tabla, una constante o una función. Aunque *expr* puede realizar un cálculo sobre un campo, COUNT simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función COUNT no cuenta los registros que tienen campos NULL.

```
SELECT COUNT (campo) AS nombre_máscara FROM tabla
```

Ejemplo

Se desea determinar ¿Cuántos Clientes tienen crédito entre Bs. 100.000,00 y Bs. 300.000,00?

```
SELECT COUNT (CREDITO) AS CLIENTES_ENTRE_100000_y_300000 FROM CLIENTES;  
WHERE CREDITO BETWEEN 100000 AND 300000
```

Como puede observar SQL proporciona el número de clientes que cumplen con la condición especificada en la consulta.

Max y Min

Devuelven el mínimo o el máximo de un conjunto de valores contenidos en un campo específico de una consulta. Su sintaxis es:

```
Min (expr) (expr)  
Max (expr)
```

En donde *expr* es el campo sobre el que se desea realizar el cálculo. *Expr* puede incluir el nombre de una tabla o una constante.

Ejemplo

Se desea conocer el monto en Bs. De la mayor carta de crédito de un cliente foráneo código postal 5102

```
SELECT MAX (CREDITO) AS CLIENTE_CON_MAYOR_CRÉDITO_5102 FROM CLIENTES;  
WHERE COD_POSTAL IN ('5102')
```

```
SELECT MIN (CREDITO) AS CLIENTE_CON_MENOR_CRÉDITO_5102 FROM CLIENTES;  
WHERE COD_POSTAL IN ('5102')
```

El código SQL anterior muestra el uso de MIN para ubicar el monto de la menor carta de crédito de la zona 5102.

SUM

Devuelve la suma del conjunto de valores contenido en un campo específico de una consulta. Su sintaxis es:

```
SUM (expr)
```

En donde *expr* representa el nombre del campo que contiene los datos que desean sumarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de *expr* pueden incluir el nombre de un campo de una tabla o una constante.

Ejemplo: Se desea conocer el monto total de las carteras de crédito de los clientes código de área 5102


```
SELECT SUM (CREDITO) AS TOTAL_CRÉDITO_5102 FROM CLIENTES WHERE COD_POSTAL  
IN ('5102')
```

Este ejemplo suma los resultados de las consultas anteriores con MAX Y MIN.

Subconsultas

Una subconsulta es una instrucción SELECT anidada dentro de una instrucción SELECT, SELECT...INTO, INSERT...INTO, DELETE, o UPDATE o dentro de otra subconsulta.

Puede utilizar tres formas de sintaxis para crear una subconsulta:

Comparación [ANY | ALL | SOME] (instrucción SQL)
Expresión [NOT] IN (instrucción SQL)

[NOT] EXISTS (instrucción SQL)

En donde:

Comparación: Es una expresión y un operador de comparación que compara la expresión con el resultado de la subconsulta.

Expresión: Es una expresión por la que se busca el conjunto resultante de la subconsulta.

Instrucción SQL: Es una instrucción SELECT, que sigue el mismo formato y reglas que cualquier otra instrucción SELECT. Debe ir entre paréntesis.

Se puede utilizar una subconsulta en lugar de una expresión en la lista de campos de una instrucción SELECT o en una cláusula WHERE o HAVING. En una subconsulta, se utiliza una instrucción SELECT para proporcionar un conjunto de uno o más valores especificados para evaluar en la expresión de la cláusula WHERE o HAVING.

Se puede utilizar el predicado ANY, ALL o SOME, los cuales son sinónimos, para recuperar registros de la consulta principal, que satisfagan la comparación con cualquier otro registro recuperado en la subconsulta.

Cuando la condición de filtro incluye ANY o SOME, el campo debe cumplir la condición de comparación en al menos uno de los valores generados por la subconsulta.

Cuando la condición de filtro incluye ALL, el campo debe cumplir la condición de comparación para todos los valores generados por la subconsulta antes de que se incluya el registro en el resultado de la consulta.

Ejemplo

Usando una subconsulta muestre una lista de clientes cuyo teléfono sea Movilnet (código de área 0416)

```
SELECT nombre, credito, TELEFONO FROM clientes WHERE SUBSTR (TELEFONO, 1, 4);  
IN (SELECT TELEFONO FROM CLIENTES WHERE SUBSTR (TELEFONO, 1, 4) = '0416')
```

Ejemplo

Suponga que se desea conocer los datos personales de los clientes con cartas de crédito por arriba de Bs. 2.500.000,00

```
SELECT nombre, credito, TELEFONO FROM clientes WHERE credito = ALL ;  
(SELECT CREDITO FROM CLIENTES WHERE CREDITO > 2500000)
```

Ejemplo

Encontrar los clientes con carteras de crédito entre Bs. 2.300.000,00 Y Bs. 2.500.000

```
SELECT nombre, credito, TELEFONO FROM clientes WHERE credito = ANY;  
(SELECT CREDITO FROM CLIENTES WHERE CREDITO BETWEEN 2300000 AND 2500000)
```

**** WHERE credito = ANY (subconsulta SQL) indica que se incluya en la consulta a cualquier registro que cumpla aunque sea una parte de la condición de la subconsulta.

Ejemplo

Liste a todos los clientes que tengan residencia en la ciudad de Mérida.

```
SELECT NOMBRE, CREDITO, TELEFONO, CIUDAD FROM CLIENTES WHERE CIUDAD = ALL;  
(SELECT CIUDAD FROM CLIENTES WHERE UPPER (CIUDAD) = 'MERIDA')
```

Observe el efecto que se produce en la consulta cuando se cambia el operador = antes de ALL

```
SELECT NOMBRE, CREDITO, TELEFONO, CIUDAD FROM CLIENTES WHERE CIUDAD > ALL;  
(SELECT CIUDAD FROM CLIENTES WHERE UPPER (CIUDAD) = 'MERIDA')
```

```
SELECT NOMBRE, CREDITO, TELEFONO, CIUDAD FROM CLIENTES WHERE CIUDAD < ALL;  
(SELECT CIUDAD FROM CLIENTES WHERE UPPER (CIUDAD) = 'MERIDA')
```

Se puede ver claramente que el resultado de la consulta varía sustancialmente cuando se cambia el operador = antes de ALL. Al utilizar > se muestran en la consulta todos aquellos registros que

cumplen con la condición de comenzar con una letra mayor a 'M'. Y al utilizar < se agregan a la consulta sólo aquellos cuya letra inicial en el campo ciudad está por debajo de 'M'.

VISUAL FOXPRO 9.0 SP2 – Capítulo 3

Ejemplo

Mostrar en pantalla los clientes con cartas de crédito menores a Bs. 200.000,00

```
SELECT NOMBRE, CREDITO, TELEFONO FROM CLIENTES WHERE CREDITO < ALL;  
(SELECT CREDITO FROM CLIENTES WHERE CREDITO = 200000)
```

Con la condición del WHERE de la consulta principal se establece que se deben mostrar todos aquellos registros menores a la condición establecida en la subconsulta (credito = 200000)

Consultas De Unión Internas

Las vinculaciones entre tablas se realizan mediante la cláusula INNER que combina registros de dos tablas siempre que haya concordancia de valores en un campo común. Su sintaxis es:

```
SELECT campos FROM tb1 INNER JOIN tb2 ON tb1.campo1 OPERADOR tb2.campo2
```

En donde:

tb1, tb2: Son los nombres de las tablas desde las que se combinan los registros.

campo1, campo2: Son los nombres de los campos que se combinan. Si no son numéricos, los campos deben ser del mismo tipo de datos y contener el mismo tipo de datos, pero no tienen que tener el mismo nombre.

OPERADOR: Es cualquier operador de comparación relacional: =, <, >, <=, >=, o <>.

Se puede utilizar una operación INNER JOIN en cualquier cláusula FROM. Esto crea una combinación por equivalencia, conocida también como unión interna. Las combinaciones de equivalencia son las más comunes; éstas combinan los registros de dos tablas siempre que haya concordancia de valores en un campo común a ambas tablas. Se puede utilizar INNER JOIN con las tablas Clientes y Pedidos (ver ilustración más abajo) para seleccionar todos los pedidos de los clientes o los pedidos de un cliente en particular.

Más formalmente INNER JOIN especifica que el resultado de la consulta contenga sólo filas para una tabla con la que coincidan una o varias filas en otra tabla.

Ejemplo

Mostrar una consulta para determinar la fecha y el monto de los pedidos de un cliente X identificado por su número de cedula.

VISUAL FOXPRO 9.0 SP2 – Capítulo 3

```
SELECT fecha, monto from PEDIDOS INNER JOIN clientes ;  
ON clientes.cedula = 2 AND pedidos.cedula = 2
```

Suponga que complementamos esta consulta añadiendo los campos nombre y teléfono de la tabla clientes, el resultado por pantalla sería:

La línea de código SQL será:

```
SELECT clientes.nombre, clientes.telefono, fecha, monto FROM PEDIDOS INNER JOIN clientes;  
ON clientes.cedula = 2 AND pedidos.cedula = 2
```

Observe la sintaxis con que se escriben los campos de la tabla que no está activa (clientes). Se antecede el nombre del campo con el identificador de la tabla y un punto.

Existen órdenes agregadas que cambiarán el resultado de la consulta dependiendo como las use. Estas son:

LEFT [OUTER] JOIN: especifica que el resultado de la consulta contenga todas las filas de la tabla a la izquierda de la palabra clave JOIN y sólo las filas que concuerden procedentes de la tabla a la derecha de la palabra clave JOIN. La palabra clave OUTER es opcional; se puede incluir para resaltar que se ha creado una combinación externa.

RIGHT [OUTER] JOIN especifica que el resultado de la consulta contenga todas las filas desde la tabla hasta la derecha de la palabra clave JOIN y sólo las filas que concuerden desde la tabla hasta la izquierda de la palabra clave JOIN. La palabra clave OUTER es opcional; puede incluirse para resaltar la creación de una combinación externa.

FULL [OUTER] JOIN: especifica que el resultado de la consulta contenga todas las filas, concuerden o no, de ambas tablas. La palabra clave OUTER es opcional; se puede incluir para resaltar que se ha creado una combinación externa.

Ejemplo:

Observe como cambia el resultado de la consulta anterior agregando las órdenes LEFT, RIGHT, FULL.

LEFT

```
SELECT clientes.nombre, clientes.telefono, fecha, monto FROM PEDIDOS LEFT JOIN clientes;  
ON clientes.cedula = 2 AND pedidos.cedula = 2
```

En este caso la condición del ON es específica el campo cedula en ambas tablas debe ser igual a 2; pero LEFT indica a SQL que ingrese los campos a la izquierda de la palabra JOIN los cuales se muestran como .NULL., en vista que no cumplen con la condición cedula = 2.

RIGHT

```
SELECT clientes.nombre, clientes.telefono, fecha, monto FROM PEDIDOS RIGHT JOIN clientes ON  
clientes.cedula = 2 AND pedidos.cedula = 2
```

El resultado será (ver siguiente página)

VISUAL FOXPRO 9.0 SP2 – Capítulo 3

Los campos a la derecha de JOIN se mostrarán como .NULL. en vista que no cumplen con la condición cedula = 2.

FULL

```
SELECT clientes.nombre, clientes.telefono, fecha, monto FROM PEDIDOS FULL JOIN clientes; ON  
clientes.cedula = 2 AND pedidos.cedula = 2
```

En este caso se muestran los campos de ambas tablas coincidan o no con la condición establecida.

