Curso de DELPHI

INDICE DEL CURSO DE DELPHI

TUpdateSQL

TstoredProc

Componentes Visuales de Acceso a Datos

Capítulo 6. Escribiendo una Aplicación de Base de Datos

Accesando Datos en Delphi

Capítulo 6.1. Cómo funciona Cliente/Servidor

Comandos SQL más comunes

Capítulo 1. La aplicación - El "Archivo de Proyecto de Delphi"

Esta sección esta enfocada a entender como funciona una aplicación en el ambiente Delphi dentro de Windows, y cuales son las principales partes de un programa de "Object Pascal" en Windows (object pascal es el nombre formal del lenguaje de programación - así como en Visual Basic el lenguaje es Basic, en Delphi el lenguaje es Object Pascal). Si quiere usted comenzar a programar en Delphi de inmediato, vaya a la siguiente sección (y asómbrese de lo fácil que es programar con Delphi!), pero asegúrese de regresar a esta sección para tener una buena fundamentación de la manera en que funciona una aplicación de Delphi.

> NOTA: No se preocupe si no entiende la totalidad de lo mencionado en esta sección. Muchos programadores muy exitosos no comprenden mucho de lo especificado en este capítulo. Pero creo que en estos días de lenguajes tan diferentes para programación en Windows (Visual Basic, PowerBuilder, C++) es muy importante tener una idea clara de lo que el lenguaje que utilizamos tiene que hacer para lograr su "magia". Los programadores de C++ para Windows reconocerán muchos contrastes con este lenguaje, y encontrarán que Delphi es casi tan flexible en este campo como C++.

Delphi se divide en tres secciones, el compilador (con su "encadenador"), la librería, y el IDE (Ambiente de desarrollo integrado, o *Integrated Development Environment*). El compilador/encadenador es un programa que crea el archivo ejecutable de Windows estilo Intel, sin ningún interprete de por medio. La librería es código que nos permite usar todas las capacidades de Delphi. La librería esta escrita en su totalidad en Object Pascal (es una librería "de clases" estilo MFC, llamada VCL), y esta totalmente orientada a objetos. Veremos objetos y su repercusión en programación mas adelante.

Una nota interesante es que el IDE esta hecho en Delphi, y utiliza las mismas librerías que usted utiliza para compilar su programa. Esto quiere decir dos cosas:

- Primero, que todo lo que puede hacer el IDE es posible para usted (lo cual es notorio contraste con lenguajes como Visual Basic).

- Segundo, que el IDE esta abierto, lo cual quiere decir que usted puede no solo extender la librería para que Delphi utilice los "componentes" diseñados por usted, sino que además puede extender el IDE para hacer "expertos" y ayudas de programación.

Un programa es una serie de instrucciones que son ejecutadas por la computadora en secuencia. En los viejos tiempos de BASIC, la secuencia estaba especificada por números de línea. En los demás lenguajes la secuencia esta simplemente especificada por las líneas del programa, que van una detrás de otra. En DOS, cuando el programa ejecuta la ultima línea, el programa termina.

Los que han programado en DOS recordaran lo siguiente: El hecho de que el programa termina después de su ultima línea obligaba a los programadores a hacer un ciclo para controlar el programa. Este ciclo (ya sea usando Goto, Do-While u otra clase de recursión) mostraba el menú principal y respondía a lo que el usuario seleccionaba. Un programa en Windows no es muy diferente en este sentido de una aplicación en DOS. El ciclo sigue ahí, pero ahora el ciclo lo controla el lenguaje y no el programador. Lo que anteriormente llamábamos "ciclo del menú principal" (o algo parecido) ahora se llama "message loop" (ciclo de mensajes).

El "programa principal" de Delphi es un archivo de texto ASCII con extensión.DPR. Esta extensión quiere decir Delphi PRoject (proyecto de Delphi).

Para cada una de las ventanas que usted diseña en el IDE, Delphi crea una "unidad". Una unidad es un archivo individual (también de texto ASCII) que representa en general a un objeto, o a una agrupación lógica de funciones. En el caso de los "objetos" que son formas, Delphi también crea un archivo "DFM" (Delphi Form) para guardar la apariencia del diseño de las mismas (las formas son simplemente archivos de recursos en un formato especial que solo funciona en Delphi - ver "archivos de recursos (RES)").

El siguiente ejemplo muestra el archivo DPR que se crea cuando comienza usted Delphi. En general usted puede crear programas muy complejos sin jamas modificar el archivo DPR (también llamado archivo de proyecto). Pero es importante saber como funciona. El archivo de Proyecto "Project1.Dpr" tiene la siguiente apariencia:

```
program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Analicemos este archivo. La sentencia "program project1" nos dice que el programa se llama projec1. El nombre de programa debe ser el mismo que el nombre del archivo (sin extensión).

La sección USES nos dice que archivos son usados. Forms es la librería de Delphi que

contiene los comandos para crear y manipular ventanas (que en Delphi se llaman Formas). El otro renglón en la sección "uses" nos dice que la unidad llamada "Unit1" se encuentra en el archivo "unit1.pas", y Delphi nos hace un comentario {} que dice que la forma en esta unidad se llama Form1.

$R es una "directiva del compilador". Una directiva es un comando para el compilador, no para el IDE. Cuando el compilador encuentra la directiva {$R *.RES} un archivo de "recursos" con el mismo nombre del programa (pero con extensión RES) es compilado junto con el programa.

 Un archivo de recursos (RES) es un archivo con especificaciones para bitmaps, textos en general, iconos, y en algunos casos cajas de dialogo o menúes. Delphi utiliza el mismo formato para sus archivos RES que C, y cualquier editor de recursos para C de 32 bits (como los que se encuentran en Visual C++ 4/5) pueden crear archivos de recursos que Delphi puede leer y utilizar.

Una vez que Delphi ha especificado todo lo que va a usar el proyecto, el programa inicializa la aplicación (Application.Initialize), crea la forma Form1 a partir de la definición de objeto "TForm1" (veremos el significado de esto mas adelante) usando "CreateForm", y la aplicación "corre" (Application.Run). Cualquier forma creada con CreateForm esta "viva" (aunque podría ser invisible) hasta que el Application.Run termina. Después de esto, como todo programa de DOS, el programa termina.

Porque Application.Run?

"Run" en Delphi es un método de la aplicación que hace al programa entrar en el ciclo de mensajes de Windows. El ciclo de mensajes es un ciclo estilo "Do While" que recibe mensajes de Windows (clicks de botones, tecleos, movimientos del "mouse") y procesa los mensajes en una manera consistente para todas las "ventanas" de la aplicación. En los viejos días de la programación de Windows, los programadores de C tenían que escribir el ciclo de mensajes ellos mismos. Pero hoy en día, librerías de clases como MFC y el VCL de Delphi manejan la tediosa programación del ciclo de mensajes.

En el siguiente capítulo veremos como Delphi nos facilita la construcción de una aplicación, y escribiremos una sola línea de código para ver el poder que nos proporciona Delphi para el desarrollo de aplicaciones.

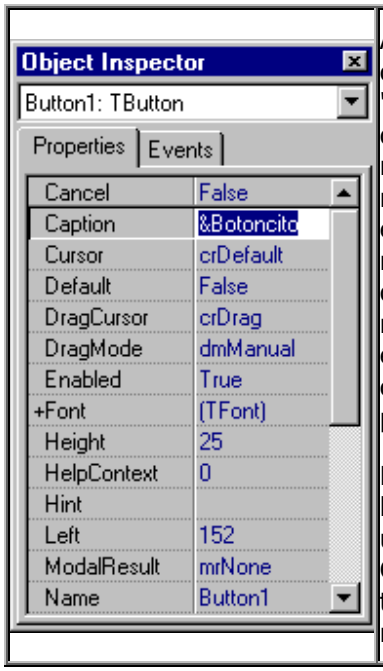
Capitulo 2. Nuestro primer programa de Delphi - Una Forma

Así que estamos listos para escribir nuestro primer programa? Comencemos entonces. Como se mencionó anteriormente, Project1 es un proyecto vacío que simplemente contiene una forma vacía.

El IDE de Delphi se divide en tres secciones:

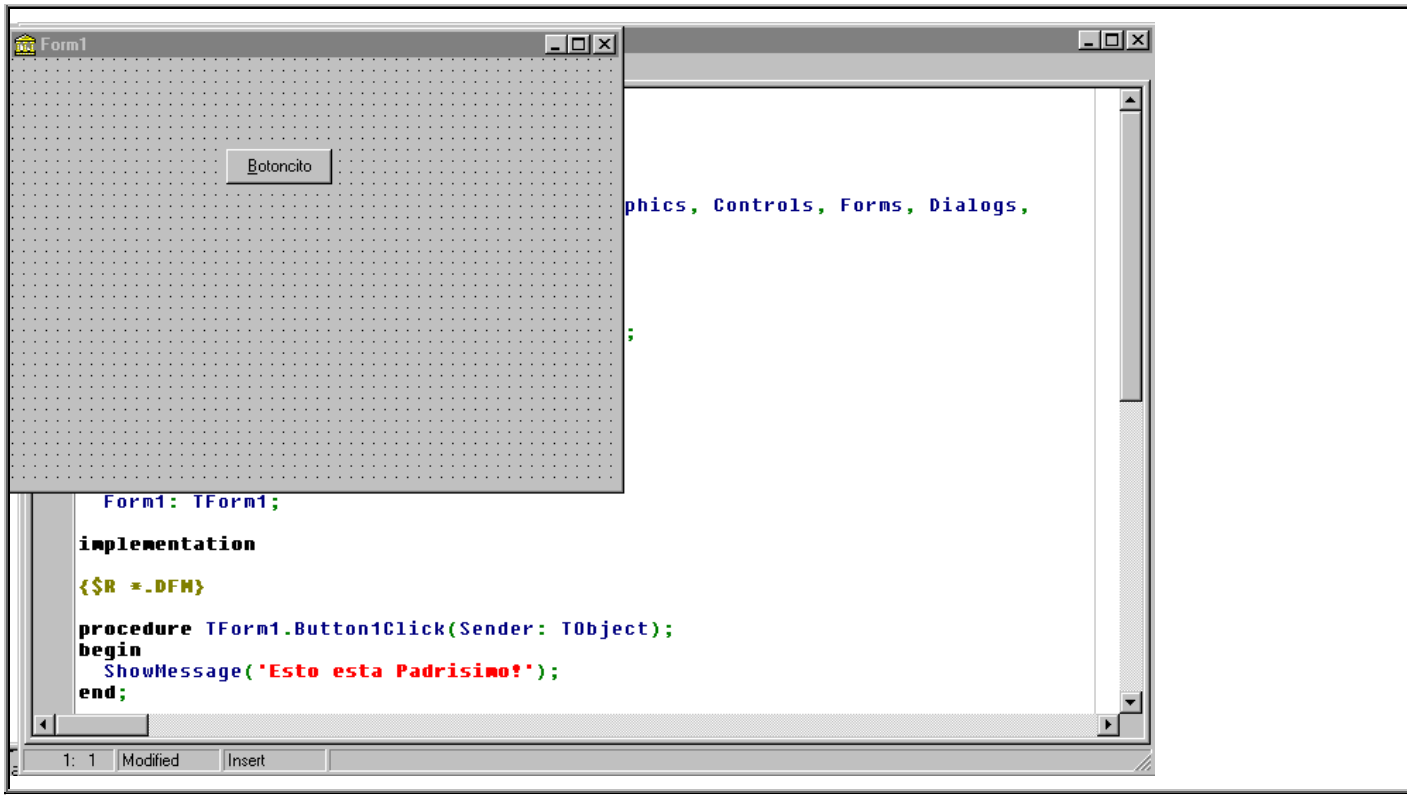
La parte de arriba contiene el menú principal de Delphi, y contiene botones para hacer cosas con el proyecto (abrir, salvar, compilar) en la parte izquierda. La parte derecha contiene la "paleta de componentes" que es un menú de botones con todos los componentes que podemos poner en la forma, en distintas paginas.

NOTA: Nuestra paleta contiene add-ons y varios componentes que pruebo. Estos add-ons pueden hacer que nuestro Delphi no sea exactamente igual al suyo.



A la izquierda, podemos ver el inspector de objetos, que contiene las "propiedades" y "eventos" del objeto seleccionado en el diseñador. Usted puede hacer click con el mouse en cualquier propiedad para modificarla. En las propiedades que vienen entre paréntesis, haga doble-click para mostrar un dialogo (por ejemplo, al hacer doble-click a la propiedad (Font), Delphi muestra el dialogo de Fonts). Cuando usted extiende la librería con sus propios componentes, usted puede diseñar sus propios editores.

La otra pagina contiene los eventos, que son los mensajes que Delphi "atrapa" y que usted puede asignar a procedimientos. Cuando diseña sus propios componentes, también puede añadir eventos y relacionarlos con mensajes.



El diseñador nos muestra la(s) forma(s) que podemos diseñar, y puede mostrar tantas formas como usted desee al mismo tiempo. Detrás del diseñador, podemos ver la ventana del editor. El editor es donde escribimos el programa.

Lo que queremos hacer para este primer ejemplo es simplemente crear un botón que, cuando se presione, de un mensaje de bienvenida (como en la imagen anterior). Para esto hagamos lo siguiente:

Seleccione, de la "paleta de componentes", el botón (button). Después de seleccionarlo, haga click en donde lo quiera ver en la forma. Utilizando la ventana "Object Inspector", cambie la propiedad "Caption", escribiendo "&Botoncito". El caracter "&" le dice a Windows que ponga un subrayado bajo la B y permita al usuario utilizar Alt-B para presionar el Botón.

Ahora haga doble click en el botón que acaba de crear. Delphi:

1. Creara un procedimiento en su programa "unit1.pas" llamado Button1Click. En este procedimiento usted especificara lo que va a ocurrir cuando el usuario haga click en el botón. El "begin/end" de pascal especificando donde comienza y termina el botón será también añadido por Delphi.

2. Asignara el evento "OnClick" del botón (que dice al botón que procedimiento ejecutar cuando el usuario haga click en el mismo) al procedimiento "Button1Click" recién creado.

Teclee lo siguiente:

```
ShowMessage('Esto esta Padrísimo!');
```

Incluya el punto y coma de al final; el lenguaje Pascal necesita saber donde termina cada línea o agrupación de líneas y el símbolo de punto y coma es lo que utiliza para este propósito.

Ahora, presione el botón "Run" de menú de botones de Delphi. Si todo sale bien, Delphi compilara y encadenara su programa. Después esconderá las ventanas de diseño y desplegara "[Running]" en su barra de titulo.

Felicidades! Acaba usted de crear, compilar y encadenar un programa de Windows. Veamos lo que puede hacer su programa "de una línea".

Su programa tiene una forma. La forma se puede maximizar, minimizar, cambiar de tamaño y de posición en exactamente la misma manera que cualquier programa de Windows - sin una línea de código de su parte (en el futuro, cuando usted sea un super-experto y empiece a manejar "mensajes de windows", en cualquier lenguaje, se dará cuenta de cuanto trabajo Delphi le ahorro).

Su forma también tiene un botón. Presiónelo. Vera un mensaje que dice "esto esta padrísimo" con un botón de Ok.

Para cerrar el programa, presione el icono de la "X" en la parte superior derecha, como en cualquier otro programa de Windows. Su programa regresara al diseñador.

Capitulo 3. Objetos, Formas, Unidades y Two-Way Tools

Este capitulo nos ayudara a entender como funciona la POO (programación orientada a objetos), como funcionan las formas y unidades en Delphi, y como maneja delphi la sincronía entre código y diseño.

Objetos

La programación orientada a objetos ya no es una "moda". Ahora la programación por objetos esta cambiando la forma en que vemos los problemas. Este no es el momento ni el lugar para hablar de la teoría de los objetos (un buen libro de tecnología de objetos de Grady Booch dará una mucha mejor explicación que este curso acerca de lo que es un objeto en programación). Pero es importante saber las bases de lo que es un objeto para saber como se aplica a Delphi.

Un objeto es un tipo de datos que incorpora datos y código (comportamiento) en un solo "paquete". Antes de la era de la orientación a objetos, el código y los datos eran dos cosas separadas. La orientación por objetos nos permite representar de un modo mucho más conveniente al mundo real.

¿Cómo podemos modelar un objeto del mundo real con objetos "de computadora"? Veamos un ejemplo:

Ejemplo de un Objeto en Object Pascal

Supongamos que estamos haciendo un juego acerca de un Acuario. Para este juego queremos diseñar un Bonito Delfín que va a brincar con el aro y jugar con la pelota, pero queremos en el futuro extender el juego para cualquier tipo de animal marino porque va a ser una simulación. Tal como en el mundo real, necesitamos comenzar con un objeto llamado "**pescado**" (Para los amantes de la biología: Ya sé que el delfín es un mamífero y no un pescado, pero este es un ejemplo). El objeto pescado tiene sus datos, como son alto, largo, peso y color. Pero el pescado también puede hacer otras cosas, como por ejemplo nadar y sumergirse. Entonces primero hacemos nuestro objeto pescado, que tiene la siguiente forma:

```
TPescado = class(TObject)
  Largo : Float;    // El largo del pescado, en centimetros
  Alto  : Float;    // La altura del pescado, en centimetros
  Ancho : Float;    // El ancho del pescado, en centimetros
  Peso  : Float;    // Cuanto pesa el pescado, en gramos
public
  procedure Nadar(AdondeXY, AdondeXZ, AdondeYZ : TPoint);
  procedure Sumergirse(Profundidad : Float);
end;
```

Este código le dice a la computadora: TPescado es una clase que hereda de TObject (o "Un Pescado es un Objeto") . Tiene los campos Largo, Alto, Ancho y Peso, que son números de punto flotante. Sus métodos públicos (lo que todo mundo sabe que un pez puede hacer) son Nadar y Sumergirse.

Si esto parece complejo, por favor sigan leyendo. Muy pronto todo quedara claro.

Ahora el Delfín. He decidido para este ejemplo que hay dos clases de delfines, los entrenados y los salvajes (no entrenados). Así que comencemos con el delfín entrenado:

```
TDelfin = class(TPescado)
  LargodeNariz : Float;     //
El largo de la nariz del delfín
public
  procedure TomarAire;
  procedure HacerRuido( Decibeles : Integer );
end;
```

Ahora bien, este código le dice a la computadora: El TDelfin es una clase que hereda de TPescado (o "Un Delfín es un Pescado"). Esto quiere decir que **un Delfín puede hacer todo lo que un pescado puede hacer** (tiene largo, alto, ancho, peso y además puede nadar y sumergirse). Entonces ya terminamos el delfín, y no tuvimos que implementar de nuevo las funciones para nadar y sumergirse, que el pescado (y ahora también el delfín)

puede hacer. Ahora vamos a entrar en la materia del jueguito, el delfín entrenado:

```
TDelfinEntrenado = class(TDelfin)
public
  procedure JugarConPelota( Segundos : LongInt; );
  procedure BrincarElAro( Circunferencia : Integer );
end;
```

El TDelfinEntrenado es una clase que hereda no de TPescado, sino de TDelfin (o "Un delfín entrenado sigue siendo un delfín, pero..."). Al igual que la vez anterior, un delfín entrenado puede hacer todo lo que un delfín puede hacer, pero además puede jugar con pelota y brincar el aro.

Porque hacer tres objetos nada mas para representar un delfín? Supongamos que ahora queremos hacer un tiburón...

```
TTiburon = class(TPescado);
  NumeroDeDientes : LongInt;
  Bocon : Booolean;
  ComeHombres : Boolean;
public
  procedure Enojarse;
  procedure ComerPersona( Quien : TPersona);
  procedure EspantarVisitantes;
end;
```

Gracias a Delphi y la orientación a objetos, ahora estamos "re-utilizando" el código que usamos para implementar nuestro delfín. Ahora, si mañana descubrimos que los pescados pueden nadar de una manera especial, o queremos hacer la simulación detallada y queremos hacer que hagan algo mas, como nacer y morirse, todos los objetos de tipo pescado (tiburón, delfín, delfín entrenado) van a nacer y morir igual que el pescado, porque la implementación aplica al tipo y a todos los tipos de la misma clase. Ahora bien, la "jerarquía" de los objetos que hemos hecho es como sigue:

```
TObject
   +-----  TPescado
              |
              +-----  Delfín
              |          |
              |          +-----  Delfín Entrenado
              |
              +-----  Tiburón
```

La jerarquía de objetos es una especie de "árbol genealógico" que nos dice que objetos son "hijos" de otros objetos. Cómo conceptualizamos esta jerarquía? Es de hecho bastante sencillo una vez que nos acostumbramos al árbol. Por ejemplo, supongamos que queremos saber que interfaces soporta el "Delfín Entrenado". Leemos todos los "padres" de la siguiente manera: "Un TDelfin ES UN TPescado, que ES UN Tobjeto". Esto quiere decir que un objeto de tipo Tdelfin soporta todas las interfaces que el objeto Tpescado y el objeto TObject.

TForm vs Form1; Clases vs variables; Concepto Básico de Punteros

Los lectores más perceptivos han notado que Delphi utiliza una T para especificar que algo es un objeto. De hecho, aquí Delphi se parece mucho a C: Un TObjeto es la definición de la ***clase*** del objeto, mientras Objeto1 es una variable de tipo TObjeto. Así, TForm es la clase, pero la variable que usted usa para sus formas se llama Form1. La variable Button1 es un "puntero" a la instancia de clase TButton.

Supongamos que usted tiene el siguiente código en algún lugar de una forma con un botón (como la que acabamos de escribir). No lo intente, esto es teoría:

```
var
  ElBoton : TButton;
begin
  ElBoton := Button1;
  Button1.Free;
  ElBoton.Caption := 'ESTO VA A TRONAR!!!';
end;
```

La definición de variable nos dice que "ElBoton" es una variable que apunta a un objeto de clase TBoton. Después hacemos que ElBoton "apunte" (recordemos que a final de cuentas son punteros) al Button1 de la forma. Después liberamos el Button1. Ahora tratamos de cambiar el texto del botón. Obtendremos una de esas horribles "Fallas de protección general". Porque?

Pues es muy sencillo. En la línea Button1.Free, la memoria a la que Button1 apunta es liberada. Ya no nos pertenece porque efectivamente "borramos" el botón. Pero ElBoton sigue apuntando a la posición de memoria que Button1 apuntaba. Delphi, aunque es un lenguaje muy elegante, es lo suficientemente poderoso para no preguntarnos cuando queremos manipular memoria (Delphi asume, como C++, que nosotros somos el jefe y sabemos lo que hacemos). Así que cometemos el error de tratar de cambiar el texto de un objeto que no existe. CRASH!

Lista de Variables	Memoria
Button1 = $01A73F ElBoton := Button1, o sea ElBoton := $01A73F	$01A73F..$01A7FF (TButton)

Esto lo mencionamos porque en cuanto un nuevo programador de Delphi trata de manipular objetos, una u otra vez se encuentra con un problema de este tipo. Es un error común para el programador de objetos tanto en C como en Pascal.

Formas y Unidades

Las formas en Delphi son objetos de tipo "TForm". Delphi viene con una tabla con la "jerarquía" de objetos de la librería VCL. El programa que diseñamos en el capitulo anterior tenia un objeto tipo "TForm". Este objeto es la ventana estándar de Windows para Delphi. Cuando hicimos nuestro programa con una línea de código, usted no tuvo que decirle a TForm como minimizar, maximizar, restaurar, mover o cambiar la ventana de tamaño. Usted solo le dijo a la computadora: nuestro TForm1 es un objeto tipo TForm. Delphi sabe que TForm tiene la capacidad de hacer todo lo anterior (y más). Entonces el comportamiento de todas las formas que usted cree basándose en TForm es el mismo. Usted no tuvo que hacer nada para utilizar la magia de TForm - ni siquiera fue necesario saber que TForm era un objeto!

(Delphi también le ofrece poder. Si usted quiere hacer cosas muy raras, puede usted evitar TForm y utilizar TCustomForm, que es una clase abstracta para que usted haga sus formas con comportamientos muy extraños, si se avienta - pero primero hay que aprender mas).

Cada forma es guardada en un par de archivos: El archivo PAS (donde escribimos el código) y el archivo DFM (donde diseñamos la forma).

El Archivo PAS

El archivo PAS es un archivo de texto donde escribimos el código de nuestro programa. Delphi nos ayuda a escribir el código, pero eso no quiere decir que tome totalmente el control del editor, o que guarde el archivo en un formato "desconocido". Es un simple archivo de texto escrito en "Object Pascal". El siguiente código es el contenido completo del programa que diseñamos.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage('Esto esta padrísimo!');
end;

end.
```

Tour de dos minutos de Object Pascal

Object Pascal divide los programas en dos secciones: La interfase (**interface**) y la implementación (**implementation**).

Examinemos la interface de este programa: La primera sección nos dice que esta unidad utiliza las librerías Windows, Messages, SysUtils, etc. Después viene la definición de tipos (**type**), donde Delphi ha definido por nosotros la TForm1 que ha creado (en Delphi, las cosas no solo **"aparecen"** como en otros lenguajes. Delphi es un lenguaje decente y siempre pone la representación de todo el diseño en archivos legibles y modificables con el editor). Object Pascal es un lenguaje "altamente tipificado", lo cual quiere decir que cualquier variable, objeto o rutina debe ser declarado antes de ser escrito. La declaración de la forma también declara el Botón que le pusimos. Si usted se pone a poner mas botones a la forma, vera que Delphi añade secciones Button2, Button3 a la unidad de la siguiente manera:

```
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

También podemos ver que Delphi definió el procedimiento Button1Click para cuando el usuario apriete el botón, y este procedimiento pasara el parámetro (Sender) lo cual le dirá a nuestra rutina que objeto fue el que ejecuto el evento (normalmente Button1, pero esto nos da el poder de compartir un "procedimiento manejador de eventos" para mas de un objeto - veremos como hacer esto mas tarde).

Si usted quiere añadir sus propias rutinas manualmente, Delphi ya nos pre-escribio una

sección privada (**private**) y publica (**public**) para que podamos añadir funciones manualmente (existen mas directivas, como "protected" o "automated", pero las mas usadas son estas dos, por lo cual Delphi las define por nosotros. Delphi trata de ayudar sin estropear nuestro código con mucha basura que no vamos a usar). Las funciones privadas solo son visibles para TForm1, mientras que las funciones publicas son visibles para cualquier objeto que use TForm1.

Ahora la ultima sección. Si usted es muy perceptivo habrá notado que he hablado de TForm1 (la clase) pero la variable que inicializaría la clase (de la cual hablamos con anterioridad) no ha sido definido. Bueno, pues la ultima sección del área de la interface lo define:

```
var
  Form1: TForm1;
```

Aquí le decimos a Delphi que la variable Form1 es una variable de tipo TForm1. Y quien inicializa la variable Form1? Bueno, presione Control-F12 (View Unit=Ver Unidad) y seleccione Project1. Una de las líneas en el archivo de proyecto dice:

```
Application.CreateForm(TForm1, Form1));
```

Dentro de la función de TApplication CreateForm, el objeto aplicación crea un TForm1 y lo asigna a la variable Form1. Y porque el objeto aplicación? Esto es porque si Windows quiere matar la aplicación, la aplicación necesita tener una lista de todas las formas en tu programa para que puede llamar el método "Close" de todas tus formas. TApplication tiene un arreglo (array) llamado Forms donde se encuentran todas las formas de tu proyecto.

Two Way Tools y el archivo DFM

Borland ha perfeccionado una tecnología llamada "Two-way tools". Esto quiere decir que su programa esta escrito en un archivo de texto que usted puede editar inmediatamente, pero al mismo tiempo esta representado en los diferentes diseñadores de Delphi. Cada vez que usted añade un objeto, modifica una propiedad o agrega un evento, el IDE "sabe" donde añadir ese objeto, propiedad o evento en la interface, y muchas veces escribe código de inicialización con bloques "**begin-end**", listo para que usted escriba su código y siga su camino. Los "two way tools" de Borland son los más estables y rápidos del mercado, y son los únicos que le permiten editar la forma en el diseñador o directamente como texto.

En el capitulo anterior, cuando usted hizo doble click en el botón, los "two-way-tools" de Delphi escribieron el código para el evento Click del botón y lo asigno al mismo.

Regresando a nuestro proyecto, ahora veamos como guarda Delphi las propiedades de nuestro objeto Form1. La forma se guarda en un archivo llamado "DFM" (Delphi Form File), en un formato estilo "res". Pero Delphi procura mantener su arquitectura abierta, así que nosotros podemos editar el archivo directamente en Delphi. Presione Shift-F12. De la lista de formas que Delphi despliega (View Form), seleccione Form1. Delphi regresa al diseñador de la forma. Ahora presione "Alt-F12". Su unit1.pas desaparece del editor, y en su lugar el texto de unit1.dfm aparece. Su forma también desaparece, porque ahora la estamos editando "como texto":