Consultas De Unión Externas

Se utiliza la operación UNION para crear una consulta de unión, combinando los resultados de dos o más consultas o tablas independientes. Su sintaxis es:

```
SELECT campos separados por comas FROM nombre tabla WHERE condición  
UNION; SELECT campos separados por comas FROM nombre tabla WHERE  
condición
```

Los campos de ambos SELECT deben estar en igual número en ambas instrucciones y deben ser del mismo tipo.

Ejemplo:

Suponga que tenemos una tabla VENDEDORES junto con la ya conocida tabla CLIENTES con la que hemos trabajado anteriormente.

Se le solicita que presente una vista por pantalla de todos los vendedores y clientes que se encuentran geográficamente ubicados en la zona 5101.

La línea SQL será:

```
SELECT CEDULA, NOMBRE, TELEFONO AS CLIENTES_Y_VENDEDORES FROM CLIENTES WHERE  
COD_POSTAL = '5101'; UNION SELECT CEDULA, NOMBRE, TELEFONO FROM VENDEDORES  
WHERE ZONA = '5101'
```

Observe que el único vendedor que cumple con la condición aparece en la consulta en el último lugar.

Ejemplo

Suponga ahora que desea ver a todos los vendedores y clientes en una sola vista la línea de comando SQL será más sencilla como se puede apreciar:

```
SELECT CEDULA, NOMBRE, TELEFONO AS CLIENTES_Y_VENDEDORES FROM CLIENTES  
UNION; SELECT CEDULA, NOMBRE, TELEFONO FROM VENDEDORES
```

Como podrá observar (en la siguiente página) la vista de la consulta incluye a todos los registros de la tabla clientes y a los dos únicos vendedores registrados.

Vista de la consulta de todos los clientes y vendedores usando UNION.

DISEÑO DE MENUS.

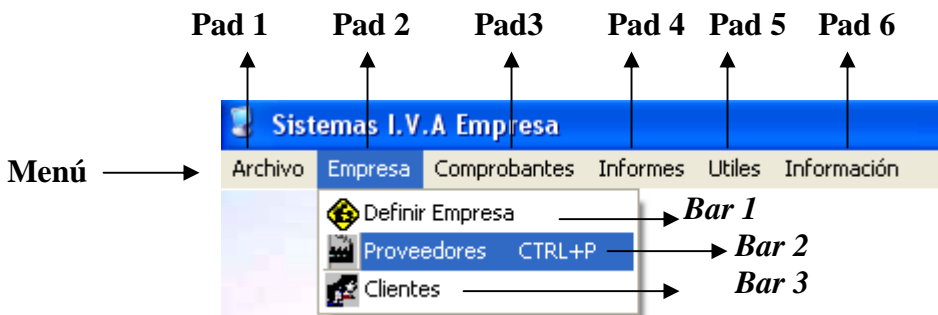
Reglas generales para diseñar menús.

- ✓ El diseño del menú debe mostrar al usuario una idea general de lo que trata la aplicación.
- ✓ Es mejor utilizar un único menú principal, que agregar varios al principal, esto hará en algunas condiciones confundir al usuario.
- ✓ Las opciones que no estén disponibles dentro del menú, deberán estar desactivadas.
- ✓ Las opciones más utilizadas deberán estar al principio de los submenús.
- ✓ Asignar teclas de función o teclas rápidas para agilizar el acceso a los submenús y opciones de la aplicación.
- ✓ Dentro de los submenús, agrupar las opciones separándolas con una línea divisoria.
- ✓ Evitar crear muchos submenús a fin de no confundir al usuario.

Menú, Pad y Bar.

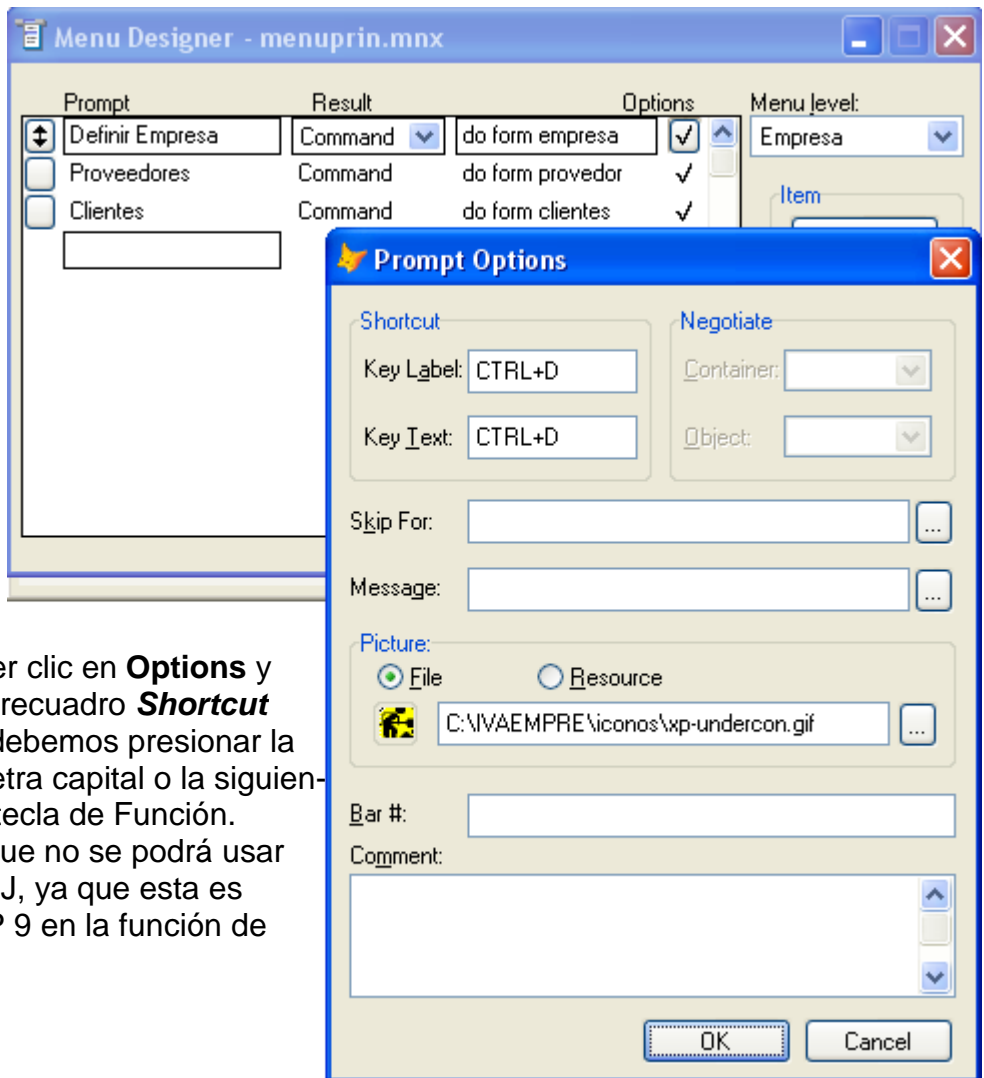
Es importante conocer la jerarquía en el diseño de menús, a fin de no confundirnos en la implementación de los mismos. La secuencia es la siguiente:

DEFINE MENU -> DEFINE PAD -> DEFINE BAR



DEFINIR TECLAS RAPIDAS.


No solo del Mouse vivirá el hombre, reza un viejo proverbio informático, por esta causa, para antiguos usuarios de Pcs, es fundamental poder usar las teclas rápidas o teclas de funciones al momento de acceder a las funciones de nuestras aplicaciones.




Debemos hacer clic en **Options** y después en el recuadro **Shortcut** en Key Label debemos presionar la tecla ctrl + la letra capital o la siguiente o cualquier tecla de Función. Recordemos que no se podrá usar La tecla ctrl. + J, ya que esta es usada por VFP 9 en la función de Escape.


INHABILITAR OPCIONES QUE NO ESTEN OPERATIVAS.

Cuando desarrollamos varias aplicaciones para distintos clientes, es normal que el tiempo no este a nuestro favor y que el cliente este apurado por usar nuestro sistema, y nos pida que le instalemos al menos las opciones que le permiten trabajar, entonces debemos **inhabilitar** las opciones que todavía no estén disponibles, los pasos a seguir serian los siguientes, tanto para los **pad** y **Bar**. Lo haremos en forma grafica a fin de hacerlo mas claro.

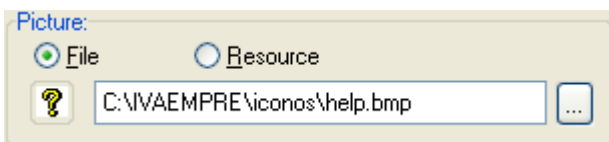
1) en el **Diseñador de Menu**, ingresamos varios **pad**, entre ellos como muestra la figura: Archivo, Empresa, Comprobantes, Informes, Utiles, Información, **Herramientas**, y en esta ultima hacemos clic en **Result**, elegimos **Submenu**, luego hacemos clic en el tilde ☒ y partir de aquí empieza la parte mas importante para inhabilitar las opciones no disponibles, en el cuadro de **Prompt Options** en el cuadro de texto **Skip For:** .T.  Escribimos .T.

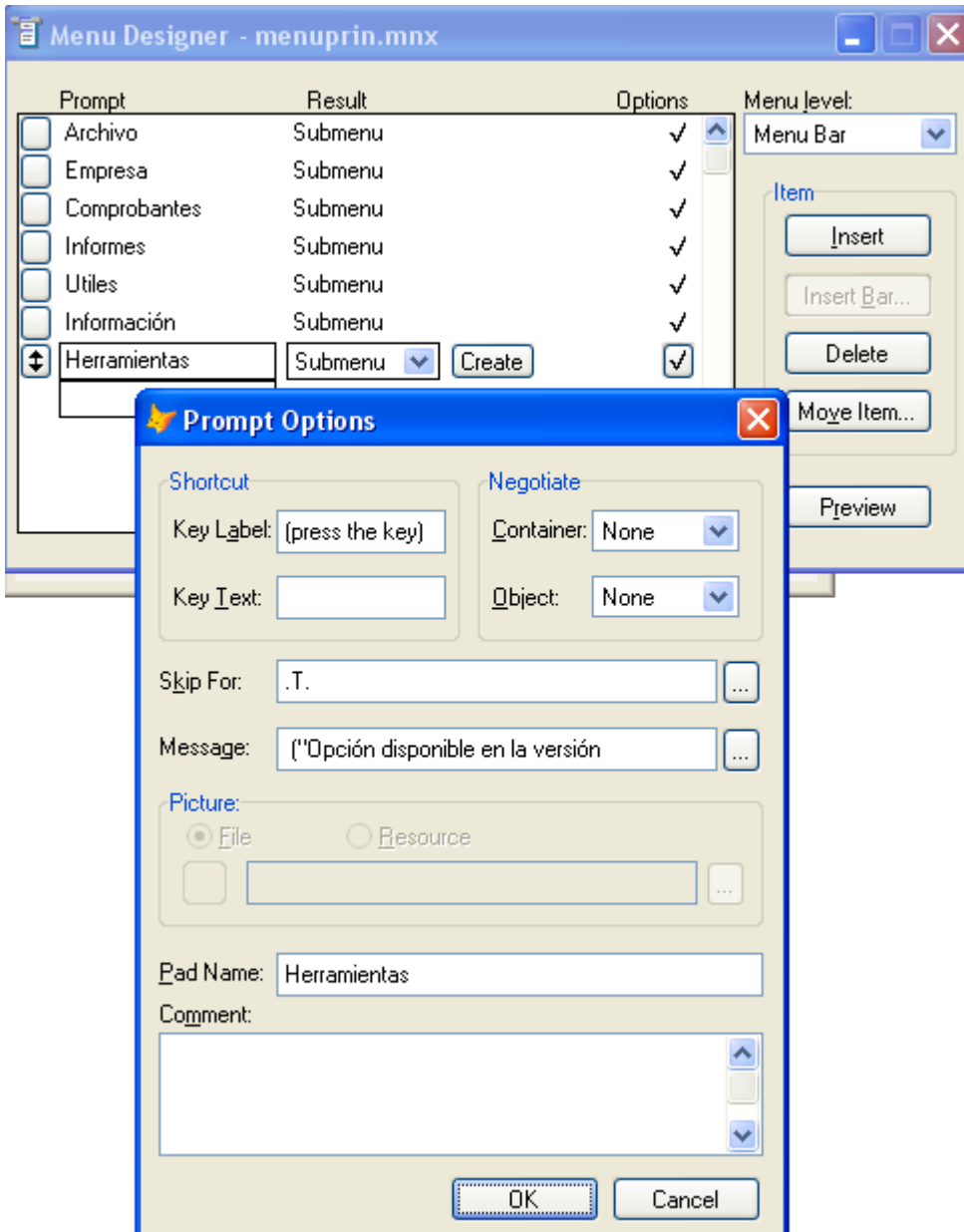
En el cuadro de texto de abajo en Message **Message:** ("Opción disponible en la versión  Escribimos ("Opción disponible en la versión profesional.