```
object Form1: TForm1
  Left = 200
```

```
  Top = 112
  Width = 1088
  Height = 750
  Caption = 'Form1'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  PixelsPerInch = 96
  TextHeight = 13
  object Button1: TButton
    Left = 240
    Top = 112
    Width = 75
    Height = 25
    Caption = '&Botoncito'
    TabOrder = 0
    OnClick = Button1Click
  end
end
```

Este es el texto de nuestra forma. Como vera, todas las propiedades "publicadas" son guardadas aquí. Cada vez que usted cambia una propiedad en el "inspector de objetos", este archivo es modificado. Por ejemplo, cuando hicimos doble-click en el botón para escribir el código de OnClick, Delphi modifico el archivo PAS para declarar la rutina "Button1Click", pero además añadió un renglón a esta definición de la forma (`OnClick = Button1Click`), diciéndole al compilador que el evento OnClick del Botón 1 esta enlazado al procedimiento Button1Click. Cuando modificamos el "Caption" del objeto botón, Delphi lo modifico en este archivo. Si quiere Ud. puede modificar el texto "&Botoncito" para que diga "&Botonzote".

Delphi ha escrito bastante código por nosotros, no creen? Pero aun así, nos permite ver y modificar lo que queramos. En general ustedes no necesitan modificar la forma de esta manera. Pero si lo quiere intentar, tenga cuidado. Delphi puede perder "el hilo" de como esta guardada la forma si usted modifica las secciones con los nombres o tipos de los objetos. Siempre mantenga un respaldo de su DFM antes de modificar la forma como texto.

Para salir de la vista de la forma "Como texto", presione Alt-F12. Delphi. Delphi leerá la forma de nuevo (para interpretar sus cambios) y regresara al diseñador.

Capitulo 4. Usando Delphi Wizards - Primera Aplicación

En este capitulo veremos otras maneras de comenzar una aplicación en Delphi. Delphi nos ayudara a escribir algún código de inicio para diferentes tipos de proyecto. Veremos que fácil es comenzar con una base estandarizada para nuevas aplicaciones. También veremos como el Code Repository nos puede ayudar a compartir código en un ambiente de trabajo (muy importante en desarrollos grandes). Además, haremos nuestra primera aplicación en Delphi!

Object Repository

El Object Repository es un directorio donde usted puede poner código fuente básico y componentes que usted utiliza una y otra vez, como formas, su "aplicación estándar" y cosas así. Cuando usted selecciona File-New, el Code Repository muestra un una pagina con todas las opciones de Wizards y lo que contiene el repositorio para que usted elija. Usted puede no solo copiar el código a su proyecto, sino "heredar" y "usar".

Heredar una forma (inherit) es una manera muy útil de, por ejemplo, hacer que las formas de su aplicación tengan una apariencia fija. Si usted siempre tiene un Toolbar vacío y su forma siempre tiene el Caption rojo, por ejemplo, usted puede heredar de esa forma para todas las formas de su aplicación, y Delphi creara las formas basadas en su forma en vez de basarlas en TForm. Cualquier objeto puede ser heredado. Simplemente "usar" quiere decir que no vamos a mantener una copia del objeto en el repositorio y tampoco lo vamos a utilizar.

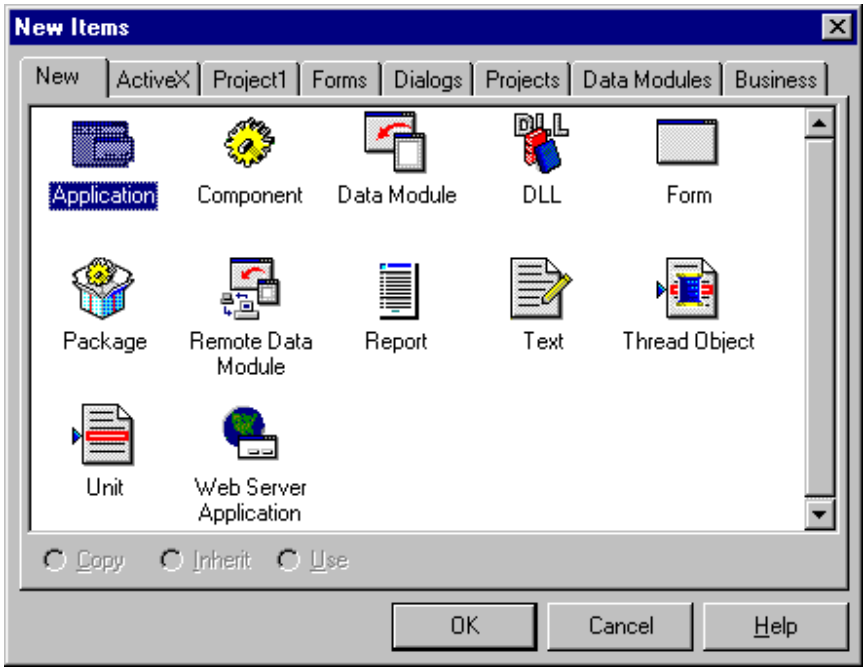
Wizards y Add-Ins

Un wizard es un DLL (o DPL) de Delphi que nos ayuda a escribir código. Los Wizards pueden ser desde muy sencillos (wizard para "New Dialog"), hasta muy complejo código que cambia la totalidad del IDE (como **Coderush**, o **GExperts**). Cuando son muy complejos y cambian el IDE se llaman "Add-Ins" o IDE Enhancements.

¿Porque el rango tan amplio de complejidad y son lo mismo? Aunque eso es para otro capitulo, por ahora le puedo decir que Delphi esta hecho en Delphi (y la librería sigue siendo el VCL), así que cuando usted hace un Add-In, todos los objetos del IDE de Delphi están disponibles para que usted los manipule. El IDE de Delphi tiene además interfaces que hacen estas modificaciones fáciles, llamadas Open Tools API. Eso quiere decir que su Add-In puede manipular el menú añadiendo items, cajas de dialogo, modificando las ventanas del IDE e incluso reemplazando tareas de Delphi! Este es el poder de un ambiente totalmente abierto a que usted juegue con él. **Coderush** es el ejemplo mas extremo en el mercado (hasta ahora) de lo que se puede hacer con este API.

Utilizando el Object Repository

Comencemos por algo fácil, utilizar uno de los wizards que vienen con Delphi. Para esto no tenemos que saber nada...! Simplemente, seleccione **File-New...** (No use File-New application porque esto creara una aplicación vacía tal como la que ve cuando comienza Delphi) - la ventana del repositorio de objetos desplegara las opciones de los nuevos elementos que puede crear:



Delphi tiene templates categorizadas por pagina:

- **New** - Esta pagina contiene templates para crear nuevos elementos como Aplicaciones, componentes, módulos de datos, DLLs, Formas, Paquetes de componentes (DPL), Módulos de datos **remotos** (MIDAS/DCOM/Multi-tier), reportes, threads y servidor de Web.

- **ActiveX** - Con esta pagina puede usted crear ActiveForms (que funcionan en la pantalla muy parecido a los applets de Java), Control de ActiveX (OCX para usar en Delphi, C++, Visual Basic o VBScript en paginas de web), Automation Object (objeto COM básico sin interface de usuario, pero con conteo de referencia integrado), Pagina de propiedades ActiveX, y Type Library.

- **Su Proyecto** - El proyecto que se encuentra abierto también aparece como pagina para que usted pueda heredar rápidamente unas formas basadas en otras. Haremos esto mas tarde.

- **Forms** - Varias formas y reportes diferentes (About, list box, labels, etc).

- **Dialogs** - Varios diálogos, así como un Wizard que lleva paso a paso por la creación de un dialogo.

- **Projects** - Wizard para aplicaciones, Aplicación MDI, SDI y aplicación compatible con el logo de Win95 (esta ultima te permite hacer una aplicación básica con los "esqueletos" de todo lo que Microsoft pide para que te ganes un logo de "Diseñado para Windows 95".

- **Data Modules**- Los módulos de datos que haga usted pueden ir aquí. En la sección de bases de datos veremos como funcionan los módulos de datos.

- **Business** - Esta pagina tiene un wizard para hacer una forma con campos (lista para

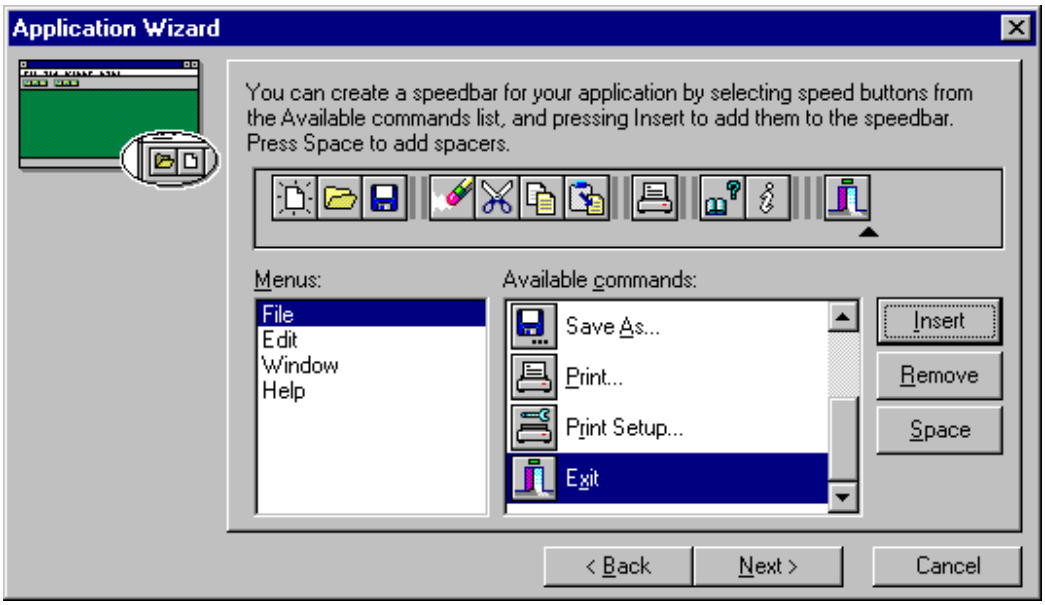
ejecutar) a partir de una base de datos, un "cubo de decisiones", un wizard para hacer reportes, y un wizard para graficación.

Cuando use Delphi para producir sus aplicaciones, podrá usar Tools-Repository del menú de Delphi para hacer mas paginas y organizar sus wizards o los módulos de datos que haga. Además, usando *Tools-Environment options - Shared repository directory* podrá usted cambiar el directorio para que este en su directorio de datos, o en algún lugar de la red (asegúrese de copiar los archivos de C:\Program Files\Borland\Delphi 3\ObjRepos, o equivalente, al nuevo directorio para que los wizards sigan funcionando).

Ahora que sabemos lo que es el repositorio de objetos, por que no usamos nuestro primer Wizard? Y de paso creamos nuestra primera aplicación de verdad...!

Seleccione **File-New...** y vaya a la pagina de **Projects**. Seleccione **"Application Wizard"** y presione OK. A continuación, veremos que el wizard nos pregunta que menúes queremos utilizar. Seleccione File, Edit, Window y/o Help (los que desee). Presione Next. Después Delphi nos pide (si seleccionamos File) que especifiquemos las extensiones. Presione **Add** y conteste **"Rich Text Format"** y **"*.rtf"** como descripción y extensión.

A continuación, nos presentara con una interface para añadir botones dependiendo en los comandos que seleccionamos:



Utilice la parte izquierda para seleccionar la pagina y la derecha para cambiar los botones. Los botones Insert, Remove y Space agregan, eliminan y ponen espacio entre los botones. En la parte superior vera usted la barra de botones tal y como se va a ver. El pequeño triángulo en esta barra representa donde se insertara el botón y puede ser movido para que usted ponga botones en medio si olvida alguno.

Cuando termine, le preguntara el Nombre de la aplicación y el directorio donde lo quiere guardar. También encontrara las siguientes opciones:

- **Create MDI Application** - Si elige esta opción, su aplicación será estilo MDI (este estilo no debe ser usado en Windows 95 según Microsoft, pero paradójicamente todas las aplicaciones de MS-Office 95 y 97 lo utilizan). El estilo MDI muestra una

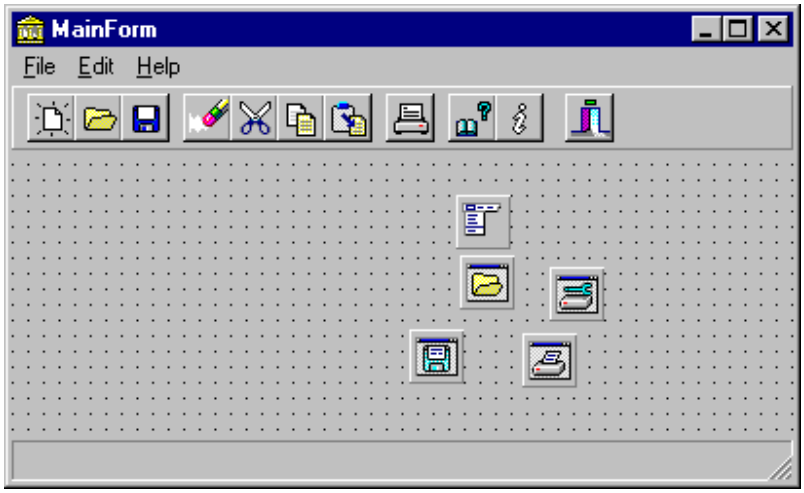
ventana "maestra" para la aplicación y documentos adentro de esta (su nombre proviene de las siglas de *Múltiple Document Interface*, en alusión a esto). Si no elige esta opción, su aplicación será SDI (single document interface).

- **Create a Status Line** - Si elige esta opción, Delphi creara una barra de status y código para que cada vez que el cursor se ponga sobre un botón el texto de "hint" se muestre en la barra de status.

- **Enable Hints** - Con esta opción, Delphi hará que todos los objetos de la aplicación tengan el desplegado de Hints automático habilitado (enabled).

Seleccione Create Status line y Enable hints. Delphi se ira a trabajar y dejara su programa listo para ejecutar por primera vez. Felicidades! Acaba usted de crear su primera aplicación en Delphi. Todavía no hace nada, pero haremos un mini-editor de textos, nada mas para ver que tan fácil es.

Nuestra Primera Aplicación en Delphi



Esta es la apariencia de nuestra primera aplicación en Delphi, la cual acabamos de crear utilizando el Wizard de Aplicación nueva dentro del Object Repository. Utilizando el Object Inspector, cambie el Caption a "Mi Editorcito". Si usted quiere, puede ejecutar este programa tal como esta. Presione el botón de "Run" en el IDE y pruébelo. Trate de presionar los botones y note que ciertos botones despliegan diálogos (Open, Save)! Pronto veremos como se hace esto. Seleccione **File-Exit** para salir de su aplicación de ejemplo y note como Delphi ya ha escrito código para salir del programa desde una opción del menú.

Como se dio cuenta al ejecutarla, esta aplicación tiene ciertos componentes que no se mostraran al usuario en tiempo de ejecución (Delphi los ha agrupado en medio de la forma sin alinearlos). El VCL de Delphi se divide en dos clases de componentes: Componentes **Visuales** y componentes **No Visuales**. Los componentes no visuales son representados por Delphi como iconos con sombra menos "obscura" en el diseñador, y no se pueden contener dentro de un panel o barra de botones (porque como no se ven, no tiene caso ponerlos "adentro" de un componente visual).

Nota: El hecho de que el VCL se llame "Visual Component Library" y

tenga componentes "No Visuales" ha inspirado a gente a decir que el nombre "VCL" esta mal utilizado. En nuestra opinión particular, el nombre esta bien utilizado porque la librería es visual durante su diseño.

Los componentes no visuales en esta forma son el Menú principal, el dialogo "Open", el dialogo "Save", el Dialogo "Print" y el dialogo "Print Setup". Estos objetos del VCL son los diálogos estandard de Windows encapsulados en un objeto de Delphi (o sea que se verán en español, ingles, alemán, etc., dependiendo del idioma de Windows en la maquina "cliente").

Nota: El menú principal se representa en la pantalla como un componente no visual, pero lo podemos ver durante ejecución y diseño. La razón es que Delphi, en su énfasis de dar un "look-and-feel" idéntico al diseñador, decidió poner un menú "de verdad" en tiempo de diseño (para que responda a los cambios globales de Font y color en tiempo real, etcétera. Pero si usted pudiera modificar ese menú en-linea, Delphi tendría que encadenar el código de modificación de TMainMenu dentro del ejecutable final, lo cual agregaría "basura" a su ejecutable (el no hacerlo violaría las reglas de orientación a objetos). Así que tenemos en este caso en particular un componente no visual que se despliega en la pantalla (o un componente visual con un editor por separado, como lo quiera ver). Cuando veamos como escribir editores de propiedades en Delphi todo esto quedara muy claro. Si no entiende, por mientras "tenga fe" en que Delphi esta bien diseñado, pero *MainMenu* es un caso un tanto especial.

Para editar el *MainMenu*, haga doble-click en el icono del menú (componente no visual) y vera un editor de menúes.

¿Cómo utilizamos un componente no visual? De hecho, no es muy difícil, pero si requiere un poquito de Código. Al ponerlo en la forma Delphi se encarga de crearlo y destruirlo sin tenernos que preocupar. Veamos el código que Delphi nos ayudo a escribir. Dentro de la ventana de diseño de su aplicación (como la que se muestra arriba), vaya al menú de File y seleccione Open. Delphi ira al evento de FileOpen dentro de su código (que por cierto, Delphi ha escrito para usted con todo y lógica básica, listo para que lo implemente):