Y no debemos dejar de escribir el nombre del Pad Name, por ejemplo, Herramientas, escribir el nombre del pad es muy útil en caso de querer hacer referencia a ella desde el código de programación, por lo tanto lo deberemos escribir siempre.

Si todo hemos hecho tal cual como lo ha sido descrito, todo quedara en forma similar como la imagen de la pagina siguiente. VFP dispone para los submenús, más precisamente para los **bar**, incorporar imágenes en forma muy practica y sin código de programación, solamente debemos tener las imágenes o iconos por ejemplo dentro de una carpeta y hacemos clic en picture y seleccionamos el option File y después hacemos clic en  y seleccionamos la imagen o el icono a mostrar a la izquierda de nuestro **Bar**.

Esto hace que nuestra aplicaciones sean profesionales y muy estéticas, la imagen de abajo muestra el paso.





PROGRAMACIÓN ORIENTADA A OBJETOS.

Clases y Objetos.

Al principio del capítulo hemos mencionado brevemente los paradigmas relacionados de la POO, aunque VFP 9, conserva las clásicas técnicas de programación estructural, con la POO extendemos la capacidad de desarrollar nuestras aplicaciones con la ventaja de poder reutilizar el código, esto convierte a VFP 9 en un lenguaje del tipo RAD. (Rapid Application Development). VFP 9 se centra en la POO ya que nuestras entidades las van a conformar entidades llamadas *objetos*.

Un objeto es una entidad que tiene características y comportamientos, al igual que un teléfono, tiene características que comparten entre sus distintos modelos; los números, el marcado, etc. A este concepto de teléfono lo llamaremos clase. Una clase es una abstracción de un objeto, es algo intangible, es el conjunto de ideas, no es algo real. A la acción de crear un objeto partiendo de una clase se le llama instanciamiento.

Aclaremos, las clases y los objetos están muy relacionados, pero no son lo mismo.

Una clase contiene la información sobre la apariencia y el comportamiento de un objeto. Un objeto es una instancia a una clase. Es decir que el usuario solo podrá actuar sobre los objetos.

Los Objetos tienen propiedades, eventos y métodos.

Cada objeto en VFP 9, tiene propiedades, el color, el ancho, si es visible, etc, y los eventos son las distintas acciones que realiza el usuario o el mismo sistema, sobre los objetos. Los métodos son procedimientos asociados a un objeto. Por ejemplo si escribimos código en el método clic en un command, podemos escribirle el clásico mensaje:

```
messagebox="Hola Mundo"
```

A modo de ejemplo, mostremos tres objetos muy distintos entre si en VFP 9, que tienen las mismas propiedad, los mismos eventos e incluiremos dentro del evento Clic en mismo código en los métodos asociados a dicho evento.

Los Objetos Serán:

Label1

Text1

Command


VISUAL FOXPRO 9.0 SP2 – Capitulo 5


Ingresamos a VFP 9 y en el menú File elegimos New , luego hacemos clic en la opción **Form** y **New File**.


Usaremos el **Form Controls Toolbar**, para hace el siguiente ejemplo, para diseñar nuestros ejemplo del comportamiento de los objetos. En capítulos siguiente detallaremos en forma detallada como trabajar con el entorno IDE de VFP 9.

Para comodidad del Desarrollador y el Programador, podemos acoplar las ventanas Command Window y Properties Window, es se logra encimando o acoplando una arriba de la otra a fin de tener a ambas dentro de una misma ventana con dos solapas.



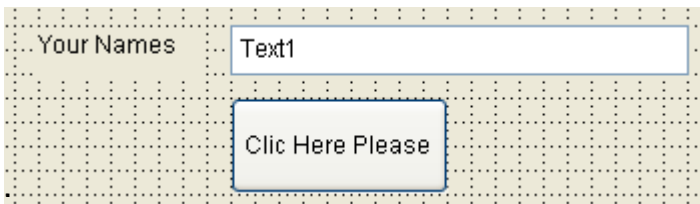
Hacemos clic en el objeto  label y lo insertamos en el formulario. En el cuadro de **Opciones** en la propiedad Caption (del label) escribimos **Your Names**.

Luego hacemos Clic en el objeto Text  y lo insertamos el formulario.

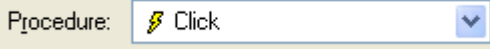
Por ultimo hacemos clic en el objeto command  y tambien lo insertamos en Formulario.

En la propiedad Caption (del command) escribimos **Clic Here Please**.

Si hemos hecho todo bien, Nuestro Formulario sera similar a la imagen que esta abajo.



Dentro de cada objeto, escribiremos 2 codigo iguales, hacemos doble clic en el “Label” Your Names y en el **procedure Click** escribimos:



```
THIS.BackColor = RGB(128,0,255)
```

VISUAL FOXPRO 9.0 SP2 – Capitulo 5

Luego hacemos doble clic en el “**Label**” Your Names y en el procedure **Init** escribimos:

```
THIS.BackColor = RGB(255,255,128)
```

Ahora nos queda hacer doble clic en el objeto “**Text1**”, luego clic en Procedure Click del **Text1** escribimos el siguiente codigo:

```
THIS.BackColor = RGB(128,0,255)
```

Luego de esto hacemos clic en el Procedure Init y escribimos el siguiente código:


```
THIS.BackColor = RGB(255,255,128)
```

y ahora para ir finalizando solo nos queda hace doble clic sobre el objeto “**Command**” y hacemos clic en el Procedure Click del Commnand y escribimos el siguiente codigo:

```
THIS.BackColor = RGB(128,0,255)
```

y para terminar hacemos clic en el Procedure Click del Command y escribimos:

```
THIS.BackColor = RGB(255,255,128)
```

Ahora nos queda hacer clic en la barra estándar sobre  y se nos pedirá un nombre para salvar al formulario en el diseñador de Formularios, hacemos clic en “**Yes**” y lo nombraremos “Example 1” y si hemos hecho todo bien tendremos una pantalla similar a la imagen de abajo. Al iniciar, el Formulario, los colores de fondo estarán en Amarillo, pero si hacemos clic en cualquiera de los tres objetos, automáticamente asumirá el color de fondo Azul! Esto se debe que al hacer clic sobre cualquiera de los objetos se ejecuta el método asociado al código de programa. El evento que ha hecho el usuario a sido simplemente clic sobre el objeto, o también podría ser doble clic por ejemplo.

En los objetos tenemos muchos métodos, muchos de ellos son iguales entre si, por ejemplo el clic o doble clic, pero en otros tenemos métodos propios a cada objeto en cuestión. Para que quede claro, el Objeto **Label1**, por ejemplo no dispone del procedure “**LostFocus**” pero en el

Command y Text1 si disponemos del procedimiento “**LostFocus**”, este procedimiento es muy útil, ya que podemos asociar código de programación (métodos), al perder el foco el objeto sobre el cual estamos situados, de esta forma podremos evitar que el usuario deje un dato útil sin completar en nuestras aplicaciones, tales como la dirección o el número de DNI, y de esta forma se le avisará que ha dejado un campo importante sin completar los datos.

Ventajas de la POO.

HERENCIA

La herencia se produce al crear subclases u objetos, estos tienen las propiedades que hayamos definido en la clase base, por lo que no será necesario repetirlos. Esto representa una ventaja a la hora de programar ya que no será necesario repetir código, esto es la reutilización de código, en las primeras versiones de FoxPro para Windows teníamos que usar el clásico copiar y pegar para los procedimientos que eran útiles de un procedimiento a otro.

En la POO escribimos el código que nos es útil en una clase, y todas las subclases y objetos que creamos partiendo de esa clase, tendrán el código heredado sin necesidad de escribirlo, solo bastará tocar nuestra clase base y nuestro objeto en cuestión y el código se ejecutará automáticamente. En la POO la herencia funciona en forma automática, solamente debemos modificar una propiedad de un objeto y este cambio se reflejará en todos los objetos heredados que hayamos creado a partir de dicha clase.

POLIMORFISMO

Significa que varios objetos, aunque partan de distintas clases, tienen métodos o propiedades con el mismo nombre y actúan de forma distinta. Esta característica tiene relación directa con la herencia ya que un método o propiedad heredado se puede sobrescribir, de modo que puede actuar de distinta de lo que lo hacía en la clase base.

ENCAPSULAMIENTO

Cuando creamos una clase lo que estamos haciendo es agrupar objetos, propiedades o métodos de los mismos, para cuando necesitemos crear otra aplicación, solamente usemos esta clase y la utilicemos sin cambiar nada, a esto se lo define como encapsulación.

En la POO todo podemos encapsularlo, creamos una clase visual por ejemplo y agregamos: Label, Text, Command, Form, Options, etc, con la encapsulación podemos reutilizar código escrito en los objetos y hacer más rápida y robusta nuestra aplicación.

En los capítulos cuando desarrollemos aplicaciones del tipo comercial, mostraremos ejemplos de todo esto, es muy fácil y a modo de ejemplo podríamos decir, que al crear una clase visual incluiremos un command con la leyenda de borrar y el mismo puede Mostrar una ventana o mensaje en color rojo informándonos que estamos a punto de borrar un registro, este mismo objeto con codigo asociado al mismo u método para decir lo mismo, se puede utilizar en todos los forms que usen esta opción de dar de baja un registro de clientes, proveedores, etc.

VISUAL FOXPRO 9.0 SP2 – Capitulo 5

CLASES VISUALES Y NO VISUALES

Las clases visuales son: todas aquellas que podemos verlas en pantalla, estas son las clases visuales, por ejemplo aquellas que tienen objetos tales como, text, command, option, label, etc.

Las clases no visuales: son una colección de propiedades y métodos que son llamados desde otros objetos pero que no forman parte de la interfaz, estas son las clases no visuales.

CLASES VISUALES DE VFP 9

VFP 9 nos entrega una serie de clases, pero nosotros podemos crear y modificar a nuestro gusto una serie de clases con métodos asociados a nuestras necesidades. Todos los objetos que forman la clase visual, fueron hechos de acuerdo al entorno Windows, pero también disponemos de clases que manipulan exclusivamente datos, que es otra extensibilidad que dispone VFP 9.

Clases Visuales de VFP 9

CheckBox	Column	ComboBox
CommandButton	CommandGroup	Container
Control	EditBox	Form
FormSet	Gris	Header
Image	Label	Line
ListBox	OleBoundControl	OLEControl
OptionButton	OptionGroup	Page
PageFrame	Shape	Spinner
TextBox	Toolbar	

CLASES NO VISUALES DE VFP 9

Con la clase **Custom** es para que nosotros creamos clases mas personalizadas. Es la clase no visual por excelencia y mediante ella podemos programar orientado a objetos de manera más pura, aunque en la práctica será un lugar donde se lo guarde como un cajón de herramientas donde tengamos código útil para nuestras aplicaciones. En las aplicaciones de ejemplo daremos ejemplos de las clases Visuales y No Visuales.

CLASES NO VISUALES DE VFP 9		
Cursor	-	Custom
Environment	-	Relation
Timer		

VISUAL FOXPRO 9.0 SP2 – Capitulo 5

CLASES CONTENEDORAS

Las clases contenedoras son aquellas que pueden contener dentro de si a otras clases, todos los objetos contenidos en ella se denominan objetos miembros y automáticamente pasan a depender de su contenedor, esto a pesar de ser un poco complejo al principio, resulta muy útil porque de esta forma seguimos encapsulando mas código, que agrupamos objetos.

CLASES CONTENEDORAS.

Column	CommandGroup
Container	Control
DataEnviroment	Form
FormSet	Grid
OptionGroup	Page
PageFrame	ToolBar

OBJETOS DEPENDIENTES DE UN FORM.

Son todos aquellos que obligatoriamente son incluidos dentro de un Form.

Controles en VFP










Cuando iniciamos Visual FoxPro, y necesitemos crear un formulario, con o sin una BD, hacemos clic en **File , New** y elegimos dentro del cuadro de opciones, **Form**, y por ultimo Clic en **New File**.

Aquí es donde tendremos nuestro **Form1** para empezar a diseñar nuestro formulario con los controles que nos ofrece VFP 9, estos son los objetos que disponemos para trabajar con nuestros Formularios, estos controles nos facilitan la POO.

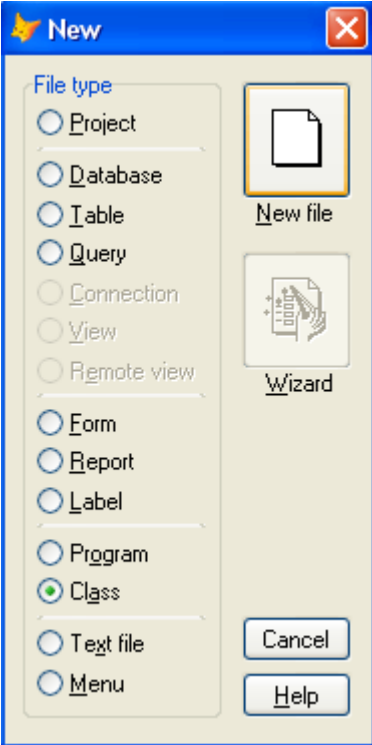
CONTROLES EN VFP 9

CheckBox	Column
ComboBox	CommandButton
CommandGroup	Container (crea un objeto que puede Contener a otros objetos.)
Control (crea un objeto control que puede contener a otros objetos.)	EditBox
Grid	Header (crea un encabezado para una columna de un control grid)
Image	Label
Line	ListBox
OLEBoundControl	OLEControl
OptionButton (cada una de los options	OptionGroup

que contiene el  OptionGroup)	
Page (cada una de las paginas que contiene el  PageFrame)	 PageFrame
 Shape	 Spinner
 TextBox	 Timer

EJEMPLO DE CÓMO CREAR UNA CLASE MEDIANTE EL GENERADOR DE CLASES DE VFP 9.

Recordemos brevemente que dentro de una biblioteca de clases tendremos a varios objetos que estaran contenidos dentro de esa clase, es decir la clase ejemplo que llamaremos “miclass” contendra varios objetos que estarán compuesto por un Label, dos Command, uno para salir y otro para ejecutar la suma y un Objeto TextBox, y un Objeto OptionGroup. Antes de empezar debemos crear una carpeta en nuestro disco duro que se llame DVFP9 y dentro de ella otra carpeta que se llame CLASS. Una vez hecho esto, ya estamos en condiciones de empezar.



Iniciemos VFP 9 y hacemos clic en **File** , después clic en **New (ctrl+N)** y tendremos el siguiente cuadro de opciones, hacemos clic en la Opción **Class** y después clic en **New file**.

A partir de esto momento, empezamos a hacer practica la POO, algo de que se habla mucho y se hace poco, pero no es nuestro caso. Ahora veremos como funciona la herencia, la encapsulación y el polimorfismo.


Esto de programar en POO, tiene solamente un secreto, el de saber pensar detenidamente, de acuerdo a la aplicación que desarrollemos podremos tener una biblioteca de clase con unos determinados objetos y otros no. Con esto queremos decir que podemos usar varias bibliotecas de clases, lo importante es no llenarnos de ellas ya que si son muchas, tenderán a


confundirnos y que desperdiciemos mucho tiempo en analizarlas, ya que un error mínimo se disparara por medio de la herencia, hacia todos objetos heredados.

VISUAL FOXPRO 9.0 SP2 – Capitulo 5

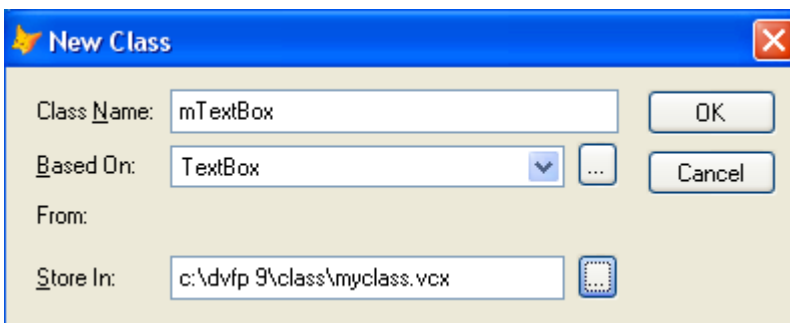
Después de hacer clic en **New file**, tendremos el siguiente cuadro de opciones, ya que vamos a construir un pequeño programa en el que se sumara dos números y tendremos la opción de cambiar el color de fondo del Formulario.

En Class Name: escribimos, mTextBox

En Based On: hacemos clic en  y buscamos donde dice TextBox, hacemos clic sobre el.

En Sore In hacemos clic en  y elegimos la carpeta que esta en la unidad C , que es la

que creamos anteriormente, “DVFP 9” y dentro de ella a CLASS, le asignamos el nombre de **myclass** y hacemos clic en save. luego clic en OK. Si hemos hecho todo correctamente tendremos la siguiente pantalla como lo muestra la figura de abajo.

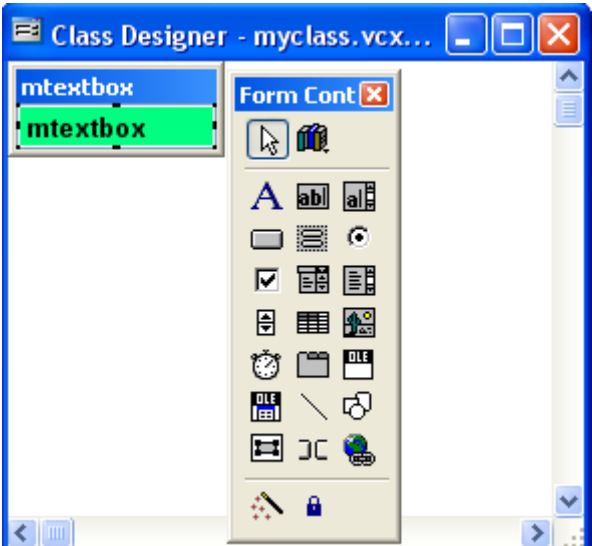



Una vez que hemos hecho clic en  tendremos en pantalla la siguiente figura del

objeto mTextBox y a la izquierda el cuadro de propiedades para editarlas de acuerdo a nuestras necesidades, la propiedad que vamos a cambiar será la de BackColor, FontBold y FontSize.

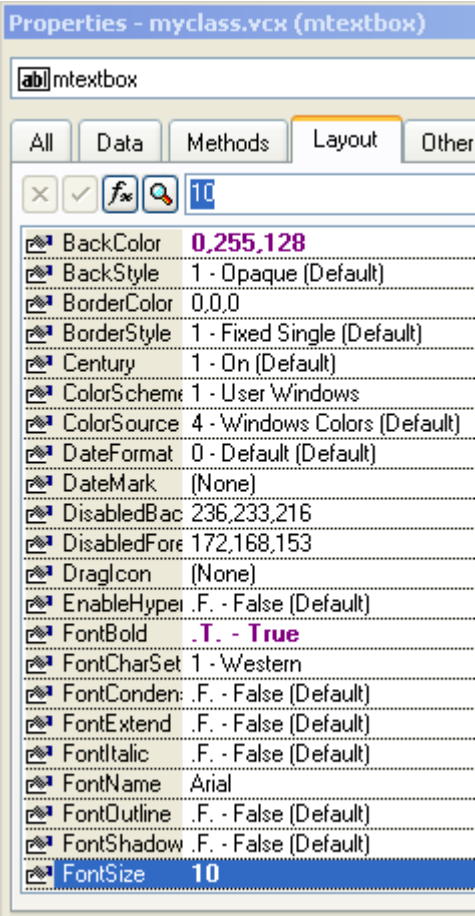
En la solapa Layout buscamos la propiedad Backcolor y hacemos doble clic sobre ella, se desplegara la ventana de cuadro de colores, elegimos el color verde (internamente VFP 9 a nivel código asigna el valor de RGB (0,255,128). Un poco mas abajo en la propiedad FontBold, haciendo doble clic la establecemos a **.T. – TRUE**, esto es nada mas ni nada menos que **negrita**, y por ultimo en la Propiedad FontSize, cambiamos por el Valor de 10.

VISUAL FOXPRO 9.0 SP2 – Capitulo 5




Si hemos hecho todo bien las imágenes que tenemos en esta pagina, deberan ser exactamente iguales a las que debemos tener en nuestras pantallas. Luego de esto para salvar o grabar los cambios solo debemos hacer clic en la imagen del diskette 


Ahora solo nos queda agregar nuevos objetos a nuestra biblioteca de clases llamada myclass. Para continuar procederemos de la siguiente manera.

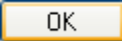


Hacemos clic en el menú de VFP 9 en File y elegimos **New**, elegimos nuevamente Class y New File.