```
procedure TMainForm.FileOpen(Sender: TObject);
begin
  if OpenDialog.Execute then
  begin
    { Add code to open OpenDialog.FileName }
  end;
end;
```

Este evento, que esta enlazado al evento "Onclick" del item del menú FileOpenItem, nos muestra que para accesar uno de estos objetos "invisibles" solo tenemos que mencionarlo por nombre y llamar uno de sus "métodos" (procedimientos o funciones de los objetos) con un punto en medio, en el formato **`Objeto.Metodo`**. (de la misma manera en que C++ utiliza el formato **`Objeto->Metodo`**). En el caso de todos los Dialogs estándar de Delphi que se encuentran en la pagina "Dialogs" de la paleta de componentes (Open, Save, Print, Print Setup, Find, etc), el método para mostrar el dialogo al usuario se llama **`Execute`**. Execute es una función que devuelve un valor Boolean (lógico). Si Execute devuelve True, quiere decir que el usuario completo el dialogo con éxito y salió con OK. Si Execute devuelve False, quiere decir que el usuario no completo el dialogo. Por esto Delphi ha encerrado la sección donde dice "Añada código para abrir..." dentro del If-Then. De este modo el archivo solo abrirá cuando el usuario seleccione OK (y Cancel funcionara tal como esperamos como usuarios, no haciendo nada). Si se asoma al código de Save, print, etc., vera que es el mismo asunto.

Ahora que ya probamos nuestro programa comencemos a hacer cosas para que de hecho haga algo. Dijimos que vamos a crear un editor de textos. Pues comencemos. En la paleta de componentes, vaya a Win32 y seleccione el componente **RichEdit**. Ponga uno de estos en medio de la forma. En el Object inspector, cambie la propiedad **Align** a **alClient**.

> **Nota**: Delphi tiene nombres para diferentes items asociados con un tipo de propiedad. Estos se llaman "enumeraciones" o "enums". En este caso, la propiedad Align del objeto TControl (y sus derivados, que incluyen a TRichEdit) es de tipo **TAlignment**, que puede ser **alNone**, **alTop**, **alBottom**, **alLeft**, **alRight** o **alClient**. Internamente durante compilación, Delphi compila esto en un set numérico. C tiene capacidades parecidas a través de la directiva **typedef enum**, pero en Delphi es mucho más simple (por supuesto). Internamente, las imágenes creadas por Delphi con este método en memoria son idénticas a las de C y por ende, es igual de rápido.

Veremos que el RichEdit contiene el texto "RichEdit1". Esto como que no va en un programa, así que vamos a borrarlo. Haga doble click en la propiedad "Lines" y vera una ventanita para editar textos. Elimine el texto.

Ahora vamos a escribir un poco de código, así que truénese los dedos y agárrese del teclado (como nosotros) ☺

Vamos de nuevo a nuestro código. Primero necesitamos permitir al usuario grabar el texto que escriba en el RichEdit. Así que vayamos a **File-Open** utilizando el menú dentro del diseñador. Delphi nos mostrara el código dentro de esa sección. En el código, escriba lo siguiente:

```
procedure TMainForm.FileOpen(Sender: TObject);
begin
  if OpenDialog.Execute then
  begin
```

```
    { Add code to open OpenDialog.FileName }
    RichEdit1.Lines.LoadFromFile(OpenDialog.Filename);
  end;
end;
```

Fue solo una línea... Pero una línea bastante poderosa. Vamos a probarla primero y después explicare. Ejecute su programa. Ahora, antes de hacer nada, ejecute WordPad (el editorcito que viene con Windows). Escriba algo bonito, con Fonts y cosas. Ahora grábelo a disco, seleccionando el formato RTF. Corra nuestro programa (que ahora puede leer un archivo) y ejecútelo. Cargue el mismo archivo. Ahora compare:

No esta mal para una línea de código, verdad? Nuestro programa, aunque todavía no tiene botones para edición especifica de fonts y colores, soporta los mismos. También, si se da cuenta, soporta "word wrapping". Felicidades! Acaba usted de terminar de hacer la parte mas pesada de los editores de Windows 3.1, el soporte interno de Fonts.

Esto es código optimizado, compilado en un ejecutable de 273K en nuestro caso. Si vamos a Project-Options y eliminamos el "debug Info" podemos reducir el tamaño un poco. Y si decidimos utilizar "paquetes" Seleccionando **Project- Options- Packages- Compile with Runtime Packages** del menú de Delphi, nuestro ejecutable se reduce a la irrisoria cantidad

de 34K, haciéndolo un paquete fácil de distribuir por el web.

> Nota: Paquetes son un tipo de DLLs especifico para Delphi y C++ Builder que son capaces de pasar objetos en vez de solo tipos básicos (usan la extensión DPL y se encuentran en Windows\System32). Los archivos son compartidos y cada objeto esta implementado en un paquete. Usted tiene que poner en el paquete de instalación los archivos DPL en los que los objetos con los que programo se encuentran, de preferencia en un "paquete de web" (CAB) diferente. En el servidor de Borland hay un paquete CAB con solo los DPLs al cual usted puede hacer referencia en las dependencias de su propio CAB. De este modo el navegador solo baja el paquete con los objetos de Delphi una vez, y las veces subsiguientes baja únicamente 32K. También hay un CAB por separado para la Database Engine que solo se usa si usted usa tablas o queries.

¿Cómo funciona? Bueno, de hecho es muy simple: El objeto RichEdit ya tiene soporte para diferentes tipos de letra, y entiende el formato RTF. La propiedad Lines del objeto es a su vez un objeto de tipo TStrings. TStrings es una muy veloz encapsulación de una lista de cadenas de caracteres. Cada línea del control TRichEdit esta "mapeada" a una línea del objeto TStrings contenido dentro del RichEdit. RichEdit despliega el contenido del TStrings siguiendo las reglas del formato RTF. Por ejemplo, la ultima sección del RTF contiene, en realidad, lo siguiente:

```
{\rtf1\ansi\ansicpg1252\deff0\deftab720{\fonttbl{\f0\fswiss MS Sans Serif;}{\f1\froman\fcharset2 Symbol;}{\f2\froman Times New Roman;}{\f3\froman\fprq2 Bookman Old Style;}{\f4\fswiss\fprq2 Arial Black;}{\f5\fswiss\fprq2 Tahoma;}{\f6\fswiss\fprq2 System;}}
{\colortbl\red0\green0\blue0;\red0\green0\blue255;\red0\green255\blue0;\red0\green255\blue255;
\red128\green0\blue0;\red128\green0\blue128;\red0\green128\blue0;\red255\green0\blue0;}
\deflang1033\horzdoc{\*\fchars }{\*\lchars }\pard\plain\f2\fs32 Tengo que poner \plain\f3\fs32
Tipos de letra \plain\f5\fs52 diferentes\plain\f3\fs32 \plain\f2\fs32 y \plain\f4\fs48\cf4
c\plain\f4\fs48\cf1 o\plain\f4\fs48\cf2 l\plain\f4\fs48\cf3 o\plain\f4\fs48\cf7 r\plain\f4\fs48\cf5
e\plain\f4\fs48\cf6 s\plain\f2\fs32 .}
```

RichEdit interpreta esto como los diferentes colores y tipos de letra específicos. Imagínese cuanto tiempo nos han ahorrado implementando todo esto! Un poco de crédito debe ir a Microsoft, ya que RichEdit es uno de los controles estándar de Win95 definidos en commctrl32.dll - pero ellos cambiaron las complejidades de usar el formato RichEdit con las complejidades de mantener Window Handles y mensajes de Windows a todo momento. Delphi nos da un componente que ponemos en la forma y punto.

El Objeto TStrings es uno de los objetos internos que más se utilizan en Delphi. Tiene la capacidad de grabarse y cargarse del disco con una método (LoadFromFile/SaveToFile), de buscar una cadena dentro de la lista con el método IndexOf, de mantener un puntero de memoria por cada objeto dentro de la cadena, lo cual lo hace ideal para poner descripciones de texto en una lista de objetos (como por ejemplo una lista de bitmaps con nombres de archivo) en memoria, y muchísimas cosas más. Vea la ayuda de TStrings para saber todo lo que puede hacer con listas, y por ende, todo lo que puede hacer con la propiedad Lines del RichEdit.

Bueno, ya explicamos como cargar archivos, ahora es responsabilidad de usted terminar el

programa y hacerlo que también grabe. Recuerde que Windows graba con dos eventos diferentes; save y save as. Save debe funcionar solo cuando el FileName de SaveDialog no este especificado. De otro modo, Save debe llamar a Save as. Si se siente aventurero, trate de hacerlo imprimir (es muy fácil). Notara que RichEdit "sabe" imprimir en colores y con sus tipos de letra, sin preocuparse por drivers, etc (lo cual sacara lagrimas de felicidad a todos los que vienen de programar ambientes DOS)...! Buena suerte!

Capitulo 5. Acceso a Bases de Datos en Delphi

En este capitulo veremos como Delphi maneja el acceso a bases de datos, incluyendo las librerías requeridas y los componentes que Delphi utiliza. Se asume que usted sabe lo que es una tabla y ha usado sistemas de base de datos básicos, como dBase. No se asume experiencia previa con SQL, pero ayudaría :-) En este curso estaremos utilizando SQL local en general. Si hay algo para lo cual necesitamos utilizar una base de datos grande (porque SQL local no soporte esa función o que no funcione como debe ser para efectos de los ejemplos), utilizaremos formato genérico SQL-92 exclusivamente (pero sí de verdad quiere saber exactamente lo que uso, estoy utilizando InterBase, que viene con Delphi 3 Profesional y Cliente/Servidor).

Estructura de Acceso de Datos en Delphi

Desde los inicios de Delphi, Borland decidió crear una estructura de acceso de datos abierta. Una buena parte de la librería de clases VCL esta dedicada a trabajar con acceso de datos de una manera abierta y flexible. Delphi y C++ Builder tienen métodos de acceso a bases de datos únicos en la industria por su flexibilidad y facilidad de uso.

> **Nota**: Si usted tiene C++ Builder (BCB), la estructura de acceso de datos es idéntica a la de Delphi, gracias a que comparten la librería de clases VCL. La versión 1 de BCB tiene el VCL de Delphi 2, y la versión 3 de BCB tiene el VCL de Delphi 3 (mas unas cuantas cosas que están en el nuevo C++ y que Delphi seguramente tendrá).

Gracias al hecho de que todo en Delphi es un objeto, los creadores de Delphi crearon, entre otros, un componente abstracto llamado "TDataset". Como su nombre lo indica, un dataset es un "set de datos". Como podrá ver, el set de datos abstracto no especifica el origen de la base de datos, modo de conexión o marca del producto que esta en el servidor. Ni siquiera especifica si nuestro set de datos es una tabla física, un query de SQL o un archivo de texto! De esta manera se generaron los componentes que heredan de TDataset bajo una base común.

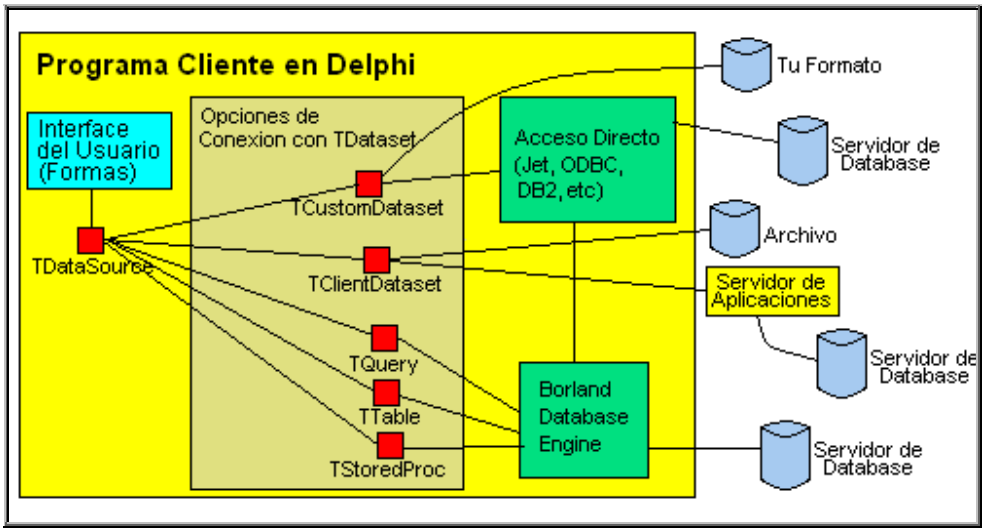
Los componentes TTable y TQuery son los componentes mas comúnmente usados para acceso básico a bases de datos (pero no los únicos). Estos componentes representan una tabla y un query de SQL, respectivamente. Para mantener el acceso de datos compatible con los diferentes productos de Borland, se desarrollo una serie de librerías redistribuibles llamada "Borland Database Engine" (BDE). Esta librería debe ser instalada en la maquina del cliente para que TTable y TQuery funcionen correctamente.

El Borland Database Engine es primordialmente una librería para manejar ***cursores*** de SQL. Muchos lenguajes son capaces de operar con un solo "renglón" de datos de la tabla a la vez, por lo cual el manejo de los cursores de SQL no es muy satisfactorio. Pero Delphi utiliza el Database Engine como una "traducción" de cursores de SQL en el formato de Queries y Tablas interno de Delphi. Además, el Database Engine nos permite accesar el origen de los datos (la base de datos para la cual estamos escribiendo) a través de "drivers" intercambiables de acceso directo, o si así deseamos, utilizar ODBC para accesar estos datos.

Un ***cursor*** es un concepto de SQL que se refiere a un set de renglones, un orden en que se encuentran esos renglones, y un "renglón actual" dentro de ese orden. Por ejemplo, cualquier "set de resultados" de un enunciado SELECT de SQL, si no devuelve un error, devolverá un cursor (vacío o con información).

Para soportar las muchas maneras en que Delphi puede accesar datos, y para facilitar enormemente al programador preocupaciones acerca de cambios en la base de datos que utilizan tus programas (por ejemplo: " Soy programador de Sybase y ahora la compañía cambio a Oracle... Oracle y Sybase utilizan DLL's de acceso muy diferentes... AAAARRRRGHHH!"), Borland ha creado un muy elaborado set de BDE (Borland Database Engine), drivers para BDE y componentes en el VCL. Cómo resultado, usted puede programar de una manera muy parecida (casi idéntica!), ya sea que usted utilice tablas de dBase (a través de una parte del BDE conocido como Local SQL) o bases de datos grandes como Oracle, Sybase o Interbase.

El siguiente diagrama muestra las maneras en que Delphi y la base de datos pueden conectarse (esta un poco simplificado, ya que hay algunos componentes intermedios - para mayor información consulte la documentación de Delphi):



Complicado? No se preocupe, esta sección cubre todas las posibilidades de acceso de datos, incluyendo la nueva tecnología MIDAS, así que no lo tiene que entender en su totalidad. Usualmente usted solo seleccionara uno de los métodos de acceso a bases de datos (dependiendo de su software existente y necesidades de conexión), y se olvidara de todos los demás. Cuando siga las secciones practicas en el siguiente capitulo, refiérase a este diagrama para "descubrir" que modo de acceso de datos estamos utilizando en cada sesión practica. Esta estructura pone todo el poder de acceso a bases de datos de muy diferentes tipos en sus manos, en forma de simples componentes que usted puede poner en sus formas. Al mismo tiempo, le proporciona suficiente poder para escribir su propia estructura de base de datos si eso es lo que usted quiere, sin cambiar los componentes en el lado del cliente.

Porque SQL

Muchos programadores de Clipper, dBase y FoxBase me preguntan "que onda con SQL, y porque es tan importante aprenderlo?". La respuesta es sencilla: **COMPATIBILIDAD**. Si bien es cierto que muchas compañías tienen su propio "tipo" de SQL, la especificación SQL-92 es soportada por todos y cada uno de los lenguajes de bases de datos grandes. Además, Microsoft y Borland han añadido soporte SQL para todos sus lenguajes. El resultado de esto es que, si usted aprende SQL-92, usted puede programar básicamente para cualquier base de datos sin importar su tamaño y complejidad. Claro esta, después de aprender SQL, se puede especializar en cualquier "sabor" de SQL en particular. Pero recuerde, cualquier cosa que programe en Delphi para un especifico "tipo" de SQL funcionara, pero le hará más difícil moverse a otro tipo de servidor en el futuro. Dependerá de usted hacer este tipo de evaluaciones.

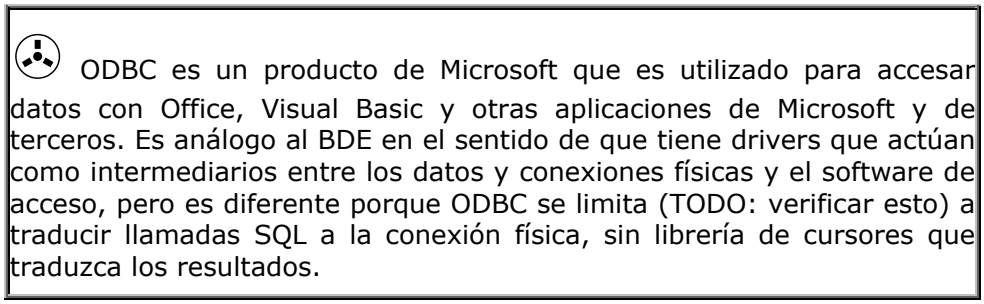
Un buen libro de SQL es "SQL for Dummies", por Allen G. Taylor. Además de ser un libro de SQL en general muy fácil de entender. El libro explica SQL en general sin especificar una versión de base de datos en particular (Yo sospecho que, ya que esta usando las herramientas de Delphi, esta programando en Interbase, como yo).

Si usted cree que le pueda interesar migrar a una base de datos SQL en el futuro y utiliza Delphi o C++ Builder, programe en Delphi como si ya tuviera una, utilizando los drivers de SQL local de Paradox, dBase o MSAccess. Después le será más fácil migrar su base de datos.

Configuración del BDE - Aliases

Un Driver de BDE es un componente de software que actúa como intermediario entre la librería de cursores del BDE y los datos Físicos. Delphi viene con drivers para dBase,

Paradox e Interbase, además de un driver para ODBC, que le permite accesar cualquiera de las bases de datos que ODBC soporta. En ediciones cliente/servidor, Delphi cuenta con drivers "nativos" de SyBase, Oracle y MS-SQL Server.



Esta estructura requiere de un concepto llamado "Alias". Un alias de BDE es un conjunto de especificaciones que describen el método que el BDE utilizara para accesar el servidor, incluyendo driver físico, mapa de caracteres internacionales, etc.

Los Aliases le permiten especificar todos aquellos parámetros que hacen a su aplicación depender de un servidor en particular afuera de su ejecutable, en un archivo de configuración en la maquina del cliente. El programa de instalación InstallShield que viene con Delphi Professional y Client Server (no estoy seguro si viene con Standard) le permite instalar el BDE con su aplicación y especificar los aliases necesarios, sin molestar configuraciones ya existentes.

Componentes No Visuales de Acceso a Datos

Del lado del cliente en Delphi, la paleta de componentes llamada "Data Access" contiene todos los componentes no visuales que usted puede poner en su aplicación para conectarse a una base de datos. Los componentes más importantes incluyen:

TSession

Un alias se conecta a una "sesión", representada por el componente TSession. Esta sesión es una conexión Física a la base de datos. Los programas de Borland pueden compartir sesiones globalmente, y cada programa puede tener cualquier numero de sesiones simultaneas. De este modo, usted puede compartir una sola conexión (sesión) a la base de datos para economizar licencias, sin importar cuantos ejecutables este utilizando para accesar los datos, o si las licencias por conexión no son limitadas en su base de datos, puede utilizar varias sesiones en una sola aplicación para programar sus consultas con multitarea.

Incluso si usted no pone un componente TSession en su aplicación, Delphi define uno por cada aplicación, representado por la variable de objeto global "Session".

TDatabase

TDatabase es la representación de una base de datos. Tal como en su servidor de SQL usted puede crear mas de un "contenedor de base de datos", Delphi puede compartir una conexión (TSession) entre varios contenedores de bases de datos (TDatabase). Tal como en la sesión, si usted no pone un TDatabase en su aplicación Delphi especifica uno en la

variable de objeto global "Database".

 **Programadores de dBase/Clipper**: Una base de datos, en el sentido de SQL, es una colección de tablas, vistas, procedimientos SQL y tipos de datos. En los viejos tiempos de dBase, muchos programadores "entendían" que una base de datos era un archivo DBF, y el concepto de Tablas no existía. Ahora, si usted utiliza DBF o Paradox, Delphi entenderá por **base de datos** el ***directorio*** donde se encuentran los DBFs en cuestión, y las **Tablas** serán ***cada uno de los DBFs***. Esto quiere decir que si esta usted acostumbrado a poner sus DBFs en diferentes subdirectorios, cada directorio representara un TDatabase diferente. En Delphi un TDatabase para tablas locales no especifica el tipo de tabla, así que usted puede mezclar tablas Paradox y dBase en el mismo directorio sin problemas.

TTable

TTable es la representación de una tabla, y es a lo que están acostumbrados los programadores en Clipper. TTable representa una tabla entera en Delphi. TTable contiene información del tipo de tabla (que solo se usa en tablas locales - paradox, dbase, etc) y del nombre del índice, así como filtros. TTable tiene entre sus métodos, First, Next, Last, etc.

Programadores dBase/Clipper/Paradox: Es MUY IMPORTANTE aprender que, mientras ustedes estarán mas inclinados en un principio a usar TTable que su "primo" TQuery, TTable representa la tabla COMPLETA. Esto puede parecer trivial para los usuarios de bases de datos locales, pero en SQL es muy importante. El abrir un TTable por si mismo (sin filtro) contra un servidor SQL hará que Delphi ejecute un SELECT * FROM LATABLAQUESEA, lo cual, en un ambiente de producción, puede querer decir traerse dos o tres millones de renglones por el cable. En general TTable no es muy usado en un ambiente Cliente/Servidor (aunque tiene sus usos). Tenga cuidado!

TQuery

TQuery es la representación de cualquier comando de SQL que regresa un cursor de SQL (ya sea de tipo SELECT o un "Stored Procedure" - vea TStoredProc). Este cursor de SQL se comporta como una tabla. Los programadores con experiencia en SQL encontraran a TQuery muy útil, ya que se comporta como un TTable (en el sentido de que tiene First, Next, Last, etc), pero al mismo tiempo es un comando de SQL. El programador de SQL no tiene que hacer "Fetch". Además, cuando el usuario edita campos asociados a un TQuery, Delphi se encarga de generar los comandos "UPDATE" y "DELETE" si el resultado del Query es Read/Write (vea TUpdateSQL).

Programadores dBase/Clipper/Paradox: Probablemente una de las mejores cosas que Delphi tiene para los usuarios de tablas locales es que Delphi tiene un pequeño SQL a través de los drivers de tablas locales. Esto quiere decir que usted puede programar TQuery como si su tabla DBF tuviera un "servidor SQL". De esta manera, usted puede mantener (si programa con atención a los detalles particulares de cada versión de SQL) una sola base de código para tablas locales y bases de datos gigantezcas.

TDataSource

El Datasource es, como su nombre lo indica, una representación del "origen de los datos". DataSource no requiere el BDE, pero es el componente que "mapea" los componentes visuales de base de datos con los no visuales. El hacer un componente intermedio es importante porque de este modo podemos tener varias representaciones de los mismos datos sin tener que "atar" la representación de los datos en pantalla a la representación "física" de los datos. Un DataSource se conecta a su vez a cualquier componente tipo Dataset, como TQuery, TTable, TClientDataset, o el que usted haya creado! DataSource nos permite ser consistentes en la representación de nuestros datos en pantalla sin preocuparnos de donde vengan los mismos (Tablas, queries, o nuestros propios métodos).

TUpdateSQL

En SQL-92, existe una limitación en cuanto a los enunciados "Select". Existen muchas circunstancias en las cuales SQL regresara un cursor de lectura únicamente, en vez del cursor de Lectura/Escritura que nos gustaría obtener. Por ejemplo, si el enunciado SQL especifica una unión de tablas, SQL nos devolverá un cursor que podemos representar como el resultado de un Query en Delphi, pero no podremos editar el resultado. Obviamente esta limitación es muy severa si queremos hacer ciertas "maravillas" con nuestra aplicación (queries editables con referencias en texto es una de estas cosas).

Aquí es donde UpdateSQL hace su magia. Todos los queries tienen un Edit, Insert y Delete. UpdateSQL puede ser encadenado a un query cuyo resultado será de lectura únicamente para decirle a Delphi que vamos a hacer cuando el usuario edite, inserte y borre un registro. En el Edit, podemos poner un SQL UPDATE, en Insert podemos poner un SQL INSERT, etcétera. De este modo nos "brincamos" la limitación de SQL utilizando sentencias especificas, pero lo mantenemos automatizado gracias a Delphi.

TStoredProc

Aunque TQuery puede ejecutar un stored procedure que devuelva un valor (y tenemos que usar un TQuery si queremos recuperar ese valor), a veces el stored procedure no devuelve nada. TQuery espera un valor de regreso, y nos devolverá un error si el servidor no devuelve un valor. Por este motivo, para esos procedimientos necesitamos un objeto que nos permita ejecutar un comando SQL sin esperar respuesta. TStoredProc es ese comando.

 En SQL, un "Stored Procedure" es un procedimiento escrito en lenguaje SQL. Puede ser tan sencillo como un Select con variables y un "join" de tablas, o tan complicado como un procedimiento Batch que calcula los salarios de todos los empleados y los pone en la tabla de nomina.

Existen más componentes no visuales de acceso a datos, pero estos son suficientes para hacer muchas aplicaciones cliente servidor sin problemas. Cuando veamos "Multi-Tier" y Objetos de Negocios (Business Objects), que es una nueva capacidad de Delphi 3 Cliente/Servidor, veremos otros modos de acceso de bases de datos un poco más "indirectos".

Componentes Visuales de Acceso a Datos

Los componentes visuales de base de datos son los que ponemos directamente en la forma para desplegar información y permitir al usuario editarla. Todos los componentes de este tipo son llamados "Data-aware", y tienen el prefijo "DB". De este modo, un componente de edición "Edit" en su versión "Data Aware" es llamado "DBEdit". ComboBox se convierte en "DBComboBox". Muchos de los componentes visuales (Incluyendo el Grid) tienen una versión "Data Aware". Además, usted puede crear sus propios componentes data aware.

Además, existe un componente llamado "DBNavigator", que es una serie de botones "mapeados" a los comandos Next, Prior, Last, Edit, Delete, etc., de cualquier DataSource.

Todo esto le permite a usted puede poner unos cuantos componentes data aware en su forma (junto con un DBNavigator), conectarlos a un DataSource (que se encontraría conectado a un Dataset abierto), y correr su programa, y así de fácil tendría usted una aplicación básica con acceso a base de datos, sin tener que escribir una sola línea de código. Lo cual nos lleva a nuestro próximo capitulo, donde haremos una aplicación de base de datos casi sin tener que usar código!

Capítulo 6. Escribiendo una Aplicación de Base de Datos

En este capitulo escribiremos una aplicación pequeña de base de datos. Como necesitamos asegurar que todos tengamos los mismos datos, comenzaremos por utilizar los datos que vienen con Delphi, en el alias DbDemos.

Nota: En este ejemplo, aunque estamos usando el alias DbDemos (que es un directorio con tablas de dBase), estoamos programando como si estuviéramos desarrollando para un servidor SQL (dBase, Paradox, etc). Con esto, nuestra intención es convencerlo de que: 1. Usted puede (y en nuestra opinión debería) programar en SQL local para hacer más fácil la migración posterior a bases de datos SQL, y 2. Si usted comprende como

funciona Delphi en programación con un TQuery, la programación usando un TTable es sencilla y trivial.

Además, usted estará listo más rápido para programación en SQL, y un desarrollador de Cliente/Servidor es mejor pagado que uno de "tablas". El saber y entender SQL es, hoy por hoy, una de las mayores ventajas para un desarrollador, ya que cualquier paquete de base de datos entiende al menos una "especie" de SQL.

Si usted quiere usar una versión poderosa de SQL para saber cómo es programar para ella, Delphi Professional y Client/Server cuentan con una versión de desarrollo de Interbase. Pero antes de desarrollar sobre Interbase, asegúrese de entender el contrato de licencia, donde Interbase se cobra "por asiento", como Oracle, Ms SQL Server o Progress.

Accesando Datos en Delphi

Delphi utiliza una librería llamada BDE (Borland Database Engine) para su acceso a bases de datos. El acceso de datos en Delphi es muy diferente, pero análogo, al acceso de datos en lenguajes como Clipper, Foxpro y dBase. En estos lenguajes, usted tiene "áreas" de trabajo. Dentro de esas áreas usted puede tener abierta una tabla, y le puede decir a esa tabla que vaya al principio, al final, o brinque o retroceda. Bueno, en principio, en Delphi puede usted hacer lo mismo:

En Clipper (área)	**En Delphi (dataset)**
`GO TOP`	`Dataset.First;`
`GO BOTTOM`	`Dataset.Last;`
`Skip`	`Dataset.Next;`
`Skip -1`	`Dataset.Prior;`

En Delphi usted puede tener un sinnúmero de estas "áreas", pero como en Delphi todo es un objeto, obviamente tienen su nombre. Las clases que representan sets de datos navegables (datasets) son llamadas "TDataset".

Nota: Nos referimos a "sets de datos navegables" en lugar de especificar tablas o áreas de trabajo. ¿Porqué es esto? Un set de datos navegable en Delphi puede representar virtualmente cualquier cosa. Delphi tiene una clase llamada TCustomDataset, que implementa los conceptos básicos de la navegación de datos (First, Next, Last, etc). Bajo esta clase, customBDEDataset está implementado. Esto es un set de datos abstracto

que utiliza el BDE. Una vez implementado BDEDataset, todos los sets de datos (TQuery, TTable) están implementados bajo el mismo.

¿Porqué tanta complicación? ¿Porqué no simplemente hacer acceso de datos directamente? Delphi tiene como concepto básico la implementación de clases y programación por objetos. El uso de estas clases intermedias garantiza que usted pueda hacer acceso a datos sin utilizar el BDE, por ejemplo (utilizando CustomDataset en vez de CustomBDEDataset). De hecho, usted puede manejar el acceso de datos de una manera totalmente diferente mientras Delphi cree que está utilizando los mismos datasets. El cambio en su programa se vuelve mínimo.

Vamos a entrar en materia y hacer nuestra primera aplicación de base de datos. Utilizaremos los datos que vienen con Delphi, ya que son los que nos aseguran que todos veamos las mismas pantallas. Más adelante veremos otros tipos.

Con Delphi recién abierto, busque en la paleta y ponga los siguientes componentes en su forma, ya sea utilizando doble-click (que pone los componentes automáticamente) o haciendo click en uno de los componentes y haciendo click donde lo quiera ver en la forma. Para cambiarle de nombre, seleccione "name" en el object inspector y escriba el nuevo nombre.

De la página..	**Inserte el componente...**	**Y Cambie las propiedades:**
Data Access	TQuery	Name: "qryClientes"
Data Access	TDataSource	Name: "dsClientes"
Data Controls	TDBNavigator	Name: "dbnNavegador"

Los componentes Query y DataSource que acabamos de insertar representan las partes "no visuales" del acceso a base de datos. TQuery es una representación de un Query de SQL, mientras que TDataSource nos permite conectar controles "data aware" a su aplicación. Si todo esto suena un tanto confuso no se preocupe, se volverá muy claro cuando el programa esté corriendo.

 **Estándares, comentarios y notación.-** Como programador profesional, tarde o temprano se dará cuenta de que las "convenciones de código" (los estándares que usted utiliza para nombrar sus objetos, y estilo en general) son muy importantes. Aunque esta convención es algo que es un tanto personal, es importante tener en cuenta que es importante tener una. Cualquier programador veterano le dirá que aunque la convención sea muy diferente, es más fácil leer el código de

alguien que tiene una (aunque sea muy diferente a la nuestra) que el de alguien que no tiene ninguna.

Así que aquí van dos de nuestros primeros estándares de programación: Primero, SIEMPRE póngale nombre a todo. Cualquier control que ponga en la forma debe tener un nombre puesto por usted, no por Windows. Eso hace su programa mucho más legible cuando regresa después de un año y necesita saber que hacía su programa.

¿Cómo nombrar sus objetos y variables? La mayoría de los programadores utilizan una de las miles de variaciones de la "notación húngara", creada por Charles Simonyi en Microsoft. La notación húngara especifica un prefijo que nos dice a qué clase de objeto o variable se refiere el nombre, y después el nombre, primero mayúscula y después minúsculas. Lo más usual es usar "i" para Integer, "b" para Boolean (lógico), "s" para string o "c" para caracter, etcétera. Yo utilizo ciertas extensiones para especificar objetos de Delphi como "qry" para Query, "ds" para DataSource, o "dbn" para DBNavigator. Esto hace el código más legible y fácil de seguir.

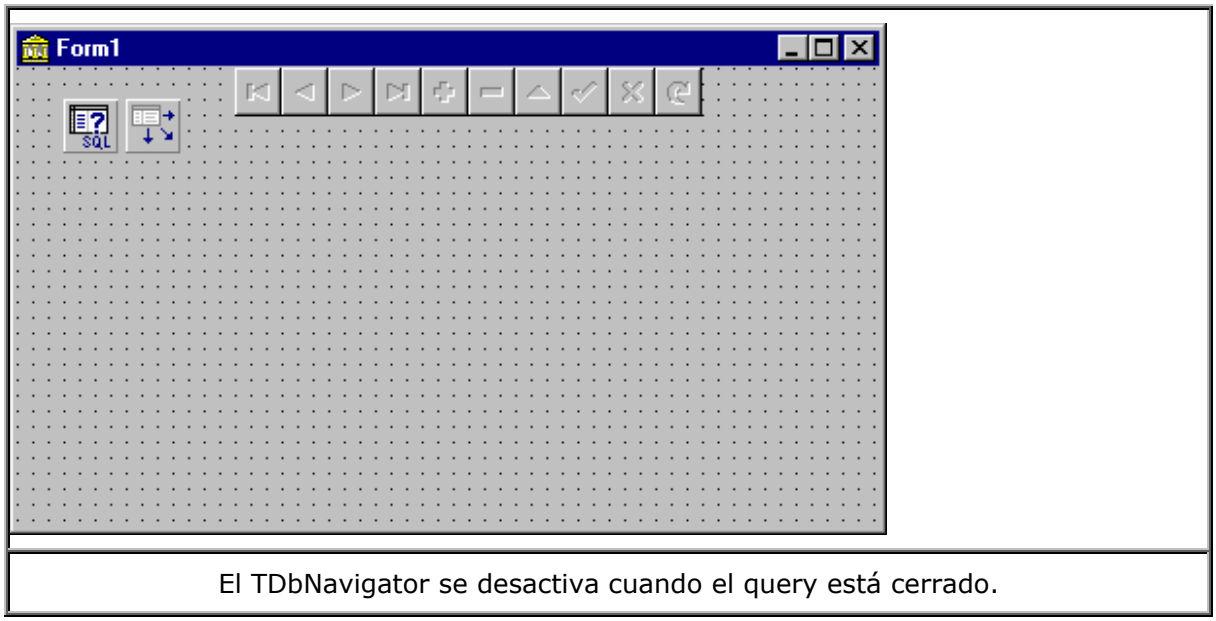
También es muy importante poner comentarios en el código. No existe programa con demasiados comentarios, así que explica todo lo que quieras. Mientras escribe, imagine que usted es un maestro que quiere enseñar el código a sus alumnos para algún ejercicio.

Ahora, observe a su código. Delphi ha insertado declaraciones dentro de su declaración de la forma especificando el nombre de la variable y el tipo de objeto que representa:

```
TForm1          =          class(TForm)
  qryClientes:                    TQuery;
  dsClientes:                  TDataSource;
  dbnNavegador:               TDBNavigator;
private
  {     Private     declarations     }
public
  {     Public      declarations     }
end;
```

Ahora, vaya al TQuery y cambie la propiedad "Databasename" a DBDemos. Lo puede escribir textualmente o seleccionarlo de la lista. DBDemos es un alias local que apunta a su directorio de Demos de Delphi. Lo puede ver utilizando el programa BDE Administrator que viene con Delphi. Ya que tenga especificado el contenedor de base de datos que desee usar por medio del alias, puede usted lanzar comandos de base de datos a cualquier tabla que se encuentre dentro del contenedor. Si usted en esta propiedad especifica un alias que esté configurado a una base de datos SQL, entonces Delphi hará una conexión a la base de datos y mostrará la caja de "login" para que usted escriba su nombre de usuario y clave de acceso en la base de datos.

Ahora seleccione el TDatasource. En la propiedad Dataset, seleccione qryClientes. Tal como en el párrafo anterior, puede seleccionarlo de la lista. Esto quiere decir que, la "fuente" de los datos va a tomar la información del set de datos qryClientes. Finalmente, en la propiedad DataSource de su TDBNavigator, seleccione dsClients. De inmediato podrá notar que los botones del navegador se desactivan, volviéndose grises y opacos. Esto es porque los componentes "data-aware" de Delphi son dinámicos, es decir, se comportan en modo de diseño tal como se comportarán en modo de ejecución. Como su Query está vacío y cerrado, el navegador está desactivado.



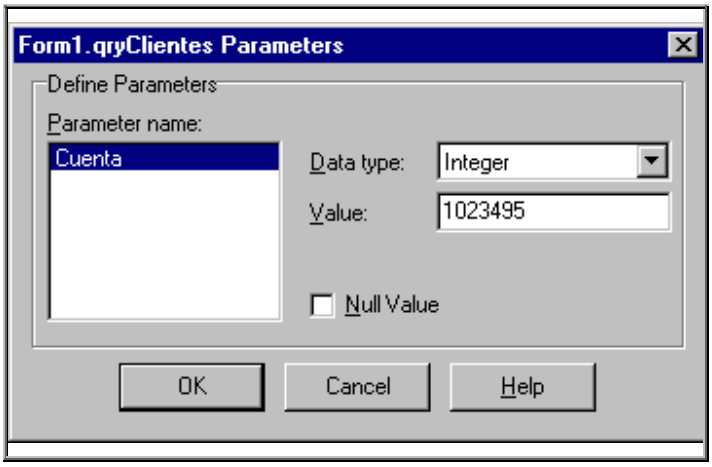
El TDbNavigator se desactiva cuando el query está cerrado.

¿Cómo le hacemos para que se active el Query? Lo que vamos a hacer es aprovechar al máximo las capacidades de diseño de Delphi. Primero, necesitamos hacer un Query en SQL que Delphi pueda utilizar en el alias DbDemos para que Delphi sepa la información de la tabla que podemos utilizar para diseñar.

Seleccione el Query, y haga doble-click en su propiedad "SQL". A continuación verá el "Editor de listas de textos" (String List Editor). Este editor es un editor de textos muy simple y pequeño que simplemente permite escribir más de una línea de texto a la vez. Escriba lo siguiente y después presione OK