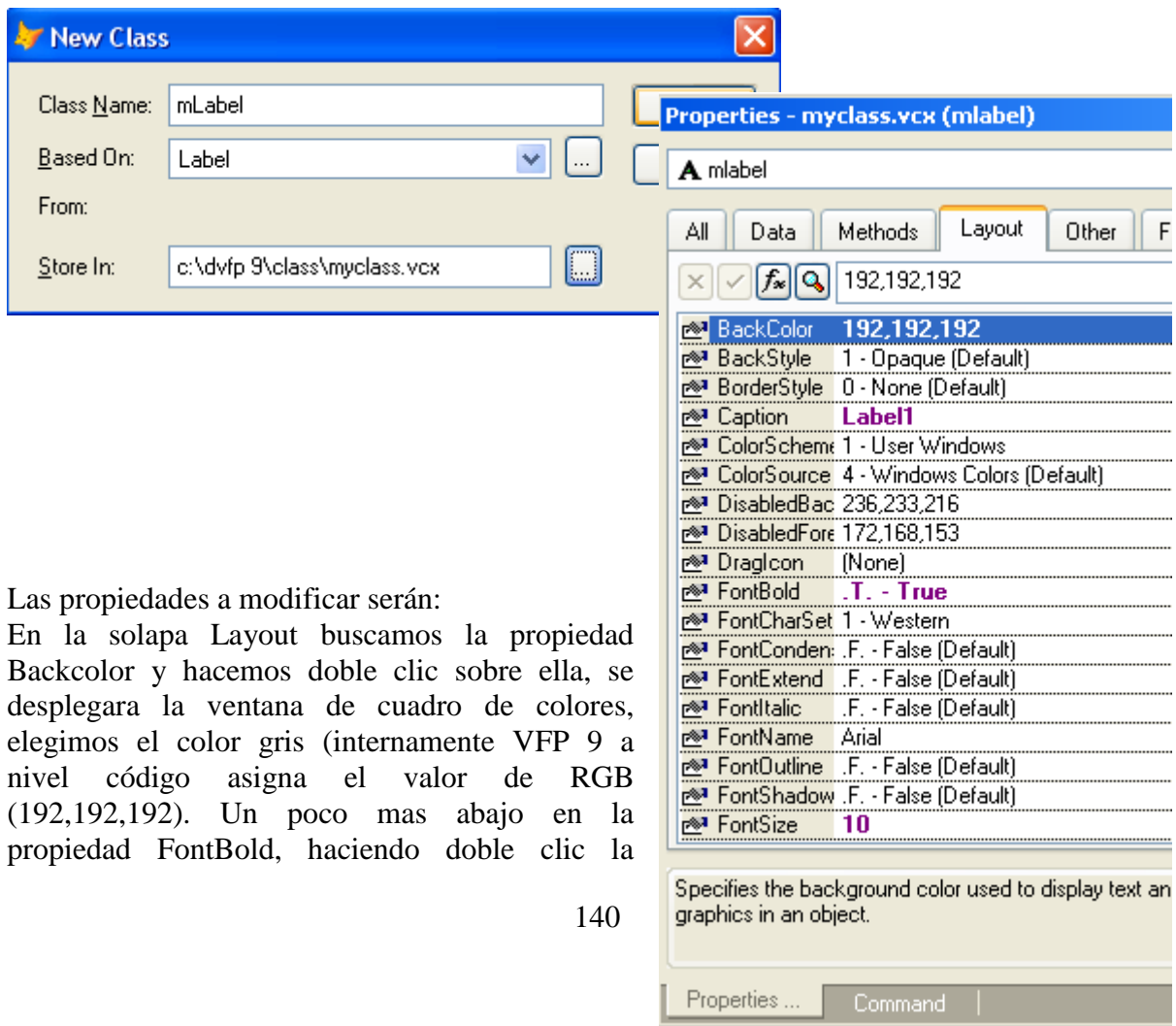
En Class Name: escribimos, mLabel

En Based On: hacemos clic en  y buscamos donde dice Label, hacemos clic sobre el.

En Sore In hacemos clic en  y elegimos la carpeta que esta en la unidad C , que es la

que creamos anteriormente, “DVFP 9” y dentro de ella a CLASS, allí estará precisamente la biblioteca de clase **myclass**, que hemos creado anteriormente, hacemos doble clic sobre ella y luego clic en  . Si esta todo bien, la imagen debe ser igual a figura de abajo.

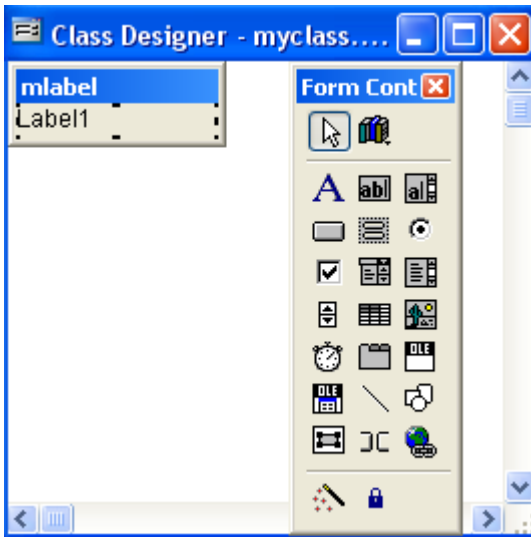
VISUAL FOXPRO 9.0 SP2 – Capitulo 5



Las propiedades a modificar serán:


En la solapa Layout buscamos la propiedad Backcolor y hacemos doble clic sobre ella, se desplegara la ventana de cuadro de colores, elegimos el color gris (internamente VFP 9 a nivel código asigna el valor de RGB (192,192,192). Un poco mas abajo en la propiedad FontBold, haciendo doble clic la

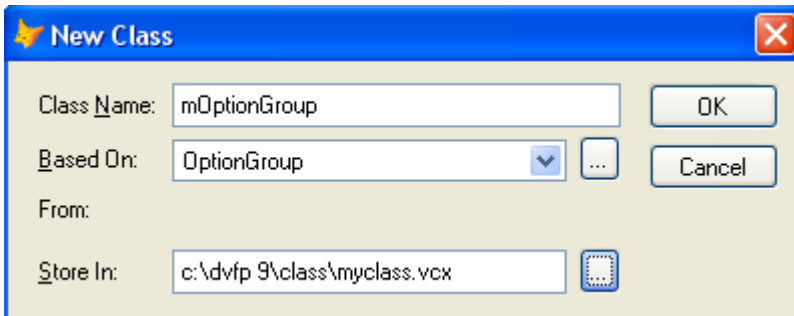
establecemos a **.T. – TRUE**, esto es nada mas ni nada menos que **negrita**, y por ultimo en la Propiedad FontSize, cambiamos por el Valor de 10. las dos imágenes de abajo muestran como debe quedar el objeto mLabel.



VISUAL FOXPRO 9.0 SP2 – Capitulo 5

Ahora solo nos queda, agregar dos objetos mas, el OptionGroup, para que a través de el podamos cambiar el color del formulario, y dos CommandButton, uno para procesar los datos y el otro para salir o cerrar el formulario.

De la misma forma en que hemos procedido para crear los demas objetos dentro de nuestra biblioteca de clases, haremos clic en **File, New** y después seleccionamos Class y hacemos clic en New File, ahora crearemos el OptionGroup. La imagen de abajo muestra como debe quedar nuestro objeto OptionGroup. En la imagen de abajo muestra como debe quedar. **NO DEBEMOS OLVIDARNOS DE HACER CLIC Store in**  para almacenar nuestro objeto en la biblioteca **myclass.vcx**, ya que también es un objeto dependiente de ella.



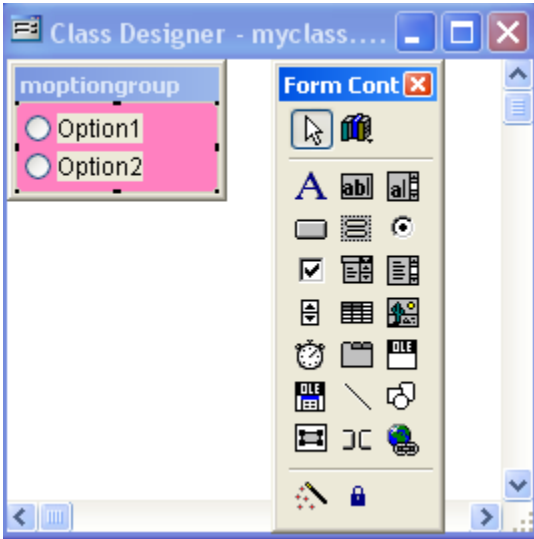
Ahora de la misma forma en que hemos procedido para crear los demás objetos dentro de nuestra biblioteca de clases, haremos clic en **File**, **New** y después seleccionamos **Class** y hacemos clic en la solapa **Layout** y modificamos la propiedad **BackColor** haciendo doble clic sobre ella y elegimos el color rosado.

BackColor = 255,128,192

Luego hacemos clic en la solapa **Data** y elegimos la propiedad **Value** y la establecemos el **valor a 0**, esto es para que ambos **Options** estén desactivados inicialmente, para el usuario pueda escoger entre uno de ellos.

La imagen de abajo muestra brevemente como debe quedar las propiedades de nuestro objeto **mOptionGroup**.

VISUAL FOXPRO 9.0 SP2 – Capitulo 5



Properties - myclass.vcx (moptiongroup)

moptiongroup

All

Data


Methods


Layout

Other


✕

✓







255,128,192

Anchor


0

AutoSize


.F. - False (Default)

BackColor


255,128,192

BackStyle


1 - Opaque (Default)

BorderColor


0,0,0

BorderStyle


1 - Fixed Single (Default)

ButtonCount


2

ColorSource


4 - Windows Colors (Default)

DragIcon


(None)

Height


46

MouseIcon


(None)

MousePointer


0 - (Default)

OLEDragPic


(None)

SpecialEffect


0 - 3D

StatusBarText


(None)

Themes

.T. - True (Default)

ToolTipText

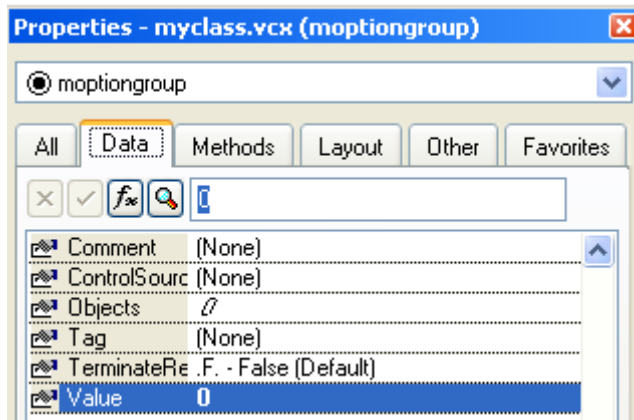
(None)

Visible

.T. - True (Default)

Specifies the background color used to display text graphics in an object.


esta es la ultima solapa (Data) en la que modificamos la propiedad **Value** 1 por el valor **0**.

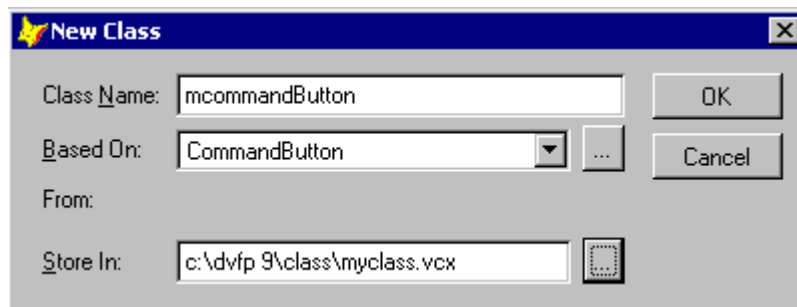


VISUAL FOXPRO 9.0 SP2 – Capitulo 5

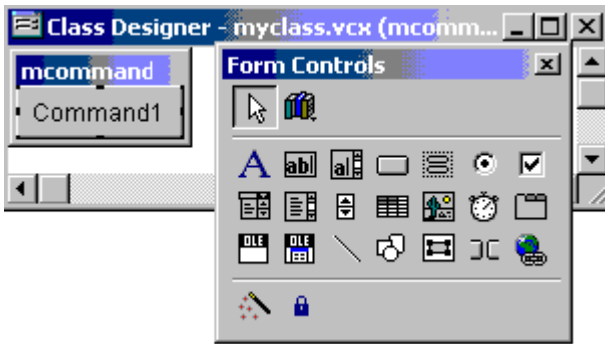
Ahora estamos a punto de finalizar nuestro ejemplo de nuestra biblioteca

myclass a modo de ejemplo.