```
SELECT * FROM "CLIENTS.DBF" CLIENTES
WHERE CLIENTES.ACCT_NBR = :Cuenta
```

La palabra ":Cuenta" le dice a Delphi que este query necesitará un parámetro llamado Cuenta, para el número de cuenta. Ahora, haga Doble-click en la propiedad "Params" del Query. Seleccione tipo de datos Integer y valor 1023495 por ahora. Después le daremos al usuario oportunidad de cambiarlo.



Asegúrese de que el número de cuenta sea el correcto (consulte la tabla usando Database Explorer si tiene dudas) para que su query devuelva un registro. Si su query no devuelve nada, todavía podrá diseñar (Delphi devuelve la información de los campos sin registros), pero no verá el registro en tiempo de diseño.

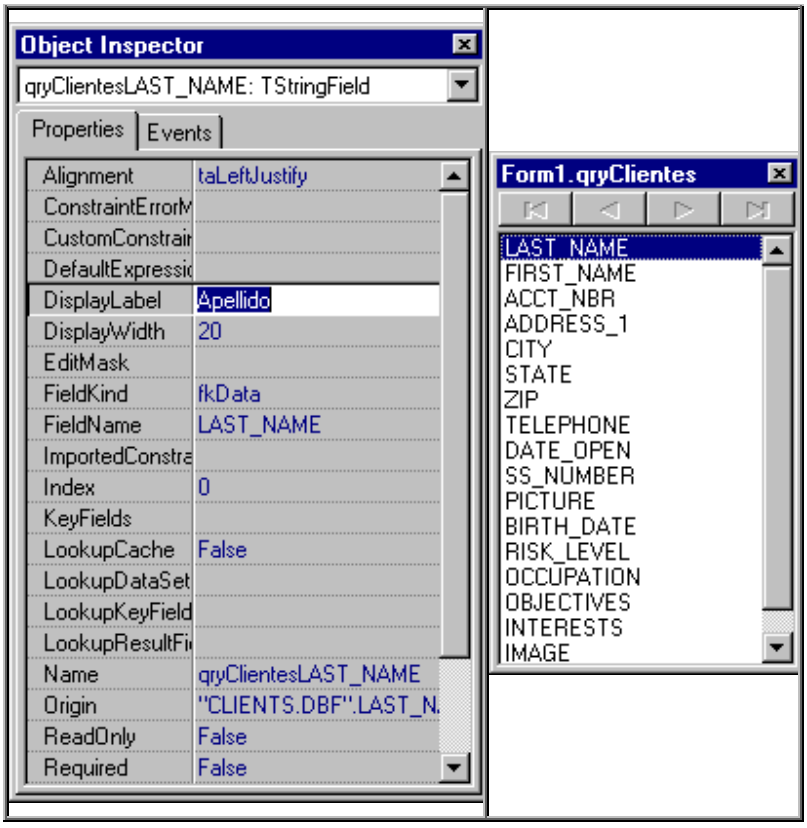
 El diseñar especificando siempre un número de cuenta (o la "clave primaria") es algo que SIEMPRE debemos de hacer en SQL, a menos que estemos **SEGUROS** de que la tabla está vacía o no tiene muchos registros. Delphi ejecutará este SQL cuando activemos el query para elegir campos, preparar relaciones o cualquier cosa. Aunque Delphi nunca abre un query dos veces si no es necesario, si usted no pone una sentencia WHERE en una base de datos grande, tendrá que esperar a que el SELECT completo ejecute antes de continuar con su diseño. Tenga esto en mente cada vez que haga un Select sin WHERE, ya sea en diseño o en tiempo de ejecución.

A continuación, vaya a la propiedad "RequestLive" de su Query y póngala en "True". Esto es importante porque SQL siempre devuelve resultados "Read-Only", o "muertos" (dead) a menos que especifiquemos que nuestro query debe ser "vivo" (live). Esto querría decir que nuestro usuario no podría editar el registro! Hay otras condiciones que podrían no permitirnos el recibir un query "vivo", pero las veremos más adelante. Cualquier manual de SQL le explicará qué clase de queries pueden ser modificados y cuales no.

Ahora lo que queremos hacer es diseñar usando todas las ayudas de Delphi, así que, con el Query todavía cerrado, haga doble-click en el query. A continuación verá una ventana vacía que dice Form1.qryClientes. Ésta es la lista de registros existentes en el resultado de nuestro query "Clientes". Presione Control-A (o haga click con el botón derecho del mouse y seleccione "Add Fields"). Delphi ejecuta el Query internamente y nos proporciona la lista de las "columnas" (campos) del resultado del Query. Todos los campos están preseleccionados (Delphi asume que usted quiere añadir todos los campos a la forma). Así que simplemente presione {Enter} para aceptar el añadir todos los campos a su definición.

La información que Delphi recupera para saber la estructura del archivo se llama Metadata. Esto quiere decir a grosso modo "datos acerca de los datos"; es decir, la estructura física de la información resultante del Query. Para extraer el Metadata, Delphi ejecuta el SQL en su Query y extrae la información a partir de los datos recibidos. Si el query no devuelve nada, pero es sintácticamente correcto, Delphi también recibe el metadata, pero sin registros.

Ahora usted tiene una definición de campos como parte de la definición de su forma Y Puede usted comenzar a diseñar. Cuando selecciona uno de los campos, el Object Inspector muestra las propiedades de los campos. Mientras no cambie propiedades importantes como Largo y Nombre del campo, puede usted cambiar la apariencia, el "display name", y el formato de edición (Edit Mask) del campo en tiempo de diseño. Por ejemplo, seleccione el campo Last_Name y trate de cambiar su "DisplayName a "Apellido". El nombre del campo para efectos de diseño sigue siendo el mismo, pero ahora desplegará "Apellido" en el TLabel cuando lo pongamos en la forma. Además, programáticamente, el objeto qryClientesLAST_NAME, de tipo TStringField, tiene como propiedad "DisplayName" el texto "Apellido". Esto será útil para hacer mensajes de error para nuestros usuarios...

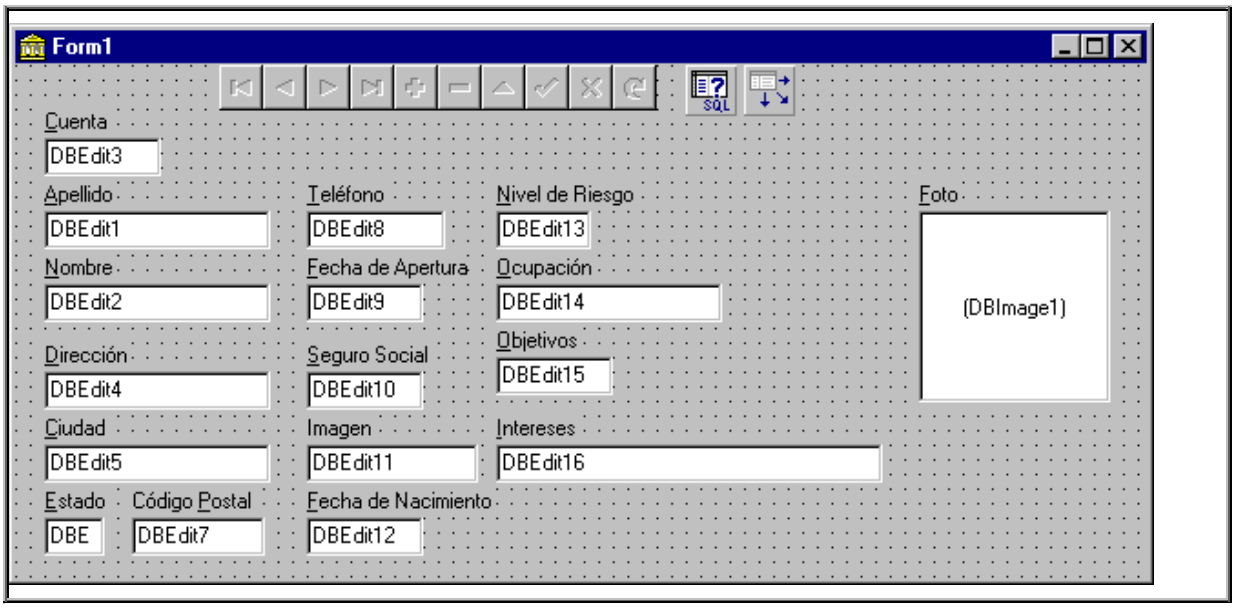


Nota: Aunque es muy bonito el poder hacer esto en tiempo de diseño, el atar su forma a los datos de esta manera quiere decir que, cuando usted cambie los nombres, añada o cambie el tamaño de las columnas de su tabla, deberá volver a esta forma y hacer que los cambios se reflejen correctamente. El diseñar visualmente de esta manera, aunque conveniente, "ata" su diseño al formato de la tabla. Más tarde veremos maneras de evitar esto hasta cierto punto (pero todo tiene su precio).

Cambie la propiedad "DisplayName" en todos sus campos y ponga títulos convenientes en cada uno de ellos (¡y por favor en Español!)... Una vez que termine de "amasar sus datos", estará usted listo para diseñar...

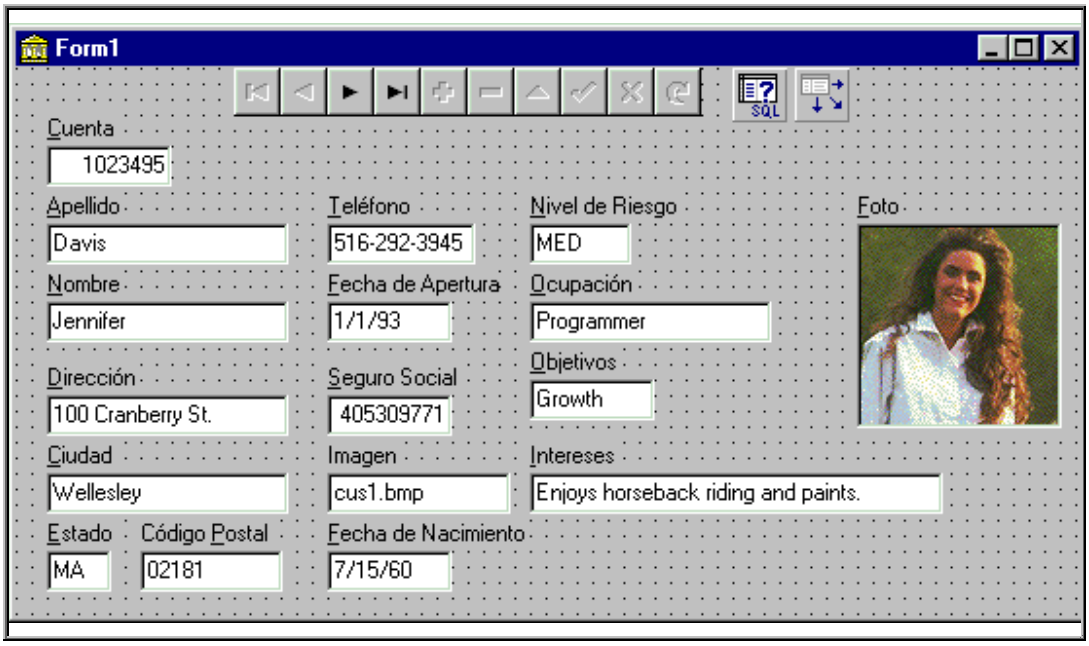
Para diseñar su forma, simplemente "arrastre" cada uno de los campos con el mouse, desde la lista de campos "Form1.qryClientes", hasta la forma, en el orden que desee que el usuario los escriba, y en la posición deseada en la forma. Cuando "suelte" cada campo, Delphi creará un componente TLabel y un DbEdit por usted, listos para recibir los datos adecuados.

Delphi es versátil también. Examine las propiedades de los "DbEdit"s que ha estado soltando en la forma. Para hacer esto manualmente, usted seleccionaría un dbEdit de "Data Controls", lo pondría en la forma, y manipularía sus propiedades "DataSource" y "DataField" para mencionar el DataSource en su forma y el campo deseado, respectivamente. No hay "gran misterio" en cuanto a la manera en que Delphi lo está ayudando a programar. Como dicen por ahí los "Borlanderos": Delphi está escrito en Delphi. A continuación presentamos un ejemplo de la manera en que probablemente se ve su forma:



"Dijimos que Delphi nos permitía diseñar visualmente.... ¿Donde están los datos?" Ah, pues es muy sencillo: Nuestro Query todavía no está abierto. Aunque Delphi lo ejecutó para

averiguar la lista de campos necesarios para el diseño, nuestro query, para propósitos prácticos, está cerrado. Vamos a abrirlo y veamos que obtenemos:



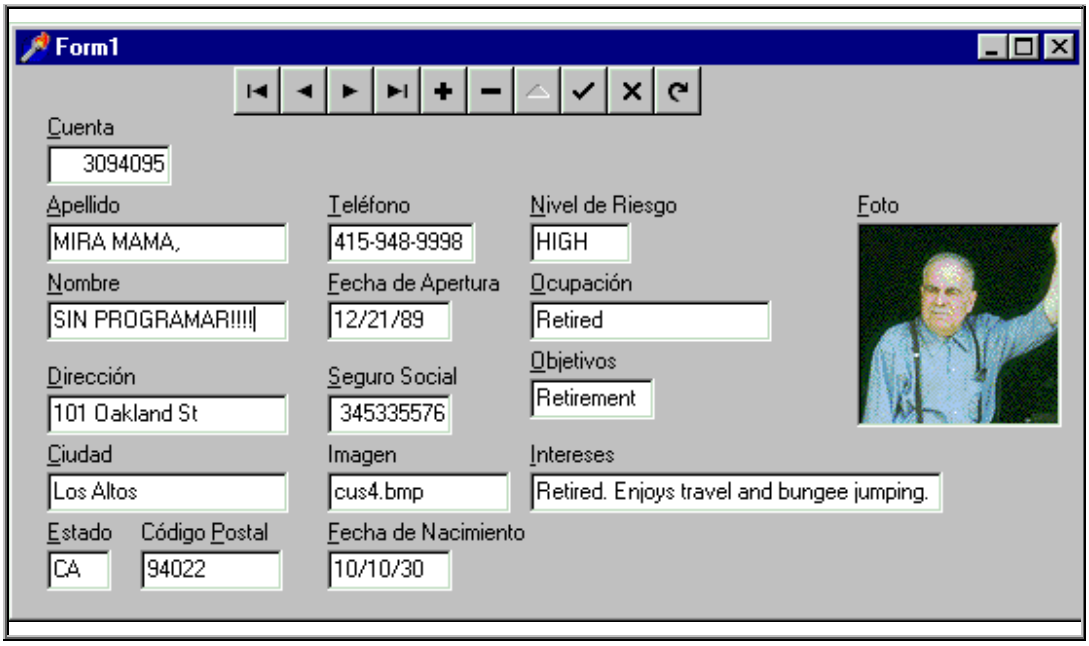
Nuestros campos han sido activados. El dbNavigator ya nos da algunas opciones. Pues sí, esa es la cuenta que especificamos en los parámetros...

Si usted quiere, puede correr ahora mismo su programa y hacer cambios, pero solo a ese registro (porque nuestro query especifica un registro únicamente). Si Delphi no le permite editar mientras ejecuta, asegúrese de que su Query tiene el RequestLive en "True".

Ahora que sabemos como se hacen las cosas, para experimentar un rato vaya al SQL en su Query y elimine la sección "WHERE" de la sentencia SELECT. Su SQL quedará como SELECT * FROM "CLIENTS.DBF" CLIENTES

Recordando todas las precauciones que vimos al principio de nuestro capítulo, sabemos que esta tabla en particular sólo tiene cinco registros, así que el tiempo de espera del select no será catastrófico (pero recuerde lo que dijimos antes; más tarde veremos varias técnicas para manejar esto). Después, abra su query de nuevo (cambie la propiedad "Active" del Query a "True", y ejecute el programa.

Felicidades! Tiene su primer programa de acceso de datos de Delphi. Todavía no hemos escrito una palabra en Pascal, pero funciona muy bien! El Navegador (La barra de botones de la parte superior) le permite editar, grabar, cancelar, refrescar los datos y moverse entre los registros. Todos sus registros funcionan, y usted puede comenzar a editar y añadir registros al archivo de Clientes de inmediato.



Finalmente, grabe su ejemplo; lo utilizaremos en otros capítulos para no tener que comenzar de nuevo cada vez. Cuando le pregunte el nombre de "unit1", teclee forma_clientes. Cuando le pregunte el nombre para el proyecto, teclee "ManejodeClientes".

En los siguientes subcapítulos de la sección "base de datos", aprenderemos mucho acerca de las muy diversas maneras en que podemos desarrollar aplicaciones de manejo de datos usando Delphi. Además nos quitaremos de algunos de los "vicios" que tuve a bien introducir en este capítulo para hacerlo más sencillo y obtener resultados más rápido, como el mantener las tablas abiertas en modo de diseño, usar SQL sin sentencias WHERE, etcétera. Siendo Delphi tan poderoso como lo es para el manejo de bases de datos, es difícil abarcar todo. Pero al menos les daremos una "probadita" de varias de las capacidades de Delphi en cuanto al manejo de información, que son muchas. A partir de ese punto, usted tendrá las herramientas suficientes para experimentar y extender sus conocimientos.

Por ahora, dése unas palmaditas en la espalda, porque ya terminó una de las secciones más largas del curso! Ha usted hecho bastante, y todavía ni siquiera hablamos del lenguaje en sí, un área fascinante que veremos más a fondo mientras viajamos por los ejemplos...

Capítulo 6.1. Cómo funciona Cliente/Servidor en 20 minutos

En este capitulo teórico explicaremos algunos de los comandos básicos necesarios para trabajar en Cliente/Servidor. Donde sea posible, utilizaremos únicamente lenguaje SQL acorde con el estándar SQL-92. La base de datos que utilizaremos para probar es InterBase 4. Es muy importante que estudie los conceptos básicos el **Capítulo 5** antes de comenzar este capítulo.

Comandos SQL más comunes

 SQL es una especie de mezcla entre un lenguaje de tercera generación como dBase y un lenguaje de línea de comando como unix shell o archivos .BAT en DOS. Los comandos en SQL son ejecutados uno por uno por el sistema de base de datos. Cada comando puede devolver un set de resultado (cursor) o no. Los sets de resultados, a su vez, pueden ser o no modificables "en vivo". SQL también permite ejecutar comandos de modificación directa sin utilizar un set de resultado de por medio.

> **Nota:** Los componentes de Delphi, por decisión de arquitectura básica, pueden manejar únicamente un set de resultado por cada llamada al SQL. Si usted necesita recuperar la información de un procedimiento que devuelva más de un set de resultado, deberá utilizar acceso directo al BDE.

Como éste curso es de Delphi y no de SQL, explicaremos muy rápido los comandos básicos que se utilizan en este lenguaje de programación, que en realidad merece un curso propio. Para mayor velocidad y entendimiento de conceptos que Delphi utiliza para enmascarar los comandos SQL en sets de datos (dataset), hemos decidido dividir la sección de comandos entre los que devuelven un set de resultado, comandos que no devuelven un set de resultado, y comandos para manipular metadatos. Por cierto, un comando SQL puede estar escrito en varias líneas.

Junto con cada comando incluyímos un poco de su sintaxis básica (varios comandos son muy complejos) y

Comandos que devuelven un set de resultado

Todos los comandos mencionados en esta sección pueden devolver un set de resultado. Estos comandos son:

- **SELECT**
- **Procedimientos (`CREATE PROCEDURE`)**
- **Vistas (`CREATE VIEW`)**

 **SELECT**

SELECT es el comando que siempre va a devolver un resultado o un mensaje de error. El comando SELECT tiene la siguiente sintaxis:

```
SELECT {Campo1 AS NombreCampo1, Campo2, ...CampoN }
       FROM {Tabla/Vista}
       [ WHERE {Condición} ]
       [ LEFT JOIN OtraTabla ]
```

Ejemplos

Por ejemplo, para hacer un Select que nos devuelve toda la información en la tabla de clientes donde el cliente tiene como domicilio la ciudad de México, DF, puede usted escribir:

```
SELECT * FROM CLIENTES
WHERE CIUDAD = "MEXICO" AND ESTADO = "DF"
```

El resultado de este SELECT será:

NUM	NOMBRE	DIRECCION	CIUDAD	ESTADO	SALDO
1234	DAVID MARTINEZ	AVENIDA DEL PROGRAMADOR #45	MEXICO	DF	$1,332.00
2374	ELVIS PRESLEY	COUNTRY MUSIC RD #485	MEXICO	DF	$503.00
5586	MARIO MORENO	LOS HUMORISTAS #24	MEXICO	DF	$45.05
1048	PEDRO INFANTE	CALLE DEL MARIACHI #4853	MEXICO	DF	$5,340.00
2894	LUIS MIGUEL	BOLEROS #5886	MEXICO	DF	$1,255.00

El comando JOIN sirve para juntar campos de dos entidades. Un uso común en SQL para el comando JOIN es la llamada "denormalización" de los resultados. Esta denormalización es solo de los resultados, para que el SQL devuelva un set consistente de resultados (siempre el mismo número de columnas). por ejemplo, supongamos que usted tiene información de una factura. Tiene usted el encabezado de la factura y además una serie de renglones con los precios y descripciones de los artículos comprados. Para devolver información que usted pueda desplegar, puede usted escribir su SELECT con un JOIN de la siguiente manera:

```
SELECT
        ENC.FACNO,
        ENC.NOMBRECLIENTE,
        ENC.CIUDAD+", "+ENC.ESTADO AS CIUDADEDO
        REN.ITEMNO,
        REN.DESCRIPTION,
        REN.PRECIO_USD,
FROM FACTURAS ENC
LEFT JOIN FACTURARENGLON REN ON ENC.FACNO = REN.FACNO
WHERE ENC.FACNO = 1
```

Examine el SQL escrito aquí. En este comando de SQL tenemos dos tablas: Una llamada **FACTURAS** que estamos "abreviando" a **ENC** (por encabezado), y la otra llamada **FACTURARENGLON**, que hemos abreviado a **REN** (por renglón). De este modo nos podemos referir a ellas más fácilmente. También hemos mezclado la ciudad y estado en un solo campo, llamado **CIUDADEDO** (para eso sirve el **AS**). De este modo puedo desplegarlo en un solo lugar para efectos de la factura. El **JOIN** junta las tablas a través del campo índice (FacturaNo), que existe en las dos tablas, y en este caso tiene el mismo nombre (**ENC.FACNO = REN.FACNO**). Finalmente, especificamos

la factura #1 únicamente para mandar eso a un reporte.

FACNO	NOMBRECLIENTE	CIUDADEDO	ITEMNO	DESCRIPCION	PRECIO_USD
1	DAVID MARTINEZ	MEXICO, DF	112232	COMPUTADORA COMPAQ PRESARIO	$2,340
1	DAVID MARTINEZ	MEXICO, DF	112522	MONITOR COMPAQ 21"	$1,340
1	DAVID MARTINEZ	MEXICO, DF	513213	3COM PALMIII	$400.00
1	DAVID MARTINEZ	MEXICO, DF	445653	LIBRO "MASTERING DELPHI 3"	$39.99
1	DAVID MARTINEZ	MEXICO, DF	445657	LIBRO "COLLAB COMP W/DELPHI 3"	$69.99

La denormalización de los resultados es algo común en SQL. Recordemos que, para el servidor, no es problema repetir los datos porque el cursor interno no se mueve de lugar. Y para nosotros, nos ahorra tráfico en la red porque, aunque el set de resultados es más grande, no tenemos que pedir más de uno. Los datos siguen normalizados en las tablas correspondientes.

Procedimientos

Un procedimiento (stored procedure) es básicamente el mismo concepto que un procedimiento en Delphi. Un procedimiento tiene un nombre y parámetros, y dentro de este procedimiento usted puede definir variables y opcionalmente devolver uno o más sets de resultado.

```
CREATE PROCEDURE MiProcedimiento
( VARIABLE1 CHAR(10), VARIABLE2 INT )
AS

{ Comandos SQL. Cualquier Select es parte del set de resultado }

TODO: TERMINAR PROCEDIMIENTOS
```

Vistas

Una vista (view) es uno o más comandos que devuelven un set de resultado y llevan consigo un nombre. Para los programas que ejecutan comandos, la vista es idéntica a una tabla. De esta manera el programador SQL puede juntar tablas y condiciones en una vista y cambiar los parámetros de la misma. Por ejemplo, la siguiente vista repite el ejemplo de visualización de facturas que vimos en la sección de SELECT y lo encapsula en una vista mucho más fácil de usar.

```
CREATE VIEW DESPLEGARFACTURA
AS
SELECT
        ENC.FACNO,
```

```
        ENC.NOMBRECLIENTE,
        ENC.CIUDAD+", "+ENC.ESTADO AS CIUDADEDO
        REN.ITEMNO,
        REN.DESCRIPTION,
        REN.PRECIO_USD,
FROM FACTURAS ENC
LEFT JOIN FACTURARENGLON REN ON ENC.FACNO = REN.FACNO
```

Note que en esta "vista" no tenemos el número de factura. Esto es intencional, para que podamos escribir comandos como:

```
SELECT * FROM DESPLEGARFACTURA WHERE FACNO = 1
```

En este comando, el set de resultado es el mismo que en la sección de SELECT/JOIN, que es lo que esperaríamos:

FACNO	NOMBRECLIENTE	CIUDADEDO	ITEMNO	DESCRIPCION	PRECIO_USD
1	DAVID MARTINEZ	MEXICO, DF	112232	COMPUTADORA COMPAQ PRESARIO	$2,340
1	DAVID MARTINEZ	MEXICO, DF	112522	MONITOR COMPAQ 21"	$1,340
1	DAVID MARTINEZ	MEXICO, DF	513213	3COM PALMIII	$400.00
1	DAVID MARTINEZ	MEXICO, DF	445653	LIBRO "MASTERING DELPHI 3"	$39.99
1	DAVID MARTINEZ	MEXICO, DF	445657	LIBRO "COLLAB COMP W/DELPHI 3"	$69.99