De la misma forma en que hemos procedido para crear los demás objetos dentro de nuestra biblioteca de clases, haremos clic en **File, New** y después seleccionamos **Class** y hacemos clic en **New File**, ahora crearemos el **CommandButton**. La imagen de abajo muestra como debe quedar nuestro objeto **CommandButton**. En la imagen de abajo muestra como debe quedar. **NO DEBEMOS OLVIDARNOS DE HACER CLIC Store in**  para almacenar nuestro objeto en la biblioteca **myclass.vcx**, ya que también es un objeto dependiente de ella y clic en **OK** para finalizar.




Ahora solo nos falta modificar las propiedades de **Backcolor=0,255,255** y después hacemos clic en el diskette para grabar la modificación y pasamos a crear nuestro ultimo objeto.

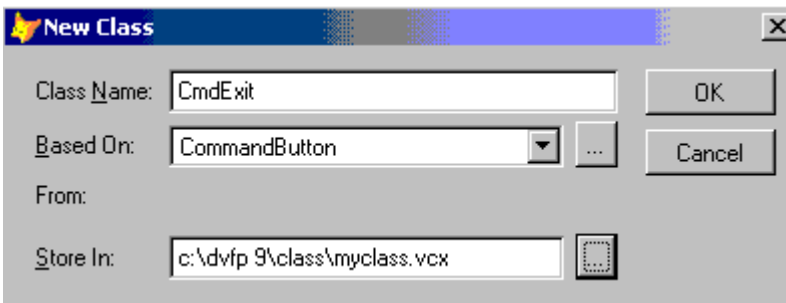


VISUAL FOXPRO 9.0 SP2 – Capítulo 5

Ahora estamos a punto de finalizar nuestro ejemplo de nuestra biblioteca **myclass** a modo de ejemplo.

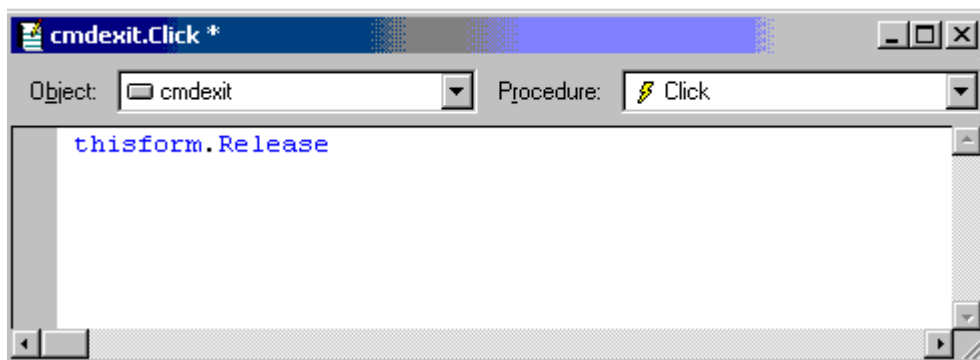
De la misma forma en que hemos procedido para crear los demás objetos dentro de nuestra biblioteca de clases, haremos clic en **File, New** y después seleccionamos **Class** y hacemos clic en **New File**, ahora crearemos el **CommandButton**. La imagen de abajo muestra como debe quedar nuestro objeto **CommandButton**. En la imagen de abajo muestra como debe quedar.

NO DEBEMOS OLVIDARNOS DE HACER CLIC **Store in**  para almacenar nuestro objeto en la biblioteca **myclass.vcx**, ya que también es un objeto dependiente de ella y clic en **OK** para finalizar.



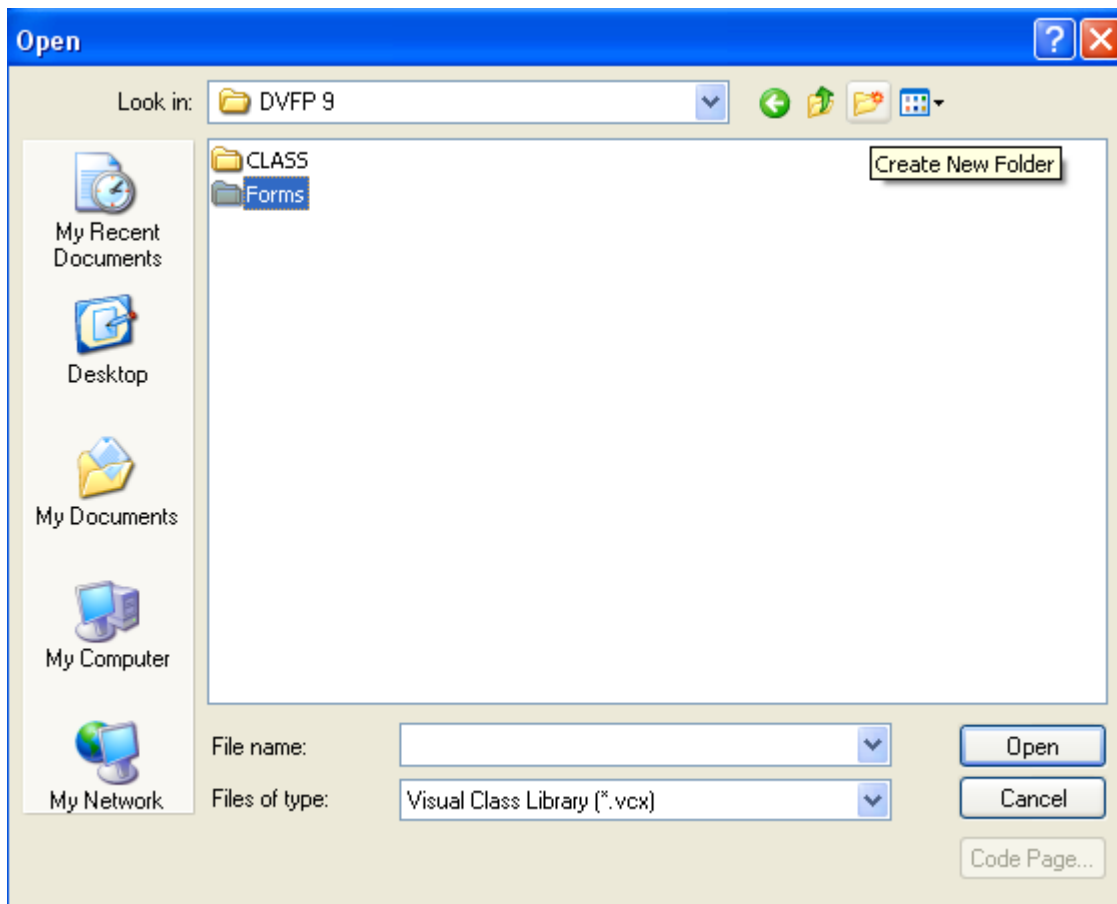
Ahora solo nos queda incluir una linea de codigo, solamente debemos hacer doble clic sobre el objeto **cmdExit** y dentro del procedure **Click** escribimos el código: **thisform.Release**

hacemos clic en el diskette y cerramos el procedimiento.
Ahora estamos listos para empezar a programar con POO.

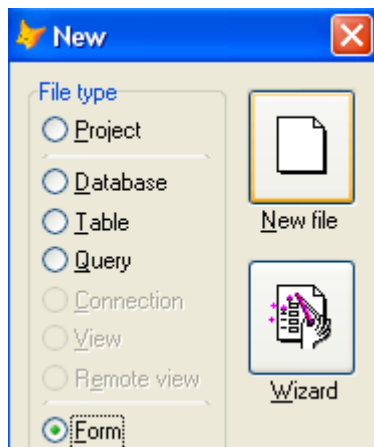


VISUAL FOXPRO 9.0 SP2 – Capítulo 5

Ahora nos falta hacer nuestro programa de ejemplo y terminamos.
Podemos crear con el navegador o explorador de Windows dentro de la carpeta DVFP 9 la subcarpeta Forms, o lo podemos hacer desde VFP 9 desde el icono que muestra una carpeta que es create new folder como lo muestra la figura de abajo.

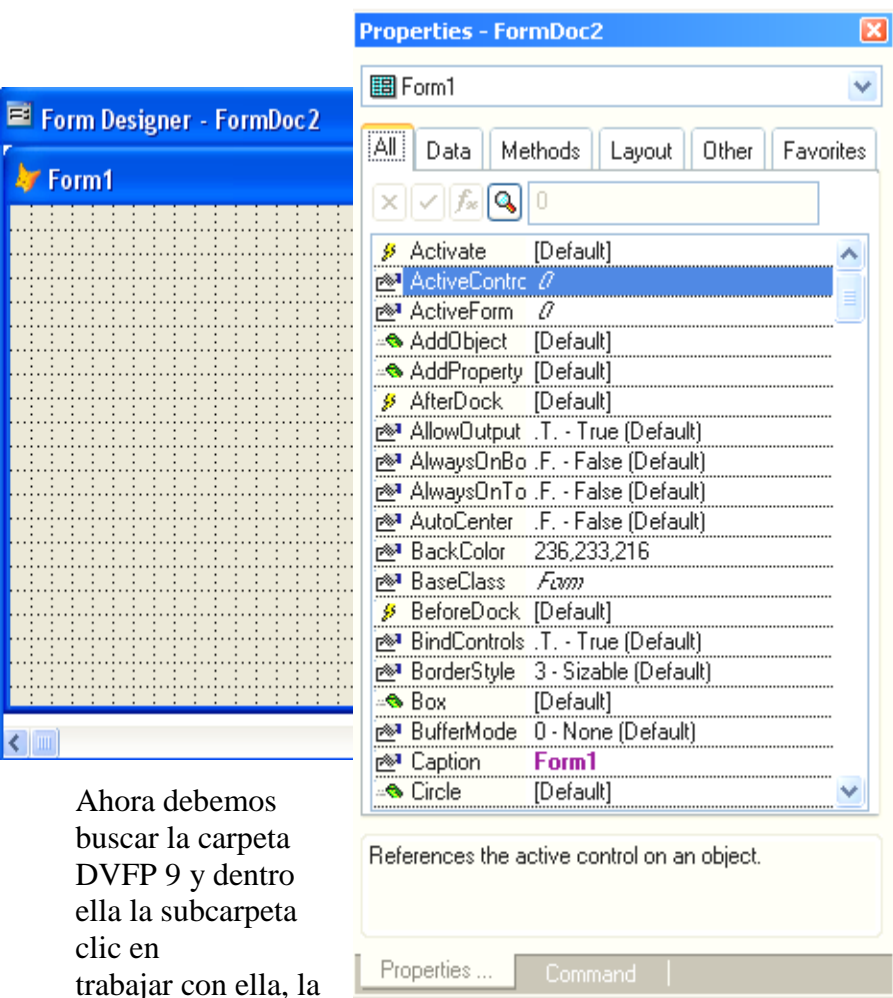


Iniciemos VFP 9 y hacemos clic en **File** , **New** y elegimos **Form** y luego en New File tal como lo muestra la figura de la pagina siguiente, de esta manera estaremos programando nuestro primer formulario.



VISUAL FOXPRO 9.0 SP2 – Capitulo 5

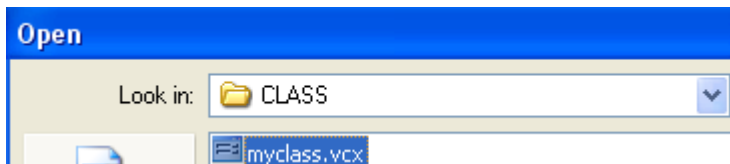
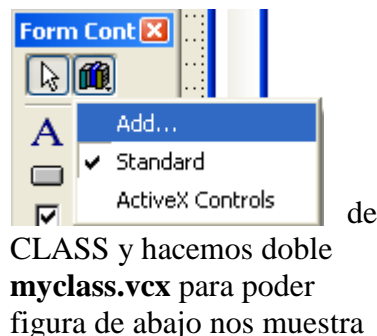
Tal como lo muestra la figura de abajo, tendremos un formulario limpio o vacío listo para insertar los objetos de la clase base que dispone VFP 9 por defecto, o podemos ingresar nuestro propio objetos creados a través de la biblioteca de clases que la hemos llamado **myclass**, de esta forma estamos programando netamente con la POO.



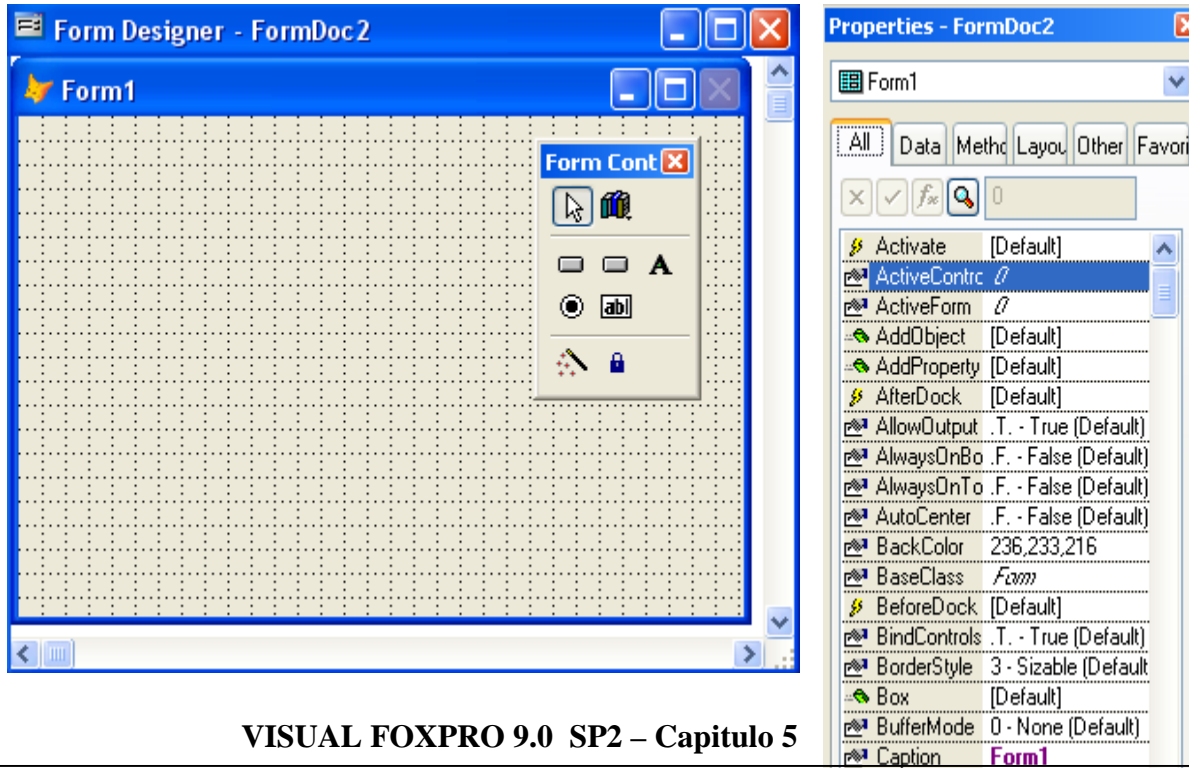
Ahora debemos buscar la carpeta DVFP 9 y dentro ella la subcarpeta clic en trabajar con ella, la donde esta ubicada nuestra biblioteca de clases **myclass.vcx**

VISUAL FOXPRO 9.0 SP2 – Capitulo 5

Ahora debemos hacer clic en la biblioteca de clases para traer nuestra propia biblioteca de clases, solamente debemos hacer clic en add para buscar nuestra propia biblioteca de clases, tal como lo muestra la figura de abajo.



Si hemos hecho todos los pasos correctamente la figura de abajo muestra como queda nuestro formulario y al margen derecho deberá estar el cuadro de propiedades del Form1(formulario)



VISUAL FOXPRO 9.0 SP2 – Capítulo 5

El ejercicio que proponemos, es muy fácil, solamente debemos sumar 2 números. Solamente debemos hacer clic en el objeto mLabel y arrastrar tres de ellos en un margen del form, luego tres mtextbox que deberan ser arrastrados al Form, y por ultimo arrastramos un objeto mcommandbutton para insertarle código para que sume y por ultimo insertamos un objeto cmdexit para salir del Form, aclaremos que podemos usar las convenciones para el nombre de los objetos, lo cual serán mas cortos y fáciles de recordar, pero al momento hemos optados por nombras de esta manera por una cuestión practica. Finalizado el ejercicio deberá quedar de esta manera.

Form1

Numero 1 2500

Numero 2 5000

Total 7500

Sumar Salir

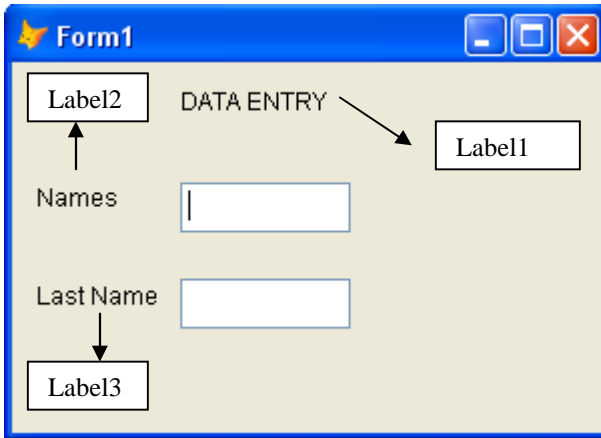
En las figuras de la pagina siguiente resumiremos la forma de cambiar las propiedades de los objetos, en caso de ser necesario, y como agregarle código a nuestros objetos commandbutton para que ejecuten acciones determinadas, también es claro que si hubiésemos utilizado la clase base que nos proporciona VFP 9, deberíamos cambiar tres veces la propiedad `backcolor` de cada objeto, esto significa que si tenemos una aplicación de 50 ventanas, nos demandaría mucho tiempo en tan solo establecer sus propiedades! Por esto y por mucho más vale la pena la POO.

VISUAL FOXPRO 9.0 SP2 – Capitulo 5

A continuación daré una breve descripción, de las propiedad de los controles, que usaremos en el desarrollo de aplicaciones, ya que describir y dar ejemplo completo de cada una de ellas de ellas, con llevaría a escribir otro libro. Manos a la obra!

LABEL

En esta herramienta puedes introducir todos los textos “fijos” que formarán parte de nuestra aplicación, claro esta que a través de código de programación se podrán cambiar los textos que estén los Label.



WordWrap

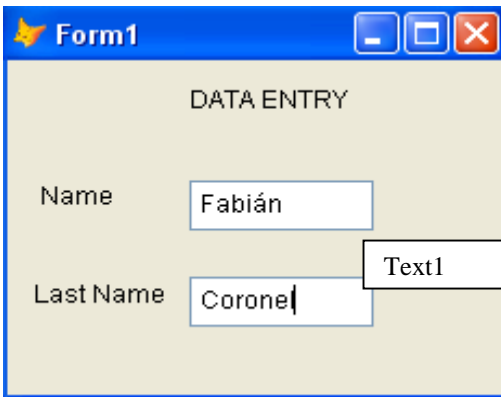
Esta propiedad nos permite incluir un texto largo que se ajustara en forma automática cuando el mismo necesite de mas de una línea, para que esta propiedad funcione debemos establecer a .T. la propiedad **Autosize** .

Otro control que lo posee es el CommandButton

VISUAL FOXPRO 9.0 SP2 – Capitulo 5

TEXTBOX

Es el control más clásico por excelencia, ya que, no solamente sirve para mostrar textos y valores sino que también a través de el, los operadores podrán ingresar los datos que el sistema requiera o modificar el mismo.



Propiedades útiles

Century

Muestra todos los dígitos en el formato de dos o cuatro dígitos, los valores posibles son:

- 0 – Off, se muestra las dos ultimas cifras
- 1 – On – Muestra todas las cifras del año
- 2 – Default – valor por defecto, esta relacionado con SET CENTURY

Es mejor, por practica que, todos los comandos de configuraciones en fechas sean tratados a nivel código de programación, ya que mas practico establecer con el comando SET CENTURY el formato de fecha largo o fecha corta para todo nuestro sistema, estableciendo así de manera homogeneidad para toda nuestra aplicación.

DateFormat

Con esta propiedad podes establecer el formato de fecha a mostrar, es muy útil si programamos aplicaciones en un entorno internacional, ya que no todos los países utilizan el mismo forma de fecha, es el equivalente a nivel código fuente de **SET DATE**. Los valores posibles son:

Value	Description	Format
0	Por defecto, Establecido por el comando Set Date	Establecido por Set Date
1	American	MM/DD/YY

2	ANSI	YY.MM.DD
3	British	DD/MM/YY
4	Italian	DD-MM-YY
5	French	DD/MM/YY
6	German/Russian	DD.MM.YY
7	Japan	YY/MM/DD
8	Taiwan	YY/MM/DD
9	USA	MM-DD-YY
10	MDY	MM/DD/YY
11	DMY	DD/MM/YY
12	YMD	YY/MM/DD
13	Corto	Determinado por la fecha del Panel de Control de Windows
14	Largo	Determinado por la fecha larga del panel de control de Windows

DateMark

Establece el delimitador para la Fecha y valores de DateTime desplegados, en una caja de texto. Disponible en tiempo de diseño y en tiempo de Ejecución.

Format

Configura la entrada y salida de datos.

Values	Description
A	Caracteres alfabéticos (sin espacios ni puntos)
D	formato de fecha actual
E	Edita los datos tipo Fecha como fecha británica
K	Hace un Select On Entry cuando el foco se sitúa en el TextBox
L	Convierte los espacios a la izquierda en ceros si los datos son numéricos

M	Permite múltiples opciones preestablecidas. La lista de opciones, se almacena En la propiedad InputMask como una lista de elementos delimitados por comas.
R	Muestra una máscara de entrada de datos. La máscara no se almacena en origen de datos del control.
I	Convierte a mayúsculas.

Otros controles que lo disponen: Column, Combobox, EditBox, Spinner.

HideSelection

Si está a .T. el texto marcado dentro del TextBox permanecerá seleccionado cuando el foco salga del él.

Otros controles que contienen: EditBox, Spinner.

Hours

Establece el tipo de visualización de un campo DateTime según estos posibles valores:

Value	Description
0	Por defecto. Fija el formato establecido mediante SET HOURS
12	Formato de 12 horas
24	Formato 24 horas

VISUAL FOXPRO 9.0 SP2 – Capítulo 5

InputMask

Establece como se mostrara cada carácter o dígito de entrada, la diferencia con **Format** en que éste se refiere a la totalidad de los datos.

Values	Description
X	Se puede agregar cualquier carácter
9	Se pueden combinar dígitos y signos
#	Pueden introducirse espacios en blanco, dígitos y signos
\$	Muestra el símbolo de moneda actual de acuerdo a la configuración regional

\$\$	Muestra un símbolo de moneda flotante que siempre aparece junto a los dígitos
*	Se muestran asteriscos a la izquierda del valor
.	Establece la posición de coma decimal
,	Separa los dígitos a la izquierda de la coma decimal

Es muy útil, todas estas opciones que tenemos con **INPUTMASK**, ya que nos permitirá tener el control total en la forma de introducción de datos. Por ejemplo cuando tenemos un dato especial el la cual podría utilizarse el **INPUTMASK** podría ser un número de pasaporte.

`PasTextBox.InputMask="99.999.999-X"`

También esta disponible esta opción en los controles : Column, ComboBox, Spinner

VISUAL FOXPRO 9.0 SP2 – Capítulo 5

PasswordChar

Permite crear un **TextBox** para introducción de un password,. La contraseña no será vista por el Operador, sino que aparecerán los clásicos asteriscos, por ejemplo, o cualquier otro carácter indicado en esta propiedad.

ReadOnly

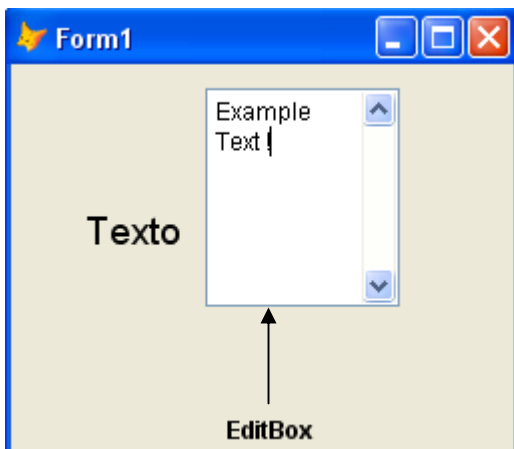
Establece como de sólo lectura el **TextBox**. Esta propiedad nos imposibilita escribir, pero no el situar el foco dentro del **TextBox**, en algunos casos habrá que devolver .F. en el método **When**, a fin de evitar confusión en el usuario.

FontChartSet

Esta propiedad es muy útil, para todos los desarrolladores que trabajan con aplicaciones internacionales, ya que nos permite establecer el idioma con un solo click ! VFP 9 no tiene límites en lenguajes, esto lo convierte en uno de los favoritos por todos los programadores. Otros controles soportados, Label, EditBox, entre otros que soportan carta de datos.

EDITBOX

Es muy utilizado para introducir datos en los campos del tipo memo, debido a que nos permite realizar un Scholl a través del texto.



VISUAL FOXPRO 9.0 SP2 – Capítulo 5

PROPIEDADES PARTICULARES

AllowTabs

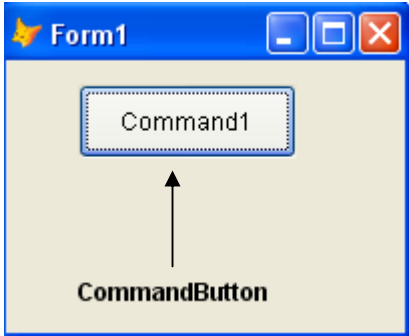
Si está a .T. podemos usar el tabulador dentro del **EditBox**, ya que al tratarse de la tecla mediante la cual se avanza de control en control, nos iríamos al siguiente.

ScrollBars

Es para que muestre las barras de desplazamiento, lo normal es dejarlo a .T. ya que, si estuviese a .F. , es como si estuviésemos utilizando un TextBox.

COMMANDBUTTON

Es uno de los objetos mas utilizado por programadores ya que ejecutada código al ser presionado, uno de sus métodos más usado es el Click.



PROPIEDADES PARTICULARES

DisablePicture

Podemos cambiar el dibujo del botón cuando esté desactivado.

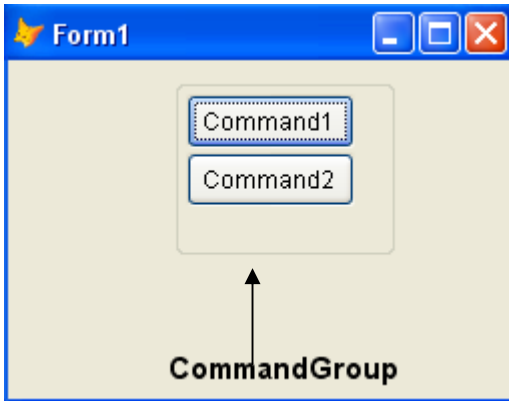
VISUAL FOXPRO 9.0 SP2 – Capitulo 5

DownPicture

Igual al anterior, pero para cuando presionamos el botón. Podemos utilizar estas dos opciones descriptas para lograr efectos sorprendentes a la vista del usuario.

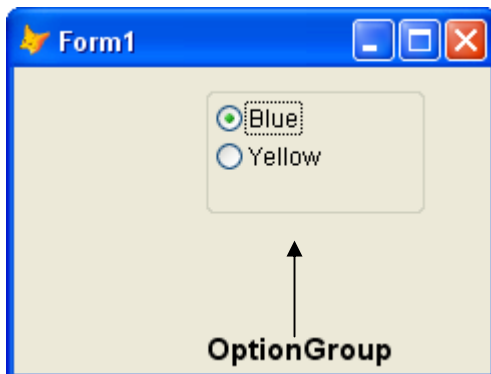
COMMADGROUP

Esto sería algo como agrupar unos botones de Comandos, esta en el gusto del programador usarlo o no dependiendo la necesidades de la Aplicaciones.



OPTIONGROUP

Esto es un conjunto de OptionButton, Para establecer la opción seleccionada por defecto debemos establecer la propiedad **Value** con la opción requerida, claro esta que si tenemos mas de 2 opciones, lo ideal sería establecer el **Value con 0**, por seguridad para evitar que el mismo sistema cometa un error ajeno al usuario, y sea el quien elija la opción correcta, con El **Value 0**, todas las opciones, estarán desactivas al principio.



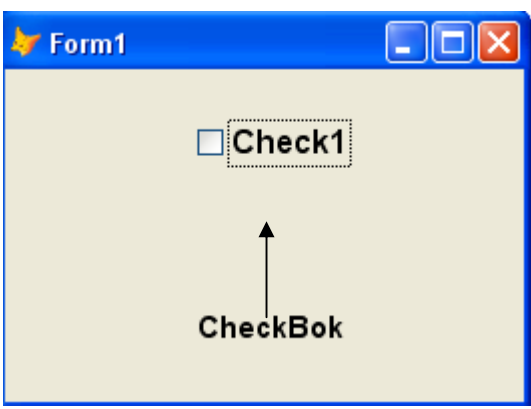
Tipos de datos soportados: Numérico, Carácter, Entero.

CHECKBOX

Casilla de verificación, la propiedad **Value**, sirve para establecer su estado inicial, debemos tener especial cuidado, ya que si tenemos dos o tres **CheckBox**, todos ellos podrán ser seleccionados, a diferencia del OptionGroup, que puede ser seleccionado uno solamente, sus valores puede ser:

Value	Description
0	No seleccionado
1	Seleccionado
2	Seleccionado y en Gris

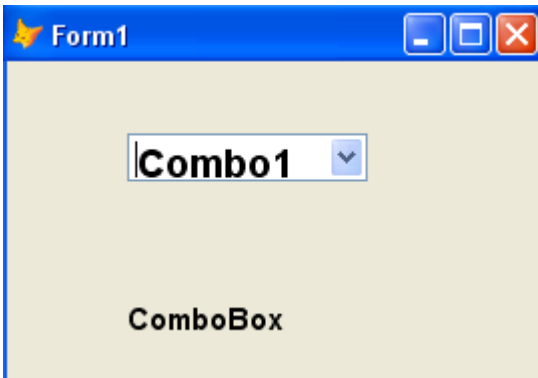
Tipo de datos de datos soportados: Logico, Entero, Numerico.



COMBOBOX

Permite seleccionar un elemento de una lista de ellos.

En breve detallaremos las propiedades del mismo, ya que deberán ser utilizadas en forma correcta a fin de evitar resultados inesperados.



Tipo de datos soportados: Carácter, Entero, Numérico.

VISUAL FOXPRO 9.0 SP2 – Capítulo 5

Propiedades Particulares

Establece cuál va a ser columna ligada con el origen de datos en el caso de haya varias, en un ComboBox podemos mostrar información de una columna, de todas maneras, se debe coger el dato el dato de la misma fila pero de otra columna.

ColumnCount

Especifica el número de objetos de la Columna en un de la columna en u **Grid**, **Combobox** o un **ListBox**. Para un **Grid** disponible en tiempo de lectura/escritura en tiempo de ejecución. Para cada **ComboBox** o **ListBox**, disponible en tiempo de diseño y en tiempo de ejecución.

ControlSource

Especifica el origen de datos al cual un objeto es ligado. Disponible en tiempo de diseño y en tiempo de ejecución.

MoverBars

Especifica si se despliegan las barras sobre un control **ListBox**, disponible en tiempo de diseño; lectura/escritura y tiempo de ejecución.

MultiSelect

Especifica si un usuario puede hacer selecciones múltiples en un control de **ListBox** y cómo las selecciones múltiples pueden hacerse. Disponible en tiempo de diseño; Lectura/escritura y tiempo de ejecución.

VISUAL FOXPRO 9.0 SP2 – Capitulo 5

RowSource

Especifica el origen de los valores en un **ComboBox** o **ListBox** control. Lectura/Escritura en tiempo de diseño y en tiempo de ejecución.

RowSourceType

Especifica el tipo de origen de los datos que se muestran en el ComboBox , es del tipo de Lectura/Escritura en tiempo de diseño y tiempo de ejecución.

Value	Description	Content RowSource
0	Ninguno. Podemos llenar la lista mediante los Métodos AddItem o AddListItem , a modo ejemplo	Ninguno
1	Valor	Lista de datos delimitada por comas
2	Alias. Muestra tantos campos como columnas se hayan Establecido en ColumnCount	Alias de la tabla
3	Resultado de un comando o instrucción SQL	Instrucción SQL
4	Consulta (QPR)	Nombre de la consulta
5	Matriz	Nombre de la matriz
6	Campos	Lista de campos delimitada por comas. Para poner el alias de la tabla se debe hacer sólo en el primer elemento.
7	Archivos	Directorio y tipo de archivos (*.dbf *.txt)
8	Estructura de campos de una tabla	Nombre de la tabla
9	Emergente	Nombre del Popup

Otros controles que lo tienen: ListBox.

VISUAL FOXPRO 9.0 SP2 – Capítulo 5

Ejemplo practico para la creación de un ComboBox.

Crearemos 2 tablas, en el cual ingresaremos el sexo, de una persona, a modo de ejemplo: clientes.dbf

Orden Carácter 3
 Nombres Carácter 40 Índice
 Sexo Carácter 5
 Dirección Carácter 40

Ciudad Carácter 20

Provincia Carácter 20

sexo.dbf

codigo Carácter 1

Sexo Carácter 5

Las propiedades a usar son las siguientes:

BoundColumn= 1

ColumnWidths= 65,65

ControlSource= "clientes.Nombres" ** tabla principal

RowSource="sexo, nombres , código ** campos que queremos mostrar.

RowSourceType= 6 * campos

Style=2

Ahora nos queda vincular el código en el campo código sexo se ha utilizado en el **VALID**, que se ejecutara cuando el usuario hace un selección.

Thisform.sexo.value=sexo.codigo

El desafío ahora es más, que el usuario investigue, los demás objetos y la forma en que se aplican las propiedades! Éxitos.